

Number 732



**UNIVERSITY OF
CAMBRIDGE**

Computer Laboratory

Computing surfaces – a platform for scalable interactive displays

Alban Rrustemi

November 2008

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
<http://www.cl.cam.ac.uk/>

© 2008 Alban Rrustemi

This technical report is based on a dissertation submitted November 2008 by the author for the degree of Doctor of Philosophy to the University of Cambridge, St Edmund's College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

<http://www.cl.cam.ac.uk/techreports/>

ISSN 1476-2986

Acknowledgements

Over the three years of my thesis work I have had the privilege and benefit of interacting with many great researchers at the University of Cambridge. I would like to thank Simon Moore, my supervisor, for his never-ending support and enthusiasm for the project. I would also like to thank the members of the Computer Architecture Group and Rainbow Research Group for all the inspiring conversations that have contributed to this work. In particular, I would like to thank Andrew West for his significant technical assistance. I would also like to thank Arnab Banerjee and Dan Greenfield for their invaluable feedback. Further thanks go to Robert Mullins and Jon May for their continuous support.

Special thanks go to Andy Harter and Ken Wood. This project would not have started without their contribution. I would also like to express my gratitude to the members of Computer Mediated Living Group in Microsoft Research, Cambridge for an inspiring internship project. This experience helped sharpen my interests in emerging sensitive display technologies. In particular, I am very thankful to Steve Hodges, Alex Butler and Shahram Izadi.

There are great many people who have contributed to these remarkable years in Cambridge, and therefore making this research project even more gratifying. First and foremost, I am more than grateful to Noel Malcolm for his moral and practical support. My appreciation goes also to St Edmund's College and all my friends that I've met there. In particular, I would like to thank Julie MacArthur, not only for being a great friend but also for her editorial input. Finally, I am indebted to Esther Laufer for her enduring support even at the most challenging times.

This work was funded by the Microsoft Research PhD Scholarship Programme and Cambridge Overseas Trust.

Abstract

Recent progress in electronic, display and sensing technologies makes possible a future with omnipresent, arbitrarily large interactive display surfaces. Nonetheless, current methods of designing display systems with multi-touch sensitivity do not scale. This thesis presents *computing surfaces* as a viable platform for resolving forthcoming scalability limitations.

Computing surfaces are composed of a homogeneous network of physically adjoined, small sensitive displays with local computation and communication capabilities. In this platform, inherent scalability is provided by a distributed architecture. The regular spatial distribution of resources presents new demands on the way surface input and output information is managed and processed.

Direct user input with touch based gestures needs to account for the distributed architecture of computing surfaces. A scalable middleware solution that conceals the tiled architecture is proposed for reasoning with touch-based gestures. The validity of this middleware is proven in a case study, where a fully distributed algorithm for online recognition of unistrokes – a particular class of touch-based gestures – is presented and evaluated.

Novel interaction techniques based around interactive display surfaces involve direct manipulation with displayed digital objects. In order to facilitate such interactions in computing surfaces, an efficient distributed algorithm to perform 2D image transformations is introduced and evaluated. The performance of these transformations is heavily influenced by the arbitration policies of the interconnection network. One approach for improving the performance of these transformations in conventional network architectures is proposed and evaluated.

More advanced applications in computing surfaces require the presence of some notion of time. An efficient algorithm for internal time synchronisation is presented and evaluated. A hardware solution is adopted to minimise the delay uncertainty of special timestamp messages. The proposed algorithm allows efficient, scalable time synchronisation among clusters of tiles.

A hardware reference platform is constructed to demonstrate the basic principles and features of computing surfaces. This platform and a complementary simulation environment is used for extensive evaluation and analysis.

CONTENTS

List of Figures	9
1 Introduction	11
1.1 Scalability impediments	13
1.2 Computing surfaces	14
1.3 Thesis statement	15
1.4 Roadmap	15
2 Technology background	17
2.1 Flat panel display technologies	18
2.1.1 Matrix addressing	18
2.1.2 Scalability limitations	19
2.2 Manufacturing considerations	21
2.3 Flexible opto-electronics	22
2.3.1 Organic electronics	22
2.3.2 Electroluminescent organic materials	24
2.3.3 Other materials	25
2.4 Sensor technologies for sensitive displays	26
2.5 Related work	28
2.5.1 Scalable display architectures	28
2.5.2 Seamless tiled displays	31
2.5.3 Interaction with surfaces	32
2.6 Summary	33
3 System architecture	35
3.1 Tiling	35
3.2 A scalable network architecture	37
3.3 Implications	39
3.4 Communication considerations	40
3.5 Software architectures for computing surfaces	41
3.6 Evaluation platform	42
3.6.1 A hardware prototype for computing surfaces	42
3.6.2 Simulation environment	45
3.7 Summary	46
4 Distributed input	47
4.1 Related work	48
4.1.1 Handwriting and gesture recognition	48
4.1.2 Distributed gesture recognition	49
4.2 Implications of tiling sensor arrays	49
4.2.1 Providing recognition services	49

4.2.2	Surface tessellation	50
4.3	A universal platform	52
4.4	Case study: Distributed unistroke recognition	56
4.4.1	Recognising from four orientations	60
4.5	Evaluation	61
4.5.1	Sensitivity analysis and the accuracy of the recognition algorithm	61
4.5.2	Performance of the distributed unistroke recognition	66
4.6	Summary	67
5	Distributing output	69
5.1	Related work	70
5.2	Distributing images in computing surfaces	71
5.3	Manipulating graphical objects	73
5.4	Distributed 2D transformations	76
5.4.1	Aims	76
5.4.2	Analysis	76
5.4.3	Coordinate systems	77
5.4.4	Partitioning	79
5.4.5	Algorithm summary	84
5.5	Performance evaluation	86
5.5.1	The hardware prototype and the corresponding simulations	86
5.5.2	Network evaluation	89
5.6	Summary	105
6	Time synchronisation	107
6.1	The system model of time	109
6.2	Related work	111
6.2.1	Synchronisation diversity	111
6.2.2	Classical time synchronisation in distributed systems	112
6.2.3	Time synchronisation in sensor networks	113
6.3	Time synchronisation for computing surfaces	114
6.4	Minimising message delay uncertainty	116
6.5	Offset synchronisation of immediate neighbours	120
6.6	Controlling the impact of clock drift	122
6.7	Chain synchronisation	124
6.8	Cluster synchronisation	128
6.9	Pre-facto synchronisation	129
6.10	Protocol synopsis	130
6.11	Evaluation	131
6.12	Summary	136
7	Conclusions and future work	137
7.1	Conclusions	137
7.2	Summary	138
7.3	Future work	139
	Bibliography	143

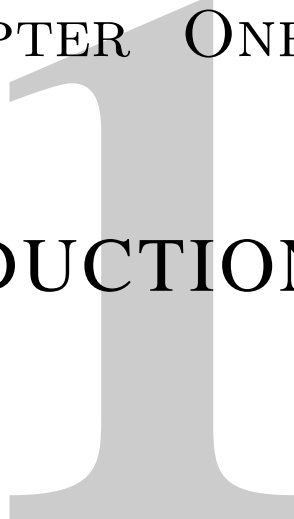
LIST OF FIGURES

2.1	A simplified display system	19
3.1	Using smaller tiles for building larger displays. The individual points can be pixels and/or touch sensors.	36
3.2	The structure of individual tiles in computing surfaces	37
3.3	The computing surface infrastructure	38
3.4	A display wall based on the computing surface architecture	43
4.1	The distributed input system	52
4.2	Disambiguating between the start of a new stroke and tile-to-tile stroke transition	53
4.3	The state transition diagram for the distributed input middleware	53
4.4	A gesture stroke distributed on several tiles. Borderline cases as the one presented in the magnifier need to be addressed appropriately	56
4.5	Unistrokes and their corresponding symbols (taken from [54])	57
4.6	Recognition of stroke that corresponds to letter ‘x’	60
4.7	Unistroke recognition from four orientations	61
4.8	The impact of parameter k in the stroke sloppiness sensitivity	62
4.9	Change of state location invariance for different resolutions	63
4.10	The impact of the smoothing constant (α) for various input resolutions in the recognition of the unistroke that corresponds to letter “L”	64
4.11	A comparison of the communication payload for the distributed and centralised recognition algorithms in the 3×3 FPGA hardware prototype	66
4.12	A comparison of the distributed algorithm communication payload for different network sizes (the size of the total input area: 300×300)	67
4.13	A comparison of communication payloads for two different strategies of transmitting stroke information to the processing tile (centralised recognition)	68
5.1	An overview of the image viewer used for computing surfaces	71
5.2	Translated coordinate systems	78
5.3	Tile partitioning	79
5.4	The positioning of control tile, an origin tile and its corresponding destination tiles in a 2D transformation	84
5.5	Task summary in different role tiles during distributed transformation	86
5.6	The temporal and structural representation of a single transformation for source-destination pair	87
5.7	The initial position of the image for the reported measurements (Table 5.1)	88
5.8	Base router model	89
5.9	FPGA measurements and simulation of execution time taken for completion of translation $T(5, 0)$ in images of different sizes, under same network density (tile size)	91

5.10	Steady state for a 15×1 network, undergoing $T(5,0)$ traffic (one VC) showing channel utilisation for horizontal channels and tile (sink) channels	92
5.11	A data-flow diagram of a 9×1 network with an image size $I(6,1)$ undergoing translation $T(3,0)$	94
5.12	Steady state channel utilisation for a 15×1 network, undergoing $T(5,0)$ for various numbers of VCs per input port	96
5.13	Simulation results for a 14×1 network with an image size $I(10,1)$ undergoing translation $T(4,0)$ under various tile injection rates	98
5.14	A queueing model of the switch during translation $T(4,0)$. The channel source has rate λ and the tile source has rate $\lambda/4$	99
5.15	A comparison of channel utilisation for a 14×1 network with an image size $I(10,1)$ undergoing translation $T(4,0)$	100
5.16	A map of resulting channel utilisations for XY, YX and O1TURN routing functions	100
5.17	Steady state channel utilisation for a 15×15 network, under $T(5,5)$ traffic	101
5.18	Vertical flip of the set of tiles presented in Figure 5.19	102
5.19	A triangle-shaped set of tiles undergoing a flip across vertical axis	102
5.20	A comparison between controlled and uncontrolled injection rates for a range of images $I(n,1)$ (n is given in x axis) undergoing vertical reflection around their centre (FPGA and simulation; partition and merge time: $8.63ms$)	103
5.21	A comparison between controlled and uncontrolled injection rates for a range of images $I(n,n)$ (n is given in x axis) undergoing 180° rotation around the centre-point of the image (FPGA and simulation; partition and merge time: $8.97ms$)	103
5.22	Network channel utilisation in steady state traffic for an image $I(20,20)$ undergoing 180° rotation around its centre	104
5.23	Tile sink rates for the same transformation as given in Figure 5.22	104
6.1	Contributors to message delay uncertainty in software implementations of timestamp transmission	117
6.2	Reading and writing of TSPs in the tile network interface	118
6.3	Time controller structure	119
6.4	Transforming neighbour time to local time	120
6.5	Offset synchronisation of distant nodes	124
6.6	Cluster synchronisation	128
6.7	Histogram of message delays depicting the range of uncertainties	131
6.8	Offset synchronisation without drift correction	132
6.9	Offset synchronisation with drift correction	132
6.10	Average transmission per tile during chain synchronisation	134
6.11	Average time taken to achieve chain synchronisation	134
6.12	Average number of transmitted packets by each node to achieve cluster synchronisation	135
6.13	Average timestamp delivery time in cluster synchronisation and the corresponding ratio between delivery times	135

CHAPTER ONE

INTRODUCTION



In this digital age, humans are surrounded with computation more than ever before. By supporting any of our everyday activities such as entertainment, communication, work, education, transport, etc, present computing devices have reached out to every aspect of human life. Apparent benefits brought to individuals and society motivate even further digital emphasis, thus gradually leading towards a central role of computation in our future and consequently resulting in profound changes in the way we live.

To undertake this role, computing devices come in an abundant diversity. The more explicit forms range from small digital assistants, to the omnipresent laptop and desktop computers, and extend all the way to astounding supercomputers. Less obvious, implicit computing systems, are even more widespread by being embedded into mobile phones, access cards, children toys, healthcare devices, consumer electronic devices, cars, etc. This thriving boom of implicit computing goes on as we witness a gradual replacement of conventional devices with more flexible and automated computerised systems.

Many specialised, computerised devices are often recreations of applications initially available only for personal computers. Despite being the best manifestation for the immense versatility of applications of computing, personal computers most often are not best suited for applications in use. During their evolution, many applications depart from the personal computer and get hosted by a dedicated computing device, aimed to bring usability advantages. This migration moves the focus away from the computing host and brings it to the actual application. In his seminal thesis on ubiquitous computing, Mark Weiser calls for a disappearing computer by commending that “the most profound technologies are those that disappear” [177]. In the given context, disappearing is the integration of computing devices into everyday life until they become indistinguishable.

Although the evolution of personal computing remains largely conjectural, it is apparent that further advances in user interaction technologies and information display technologies are necessary for a disappearing computer. Display technologies have recently demonstrated impressive technological advances. At the turn of the millennium, while the cathode ray tube (CRT) was the standard for almost all entertainment and professional displays, most of the mobile phone displays were monochrome and had very limited resolution. In contrast, within less than a decade, CRT based displays were re-

placed with more efficient liquid crystal displays (LCD) and current mobile phone devices have embedded high resolution displays.

Even with this impressive record of advances, most appliances that involve information display remain constrained by the limited area of the typical display device. The witnessed improvements have not yet managed to accommodate the need for larger and more compliant displays. Nonetheless, the outlook on display technology advances remains bright and offers much more. Some of the most radical and promising improvements are expected from flexible display research. Flexible displays can open up possibilities for many compelling applications that are not satisfied by rigid displays. More importantly, they promise lower costs due to a more efficient reel-to-reel manufacturing approach. A combination of alleviated cost factor and physical conformability can make possible large displays that would satisfy the requirements of any application.

Many research labs envision gigapixel (GPix) and even terapixel (TPix) displays. Embedding such displays into working desks could take the common desktop metaphor user interface to another level. Wall sized displays could be used in conference room settings and digital signage. Display-covered floors could provide directions and other information inside buildings. It seems that information display technology may be sharing computation's path to profoundness by integrating into the fabric of everyday life.

An equally important part to a computing infrastructure that would employ such displays is interaction. Researchers are exploring various interaction possibilities with large display areas. The suggested novel interaction techniques drift away from the use of keyboard and mouse and move towards a direct interaction with the digital content in display. Quite often, a range of tangible digital and physical artefacts are employed to make the interaction more intuitive. This form of interaction relies on displays with embedded touch-sensitivity.

Embedding touch sensitivity into flat panel display technologies is gradually becoming more robust. Recent years have witnessed the introduction of a wide range of multi-touch sensing technologies. Further on, some emerging embedded sensor technologies are capable of detecting much more than touch (e.g. bar-codes and document printouts). In order not to differentiate among various sensing features, we will refer to such technologies as simply 'sensitive displays'. When integrated right from the manufacturing process, sensitive displays have the potential to become more economic and widespread. In return, this would enable the widespread adoption of novel interaction techniques on large displays.

Commensurate advances in information display technologies, sensor technologies and electronic process technologies enable a compelling and far-reaching concept: *scalable, thin, interactive display surfaces*. Conforming to Mark Weiser's vision, computing would be embedded into the surface for enabling interaction, but users would not need to be aware of it. The surfaces that are envisaged to be covered with such interactive displays come in an unlimited range of shape and size. While current displays support different sizes, the underlying technology remains very limited in providing the flexibility that is required by the surrounding 'everyday life surfaces'. An inherently scalable technology that allows such interactive display surfaces to come in any size and shape is necessary. The thin form factor is necessary for permeation into the surfaces encountered in everyday life; other form factors produce undesirable constraints and would obstruct their wide-

spread integration into desktops, walls, etc. This concept of surface would open a path towards ubiquity for interactive displays.

1.1 Scalability impediments

A quick observation of potential applications that would benefit from embedded interactive displays indicates that the envisaged surfaces may come in almost any size and shape. Assuming constant display pixel density, such surfaces would result having pixel resolutions of an unlimited range. Although the fundamental technologies for producing thin, sensitive display surfaces may exist, the resolution scalability of such surfaces can be effortlessly brought into question.

Since the emergence of electronic information displays, optimal solutions have traditionally evolved into standards that defined display formats and specified fixed pixel resolutions. A limited range of such standards (e.g. NTSC, PAL, VGA, QXVGA, etc.) define the specifications of most conventional display systems. The information content is rendered and formatted for optimal display only for the given set of standard resolutions. Communication links (e.g. VGA, DVI) that connect the computing device to the display device are also designed to transfer information for the same set of resolutions. On the other hand, one of the first design specifications of building a new display device is its pixel resolution – also, belonging to the same set of standards. With the gradual progress of computing performance, more information can be rendered and formatted. Alongside, display devices that support higher pixel resolutions emerge continuously. New display standards follow up in the same stream. This abundance of pixel resolutions defeats the primary purpose of standards. Accordingly, Mayer in [100] proposed to declare display standards as “dead”.

Nevertheless, cunning engineering is necessary to design high resolution displays due to a wide range of electrical and physical constraints. One manifestation of such challenges is the time it takes to launch a display system with a resolution that is higher than its predecessor (currently ranging in years). On the other hand, while computational resources for driving high resolution displays are widely available, it is less clear how to *drive* displays with much higher (e.g. 100×) resolutions. Efficient scaling of computational resources to support higher resolutions is an active research topic.

It is clear that standardised pixel resolutions are not conformant to the concept of interactive display surfaces – assuming constant pixel density, arbitrary pixel resolutions are essential in order to meet spatial requirements and constraints of desired applications. Current technologies do not offer a framework for presenting information on display systems that span to arbitrary resolutions. Chapter 2 will elaborate further on scalability limitations of information display systems, arguing that the only feasible approach to build larger displays is by joining together smaller ones.

As a result, scalable, interactive display surfaces require novel technologies in order to achieve the desired versatility. The following section introduces a new approach for developing such surfaces.

1.2 Computing surfaces

The solution is based on the observation that there is a significant amount of knowledge and expertise available to build small, efficient, sensitive displays. With prudent engineering, larger units could be built by adjoining smaller ones to create the illusion of one large homogeneous sensitive display. We will refer to the smaller building blocks as *tiles*. Each tile, in addition to its sensitive display contains computation and communication resources. Local computation resources are used to support the necessary processing of local tile input and output. Providing local computation to each tile also resolves the communication with the actual display device – the frame buffer of the computing surface is distributed among individual tiles, thus avoiding a potential bottleneck. Although tiles can be treated as self-contained display systems, they are envisaged to be an integral part of a larger display. In fact, the primary purpose of the local computation resource is to support the tile’s functionality as part of a larger entity. This leads to the concept of *computing surfaces* – a sensitive display surface assembled by a homogeneous collection of computing tiles.

The activity of individual tiles is tightly coupled with the activity of surrounding tiles. The input and output of individual tiles has broader, surface-level context – many tiles may take part in presenting information content by each of them displaying fragments of it. On the other hand, when interacting with the surface, local tile input is most likely to affect many tiles. For illustration purposes, consider a scenario where a picture is displayed on the surface. This picture is moved to another location of the surface by sliding the finger on the part of the surface where the picture is located. The content of the picture is most likely spread out among several tiles. The sensor input coming from touching the surface may be received by one or more tiles. Yet, each update on this input instigates a picture movement that affects all the tiles that contain a picture fragment. Input tiles can trigger the necessary computations in support of the process of moving the picture. The computational resources available in every tile can be used to execute the relevant process. Local computation resources available in one tile are limited and could result in poor performance. Considering that identical resources are available in every tile, task sharing possibilities should be investigated for improved performance.

In order to achieve such operations, tiles need to be able to exchange information with one another. This is facilitated with communication resources available on every tile. A scalable communication infrastructure is essential for a scalable surface. Considering the inherently large communication demands of display applications and the potentially vast areas where computing surfaces could spread, the aggregate amount of information transfer can easily take staggering magnitudes. Data communication is often the bottleneck of large scale architectures and results in significant proportion of power consumption. As a result, it is essential to design algorithms that require minimal communication.

It is evident that computing surfaces are a new type of distributed systems. Classical distributed systems do not enforce any accurate constraints on spatial relations between their nodes. This spatial relationship between tiles and the corresponding input and output activities accounts for the fundamental differences between computing surfaces and other distributed systems.

This approach allows the resulting computing surfaces to have rectilinear shapes. This property is very desirable for the envisaged interactive surfaces, since many physical ob-

jects in our surroundings are not rectangular. The architecture of conventional display technologies does not leave much room for non-rectangular displays.

1.3 Thesis statement

Using computing surfaces as the driving approach towards thin, interactive display surfaces, we state the thesis of this dissertation: *Thin, scalable, interactive display surfaces can be viably constructed by a homogeneous network of tiles joined together to form an arbitrarily large entity, capable of independently supporting a range of useful interaction applications by employing its own potentially vast and uniformly distributed resources of computation and communication.*

In support of this thesis statement, this work presents several contributions:

- *Networking* – a 2D mesh network specification for interconnecting tiles to create arbitrarily large sensitive displays.
- *Distributed input* – an efficient, middleware platform for the recognition of gestures with fragments distributed among several tiles. A distributed gesture recognition algorithm for a particular class of stroke gestures is presented to demonstrate the functionality of the platform.
- *Distributed output* – a distributed algorithm for applying two-dimensional transformations on images that are distributed among several tiles is presented. In support of such transformations, mechanisms for significant improvements on the network performance are presented.
- *Time synchronisation* – a novel, time synchronisation protocol, as an essential service for more elaborate applications is presented. Taking advantage of the exclusive, point-to-point channels in 2D mesh and a customised network interface, sub-microsecond precision is achieved at very low communication cost.


1.4 Roadmap

The remainder of this dissertation lays out the rationale and explores the concepts in support of this thesis. Chapter 2 reviews the background and related work on the key technologies in support of the concept of computing surfaces – display technologies, embedded touch sensing and interaction technologies. Chapter 3 lays out the rationale behind the suggested tile-based architecture, presents the hardware reference platform, and the simulation environment for evaluating purposes. Chapter 4 discusses the challenges brought by a tiled architecture on reasoning about gesture input. It presents a universal middleware platform that mitigates the gesture distribution among several tiles. The feasibility of such a platform is evaluated with a case study. Chapter 5 presents a fully distributed algorithm for applying two-dimensional transformations on images distributed among several tiles of the surface. In addition, it identifies performance issues with the allocation of network resources and suggests modifications that yield significant performance improvements. Chapter 6 addresses the need for a time synchronisation service

with a precise synchronisation algorithm. Finally, Chapter 7 states the conclusions and speculates on avenues for future work.

CHAPTER TWO

TECHNOLOGY BACKGROUND



The implementation of computing surfaces as thin, conformable, sensitive display surfaces requires the confluence of many technologies. Among its hardware requirements, the most essential would be: suitable display technologies that would extend throughout the surface, sensor technologies for providing sensitivity to the surface and suitable electronics to support internal communication and computation. These technologies are a prerequisite to the material completion of computing surfaces and at present time, their state-of-the-art is insufficient for an immediate implementation. Nevertheless, many emerging technologies hold the promise to take this compelling concept from the present, visionary state to an actual implementation.

This chapter will present a review of candidate technologies and their feasibility. It begins with a quick introduction to flat panel display technologies (FPDs) in order to discuss the main resolution limitations in display systems. Section 2.2 elaborates on current and future manufacturing processes and reveals the great potentials and implications of reel-to-reel manufacturing process. A reel-to-reel process that is adopted for display systems requires conformable materials for displays, sensors and the supporting electronics. Section 2.3 presents an overview on the emerging flexible opto-electronic technologies. As our concept of computing surfaces envisions surfaces to be touch-sensitive, the technology candidates for this feature are presented in Section 2.4.

The vision of large format displays has attracted the attention of many research communities, starting from display manufacturers and extending to researchers in HCI. Section 2.5 reports on their work, by focusing on three particular aspects:

- scalable display architectures built with commodity computers, networks and an array of tiled displays, emulating a larger seamless display (Section 2.5.1),
- use of the concept of tiling in display design and manufacturing (Section 2.5.2) for building larger displays
- novel interaction techniques that envisage large, seamless, sensitive display areas (Section 2.5.3)

But first, we begin with a quick introduction on FPDs.

2.1 Flat panel display technologies

At the present time, there are two main approaches towards covering relatively large areas with information display: FPDs and optical projection. Displays of theatrical scope are traditionally achieved with optical projection, where the display content is projected from the display source (projector) onto a passive, usually flat surface. This decoupling of the display source from its surface enables efficient distribution of the display content onto large destination areas, but also instigates spatial constraints, subsequent occlusion problems and setup/calibration overheads. In addition, they often require specific environmental conditions (e.g. dimmed rooms/halls) to maintain satisfactory display quality. In the FPD approach, the display source is embedded into the encompassing package of the display surface, and therefore overcoming the spatial disadvantages of projection displays. The physical proximity of the light source, relative to the surface results in higher display quality. The combination of FPDs' reduced form factor, good display quality and competitive cost makes them the most popular information displays for non-theatrical applications.

There is a wide range of display technologies captured by FPDs that differ on the way the pixel light and/or colour is generated. The most well-known examples include organic light emitting diodes (OLED), liquid crystals (LCD), plasma displays (PDP), surface-conduction electron-emitter (SED), field emission (FED), nano-emission (NED), electrophoretic, etc [110]. Although the underlying technologies for generating pixel light of appropriate colour are very different from each other, all FPDs have a similar approach to pixel addressing.

2.1.1 Matrix addressing

Some means of addressing is required to display the intended colour and intensity at the correct pixel, at the right time. Direct addressing of each pixel with a separate pixel driver and a corresponding electrical connection would involve intolerable wiring and circuitry. Efficient addressing of the full array of pixels requires reducing this number of wires and drivers. Reduction in interconnect also brings improvements in area efficiency, cost, signal integrity, driver electronics, etc. To achieve this reduction, some means of multiplexing is necessary. Multiplexing trades off duty cycle – the fraction of the time drivers and corresponding wires dedicate to each pixel. The furthest extreme from direct addressing is sequential addressing, where only one pixel is selected at a time. This method is used in Cathode Ray Tube (CRT) displays [110], where three electron beams (one for each primary colour) traverse through all display pixels to address them individually. For a screen with pixel resolution of $W \times H$, the duty cycle of direct addressing would be equal to one, while the duty cycle for sequential addressing would be $1/(W \times H)$. On the other hand, the number of drivers and data connections to pixels for direct addressing is $W \times H$, while sequential addressing requires only one driver. Clearly, both extremes of this trade-off are inadequate. Matrix addressing [110] represents the best known tradeoff between control resources and duty cycle, where one row of pixels is selected at a time. While a row is selected, the pixel data is passed through column drivers. In the simplest form, addressing the entire screen is accomplished by sequentially selecting each row of pixels (i.e. line-at-a-time [110]). The duty cycle is $1/H$ and the number of drivers is

$(W + H)$. For matrix addressing to work, display pixels must have non-linear response, since otherwise row selections would interfere with each other, thus degrading display quality. The amount of interference is closely related with the non-linearity of the pixel response. Some addressing schemes such as [4] resolve this interference but work only with a limited numbers of rows and limited operating voltages. Sharp thresholds are essential for good display quality and can be achieved by using transistors to control individual pixels. FPDs that employ this approach are referred to as active matrix (AM) [132] displays. Thin film transistors (TFT) are used to provide one transistor for each colour in the pixel of the AM display. At present state, AM addressing seems to be necessary for achieving satisfactory qualities for FPDs. Even with efficient TFTs, AM FPDs encounter some fundamental limitations on the number of multiplexed rows they can support.

2.1.2 Scalability limitations

Figure 2.1 presents a simplified view of a system composed of an FPD, connected to a computer system. The computer system serves as an information source for the presented content.

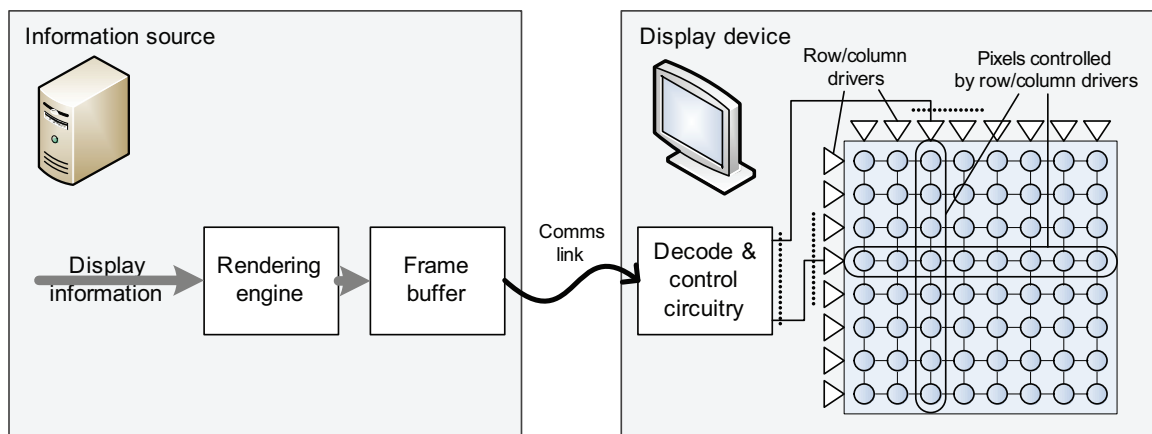


Figure 2.1: A simplified display system

Display information, represented in some higher level (e.g. text or images) is rendered by a graphical rendering engine and is stored in the frame buffer. The frame buffer content is continuously transmitted over the communication link to the display device, where it is decoded and transformed into stimulus for row and column drivers. Although simplistic, Figure 2.1 reveals some central points where the current approach to information display exhibits limited resolution scalability. These limitations can be categorised as follows:

- *Computation/rendering* – higher resolutions allow more information to be displayed, thus instigating bigger rendering tasks. This increase in rendering tasks requires either additional computational resources or additional time. This problem is widely recognised by the research community, and as a result, several scalable graphics engines have been proposed (Section 2.5.1). While the limitations introduced by computational resources can be considered as a ‘soft’ bound, the memory size of the frame buffer sets a clear limit to the supporting resolution.

- *Communication bottleneck* – the bandwidth limitation of the communication link between the computing system and the display device establishes another upper bound on the resolution of the displayed content.
- *Display architecture* – a lower bound on the pixel addressing duty cycle is set by the minimum response time of the pixel. More practically, the lower bound on the duty cycle is set by the response time of all the pixels in the row. On the other hand, the upper bound of the duty cycle is limited by the number of rows (Section 2.1.1). The exact upper bound is dependent on the specific details of the addressing algorithm – e.g. for those based on ‘line-at-a-time’ approach, it can be given with the following relation:

$$N \leq \frac{T_f}{f(G)T_a}$$

where N is the number of rows, T_f is the frame time period, T_a is the minimum response time of a row, G is the number of gray levels and $f(G)$ is the number of addressing cycles required to achieve G gray levels. Many addressing schemes attempt to increase the number of scan lines by adjusting the way gray levels are formed (i.e. amend the function $f(G)$). In a basic scheme, $f(G) = G$, whilst more economical schemes trade maximum brightness for more scan lines ($f(G) = \log_2 G$) [110]. It can be seen that a continuous increase in the number of scan lines can be guaranteed only if T_a can be arbitrarily small, which is impossible due to physical properties of pixels and interconnect.

Addressing schemes that allow selecting multiple rows at a time are referred as active addressing and are implemented with the cost of more extensive decode and control electronics. The number of selected rows at any given time depends on the algorithm used and on the frame content. Certain pixel configurations prohibit the simultaneous selection of multiple rows – a typical example is a frame consisting of a coloured/white diagonal line on a black background. Active addressing schemes are only opportunistic when it comes to simultaneous selection of multiple rows and therefore do not impact the fundamental bounds that are set by the inverse relation between duty cycle and number of rows. On the other hand, active addressing schemes often improve a range of other properties, resulting in more efficient displays. For example, Total Matrix Addressing for OLED passive panels, introduced in [150] improves pixel efficiency and reduces losses from capacitive charging.

- *Physical* – caused partly by long, thin wires connecting drivers and pixels. The resistance of the wire will increase with its length which, in addition will also affect its capacitance. Increased resistance over a long wire results in voltage (IR) drops, whilst increased capacitance will subsequently result in slower charge times (RC), and thus increasing the value of T_a . Since the drivers in flat panel displays are placed outside the area covered by the pixel array, this dependency between distance and signal strength determines an upper limit to the display size.

In addition, the power applied to column drivers operating with voltage V is proportional to fNV^2 and sets a practical limit to N or f [110]. Assuming displays that support video frame rates, the power specifications of the column drivers set a physical limit to the number of rows.

This list is not at all exhaustive and only attempts to depict more fundamental factors. Another major limitation not captured in Figure 2.1 is display manufacturing. Imperfect manufacturing processes heavily influence the economies of large format displays. An important issue is the decreasing yield of thin film transistors for larger displays [152]. Manufacturing processes set physical and economical limits on the size of the display. Such limitations are often encountered before the saturation of computation, communication and addressing limitations.

2.2 Manufacturing considerations

FPDs are very complex devices and with their detailed structure greatly depending on the actual display technology. Due to their form factor, thin films of various materials and functions are an essential component for all FPDs. Thin films are very sensitive to manufacturing variabilities, and thus their robust manufacturing is one of the most important components of FPD manufacturing. Depending on their role and the composing material, thin films are produced in various manufacturing processes. One of the most popular processes in the traditional development of many complex thin films is vacuum thermal evaporation (VTE).

VTE is the most common method for depositing small-molecular-weight thin films [45]. One particular advantage of VTE is its ability to grow an unlimited number of layers, necessary to complete the device structure. The lack of solvents in the evaporation process brings tremendous flexibility in device design since the materials from several deposited layers do not physically interact with each other [46]. Consequently, this compatibility between layers allows flexible choice of materials and structures to be used in complex thin film devices.

The use of vacuum chambers in a manufacturing process involving VTE necessitates batch processing, which handles only one component at a time. Although batch processing is a norm across manufacturing in the display industry as well as the semiconductor industry, it is not as efficient as an ideal reel-to-reel manufacturing process. The concept of a reel-to-reel manufacturing process is similar to that used in the printing industry. By using a continuous processing flow, it has the potential to yield radical efficiency improvements against batch processing. A mature reel-to-reel manufacturing process for displays could reduce capital equipment costs, significantly increase throughput, and thus reduce the overall manufacturing costs per product unit [29].

Reel-to-reel processes inherently require operating with conformable materials. Plausible conformability, combined with suitable optical and electrical (conducting and semi-conducting) properties makes organic materials a serious candidate for a comprehensive reel-to-reel FPD manufacturing process. In addition, organic materials have generally low cost. The promising efficiency of reel-to-reel manufacturing and the inherent low cost of the composing materials suggests that flexible, organic FPDs may have substantially lower costs, compared to conventional FPDs. This cost incentive constitutes a pragmatic motivation for many industrial research labs to develop flexible displays as replacements for conventional FPDs [3]. From another end, demanding military applications provide a significant source of novel applications for robust, flexible displays [108]. This combination of novel applications and improved manufacturing efficiencies for current FPD applications offers immense motivations for the display research community and industry

to provide robust reel-to-reel manufactured flexible displays. Accordingly, the FPD community often uses words “holy grail” to reflect on their desire to achieve commercialised flexible display technology [29, 79].

Local patterning of thin films is essential for any substrate that contains electronic circuits. For this purpose, appropriate shadow masks are used in evaporation-based processes. The most promising patterning technology that is compatible with reel-to-reel manufacturing process is ink-jet printing [147]. This approach has been around for many years in other applications such as office colour printers. In thin film patterning and depositing, ink-jet printing requires precise control of the material chemistry to satisfy all the electrical and optical demands of the application (e.g. display). In addition, since ink-jet printing operates with solution processed materials, special care is required with the choice of solvents as they may undesirably interact with previous layers. Despite these potential deficiencies, early results of ink-jet printed organic electronic devices have been very promising [140]. The feasibility of this approach has been demonstrated in many prototypes. Examples include the fabrication of a 17” polymer-based, full-colour display in [107], or the fabrication of the world’s first 40” full-colour OLED display [124]. This demonstration of the patterning technology over such a large spatial scale indicates the potentials of ink-jet printing for manufacturing very-large scale and complex organic electronic circuits.

In addition, it should be emphasised that reel-to-reel can also be applied to vacuum-deposited organic semiconductors [37]. A technique that is similar to ink-jet printing, known as organic vapour-jet printing (OVJP), has been proposed and developed for use with small-molecular-weight materials [142, 143]. However, OVJP is in early stages of development, and there are still no clear demonstrations that this technology will eventually find use in the large-scale production of low-cost organic electronics [46].

From the computing surfaces perspective, reel-to-reel reveals another advantage in comparison to manufacturing techniques that use vacuum chambers. The size of the display substrate is limited by the size of the vacuum chambers. On the other hand, in a reel-to-reel process, one dimension in the size of the display is unconstrained by the manufacturing equipment.

2.3 Flexible opto-electronics

2.3.1 Organic electronics

For the past five decades, inorganic Si and GaAs, silicon dioxide insulators, and metals such as aluminium and copper have been at the core of the electronics industry. At the very centre of inorganic electronics and dominantly the CMOS technology are its fascinating trends of integration [73], which currently accommodate billions of transistors in mm^2 areas. In fact, these are the very trends that have spurred a range of technological advancements that have together contributed to the ongoing digital revolution. The underlying process technologies that enable the production of these electronic circuits have been continuously pushing scientific frontiers in order to cram more and more transistors into mm^2 area budgets. These advances have both improved the transistor performance and drastically reduced the cost of individual transistors. However, the cost of these

transistors remains low, only under this highly integrated environment. An attempt to use this technology for FPD TFTs would be prohibitively expensive.

The past three decades have witnessed a growing research effort in electronics that are based on a wider range of materials, most of them being organic. Organic electronics are continuously going through significant advancements, and if the field maintains the progress at its current pace, it can soon reach the ubiquity status of inorganic semiconductors [46]. One of the main technological attractions of organic electronics is based on its potential that all the necessary layers can be deposited and patterned at low/room temperature by a combination of low-cost solution processing and direct-write printing on potentially large areas. Up to now, there has been a range of considerable obstacles towards this goal, mainly consisting of suboptimal device performance, manufacturing and robustness.

The main performance impediment of organic materials has been charge mobility. Measurements of some of the early organic semiconductors (e.g. polythiophenes) provided charge mobilities ranging around $10^{-5} \text{ cm}^2\text{V}^{-1}\text{s}^{-1}$ [138], which is insufficient for any practical application. Considerable progress has been made since then to improve the semiconducting and conducting properties of polymers and organic-inorganic composites through novel synthesis techniques. Materials such as Pentacene [92] exhibit comparable mobilities with amorphous silicon (a-Si) [37], but their performance is still rather far from inorganic semiconductors. Measurements on single organic crystals which currently exhibit the best performance in organic electronics show mobilities that are still orders of magnitude lower than single-crystal silicon [138]. Furthermore, it may be difficult to further improve the current set of organic semiconductors due to weak van der Waals interactions between molecules [37]. Under such limiting circumstances, higher performance is sought with more success on hybrids of organic and inorganic materials, discussed in Section 2.3.3.

Although charge mobility is critical for achieving good performance, materials' stability under air, moisture and light exposure is just as important for device robustness and reliability. Dopants such as oxygen or water can enter the organic film through unintentional exposure to the environment, leading to degradation of the device performance over extended periods of operation. Traditionally, some form of device encapsulation has been necessary, which is usually achieved through the use of substrates such as glass or metal. Encapsulation becomes much more acute in applications requiring flexible substrates since their permeability is much higher [3, 38]. Significant progress has been made on the development of organic semiconducting materials that exhibit higher stability under the aforementioned dopants. According to work presented in [127, 146], novel, unencapsulated polymer-based FETs have been continuously exposed for weeks in light and oxygen without showing any device degradation.

Practical implementations of organic electronics will ultimately be decided by the ability to produce devices that have reel-to-reel compatibility. In this respect, direct ink-jet printing of complete transistor circuits, including via-hole interconnections based on solution-processed polymer conductors, insulators, and self-organising semiconductors has been demonstrated in [147].

Conclusively, reel-to-reel compatible, polymer-based FETs that exhibit no susceptibility to dopant exposure, and approach the charge mobilities of a-Si have been demonstrated [146]. Many practical applications require higher device performance than current state-of-the-art polymer-based FETs, and therefore more work remains to be done before any

widespread adoption. However, the outlook remains good as organic systems offer a great deal of flexibility in their synthesis. As chemists develop new materials and learn how to better order and process them, it is hoped that mobility will continue to improve, reaching the performance of poly-Si and expanding the applications of such materials for low-cost logic chips. A witness to such progress is presented by the commercial ventures such as Plastic Logic, PolymerVision, etc.

2.3.2 Electroluminescent organic materials

In the past decade, unremitting improvements in manufacturing processes, power consumption and display quality of the LCD technology have resulted in a far-reaching replacement of CRTs. At present, LCD dominates for most information display applications. Nonetheless, a closer look at this technology reveals many inherent problems that are mainly related to its complex and expensive manufacturing process, power inefficiency resulting from the back-light approach, non-conformability, etc [89]. FPDs that are based on organic light emitting diodes (OLED) promise to overcome many of the problems that are present in the LCD technology. OLED displays are usually much simpler to manufacture than the LCD counterparts [42]. These simplifications result from the emissive approach of OLED displays, whereby each pixel generates its own light of appropriate colour. In contrast to LCD displays, OLED displays do not require back-light, polarising filters, colour filters and other films, which not only simplifies the manufacturing process, but also results in substantial improvements in power consumption efficiency. At the same time, the emissive nature of OLED displays also offers much higher contrast and wider viewing angles.

Electroluminescence in organic materials was first discovered in 1950s by applying a high-voltage alternating current (AC) field to crystalline thin films of acridine orange and quinacrine [10, 11]. Advances followed by the demonstration of electroluminescence resulting from direct currents using single crystals of anthracene [120], thus giving rise to the first light emitting diodes based on organic materials. OLED devices come in two large categories – Small Molecule¹ OLED (SMOLED) [89] and Polymer² OLED (POLED) [16].

The first efficient organic light-emitting devices based on small molecules in multilayer configuration were presented by Kodak Chemical in [162]. On the other hand, the first demonstration of POLED devices was done at the Cavendish Laboratory, University of Cambridge using unsubstituted poly (p-phenylenevinylene) or PPV [47].

From a performance perspective, the operating lifetime and the manufacturing processes are among the main qualitative discriminators between SMOLED and POLED. SMOLED have satisfactory operating lifetimes for several information display applications [89] and as a result appear in many products. Traditionally, SMOLED are not solution processible and thus their manufacturing has been done with VTE deposition. This potential reel-to-reel incompatibility has been considered as the main disadvantage of SMOLED. On the other hand, POLED devices are usually solution processed

¹Small molecule is a term broadly used to refer to those compounds with a well-defined molecular weight [46].

²Polymers are long-chain molecules consisting of an indeterminate number of molecular repeat units [46].

and traditionally ink-jet printed [147], which gives them a significant advantage against SMOLED. Nonetheless, POLED devices have relatively short operating lifetimes (especially for blue OLEDs) [89], and this has been one of the main obstacles to widespread adoption. The vast commercial potentials of OLED displays has driven a lot of research effort towards overcoming these disadvantages, and as a result new advancements are continuously presented. For example, DuPont has recently developed solution processing of SMOLED materials, thereby potentially combining the advantages of solution processing with the good performance of SMOLED devices [42]. On the other hand, POLED lifetimes keep increasing alongside gradual, quality improvements of blue POLEDs [125].

At present, medium to large FPD products based on OLED devices are gradually entering the commercialisation phase – e.g. Sony Corporation offers 11” OLED TV products [126]. Epson’s 40” full-colour active matrix OLED (AMOLED) display is also reported to be ready for commercialisation [124].

OLED-based FPDs have another intrinsic advantage – the physical properties of OLED materials make it a suitable technology for flexible displays. High-mobility conjugated polymer FETs driving POLEDs of similar size are presented in [148], therefore demonstrating all-polymer display/semiconductor integrated devices. The performance of the presented polymer FETs approaches that of inorganic a-Si FETs. Given the fact that OLED devices are current-driven, TFTs based on organic transistors and a-Si are not yet suitable due to their low current drive.

Although polymer-based FETs may have overcome their dopant exposure problems [146], OLED devices still remain very susceptible to oxygen and moisture [3]. To address this issue, several barrier structures based on various materials have been proposed. A multibarrier structure is used in [155] to develop a flexible OLED with 160×120 RGB pixels. In another investigation presented in [26], a combination of flexible metal foil with poly-Si TFTs has been used for substrate suggesting that LTPS TFT backplanes are suitable for stable, flexible AMOLED displays.

The lack of suitable, organic-based TFTs and flexible substrates with appropriate encapsulation [3], are the main pressing issues currently facing flexible OLED FPDs. High charge mobility of poly-Si as well as its stable structural TFT characteristics currently make poly-Si TFTs the technology of choice for AMOLED [112]. The search for suitable TFTs and flexible substrates is still very much under way. While the presented suggestions provide only incremental improvements, they demonstrate the resolve of the research community to develop stable flexible AMOLED displays.

2.3.3 Other materials

The search for suitable materials in support of flexible electronics is not limited to organic compounds. Inorganic and organic materials are often mixed together in order to combine their advantages – inorganic materials offer improved charge transport because of stronger ionic and covalent bonds compared to the weaker intermolecular interactions in organic semiconductors, while the organic portion of the system provides the required processibility and mechanical properties. It is not surprising, that initial reports of such hybrid materials reach mobilities of state-of-the-art pure organic materials. For example, a class of organic-inorganic perovskites [105], has already achieved the mobility of amorphous silicon [77]. Further on, these perovskites have demonstrated a Hall mobility of 50

$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$, providing a possible path to increased performance [106]. Consequently, suitable hybrids with higher performances are actively being pursued [138].

Recent progress has been reported in efforts towards developing flexible and transparent electronics and displays using nano-wire transistors (NWT) [76]. The optical transmission of these devices is reported to be high (82%) and device performance characteristics are comparable to standard poly-Si. The combination of good optical transparency, mechanical flexibility and satisfactory device performance metrics make NWTs an attractive technology for realising transparent and flexible circuits and AMOLED displays. Transparent TFTs would be beneficial for AMOLED displays due to inherently larger aperture ratios. However, their adoption in flexible AMOLED displays is currently infeasible due to high manufacturing costs, involving photolithography, etching, plasma enhanced chemical vapour deposition, etc. [76].

Various forms of integration are also pursued for sensing applications. Integrated structures of organic transistors and rubber pressure sensors have been demonstrated with the aim to propose implementations of artificial skin surfaces [151]. In this study, pressure information is collected by flexible active matrix drivers using organic transistors. The device is reported to be electrically functional even when it is wrapped around a cylindrical bar with a 2-mm radius. In addition to organic transistors, thin films of rubber pressure sensors can also be produced by low-cost printing technology, making such a technology reel-to-reel compatible.

Although the current drive of TFTs based on solution processable organic transistors may currently be insufficient for OLED, this technology is suitable for capacitive media such as electrophoretic displays [7, 50]. Electrophoretic motion of particles inside small capsules [28] is used in electronic paper concepts (e-paper) [129]. Several demonstrations and product announcements have been made [119]. A flexible 4.7 inch QVGA active matrix display with solution processed organic transistors is presented in [68] using E-Ink[®] electrophoretic ink [129]. A similar implementation that includes row shift registers in the same manufacturing process is demonstrated in [50]. The reported operating frequency of 5 kHz is sufficiently high to support video-speed frame rates. These displays can be bent to a radius of 1cm without significant loss in performance. Electrophoretic displays are currently the closest technology to full compatibility with a reel-to-reel manufacturing process. Display sheets may already be produced in a reel-to-reel process using metal foil or polymeric substrates with organic transistors for TFTs and other electronics [6, 35, 186]. As early as 2002 SiPix reported that it produces its fully assembled flexible panels with reel-to-reel process with an output capacity of greater than 3 metres per minute [90, 186, 187].

2.4 Sensor technologies for sensitive displays

Direct interaction with displays by using touch input has been around for over two decades [17]. Most of the early solutions provided touch input support with only one point of contact (known as ‘single-touch’). Single-touch proves too restrictive for a range of novel interactions [59, 75, 96, 122, 182, 184], which have spawned a range of proposals for touch input technologies supporting multiple points of contact – ‘multi-touch’ [62]. Initial multi-touch platforms were based on camera systems tracking users’ hands and fingers. The combination of high resolution input and computer vision algorithms results in very

flexible and robust implementations of camera based systems [59, 75, 99, 181, 182]. Akin to projective displays, touch input which is based on camera systems suffers from spatial constraints, occlusion problems and other environmental limitations. In addition, these systems incur significant computational demands of processing high resolution images and problems of motion blur [182].

The associated drawbacks of camera-based systems have resulted in a number of parallel research efforts to develop non-vision based multi-touch displays. One approach is to embed a multi-touch sensor of some kind behind the surface of a projected display [36, 87, 122]. Since these systems are based on projection displays, they bear all their associated disadvantages.

Several implementations have been based on transparent overlay systems. The Lemur music controller from JazzMutant [86] uses a proprietary resistive overlay technology to simultaneously track up to 20 touch points. The Philips Entertaible [63] uses infra-red (IR) emitters and detectors placed on a bezel around the screen to detect up to 30 touch points.

Ultimately, for maximal portability and flexibility, multi-touch needs to be monolithically integrated with the FPD system. Recently, there have been a number of successful implementations towards this integration. Toshiba has reported a 2.8" LCD Input Display prototype which can detect shadows resulting from fingertips on the display [123]. In this approach, photo-sensors are integrated onto the LCD glass substrate. Furthermore, a 3.5" LCD prototype with the ability to scan images placed on it has also been demonstrated. Integrating multi-touch sensitivity on larger displays has been demonstrated with ThinSight [62], where LCD displays of arbitrary size can be integrated with multi-touch sensing arrays. ThinSight employs a two-dimensional array of IR emitters/sensors placed directly behind the LCD backlight, and covering the whole display area. As in the case of controlling individual pixels in FPDs, matrix addressing becomes necessary for larger arrays of sensor elements. Further on, matrix addressing exposes its inherent problems in achieving row multiplexing that is free from side-effects. Under various conditions, simple passive matrix addressing schemes enable unwanted current paths between adjacent sensing and emitting elements. In order to resolve such issues, ThinSight borrows active matrix addressing from the traditional FPD design methodology. Scalability issues are resolved by tiling smaller sensor-array boards to cover large display areas.

Monolithic integration of multi-touch sensing with FPDs has much greater potential. TFT substrates exhibit photo-sensitive properties – photo current is generated when external light is radiated on the a-Si or p-Si materials [185]. This effect has mainly been considered as disruptive where making conventional LCDs is concerned, and special care is taken over minimising its effects on the display quality. Conversely, with the recent popularity of multi-touch interaction, researchers are focusing on maximising TFT sensitivity for implementing truly integrated multi-touch sensor displays. Samsung Electronics reports customising a-Si technology for sensing ambient light for automatic dimming of displays, which is also reported to be used on making multi-touch sensitive displays [78]. On the other hand, a-Si TFT was also used to develop a 4" QVGA (320×240) LCD with an embedded color image scanner [185]. This panel makes it possible to scan documents with scanning resolution equal to display resolution. The newly proposed TFT integrates sensing and display functionality onto the same substrate and can be fabricated by the conventional a-Si process.

Monolithic integration is also suitable for OLED-based displays. Generally, LEDs exhibit photosensitive properties when reverse-biased [115]. This property allows LEDs to have a multiplexed role - as light emitters and photo-sensors. A prototype 0.7" monochrome display where LED pixels have this dual role is reported on [67]. Reverse-biased LEDs are not very sensitive, and therefore result in poor system performance in environments with insufficient ambient light. Specialised photo-sensors generally give better performance. In [15], polymer LEDs and polymer photo-detectors have been integrated in a common substrate for demonstrating a sensor with touch and proximity detection functionality, with a range of up to a few centimetres.

This intrinsic integration of multi-touch with the display device brings unique cost advantages – the new systems would only require additional control circuitry for the sensing functionality, which would have negligible cost overheads for the overall system. The reported sensor technologies integrated on display TFTs and OLED displays open up the opportunity for their use on flexible substrates. Although there are still no reports of implementations of multi-touch sensing on flexible substrates, the underlying technologies have been demonstrated, thus opening up the possibility for a truly flexible sensitive display surface.

It is clear from this survey that the necessary technology for thin, sensitive display surfaces is gaining ground. Computation is necessary to make these surfaces interactive. Although the basic components for building flexible electronic circuits have been demonstrated, their performance is insufficient for designing processing units that would support interaction with the surface. The upper limits in charge mobilities currently give a bleak prospect for processing applications. On the other hand, although CMOS circuits are rigid, they do not pose any limitations for use in flexible substrates. Rigid circuits of compact size are often connected to flexible substrates. Consequently, computation and communication electronics for thin, sensitive display surfaces can be implemented with efficient CMOS technology.

2.5 Related work

The vision of large format displays and interactive surfaces has attracted the attention of many research communities. The very problem of constructing large format displays is investigated by display manufacturers. On the other hand, researchers in Human-Computer Interaction (HCI) investigate the usability of such displays. An immediate demand that is revealed with large format displays is larger computational requirements corresponding to larger rendering tasks. Consequently, scalable computing architectures are required to efficiently render and drive displays with ever growing resolution.

2.5.1 Scalable display architectures

In addition to the bright prospects of emerging FPD display technologies, research in display architectures that support scalable resolutions has also been prompted by other factors, with the main ones including:

- *Data visualisation applications* – The ability to visualise large datasets has been very beneficial to many research fields such as genomics, astrophysics, geoscience,

biological engineering, etc. These disciplines are experiencing a rapid expansion of collected data, and thus requiring advanced visualisation capabilities for more efficient data analysis.

- *Unsuitable commercial solutions* – In response to specialised needs for high performance systems supporting high resolution displays, commercial solutions have often resulted in extremely expensive and often inappropriate systems [19]. The traditional approach has been to construct large, tightly-integrated graphics systems, offering connections to multiple, but limited video outputs. The Power Wall at the University of Minnesota [169], the Infinite Wall at the University of Illinois at Chicago [30] (both driven by SGI Onyx2 [114] with multiple graphics pipelines) and IBMs Scalable Graphics Engine [117] are examples of implementations using specialised, inflexible architectures.
- *Usability research* – Investigation of large displays’ effects on productivity, user experience, etc [31]. The recent affordability of relatively large displays for personal computing environments has motivated researchers to investigate the use of even larger ones.

As mentioned in Section 2.1, one physical limitation of display resolution comes from the memory size of the frame buffer. Additionally, the computational resources for rendering the display content provide a ‘softer’ upper bound for the output resolution. Depending on the number of processing units and the organisation of the frame buffer, computing architectures with display support can be categorised into four groups, presented on the table below:

	Centralised frame buffer	Distributed frame buffer
Centralised processing	<ul style="list-style-type: none"> • many embedded systems such as mobile phones, etc • desktop workstations • game consoles 	X Windows System [134] and VNC [183] servers concurrently serving to multiple clients
Distributed processing	Intensive rendering applications (e.g. computer generated movies, etc.)	Scalable display platforms, represented by some implementations of display walls and computing surfaces.

It is important to emphasise that, this table categorises systems with GPUs as systems with centralised processing. Although GPUs may have several (hundreds of) processing units, they are often specialised and remain highly inflexible and difficult to scale. In order to provide a scalable architecture in both, rendering performance and display resolution, the only feasible solution in sight is to allow both, distributed processing and distributed

frame buffer. Consequently, the architectures of interest belong to the lower right cell of the presented table.

Among systems with limited scalability, IBMs Scalable Graphics Engine [117] allows the reception of pixels streamed over computer networks to synchronously drive an array of displays. The developed implementation is limited to 16 1GigE network inputs and 4 DVI outputs. At the core of current commercial solutions for larger display walls are video processors (also referred to as video servers) that are provided by companies such as Pixell³ or 9XMedia⁴. These highly integrated systems aggregate a large number of video inputs in various formats (e.g. VGA, S-video, Ethernet, etc) and forward them to an array of DVI outputs that drive the actual display wall. As they receive streamed pixels, the main activity of the system is to forward the input to the correct location in the display wall. Currently they achieve resolutions of up to 200MPix (with up to 64 DVI outputs).

The Scalable Display Wall Project at Princeton University [24], combines a network (Ethernet and/or Myrinet [13]) of multiple commodity processors to construct a parallel rendering system capable of driving multiple displays. Motivated by the idea of building a scalable architecture with inexpensive commodity parts, this architecture provides scalability in rendering performance and resolution. An implementation of this system comprises of 24 projectors arranged in a 6×4 grid to form a seamless image over a rear projection screen. The cumulative resolution of this implementation is approximately 18MPix (6K×3K) [24].

Alignment is a concern for projector-based display walls. Since manual alignment is tedious, considerable research has been done on automatic alignment systems based on vision-based feedback [22, 23, 25, 174]. Additional issues such as colour matching arise and stress tile discontinuities. Colour matching is important for the seamless appearance of tiled displays. [173]

Researchers at the University of Illinois at Chicago have developed a display system called LambdaVision, consisting of an array of 11×5 tiled displays with a total resolution of 100MPix. Lambdavisision is driven by a networked architecture referred as LambdaBridge [175]. Alongside, a graphics architecture named SAGE – the Scalable Adaptive Graphics Environment is developed for rendering on this system. The computing and networking infrastructure consists of a number of rendering resources (from single desktop computers to clusters of local or distributed PCs capable of rendering graphics either with dedicated graphics hardware or software), connected over the network to the scalable frame buffer – LambdaVision [74].

Many of these scalable graphics architectures are accompanied by similar graphics engines. Complex 3D rendering tasks have motivated the design of graphics engines that utilise a flexible amount of computing resources. WireGL [69] is a prime example of such scalable engine which unifies the rendering power of a collection of graphics accelerators in a cluster of workstations, treating each separate frame buffer as part of a single tiled display. A single serial application is allowed to drive a tiled display by streaming graphics primitives that will be rendered in parallel on display tiles. Its successor, Chromium [70] supports more strategies for distribution of rendering tasks, resulting in a more flexible and efficient engine. Despite this increased flexibility, Chromium is not designed to execute

³PIXELL: <http://www.pixell.com/>

⁴9X Media, Inc: <http://www.9xmedia.com/>

multiple applications on single displays. Its applications have a static layout on tiled displays. Chromium can divide the tiled display into several parts and execute multiple applications, but each tile can only support one application. SAGE [74] provides another solution to scalable graphics engines addressing more specifically the heterogeneity and scalability issues. The approach taken in this work is to decouple rendering from display, which brings greater flexibility in managing such resources. However, this decoupling comes at a huge communication cost and is therefore not suitable for this work. Another similar graphics engine that utilises a flexible collection of computing resources comes from the Virtual Reality (VR) community, supporting the development of an interactive scientific visualisation tool [51].

Several projects have been focused on exploiting parallelism to accelerate video decoding. [117] presents a customised parallel MPEG decoder dedicated to run on network platforms using IBM Scalable Graphics Engine. An efficient parallel MPEG decoder is presented in [24] and uses a hierarchical decoding system to address computational bottlenecks of parsing large MPEG streams.

These architectures and the corresponding graphics engines address the scalability of the display system at a coarse-grained level. The resulting graphics engines utilise potentially large computing clusters for storage, control, and rendering tasks and are interconnected in high performance networks. While such resources may be necessary for many applications, they also pose a significant logistical challenge for more wide-spread deployment (computing clusters, etc). In addition, it is not clear how to adopt such architectures in potentially large, but thin form factor display systems. As a result, the adoption of such architectures for scalable interactive displays becomes unsuitable. In contrast, computing surfaces suggest much more fine-grained distribution of processing, memory and display.

2.5.2 Seamless tiled displays

Specialised systems that require resolutions that are not achieved by current display technologies rely on tiling individual displays together. Layout and packaging constraints of conventional LCDs preclude visual uniformity in such tiled configurations. Some systems use projected displays to overcome this problem. However, as previously mentioned, projective displays bring their spatial and environmental constraints, which often prevents them from being widely adopted.

A novel approach to seamless tiling of traditional LCDs has been described in [93], where arrays of shaped, moulded light guides are used to create a magnified image of the LCD screen. The output faces of the light guide array end up being larger than the LCD package, and thus allowing seamless tiling of an arbitrary array of conventional LCD displays.

Tiling has also been recognised as a promising approach in the manufacturing stage to resolve low yields of large LCDs. Some limited forms are discussed in [48, 56, 141]. The potentials of this approach have been demonstrated by the design and manufacturing of a 1×3 tile array, of a 16×9 aspect ratio, 852×480 format (WVGA), seamlessly tiled color active matrix LCD (AMLCD) [82]. Seamlessness is achieved by special AMLCD tile design, where access to tile row and column driver chips is moved along a single edge of each tile. This design allows seamless layout of up to $2 \times n$ configurations.

Tiling is also considered for OLED based FPDs. Production using shadow masks becomes rather difficult for large area displays using evaporation-based deposition methods. This is due to mask alignment problems for substrates that are spread over large areas. The first 24" AMOLED display reported in [164] addresses this problem by using smaller TFTs for building larger displays, in this case by using 2×2 TFT tiles. Further on, the world's first 40" full-colour AMOLED [124] was built by abutting an array of 2×2 low-temperature TFT substrates of 20-inch diagonal and then printing polymer OLED materials with ink-jet printing technology.

2.5.3 Interaction with surfaces

In conjunction with Mark Weiser's vision of ubiquitous computing [177], researchers in HCI have since proposed a range of novel interfaces that drift away from the omnipresent desktop metaphor introduced in the Xerox Star Graphical User Interface (GUI) [149]. An influential concept towards the newly proposed interaction techniques is 'Tangible User Interfaces' (TUIs), first presented in [72]. TUIs attempt to improve the interaction quality and bandwidth between people and the surrounding information. Some of the main interaction methods between users and computing environments in TUIs are based on Graspable User Interfaces [44]. Such interfaces allow direct control of virtual objects through associated physical handles referred to as "bricks". This enables virtual objects to be physically graspable. Many TUIs are motivated by the idea of transforming surfaces within the architectural space (e.g. desktops, walls, etc) into interactive surfaces [72]. Using a combination of Graspable User Interfaces and pervasive interactive surfaces, TUIs aim to offer an increased degree of control as well as comfort and intuitiveness of user input. Scalable sensitive displays are an essential component for practical implementations of these concepts.

Various elements of TUIs are best observed in digital tabletops. Due to their practicality in conducting human to human collaboration, tables often form the focal point around which meetings take place. In an effort to support and enhance this type of collaborative work, desktop activity has motivated researchers to explore the use of computationally enabled tabletops [18, 36, 122, 139, 153, 161, 170, 180]. Design guidelines are based on the understanding of people's working practices during traditional tabletop collaboration tasks [135].

In addition to TUIs, several beneficial practices for building tabletop interfaces have been proposed, with the most popular being tabletop territoriality (motivated by human territoriality) and casual groupings of workspace content [136]. In order to facilitate user interaction within each of such workspaces, a range of gestural multi-finger and hand interactions have been proposed [36, 122, 184]. The digital objects that are present in the workspace are directly manoeuvred with finger based gestures such as tapping, double-tapping, dragging, flicking, catching, rotation, scaling, etc., and hand based gestures such as 'horizontal hand', 'two corner shaped hands', etc. [184]. Akin to Graspable User Interfaces, these techniques aim to provide an impression of treating digital objects as physical (e.g. moving photos from one side of the table to another by dragging them). Furthermore, physical artefacts are often used to further enhance the interaction experience [122]. Such artefacts have their associated semantic roles in the interaction process and are usu-

ally represented by passive devices that contain distinct features that are recognisable by the sensing system of the tabletop display.

The tabletop metaphor extends to new horizons such as novel musical instruments. Reactable[★] [75] is a multi-user, electro-acoustic music instrument with a tabletop tangible user interface. Users interact with it by moving around passive objects of different shapes, which are distinguished from each other by the sensing system. The objects' dynamics and spatial relationships enable performers to construct and play instruments at the same time. The internal parameters of these virtual instruments are controlled by spinning the associated objects as rotary knobs.

There are three features of interest that are common to all emerging surface-based interaction technologies:

- *Locality of application activity* – there is a high degree of locality between the location of interaction on the surface and the area of the display that changes in response to that interaction. Moreover, the notions of territoriality and casual grouping in workspaces imply that information which belongs to the same application context is likely to be spatially localised. Consequently, although computing surfaces may extend to large areas, significant processing and communication activities of individual applications are assumed to be localised only onto relatively small areas (extending to users' workspaces that are using the same application)
- *Gesture based interaction* – gestures play a central role in these novel user interfaces. As a result, gesture recognition support is important for computing surfaces to extend to such applications. In addition, supporting physical artefacts would be desirable, in which case, FPD sensor implementations such as in [185] would be beneficial.
- *Direct manipulation with digital content* – based on TUIs that are developed for interactive surfaces, digital objects are expected to be dragged, scaled, rotated, etc. Such operations require inherent support for two-dimensional transformations of graphical objects.

Since computing surfaces are aimed to serve as an underlying implementation platform for interactive displays, they need to support the interaction techniques that are designed for this purpose. The common features that stand out from the work on interaction summarised above motivate the problems that are addressed in the forthcoming chapters.

2.6 Summary

Flat panel display technologies have the potential to evolve towards thin, conformable, sensitive displays. Thin film electronics is critical for FPD design and current Si-based solutions have a combination of manufacturing and performance disadvantages. Suitable replacements have been sought for decades, with much focus being put on organic semiconductors. Significant advances are continuously reported in structures that are based on hybrids of organic and inorganic materials. Such technologies promise to be cheaper, provide satisfactory performance and be compatible with reel-to-reel manufacturing processes. OLED devices are making rapid progress in their operating lifetime and manufacturing, and thus opening the possibility of replacing the all-pervading LCD technology.

In addition, they are reel-to-reel compatible. Further on, several FPD technologies were recently presented with sensors embedded in pixel resolution, demonstrating a feasible future for FPDs to also serve as input devices. Early prototypes of integrations of these technologies with reel-to-reel compatibility are also emerging. Uninterrupted work in this field offers realistic prospects towards efficient manufacturing of large, thin, conformable sensitive displays.

Although the technology for producing very large FPDs appears to be in sight, the underlying architecture reveals several scalability limitations. These limitations are first encountered in manufacturing, but more fundamental limitations in computing and communication infrastructure are just as prohibitive. Constructing larger displays using smaller units is sporadically adopted in various stages of display design. This is demonstrated by the use of tiling in manufacturing of some larger displays. Current implementations of large format displays emulate their functionality by tiling smaller, off the shelf units. Nonetheless, comprehensive solutions that address such limitations have not been reported.

The rest of this work presents computing surfaces. The interaction methods that are envisaged as appropriate by the HCI community motivate the work on gesture recognition and 2D transformations. More elaborate interaction methods rely on the notion of time, which motivates the work on time synchronisation.

CHAPTER THREE

SYSTEM ARCHITECTURE

The architectural foundations of computing surfaces are presented this chapter. Firstly, it presents the rationale behind building large computing surfaces by abutting smaller, self-contained computing tiles. Further on, it introduces the central idea behind this work - interconnecting computing tiles into a two dimensional mesh network in order to support arbitrarily large surfaces. For consistency with future chapters, a concise model of the system is presented. For a comprehensive evaluation of the presented work, a hardware reference platform using off-the-shelf devices is developed. Section 3.6 gives implementation details of the hardware prototype. Although this platform provides a very compelling testbed for demonstrating the validity of the presented algorithms, its limited size makes it difficult to reveal scaling properties. To address such issues, a comprehensive suite of simulations has been conducted.

3.1 Tiling

Chapter 2 emphasised the restrictions of current display systems. There are only a handful of supported resolutions, each wrapped into a display format standard. Most importantly, as presented in Section 2.1 these systems are not scalable. Tiling is recognised as a solution only in isolation. Seamless tiled displays discussed in Section 2.5.2 provide partial device scalability for displays, but their current interface to computing infrastructure presents them as indivisible displays. On the other hand, scalable graphics architectures discussed in Section 2.5.1 address the computing bottleneck. The communication bottleneck is also addressed through the use of distributed frame buffers. Such distributed frame buffers drive tiled arrays of off-the-shelf displays with the purpose of emulating the work of one large display. Connecting such scalable display architectures to seamless tiled displays is currently not possible. It will remain the same unless architectural details of the tiled displays are revealed at the interface.

There is something very natural to the process of tiling smaller displays for building larger ones; this is the very principle of building individual FPDs - pixels are abutted

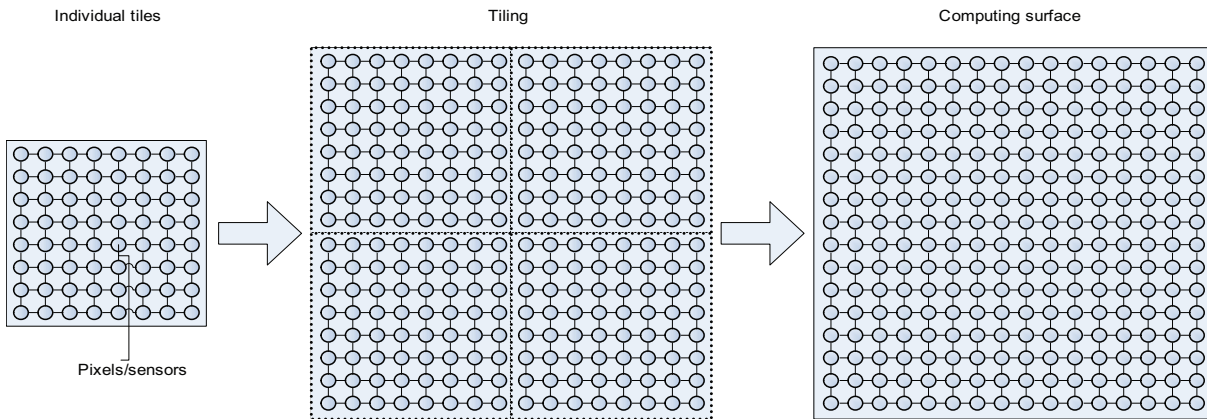


Figure 3.1: Using smaller tiles for building larger displays. The individual points can be pixels and/or touch sensors.

next to each other to form the display matrix. Tiling displays takes this approach one step further by placing replicated groups of pixels next to each other (Figure 3.1).

Similar arguments can be presented for platforms that embed multi-touch sensitivity into such large surfaces. As demonstrated in [62], the tiling principle can be used to form larger structures using replicated individual blocks.

So far, our concept of a tile unit consists of a relatively small, sensitive FPD. As previously elaborated in Chapter 2, the problems associated to the scalability of display systems do not only concern display devices but also extend to computing resources driving them. Put simply, *displaying vast quantities of information requires vast quantities of computing and display resources for processing and presenting such information*. Moreover, computing resources are required at all stages of information preparation and presentation. Computing surfaces should be flexible on the span of the presented content. In one hand, the surface should allow for it to be considered as one display entity. On the other hand, it should also support individual, tile level activities at the maximum rate as if they would be individual displays. Consider user scenarios based around table-tops (Section 2.5.3) – a potentially large number of users may be simultaneously using the surface. The users query for information and the system responds by returning the results in some proximity to where the query was made. In a centralised architecture, all the requests from all the users would have to be queued for servicing at some central resource, which immediately results in communication and resource management overheads. A distributed approach with many computing resources available for servicing these requests would reduce such overheads. We argue that computational resources at the tile level would be beneficial.

From the user perspective, a computing surface is a uniform, uninterrupted platform where the tile composition is only an architectural construct, and therefore should be oblivious. This uniformity is essential to the concept of computing surfaces and needs to be supported at every level of abstraction. As a result, tiles will have to collaborate with each other to present this uniformity abstraction. Collaboration involves communication which requires an appropriate tile interconnection infrastructure to be deployed within the surface.

The previous user scenario involves communication transactions with some service which may be located anywhere. In order to facilitate such transactions, tiles need not

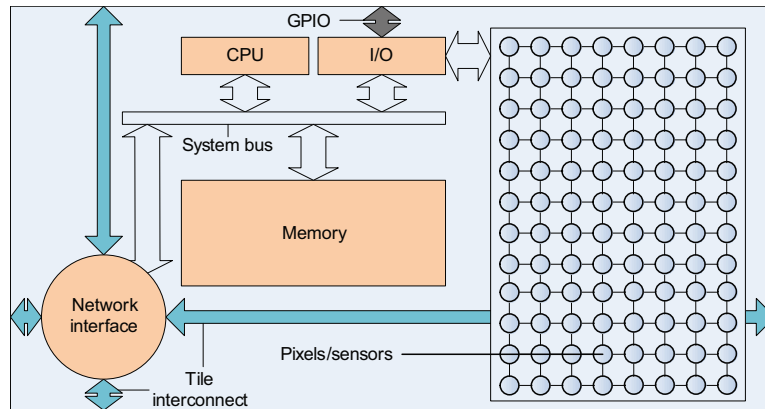


Figure 3.2: The structure of individual tiles in computing surfaces

only communicate with each other, but also to communicate with the “outside world”. This, external communication can be achieved through some I/O mechanism available at tile granularity. Depending on application needs, only a few tiles may need to be simultaneously connected to other communication terminals as they can share data with other tiles from the same application region using tile interconnect. Tiles that serve as input ports for such communication are referred to as *source tiles* and the ones that serve as output ports are referred to as *sink tiles*.

The need for processing and I/O at tile level brings an elusive question: Which tiles need processing and which tiles need I/O? Since each application has unique processing and communication requirements, the answer lies with the range of targeted applications. As the range of target applications can evolve with time, it is critical for the architecture to be flexible. Maximal flexibility is achieved by providing *every* tile with processing and I/O. The approach taken in this work is to provide homogeneous computational resources for all tiles. The proposed structure of these tiles is illustrated in Figure 3.2. A computing surface architecture composed of a network of identical, self-contained, processing tiles provides a generic execution substrate for a wide range of applications. In addition, it also brings comprehensive architectural uniformity and design simplicity.

3.2 A scalable network architecture

The performance of most modern digital systems is limited by their communication [34]. As it will become clear in later sections, many applications of computing surfaces have significant communication demands, and an appropriate selection of interconnection network is crucial for achieving good performance.

Since a scalable display platform is sought, it is imperative for the surface interconnection network to also be scalable. Bus interconnects suffer from poor scalability, and therefore are not a good fit [34]. Among the first steps towards selecting an appropriate network architecture is the choice of topology.

Our choice of topology is strongly influenced by the physical arrangements of individual tiles in the computing surface and the desired thin form factor. The static arrangements of channels and nodes has a significant impact not only on packaging constraints, but also on the performance of the interconnection network. Considering this physical layout,

it is rather obvious that a two dimensional mesh topology structurally fits the best. In butterfly topologies, terminal nodes communicate with each other via intermediate crossbar switch nodes. Placement of such intermediate nodes impedes the form factor uniformity of the surface and makes it very difficult to map this topology to an arbitrarily sized $(n \times m)$ array of tiles. Among form factor problems, this immediately reduces the desired flexibility of choosing the network size. In addition, path diversity is rather poor and packaging may result in very long wires [34].

Mesh topologies are attractive for many reasons. The regular physical arrangement of nodes is well suited for placement on a physical space using uniformly short wires, hence resulting in relatively low packaging constraints. The 2D mesh topology is particularly well suited for two-dimensional physical surfaces due to a clear correspondence between the topological and physical locality. Logically minimal paths are always physically minimal, which allows the exploitation of physical locality for local communication patterns. In comparison to logarithmic networks [34], they have larger average hop-count, which gives them slightly higher latency than the minimum bound. However, the application locality discussed in Section 2.5.3 instigates more network traffic among proximal nodes (Figure 3.3). Consequently, this overcomes the hop-count drawbacks of this topology.

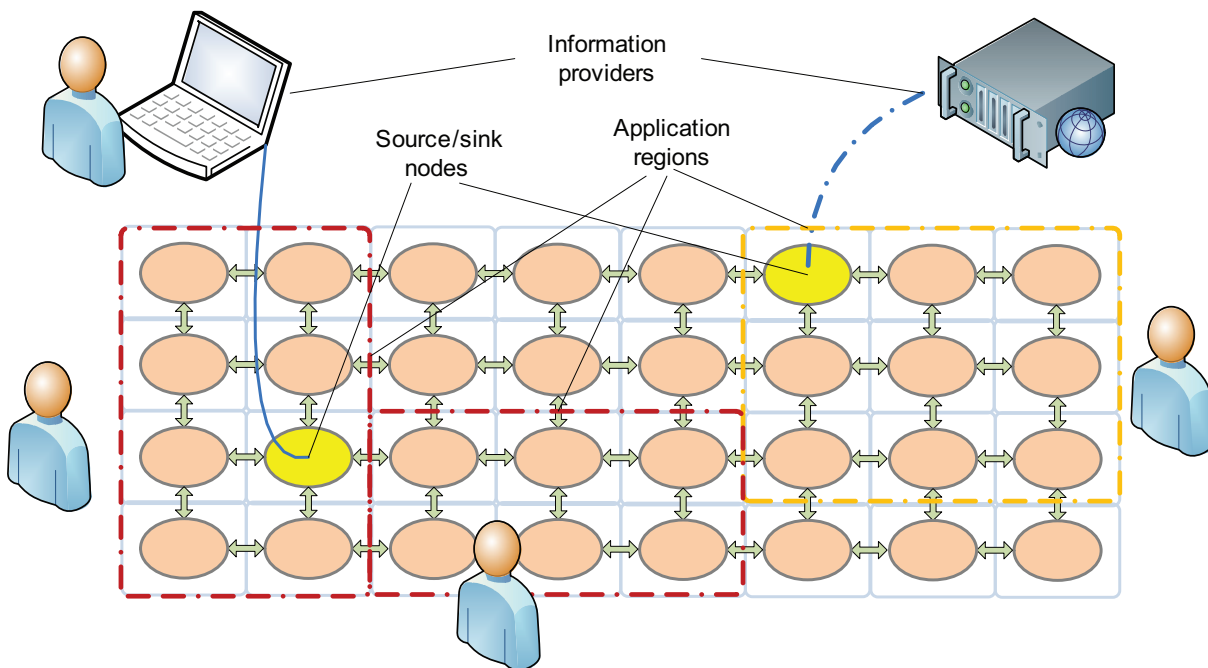


Figure 3.3: The computing surface infrastructure

The 2D mesh topology produces a direct network where each node contains a tile and a network subsystem. The network subsystem is composed of a network interface and a network router capable of receiving packets, determining their destination and forwarding packets to the appropriate neighbouring node. Details of the router architecture will be given in Section 3.6 and 5.5.

The combination of in-tile granularity of computing resources and the 2D mesh network topology offers computing surfaces two compelling properties: homogeneity and autonomy.

Homogeneity

The combination of the use of identical tiles and the choice of 2D mesh topology makes the system architecture comprehensively homogeneous. While the question of whether such homogeneity is advantageous deserves a separate comparative study, some benefits are already apparent. A single replicated processing element offers ease of silicon implementation and economic efficiency, and suggests a well-structured constraint on the software environment. This regularity will be exploited in the developed algorithms presented in the oncoming chapters. In contrast, managing heterogeneity with thousands of threads may make an already difficult problem intractable [9].

Autonomy

An interesting property of the computing surface is its manifestation of autonomy in multiple levels of abstraction. Individual tiles can be treated as self-contained units of computation, capable of execution even in a solitary environment. Furthermore, a collection of interconnected tiles, forming any subset of a given computing surface, performs its tasks autonomously from any other part of the computing surface (Figure 3.3).

This networked approach offers other advantages that are not available in most of the contemporary single-user computing systems. With careful design, hardware faults that are associated to individual tiles can be isolated in order not to affect the functionality of other tiles in the computing surface.

The combination of a self-contained tile unit and a 2D mesh network also allows *on the fly extensibility and reducibility* - two instances of computing surfaces that have the same building blocks can be connected to each other, resulting in one larger surface. In the same manner, with correct implementation of algorithms, a computing surface can be split during run-time into two smaller surfaces without any disturbance to the processes running in tiles not directly affected by the split.

3.3 Implications

The suggested architecture has profound implications for the way information is processed in computing surfaces. The replication of processing elements suggests some resemblance with supercomputers (at least in the number of processors) but that's where the similarities end. On the other hand, comparing them to more conventional computing devices makes these networked tiles even more alien due to differences in the way information is received, processed and presented.

Some examples of such differences are

- *User interaction* - when interacting with computing surfaces, the user information is received from individual tiles. Although tiles are the building blocks of computing surfaces, they are not expected to be serving users individually. Instead, the information entered by the user may be part of some larger context. Consider a UI Window (with a "close" button) spread among many tiles. When the button is pressed (touched), the tile hosting the event handler of the button receives the "press" message from the sensor input unit. The source of this message is in one tile but it concerns at least all the tiles displaying the window.

- *Communication with other systems* - in another scenario, the user may press the “receive mail” button, which sends a request to the email server, and in response, the received emails are presented on a display region spread among several tiles. Again, considering the homogeneity of the surface, this request needs to be sent off by one of the sink tiles through its I/O and received by one or more source tiles.
- *Processing* - the previous two points are concerned with input and output of information in computing surfaces. The next major difference is the location where this information is processed. In the previous email example, when the email information is received from I/O, an appropriate email window needs to be created and displayed. A simple implementation that mimics conventional architectures would assign a control tile that does this processing and sends appropriate messages to the tiles covering the area where the window is to be presented.
- *Multi-user support* - as these envisaged computing surfaces can stretch out for meters (and beyond) and cover large desks and walls, they are intended to be used by many users simultaneously. In contrast, most of the available computing systems and the corresponding applications are intended to be used by individual users. Servers may seem to be an exception, but the actual interaction with these machines is still done via other single-user devices serving as proxies.

Surface homogeneity raises many questions about execution efficiency. This architecture may not be suitable for all the applications that require high performance sequential execution. Individually, each tile is poorly resourced to do any significant work, resulting in a performance bottleneck. This affects many applications designed for conventional architectures. Although it may be too ambitious to think of computing surfaces as a replacement for desktop computing, it still provides a credible infrastructure for use as thin clients for desktop applications. On the other hand, the limited available resources on each and every tile may be aggregated into potentially enormous capacities. As previously noted, the utilisation of these aggregate resources requires tile collaboration, which involves communication. Considering the relatively high cost of communication relative to computation, many applications that have high messaging demands may have communication rather than processing bottlenecks.

These new challenges also bring new opportunities. Surface homogeneity brings the opportunity to migrate or replicate any execution thread running in any physical processor. This can be exploited to build fault tolerance mechanisms for running software systems.

3.4 Communication considerations

Computing surfaces provide an underlying architecture for building scalable, thin, interactive display surfaces. Continuous technology advancements discussed in Chapter 2 could bring such surfaces to reality. Consider the possibility of covering a whole building with such display surfaces. If the building has 100,000 m² of surface area on the walls, ceilings, and floors, it could accommodate around 0.8TPix of display at (72 dpi). Assuming that each pixel is updated at 30Hz, the complete display system must be capable of updating 24TPix every second. At four bytes per pixel, this display would require a

staggering communication bandwidth of $4 \times 24 \text{TBps} = 96 \text{TBps}$. These numbers remain overwhelming despite exponential improvements in recent technologies. Nevertheless, in most reasonable scenarios, only a small portion of the display fabric needs to be updated at full resolution, at maximal frame rate. Yet the given example reveals the scale of communication in such systems.

Computing technologies are limited by communication bottlenecks at many levels of abstraction. At the architectural level, the traditional memory hierarchies address the latency problems of communication between processing and memory units. In the micro-electronics level, on-chip communication is becoming a bottleneck as technology scaling favours transistors over wires [73].

Computing surfaces are expected to communicate vast amounts of information under such limiting conditions. Many tasks will be executed across several tiles, which may further increase communication demands. It is of utmost importance to take into account communication limitations whilst designing algorithms. Any communication overheads will lead to the reduction in available bandwidth for other applications and wasted energy. Consequently, communication efficiency is of high priority for all the algorithms that are presented in this work.

3.5 Software architectures for computing surfaces

An essential component of any computing system is its underlying software architecture. A software framework is necessary to support efficient development of software applications. Frameworks that are aimed for high resolution interactive displays with multi-user support have been investigated by the table-top research community. Among them T3 [168] and the Buffer Framework [71] serve as a good starting point for investigating software architectures for computing surfaces.

T3 is a software toolkit that enables the rapid development of high resolution multi-user table-top applications for collocated and remote collaborators. T3 has been initially designed for a multi-projector system, but the underlying framework is able to support any form of tiled displays. The tiled composition of the display is hidden from the developer. There are two main components of the T3 system - the T3 API and the T3 client. Applications are created using a single, flexible Java based API - the T3 API. The T3 client controls the actual display - it has no limitations on the resolution or the number of tiles it supports, which makes it conformable with computing surfaces. It communicates with the application through the T3 API by receiving tile updates and sending user interaction information. The API and the client in the current implementation of T3 run on the computer that hosts the display system. Computing surfaces could be interfaced for using T3 in a thin client approach. To the T3 platform, computing surfaces would be presented as one large framebuffer. Rendered display information would be sent over to the surface. On the other hand, user input information would be routed through the computing surface network infrastructure to the computer that hosts the T3 toolkit. Whilst this toolkit makes the application development very easy, the solution is not very scalable since all the processing is done centrally, thus not utilising the available processing in the computing surface.

The Buffer Framework is another suitable software architecture for table-top applications with multi-user support [71]. A typical user interface consists of two entities visible

to the user: visualisation objects that are typically carrying the displayed information and interface components for organising and interacting with visualised content. Essential to this framework is the concept of buffers that store information about properties of visualisation objects (e.g. object size, orientation, colour, etc). Here, interactions and animations of visualisation objects are derived from the information read from the buffers rather than (traditionally) being controlled directly by the interface components. This relieves the interface components from complex administration and steering tasks, thus reducing the complexity of the system. On the other hand, the interface components modify the buffers (based on the user input they receive), thus indirectly affecting the visualisation objects. An advantage of this system is that visualisation objects are modelled as separate entities only with local awareness. In computing surfaces, this framework would allow for a more distributed implementation of interfaces, thus allowing the computing surface to perform some degree of local processing.

Many implementation details of the software in oncoming chapters have been influenced by the Buffer Framework. However, a comprehensive implementation of a complete software framework is outside the scope of this work and remains an important part of future work.

3.6 Evaluation platform

Two methods were used for the evaluation of proposed algorithms - a hardware prototype and a corresponding simulation model.

3.6.1 A hardware prototype for computing surfaces

The computing surface architecture may be embodied into a wide range of hardware implementations. On one side of this spectrum are display walls with large, tiled LCD screens. Since multi-touch sensitivity is not widely deployed in current large format desktop displays, such an implementation would not offer any direct interaction. Nevertheless, display technologies with embedded touch sensitivity are gradually becoming available (see Section 2.4). Such display technologies would allow the construction computing surfaces that support direct interaction. This would enable applications such as large scale table-top displays. On the other end of the spectrum would be future implementations of flexible displays that are tiled and networked during the manufacturing stage. A hardware prototype consisting of 3×3 tiles has been developed. Due to economic constraints, a hardware implementation for this prototype was restricted to the most available technologies, which resulted in some compromises and tradeoffs. The display unit of each tile consists of Dell Ultrasharp 1702FP LCD monitors. The computing nodes that drive the displays were implemented in specialised hardware - Altera FPGAs in DE2 boards. There are two reasons for this decision:

1. At the very core of the architecture of computing surfaces lies the 2D mesh network and the corresponding radix-5 network router. Off the shelf radix-5 routers that can be plugged in to commodity PCs are not available. The development of such a device is a significant undertaking. On the other hand, HDL implementations of radix-5 routers that are suitable for FPGAs are widely available.

2. In addition to the lack of appropriate routers needed to construct a 2D mesh network, there are form factor implications. The physical size of PCs does not leave any room for constructing a thin display wall. In contrast, the resources that are used in the Altera DE2 board (FPGA chip, memory chips, VGA adapter, etc) can be accommodated in a very small package (relative to the size of the display) and would cause no spatial overheads.

The choice of specialised hardware brings its own implications. Although it demonstrates the viability of the architecture, it makes the implementation much more cumbersome. Furthermore, insufficient performance and the lack of supporting software infrastructure have proven to be very disadvantageous.



Figure 3.4: A display wall based on the computing surface architecture

The central processing unit and the memory hierarchy

In accordance with the choice of the FPGA development boards, a Nios II[®] [5] processor architecture was selected for the tile CPU. This processor architecture was chosen due to its mature programming and debugging tools and the abundance of available off-the-shelf peripheral connectivity interfaces (e.g. timers, serial communication interfaces, general-purpose I/O, SDRAM controllers, etc). The Nios II processor is a configurable soft-core RISC processor. Individual features can be added on a system-by-system basis to meet certain performance goals. In this implementation, a NIOS II instance contains 4kB of instruction cache and 4kB of data cache. For better performance a hardware multiplier, divider and barrel shifter are included. This processor configuration uses only 1852 logic elements (LEs) and less than 64kb of FPGA memory, which is less than 6% of LE resources (14% of memory resources). The CPU runs at 100MHz while the rest of the system runs at 50MHz. The CPU is connected with the main memory and other peripherals via an Avalon compliant system interconnect fabric (bus) [5]. All the peripherals are memory mapped. An 8MB SD-RAM module serves as the main memory for the system. An off-the-shelf DMA controller reduces the CPU load for operations involving many memory transfers. In the current implementation, support for non-volatile memory is not included. Finally, for many debug and configuration options, some basic I/O is supported with the use of on-board switches and LEDs.

The network subsystem and tile interconnect

The network router is composed of a collection of datapath and control circuits required to buffer and forward messages en route to their destination from the input port to the output channel. This implementation uses a radix-5, packet switched router with virtual channel flow-control which has been adopted from the Netmaker Islay architecture [109]. Modern router designs associate several virtual channels to one physical channel. Virtual channels improve performance by allowing active packets to pass blocked ones. They can also be used to avoid deadlock situations. In addition, they can be used to provide multiple levels of priority by separating different classes of traffic onto different virtual channels. Virtual channels come with the complexity of managing this abstraction, which has implications for the design of the router, but mostly on allocation mechanisms. The details of the router architecture will be discussed further in Section 5.5.

Network routers are interconnected with 16.7Mbps serial links. These serial links are interfaced with the router by serialiser circuits, with their core consisting of parallel in, serial out (PISO) shift registers. The switch allocator in the on-chip router assumes it is able to send a flit every cycle at any output port. This assumption is not valid in our application since the router works at much higher speed than the serial links. In order to resolve this, the switch allocator checks if the output port can take a new flit and invalidates the request if the output port is not ready.

The display subsystem

The DE2 board contains VGA digital-to-analog (DAC) converter chip, which allows plugging conventional PC monitor screens to the system. The display interface module consists of

- the Avalon interface
- a 512kB SRAM memory module serving as a frame buffer
- a VGA controller, which continuously reads information from the frame buffer and sends it to the VGA DAC chip.

The table below gives a summary of the tile components.

Component	Description
CPU	Altera NIOS®II processor
Bus	Avalon Memory Mapped Interconnect
Memory	SD RAM, 8MB
PC interface	RS-232 UART
Input device	PS/2 mouse interface
Network router	radix-5, packet switched router with virtual channel flow-control
FrameBuffer/Console	The interface to the VGA DAC chip with 512kB frame buffer that is also able to serve as a text console
Flash memory	Used for storing the FPGA configuration and the software code

Although this prototype does not immediately convey the thin form factor of computing surfaces, its building components offer this possibility. The complete tile electronics can be shrunk down to a System-on-Chip or a System-on-Package solution, which would fit in a thumbnail size. These solutions could be easily integrated with the display unit, resulting in the complete tile as one physical unit.

3.6.2 Simulation environment

While the intention is to provide a scalable solution for the given problems, the constructed hardware testbed is too small to give conclusive results for any scalability claims. Building a sufficiently large hardware testbed would have been prohibitive in cost and space. In order to reason about the performance of larger systems, a combination of careful measurements from the developed prototype and simulations of larger networks were conducted. There is a range of system level modelling languages that would be suitable for developing a simulation environment, including SystemC [159], BlueSpec [12] and SystemVerilog [157]. All these languages provide support for both simulation and synthesis. The fact that SystemC is based on C++, makes it appealing for adopting existing implementations of high level algorithms. However, both BlueSpec and SystemVerilog provide their own comparable alternatives. Network simulation environments are often built using these modelling languages. For the purposes of this work, a dedicated network simulation environment based on Netmaker [109] was developed. Netmaker is a library of synthesizable parameterized networks mainly dedicated for Network-on-Chip research. These networks are designed to provide packet-based communication for complex multi-processor systems. The aim of the library is to aid the accurate characterisation of potential router microarchitectures, routing algorithms and network topologies, and currently supports a wide-range of routers and networks. Controlling the parameters of

the network is done through configuration files in many levels of detail. The compilation and the execution of the simulations is done using the simulation kernel provided in Synopsys VCS ®[158]. Most importantly, Netmaker allows users to plug in any network system configuration and interconnect of choice. For the purposes of this work, the network subsystem used in the FPGA prototype described above is identical to the one used in Netmaker to simulate the network subsystem of computing surfaces that are larger than 3×3 . As a result, this simulation environment allows us to efficiently capture the communication requirements of the developed algorithms in larger systems. On the other hand, characterising the performance of larger systems needs to take into account the processing activity that occurs in source and destination nodes. The details of these measurements are reported on the individual chapters.

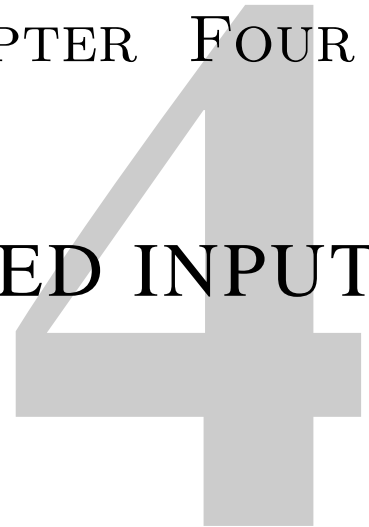
3.7 Summary

This chapter presents the architectural details of computing surfaces. Tiles are defined as basic units of sensitive displays with embedded I/O, computation and communication. In terms of form factor, it is assumed that tiles are thin and have no boundary margins in their sensitive display areas. Computing surfaces are created by contiguous abutting and networking of an arbitrary number of tiles. Tiles are interconnected with a 2D mesh network via embedded routers available on each tile. The use of homogeneous tiles and 2D mesh networking brings design simplicity.

For a comprehensive evaluation of the presented work, a hardware reference platform using off-the-shelf devices was designed and implemented. This platform provides a compelling testbed for demonstrating the validity of algorithms that are designed to run in computing surfaces. Its limited size makes it difficult to reveal comprehensive scaling properties, and therefore an appropriate simulation environment has also been adopted.

CHAPTER FOUR

DISTRIBUTED INPUT



An imperative component of any computing system is its input. In computing surfaces, computer-computer interaction is facilitated with the network component and peripheral communication ports. These ports could be used for indirect human-computer interaction, e.g. a keyboard can be interfaced through an intermediate computer to the computing surface. Chapter 3 states that the sensitivity of the surface constitutes an important part of the concept architecture of computing surfaces. Multi-touch sensitivity is envisaged to facilitate the main approach to direct human-computer interaction.

Section 2.5.3 reveals that gestures play a central role in surface-oriented user interfaces. It is therefore critical for computing surfaces to provide all the necessary support for achieving such interaction experiences. Current implementations of interactive displays are driven by conventional computing architectures and, as a result, the corresponding algorithms are not directly applicable to computing surfaces. For instance, gestures that are applied to a computing surface may be spread onto several tiles, where each of them is responsible for processing its local input. This fragmented information which is spread among several tiles needs to be aggregated for complete comprehension. Some form of tile collaboration is necessary to achieve this aggregation.

This chapter addresses the differences that the distributed architecture of computing surfaces brings to implementations of gesture recognition algorithms. Section 4.2 discusses the implications of the tiled network architecture for the subject of gesture recognition. Section 4.3 presents a universal software environment that conceals the tiled architecture and enables the use of conventional algorithms to process input information which is spread across any number of tiles. Section 4.4 demonstrates the validity of the presented platform on a distributed recognition algorithm for a specific set of gestures. Section 4.5 evaluates the presented algorithm and emphasises its low communication demands. But first, we begin with some related work.

4.1 Related work

Some insight into uni-processor implementations of gesture recognition algorithms will be beneficial for implementing support for gesture recognition in computing surfaces.

4.1.1 Handwriting and gesture recognition

Handwriting recognition is the task of transforming language-associated spatial structures of graphical marks or *strokes* into their symbolic representation. Similarly, graphical gesture recognition concerns transforming predefined strokes into corresponding symbolic representation. Computer based handwriting and gesture recognition is a mature discipline that has found many commercial uses.

Algorithms that receive image based symbols obtained from optical scanning of documents are referred to as offline recognisers. Conversely, systems which receive spatial data from an input system and directly perform the recognition are referred to as online recognisers. Temporal information is beneficial in the process of recognition and as a result, reported recognition rates are much higher for the online case in comparison with the offline case [118].

In one of the taxonomies presented in [118], algorithms for handwriting and gesture recognition can be classified into two families:

1. Structural and rule-based methods
2. Statistical methods

The first family is based upon the idea that a graphical shape representing a character or a gesture can be described in an abstract fashion without paying too much attention to the irrelevant shape variations that necessarily occur during the application. Due to the lack of robust and reliable rule definitions, this approach is successful only in limited gesture recognition [54] and rather unsuccessful in handwriting recognition.

In statistical methods, a shape is described by a fixed number of features defining a representation space in which different classes are described with probability distributions. Three groups of methods are based on this approach: explicit, implicit and Markov modelling methods. Explicit methods are derived directly or indirectly from pattern recognition approaches such as linear discriminant analysis, principal component analysis and hierarchical cluster analysis and are thus well supported mathematically [176]. Implicit statistical approaches generally refer to methods relying on artificial neural networks. Variants of this approach based on convolutional time-delay neural networks [27] are used in commercial solutions such as Microsoft Windows[®] handwriting recognition. Considering the successful application of Hidden Markov Models [121] in a wide range of pattern recognition problems, it is no surprise that this approach has also been productive in many handwriting recognition algorithms [118].

In computing surfaces, the tile running the recognition algorithm may be different from the tile that receives the input and therefore higher storage requirements result in a higher communication footprint. Assuming 300dpi scanning resolution, the storage requirements for offline recognisers can be a few hundred kilobytes. On the other hand, assuming 100 samples per second, online recognisers require storage in the range of hundreds of bytes [118].

4.1.2 Distributed gesture recognition

Gestures that are applied across a distributed system are generally not very common. This is mostly attributed to the fact that conventional distributed systems have not been developed for use with input systems dedicated to *observe* the same physical event or a set of correlated physical events. Nevertheless, new research in specialised systems such as [91] reveals the potential of distributing gestures. This work presents a peer-to-peer multiple-camera architecture for a distributed real-time gesture recognition system. It uses a network of relatively inexpensive cameras to gather images in order to provide high resolution at low cost. Computations are done at the embedded processors in each camera, with no centralised server. A methodology for transforming single-camera algorithms to multiple cameras is proposed. The significance of communication bandwidth and power consumption is recognised and reduction measures are taken at algorithmic levels. Although the distributed nature of target-gestures is in common with hand-based gesture recognition in computing surfaces, the context of this work remains different.

Distributed gestures also appear in [61] as patterns of activity contributed by multiple users and multiple devices, which take a new meaning when their occurrence has some specific time relation. An example of a synchronous gesture demonstrated in this work is the act of bumping one Tablet PC device into another one that is resting on the desk. As a response to this gesture, the display content of both Tablet PCs is unified. Such gestures are not directly related to the ones that appear in computing surfaces, since display tiles are stationary and don't change their relative position.

4.2 Implications of tiling sensor arrays

4.2.1 Providing recognition services

A system that provides handwriting and gesture recognition services needs to address several issues, including a suitable hardware input system, a recognition algorithm, reporting interface, etc.

The general details of the computing surface architecture are given in Chapter 3. The touch input system is assumed to be embedded on the surface, like the display system. The tile touch input system consists of a matrix of sensors, which feeds sensor data into a controller that performs any necessary signal processing. The controller output is a matrix representation of sensor data, with each element reporting the measurement intensity of the corresponding sensor. For our purposes, we assume a 1-bit measurement resolution, i.e. the area covered by one sensor is either touched or not touched. Sensor systems typically provide higher resolution measurements that can be reduced to bimodal states with various thresholding techniques discussed in [62]. In the event of sensed touch, measurement snapshots are stored in an *input data queue* (see Figure 4.1 in Section 4.3), waiting to be processed by a gesture recognition algorithm.

Modes of interaction between the user and the recognition service also need to be defined. One approach is for the user to write on a dedicated region of the surface, with recognised content going automatically to the correct application. Another is for all input to be local to the application. That is, all characters and gestures are written in writing areas that belong explicitly to individual applications. An appropriate programming

model that defines the interface for the input recogniser from the application programmer's perspective also needs characterising. There are a range of choices available in selecting the tightness of the coupling between the recogniser and the application. Some applications may record strokes themselves, invoking the recogniser using their own application-specific parameters and handling the associated response. Alternatively, other applications may not want to be aware of recognition at all, preferring to have the recogniser run separately and emulate a keyboard. Such applications can be supported by a standalone service application in which the user writes on a special writing canvas. On recognition, this service sends recognised text to the relevant applications.

For the purposes of this treatment, it is assumed that the user writes on a dedicated region of the surface. The gesture recognition algorithm runs separately from other applications. This approach is suitable to many existing applications and from the computing surfaces perspective, it has the compelling advantage of distributed processing – the recognition may be finalised in any tile, with the result being forwarded to a dedicated tile. This tile runs a service which interfaces with any application that inputs the recognition results.

4.2.2 Surface tessellation

Individual tiles may be too small to accommodate complete applied gestures. Besides, even if the tile size is sufficient, the user may decide to enter strokes in an area that spans more than one tile. This necessitates a mechanism that hides this distribution of the recognition area and presents the stroke fragments as one seamless entity, ready to be treated by an off-the-shelf recognition algorithm. Using the network provided, tiles can exchange information that enables the consolidation of these fragments into one seamless entity. A software service, represented in form of a middleware can be used as an interface between the tiled network architecture and the gesture recognition algorithm. This middleware service needs to be able to distinguish between the following events:

- beginning of stroke
- tile-to-tile stroke transitions
- end of stroke

To illustrate the necessity of such disambiguations, consider the case when a stroke is entered across one tile. Without any neighbourhood sensor information, it is impossible to distinguish if the stroke had begun on the edge of the current tile or if it continued developing from a neighbouring tile. A similar situation would occur when disambiguating the end of the stroke. Additionally, this software environment also needs to conceal tile transitions when a stroke develops from one tile to another. In order to be able to utilise off-the-shelf algorithms, the impression of one, uniform sensor input system needs to be provided or equivalently, the architectural construct of the networked tile needs to be hidden.

An apparent approach is to emulate a uniprocessor architecture. This could be implemented on a dedicated (processing) tile that performs all the processing with the rest of them only serving as input tiles. In this approach, while gesture strokes are being

applied to any input tile, the information received from the sensors is timestamped and immediately transmitted to the processing tile. An execution process instance that performs this operation would be placed on all input tiles that receive stroke input. Once the processing tile receives the complete stroke information, it will present it to the gesture recognition algorithm. For offline recognition algorithms, a bitmap reconstruction of the stroke would be provided. Timestamps associated to stroke information will be used to reconstruct the temporal information required by online recognition algorithms. The information associated with the earliest timestamp marks the beginning of the stroke. Similarly, the latest timestamp marks the end of it. This approach would implicitly deal with tile-to-tile stroke transitions by using a coherent coordinate system through all tiles.

Although this approach is achievable, it scales poorly with the increase of the stroke recognition area or decrease in tile size. Even that stroke sizes are usually limited, large stroke recognition areas enable simultaneous entering of multiple strokes, which increase the load of a centralised approach. For tiles with small size, the network traffic is increased since stroke information has to travel through more tiles to reach the dedicated processing tile. In both cases more tiles end up supplying data and recognition workload to the processing tile and hence creating a potential bottleneck. In addition, communication demands increase with any improvements in resolution. Furthermore, instead of embracing the power of online recognition algorithms, such systems would have to include a dedicated emulation mechanism to support them.

Better recognition rates and lower communication footprint make online recognition algorithms more attractive. In addition, these algorithms continuously build intermediate results while stroke information is processed. If an instance of such an algorithm is executed in every tile, it is possible to locally process tile stroke fragments. When the stroke develops from one tile to another, the intermediate recognition result needs to be communicated to the new tile. The processing state needs to contain enough information so the next tile can continue precisely where the current tile had stopped. This is achieved by communicating the last intermediate results and exit location coordinates. The precise amount of state information depends on the specific algorithm and usually consists of several parameter values (e.g. probability values) and state variables (e.g. Markov model states) but is comparably smaller than stroke spatial and temporal data. Consequently, local processing of stroke information results in lower communication demands. Communication demands are further decreased by the fact that this state information is sent to the neighbouring tile where the stroke continues developing. Finally, the recognition result is sent to a dedicated tile for forwarding to any applications that are subscribed to receive recognised gestures. Since communication is time and energy intensive, this approach promises to be faster and more energy efficient.

A distributed approach eliminates the need for a dedicated processing tile to perform gesture recognition. In spite of this, a dedicated tile would still be required - its new role is to interface to other applications that subscribe to recognised gesture input and consequently, it has to serve as a destination for stroke recognition results. We will continue referring to such a tile as the *control tile*.

The size of the gesture stroke depends on the type of application and user context. While strokes entered on PDA devices are relatively small sized, other strokes that may be entered on tabletops or interactive walls are larger. A scalable solution is desirable in order to cover this wide range of contexts. In addition, a flexible middleware should

encompass the advantages of online recognition whilst still accommodating any other off-the-shelf solution.

4.3 A universal platform

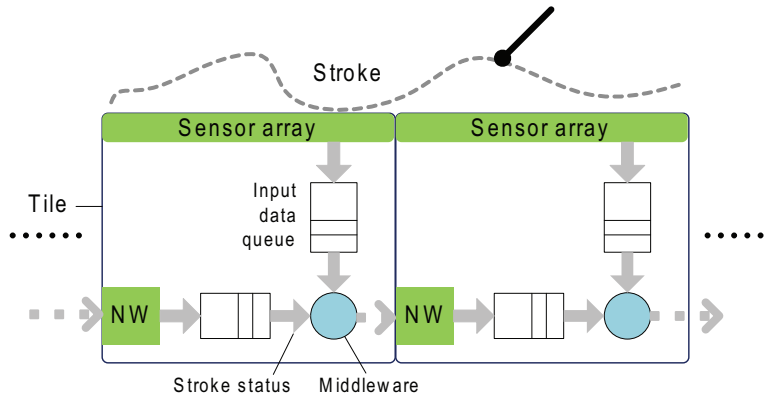


Figure 4.1: The distributed input system

The middleware for interfacing the tiled network architecture and the gesture recognition algorithm is implemented with a finite state machine. As depicted in Figure 4.1, any information about strokes being developed in the neighbouring tiles is received through the network interface. We will refer to this incoming information as *stroke data*. As mentioned above, for online recognition algorithms, the stroke data consists of the intermediate recognition state and the stroke exit location on the previous tile. On the other hand, for offline recognition algorithms this state may consist of raw (or compressed) sensor data. Upon arrival, stroke data is further processed with information coming from the tile sensor array input data queue. If the stroke develops beyond the current tile, the updated state is transmitted on the network to the tile hosting the next stroke fragment.

The middleware has to communicate with the neighbouring tiles to disambiguate between tile-to-tile stroke transitions and the start of the new stroke. Communicating the developing stroke data from one tile to another is unavoidable. However, communicating only stroke data among tiles is not practical to achieve this disambiguation. Consider the scenario given in Figure 4.2. The stroke starts at time t_1 at tile T1 and enters tile T2 at time t_2 . The processed stroke data from tile T1 will not be ready for tile T2 until some time $t_2 + t_{P1}$, where t_{P1} denotes the time taken for processing in tile T1 and the communication time from tile T1 to tile T2. Since tile T2 requires the stroke data from T1 to process its local sensor data, it will not be able to start processing before $t_2 + t_{P1}$. In environments with limited processing resources, t_{P1} in tile T1 can have an impact on the length of t_{P2} , thus potentially delaying further the time when T2 delivers stroke data to T3 ($t_3 + t_{P2}$). This process of increasing processing times continues throughout all the tiles where the stroke develops. Furthermore, tile T1 cannot begin processing local sensor data before inferring that the stroke did not extend from T0. A timeout period t_S can be used by tile T1 to determine if the stroke is extending from tile T0 – with no stroke data arriving after t_S , T1 can assume that the stroke begins there. However, a timeout period $t_S = \max\{t_{Pi}\}$ would have to cover any expected processing delay. As a result,

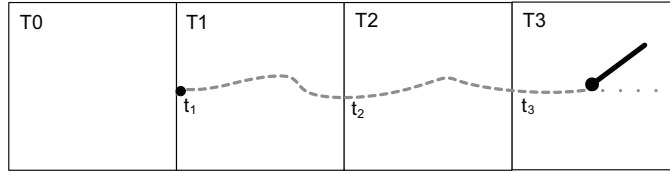


Figure 4.2: Disambiguating between the start of a new stroke and tile-to-tile stroke transition

tile T1 will not deliver stroke data to T2 before $t_1 + t_S + t_{P1}$. The problems appear if $t_1 + t_S + t_{P1} > t_2 + t_S$. That is, the stroke data from tile T1 arrives at the tile T2 after the timeout period in tile T2. Therefore, for $t_{P1} > t_2 - t_1$, in absence of incoming stroke data from T1, the tile T2 decides that a new stroke begins there. The time difference $t_{i+1} - t_i$ is in a range of milliseconds, and is proportional to the size of the tile. This problem becomes relevant in tiles with slow processing, or physically small tiles.

In order to avoid providing tight timing guarantees (i.e. $\max\{t_{i+1} - t_i\} < t_S$), early notification messages are used. Whenever local sensor data is received, the tile immediately informs the surrounding tiles about it. By responding with their local stroke processing activity, the surrounding tiles provide sufficient information to disambiguate between a start of a new stroke and tile-to-tile transition.

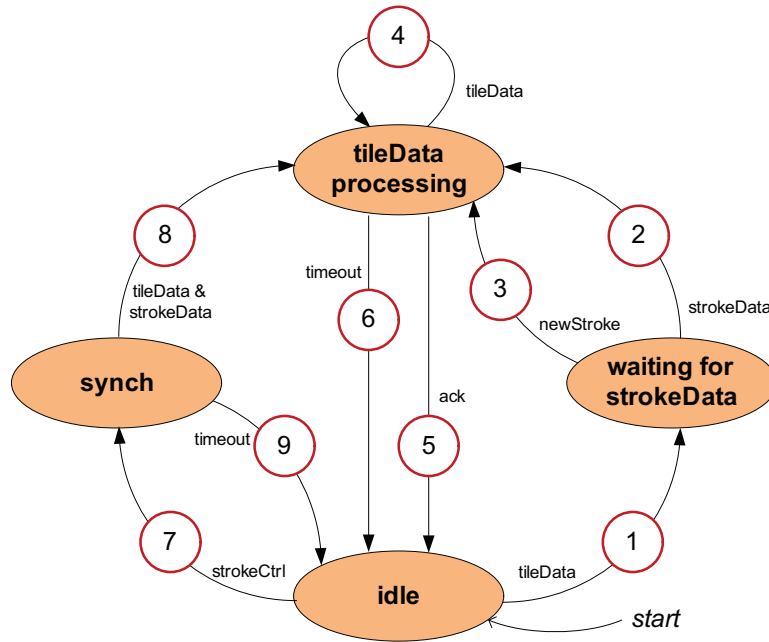


Figure 4.3: The state transition diagram for the distributed input middleware

Figure 4.3 presents the state transition diagram for a middleware solution which enables the networked tile architecture to synchronise local tile input with the stroke information from other tiles. The ‘tileData’ input signal corresponds to the input data queue having a minimal number of data points. The ‘strokeData’ input represents the receipt of stroke state information from the network interface. The ‘strokeCtrl’ input signal represents the receipt of an early notification message from the network interface. The ‘newStroke’ input represents the beginning of a new stroke. The middleware starts in the ‘idle’ state and processes its local input data in the ‘tileData’ processing state.

Two intermediate states ‘synch’ and ‘waiting for strokeData’ are used for synchronisation between network stroke data information and local stroke data. The ‘synch’ state is used to manage with early notification messages, whilst the ‘waiting for strokeData’ is needed for detecting new strokes. Details of individual actions that are associated with state transitions are given in the table below.

No	Action	Description
1	<i>notify neighbouring tiles;</i>	New data is received from the sensor input data queue. In order to disambiguate between the start of a new stroke and tile-to-tile transition, neighbouring tiles are notified and their response is expected.
2	<i>check exit point; ack the receipt of stroke data;</i>	Following the notification described in transition (1), one neighbouring tile responds that it is processing a stroke. At the end of processing, it transmits the stroke data to this tile. After verification of the exit point, this tile confirms the tile-to-tile transition and is ready to process the data from the sensor input queue.
3	<i>process data in the local input queue;</i>	‘newStroke’ signal is generated when all neighbouring tiles, having received the notification described in transition (1), respond that they are idle. Consequently, the tile begins processing the local stroke data as a new stroke.
4	<i>process data in the local input queue;</i>	Local stroke processing continues as more data is received from the local sensor input queue.
5	<i>(no action)</i>	When the local sensor data is processed and the exit point is identified, the processed state is communicated to the neighbouring tile. The receipt of an acknowledgement confirms that the stroke continues further.
6	<i>finalise;</i>	The end of the stroke is marked with no activity for a fixed period of time. At this point, the system finalises the stroke processing and moves to the ‘idle’ state.
7	<i>(no action)</i>	The neighbouring tiles respond with their stroke activity to neighbours’ notifications described in transition (1). As the stroke progresses further, the current tile may be affected. As a result, it moves into a ‘synch’ state that serves as a synchronisation point for local and network sensor data.

8	<i>check exit point; ack the receipt of stroke data; process data in the local input queue</i>	The developing stroke described in transition (7) reached this tile. In addition, the data associated with that stroke has also arrived and therefore the system is ready to process the local data from the queue.
9	<i>(no action)</i>	The stroke developed in the neighbouring tiles did not reach this tile, therefore the system moves back to the 'idle' state.

The central part of this execution environment consists of the processing of data from the sensor input queue. The implementation details are closely tied to the type of gesture recognition algorithm in use. For online recognition systems, this function wraps the actual recognition algorithm and reveals the incremental internal recognition state. For offline systems, it only accumulates the new information coming from the input data queue and merges it with the state information coming from the tile where the stroke had been previously developing. This new stroke state is further developed and forwarded till the gesture is completed in which case the complete stroke information is sent to the control tile where an offline recognition algorithm performs the recognition.

The fact that each tile processes its own sensor data can result in undesired behaviour when parts of the stroke simultaneously cover two tiles. Figure 4.4 depicts one such situation where the majority of the stroke is entered in the main (magnified) tile but a small portion of it covers the neighbouring tile (highlighted in yellow). Any processing from the neighbouring tile is not helpful to the overall stroke recognition since the main tile never stopped processing stroke information. Furthermore, the state update received from the neighbouring tile may be disruptive on the processing of the current tile. In order to avoid such situations, the system can discard any incoming stroke states that are received while there is already a developing stroke being processed. As in the example above, for small stroke fragments that contain insufficient information, the corresponding input data queue can be ignored. This situation motivates restricting the triggering of the 'tileData' event only in situations where the load in the input data queue exceeds a given threshold.

In addition to consolidating tile fragments of a single stroke, gestures that are composed of multiple strokes necessitate some ability to make individual strokes coalesce in a larger semantical construct. The fact that the next part of the stroke may begin in a different tile makes the problem more challenging. The location of the beginning of the next stroke cannot be determined in advance, so the next host tile is not known. Speculation can be used to address this problem by notifying a set of candidate tiles that may be receiving the next part of the stroke. Information about this set of candidate tiles can sometimes be extracted from online recognition algorithms. In a more conservative approach, a notification can also be broadcast to all the relevant tiles. This is an approach that has to be employed when offline recognition algorithms are used.

The duration of the timeout period in 'tileData processing' state depends on various factors such as network speed, input system speed and time gaps between tiles being revisited by strokes. Its lower bound is set by the network and processing speed, since the state needs to allow sufficient time for the neighbouring tile to respond with an acknowledgement. Since several strokes cross back on themselves (e.g. 'o', 'p', 'q', 'x', etc.), tiles

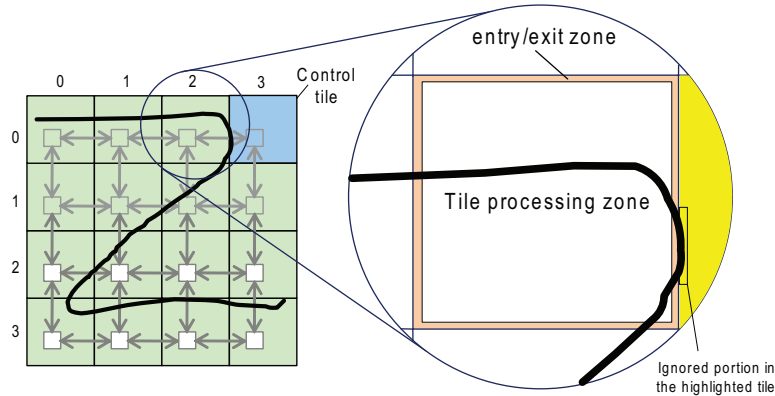


Figure 4.4: A gesture stroke distributed on several tiles. Borderline cases as the one presented in the magnifier need to be addressed appropriately

can be revisited by the same stroke. This sets the upper bound to the timeout period for a valid operation of the middleware. Similarly, the duration of the timeout period in ‘synch’ state is also determined by the network and processing speed. In addition, the size and the resolution of the stroke have an impact since this timeout needs to allow long and potentially slow processing.

When using this middleware, stroke processing is done by tiles that are part of the stroke trajectory (physical path). Furthermore, as the stroke gets developed, the corresponding host tile performs its recognition, which implies that online recognition follows the stroke development not only temporally, but also spatially. This in itself constitutes a curious property of processing locality in computing surfaces. In addition, the state update mechanism for consolidating global stroke information from locally processed data does not depend on tile size - the platform will be functionally valid even in the extreme case of one touch sensitive pixel per tile (assuming an input of the same thickness).

4.4 Case study: Distributed unistroke recognition

In order to demonstrate the feasibility of the suggested middleware platform, a distributed stroke gesture recognition algorithm is developed and tested. The set of *unistrokes* is selected as a set of stroke gestures to be recognised. Unistrokes were suggested by Goldberg and Richardson in [54] for entering information with styli and consist of one stroke. Entering unistrokes is proven to be faster than entering Roman letters. Basic strokes have been selected such that they are well separated in sloppiness space, i.e. can be robustly distinguished even when written sloppily. In addition to that, basic strokes are also simple, therefore easy to learn and can be entered without looking.

The alphabet of unistrokes is based on five basic easily distinguishable strokes. These are the strokes that correspond to Roman letters ‘i’, ‘s’, ‘f’, ‘n’, and ‘o’. Additional unistrokes are obtained by rotating base unistrokes by 90 degrees each time. Furthermore, these strokes can be written in two directions. By rotating and redirecting, basic unistrokes ‘s’, ‘f’, ‘n’, and ‘o’ can be mapped to up to eight different symbols. Assigning unistrokes to Roman letters is done with the intention of maintaining natural mapping to as many symbols on their ordinary representation as possible. Seven symbols map directly to their ordinary representation: i, j, L, o, s, v, z.

while the stroke is being processed. Let's assume that the sensor information produced by the tile is a list of time ordered local data points (representing local coordinates). By successive comparison of local data points, a list of direction vectors can be derived. In order to determine the direction of the stroke and mitigate sloppiness, the elements of this list are averaged. Unweighted averaging turns out to be very constrained as we are unable to control sloppiness. An exponential moving average [128] provides more control to the weight of particular elements of the list. The smoothing formula used here is based on the basic exponential moving average formula:

$$SDV_{n+1} = (1 - \alpha)SDV_n + \alpha d_n, \quad d_n \in \{N, NE, E, SE, S, SW, W, NW\} \quad (4.1)$$

SDV stands for *Smoothing Direction Vector* and accumulates weighted average data from the direction vectors d_n . d_n is calculated directly from the comparison of two consecutive data points (n and $n + 1$) and always takes one of the values from the set of base direction vectors $D = \{N, NE, E, SE, S, SW, W, NW\}$. Values of the smoothing constant α close to 1 have less of a smoothing effect and give greater weight to recent changes in the data, while values closer to 0 have a greater smoothing effect and are less responsive to recent changes. Specific values of α depend on the expected length of the segment and input system resolution.

The SDV contains sufficient information about the direction and the sloppiness of the stroke. When the stroke development moves from one tile to another, the value of SDV is communicated. As the stroke progresses from one tile to another, tiles update the value of the SDV with new direction vectors coming from the input system. In the current state, tiles have no information that can help recognise segment lengths, and as a result unintended strokes can be reported as recognised. In order to avoid this, an increasing counter is also communicated as an indicator of the length of the segment. The index n in Equation 4.1 is used for this purpose.

Recognizing compositions

Unistrokes that represent letters 'b', 'g', and 's' can be considered as compositions of interconnected segments. The aforementioned method for recognising segments can also be extended to recognise these strokes. This extension involves a mechanism to detect transitions from one segment to another. In addition, it involves a state machine to track the complete sequence of encountered transitions. Consider the unistroke for the Roman letter 'g' which can be perceived as being composed from two segments: a horizontal line heading east continued by a vertical line heading north. The algorithm first detects the horizontal line, followed by the vertical one. In the process of recognising the unistroke for this letter, the isolated recognitions of the horizontal and the vertical line are separated by a rapid change of direction from east to north. The detection of this abrupt change is necessary for reasoning about transitions from one state (entering the horizontal part of the unistroke) to another (entering the vertical part of the unistroke). Since, abrupt changes in the stroke direction will have an impact on the value of SDV, the associated state transitions can be detected by continuously monitoring this value. This monitoring

is done by testing the following condition:

$$\|SDV - \bar{d}\| < \epsilon. \quad (4.2)$$

The 1-norm vector norm is used for this application [66]. Under this norm, the difference between any two base vectors is 2, whilst the difference between any two vectors is not greater than 2. For the purposes of this algorithm, two vectors u and v are considered to have different directions if $\|u - v\| \geq 1$. In this formula, \bar{d} is the vector representing the direction being tested and ϵ is the error threshold. The value of the error threshold indirectly determines the amount of tolerated sloppiness. A developing stroke is considered to change its direction if its SDV fails the test in relation 4.2. This can be achieved by a sufficient number (k) of successive base direction vectors that are different from the standard direction course. After the application of k base direction vectors using Equation 4.1, the value of SDV will be transformed from some value SDV_n to SDV_{n+k} . Given the relation 4.1, it is easy to show that the difference between vectors SDV_n and SDV_{n+k} is:

$$\|SDV_{n+k} - SDV_n\| = \left\| \alpha \sum_{i=0}^{k-1} (1 - \alpha)^i d_{n+k-i} + (1 - \alpha)^k SDV_n - SDV_n \right\|$$

Assuming that all the new direction vectors are the same $d_n = d_{n+1} = \dots d_{n+k} = d$, the formula above looks like:

$$\|SDV_{n+k} - SDV_n\| = \|(1 - (1 - \alpha)^k)(d - SDV_n)\| = (1 - (1 - \alpha)^k) \|d - SDV_n\|$$

The base vector d can be considered to have a different direction to the cumulative value SDV_n if $\|SDV_n - d\| \geq 1$. Therefore, the application of k successive different direction vectors will result in $\|SDV_{n+k} - SDV_n\| \geq 1 - (1 - \alpha)^k$. This value can serve as a threshold value:

$$\epsilon = 1 - (1 - \alpha)^k \quad (4.3)$$

In practice, during change of direction, the sequence of direction vectors that are different from the current course of direction does not appear strictly as assumed above but rather combined with other direction vectors. Consequently, an overestimate $k' > k$ is used. While detecting the unistroke for letter 'g', after entering the horizontal part, depending on whether the new course is north or south, the outcome of the change from horizontal line can be either the stroke for 'g', 'h' or *unknown*. An immediate transition from horizontal to vertical does not give good results due to sloppiness. In order to address this problem, the state machine moves to an intermediate state. The transition to an intermediate state replaces the current SDV with a fresh one where information for the new course of direction is recorded. The intermediate state eventually results in a new transition to a stable state (corresponding to 'g', 'h' or *unknown*).

Including this extension to the recognition algorithm, the stroke recognition state now consists of: the current state of the state machine, the most up to date value of the SDV and index n .

Recognising curved strokes

Unistrokes corresponding to letters ‘q’, ‘x’, etc., are derived from the same base unistroke, which is composed of a curved line. Curved strokes can be approximated with compositions of segmented strokes and therefore can be recognised using the same approach. Figure 4.6 presents the recognition of the stroke that corresponds to letter ‘x’. Depending on the selected values of α , the state transitions during the recognition of this curved stroke need to provide more flexibility. The example in the figure gives one of these possibilities that would need to be addressed. Since this is the only curved unistroke, there is a wide range of non-conflicting state transitions that would be allowed to capture the same curved stroke.

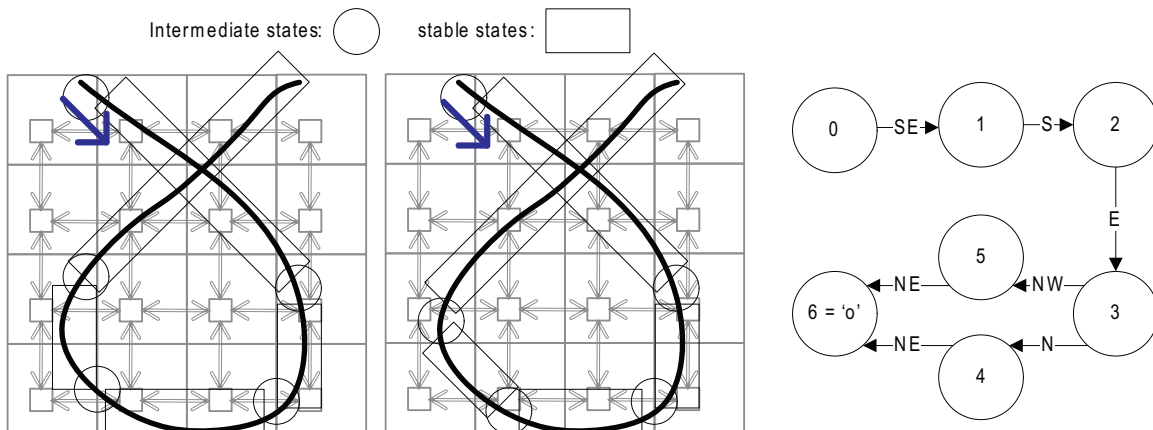


Figure 4.6: Recognition of stroke that corresponds to letter ‘x’

4.4.1 Recognising from four orientations

Computing surfaces are intended to be shared by many users. In a tabletop deployment, users can be sitting opposite each other. Under this scenario their applications also need adjusting relative to their positions. This brings the need for the recognition algorithm to be aware of the user position in order for it to return the appropriate recognised symbol. Figure 4.7 depicts such a scenario.

The fact that all remaining unistrokes are defined by rotation and reflection of five base strokes offers a simple way to determine the orientation specific symbols. The orientation of the user is assumed to be known in advance to the recognition algorithm. It can be provided by the GUI or any parent application.

At the final stage of recognition, instead of immediately outputting the recognised symbol, the final result of the state machine returns a unique identifier, which when combined with the orientation information produces the recognised symbol. The mapping between the unique identifier and the orientation can be stored in a lookup table. One column of this table would contain the ‘non-oriented’ recognition results, whilst others would map this recognition to other orientations. A fragment of this lookup table is given below.

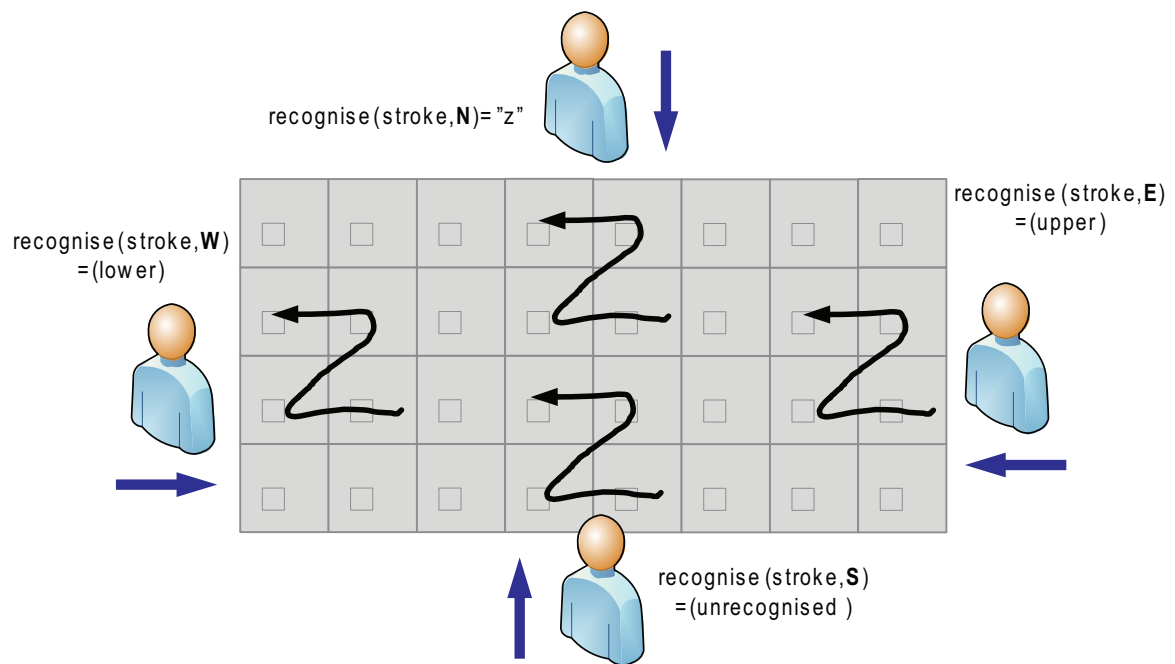


Figure 4.7: Unistroke recognition from four orientations

UID	N	E	S	W
...
023	'z'	(upper)	(unrecognised)	(lower)
...
037	'u'	'b'	'n'	'c'
...

The orientation of the user relative to the computing surface is defined as follows: Orientation S corresponds to the user that does not need rotating coordinates of the computing surface. User with orientation W needs to rotate computing surface coordinates clockwise for 90°. User with orientation N needs to rotate coordinates for 180° clockwise, etc.

4.5 Evaluation

4.5.1 Sensitivity analysis and the accuracy of the recognition algorithm

In order to evaluate the sensitivity and the accuracy of the recognition algorithm, a dedicated GUI application is developed for a Tablet PC computer. This application reads the pen inputs from a canvas with 600×600 resolution and applies the distributed unistroke gesture recognition algorithm. In order to assess the impact of the input resolution on the performance of the recognition algorithm, the application re-interprets pen input in smaller resolutions.

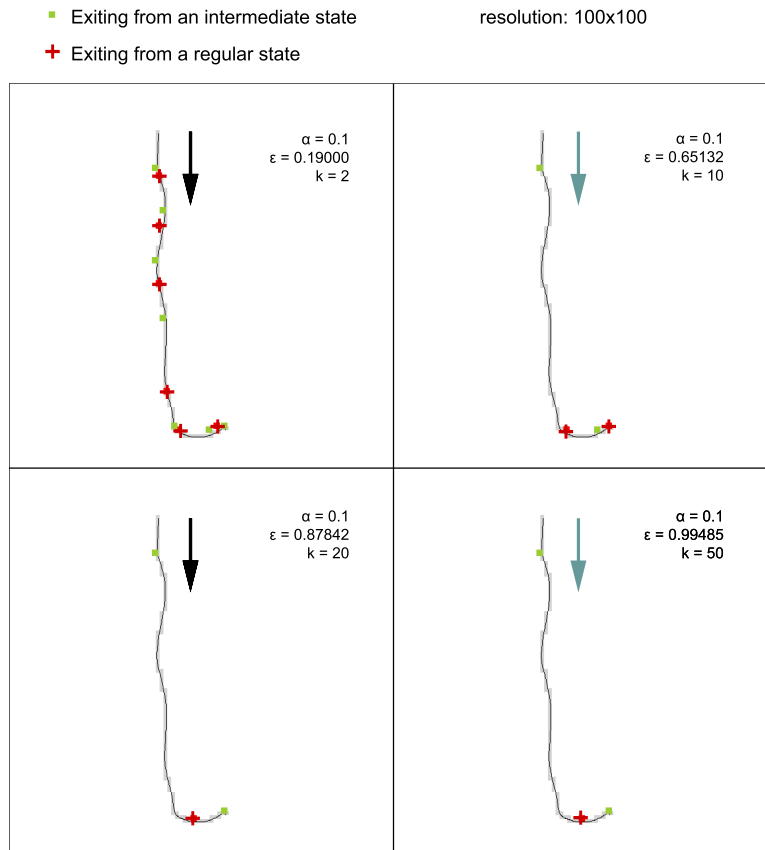


Figure 4.8: The impact of parameter k in the stroke sloppiness sensitivity

As previously suggested, the values of the smoothing constant (α) and the error threshold (ϵ) determine the amount of tolerated sloppiness. The relationship between these two parameters is given in Equation 4.3. This equation shows that the error threshold is also dependent on parameter k (the number of direction vectors that are different from the current course of direction of the stroke segment). Figure 4.8 shows the performance of the direction test for the unistroke that corresponds to letter “i” that has an extended, deformed tail. Small values of parameter k exhibit intolerance to minor sloppiness, whilst large values result in a delayed detection of the deformed tail.

For $k = 10$, the detection of the change of direction state is performed at a suitable location. This value corresponds to 10% of the input resolution across one axis. Figure 4.9 shows that the location of the state change remains invariant with the increase or decrease in resolution, provided that k remains appropriately proportional. Measurements of the length of unistroke fragments where the change of directions occur show that unistrokes change from one stable direction to another in between 2% to 7% of the axis length. These measurements, along with the previously mentioned fact that an overestimate of the coefficient k is needed, explain the fact that 10% of input resolution axis is appropriate.

The sloppiness sensitivity is also affected by the relationship between the smoothing coefficient α and the input resolution. Figure 4.10 shows that with a decrease in resolution, small values of α become overly sensitive, despite the fact that they are meant to have greater smoothing effect. This is attributed to our model of calculating the error threshold.

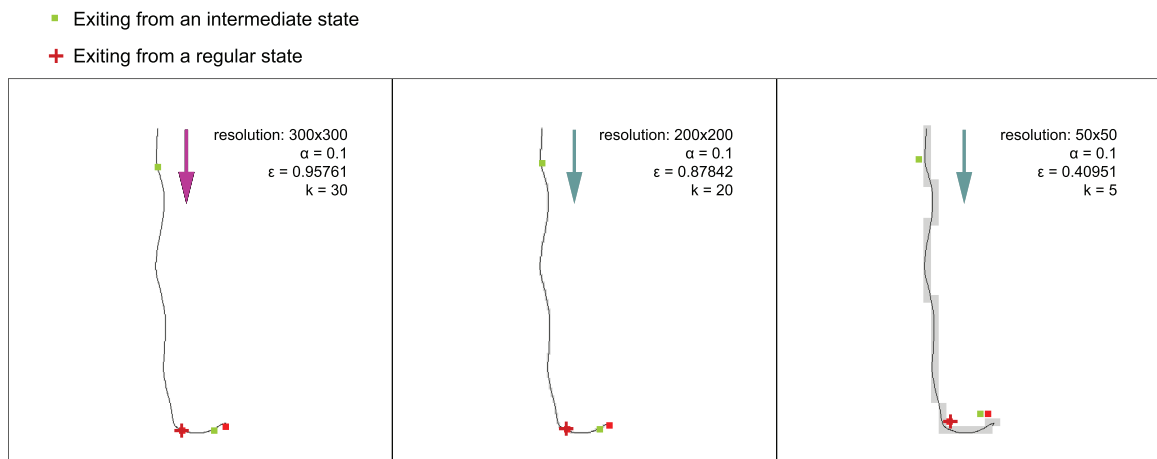


Figure 4.9: Change of state location invariance for different resolutions

It can be seen from Equation 4.2 that changes in k (and hence, changes in resolution) have a greater impact on the error threshold in comparison to changes in α .

In order to evaluate the accuracy of the recognition algorithm presented in Section 4.4, a comparison was done with a centralised implementation of the unistroke recognition. Goldberg presents an off-line algorithm for recognising unistrokes in [53]. In brief, the algorithm follows these steps:

1. Accumulate ordered list of coordinate pairs of the points that constitute the unistroke.
2. Filter list to remove noise and smooth the characterisation of the stroke.
3. Test for straight lines and, if successful, use slope to perform character lookup.
4. If not a straight line, normalise the stroke and compute additional features.
5. Find the unistroke whose features are closest to the ones computed in step 4 using a lookup table and if necessary resolve any remaining ambiguities.

In this implementation, the stroke characterisation (i.e. the collection of an ordered list of coordinate pairs) is undertaken online whilst the stroke recognition is done offline. This algorithm is proven to work very well in conventional writing surfaces using electronic pen technology.

Test samples of unistrokes were collected with the GUI application mentioned above. This application is designed to emulate the processing activity of the whole surface by executing the distributed input middleware of each tile on a separate processing thread. The application can read pen inputs with up to 600×600 resolution from the Tablet PC or it can read stored traces (consisting of spatial and temporal information of gestures). This stroke information can be split in any number of tile configurations and resolutions, which allows for the evaluation of the algorithm in different sizes of computing surface. The processing requirements of the resulting GUI testing application are very small - with most of the activity noticed during context switching between threads. The value of SDV and the corresponding state is updated in less than $5\mu s$, which suggests that the

- Exiting from an intermediate state
- + Exiting from a regular state

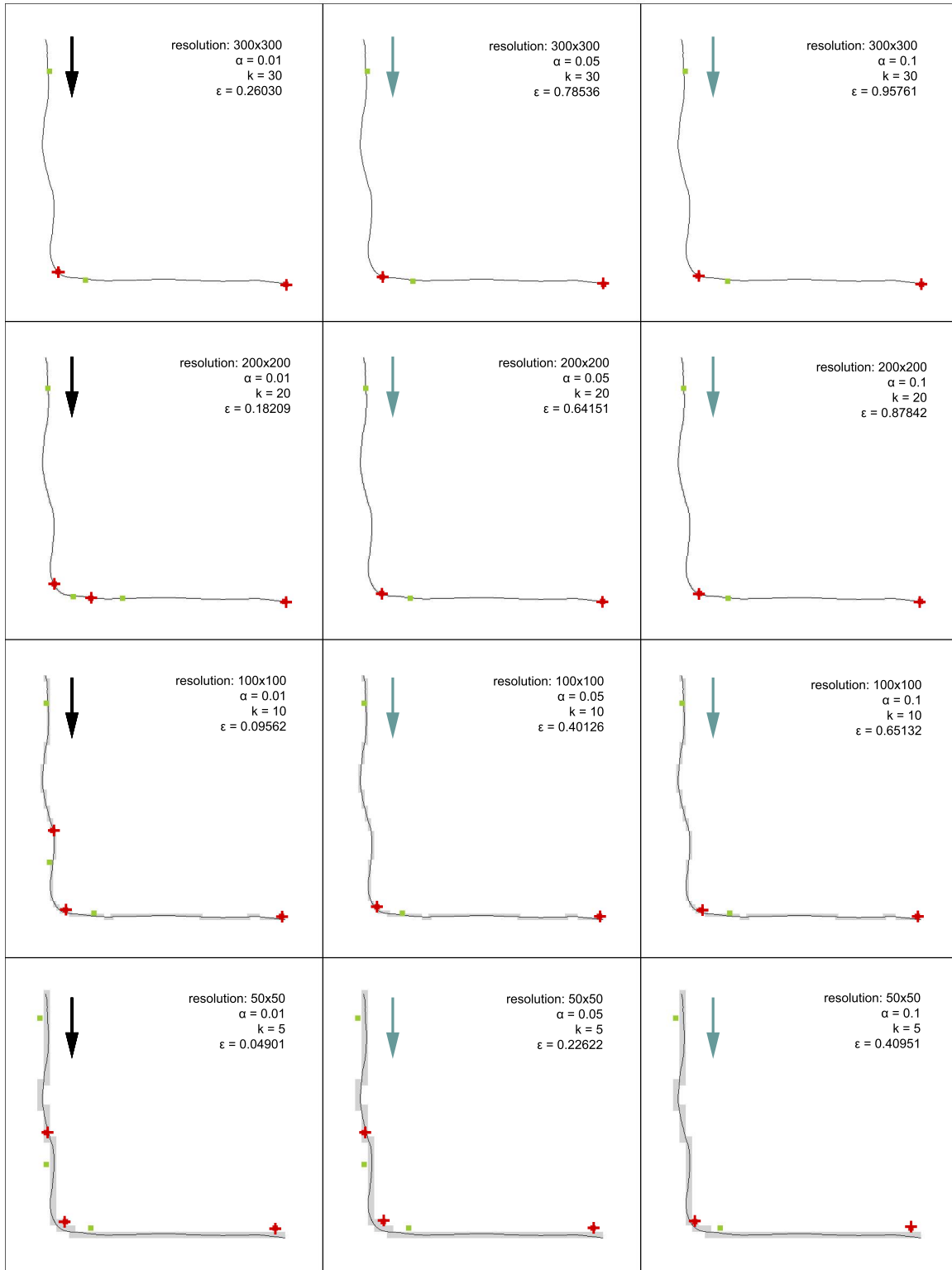


Figure 4.10: The impact of the smoothing constant (α) for various input resolutions in the recognition of the unistroke that corresponds to letter “L”

Symbol	C	D	$D \cap C$
'a'	98.00%	94.00%	93.00%
'b'	99.00%	94.00%	93.00%
'h'	98.00%	95.00%	93.00%
'o'	91.00%	89.00%	87.00%
's'	92.00%	94.00%	92.00%

Table 4.1: A comparison between recognition rates for the distributed algorithm and Goldberg’s algorithm

recognition algorithm does not pose any significant processing overheads. The distributed input middleware is also ported for the hardware prototype where the gesture input is entered with the use of mouse. The implementation of the distributed input middleware in the hardware prototype allowed not only a more comprehensive validation of the state-machine (Figure 4.3) but also for a better characterisation of the communication demands for the algorithm. For a direct comparison with the FPGA prototype, the GUI application can process pen inputs of 300×300 resolution in a 3×3 tile configuration.

For the centralised algorithm, the middlewares processing of local input (transition action 4 in Figure 4.3) collects the spatial and temporal information from the input data queue and sends it to the next tile affected by the stroke. When the end of the stroke is detected (transition action 6 in Figure 4.3), the collected information is forwarded to the control tile (0, 0), which runs the algorithm presented above. For the distributed unistroke recognition, as described in Section 4.4, only the value of the SDV vector, the recognition state and the exit position are communicated.

The recognition performance of the suggested distributed unistroke algorithm is compared relative to the recognition performance of Goldberg’s algorithm [53]. 100 samples were recorded for each of the five base unistrokes (representing letters ‘a’, ‘b’, ‘h’, ‘o’, ‘s’). These samples were processed by both recognition algorithms in the GUI application. The table below presents the recognition rates for the centralised algorithm (C) and the distributed algorithm (D). The intersection columns ($D \cap C$) show the fraction of samples recognised by both algorithms.

Table 4.1 shows that recognition rates for the distributed algorithm are similar with those of the centralised algorithm for $\alpha = 0.01$, $\epsilon = 0.26030$. In this specific configuration, the recognition of unintended strokes is tackled with two measures:

- the index n in Equation 4.1 is used to measure the length of individual segments and composing curves and as a result prevent the recognition of unistrokes with unsuitable proportions
- as noticed in Section 4.5.1, the values of parameters α and k have an immediate impact on the sensitivity of stroke recognition.

The lack of standardised recognition benchmarks for unistrokes has prevented a more thorough study of the recognition of unintended strokes and false miss rates.

4.5.2 Performance of the distributed unistroke recognition

The advantages of the distributed algorithm are revealed when the communication demands of two presented algorithms are compared. During the simulations described above, the total number of bytes transmitted on the network for each sample was also measured. For the distributed unistroke recognition algorithm, the state information (SDV, index, current state, exit position) amounts to 32 bytes. For the centralised algorithm, the spatial and temporal information for each input data point is encoded in 2 bytes. In this case, the amount of data transmitted by one tile depends on the length of the stroke fragment developed in that tile. Similar measurements were performed on the FPGA-based hardware prototype. Network traffic measurements from the hardware prototype show that the centralised algorithm transmits at least one order of magnitude more packets (i.e. bytes) than the distributed algorithm for all five base unistrokes. Figure 4.11 shows the averaged values of these measurements. Network traffic used by the mouse pointer is not included in these measurements.

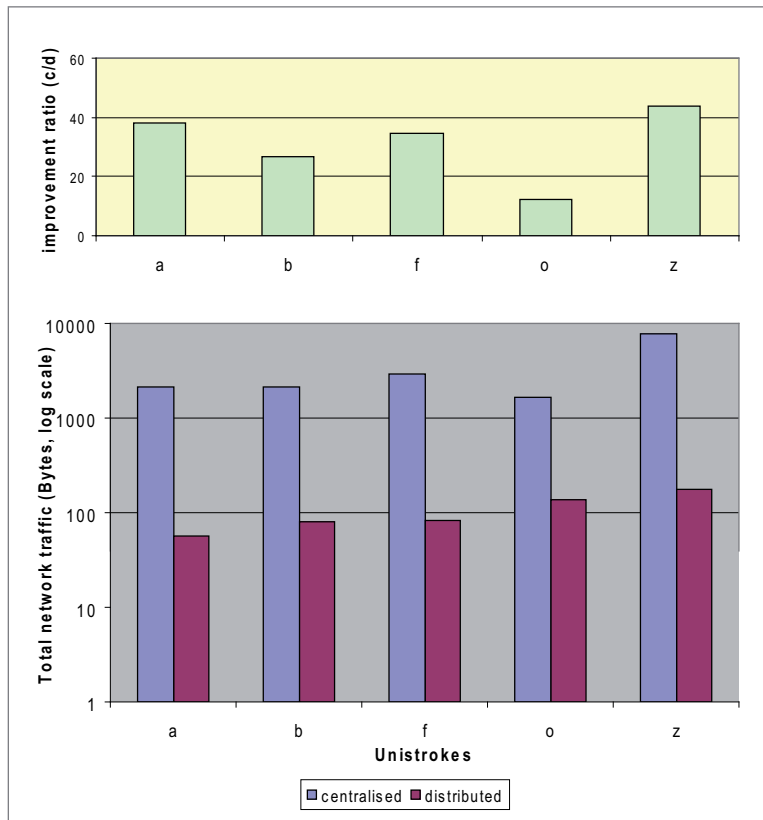


Figure 4.11: A comparison of the communication payload for the distributed and centralised recognition algorithms in the 3×3 FPGA hardware prototype

With the decrease in tile resolution (or size), the amount of state information transmitted by each tile in the centralised algorithm is also decreased. On the other hand, the state information remains constant for the distributed algorithm. This gives a natural advantage to larger tiles, although the differences are small in comparison to the changes from the centralised algorithm. Based on the communication payload observed in the hardware prototype (Figure 4.11), using the GUI application configured in more tiles, we

can estimate the communication payloads for larger networks. This is possible due to the fixed communication cost of passing the state information (SDV, index, current state and exit position) from one hardware node to another. Figure 4.12 shows this change in communication load when the 300×300 input array is split in networks of 3×3 (with tile size of 100×100), 5×5 (tile size: 60×60) and 10×10 (tile size: 30×30).

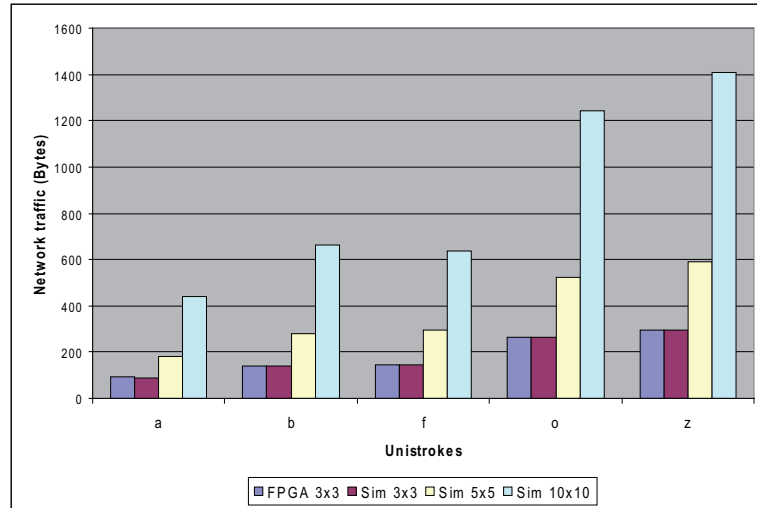


Figure 4.12: A comparison of the distributed algorithm communication payload for different network sizes (the size of the total input area: 300×300)

Finally, it must be emphasised that the suggested middleware may result in overheads for the centralised processing algorithms since the evolving stroke state follows all the tiles wherever the stroke is developed before eventually returning to the control tile. In an alternative approach, the collected data from all tiles could be forwarded immediately to the control tile, where it would finally be processed. Figure 4.13 presents a comparison between the distributed middleware traffic and the approach where the data is directly forwarded to the control tile in a simulated 5×5 network. The control is placed in tile $(0, 0)$. It is clear that this approach yields only limited improvements. More specifically, this approach will result in less overhead for strokes that finish at tiles that are distant from the control tile. That is because the complete stroke state needs to be returned for processing to tile $(0, 0)$. This is especially evident in Figure 4.13 for the traffic that is incurred by unistrokes corresponding to Roman letters ‘f’ and ‘z’.

4.6 Summary

Computing surfaces can achieve their full potential only if direct human-computer interaction is supported. Although the networked tile architecture is fundamental to its scalability, at the same time it hinders its perceived homogeneity. This tessellation needs to be abstracted away so applications can make advantage of receiving input from arbitrarily large surfaces.

This chapter presented a scalable middleware solution that conceals the tiled architecture when used in hand based inputs. While the gesture stroke develops across the

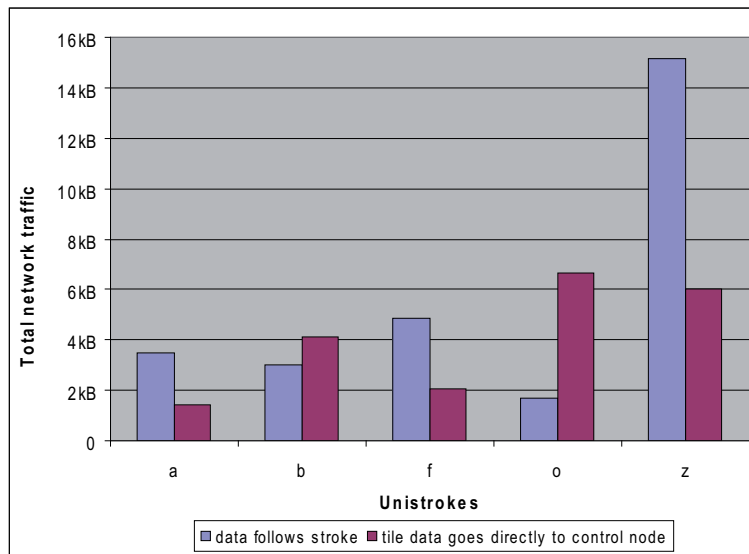


Figure 4.13: A comparison of communication payloads for two different strategies of transmitting stroke information to the processing tile (centralised recognition)

surface, affected tiles communicate with each other to consolidate complete stroke information. The process of consolidation is done while the stroke is being entered and traverses through the same tiles that are affected by the stroke. With this platform, the expertise embodied in many off-the-shelf gesture recognition algorithms can be quickly adopted for use in computing surfaces.

The presented method becomes particularly advantageous for online recognition algorithms, since the data input can be processed locally within tiles and only global stroke state information is communicated to other tiles. The amount of information that defines the global stroke state depends on the specific gesture recognition algorithms but is relatively small. A fully distributed algorithm for recognising unistrokes is presented and evaluated. The presented algorithm achieves comparable recognition rates to off-the-shelf solutions. Network traffic evaluation of this algorithm shows that its communication footprint is significantly smaller, which makes it more desirable in comparison to centralised solutions.

CHAPTER FIVE

DISTRIBUTING OUTPUT

An essential requirement for most computing systems is their ability to display processed information. The display feature has a central role in computing surfaces, therefore efficient methods for rendering and presenting information are crucial. In addition, Section 2.5.3 emphasises the importance of direct manipulation with display content in surface-oriented user interfaces. This strong emphasis on interaction necessitates efficient algorithms that enable such manipulations in computing surfaces. Fast implementations of 2D transformations are at the core of such algorithms.

Current prototypes of table-tops and interactive surfaces rely on substantial computational resources for achieving the desired interaction experience. A tiled network architecture brings new challenges. Considering the limited resources present on each tile, central storage, processing and management of large quantities of display information becomes unfeasible if not impossible. Nevertheless, the complete surface aggregates potentially enormous resources that are distributed on individual tiles. Efficient utilisation of this potential for seamless display operations requires algorithms which take into account this distribution. As many traditional display methodologies and algorithms have been designed and optimised for rather conventional computer architectures, their performance in the surface architecture may be sub-optimal.

This chapter presents a novel algorithm that efficiently utilises surface resources for applying two dimensional transformations on graphical objects. First, the issue of distributing graphical content in the surface is discussed. Once the graphical objects are presented on the surface, many interaction tasks involve manipulating them (e.g. dragging icons or zooming pictures). Considering that each manipulation is most likely to involve many tiles, efficient tile coordination and collaboration are crucial. Limited tile resources necessitate a distributed algorithm. Finally, a comprehensive evaluation of these transformations is presented. But first, we will begin with some treatment of related work on generating distributed graphical content.

5.1 Related work

The graphics engines presented in Section 2.5.1 are mainly concerned with rendering display content. Other forms of information display are represented in the form of stored image datasets, which for some applications (e.g. medical, engineering, genetics, astronomy, etc) can be very large. Applications such as JuxtaView [81] are more specifically concerned with efficient presentation of these large image datasets. JuxtaView is a cluster-based application for viewing ultra-high-resolution images in the SAGE [74] architecture. The data is distributed using a dedicated distributed memory structure called LambdaRAM [188] to all the nodes of a PC cluster driving a tiled display wall. In addition to viewing images, Juxtaview also provides some restricted 2D transformations of the displayed content. TimV [166] is another cluster based image viewer for tiled displays, built over ImageMagick, MPI and OpenGL libraries. A disadvantage of TimV is its inherent dependence on the virtual memory of a single machine for the image [167].

Some graphics engines such as SAGE and WireGL provide the basic support for building content viewers but such implementations turn out to be inefficient. In a naive implementation of a WireGL based image viewer by using texture mapping, a client program reads an image file from storage upon user request, decodes it in local memory, and then sends the decoded pixels to the frame buffer. Although this viewer would be easy to implement, as the display size gets larger the client becomes a bottleneck in terms of both computation and network bandwidth. Juxtaview has addressed the scalability issue with the cost of specialisation. It targets the needs of SAGE [74] architecture by heavily depending on the LambdaRAM memory platform – a network memory abstraction that collects the memory in a computer cluster and allocates it as a cache to minimise the effects of latency over long-distance, high-speed networks. The emphasis of this architecture in large system clusters that are interconnected with high performance networks makes LambdaRAM unsuitable for computing surfaces.

There are two key differences between the presented related work and the following treatment

- *Architecture* - a 2D mesh topology present in computing surfaces does not appear in any other display network architecture. The architectures presented in Section 2.5.1 are heterogeneous by default, usually separating resources for storage, rendering and frame buffer. This separation has a profound impact on algorithm design and implementation. Due to the homogeneity of computing surfaces, there will be no separation on the physical level.
- *Interaction emphasis* - the main application drivers for scalable display walls have been *static* representations of large datasets. While many display walls were built with the aim of collaboration, the interaction with the content presented in the wall is done mainly through a proxy device - usually being another computer. The work in this chapter is not concerned with rendering but rather direct transformation of displayed content. A platform such as Chromium could be used as a rendering engine, whose output would be shown in computing surface tiles. A thin client approach could also be well suited, where the information is created in any system and delivered to the computing surface. In order to reduce communication costs, the manipulation of this content would be locally processed in the surface.

It is clear that a networked collection of computing elements is utilised in all the related work. Although the importance of efficient communication over the network is stressed in some of the studies (e.g. [24]), no detailed report of network performance is encountered. The work presented in this chapter will emphasise the essential impact of the communication cost in the performance of the presented algorithms.

5.2 Distributing images in computing surfaces

Traditional image viewers are designed and optimised mainly for desktop workstations and their limited memory systems. As a result, they effectively handle small images and become increasingly inefficient for images reaching orders of several gigabytes. In order to view high resolution images in the computing surface prototype (described in Section 3.6), a dedicated application is developed. This application takes an image from some storage source and delivers it to the hardware prototype. It consists of three main modules

- *Source stream* - used to control the delivery of the image to the computing surface
- *Distributor* - coordinates the distribution of the image to destination tiles
- *Viewing area* - consisting of a collection of tiles used for displaying the image content

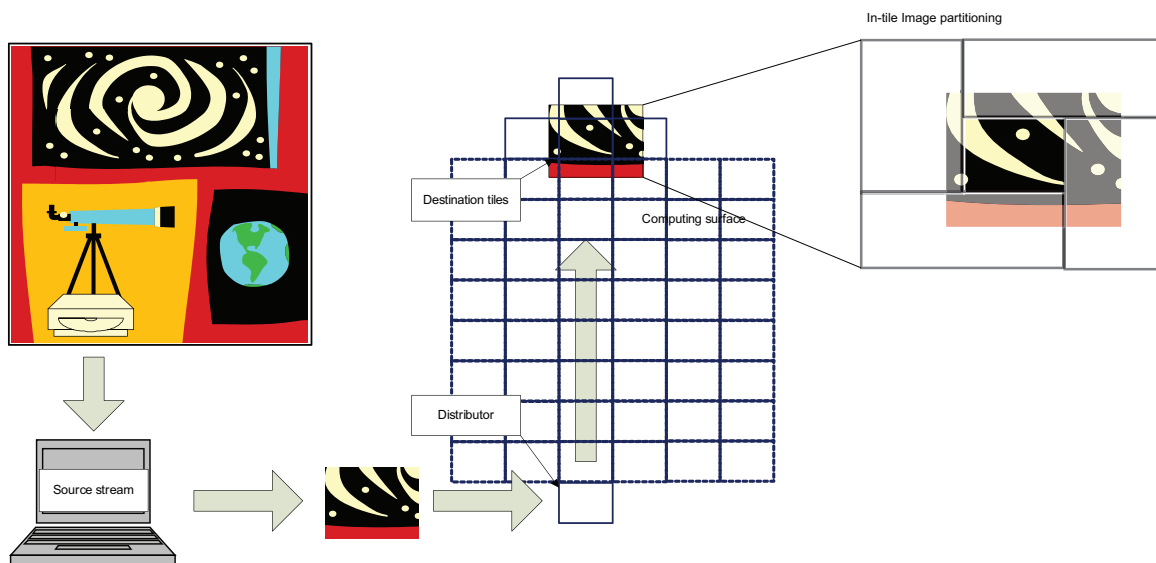


Figure 5.1: An overview of the image viewer used for computing surfaces

Figure 5.1 summarises the image data flow of under this application. The image is assumed to be stored in some storage server, connected to the computing surface through one or more source tiles. In the computing surface hardware prototype, a PC workstation operates as a storage server and is connected via JTAG to any tile, serving as a source tiles. The storage server runs the source stream module, which encodes the image and transmits

it over the JTAG. The image distributor receives the encoded image and forwards it to the appropriate tiles of the viewing area. For improved efficiency, the distributor execution thread is placed at the source tile where the header of the message is examined and appropriate forwarding is executed.

The information received from the source stream is stored in the local memory of the source tile, which implies that information coming from the source stream is limited by the memory of the source tile. Consequently, the source stream mitigates this limitation by splitting large images at the storage server into smaller *blocks* and transmitting them one by one to the computing surface. In the current implementation, images larger than 2MB are split. The source stream adds placement coordinates (relative to the viewing area) to the image content of the block. The distributor will deduce the destination tile(s) from these coordinates and forward the image content. It is very likely that the image block in display will extend beyond one destination tile and therefore needs to be partitioned before being displayed. While it would have been possible to perform tile level partitioning in the source stream, this would result in a considerable load for both the source stream and the distributor. In addition, this feature offers flexibility e.g.: it would be handy when zoom settings in the viewing area are not available to the source stream. Also, the overhead of additional information sent with the image content increases as the size of the block decreases.

Image partitioning, both in source stream and tile display, involves the following steps:

1. decoding/decompressing the image up to the level where raw pixel coordinates are revealed
2. regrouping the image content into new smaller partitions
3. encoding/compressing and forwarding the resulting partitions

The source stream adds destination coordinates to each block at the last stage of encoding.

Efficient image compression brings tremendous benefits for reducing storage needs and the communication load of transferring the image from the host to destination tiles of the surface. Considering that most image compression algorithms do not maintain 2D pixel locality in storage (i.e. data stored in any part of the file does not map to a group of nearby pixels), partitioning in the source stream becomes a very significant processing task. For compression schemes such as JPEG, some encodings require the complete image to be parsed before it can be partitioned. In this case, the source stream exhibits the very same disadvantages that traditional image viewers have. Our implementation of the source stream uses RLE [133] compression for bitmap images due to its simplicity and the preservation of row pixel locality. This results in a tradeoff for computing surfaces: the ability to load images of arbitrary size against utilisation of efficient compression schemes in order to reduce communication demands. The choice of RLE enables the stream source to load images of any size with the benefits of a modest compression.

For JPEG compression [133], the locality problem has been addressed in [24] by augmenting the JPEG format with index and block position information, which saves the MCU's (JPEG's basic compression blocks) dependencies and position in file at regular intervals, thus breaking the dependency chain. When employing such a format, while the

image is being streamed it can be routed to the relevant tile(s), thus reducing overhead on the source tile.

Once a destination tile receives a partition from the distributor, its image content is decompressed. The partition includes information about the fragment to be displayed in this specific destination tile. This information is encoded as a rectangle relative to the coordinates of the partition image. The part of the image that corresponds to the tile's display rectangle is forwarded to the frame buffer while the remains are divided in (up to) four partitions and forwarded to four corresponding neighbouring directions (see in-tile partitioning in Figure 5.1). These partitions will contain the same encoding as the partition that is received from the distributor. In this way, once partitions are received by neighbouring tiles, they can execute the very same algorithm. This symmetrical solution brings simplicity to the distributed algorithm and exhibits good load balancing which results in overall performance improvement. One may note that peripheral parts of partitions may end up being compressed/decompressed several times, which may suggest computational overhead. Once the partition is decompressed and the tile content is removed, there are two possibilities for the remainder - compress and forward to neighbouring tiles or skip the compression and transmit the content uncompressed. Transmitting the neighbouring partitions uncompressed results in higher communication demands, resulting in longer transmission times and higher energy costs than recurrent compression.

5.3 Manipulating graphical objects

Transformations of graphical objects are fundamental to computer graphics. Moving icons or windows from one location of the display to another constitutes one of the most basic operations in Graphical User Interfaces (GUI). When working with graphical objects, operations such as scaling and rotation are very common, hence necessary.

Algorithms for performing 2D transformations of graphical objects are omnipresent in most graphics packages. The implementation details of many transformations are rather similar. Among similarities, off-the-shelf solutions assume a conventional setup of storage and processing of targeted computing systems. While these implementations are well suited for execution in individual tiles, they will need to be augmented in order to take into account the network architecture. The aim of such augmentation is to efficiently utilise surface resources for distributing the process of transformation and make it suitable for execution in such architectures. To begin with, a review of mathematical foundations behind 2D transformations is given, which will reveal the insights that lead towards a fully distributed solution.

2D transformations

Each graphical object is composed of points (object vertices), which are mathematically represented with their coordinates. A coordinate system assigns n-tuples of scalars to each point in an n-dimensional space. Almost all graphical environments use Cartesian coordinates or a slight modification of the Cartesian coordinate system [178]. A 2D coordinate system is sufficient for representing 2D graphical objects. Transformations of graphical objects are accomplished by performing mathematical transformations on

vertices that compose the object. Some of the most commonly used operations such as translation, scaling, and rotation are all accomplished using linear transformations.

A general linear transformation of the point \mathbf{P} with coordinates (x, y) onto the new point \mathbf{P}^* with coordinates (x^*, y^*) is performed with the following equations

$$\begin{aligned}x^* &= ax + cy + m \\y^* &= bx + dy + n\end{aligned}$$

Coefficients a, b, c, d, m, n , are real numbers that fully characterise the transformation. A more compact representation of this transformation can be given by using vectors to represent points and matrices to represent corresponding transformations. Using vector/matrix notation, the above equation can be written as:

$$\mathbf{x}^* = \mathbf{T}\mathbf{x}$$

Clearly, two dimensional points can be represented with two dimensional point vectors $[x, y]$, but it turns out that this representation of points is too restrictive for expressing transformations involving translations. This difficulty can be overcome by introducing homogeneous coordinates [2]. The homogeneous coordinates of a non-homogeneous position vector $[x, y]$ are $[x', y', h]$, where $x = x'/h$ and $y = y'/h$ and h is any non-zero real number. In this representation, the equation $\mathbf{x}^* = \mathbf{T}\mathbf{x}$ has the form

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & m \\ b & d & n \\ p & q & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Specialised forms of the transformation matrix perform some familiar categories of transformations:

Translation - in order to translate a graphical object for any offset (dx, dy) , we can use the following matrix:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling a graphical object in a given dimension $x(y)$ for $s_x(s_y)$ can be done by using the given matrix on the right. If $s_x = s_y$ then a simple expansion ($s_x = s_y > 1$) or contraction ($s_x = s_y < 1$) occurs.

$$\mathbf{T} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotating a point around the origin of the coordinate system for a given angle α :

$$\mathbf{T} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection around an arbitrary axis, e.g. reflection around the axis $y = 0$ (the x-axis) is obtained by using:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The object can be sheared with the use of the following matrix for b and c , such that $bc \neq 0$:

$$\mathbf{T} = \begin{bmatrix} 1 & b & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Composed transformations

Although the transformations given above are fundamental, they are rather basic and limited when used individually. More elaborate operations can be achieved by performing successive applications of combinations of these basic transformations. For instance, in order to implement rotation with an arbitrary angle α from an arbitrary point $[m, n]$, we first apply a translation of the coordinate system for $[-m, -n]$ in order to bring the rotation centre point to the origin of the coordinate system, proceeded with rotation with angle α , and finally with another (inverse) translation for $[m, n]$. The successive application of various transformations is equivalent to applying the product of all matrix transformations. The transformation matrix looks like:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & -m \\ 0 & 1 & -n \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \\ 0 & 0 & 1 \end{bmatrix}$$

Some important properties of linear transformations

Linear transformations exhibit some interesting mathematical properties, out of which some will prove useful in the sections ahead. The most relevant properties of linear transformations with regard to 2D transformations of graphical objects are listed below:

1. Matrix multiplication is non-commutative [66], and therefore the composition of transformations is non-commutative.
2. Linear transformations are affine and preserve convexity.
3. Non-singular transformations are bijective and have an inverse transformation. The inverse is also bijective and a linear transformation [66]. If linear transformation \mathbf{T} is invertible, its inverse \mathbf{T}^{-1} exists and the following equality holds $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}$. For a general matrix \mathbf{T} , the inverse calculation formula is given below:

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\mathbf{T}^{-1} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \frac{1}{|\mathbf{T}|} \begin{bmatrix} | & a_{22} & a_{23} & | & | & a_{13} & a_{12} & | & | & a_{12} & a_{13} & | \\ | & a_{32} & a_{33} & | & | & a_{33} & a_{32} & | & | & a_{22} & a_{23} & | \\ | & a_{23} & a_{21} & | & | & a_{11} & a_{13} & | & | & a_{13} & a_{11} & | \\ | & a_{33} & a_{31} & | & | & a_{31} & a_{33} & | & | & a_{23} & a_{21} & | \\ | & a_{21} & a_{22} & | & | & a_{12} & a_{11} & | & | & a_{11} & a_{12} & | \\ | & a_{31} & a_{32} & | & | & a_{32} & a_{31} & | & | & a_{21} & a_{22} & | \end{bmatrix}$$

4. The area of the transformed unit square is given by the transformation matrix determinant [2].

It should be emphasised that algorithms for transforming vector graphics may be very different from the ones used in raster graphics (see [8, 52, 60, 80, 113]). While in raster graphics, all raster points undergo the transformation, in vector graphics only the vertices and corresponding equations need transforming (assuming that re-rendering follows). While in conventional computers re-rendering transformed vector objects may take even more time than transforming their raster versions, this will not necessarily hold in the surface architecture as the networking infrastructure connecting individual computing tiles is generally slower than the communication infrastructure within computing systems. Due to their high communication demands, this work investigates image/raster transformations in more detail. However, the same approach can be used for vector objects.

5.4 Distributed 2D transformations

The transformation formula presented above takes a pixel coordinate and calculates its new coordinate. During this process, there is no data-dependency between pixels, which means that this calculation can be done in parallel. This inherent parallelism and the resource distribution in the surface offer more advantages to a distributed approach for two-dimensional transformations.

5.4.1 Aims

Scalability - the resulting algorithm should not depend on tile dimensions/resolution nor have any restrictions on the size of the image being transformed.

Minimal communication overhead - due to high communication costs, overheads need to be minimal.

5.4.2 Analysis

One noticeable approach is to emulate the work as it is done in a conventional uniprocessor architecture. This could be achieved by assigning a master tile, which would perform the transformation and send off the transformed object to be drawn to the appropriate tiles. Although off-the-shelf algorithms can be adopted, this approach does not take into consideration the radical differences of the execution environment, resulting in poor resource utilisation and large communication overheads. Poor resource utilisation results

from the availability of many tiles where computation can occur, yet only the master tile is performing all the computation. Communication overheads result from the need for the source image to be transferred to the master tile for transformation, and then transferred again to the destination tile for display.

In linear transformations there are no data dependencies when transforming individual pixels, thus enabling parallel processing of all pixels. As a result, each tile can perform the transformation of its local content, resulting in much better utilisation of processing units and rather natural load balancing. Off-the-shelf algorithms can be employed to perform in tile local transformations.

It is inherent in this architecture for pixels to travel from one tile to another. This concept of ‘travelling pixels’ presents an important difference when compared to uniprocessor implementations, where pixels are only expected to move from one memory location to another. We will refer to the tile where the pixel resides before the transformation as the *origin tile*. Similarly, we will refer to the tile where the pixel ends up as the *destination tile*.

2D transformations of images involve more than just determining new pixel locations. Image pixels will be transformed into parallelograms, which won’t exactly fit into a set of physical pixels. As a result, a range of blending and mapping methods are used to reduce such discretisation effects. This problem has been addressed in a range of algorithms, with specialised optimisations for individual transformations (see [8, 52, 60, 80, 113]). This work assumes utilisation of such off-the-shelf algorithms for transformations done in individual tiles.

This brings two locations on where such off-the-shelf transformations can occur:

1. The transformation is performed at the origin tile and the result is transported to destination tiles
2. Origin tiles can calculate the destination tile for each pixel and transmit the data there. Upon arrival, the destination tile performs the transformation

In both cases, the content of one source tile may end up in many destination tiles. Correspondingly, destination tiles may receive content from many source tiles. As a result, algorithms for partitioning the tile content onto fractions that are suitable for delivery to destination tiles are necessary. Correspondingly, algorithms that merge incoming fractions at their destinations are also necessary. In addition, origin and destination tiles need to use coherent coordinates during transformations.

5.4.3 Coordinate systems

When a graphical object distributed among several tiles undergoes a transformation, tiles need to work with coherent coordinates. Consider the example of an image being rotated around some point. Coordinates used by each tile involved in the transformation should refer to the same rotation centre point. This implies that individual tiles would have to agree on a common coordinate system. There are two evident approaches:

- *a common coordinate system (for all tiles)* – assign an absolute coordinate system where the origin is some arbitrarily selected point in the surface. For valid operation, before using the coordinate system, all tiles would require the knowledge of their

position relative to this origin. This can be achieved with an initialisation task that would generate the coordinate details and flood the relevant tiles with them.

- *individual coordinate systems for each tile* – an alternative approach is to avoid using an absolute coordinate system by assigning each tile its own coordinate system. When communicating to other tiles, relative transformations are applied for maintaining coordinate coherence.

Both approaches are equivalent in their expressive power, i.e. every point in the surface can be expressed by both coordinate systems. The advantage of the common coordinate system is that there is no need for any coordinate translation between tiles. On the other hand, when each tile has its own coordinate system, no initialization is required.

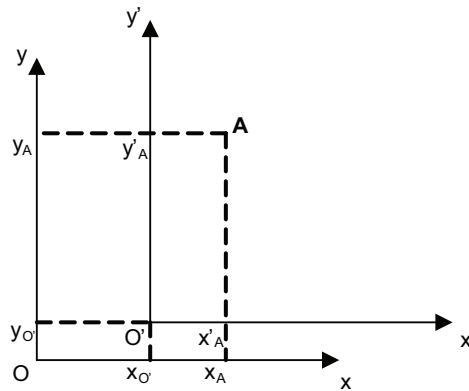


Figure 5.2: Translated coordinate systems

When each tile uses its own coordinate system, coordinate coherence can be embedded onto transformation details. We can maintain this coherence by updating the transformation matrix when it's passed from one tile to another. Consider an arbitrary point A and two coordinate systems as in Figure 5.2. The coordinates of point A in xOy are (x_A, y_A) , whilst in $x'O'y'$ are (x'_A, y'_A) . The origin of the coordinate system $x'O'y'$ relative to xOy is (x'_O, y'_O) . The relationship between these two systems can be given in matrix transformation with the following equation:

$$\mathbf{x}_A = \mathbf{T}_{O'}(\mathbf{x}_{A'})$$

where $\mathbf{T}_{O'}$ is the translation of the origin $O'(x'_O, y'_O)$. In order to apply transformation \mathbf{T} in the tile that uses coordinate system $x'O'y'$, each point has to be translated into corresponding coordinates in xOy , followed by the actual translation of interest \mathbf{T} and finally, the results are translated back into $x'O'y'$ context. As a result, transformation \mathbf{T} in tile that uses $x'O'y'$ is $\mathbf{T}' = \mathbf{T}_{O'}^{-1}\mathbf{T}\mathbf{T}_{O'}$.

In the work that follows, references to inter-tile coordinates will also be used. Inter-tile coordinates of tile $n_{x,y}$ will be denoted as (\bar{x}, \bar{y}) . For the pixel $P(x, y)$ in tile with size $W \times H$ and inter-tile coordinates (\bar{x}, \bar{y}) the following equations hold

$$\begin{aligned} x &= \bar{x} * W + x \bmod \bar{x} \\ y &= \bar{y} * H + y \bmod \bar{y} \end{aligned}$$

5.4.4 Partitioning

As mentioned earlier, 2D transformations can spread the content of one tile in several destination tiles. A partitioning algorithm is used to break up the tile content into portions that correspond to individual destination tiles. It is possible to avoid explicit partitioning if the tile content is transformed at the origin and recursively drawn at the destination tiles. This approach is less flexible as it does not take into account any network environment conditions. Alternatively, if the tile content is partitioned before it is sent over the network, the addresses of destination tiles could be used to choose the most appropriate network resources for the delivery. Note that in both cases, considering the fact that a regular (non-singular) transformation is a bijection, every pixel from the tile belongs to one and only one partition, resulting in a particular tessellation.

The partitioning process involves two steps:

- characterising partition polygons which define the tile tessellation
- creating partitions based on tessellation polygons

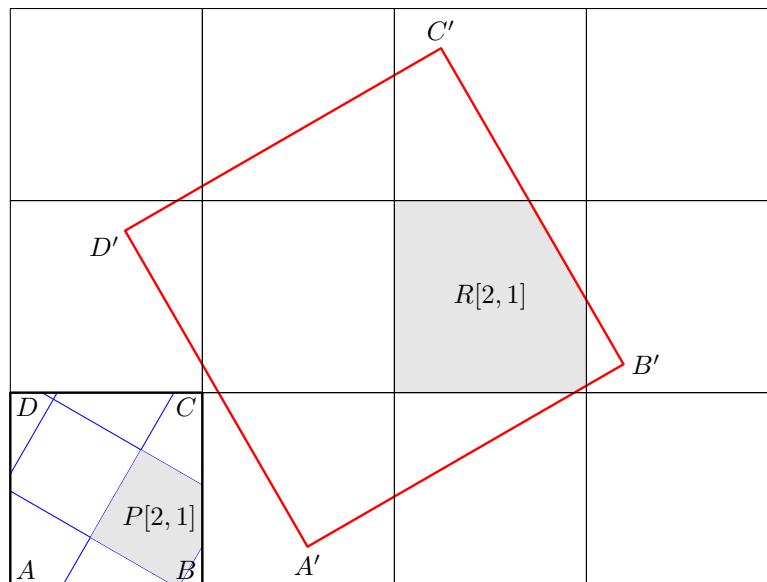


Figure 5.3: Tile partitioning

Figure 5.3 shows the rectangle $R_1R_2R_3R_4$ which defines the area of the tile undergoing transformation \mathbf{T} . We denote the rectangle that defines the border of the source tile with $B_0 = B[m_0, n_0]$. Similarly, let $B[m, n]$ denote the rectangle defining the borders of the tile with tile coordinates (\bar{m}, \bar{n}) . Rectangle $B[m, n]$ is formed by vertical tile division lines $\mathbf{v}_m : x = mW$, $\mathbf{v}_{m+1} : x = (m + 1)W$ and horizontal division lines $\mathbf{h}_n : y = nH$, $\mathbf{h}_{n+1} : y = (n + 1)H$. The resulting transformation of rectangle $R_1R_2R_3R_4$ is spread onto the area defined by quadrilateral $R'_1R'_2R'_3R'_4$. Partition polygons that define the portions of B_0 which are transformed onto individual destination tiles $B[m, n]$ are denoted with $P[m, n]$.

The calculation of partition polygons will depend on whether the transformation is applied in the local tile or destination tile.

When the transformation is performed at the destination tile, the original tile content is tessellated and the partition polygons are calculated by applying the inverse transformation \mathbf{T}^{-1} to the intersection of quadrilateral $R'_1R'_2R'_3R'_4$ with individual destination tiles (e.g. $B[m, n]$). As previously mentioned, linear transformations \mathbf{T} that satisfy $\|\mathbf{T}\| \neq 0$ are bijective functions. For bijective transformations \mathbf{T} , the corresponding inverse transformation \mathbf{T}^{-1} exists and is also bijective. Therefore,

$$P[m, n] = B_0 \cap \mathbf{T}^{-1}(B[m, n] \cap R'_1R'_2R'_3R'_4)$$

If the transformations are performed at the origin tile, the transformed content needs to be tessellated and the partition polygons $P'[m, n]$ are defined with the following formula:

$$P'[m, n] = \mathbf{T}(B_0) \cap B[m, n]$$

Let \mathfrak{B} be the set of all border rectangles $B[m, n]$ in the surface. We will refer to tile tessellation with \mathfrak{T} , defined as the set of all the partitioning polygons. In summary, tile tessellation can be expressed with the following formulae:

$$\text{Transformation applied in the source tile: } \quad \mathfrak{T} = B_0 \cap \mathbf{T}^{-1}(\mathfrak{B})$$

$$\text{Transformation applied in the destination tile: } \quad \mathfrak{T} = \mathbf{T}(B_0) \cap \mathfrak{B}$$

The rest of this Section assumes that the transformation is applied in the destination tile. An overview of the partitioning algorithm is given in Algorithm 1.

Algorithm 1: Tile partitioning

input : Bitmap to be partitioned B
input : A set of partitioning polygons P
output: A set of tile partitions T_P
foreach *polygon* P **do**
 calculate $P \cap B$ and store in T_P ;
end

Polygon clipping algorithms can be used for calculating the required intersections. Such algorithms are abundant and well studied [57, 156]. Nevertheless, the tessellation nature of our clipping polygons creates an opportunity for a more efficient clipping method and corresponding intersection calculations.

First, the bijection property of \mathbf{T} guarantees that almost every pixel belongs to one and only one partition. An exception to this rule are the pixels that contact the borders of the tessellation polygons in the source tile. These bordering pixels may contribute to multiple bordering partitions and need to be dealt appropriately. Since the overall goal is to calculate all partitions, an efficient algorithm would iterate through every tile pixel only once and determine (and/or store to) its partition. Furthermore, considering partition's spatial locality, the efficiency of an algorithm would be improved further if the determination of partitions were done for more than one pixel at a time.

The borders of tessellation polygons in the source tile are formed from its border lines and inverse transformations (see Section 5.3) \mathbf{T}^{-1} of tile division lines \mathbf{v}_i and \mathbf{h}_j . For any

given raster line defined by $y = y_R$, we can calculate its intersections with $\mathbf{T}^{-1}(\mathbf{v}_i)$ and $\mathbf{T}^{-1}(\mathbf{h}_j)$, thus effectively determining the bordering pixels.

The inverse transformation of \mathbf{v}_i is a new line described by the following parametric equations:

$$\begin{aligned} x &= x(s) = im_{11}W + m_{12}s + m_{13} \\ y &= y(s) = im_{21}W + m_{22}s + m_{23} \end{aligned}$$

We denote (x_i, y_R) to be coordinates of the intersection point of $\mathbf{T}^{-1}(\mathbf{v}_i)$ with the line defined by the current raster line $y = y_R$. The value of x_i is calculated with the following equation, which derives from the parametric equations given above:

$$x_i = \frac{iW}{a} - \frac{b}{a}(y_R - m_{23}) + m_{13} \quad (5.1)$$

There are two points to note from this equation:

1. The equation above defines an arithmetic sequence of numbers \mathbf{x}_n^V where $\mathbf{x}_{n+1}^V = \mathbf{x}_n^V + \frac{W}{a}$
2. From the equation given above, we can derive the range of numbers $i_0, i_0 + 1, \dots, i_0 + l_{y_R}$ such that all bordering pixels between $P(0, y_R)$ and $P(W, y_R)$ of the raster line $y = y_R$ are covered, or mathematically:

$$\mathbf{x}_{i_0}^V \leq 0 < \mathbf{x}_{i_0+1}^V \quad \wedge \quad \mathbf{x}_{i_0+l_{y_R}-1}^V < W \leq \mathbf{x}_{i_0+l_{y_R}}^V \quad (5.2)$$

Consequently, in order to find out the intersections of inverse transformations of \mathbf{v}_i with the raster line $y = y_R$, we only need to calculate i_0 and increment $d = W/a$.

Similarly, the inverse transformation of \mathbf{h}_j is a new line described by parametric equations:

$$\begin{aligned} x &= x(s) = m_{11}s + jm_{12}H + m_{13} \\ y &= y(s) = m_{21}s + jm_{22}H + m_{23} \end{aligned}$$

The intersection point of $\mathbf{T}^{-1}(\mathbf{h}_j)$ with the raster line $y = y_R$ is:

$$x_j = \frac{jH}{c} - \frac{d}{c}(y_R - m_{23}) + m_{13} \quad (5.3)$$

A similar sequence of numbers \mathbf{x}_n^H is formed such that $\mathbf{x}_{n+1}^H = \mathbf{x}_n^H + \mathbf{d}$, where $d = \frac{H}{c}$ and a subset of these numbers captures the raster line $(0, y_R) \dots (W - 1, y_R)$.

Lemma 1 *For every pixel $P(x, y_R)$ belonging to the raster line $(0, y_R) \dots (W - 1, y_R)$, there is a pair (i, j) such that $\min(\mathbf{x}_j^H, \mathbf{x}_{j+1}^H) \leq x < \max(\mathbf{x}_j^H, \mathbf{x}_{j+1}^H)$ and $\min(\mathbf{x}_i^V, \mathbf{x}_{i+1}^V) \leq x < \max(\mathbf{x}_i^V, \mathbf{x}_{i+1}^V)$. Pixel $P(x, y)$ belongs to partition $P(i, j)$.*

The proof of this lemma is rather straightforward. The first claim is true due to the fact that sequences \mathbf{x}_n^H and \mathbf{x}_n^V are unbounded. The second claim results from the fact that the transformation of pixel $P(x, y)$ has the properties: $iW \leq \mathbf{T}(x) < (i + 1)W$ and

$jH \leq \mathbf{T}(y) < (j+1)H$, which are the bounds of destination tile $B[i, j]$, hence $P(x, y)$ belongs to partition $P[i, j]$.

This method of partitioning is summarised in Algorithm 2. In this algorithm, the function ‘ I_0 ’ calculates the value of i_0 as constrained by the inequality 5.2. Similarly, the function ‘ J_0 ’ calculates the corresponding j_0 value. The ‘ x_i ’ function effectively calculates the corresponding value of x_i based on Equation 5.1. Similarly, function ‘ x_j ’ is based on Equation 5.3. The ‘ sgn ’ function represents the traditional sign function [179]. Arguments $\mathbf{T.a}$ and $\mathbf{T.c}$ represent matrix elements a and c of the matrix \mathbf{T} .

Algorithm 2: Single-scan tile partitioning

```

input : A bitmap/raster  $B$  to be partitioned
input : Inverse transformation: of  $\mathbf{T}$ 
output: Partitions of  $B$  represented as a set of partition data structures
          $P[i, j]$ 

for  $y \leftarrow 0$  to  $H - 1$  do
   $x \leftarrow 0$ ;
   $i \leftarrow I_0(y)$ ;
   $j \leftarrow J_0(y)$ ;
  while  $x < (W - 1)$  do
    Add pixel range  $B[x, y] \dots B[\min(x_i(i, y), x_j(j, y), W - 1)]$  to  $P[i, j]$ ;
     $x \leftarrow \min(x_i(i, y), x_j(j, y), W - 1)$ ;
    if  $x = x_i(i, y)$  then
       $i \leftarrow i + sgn(\mathbf{T.c})$ ;
    else
      if  $x = x_j(j, y)$  then
         $j \leftarrow j + sgn(\mathbf{T.a})$ ;
      end
    end
  end
end
  
```

Data structures for non-rectangular image content

Many graphics formats allow only rectangular image formats (e.g. bitmaps) and consequently will not be suitable for handling arbitrary non-rectangular images. In a simplistic approach, partitions can be augmented with a bounding box which contains ‘no-data’ pixel information for pixels outside partitions. The disadvantage of this approach is that it results in some overhead of storing information about the location of ‘no-data’ pixels.

It is possible to store the 2D information of the partition in a linear array if reconstruction information about the 2D partitions is provided. Since the destination tile can deduce the polygon information of the partition (knowing the source tile of the partition), the partition data structure consists only of a linear array of pixel information. This approach is adopted in Algorithm 2, where $P[i, j]$ are dynamically sized linear arrays of pixel data. The process of adding a pixel range involves simply copying the pixel data to the end of the array.

The process of decoding this representation of tile partitions is given in Algorithm 3, which is executed in destination tiles. Based on the line-at-a-time encoding approach adopted in Algorithm 2, this algorithm can decode the necessary coordinate information of the received partition data. Partition pixel data is assumed to be stored in the ‘pixels’ array. Functions ‘ymin’ and ‘ymax’ calculate the minimal (maximal) y coordinate values of the partition polygon. Functions ‘xmin’ and ‘xmax’ return the minimal (maximal) x coordinate value for the intersection points between the partition polygon and the horizontal line ‘H(y)’ that is determined by the value of the coordinate ‘y’.

Algorithm 3: Decoding partitions

```

input : partition array: pixels[]
input : Source tile coordinates  $(m_0, n_0)$ 
output: RestoredPartition[, ] (a 2D array)
Destination tile is  $B[n, m]$  and source tile is  $B[n_0, m_0]$ ;
PartitionPolygon =  $\mathbf{T}^{-1}(B[n, m]) \cap B[n_0, m_0]$ ;
 $y_{min} \leftarrow \text{ymin}(\text{PartitionPolygon});$ 
 $y_{max} \leftarrow \text{ymax}(\text{PartitionPolygon});$ 
 $i \leftarrow 0;$ 
for  $y \leftarrow y_{min}$  to  $y_{max}$  do
   $x_{min} \leftarrow \text{xmin}(\text{PartitionPolygon} \cap \text{H}(y));$ 
   $x_{max} \leftarrow \text{xmax}(\text{PartitionPolygon} \cap \text{H}(y));$ 
  for  $x \leftarrow x_{min}$  to  $x_{max}$  do
    RestoredPartition[x, y]  $\leftarrow$  pixels[i++];
  end
end

```

Receive and merge

The details of the transformation can be used by the destination tile to produce a list of origin tiles that will be delivering partitions to it. The list of origin tiles can be calculated in a similar way to the list of destination tiles. However, it was noticed in the partitioning algorithm that the list of destination tiles was never explicitly created. Since the temporal distribution of incoming origin tile packets will not preserve the spatial locality (individual portions are likely to be interleaved), it is useful to create the list of incoming partitions so that the destination tile has the means to assess the delivery progress. The list of source tiles is created by determining individual tiles which have a non-empty intersection with the inverse transformation of the current, destination tile.

Algorithm 4 is used to produce this list of inter-tile coordinates. This algorithm begins by calculating the inter-tile coordinates of the ‘bounding box’ set of tiles that cover the inverse transformation of the destination tile. Since this set of tiles may contain elements that are not source tiles, an intersection test is used for confirmation. The algorithm uses a utility function ‘FillShape’ that returns the filled shape of the input polygon. A tile with inter-tile coordinates (\bar{x}, \bar{y}) is a source tile if it’s filled shape intersects with the filled shape of the inverse transformation of the destination tile.

Algorithm 4: Calculating the list of origin tiles for a given destination tile

```

input : transformation matrix  $\mathbf{T}$ 
output: sourceList - a list consisting of inter-tile coordinates of all source
        tiles, i.e.  $\mathbf{T}^{-1}(B[n, m]) \cap \mathfrak{B}$ 
Destination tile is  $B[n, m]$ ;

inverseDestination =  $\mathbf{T}^{-1}(B[n, m])$ ;
 $x_{min} \leftarrow \text{xmin}(\text{inverseDestination})$ ;
 $x_{max} \leftarrow \text{xmax}(\text{inverseDestination})$ ;
 $y_{min} \leftarrow \text{ymin}(\text{inverseDestination})$ ;
 $y_{max} \leftarrow \text{ymax}(\text{inverseDestination})$ ;
for  $\bar{x} \leftarrow x_{min}/W$  to  $x_{max}/W$  do
    for  $\bar{y} \leftarrow y_{min}/H$  to  $y_{max}/H$  do
        if  $\text{FillShape}(B[\bar{x}, \bar{y}]) \cap \text{FillShape}(\text{inverseDestination}) \neq \emptyset$  then
            sourceList.Add( $(\bar{x}, \bar{y})$ );
        end
    end
end

```

5.4.5 Algorithm summary

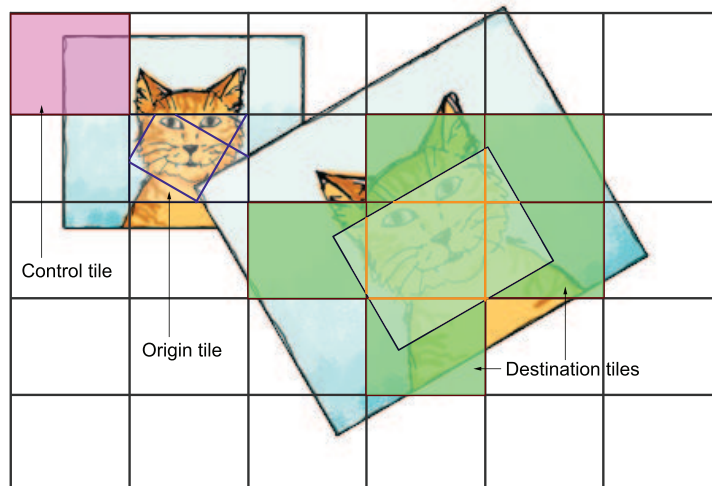


Figure 5.4: The positioning of control tile, an origin tile and its corresponding destination tiles in a 2D transformation

A distributed 2D transformation undergoes the following steps:

Transform trigger

Transformations are triggered by user input, which depending on the implementation of the user interface, can be invoked in many ways. In a simple example, a click on a UI button is assigned to a specific transformation. Transformations can also be triggered by a vast range of interaction possibilities, e.g., the user may use their hands to manipulate

the image, in which case, another layer of software will have to determine the details of the transformation matrix. Such algorithms are not treated here, but it is assumed that the transformation details are delivered to some dedicated tile, which will be referred to as the *control tile*.

Propagation

Once the transformation is triggered, each tile hosting any part of the graphical object needs to be notified. For instance, in the first example given above, since the button may be located in any tile, the information about the undertaken transformation needs to be propagated to every origin and destination tile. A control tile which is dedicated to oversee the allocation and release of tile resources can be used to ensure correct co-ordination of the execution of the transformation. Each image will contain its control tile. During transformations, the control receives transformation requests and services them by propagating relevant information to all the host tiles. Whilst not essential for the functional correctness of the distributed transformation algorithm, the control tile is instrumental in coherent use of surface resources for executing transformations. In order to reduce the trigger propagation latency, the control tile should be located in proximity to all the tiles that host the corresponding image. Its location is static in the current implementation, but for larger computing surfaces where images could get potentially distant from the control tile, it would be desirable to dynamically set the location of the control tile so it remains close to its image. In addition to propagation, the control tile is also used to serialise transformations, discard invalid transformations or compose two incoming transformations into one. In addition, the control tile can be used to provide some level of protection and security by, e.g., preventing malicious tiles sending conflicting transformation requests. These potential features are not part of this research and therefore are not addressed here.

Partition and communicate

Once the request is received by host tiles, the tile content is partitioned and the relevant information is communicated to destination tiles over the network interface.

Receive and merge

The partitions sent by origin tiles are collected by destination tiles. The number of incoming partitions is calculated and once all partitions have been received, the content can be transformed and displayed.

A summary of the algorithm with functions associated to each tile role is given in Figure 5.5.

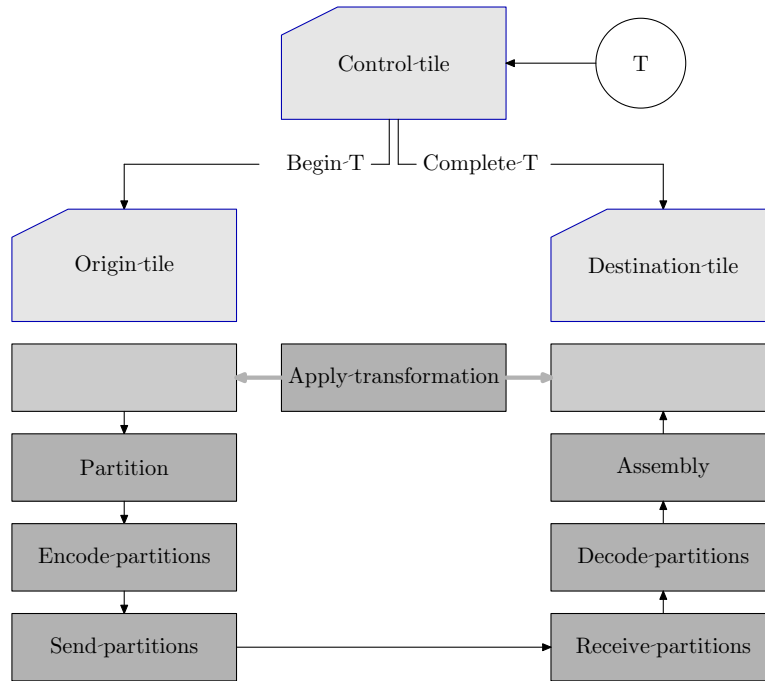


Figure 5.5: Task summary in different role tiles during distributed transformation

5.5 Performance evaluation

5.5.1 The hardware prototype and the corresponding simulations

Distributed 2D transformations have been implemented for the FPGA-based hardware prototype. A mouse device is used for generating transformation triggers for translation, rotation, shear and scale. The user is able to switch between different transformations. In addition, the application allows the user to replicate images and transform them individually. Multiple users can manipulate images simultaneously. Due to the lack of a dedicated graphics engine, the NIOS II CPU performs the graphical rendering in addition to all other processing tasks in the system. This results in a significant performance penalty for the system.

In brief, the information about mouse pointer movements is received by a dedicated node and translated into transformation requests. These transformation triggers are forwarded to all the nodes that host the image that is being transformed. The source nodes execute algorithms 1 and 2 and store the resulting output to a memory location dedicated to the network interface. The network device drivers forward this information to the network router from where the data is transferred to the destination nodes. Destination nodes execute algorithm 3, the corresponding transformation, and finally render the resulting image.

Since the network hop latencies are small (56 clock cycles, or less than $1\mu s$) when compared to the overall processing times, the transformation trigger is considered to be delivered to all the nodes at approximately the same time. Therefore, it is fair to assume that the partition stage of the transformation starts simultaneously in all the nodes. When looking at the execution of a single transformation, the partition data begins the transfer

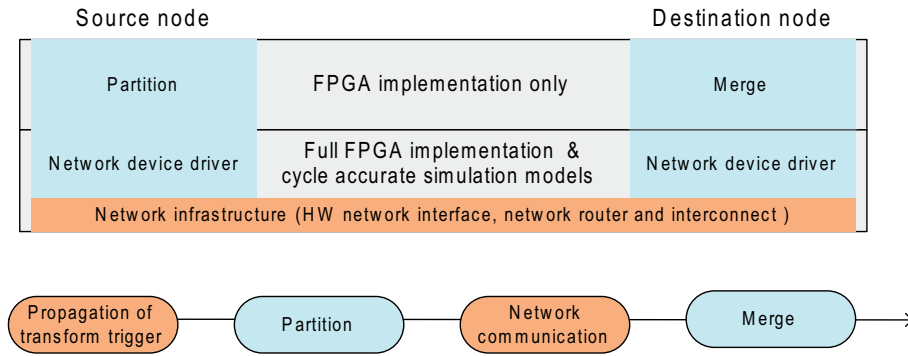


Figure 5.6: The temporal and structural representation of a single transformation for source-destination pair

over the network only after partitions are fully prepared and processed by the CPU. In the particular transformations that are evaluated in this section, the processing times for partition and merge are identical in all the tiles where the execution is done. This ensures that all the tiles spend an equal amount of time in processing the transformation (for both, partition and merge stages) and that the network stage is also started at the same time.

This temporal separation between processing at the source node, network communication, followed by processing at each destination node allows us to reason about the performance of the algorithm in larger computing surfaces. Since the RTL model of the network subsystem used for FPGA synthesis is identical to the one used in the simulation model, we are able to perform accurate network simulations of larger computing surface networks. As the activity of the network device driver is tightly coupled with the router activity, its execution is also included in the RTL model (see Figure 5.6). The processing stages of the algorithm (partition and merge) have not been simulated because cycle accurate simulations of the system would have been prohibitively expensive in simulation time. More importantly, since the processing times are effectively constant (as mentioned above), the FPGA implementation is used to obtain a one-time direct measurement of this figure for each transformation. Therefore, simulations are only used to measure the network activity. The corresponding processing time of each transformation, measured from the FPGA system, is then simply added to the timing results from network simulations.

The performance of the FPGA-based prototype performing some individual transformations over an image of size (400, 200) (see Figure 5.7) is presented in table 5.1. There is only one column for partition and one column for merge since only FPGA measurements are taken for this part of the transformations.

The trigger propagation time is directly dependent on the number of nodes that host the image and the maximum hop distance between the hosting nodes and the control node. In these measurements, it is constant since it sends the data to the same destination nodes (the control node is the lower left node in Figure 5.7). Partition and merge algorithms are the most time consuming processes due to the memory transfers that they invoke. On the other hand, the transmission of data over the network is not significant due to the fact that there is only a relatively small number transferred pixels (less than 300 pixels

Transform.	Trigger propag. (FPGA)	Trigger propag. (sim)	Network (FPGA)	Network (sim)	Partition time	Merge time
T(0,-1)	80.00 μ s	79.92 μ s	1.211ms	1.212ms	6.43ms	6.11ms
T(-1,0)	80.00 μ s	79.92 μ s	0.613ms	0.612ms	6.46ms	6.57ms
T(-1,-1)	80.00 μ s	79.92 μ s	1.945ms	1.944ms	6.42ms	6.41ms
Shear(H)	80.00 μ s	79.92 μ s	0.808ms	0.809ms	11.11ms	12.18ms
Rotate(1°)	80.00 μ s	79.92 μ s	1.321ms	1.321ms	31.11ms	32.74ms

Table 5.1: Measurements of execution time for a range of transformations on the FPGA-based hardware prototype and on the network simulation environment

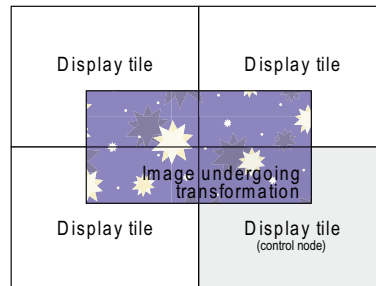


Figure 5.7: The initial position of the image for the reported measurements (Table 5.1)

for the given transformations). The network simulation results (for the network column) are within 1 μ s compared to the measurements presented in the table above (1 μ s is the resolution of the system timer user for performance measurement).

The measurements presented above were performed with no other network traffic present in the system. This is not always the case since there may be other messages transmitted over the network (e.g. mouse pointer updates, other transformations, etc) and as a result, the performance will be affected. Depending on other node activity, degradation of up to 40% has been observed.

In order to reliably support a series of unremitting transformations (i.e. dragging an image over the display area), special care has to be taken to preserve the order of transformation execution. Some nodes can complete a currently executing transformation faster than others, in which case the immediate execution of the next transformation may result in incoherent state of the transformed image. This issue is addressed by adding a flow control mechanism for transformation requests.

The overall performance of the FPGA hardware prototype is very low when compared to existing computer systems. Moreover, such modern systems are able to process much more data compared to the amount they are able to transmit or receive through the network interface. This suggests that when distributed transformations are ported to higher performance computer systems, the speed of partition processing has much scope for improvement compared to the speed of network transmission. The next section will discuss the performance of the network.

5.5.2 Network evaluation

The significant cost of transporting information over the network infrastructure necessitates efficient communication for achieving good performance.

This section will investigate the performance of a selected group of transformations in the simulation environment. Whenever possible (i.e. tests in small networks), these results are compared to the performance of transformations executed in the FPGA-based hardware prototype. These simulations are performed without any other traffic present in the network. There are two motivations for doing this:

- The ease of implementing non-interfering networks [33] with virtual channels offers future implementations the possibility to transfer this data in isolation from any other traffic.
- Isolated simulations give a good examination of particular effects of routing and flow control on individual traffic patterns

The design of the router architecture has a significant impact on the performance of the network for the traffic patterns exhibited by 2D transformations. More details of the Islay router architecture introduced in Section 3.6.1 will be helpful for analysing the performance results.

Router architecture

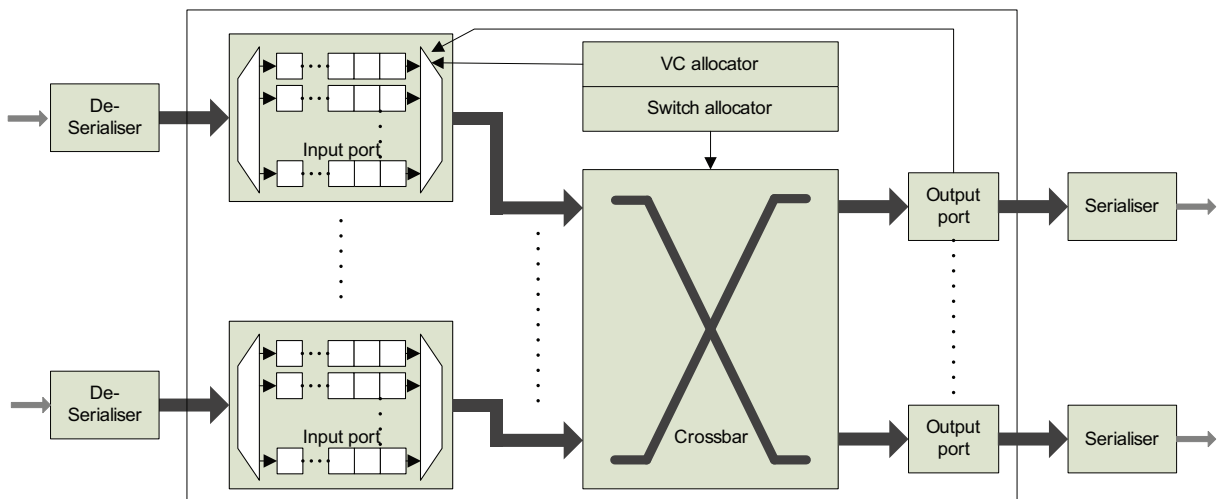


Figure 5.8: Base router model

Figure 5.8 gives a diagram of a generic virtual channel router. Each packet arrives on a physical input port and is delivered to the appropriate virtual channel buffer (input buffer associated with a virtual channel) by examining the virtual channel identifier (VCID) included with each flit's control information. It then progresses through various stages in the router before it is delivered to a neighboring router. First, the output port of the flit is determined by the routing function. Next, the packet is in the VC-allocation state where

the allocator examines all input packets and their destinations and assigns free output virtual channels to the packets, if available. Thirdly, the packet competes for a switch port in the switch allocation stage. Finally, on switch grant, the packet is delivered across the crossbar and the physical channel to the neighboring tile.

This traditional implementation suggests independent arbiters for providing access to virtual and physical channels. The dependency between these two operations often means that they are performed in consecutive pipeline stages. Alternatively, speculation may be used to enable both allocators to operate concurrently [116]. The Islay architecture takes a simpler approach where the VC arbitration is done implicitly. A virtual-channel is allocated when a waiting packet is granted access to an output port for the first time. To ensure performance does not suffer, switch requests from packets not yet assigned a VC are only made if the VC free pool at the required output port is not empty. The router is now constructed only with a switch allocator. A similar approach has been taken in the design of a single-cycle router reported in [83].

The Islay architecture uses matrix arbiters for contention resolution. A matrix arbiter implements a *least recently served* priority scheme. This arbiter is very efficient for small number of inputs because it is fast, inexpensive to implement and provides strong fairness where requesters will be served equally often [34]. In this implementation all requesters have equal weights. Although this arbitration policy is locally fair, as we shall see, the overall system arbitration can be globally unfair. This has significant performance consequences for traffic patterns exhibited by 2D transformations.

An input-first separable switch allocator [34] is used in this router. The allocation is performed with two phases of arbitration: one across the inputs and one across the outputs. Since there is at most one grant asserted for each output, the result is a legal matching. However, this matching is not always maximal. The implications of this allocation scheme will be discussed further below.

The Islay architecture supports a *dimension-ordered* (DOR) routing function. DOR routing is an oblivious, deterministic routing, where packets are forwarded along one dimension first and then along the other dimension. In 2D mesh networks, there are two possibilities of such routing functions: XY (YX) routing, where priority is given to X (Y) dimension. The implementation of this category of routing functions is relatively straightforward, which makes them very attractive. However, this advantage comes with the cost of no path diversity, which can result in large load imbalances in the network.

In order to provide comparative performance results for the impact of routing functions in the work presented here, *O1TURN* [137] routing was also implemented. This is another oblivious routing function, where packets are allowed one of two possible dimension-ordered routes that differ only on the order of dimension traversal. The source node randomly selects between XY and YX for each sent packet, which is preserved along the route of the packet. O1TURN offers provable, near-optimal bounds on its worst-case throughput. On the latency front, O1TURN guarantees the minimum number of network hops. The complexity of a router implementation for O1TURN is comparable to that of dimension-order routing. Although each packet takes at most one turn to reach the destination, the choice of turns is determined only when the first traversal dimension is selected, thus opening the possibility of selecting turn combinations that may yield deadlock. Deadlock is avoided by using separate virtual channels for separate first dimension properties - half of the virtual channels are used for XY routing and

the other half for YX routing. Once a packet is injected into the network, the packet is constrained to remain within the virtual channels belonging to the routing mode it was injected into. Since the VC allocation mechanism needs to enforce this constraint, Islay's implicit allocation needs to be modified to support O1TURN routing. In order to maintain implicit VC allocation in Islay architecture but also support O1TURN routing, two VC free pools are used instead of one - one for each group of virtual channels.

Other routing functions are not considered for two reasons:

- Adaptive routing schemes result in significant complexities in router design.
- Since each flit transmission at every hop incurs energy consumption, minimal routing algorithms result in lower energy consumption

Where minimal, deterministic routing is concerned, the performance of ROMM [111] functions are not considered due to their comparable performance with O1TURN.

Translation locality

Consider the translation T of a rectangular image that is distributed among tiles cornered by $(0, 0)$ and (m, n) . Each tile contains $W \times H$ pixels. Undergoing T , each pixel $P(x, y)$ is moved to $P'(x + dx, y + dy)$. For simplicity, let's assume that $dx = \bar{dx} \times W$ and $dy = \bar{dy} \times H$. Under this transformation, each pixel travels \bar{dx} tiles horizontally and \bar{dy} tiles vertically; we will refer to this translation as $T(\bar{dx}, \bar{dy})$. This means that each pixel in the image has an equal amount of network (and processing) resources available for travelling to its new destination. Moreover, in its first appearance, this gives us the impression that the time to complete the network transmission would not depend on the image size. We will refer to this desired property as *translation locality*.

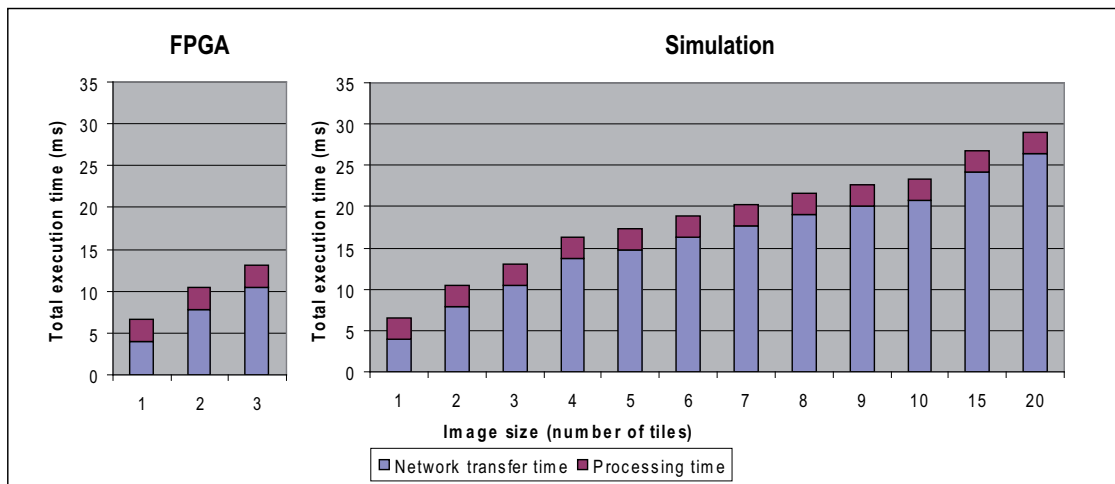


Figure 5.9: FPGA measurements and simulation of execution time taken for completion of translation $T(5, 0)$ in images of different sizes, under same network density (tile size)

Simulated translations of images with different sizes were conducted in the computing surfaces simulation environment (see Section 3.6.2). Depending on the size, these images get spread among several contiguous tiles. The tile display size is $W \times H$, where $W =$

50Pix and $H = 50\text{Pix}$. The sizes of the simulated images are $(n \times W) \times (1 \times H)$ where n ranges from 1 to 20. In the rest of this section, we will adopt the notation $I(n, 1)$ to represent an image with size $(n \times W) \times (1 \times H)$. This notation makes it clear that an image of size $I(n, 1)$ spreads in n horizontally contiguous tiles. Figure 5.9 presents the transmission time of these images undergoing $T(5, 0)$ translation. The sizes of images $I(n, 1)$ are given in the horizontal axis. In order to gather the measurements from the FPGA-based prototype, the topology of the nodes used for the display wall was modified in order to construct the necessary 8 node chain. The results are somewhat surprising and rather non-intuitive. Despite the fact that each pixel has equal access to network resources available to complete the transfer, the time to complete translation depends not only on the tile size (i.e. W, H) and network bandwidth, but also on the overall size of the image. Detailed examination of channel utilisation gives some insights into this behaviour.

Once the packet transmission begins, the overall network traffic quickly settles into a steady state. In order to reveal the details of this state, additional simulations with an image of size $I(10, 1)$ translated by $T(5, 0)$ have been conducted. The tile size has been increased to $W \times H = 200\text{Pix} \times 200\text{Pix}$ to validate the duration of the steady state. Figure 5.10 depicts the utilisation of the network channels relevant to this translation during the steady state. The horizontal axis of the chart positions the tile locations in a linear chain network configuration. The image in its initial position is stored in tiles 1 to 10. After translation $T(5, 0)$, the image is located in tiles 6 to 15. The blue data series of the chart represents the utilisation of the horizontal channel E of each tile. The red data series represents the utilisation of the channel that sinks the data from the router to the tile. It is clear from the chart that the majority of channels are underutilised with only one channel reaching full utilisation. Certainly, the very same results are reported for image sizes $I(10, k)$, where $k > 1$.

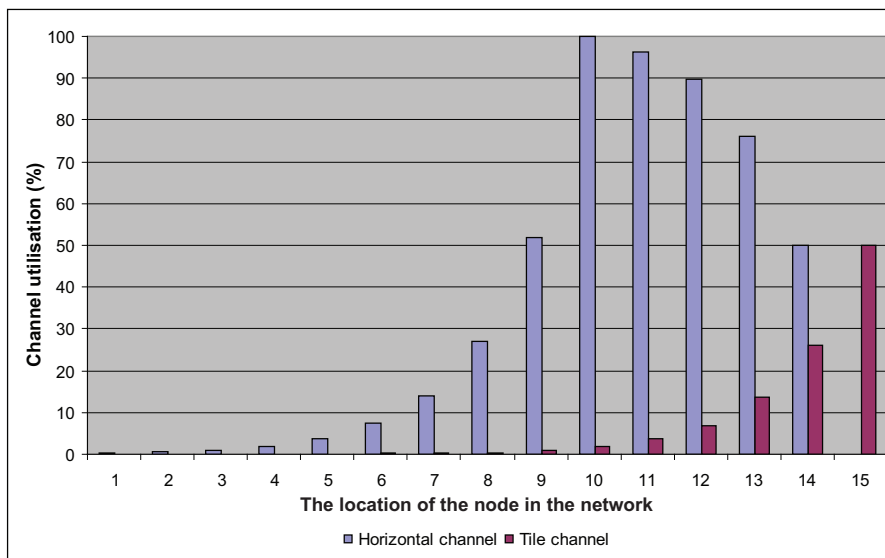


Figure 5.10: Steady state for a 15×1 network, undergoing $T(5, 0)$ traffic (one VC) showing channel utilisation for horizontal channels and tile (sink) channels

A multi-commodity flow model of this network gives a more comprehensive insight into this network behaviour. Before we analyse the problem, the notation that follows will be defined. Let N_W, N_H be positive integers representing width and height, correspondingly. Let the set of nodes be $N = \{(x, y) | x \in \{0, 1, \dots, N_W - 1\} \wedge y \in \{0, 1, \dots, N_H - 1\}\}$, (i.e. $N = \{0, 1, \dots, N_W - 1\} \times \{0, 1, \dots, N_H - 1\}$, where “ \times ” denotes the set Cartesian product [178]). For clarity, we will refer to node (x, y) as $n_{x,y}$.

Definition 1 A 2D mesh is a graph $G = (N, C)$, such that $C \subset N \times N$ defined as follows

$$C = \{(\mathbf{u}, \mathbf{v}) | \begin{array}{l} \mathbf{u} = (x_u, y_u) \in N, \mathbf{v} = (x_v, y_v) \in N, \\ ((x_u = x_v \wedge y_u = y_v \pm 1) \vee (x_u = x_v \pm 1 \wedge y_u = y_v)) \end{array} \}$$

We will distinguish between the set of horizontal channels

$$C_H = \{(\mathbf{u}, \mathbf{v}) | \mathbf{u} = (x_u, y_u) \in N \wedge \mathbf{v} = (x_v, y_v) \in N \wedge (x_u = x_v \pm 1 \wedge y_u = y_v)\}$$

and the set of vertical channels

$$C_V = \{(\mathbf{u}, \mathbf{v}) | \mathbf{u} = (x_u, y_u) \in N \wedge \mathbf{v} = (x_v, y_v) \in N \wedge (x_u = x_v \wedge y_u = y_v \pm 1)\}$$

The port connecting node $n_{x,y}$ to channel $((x, y), (x, y + 1))$ is referred as *north* (or N). Similarly, ports connecting the same node with channels $((x, y), (x + 1, y))$, $((x, y), (x, y - 1))$, $((x, y), (x - 1, y))$ are referred to as east (E), west (W) and south (S).

For ease of notation, we will denote as $c_{i,j}^h$ the horizontal channel that connects the eastern output of node $n_{i,j}$ with western input port of node $n_{i+1,j}$ ($i < N_W - 2$). Similarly, we will denote as $c_{i,j}^v$ the vertical channel that connects the output of node $n_{i,j}$ with input of $n_{i,j+1}$. Although this notation is not exhaustive for all the channels in C (e.g. there is no notation for vertical channels that head north, or west), it will be sufficient for most of the treatments that follow.

Consider the translation $T(\bar{k}, 0)$ of an image with size $(n \times W, 1 \times H)$. Let $\gamma(c)$ be the load of channel $c \in C$ as defined in Definition 1. For this particular traffic pattern, channel loads on each router can be modelled with the following equations:

$$\gamma(c_{i-1,0}^h) + \gamma(t_{i,0}^O) = \gamma(c_{i,0}^h) + \gamma(t_{i,0}^I)$$

where, $t_{i,j}^I$ ($t_{i,j}^O$) is the tile input (output) channel for tile $n_{i,j}$. This equation describes the equilibrium between the input and output channel loads during the steady state of the given traffic pattern. That is, for any given tile, the sum of the input channel loads coming from the current tile and the upstream tile (if it exists) is equal to the sum of output channel load of the downstream tile and the current tile. The constraints set by the transformation $T(\bar{k}, \bar{0})$ are:

$$\begin{aligned} \gamma(t_{0,0}^I) = \gamma(t_{1,0}^I) = \dots = \gamma(t_{k-1,0}^I) = 0 \\ \gamma(c_{0,0}^h) = \gamma(t_{0,0}^O) \end{aligned} \quad (5.4)$$

The first equation specifies that no data is sunk at the first k tiles (Figure 5.11). The second equation depicts the fact that the network activity of the first tile consists of transporting all its data to the downstream tile. During the steady state of translation $T(\bar{k}, \bar{0})$, the injection rate in tile $n_{i,j}$, will be the same as the traffic leaving the network in tile $n_{i+\bar{k},j}$, or in other words $\gamma(t_{i,0}^O) = \gamma(t_{i+\bar{k},0}^I)$.

For $i \geq \bar{k}$, the equation looks like:

$$\gamma(c_{i-1,j}^h) + \gamma(t_{i,j}^O) = \gamma(c_{i,j}^h) + \gamma(t_{i-\bar{k},j}^O) \quad (5.5)$$

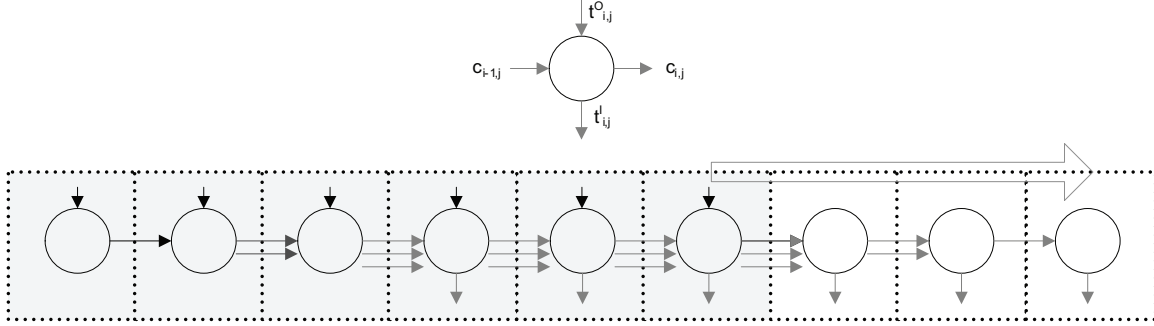


Figure 5.11: A data-flow diagram of a 9×1 network with an image size $I(6,1)$ undergoing translation $T(3,0)$

Looking at Figure 5.11, it is obvious that two input channels ($c_{i-1,j}^h$ and $t_{i,j}^O$) are contending for channel $c_{i,j}^h$. Assuming a fair arbitration policy with equal priority for all input channels and non-empty input queues (such as in steady states presented above), channels $c_{i-1,j}^h$ and $t_{i,j}^O$ equally contribute to channel $c_{i,j}^h$. Considering that channel $t_{i,j}^I$ is supplied exclusively from $c_{i-1,j}^h$, the impact of the arbitration policy can be depicted with the following equations:

$$\gamma(c_{i-1,j}^h) - \gamma(t_{i,j}^I) = \gamma(t_{i,j}^O) = \frac{\gamma(c_{i,j}^h)}{2} \quad (5.6)$$

There are two major points to be derived from the equation above

Lemma 2 For an image of size $I(n,1)$ undergoing translation $T(\bar{k},\bar{0})$, the following inequality holds:

$$\gamma(t_{0,j}^O) < \gamma(t_{1,j}^O) < \dots < \gamma(t_{n,j}^O)$$

This lemma can be proved by mathematical induction. It is very clear from equations 5.6 and 5.4 that for $i < \bar{k}$, $\gamma(t_{i,j}^O) = 2^i \gamma(t_{0,j}^O)$, which proves the lemma true for $i < \bar{k}$ - thus proving the *basis* of induction. Now, let's assume that the statement holds for up to $i \leq n_0 < n$ (i.e. $\gamma(t_{i-1,j}^O) < \gamma(t_{i,j}^O)$).

$$\begin{aligned} \gamma(t_{n_0+1,j}^O) - \gamma(t_{n_0,j}^O) &= \\ (\text{from 5.5}) &= \gamma(c_{n_0,j}^h) - \gamma(t_{n_0-k,j}^O) - \gamma(t_{n_0,j}^O) \\ (\text{from 5.6}) &= 2\gamma(t_{n_0,j}^O) - \gamma(t_{n_0-k,j}^O) - \gamma(t_{n_0,j}^O) \end{aligned}$$

From the induction basis, we have:

$$\gamma(t_{n_0,j}^O) - \gamma(t_{n_0-k,j}^O) > 0 \Rightarrow \gamma(t_{n_0,j}^O) > \gamma(t_{n_0-k,j}^O)$$

Lemma 3 For an image of size $I(n,1)$ undergoing translation $T(\bar{k},\bar{0})$, the following inequality holds:

$$\gamma(c_{0,j}^h) < \gamma(c_{1,j}^h) < \dots < \gamma(c_{n,j}^h)$$

Similarly, this lemma is also proved with mathematical induction. It is clear from 5.4 that $\gamma(c_{i-1,j}^h) = \gamma(c_{i,j}^h)/2 < \gamma(c_{i,j}^h)$ for $1 \leq i < \bar{k}$. For these tiles, during steady state the following holds:

$$\gamma(c_{i-1,j}^h) = \gamma(t_{i,j}^O) = \gamma(t_{i+k,j}^I)$$

and therefore, $\gamma(c_{i-k-1,j}^h) = 2\gamma(t_{i,j}^I)$.

Now, assume that the statement holds for up to $i \leq n_0 < n$ (i.e. $\gamma(c_{i-1,j}^h) < \gamma(c_{i,j}^h)$). As a result:

$$\begin{aligned} \gamma(c_{i,j}^h) > \gamma(c_{i-1,j}^h) &\Rightarrow 2(\gamma(c_{i-1,j}^h) - \gamma(t_{i,j}^I)) > \gamma(c_{i-1,j}^h) \\ &\Rightarrow \gamma(c_{i-1,j}^h) - 2\gamma(t_{i,j}^I) > 0 \\ &\Rightarrow \gamma(c_{i-1,j}^h) - \gamma(c_{i-k-1,j}^h) > 0 \end{aligned} \quad (5.7)$$

And now for the inductive step:

$$\begin{aligned} \gamma(c_{n_0+1,j}^h) - \gamma(c_{n_0,j}^h) &= 2(\gamma(c_{n_0,j}^h) - \gamma(t_{n_0+1,j}^I)) - \gamma(c_{n_0,j}^h) \\ &= \gamma(c_{n_0,j}^h) - 2\gamma(t_{n_0+1,j}^I) \\ &= \gamma(c_{n_0,j}^h) - \gamma(c_{n_0-k,j}^h) \\ &> 0 \end{aligned}$$

The last inequality holds due to inequality in 5.7.

Lemma 3 implies that when using an arbitration policy that gives equal priority to the west input port and the tile input port, the translation $T(\bar{k}, \bar{0})$ permits maximal utilisation of only one point-to-point channel ($c_{n,j}^h$). Other channels cannot reach saturation. Since the sequence of tile injection rates ($\gamma(t_{i,j}^O)$) is strictly monotonous and bounded (Lemma 2), the average injection rate decreases as the number of tiles in the flow increases. Simulation results presented in Figure 5.10 comply with the statements given in these lemmas. This justifies the behaviour presented in Figure 5.9.

Increasing the number of virtual channels per input port generally improves the router performance [34]. When this is done in our experiments (except the tile input/output port), the channel utilisation figures drift away from the predictions of the flow model given above. Figure 5.12 presents channel utilisation under a range of virtual channels per input port (the tile port still has only 1 VC).

There is one important observation arising from Figure 5.12: despite port level fair arbitration, as the number of VCs increases, so does the gap between $\gamma(c_{i-1,j}^h) - \gamma(t_{i,j}^I)$ and $\gamma(t_{i,j}^O)$, which conflicts with the strong fairness assumption resulting from the matrix arbiter (equation 5.6). This gap originates from the mismatch of the number of VCs on the tile input port and other input ports. In the case of 4 VCs per input port (except the tile), the only VC of the tile input port in the tile is contending for the same output port with four VCs in input port W. Once a VCID is allocated to the tile input port, it will hold on to it until the packet is transmitted. In the meantime, input port W may have up to four allocated VCIDs for its VCs. If the VC corresponding to the VCID allocated to the input tile in the downstream router is full, the tile input port cannot write to the output port, nor can it switch to another VC in the meantime. The same happens for individual VCs in the input port W. However, since there are four VCs in W, there are also more VC allocation requests coming from W and consequently more VCID grants. It should be emphasised here that the discussed gap does not appear at tile $n_{9,0}$. Since

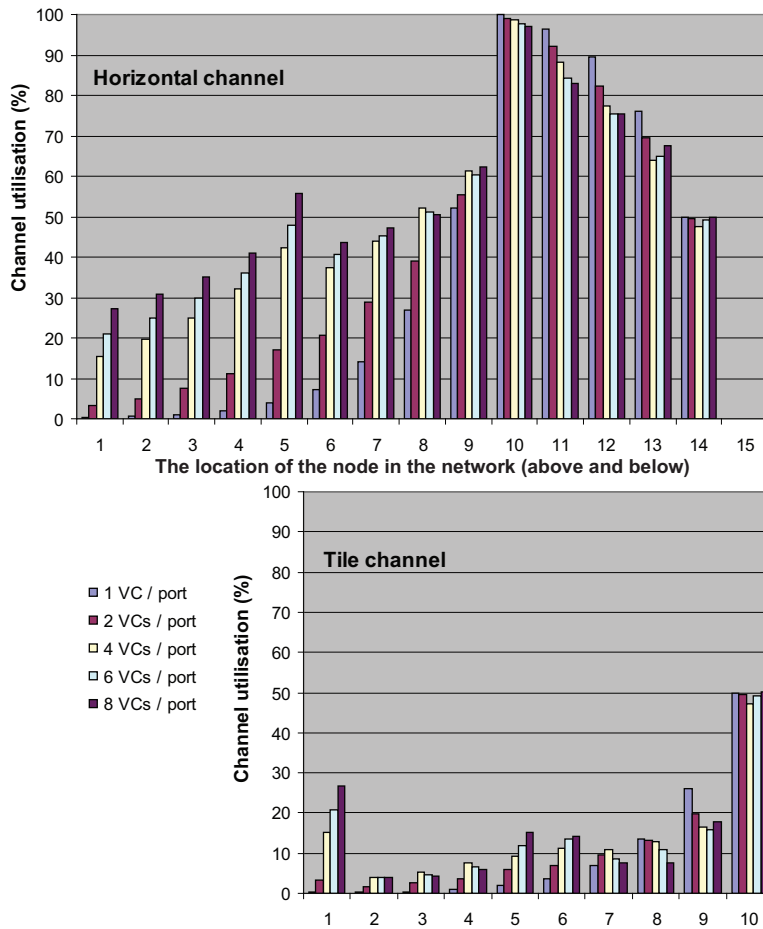


Figure 5.12: Steady state channel utilisation for a 15×1 network, undergoing $T(5,0)$ for various numbers of VCs per input port

downstream router $n_{10,0}$ does not take any traffic from the tile, it is not overloaded and therefore never blocks any of the virtual channels of the input port W. As there is no blocking in the output port E of tile $n_{9,0}$, all the switch grants from both input ports (tile and W) result in writes to output port E.

Due to the lack of any contention, the tile input port of tile $n_{0,0}$ manages to keep full input buffers of all virtual channels in the input port W of tile $n_{1,0}$, and this produces another side-effect: $\gamma(t_{0,0}^O) > \gamma(t_{1,0}^O)$, which is clearly visible in Figure 5.12.

Figure 5.12 also suggests that the increase in the number of virtual channels improves performance. However, a large number of virtual channels comes with a high implementation cost. In addition, flow control imperfections caused by non-maximal matching in separable switch allocators become more relevant.

On the whole, this grave performance penalty results from an arbitration policy which is locally fair but globally unfair. Strong system-level fairness can be achieved using queuing arbiters [34] and global timestamps for packets in the network. Queuing arbiters use more resources than matrix arbiters and adding timestamp information on packet or

flit granularity results in high communication overheads and therefore lower overall data throughput.

Controlling tile injection rate

We show that a much more efficient solution can be achieved by controlling the tile traffic injection rate. Controlled injection rates are enforced in order to avoid network saturation. From this perspective, this approach resembles aggregate resource allocation policies for providing network quality of service [34]. Optimal injection rates can be calculated from the system of equations which can be derived from the multi-commodity flow model presented above. For an image of size $I(n, 1)$ undergoing translation $T(\bar{k}, \bar{0})$ where $\bar{k} < n$, the system of equations is:

$$\begin{aligned}
 \gamma(c_{0,0}^h) &= \gamma(t_{0,0}^O) \\
 \gamma(c_{1,0}^h) &= \gamma(t_{0,0}^O) + \gamma(t_{1,0}^O) \\
 &\dots \\
 \gamma(c_{\bar{k}-1,0}^h) &= \gamma(t_{0,0}^O) + \gamma(t_{1,0}^O) + \dots + \gamma(t_{\bar{k}-1,0}^O) \\
 \gamma(c_{\bar{k},0}^h) &= \gamma(t_{1,0}^O) + \gamma(t_{2,0}^O) + \dots + \gamma(t_{\bar{k},0}^O) \\
 &\dots \\
 \gamma(c_{\bar{k}+i,0}^h) &= \gamma(t_{i+1,0}^O) + \dots + \gamma(t_{\bar{k}+i,0}^O) (i < n) \\
 \gamma(c_{n-1,0}^h) &= \gamma(t_{n-(\bar{k}-1),0}^O) + \dots + \gamma(t_{n-1,0}^O) \\
 &\dots \\
 \gamma(c_{n-1+\bar{k},0}^h) &= \gamma(t_{n-1,0}^O)
 \end{aligned}$$

The given system of equations can be augmented with channel load limitations (i.e. not exceeding 100%) into a linear programming model. The linear function to be maximised is:

$$l = \sum_{i=0}^{n-1} t_{i,0}^I$$

This problem is under-constrained and therefore has many solutions. Most of the possible solutions result in tile injection rates that are not related to one another, which are unacceptable for any arbitration policy. The best solution for fair arbitration is:

$$\gamma(t_{i,0}^O) = \frac{1}{\bar{k}}, \forall i \in \{0, 1, \dots, n-1\} \quad (5.8)$$

Figure 5.13 presents the performance improvements made by controlling the tile injection rate. Controlling the injection rate at the source tile greatly improves the channel utilisation and therefore the overall transmission time. For a chain of 10 tiles (Figure 5.13) this improvement is roughly two-fold. Results from Figure 5.9 suggest that this performance improvement becomes larger with the increase of image size. Measurements of translation of image $I(50, 1)$ undergoing $T(5, 0)$ conclude that this translation is done 6 times faster with a tile injection rate of 0.17 in comparison to an injection rate of 1.00.

Figure 5.13 also makes it clear that the suggested theoretical injection rate of $1/\bar{k} = 0.25$ does not give the best performance. This results from two factors:

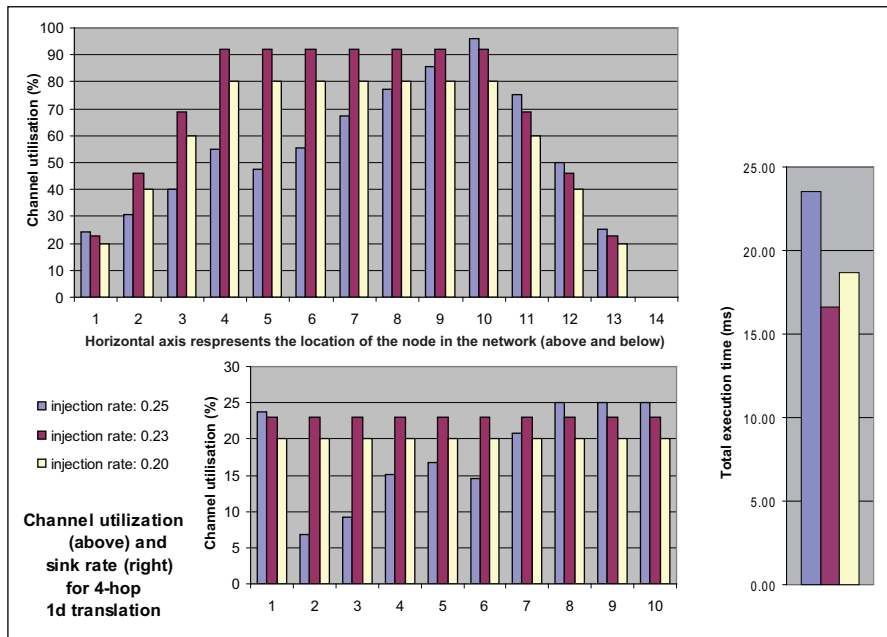


Figure 5.13: Simulation results for a 14×1 network with an image size $I(10,1)$ undergoing translation $T(4,0)$ under various tile injection rates

- *Non-maximal matching* in the switch allocator – The results presented in Figure 5.13 were gathered with routers having 4 VCs per input port. Consider the following scenario: input port W has four VCs, one requesting the tile output port and the remaining three requesting output port E. At the same time, the only VC in the tile input port is also requesting output port E. One of three VCs requesting output port E wins the first allocation stage in input port W and loses switch access at the second stage of allocation against tile input port. Measurements show that during steady state, this scenario accounts for up to 10% of allocations for routers with contending input ports (routers in tiles $n_{1,0}, n_{2,0}, \dots, n_{10,0}$ in the example given in Figure 5.13). Evidently, in the given scenario, a better allocation policy would have granted switch access to the only VC in the input port W requesting the tile output port.
- *Channel saturation* – The router activity during this type of translation (i.e. $T(4,0)$, or generally $T(\bar{k}, 0)$) can be approximated with a simple queuing system model developed in Queuing Theory¹. Figure 5.14 presents the model of the router switch for six tiles that source and sink data (i.e. routers in tiles $n_{5,0}, n_{6,0}, \dots, n_{10,0}$) when undergoing this translation. Considering that all incoming packets have equal length, both the channel server and the tile server have deterministic service times, allowing us to model the queues using the M/D/1 model. This model assumes that inter-arrival times are exponentially distributed. The split of the incoming packets in this model takes advantage of the properties of the Poisson arrival process – splitting a stream of packets with an exponential inter-arrival time distribution gives multiple streams of the same distribution. This model also assumes that during the process

¹For a quick introduction to Queuing Theory with emphasis on network performance analysis, the reader is referred to Chapter 23 in [34]

of splitting, the packets that end up in the current tile do not access the switch, and therefore do not interfere with the packets that continue to the next tile. This assumption makes our model overly optimistic since, as discussed in the previous point, these packets do access the switch and may be blocked due to non-maximal matching in the allocation policy. The switch is modelled as an output queued switch because there is no easy way to model the switch contention of an input queued switch [34]. Under these conditions, the expected occupancy of the M/D/1 output queue at the switch/router of tile $n_{i,0}$ is:

$$E(N_Q(i)) = \frac{\lambda}{2(\mu_i - \lambda)} \quad (5.9)$$

The service rate μ_i at tile $n_{i,0}$ is affected by the back-pressure from tile $n_{i+1,0}$, thus $\mu_5 \leq \mu_6 \leq \mu_7 \leq \mu_8 \leq \mu_9 \leq \mu_{10} = 1$. Equation 5.9 shows that even in this optimistic model, the network saturates at $\lambda = \min\{\mu_i\}$. A controlled injection rate of $1/k = 0.25$ for T(4,0) results in $\lambda = 1$, thus always saturating the network.

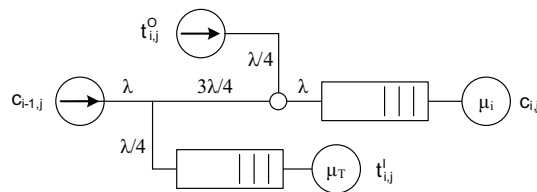


Figure 5.14: A queueing model of the switch during translation T(4, 0). The channel source has rate λ and the tile source has rate $\lambda/4$

For non-maximal matching switch allocators, blocked output ports may further contribute to poor resource allocation. In the allocation scenario given above, the VC from the tile input port, having been granted switch access, may end up facing a blocked output port, resulting in no flits transmitted during this allocation process. This can be avoided by masking switch requests that have blocked output ports.

Figure 5.15 presents transmission rate improvements when requests from input VCs that seek blocked output ports are masked during switch allocation. It is clear that improvements are barely noticeable. This technique only reduces some diminishing effects of non-maximal matching. More elaborate switch allocation mechanisms such as iSLIP [101] or Wavefront [160] may yield further performance improvements at the cost of increased router complexity. However, in the example given in Figures 5.13 and 5.15, this improvement would only close the gap between performance results from 0.23 and a theoretical limit that an injection rate of 0.25 promises. The difference between these two injection rates suggests that when performing translation T(4, 0) a better switch allocator would not give performance improvements of more than 2%.

O1TURN routing

O1TURN [137] routing improves performance by utilising up to two non-minimal paths for the same source-destination pair. When performing translations of type $T(\bar{d}x, 0)$ or $T(0, \bar{d}y)$, these two paths are identical, and therefore O1TURN does not offer any

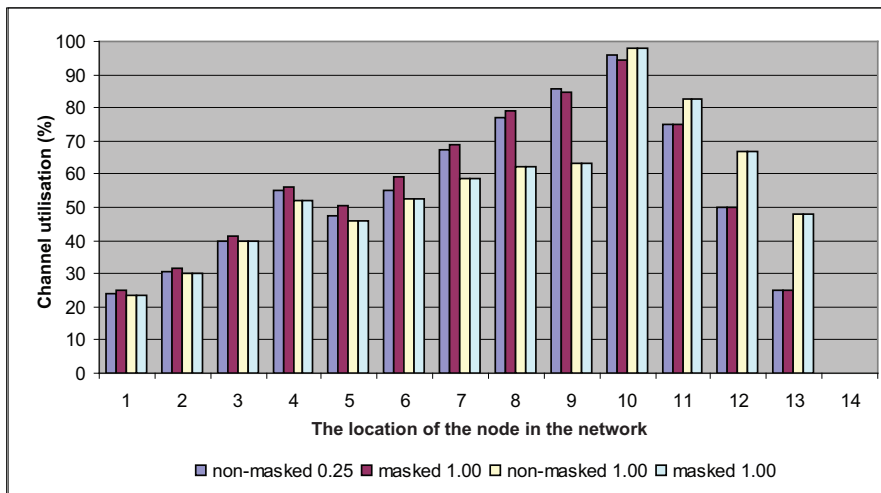


Figure 5.15: A comparison of channel utilisation for a 14×1 network with an image size $I(10,1)$ undergoing translation $T(4, 0)$

performance improvements. At the first sight, one may think that O1TURN routing yields performance improvements for all other types of translation. Figure 5.16 presents a diagram that distinguishes types of translation that benefit from O1TURN routing.

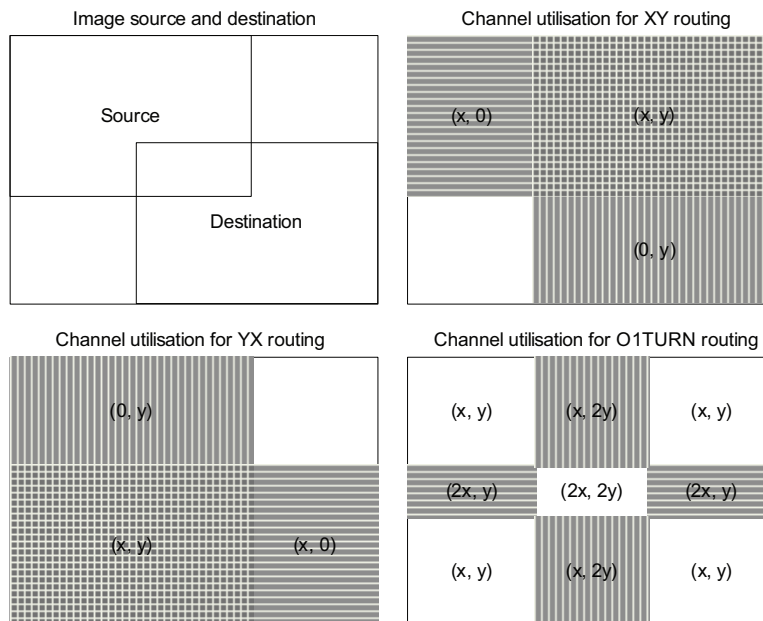


Figure 5.16: A map of resulting channel utilisations for XY, YX and O1TURN routing functions

In Figure 5.16, (x, y) denotes maximum channel utilisation in horizontal and vertical channels for the corresponding region. Considering the aforementioned implications of overloading channels, it is critical to keep the channel load under $(1, 1)$ at every network router. Unless injection rates are set in tile granularity, no performance improvement can be gained with O1TURN routing since regions with channel load of $(2x, y)$, $(x, 2y)$ and

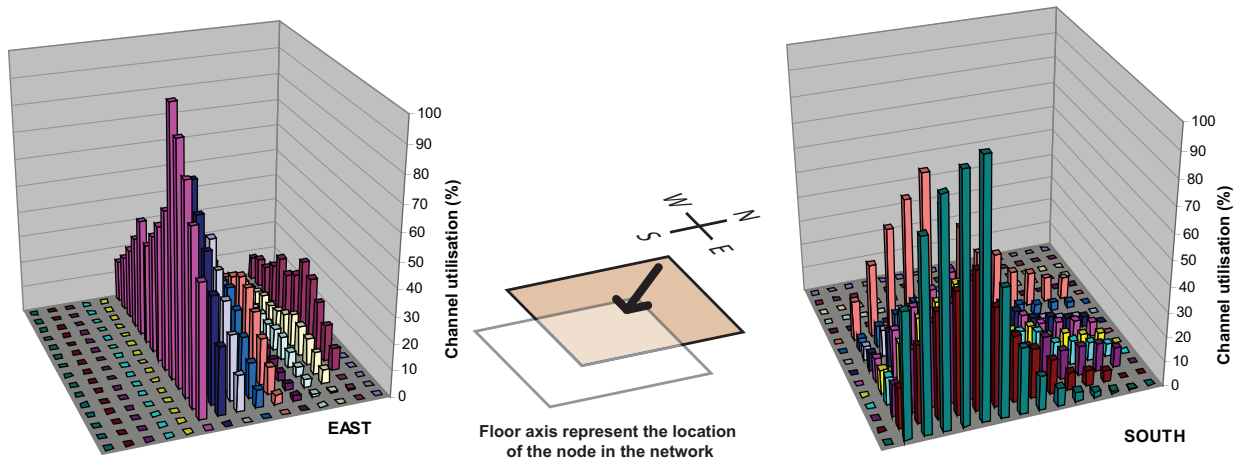


Figure 5.17: Steady state channel utilisation for a 15×15 network, under $T(5, 5)$ traffic

$(2x, 2y)$ will bring the upper limits of tile injection rate down to injection rates tolerable for XY or YX routing.

Figure 5.16 also suggests that O1TURN routing brings performance improvement only for translations where overlapping regions $((2x, y), (x, 2y)$ and $(2x, 2y)$) are empty. For images $I(m, n)$ being translated with $T(\bar{d}x, \bar{d}y)$ O1TURN routing yields faster translations when $|\bar{d}x| > m$ and $|\bar{d}y| > n$. Under such conditions the tile injection rate can be increased to twice the injection rate tolerable by XY or YX routing and therefore complete the transmission twice as quickly.

Translations of the form $T(\bar{d}x, \bar{d}y)$ where both $\bar{d}x$ and $\bar{d}y$ are non-zero undergo through similar performance penalties due to global unfairness in allocation. Figure 5.17 shows both, horizontal and vertical channel utilisation when image of size $I(10, 10)$ undergoes translation $T(5, 5)$.

In a similar approach to the 1D case, the controlled injection rate is derived by the number of hops per dimension. In order to guarantee that any channel load (horizontal or vertical) never exceeds 1.00, controlled injection rate is kept below $1/\max(\bar{d}x, \bar{d}y)$.

Reflection

Other transformations also yield suboptimal performance as a result of the system level allocation unfairness. Figure 5.18 presents horizontal (E) and tile channel utilisation values during a vertical flip of the object presented in Figure 5.19. The triangular shape of the image reveals the channel utilisation drop as h increases.

A controlled injection rate improves utilisation, but the gains are not as significant as in translation. Figure 5.20 gives a comparison. The results for the first datapoint ($I(3,3)$) are measured both in the hardware prototype and in the simulation environment.

This is due to the fact that the stationary state captured in Figure 5.18 holds for a much shorter time than in translation and therefore has less significance to the overall transmission time. The origin tile that injects the highest traffic delivers to the nearest destination. As a result, the complete tile is transmitted faster, which then releases more network resources for other more distant tiles.

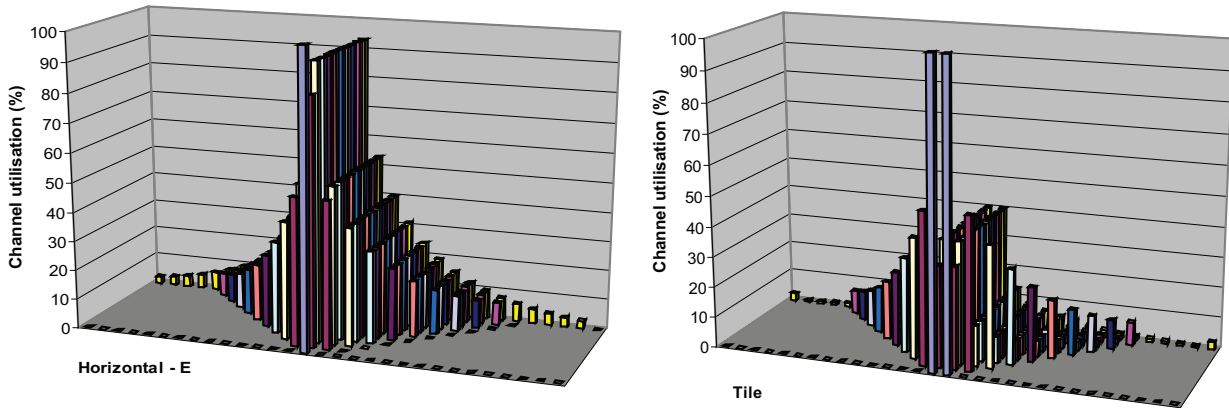


Figure 5.18: Vertical flip of the set of tiles presented in Figure 5.19

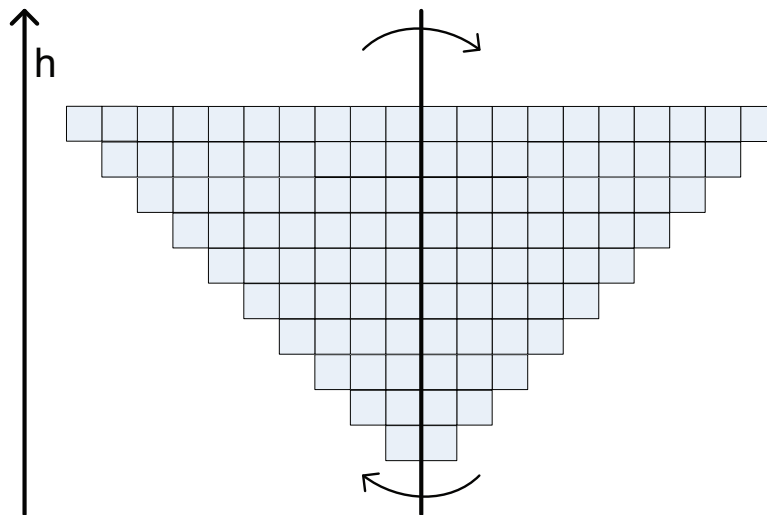


Figure 5.19: A triangle-shaped set of tiles undergoing a flip across vertical axis

Due to identical paths for XY and YX routing, O1TURN routing does not yield any improvements for horizontal and vertical flip. Improvements are possible for reflections that result in object displacement in both dimensions.

Rotation

Rotation of images around their centre-point, reveal similar performance issues. Controlled injection rates improve network transmission through better channel utilisation. Simulations of images with tile size of 200×200 pixels, undergoing 180° rotation around their centre-point reveal improvements in network channel utilisation and consequently in transmission times.

A conservative policy for injection rates of $\gamma(c) = (1/2n - 0.02)$ is employed in order to avoid any channel overload. Figure 5.21 shows that the transmission time improves between 34% and 41%. Similar to the measurements in reflection, the results for the first datapoint (I(3,3)) are measured both in the hardware prototype and in the simulation environment.

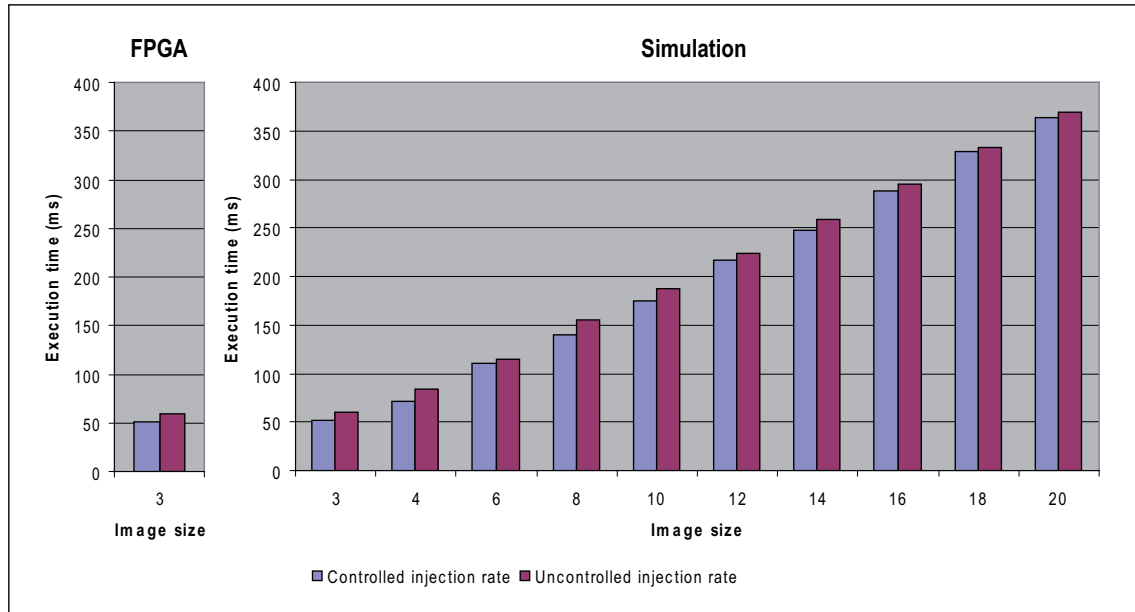


Figure 5.20: A comparison between controlled and uncontrolled injection rates for a range of images $I(n,1)$ (n is given in x axis) undergoing vertical reflection around their centre (FPGA and simulation; partition and merge time: $8.63ms$)

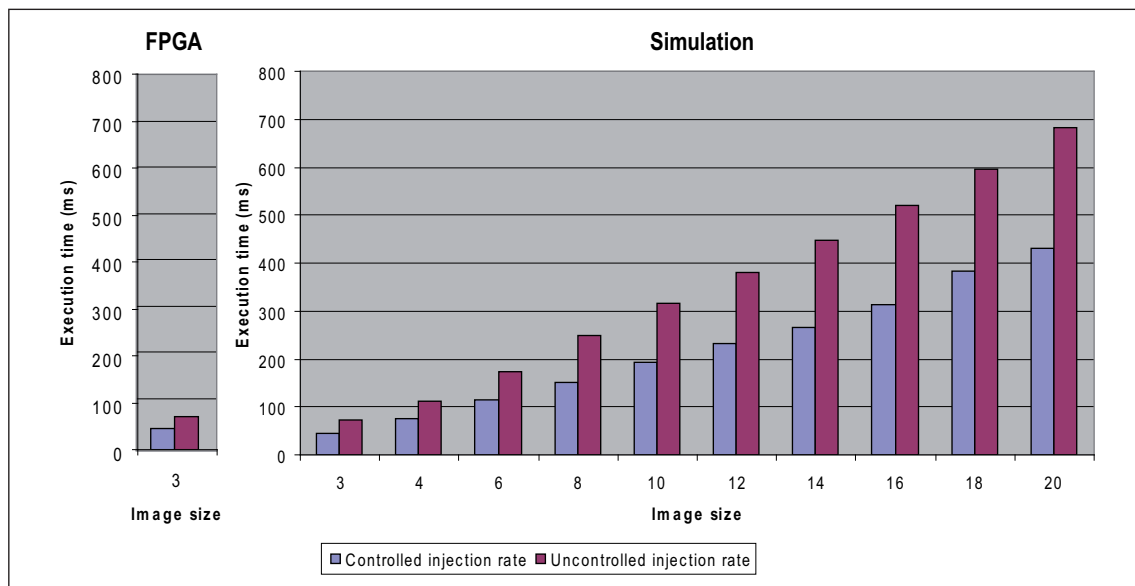


Figure 5.21: A comparison between controlled and uncontrolled injection rates for a range of images $I(n,n)$ (n is given in x axis) undergoing 180° rotation around the centre-point of the image (FPGA and simulation; partition and merge time: $8.97ms$)

Figure 5.23 presents the channel utilisation when injection rates are not controlled. Due to XY routing algorithm, the utilisation of south channels under this transformation is symmetrical to that of north channels. Similarly, utilisation of east channels is symmetrical to that of west channels.

Figure 5.23 presents the spatial distribution of tile-channel utilisation, exposing high sink rates in central tiles, which reduces the life-span of this steady-state traffic.

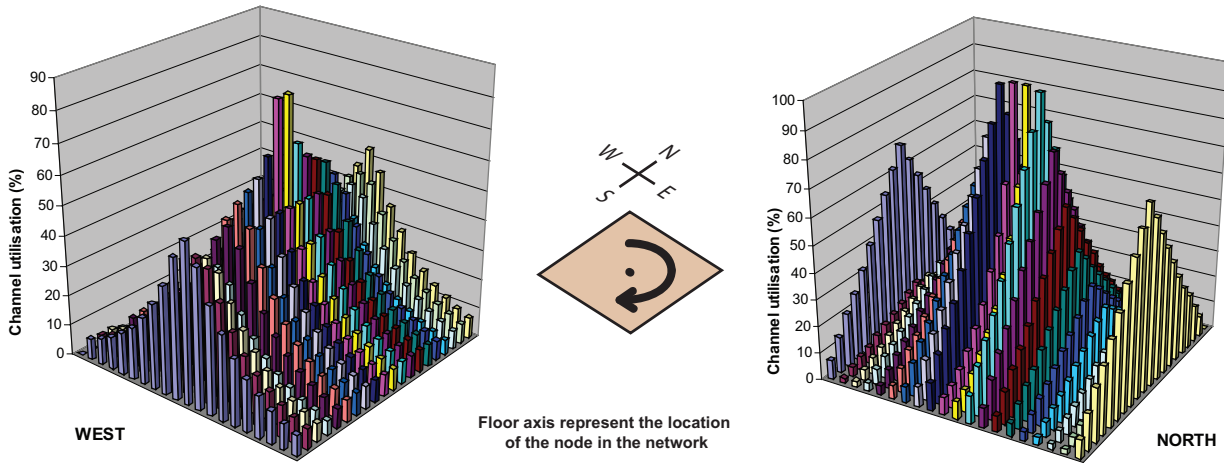


Figure 5.22: Network channel utilisation in steady state traffic for an image $I(20,20)$ undergoing 180° rotation around its centre

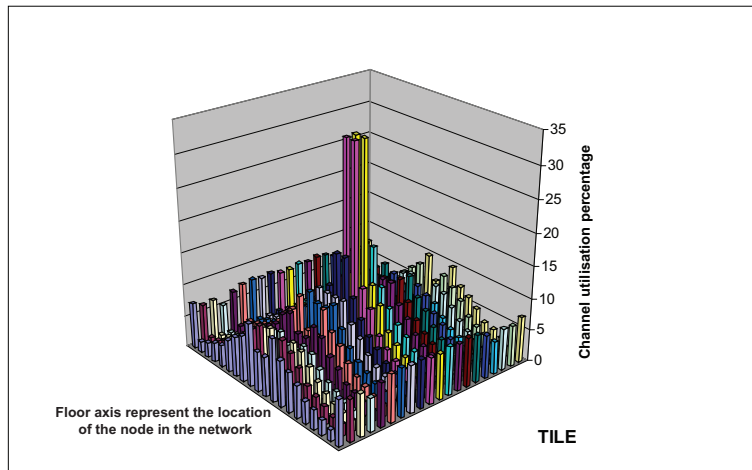


Figure 5.23: Tile sink rates for the same transformation as given in Figure 5.22

Finally, we revisit the initial aims that were set out for this algorithm at Section 5.4.1.

- *Scalability* – the approach to 2D transformations presented here distributes the processing and communication workload to the origin tiles and destination tiles. The amount of processing required on each tile (i.e. algorithms 2 and 3) scales linearly with the tile size. Furthermore, as noticed in the transformation examples given in Section 5.5, the controlled injection rates are determined by the image size and the corresponding transformation matrix but not by the tile size. Since the amount of data that tiles inject into the network is proportional to the tile size, under controlled injection rates the transmission of this data over the network also scales linearly.

As noticed with the translation locality, with controlled injection rates, the image size does not have an effect on the total transformation time. However, this is not the case with other transformations, since the image size and the transformation matrix determine the amount of network resource contention.

- *Communication overhead* – if we assume as overhead any communicated information except the essential pixel data movements from source to destination tiles, then the main overheads incurred by this algorithm are produced by the control tile. However, the amount of information communicated by the control tile (e.g. transformation matrix) is very small compared with the amount of pixel information that is transferred over the network. The number of such control messages is mainly determined by the number of tiles where the corresponding image is displayed, but it also depends on the distance between the control tile and the origin tiles. In the simulations presented above, such control messages always resulted in less than 1% of total network traffic.

5.6 Summary

This chapter presented a distributed algorithm for performing 2D transformations on images for computing surface platforms. The algorithm spreads the work among tiles which host the original image and tiles where the destination image ends up. The only communication overheads are incurred by control tiles messaging and are negligible. A naive implementation of the transmission mechanisms may result in drastic performance degradation.

Each transformation matrix conveys information about the network traffic pattern it exhibits, which when combined with the routing function information can be used to calculate optimal tile injection rates. Controlling the tile injection rate improves the network performance and avoids the need to consider more complex router architectures. More specifically they minimise the effects of globally unfair arbitration policies, which otherwise would require FIFO arbiters and global timestamping. It is also shown that a presence of better allocation mechanisms would not bring great improvements in performance.

TIME SYNCHRONISATION

We live in a world where the concept of time is of critical importance in most of our day-to-day activities. Due to the widespread availability of relatively well synchronised clocks, this dependence on synchronised time is mostly taken for granted. Many applications running in distributed systems also depend on a coherent notion of time. Furthermore, activities in distributed applications often require much higher precision of synchronised time than human activities. Any coordination or measurement process, distributed among a set of networked nodes is likely to require synchronised time of millisecond or even microsecond precision. Dedicated algorithms are necessary to achieve and maintain time synchronisation with such precision.

Distributed input and output algorithms presented in Chapters 4 and 5 barely required any explicit time synchronisation or timekeeping. The distributed input gesture recognition platform is self-timed – the common middleware platform coordinates its execution by events triggered from gesture input or network messaging. As we move further to more sophisticated modes of interaction, the need for synchronised time becomes more apparent. Consider the ever-present act of ‘double-clicking’ in personal computers – the time length between two clicks is measured to distinguish between two consecutive clicks and a double-click. Correspondingly, an implementation of the analog concept of ‘double-tapping’ (see Section 2.5.3 and [184]) in computing surfaces would also require explicit time measurements. The presented distributed output transformations are also insensitive to time when executed in isolation. Nevertheless, their use in other algorithms will often require coordinated timed actions. Consider the graphical modelling of movements in the physical world. While distributed transformations could be at the core of movement implementations, a common notion of time across affected tiles is imperative. Interactive games often rely on temporal information of users’ input and the user input may be coming from a number of tiles, so synchronised time is necessary to provide temporal ordering of input events. Also, a distributed implementation of video playback across tiles would require synchronised clock phases for achieving simultaneous frame transitions and synchronised clock rates for displaying frames for the same amount of time. All existing commercial systems of large-display solutions rely on either centralised rendering or centralised frame-buffer. In both situations, synchronisation will be implicitly achieved

at the centralised point. Scalable display architectures that are developed in research institutions (Section 2.5.1) also encounter the time synchronisation problem. They address this problem implicitly: the output of all displays is controlled by a central node that broadcasts explicit coordination messages for actions such as frame buffer swapping (e.g. [20, 21]). Given that such display walls rely on high performance systems, the fast delivery and processing of coordination messages will result in small asynchronies between individual display tiles.

For larger systems with distributed rendering and distributed frame-buffer that have no broadcast channels, some explicit means of time synchronisation will be required. The requirements for synchronised time vary for the applications mentioned above. User interactions would tolerate higher asynchrony compared to more demanding applications. Among the applications mentioned above, distributed video-playback would be the most taxing application by expecting simultaneous frame transitions. In order to achieve this, all the involved nodes would need to maintain their time differences to within a fraction of the frame (i.e. several milliseconds).

A time synchronisation algorithm in a distributed system aims to achieve a common notion of time among a set of network-connected nodes. In its simplest form, this notion can be achieved by selecting a clock source that broadcasts incremental time-steps of the highest resolution to all the network nodes. This approach is equivalent to the provision of a clock source in synchronous circuit designs. It would rely on using an exclusive physical channel for transmitting time references. This channel could be implemented by distributing a cable that is suited for long distance communications (e.g. cables used in telephony) to all the network nodes. A significant advantage of this approach is that time references are delivered very accurately. On the other hand, it also brings its own disadvantages: the channel would be used exclusively for time information and the whole approach is subject to a single-point of failure – the broadcaster of time information. Time synchronisation algorithms in distributed systems rely on the existence of clock oscillators in every node for locally reproducing local time. In addition, they use the existing network infrastructure to communicate time information among the network nodes. With ideal local clock sources and fixed network message delays, nodes would initially communicate to achieve time synchronisation, which would then be individually maintained by every node.

However, time synchronisation is hindered by clock drift and message delay uncertainty. Time information is degraded by the drift of imperfect hardware clocks, and cannot be communicated without further impairments due to delay uncertainties. The influence of drift and delay uncertainty on the quality of synchronisation is dependent on network constraints and synchronisation requirements. Rare updates of time information, which happen in networks with low communication rates result in high uncertainties caused by clock drift. Continuous updates of time information throughout the whole network are costly in network bandwidth. Various strategies that take into account application requirements are employed to reduce this cost. In some scenarios (e.g. distributed video-playback), only a subset of network nodes requires time synchronisation at any given time. For other scenarios (e.g. acoustic ranging and data fusion [40]), time synchronisation services may be required only for a limited duration. The goals of an ideal time synchronisation algorithm are to provide flexible, accurate synchronisation with minimal resource requirements.

6.1 The system model of time

Digital clocks typically consist of a counter that produces incremental time steps of an ideally fixed length. The output of this counter at node n_i at real time t will be denoted as $L_i(t)$. We will refer to this reading as *the local clock*. For the purposes of this model, the output of the local clock is assumed to be the discrete version of a continuous, differentiable function of t denoted as $l_i(t)$. The *clock rate* ρ at time t is given as the first derivative of $l_i(t)$: $\rho(t) = dl_i(t)/dt$. An ideal clock would have rate $\rho = 1$ at all times, but the rate of a real clock source over time is impacted by fluctuations in oscillator frequency caused by environmental changes in supply voltage, temperature, etc [172]. The deviation of the clock rate from the standard rate of 1 is called *clock drift* and is $\delta(t) = \rho(t) - 1 = dl_i(t)/dt - 1$. Clock drift is limited by bounds in the range of tens or hundreds of PPM¹ which, although may seem small, still pose a challenge to time synchronisation algorithms. The impact of clock drift in time synchronisation algorithms depends on the target precision of time synchronisation and the duration for which satisfactory synchronisation is desired. In order to capture its impact, three different clock models are suggested [131]:

- *Constant rate model*: Assumes a constant clock rate ($\rho(t) = \rho, \forall t$) and is a reasonable model only when the required precision is small compared to rate fluctuations.
- *Bounded drift model*: Assuming a bounded drift $|\delta(t)| \leq \delta_{max}$ is a suitable model since such bounds are typically guaranteed by oscillator manufacturers.
- *Bounded drift-variation model*: This model is a restriction on the bounded drift model which assumes that the rate of change of the clock drift $\vartheta(t) = d\rho(t)/dt = d\delta(t)/dt$ is also bounded. When the external factors that impact the clock rate have gradual and slow changes over time (e.g. temperature or supply voltage), a bounded drift-variation model is suitable. When this bound is sufficiently low, it makes drift compensation possible.

Assuming that the local clock value is stored in an n -bit register and starts ‘ticking’ at time t_0 with frequency ϕ , the output of the local clock at time $t > t_0$ is given with the following formula:

$$L_i(t) = \lfloor \phi(1 + \delta_i(t))(t - t_0) \rfloor \text{ mod } 2^n \quad (6.1)$$

where t is time, measured in seconds. For relatively short periods of time, register overflow can be ignored and a constant drift can be assumed $\delta_i(t) = \delta_i$. Under these assumptions, the above equation is simplified to:

$$L_i(t) = \lfloor \phi(1 + \delta_i)(t - t_0) \rfloor \quad (6.2)$$

Time synchronisation algorithms often transform the readings of $L_i(t)$ to new values in order for them to be consistent with time readings of other nodes, and thus achieve time synchronisation. Therefore, the globally synchronised time $\mathbb{S}(t)$ can be expressed as:

$$\mathbb{S}(t) = S(L_i(t)) \quad (6.3)$$

¹PPM = parts per million (10^{-6})

where S is the transformation applied to the local clock, with details defined by the synchronisation algorithm.

In order to be able to measure the accuracy of time synchronisation algorithms, the notion of a standard reference time is used. Local times in different nodes are compared only after transforming their time readings into standard time. For the clock model presented with equation 6.2, any local time reading $L = L_i(t)$ corresponds to a time interval T in real time. This interval is defined and calculated with the following equations

$$\begin{aligned} Inv_i(L) = T &\Leftrightarrow L_i(t) = L, \forall t \in T \\ T = \{t \mid \frac{L}{\phi(1+\delta_i)} + t_0 \leq t < \frac{L+1}{\phi(1+\delta_i)} + t_0\} \end{aligned} \quad (6.4)$$

The system will not be able to reason on the order of events that occur within the time interval $Inv_i(L) = T$. This in itself limits the achievable accuracy of synchronisation algorithms, since synchronised nodes reason about events only by communicating and comparing local time readings L .

A useful measurement for reasoning about the precision of time synchronisation algorithms is clock skew. In the example given above, assume that only node n_2 registers the event E with the timestamp L_{2E} . Node n_1 receives this timestamp from node n_2 and reasons about the event E in local time. Information about relative differences between local times of these two nodes is used for transforming this timestamp to local time: L_{1E} . Clock skew between node n_1 and n_2 at time t_E is defined as

$$\Delta L_{1E} = |L_1(t_E) - L_{1E}| \quad (6.5)$$

The floor function

The model of the local clock presented in equation 6.2 will be used for characterising the details of our approach to time synchronisation. This model uses the floor function [55] to produce discrete readings of continuous time. Some properties of the floor function will be useful for many transformations given in the forthcoming sections.

The floor function $\lfloor x \rfloor$ gives the largest integer less than or equal to x [55]. For all integers b , the floor function satisfies the identity

$$\lfloor a \rfloor - b = \lfloor a - b \rfloor \quad (6.6)$$

This identity becomes particularly useful for transforming timestamps.

For any node n_1 that writes the timestamp with value L at real time t_E , we have

$$L = L_1(t_E) = \lfloor (t_E - t_{01})(1 + \delta_1)\phi \rfloor = (t_E - t_{01})(1 + \delta_1)\phi \quad (6.7)$$

Using this observation and the identity 6.6, the number of elapsed time increments at any given time t since the timestamp L was written at real time t_E can be calculated as follows

$$L(t) - L_1(t_E) = \lfloor (t - t_{01})(1 + \delta_1)\phi \rfloor - (t_E - t_{01})(1 + \delta_1)\phi = \lfloor (t - t_E)(1 + \delta_1)\phi \rfloor \quad (6.8)$$

For events that are not timestamped locally, the following inequality becomes useful

$$\lfloor a - b \rfloor \leq \lfloor a \rfloor - \lfloor b \rfloor \leq \lfloor a - b \rfloor + 1 \quad (6.9)$$

or written as an equality

$$\lfloor a \rfloor - \lfloor b \rfloor = \lfloor a - b \rfloor + i \quad (6.10)$$

where $i \in \{0, 1\}$.

6.2 Related work

Time synchronisation in distributed systems has been studied for well over three decades with Lamport, Marzullo, Mills and Lynch being among the most distinguished contributors. By embracing distributed systems in new applications, the requirements and expectations of synchronised time become more diverse.

6.2.1 Synchronisation diversity

Local clocks throughout the network are often synchronised to a reference time provided from outside the network. This type of synchronisation is referred to as external synchronisation and is very popular on computer systems due to their need to be synchronised to the standard ‘time of day’ (e.g. computers, electronic appliances, etc). In contrast, many applications require synchronisation of local clocks only within the network. This is referred to as *internal* synchronisation with its main goal being consistency of time among network nodes. The need for external synchronisation arises only when the time information associated to network internal events is compared with time references from other systems. For applications that are internal to the system and applications that don’t need to reveal any time information, internal synchronisation is sufficient.

Synchronisation algorithms can be categorised based on the period of time during which synchronisation is required to sustain. For applications that observe external phenomena that are persistent over time, *continuous* synchronisation is necessary – i.e. network nodes constantly maintain synchronisation of a predefined accuracy. For some synchronisation algorithms, this constant maintenance of accurate synchronisation incurs a constant amount of communication over a unit of time. This can result in significant energy demands and is especially wasteful for applications that encounter long idle periods. This motivates the development of *on-demand* synchronisation algorithms where network synchronisation can be activated only during ‘interesting’ moments of time. Activation of time synchronisation can be triggered by the very events that require the service. In this case, network nodes execute the time synchronisation algorithm and then reason about the event that triggered its execution. This type of synchronisation is referred to as *post-facto* synchronisation with the most popular implementation reported in [40]. Post-facto synchronisation does not provide sufficient mechanisms for executing coordinated time-triggered events that are set to happen in the future. Algorithms that support such coordination are referred to as *pre-facto* synchronisation algorithms [131].

Time synchronisation enables distributed systems to provide two types of measurements:

- temporal ordering, and
- time-interval length

These two problems are different in that they require different synchronisation mechanisms. Clock rate synchronisation is required for nodes that measure identical time-interval lengths. In a scenario where nodes measure the time between the start and termination of a process, rate synchronisation is a necessary and sufficient condition to compare the duration of the lifetime of the process within a range of nodes. On the other hand, rate synchronisation on its own cannot reason about global temporal ordering. Explicit offset synchronisation on a distributed system provides the property that at any given time t , all nodes report local time with the same value ($\mathcal{S}(t)$ in Eq. 6.3). This consistency of time readings is able to provide temporal ordering of events. Rate synchronisation can be achieved by clock discipline methods [104]. Alternatively, it is mitigated with continuous offset synchronisation, which often proves to be costly, mainly due to large message delay uncertainties.

Offset synchronisation can also be achieved without setting local times to one synchronised time reference. An alternative approach is for each node to use local time but also collect sufficient information about neighbours' local times. When a timestamp is received from any neighbour, it is transformed into local time before being used by any application. This type of time synchronisation is referred to as *time-scale* transformation [131]. Time-scale transformation is equally expressive when compared to other time synchronisation algorithms. Conventional synchronisation has higher communication demands (due to communicating the values of time) while time-scale transformation may incur additional computation and memory overhead.

Some synchronisation algorithms communicate time intervals instead of time instants. Time intervals convey the time information and the accumulated inaccuracy error. This approach, although important for analysis of error propagation, has not received much attention in implementation. In addition, many algorithms can be designed to be aware of propagation bounds so the interval information can be reconstructed whenever necessary.

The time information can be communicated explicitly in the network or piggybacked with other data packets. Since the time information usually constitutes only several bytes, networks with large message headers incur significant overheads for explicit time information communication. For such networks, piggybacked approaches become more compelling.

6.2.2 Classical time synchronisation in distributed systems

Lamport in [85] discusses the fundamental importance of event ordering and introduces the concept of logical clocks for achieving consistent total ordering in a distributed system. He also presents a simple algorithm that enables asynchronous processors to maintain a discrete clock that provides a consistent concept of time. This internal synchronisation algorithm is extended to be applicable in conventional partially synchronous models [95] and characterises the upper and lower bounds of error imposed by delay uncertainties and clock drifts.

Marzullo [98] presents an external time synchronisation algorithm and analyses the problem of keeping clocks synchronised and correct. This work provides an analysis of

error propagation by assuming each node's knowledge of its worst case clock drift. Each node maintains an upper bound on the error of its clock, which allows an interval to be constructed that includes the correct reference time. Periodically, each processor requests the time from each of its neighbours and sets its new interval to be the intersection of its current one with the interval received in response, after adjusting for further error that could be introduced by message delay uncertainties.

Network Time Protocol (NTP) [165] – the Internet time synchronisation protocol is one of the longest running, continuously used, ubiquitous Internet protocols and consists of a suite of algorithms. Marzullo's algorithm is at the very core of the NTP which maintains fault-tolerant synchronisation between time servers.

Time synchronisation has attracted much theoretical treatment [58, 94]. In addition to formal determinations of the bounds of synchronisation accuracy, theoretical results also reveal that time synchronisation is not even possible in some distributed systems under the presence of faults [43]. Additional details and reviews of more specialised, fault-tolerant algorithms can be found in [145].

The problem posed by message delay uncertainty to time synchronisation algorithms has been addressed in several projects. Reducing this uncertainty is possible in dedicated hardware solutions. Horauer et al. [64, 65] present a Network Time Interface module for Ethernet LANs that supports high accuracy external clock synchronisation by hardware. Message delay uncertainties are minimised by delaying the timestamp write as much as possible. The idea is to insert timestamps on the fly into the memory holding the time synchronisation packet in a way that minimises the transmission/reception uncertainty. The high accuracy of timestamp measurements provides information about the relative clock drift differences for a pair of nodes. This information can be used to correct clock rates as used in this work. An accurate clock synchronisation algorithm based on minimal message delay uncertainties is also presented in [1]. This scheme is designed for IBM SP2 parallel system nodes, whose multistage interconnection network is driven by a single 40MHz oscillator (25ns cycle). SP2 contains a set of distributed counters whose values can be transferred over the network without any delay uncertainties. The only induced error comes from the phase difference between different processors which has a known upper bound from 25ns to 200ns depending on the network size. As a result, a time synchronisation algorithm based on a set of distributed counters, serving as clock sources is synchronised to within 25ns (to 200ns) for a network of 16 (to 512) nodes.

6.2.3 Time synchronisation in sensor networks

High communication demands make the use of classical time synchronisation algorithms undesirable for the low energy budget of sensor networks. In addition, the dynamic network topology of sensor networks often makes the use of classical time synchronization algorithms unsuitable. Reports in [41, 131] give a thorough justification of why classical time synchronization algorithms are not suitable for sensor networks.

Tiny-Sync and Mini-Sync [144] are used for pairwise node synchronisation. They use multiple round-trip measurements and line-fitting techniques to obtain the offset and rate difference for the pair of nodes. In contrast, algorithms such as Lightweight Time Synchronization (LTS) [171], Timing-Sync Protocol for Sensor Networks (TPSN) [49], Hierarchy Referencing Time Synchronization Protocol (HRTS) [32] and Flooding Time

Synchronization Protocol (FTSP) [97] provide synchronisation for the whole network. LTS and HRTS provide on-demand external time synchronisation while TPSN and FTSP provide continuous synchronisation. All these algorithms are based on the existence of master nodes which propagate the time reference to remaining nodes. Time Diffusion Synchronization (TDP) [154] supports internal and external synchronization. This algorithm elects a set of master nodes which then communicates relevant synchronisation information to the remaining nodes. For external synchronization, these nodes must have access to a global time.

An example of an algorithm that does not use any master nodes and has a flat hierarchy is the Asynchronous Diffusion scheme (AD) [88]. AD supports the internal synchronization of a whole network. The algorithm is very simple: each node periodically sends a broadcast message to its neighbours, who then reply with a message containing their current time. The receiver node averages incoming timestamps and broadcasts the average to the neighbours. Surrounding neighbours adopt this value as their new time. It is assumed that this sequence of operations is atomic.

Time-scale synchronisation (TSS) was introduced by Römer [130] and provides a flexible, on-demand internal synchronization. Node clocks run unsynchronised and employ timestamps that are valid only in the node that generated them. Whenever a time-stamp is communicated to another node, it is transformed to the timescale of the receiver. The necessary information for completing this transformation is communicated during the synchronisation process. TSS is not communication-intensive – only nodes that require synchronisation can exchange such information. However, this does not prohibit global network synchronisation. The TSS implementation presented in [130] achieves implicit offset synchronisation. Another algorithm that uses time-scale transformation is Reference Broadcast Synchronization (RBS) [39]. RBS provides global, internal synchronisation. The basic synchronization primitive is a reference broadcast to a set of client nodes located in the one-hop neighbourhood of a beacon node. The clients then exchange their respective reception times and use linear regression to compute relative offsets and rate differences to each other. Using offset and rate difference, each client can transform a local clock reading to the local timescale of any other client. These one-hop neighbourhoods are then synchronised with each other via nodes that belong to two internally synchronised neighbourhoods. These gateway nodes enable global synchronisation.

6.3 Time synchronisation for computing surfaces

Evidently, there is a wide range of approaches to time synchronisation tailored for various system requirements. In practice, many application developers resort to the use of a common global time service without considering other alternatives. Common global time is often chosen because of its conceptual simplicity, but it requires continuous offset and rate synchronisation. Most importantly, global time is often unnecessary. In [84], Kusý et al define four canonical services desirable from a time synchronisation algorithm, which are also suited for computing surfaces:

- *Event timestamping* – a single, isolated event is observed and timestamped by one or more tiles at possibly different times and forwarded to one or more sink tiles.

- *Data series time-stamping* – one or more observer tiles are periodically sampling some phenomena and producing some corresponding data series, which are collected at a sink tile and analysed. In this type of application, both the temporal ordering of observations and their time length are critical, hence necessitating rate and offset synchronisation.
- *Coordinated action* – otherwise known as pre-facto synchronisation, coordinated action is identical to event timestamping, with the only difference being that the events happen in the future. Distributed applications that have scheduled, coordinated execution rely on this form of time synchronisation.
- *Global time* – although not essential, many existing applications rely on it. This simple and consistent representation of time throughout the network is resource intensive and is often replaced with more efficient alternatives.

For computing surfaces, event timestamping and data series timestamping are necessary to implement interactive games and recognition packages that include time-sensitive gestures. Post-facto synchronisation is required in timed applications such as distributed video playback. Computing surface applications that rely exclusively on the notion of global time without any substitutes are difficult to foresee. In other distributed systems, global time is often used for debugging traces stored in local tiles. Looking at other possible applications, it is hard to find instances that require some form of synchronised time that cannot be delivered by these four canonical services. Subsequently, it is desirable to have a time synchronisation algorithm that supports these four canonical services in computing surfaces.

Due to spatial and cost restrictions, computing surfaces are envisaged to be using clock oscillators of small form-factor. High quality oscillators are often expensive and come with unsuitable form factors. One potential suitable example would be to use internal, silicon based oscillators. Such oscillators may have significant clock drift (e.g. Microchip microcontrollers [102]) and may encounter considerable clock rate variability for small temperature changes. Embedded computing systems that are designed to run on environments that encounter rapid temperature changes need to account for such variability. On the other hand, computing surfaces are assumed to be used in environments with ‘stable’ temperatures – i.e. abrupt changes in temperature are not expected under normal operating conditions. For time periods of 1-10 seconds, small changes in temperature have little impact on the clock rate and therefore clock variability can be ignored. Consequently, for time periods of 1-10 seconds, Equation 6.2 appropriately models the clock sources of computing surfaces.

Although computing surfaces don’t share many network properties with wireless sensor networks, their low communication demands are in common. In addition, limited resources per network tile are also mutual. This makes time synchronisation algorithms for sensor networks appealing for computing surfaces. Nevertheless, most sensor network algorithms exploit the presence of a shared communication medium in a wireless network for broadcasting time information. Some form of multicasting is also used in algorithms that create hierarchical topologies from sensor network nodes. Conversely, a 2D grid topology in computing surfaces is not suitable for broadcasting. In addition, implementations of spanning trees in 2D mesh networks result in asymmetries between spatial and topological locality (neighbouring tiles can be distant in the virtual topology).

On the other hand, TSS algorithms (discussed in Section 6.2.3) provide flexibility and a low communication footprint. The TSS implementation in [130] is intended for ad-hoc networks where message delay cannot be determined in advance. RBS is another time-scale transformation algorithm that also provides compensation for clock drifts [40]. However, its concept and implementation is tightly coupled with broadcasting, which makes it infeasible for computing surfaces.

Computing surfaces differ from sensor networks in synchronisation requirements – while some applications in sensor networks may rely on global synchronisation, computing surface applications are more localised and only require cluster synchronisation. Tiles that execute distributed applications that require time synchronisation would be grouped in a synchronised cluster. Since the size of such clusters is determined by the running application and cannot be foreseen at this point, it is imperative to provide scalable cluster synchronisation. Applications may need to communicate time information with each other. If two such applications belong to different synchronisation clusters, it is necessary to provide some means for intercluster synchronisation.

As previously mentioned, communicating time information is hindered by message delay uncertainties. Very few implementations of distributed systems provide explicit means of measuring message delays. For algorithms that target conventional network interfaces (e.g. classical time synchronisation algorithms and sensor networks), message delays are most often estimated from round-trip measurements. This approach is inefficient as it assumes that the latency is symmetric, which may not be true, especially in the presence of high network traffic. Consequently, round-trip measurements provide only low accuracy with considerable communication and computation overheads. Instead, hardware support for reducing message delay uncertainty provides the foundations for accurate offset and rate synchronisation with very low communication requirements. In comparison to well established infrastructures of distributed systems, computing surfaces can adopt a hardware solution much more easily.

In summary, a time synchronisation algorithm for computing surfaces needs to provide scalable intra and inter-cluster synchronisation. The time synchronisation service proposed in this Chapter follows these guidelines. It will provide support for the four canonical services as described above. In order to reduce communication demands, accurate offset and rate synchronisation is achieved with hardware support for low message delay uncertainty.

6.4 Minimising message delay uncertainty

Message delay uncertainty arises from resource contention. Several resources such as CPU time, cache lines, network switches, buffers, channels, etc. are needed at several stages of message preparation and delivery. While resource arbitration may be deterministic, the number of tasks requiring any given resources at any given time is not deterministic, resulting in cumulative execution non-determinism. Consider a software implementation of a timestamp transmission in a computing surface tile. Every timestamp communication goes through:

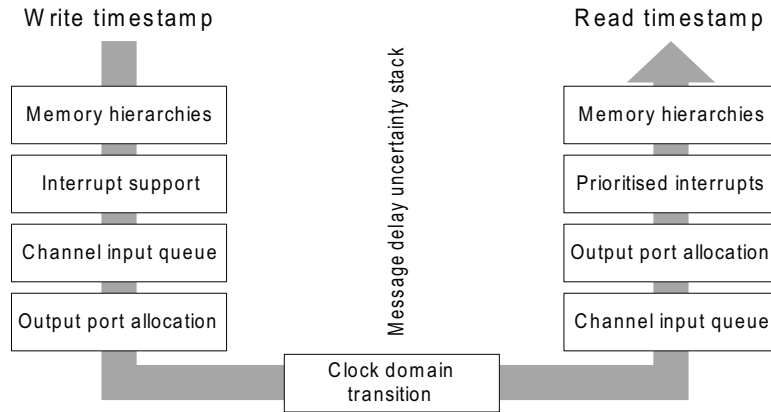


Figure 6.1: Contributors to message delay uncertainty in software implementations of timestamp transmission

- *Software preparation* – interrupts and cache misses introduce message delay uncertainty during timestamp calculation and corresponding memory transfers to the network interface.
- *Network preparation* – the induced delay while allocating the necessary resources of the network router (queues, switches and physical channel) depends on the network traffic load, hence is non-deterministic.
- *Propagation* – transition from the clock domain that drives synchronous circuits of the current router to the clock domain driving synchronous circuits of the downstream router induces some message delay uncertainty.
- *Network reception* – as in the network preparation phase, contention for necessary network resources upon arrival causes further uncertainties.
- *IRQ servicing* – depending on the implementation details of the IRQ controller, the interrupt servicing may not be immediate (e.g. in prioritised interrupts). In addition, memory hierarchies can once again induce further uncertainties due to potential cache misses.

These contributors visually can be perceived as forming two stacks – a transmission stack and receiving stack of delay uncertainties as presented in Figure 6.1.

One way to reduce message delay uncertainty would be to provide exclusive resources to facilitate communication of time synchronisation packets. This would result in significant additional hardware that would be poorly utilised. Alternatively, message delay uncertainty can be minimised by delaying the timestamping itself to the very last moment, similar to [64]. While timestamp packets are prepared by software and processed by the network interface, the actual value of the current time is written only when the physical transmission channel is allocated for this packet. In order to resolve the uncertainties at the receiving end, the packet arrival is recorded before it requests any shareable resource that could lead to contention. This approach of delayed writing and early reading of timestamps requires appropriate modifications to the network interface and is not available in most widespread network interfaces. As a result, the lack of this feature has

reduced the adoption of hardware-based time synchronisation mechanisms in more conventional architectures and wireless sensor networks. This is not an issue for computing surfaces, so the desirable hardware based approach is chosen. It is clear from Figure 6.1 that delayed timestamping and early reading of timestamps deals efficiently with all tile-based uncertainties. Clock domain transition remains undetermined, but its contribution is very small in comparison with tile-based uncertainties.

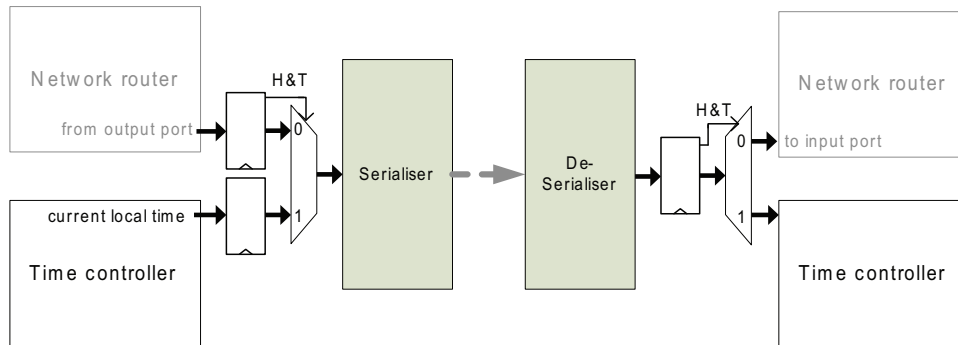


Figure 6.2: Reading and writing of TSPs in the tile network interface

Time Synchronisation Packets

Neighbouring tiles communicate timestamps to each other to reason about their relative differences in local times. These timestamps enable pairwise synchronisation of immediate neighbours.

In order to make minimal changes to the network router architecture without compromising on the induced delay uncertainty, timestamp writing and reading are managed at the interface between the network router and serialiser/de-serialiser circuits. During flit transmission, the time taken from the point when the serialiser receives the outgoing data up to the moment when the flit is placed in the input queue of the downstream router incurs non-determinism caused by resynchronisation. At the downstream router input port, instead of storing the delay-sensitive timestamped packet in the router input queue, it is diverted to a dedicated module for time synchronisation. This module stores the value of the received timestamp and the current local time to a register for later processing by the CPU.

This approach assumes that packets carrying timestamps can be easily distinguished from other data packets. One method to achieve this early distinction is the introduction of an additional control bit. The main disadvantage of this method results from increased bandwidth overheads. The approach taken in this work is to introduce a special packet – the Time Synchronisation Packet (TSP). The TSP is a distinctive packet that carries local time readings. The TSP consists of only one flit - in fact, this is what distinguishes the TSP from any other packet. In the ISLAY router design and many other packet-based architectures, flits contain two control bits to indicate if the flit is head, body or tail. A one-flit packet has both of these control bits set to 1 and is not essential for ordinary data communication. The value of these control bits can be read at the de-serialiser as soon as the complete flit is received, thereby making the detection of one-flit packets

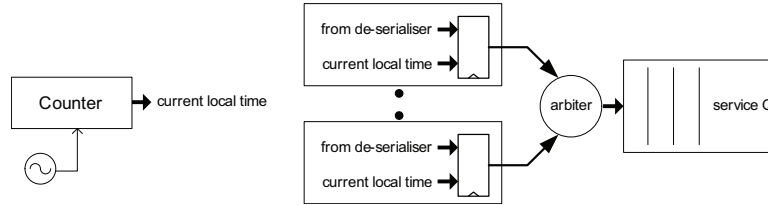


Figure 6.3: Time controller structure

straightforward. Subsequently, the ease of distinguishing one-flit packets makes them suitable for their role as TSP packets.

If the serialiser in the output port encounters a flit with head and tail, instead of transmitting the actual content of the flit, it begins transmission of the current reading of the local time register. On the other end of the serial communication channel, once the complete flit is received by the de-serialiser circuit, the control bits are tested for a TSP packet. Upon receiving a TSP, the de-serialiser will forward its value to the time synchronisation module instead of the input port.

The time synchronisation module receives TSPs from router de-serialisers and prepares them to be processed by the time synchronisation service. Its central role is to make TSP processing delay-insensitive. The Time controller module does this by timestamping the receipt of the TSP. Subsequently, this pair of timestamps, i.e. the time information carried in the TSP and the corresponding local time that marks the receipt of the TSP, are queued for processing. Local timestamping enables the processing of the TSP at any time in the near future. The receipt of TSPs from two neighbouring tiles is executed independently from each other, which raises the possibility of two TSPs arriving in the same clock cycle. In order to prevent any resource contention that would result in delay uncertainty, each TSP source (i.e. de-serialiser) can timestamp its incoming TSPs before queuing to the main service queue. The data from this queue is revealed to the CPU through its memory-map. Finally, the time synchronisation module raises an interrupt request for servicing received TSPs.

With the removal of delay uncertainties introduced by resource contention, the TSP message delay uncertainty Δ_{md} is limited to clock domain transitions. In synchronous transmission, non-deterministic phase differences between transmitters and receivers generate this part of delay uncertainty. For a pair of transmitter-receivers with a transmission rate frequency Φ , the clock phase differences are limited to one clock period, and therefore:

$$\Delta_{md} < \frac{1}{\Phi} \quad (6.11)$$

System timers often report time in microsecond level granularity. In most digital systems, the frequency of the transmitter-receiver pair (Φ) is likely to be higher than the frequency of the system timer ϕ . Under these conditions, Δ_{md} is smaller than the time period of one increment of local system time ($1/\phi$):

$$\Delta_{md} < \frac{1}{\phi} \quad (6.12)$$

thus making Δ_{md} indistinguishable.

6.5 Offset synchronisation of immediate neighbours

As previously mentioned, TSPs are exchanged for synchronising immediate neighbours. Figure 6.4 presents the timeline of the process of this pairwise synchronisation. The local time of tile n_1 starts ticking at real time t_{01} , while for tile n_2 , this process starts at t_{02} . It is assumed that local clocks start ticking at different times. This is a necessary assumption, since there are many factors that may impact the time when local clocks are started (e.g. power-on reset, system restarts, etc.).

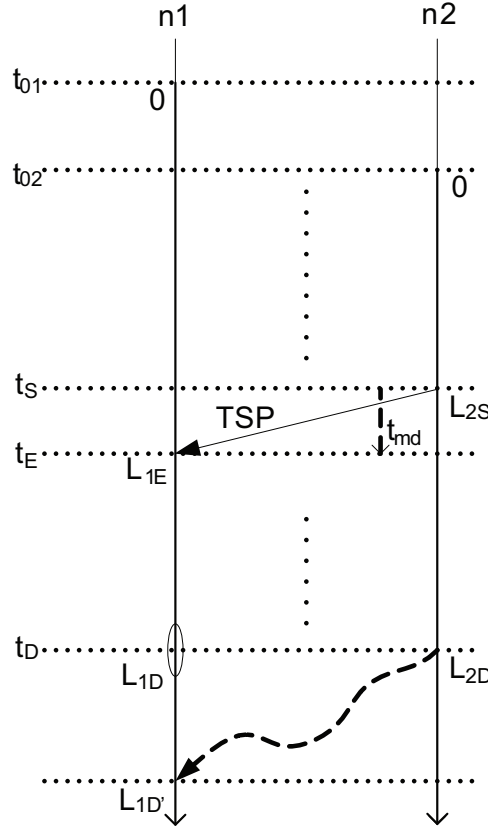


Figure 6.4: Transforming neighbour time to local time

Upon request, tile n_2 starts with transmission of a TSP packet to tile n_1 at real time t_S . Tile n_1 becomes aware of the incoming TSP at real time t_E . Tiles assume that message delay for transmitting flits in the physical layer is md cycles - while the actual transmission time $t_E - t_S$ is denoted with t_{md} . The relationship between the message delay and the number of cycles can be expressed as follows

$$\left| t_{md} - \frac{md}{\phi(1 + \delta)} \right| < \Delta_{md} < \frac{1}{\phi(1 + \delta_{max})} \quad (6.13)$$

Upon arrival of the TSP, tile n_1 registers its time L_{1E} . The value L_{2S} is conveyed in the TSP and is paired with local time L_{1E} and stored in the service queue of the Time controller module. Once ready to be serviced by the CPU, the pair (L_{2S}, L_{1E}) is used to calculate the offset O_{12} between local times:

$$O_{12} = L_{2S} - (L_{1E} - md) \quad (6.14)$$

This calculation can be used for offset synchronisation and forms the basis of our approach to time synchronisation in computing surfaces – any local time information received from tile n_2 can be transformed to local time of tile n_1 . This time transformation reference derived from the TSP enables other time information to be sent in ordinary data packets.

While the calculated offset value remains fixed, the actual difference between the clock readings of the neighbour clock and the local clock changes over time due to relative differences of clock drift. In fact, equation 6.14 is an approximation of the function given below for $t = t_S$:

$$O_{12}(t) = L_2(t) - L_1(t) \quad (6.15)$$

The absolute error of the offset calculation is:

$$\begin{aligned} \Delta O_{12} &= |O_{12}(t_S) - O_{12}| \\ &= |(L_{1E} - md) - L_1(t_S)| \\ &= \left| \left[(t_E - \frac{md}{(1+\delta_1)\phi} - t_{01})(1 + \delta_1)\phi \right] - \left[(t_S - t_{01})(1 + \delta_1)\phi \right] \right| \\ (from\ 6.7) &= \left| (t_E - \frac{md}{(1+\delta_1)\phi} - t_{01})(1 + \delta_1)\phi - \left[(t_S - t_{01})(1 + \delta_1)\phi \right] \right| \\ &= \left| \left[(t_E - t_S - \frac{md}{(1+\delta_1)\phi})(1 + \delta_1)\phi \right] \right| \\ &= \left| \left[(t_{md} - \frac{md}{(1+\delta_1)\phi})(1 + \delta_1)\phi \right] \right| \end{aligned} \quad (6.16)$$

It is clear that at real time t_S , the offset absolute error is determined by the upper bound of message delay uncertainty. For timestep increments that are larger than message delay uncertainty (relation 6.13), the absolute error is zero:

$$\Delta O_{12} = \left[\left(\frac{md}{(1 + \delta_1)\phi} - t_{md} \right) (1 + \delta_1)\phi \right] = 0 \quad (6.17)$$

Consider an event that occurs at real time t_D , registered in tile n_2 with the timestamp value L_{2D} . Tile n_2 transmits this timestamp to tile n_1 using an ordinary data packet. Tile n_1 receives this packet with some uncertain delay at time L_{1D} without hindering on the transformation accuracy. Using the offset O_{12} calculated at time t_S , tile n_1 will transform n_2 's local time L_{2D} , with:

$$L_{1D} = L_{2D} - O_{12} \quad (6.18)$$

This calculation is an approximation of the actual value of $L_1(t_D)$. The difference is characterised as follows:

$$\begin{aligned}
L_1(t_D) - L_{1D} &= (L_2(t_D) - O_{12}(t_D)) - (L_{2D} - O_{12}) \\
&= (O_{12} - O_{12}(t_S)) - (O_{12}(t_D) - O_{12}(t_S)) \\
(\text{from 6.16}) &= O_{12}(t_S) - O_{12}(t_D) \\
&= (L_2(t_S) - L_1(t_S)) - (L_2(t_D) - L_1(t_D)) \\
&= (L_1(t_D) - L_1(t_S)) - (L_2(t_D) - L_2(t_S)) \\
&= (\lfloor (t_D - t_{01})(1 + \delta_1)\phi \rfloor - \lfloor (t_S - t_{01})(1 + \delta_1)\phi \rfloor) - \\
&\quad (\lfloor (t_D - t_{02})(1 + \delta_2)\phi \rfloor - \lfloor (t_S - t_{02})(1 + \delta_2)\phi \rfloor) \\
(\text{from 6.10}) &= (\lfloor (t_D - t_{01})(1 + \delta_1)\phi - (t_S - t_{01})(1 + \delta_1)\phi \rfloor - i) - \\
&\quad ((t_D - t_{02})(1 + \delta_2)\phi - (t_S - t_{02})(1 + \delta_2)\phi) \\
&= \lfloor (t_D - t_S)(1 + \delta_1)\phi \rfloor - i - (t_D - t_S)(1 + \delta_2)\phi \\
&= \lfloor (t_D - t_S)(\delta_2 - \delta_1)\phi \rfloor - i
\end{aligned} \tag{6.19}$$

where $t_D - t_S$, is the elapsed time since the last TSP is sent and $i \in \{0, 1\}$. In brief,

$$\Delta L_{1D} = |L_1(t_D) - L_{1D}| \leq |\lfloor (t_D - t_S)(\delta_1 - \delta_2)\phi \rfloor| + 1 < \Delta_{max} \tag{6.20}$$

Clearly, clock skew is linearly affected by the elapsed time since the last TSP is sent. If the skew becomes too large, another TSP needs to be communicated for updating the offset value. The TSP retransmission can be repeated in order to maintain clock skew in tolerable limits. Given a skew budget of $\Delta_{max} \geq 2$, it can be maintained by tile n_1 for tile n_2 for a period of time

$$t_D - t_S < \frac{\Delta_{max} - 1}{(\delta_1 - \delta_2)\phi} \tag{6.21}$$

Although tiles cannot know the precise clock drifts of their individual clock sources, information on drift bounds can be provided from the manufacturer. Drift bounds enable a conservative calculation of the number of clock cycles p that the tile will maintain a given clock skew budget Δ_{max} :

$$p = \frac{(\Delta_{max} - 1)(1 + \delta_{min})}{(\delta_{max} - \delta_{min})} \tag{6.22}$$

6.6 Controlling the impact of clock drift

In order to maintain the given clock skew, tile n_2 periodically transmits TSP packets every p cycles. The number of TSP communications can be further reduced if relative differences of clock drifts are taken into account.

In order to use this equation, the relative clock differences need to be estimated. This drift difference can be estimated by measuring changes to offset values in two consecutive TSP synchronisations. Let t_{S1} and t_{S2} be the moments in time when tile n_2 sends TSP packets to tile n_1 . Calculated offsets upon receipt of TSP packets in tile n_1 have values of $O_{12}(t_{S1})$ and $O_{12}(t_{S2})$ (relation 6.16).

We denote with δO the difference between calculated offset values:

$$\delta O_{12} = O_{12}(t_{S1}) - O_{12}(t_{S2}) \tag{6.23}$$

The value of δO is also expressed with the following equations:

$$\begin{aligned}
\delta O_{12} &= (L_2(t_{S1}) - L_1(t_{S1})) - (L_2(t_{S2}) - L_1(t_{S2})) \\
&= (L_1(t_{S2}) - L_1(t_{S1})) - (L_2(t_{S2}) - L_2(t_{S1})) \\
&= (\lfloor (t_{S2} - t_{01})(1 + \delta_1)\phi \rfloor - \lfloor (t_{S1} - t_{01})(1 + \delta_1)\phi \rfloor) - (t_{S2} - t_{S1})(1 + \delta_2)\phi \\
&= \lfloor (t_{S2} - t_{S1})(1 + \delta_1)\phi \rfloor - i - (t_{S2} - t_{S1})(1 + \delta_2)\phi \\
(\text{from 6.6}) &= \lfloor (t_{S2} - t_{S1})(\delta_1 - \delta_2)\phi \rfloor - i \\
&= \lfloor (t_{S2} - t_{S1})(1 + \delta_2)\phi \frac{\delta_1 - \delta_2}{1 + \delta_2} \rfloor - i \\
&= \lfloor p \frac{\delta_1 - \delta_2}{1 + \delta_2} \rfloor - i
\end{aligned} \tag{6.24}$$

Consequently, δO can also be expressed with the following inequality:

$$p \frac{\delta_1 - \delta_2}{1 + \delta_2} - 1 \leq \delta O_{12} < p \frac{\delta_1 - \delta_2}{1 + \delta_2} + 1 \tag{6.25}$$

As a result, drift differences can be approximated with $\delta O_{12}/p$ with the following absolute error:

$$\left| \frac{\delta_1 - \delta_2}{1 + \delta_2} - \frac{\delta O_{12}}{p} \right| \leq \frac{1}{p} \tag{6.26}$$

Consequently, large p enables accurate estimation of relative drift differences. On the other hand, clock skew growth characterised in relation 6.20 can be also expressed as:

$$L_1(t_D) - L_{1D} = \lfloor (L_{2D} - L_{2S}) \frac{\delta_1 - \delta_2}{1 + \delta_2} \rfloor - i \tag{6.27}$$

With the approximation of clock drift presented in relation 6.26, clock skew can be approximated as follows:

$$\lfloor \delta O_{12} \frac{L_{2D} - L_{2S}}{p} \rfloor - 1 \leq L_1(t_D) - L_{1D} \leq \lfloor \delta O_{12} \frac{L_{2D} - L_{2S}}{p} \rfloor \tag{6.28}$$

Coming back to the initial, uncorrected offset transformation, we define a new clock transformation with drift correction:

$$L'_{1D} = L_{2D} - O_{12} + \lfloor \delta O_{12} \frac{L_{2D} - L_{2S}}{p} \rfloor \tag{6.29}$$

In the rest of this document, when talking about transformations, we mean transformations with drift correction and denote it with L_{1D} (instead of L'_{1D}). The error in this calculation can be extracted from 6.26 and is expressed with the following inequality:

$$|L_{1D} - L_1(t_D)| \leq \frac{L_{2D} - L_{2S}}{p} + 2 \tag{6.30}$$

For the treatment in this work, the upper bound of clock skew presented above will be denoted with $\epsilon_2(t_D)$. Consequently, for a given skew budget Δ_{max} , the error can be maintained P number of cycles after the last TSP, where

$$P \leq p(\Delta_{max} - 2) \tag{6.31}$$

This relation between the resynchronisation period with clock correction P and the corresponding period without clock correction p , shows the potential reduction in communication needs when drift correction is applied. However, for very large values of P , drift variability can cause this method to fail. In order to prevent such failures, a timeout T_{max} period is defined to prevent the resynchronisation period P from taking very large values:

$$P = \min(T_{max}, p(\Delta_{max} - 2)) \quad (6.32)$$

The value of T_{max} depends on the ability of the clock source to maintain drift variability in negligible levels. It is therefore determined by the quality of the clock source and the surrounding environmental factors. Unstable environments or low quality clock sources require shorter T_{max} while stable environments and high quality clock sources tolerate longer T_{max} .

6.7 Chain synchronisation

The method presented in the previous two sections provides time synchronisation only within immediate neighbours. A comprehensive algorithm requires synchronisation among any pair of tiles. The method for synchronising immediate neighbours can be used implicitly for synchronising distant tiles.

This section discusses the synchronisation of any given pair of tiles. Consider the synchronisation of tile n_1 with tile n_k , having intermediate tiles n_2, n_3, \dots, n_{k-1} in between. If tile n_i is synchronised with tile n_{i+1} (for $1 \leq i < k$), then the local time of tile n_{i+1} is transformed onto the local time of tile n_i . The same applies for tiles n_i and n_{i-1} . Consequently, the local time of tile n_{i+1} can be transformed onto the local time of tile n_{i-1} by one intermediate transformation in the local time of tile n_i . In the same way, the local time of tile n_k is transformed onto the local time of tile n_1 by intermediate transformations to local times of tiles n_2, n_3, \dots, n_{k-1} .

For any time information L_{kD} , tiles would perform the following transformations:

$$\begin{aligned} n_k & : L_{kD} \\ n_{k-1} & : L_{(k-1)D} = L_{kD} - O_{(k-1)k} + \lfloor \delta O_{(k-1)k} \frac{L_{kD} - L_{kS}}{p} \rfloor \\ & \dots \\ n_3 & : L_{3D} = L_{4D} - O_{34} + \lfloor \delta O_{34} \frac{L_{4D} - L_{4S}}{p} \rfloor \\ n_2 & : L_{2D} = L_{3D} - O_{23} + \lfloor \delta O_{23} \frac{L_{3D} - L_{3S}}{p} \rfloor \\ n_1 & : L_{1D} = L_{2D} - O_{12} + \lfloor \delta O_{12} \frac{L_{2D} - L_{2S}}{p} \rfloor \end{aligned} \quad (6.33)$$



Figure 6.5: Offset synchronisation of distant nodes

Figure 6.5 illustrates the flow of timestamps and synchronisation packets in a chain of tiles in order to transform L_{kD} to a local time of any other tile. This approach forms the basis of transforming local times of distant tiles as suggested in [130]. However, it requires every timestamp to be transformed in all the intermediate tiles before being used by the final tile. This method incurs computational overhead in every intermediate tile. In addition, the end-to-end timestamp transmission time increases drastically due to necessary intermediate stops. While in [130], such intermediate stops may be necessary, the same does not apply for computing surfaces. These overheads can be avoided if a cumulative offset for tile n_k is calculated in relation to tile n_1 . If the information required for transformations given in Equation 6.33 is available, it is possible to derive a direct transformation of local time of tile n_k in tile n_1 :

$$L_{1D} = L_{kD} - \sum_{i=1}^{k-1} O_{i(i+1)} + \sum_{i=1}^{k-1} \left[\delta O_{i(i+1)} \frac{L_{(i+1)D} - L_{(i+1)S}}{p} \right] \quad (6.34)$$

The transformation given in 6.34 allows tile n_1 to directly transform local time L_{kD} of tile n_k into its local time. In addition, information used by this transformation is sufficient to transform the local time of any intermediate tiles n_2, \dots, n_{k-1} into the local time of n_1 . In a very similar approach, tile n_i can be synchronised with tiles $n_{i+1}, n_{i+2}, \dots, n_k$, where $1 < i < k$. In order to achieve this, each tile n_i requires offset information $O_{i(i+1)}, \dots, O_{(k-1)k}$ and $\delta O_{i(i+1)}, \dots, \delta O_{(k-1)k}$. Furthermore, transformation information of tile n_{i+1} can be used by tile n_i to construct its own information base for synchronising with tiles n_{i+1}, \dots, n_k . This approach enables tile n_i to transform the local time of any tile n_j , as long as $j > i$. From equations 6.29 and 6.18, we can deduce that $O_{ij} = -O_{ji}$ and $\delta O_{ij} = -\delta O_{ji}$, therefore the same offset information could be used for time transformations on the opposite direction. Synchronisation of all the tiles that belong to a linear chain will be referred as *chain synchronisation*.

A suitable implementation for chain synchronisation is given in Algorithm 5. It utilises a data structure named `ChainSynch` which can have the following implementation:

```
struct ChainSynch
{
    bool isReturning;
    int chainId, tileId, chainLength;
    OffsetData[] chainOffsetData;
    Directions[] chainPathDetails;
}
```

This algorithm is executed upon receipt of the chain synchronisation message in all tiles except the setup tile n_0 . The setup tile triggers the process of chain synchronisation and is usually located at one end of the chain. It generates chain details such as a unique chain identifier, length and the chain path in an initial `ChainSynch` structure. The path details of the chain are represented as a one-dimensional array (`ChainSynch.chainPathDetails`) of simple directions (N, E, S W). These directions are used by the tile to identify the next (immediate neighbour) tile in the chain. After initialisation, the setup tile looks up the first element of this array to identify the next tile in the chain and sends the data structure to it. Upon receipt, the next tile will execute the presented algorithm. During

execution, the tile identifies its position in the chain (by setting its chain *id*). It updates its chain offset record with data carried by the incoming chain synchronisation message (`ChainSynch.chainOffsetData`). Offset information $O_{i(i+1)}$ and $\delta O_{i(i+1)}$ is stored in a data structure called `OffsetData`. The `ChainSynch` data structure consists of an array of `OffsetData` holding synchronisation information for every pair of immediate neighbours. Furthermore, the algorithm identifies the next tile where this execution should take place. Finally, it initiates offset synchronisation with this tile and forwards the updated offset and chain information to it. This is repeated in all the tiles up to the end of the chain ($tileId = chainLength - 1$). In this tile, the synchronisation information flow changes direction – the execution of the algorithm visits each tile again and finally returns to the setup tile. A utility boolean variable `ChainSynch.isReturning` facilitates the return of the data structure back to the initiator node after visiting all the tiles of the chain. The utility function *reverse* returns the reverse of basic directions (i.e. $N \rightleftharpoons S$ and $E \rightleftharpoons W$) and is used for determining the returning path in the chain.

There is one potential drawback associated with the transformation given in 6.34 – tile n_1 needs to be notified of any individual updates of values $O_{i(i+1)}$ and $\delta O_{i(i+1)}$ performed by the corresponding tiles. Orchestration of offset updates would be desirable to simplify the notification mechanisms. Since a TSP packet is sent before each synchronisation message (`ChainSynch`), tiles will have freshly calculated offsets. This prior communication of TSPs provides the orchestration of offset updates.

Certainly, this time transformation of distant tiles will incur some calculation error. Within a synchronised chain, it is of interest to determine the error of transforming the local time of tile n_j in tile n_i . Consider an event registered by tile n_j at real time t_D . This tile communicates local time L_{jD} to tile n_i . Based on the characterisation of clock skew on immediate neighbours (inequality 6.30), the clock skew incurred while transforming L_{jD} into local time of tile n_i can be expressed as follows

$$\begin{aligned}
|L_i(t_D) - L_{iD}| &\leq |L_{i+1}(t_D) - L_{(i+1)D}| + \epsilon_{i+1}(t_D) \\
&\leq |L_{i+2}(t_D) - L_{(i+2)D}| + \epsilon_{i+1}(t_D) + \epsilon_{i+2}(t_D) \\
&\quad \dots \\
&\leq |L_{j-1}(t_D) - L_{(j-1)D}| + \sum_{k=1}^{j-1-i} \epsilon_{i+k}(t_D) \\
&\leq |L_j(t_D) - L_{(j)D}| + \sum_{k=1}^{j-i} \epsilon_{i+k}(t_D) \\
&\leq (j-i)\epsilon(t_D)
\end{aligned} \tag{6.35}$$

where $\epsilon(t_D) = \max(\epsilon_i(t_D))$. It is clear from this relation that in the worst case, the error grows linearly with the hop-distance between tiles. This inherent property of clock synchronisation algorithms is similarly characterised in many theoretical treatments such as [94]. For homogeneous networks, the accuracy of time synchronisation is dependent on the synchronisation accuracy of immediate neighbours and topological properties of the network. As a result, when referring to the accuracy of time synchronisation algorithms, it is logical to specify the one-hop distance synchronisation accuracy.

The dependency of synchronisation accuracy on hop-distance introduces a design restriction on chain synchronisation. In order to avoid diminishing accuracy, synchronised chains should be convex – i.e. for any pair of tiles (n_i, n_j) that belongs to a synchronised chain, all intermediate tiles that belong to one of the several possible minimal paths connecting n_i to n_j (and vice versa) should also belong to the same synchronised chain.

Algorithm 5: A distributed algorithm for chain synchronisation

Data: localOffset: information from local offset synchronisations
Data: currentChain: stores current chain's synchronisation information
Data: tileChainInfo: tile's list of all chain memberships and information
Data: directionToNextTile: takes values N, E, S or W; extracted from chainPathDetails
input : ChainSynch inMsg
output: ChainSynch outMsg
 outMsg \leftarrow inMsg;
if (*inMsg.isReturning*) **then**
 currentChain \leftarrow tileChainInfo.Lookup(inMsg.chainId);
else
 currentChain \leftarrow inMsg;
end
if (*currentChain.tileId* \neq 0) **then**
 if (*inMsg.isReturning*) **then**
 currentChain.chainOffsetData \leftarrow inMsg.chainOffsetData;
 else
 tileChainInfo.Add(currentChain);
 end
 outMsg.isReturning \leftarrow inMsg.isReturning \vee (inMsg.tileId = inMsg.chainLength - 1);
 if (*outMsg.isReturning*) **then**
 outMsg.tileId \leftarrow currentChain.tileId - 1;
 directionToNextTile \leftarrow reverse(currentChain.chainPathDetails[currentChain.tileId]);
 else
 outMsg.tileId \leftarrow currentChain.tileId + 1;
 directionToNextTile \leftarrow currentChain.chainPathDetails[currentChain.tileId];
 end
 if (*inMsg.isReturning* = FALSE) **then**
 currentChain.chainOffsetData[currentChain.tileId] \leftarrow localOffset;
 send(TSP, directionToNextTile);
 end
 outMsg.chainOffsetData \leftarrow currentChain.chainOffsetData;
 send(outMsg, directionToNextTile);
end

Chain synchronisation makes possible another compelling feature - any pair of tiles (n_i, n_j) within a chain can determine the round-trip clock skew at a given moment in time. In order to do this, tile n_i sends its current time L_i to tile n_j and requests the return of local time L_j that corresponding to L_i . Upon return, tile n_i can transform timestamp L_j into local time L'_i . The clock skew is calculated as $\Delta L_i = |L_i - L'_i|$. This feature is a simple and powerful method for monitoring the performance of the time synchronisation algorithm.

6.8 Cluster synchronisation

Chain synchronisation forms the basis for cluster synchronisation in computing surfaces. For the purposes of this work, a cluster is defined as a set of tiles that form a connected graph [14]. Synchronised clusters can be built with synchronised chains. As in the case of synchronised chains, to avoid undesired inaccuracies, clusters also need to be convex – the intermediate tiles contributing to some shortest path between any pair of cluster tiles must belong to the cluster. Every cluster tile will be part of at least one synchronised chain. Tiles that are part of more than one synchronised chain are used to transform time information from one synchronised chain to another – i.e. they serve as inter-chain synchronisation gateways.

Cluster synchronisation with the use of synchronised chains can be described in more detail with the example of a rectangular cluster presented in Figure 6.6. Two arbitrary tiles, (in this case $n_{0,1}$ and $n_{i-1,j-1}$) need to be synchronised but they don't share a synchronisation chain. These two tiles belong to two synchronisation chains that intersect at tile $n_{0,j-1}$. Tile $n_{i-1,j-1}$ sends its time information $L_{i-1,j-1}$ to tile $n_{0,j-1}$. After transforming this timestamp into its local time $L_{0,j-1}$, tile $n_{0,j-1}$ forwards it to tile $n_{0,1}$, which can transform $L_{0,j-1}$ into $L_{0,1}$.

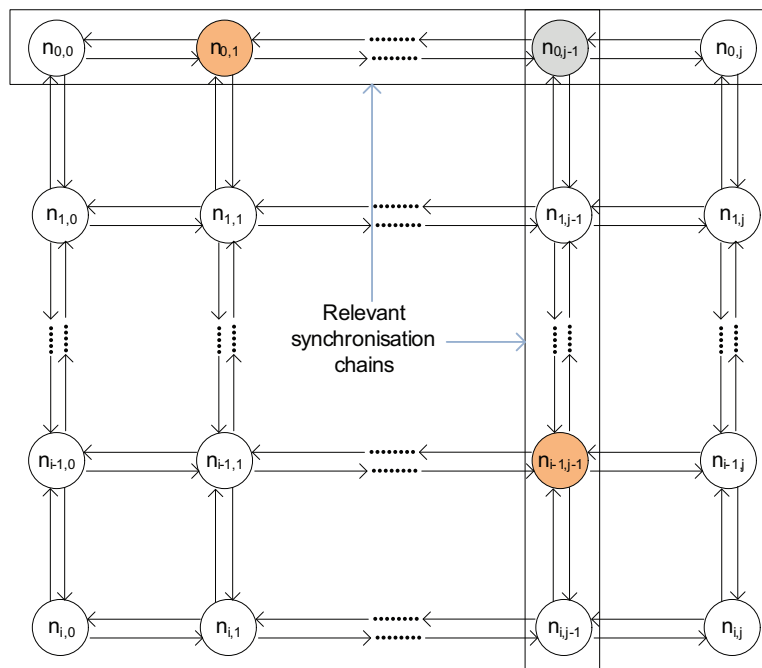


Figure 6.6: Cluster synchronisation

An application that requires a synchronised cluster elects a master tile, whose role is to initiate the synchronisation of the cluster. Firstly, it determines if the given cluster is convex. If the cluster is not convex, it selects a minimal convex superset. Secondly, a minimal list of synchronisation chains that span the cluster is selected. Then, the master tile selects a leader for each chain at one of the ends of the chain. Chain leaders receive chain synchronisation requests from the master tile, and begin executing chain synchronisation. Upon completion, each chain leader sends a confirmation back to the master tile.

The receipt of all confirmations implies that time synchronisation is established within the cluster.

It would be possible to avoid inter-chain synchronisation gateways altogether by ensuring that there is a synchronised chain over a minimal path for every pair of tiles that requires synchronisation. For large clusters, this may result in significant memory requirements for storing offset information. Inter-chain synchronisation provides a tradeoff between local memory requirements on the one hand and speed of timestamp delivery and computational overheads on the other. Also, non-minimal lists of synchronised chains would incur communication overheads as more synchronised chains would have to be established.

6.9 Pre-facto synchronisation

The time-scale synchronisation approach presented so far provides implicit synchronisation with rate correction. This approach can provide only limited support for pre-facto synchronisation. Consider a synchronised cluster with a dedicated tile n_i for orchestrating future events. This tile broadcasts a future local time L_{iD} to all tiles that belong to the cluster. Each tile receives this timestamp and transforms it into local time L_D . The corresponding event is triggered when the reading of the local clock is equal to L_D .

However, in some cases it is necessary to periodically execute future events, with time period T . A video playback application, distributed among several tiles would execute frame transitions in such future events where T would be the frame length. If the time period T is communicated simply as a number of time steps, clock drifts would map this time period onto different real time lengths. When playing a 25fps video source, two tiles with relative drift differences of 100PPM would lose frame synchronisation after 400 seconds.

Pre-facto synchronisation requires rate synchronisation in addition to offset synchronisation. Offset synchronisation presented in Section 6.5 is an implicit form of offset synchronisation. From the same perspective, the drift correction mechanisms presented in Section 6.6 provide implicit rate synchronisation. Tiles are aware of their relative drift differences, and therefore their relative clock rate differences. Similarly to communicating future timestamps by transforming them from one time-domain to another, implicit rate synchronisation can be achieved by transforming time periods T between time domains.

For a given pair of synchronised tiles, the remote time length T_2 can be transformed into local time length T_1 with the following equation that derives directly from 6.29:

$$T_1 = T_2 + \lfloor \delta O_{12} \frac{T_2}{p} \rfloor \quad (6.36)$$

This calculation incurs truncation errors, which become significant when calculating multiples of T_1 . In such a scenario, when scheduling the next n occurrences of a periodical event with time period corresponding to T_2 , the following calculation is more appropriate than the straightforward calculation nT_1 :

$$next(n, T_2) = nT_2 + \lfloor \delta O_{12} \frac{nT_2}{p} \rfloor \quad (6.37)$$

6.10 Protocol synopsis

A middleware implementation of the presented time synchronisation algorithm can be realised with the following functions:

```
void sendTSP(Direction neighbour)
```

Sends a TSP packet to the immediate neighbour, specified with its direction from the host.

```
Time getTime()
```

Reports the current value of the local time by reading the local time register in the Time Synchronisation module.

```
Time transformTime(Time remoteTime, Node source)
```

Calculates the corresponding local time from the value `remoteTime`, that originates from node `source`.

```
int transformTimeLength(int remoteTimeLength, Node source)
```

Calculates the local time length that corresponds to the time length with value `remoteTimeLength`, that originates from node `source`

```
void initSynchronise(Cluster synch)
```

Initiates time synchronisation in a cluster of nodes, by decomposing the cluster into a set of chains, and elects leaders for chain synchronisations.

```
void initSynchronise(Chain synch)
```

Initiates time synchronisation in a chain of nodes based on the details presented in Section 6.7.

```
void synchronise(Msg timeSynch)
```

Triggered by the receipt of time synchronisation message, it executes an implementation of the algorithm given in Section 6.7.

Coming back to the canonical services presented in Section 6.3, the presented time synchronisation algorithm provides support for all four services. Event timestamping is achieved by locally timestamping events and sending them to the sink tile. Upon receipt, the sink tile transforms these timestamps into the local time of the sink tile. On the other hand, transformation of time intervals becomes necessary in data series timestamping (Section 6.3). In this case, the sink tile will transform the timestamps and time intervals into its own local time. The pre-facto synchronisation method presented in Section 6.9 is sufficient for coordinated action services. A virtual global time in the whole span of a computing surface is not envisaged to be useful. A cluster that would span the whole area could provide such a service if necessary. As a result, it is more logical to provide a virtual cluster time. This service is achieved with a special future event that is executed simultaneously in every tile that requires all cluster tiles to set their local time to a common cluster time. In this form, virtual global time is not provided with rate

synchronisation and therefore needs to be refreshed every p clock cycles. Implicit rate synchronisation could be used to reduce the number of updates.

The time synchronisation algorithm presented in this chapter provides internal synchronisation. It is possible to provide external synchronisation for tiles that are internally synchronised. One such approach of providing UTC is presented in [1]. This method assumes that a dedicated tile in a synchronised cluster is also synchronised to UTC. With the help of internal time synchronisation, this tile propagates the details of UTC to the cluster. This approach could be implemented relatively easily in computing surfaces, should it be required.

6.11 Evaluation

The performance of the presented time synchronisation protocol is evaluated in the computing surface prototype presented in Chapter 3. The Time controller module is implemented and integrated in the network subsystem of the FPGA setup. The frequency of the system is 50MHz, which sets the maximum resolution of local time reporting.

Although the system runs at 50MHz, the data is transmitted over the serial link only every third clock cycle resulting in a bit rate up to 16.7Mbps. To measure message delay uncertainty, the serialiser outputs a pulse signal at the beginning of a TSP transmission. At the other end, the de-serialiser outputs a similar pulse signal upon registering (timestamping) the receipt of a TSP. The time difference between the pulses of the serialiser on a transmitter tile and the de-serialiser on the receiving tile represents the message delay. Measurements for a pair of tiles running in separate FPGAs are completed with an oscilloscope and saved onto a data file. Figure 6.7 illustrates the message delay uncertainty for a measurement of 10,000 TSP transmissions. The distribution of transmission delays is approximately normal and is similar to other measurements of message delay uncertainty in the literature (e.g. see [65] and [40]).

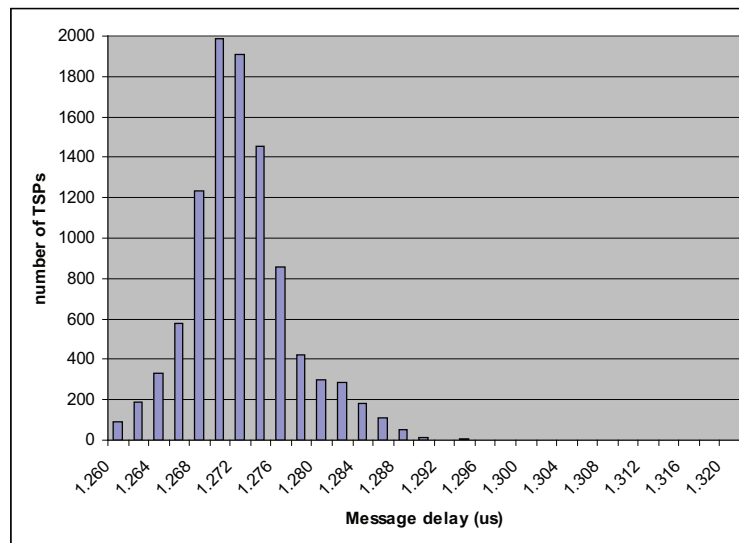


Figure 6.7: Histogram of message delays depicting the range of uncertainties

In order to test the accuracy of offset synchronisation in a pair of immediate neighbour-tiles running in FPGAs, a third, measuring FPGA is used. Tiles reveal their time register and their stored offsets to the GPIO. The measurement FPGA reads these time readings and the corresponding offsets. It measures the clock skew by calculating the difference between the registered offset and the actual local time differences and finally reports the measurements. In order to emulate low quality clock sources, the incrementing of local time is driven by a separate clock source – an internal clock oscillator from an off-the-shelf micro-controller [102]. To reveal the impact of the clock drift, these two micro-controllers are kept in stable but different temperatures ($\pm 10^{\circ}\text{C}$), by placing one of them under a desk lamp. The local time is incremented approximately every $1\mu\text{s}$ (i.e. the frequency is 1MHz), which is much larger than message delay uncertainty. TSPs are transmitted approximately every 1s and restore the clock skew down to zero. Measurement results for five pairs of tiles are shown in Figure 6.8.

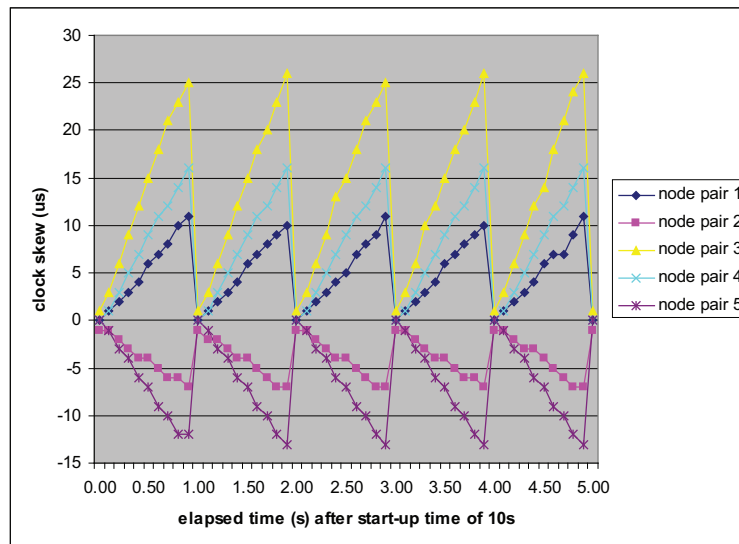


Figure 6.8: Offset synchronisation without drift correction

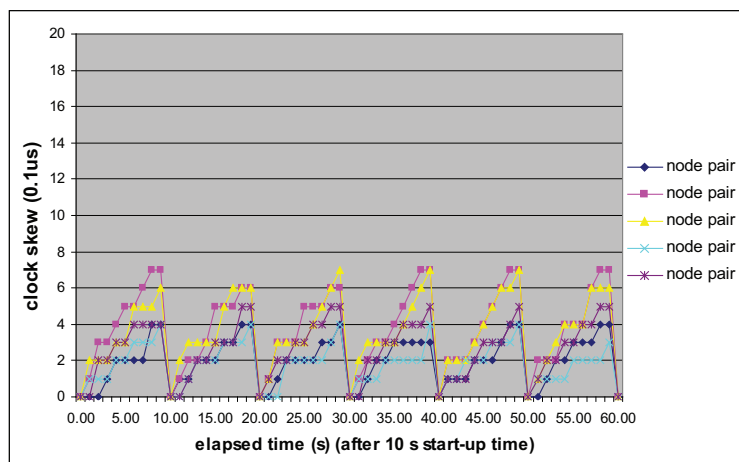


Figure 6.9: Offset synchronisation with drift correction

It is clear from the results given in Figure 6.8 that the clock skew increases linearly up to $27\mu s$ within a second, which corresponds to a relative rate difference of approximately 27PPM. As the clock skew between network nodes is directly proportional to the hop distance (Equation 6.35), $27\mu s$ skew between immediate neighbours allows a network with a diameter of 37 hops (e.g. 20×18 2D mesh) to have a maximum network clock skew of less than 1ms. In order to achieve sub-microsecond accuracy, first it is necessary to increase the resolution of local times. Since the message delay uncertainty is less than $0.06\mu s$ (Figure 6.7), the operating frequency of the local clocks can be comfortably increased to 10MHz without violating the constraints set in 6.12. Figure 6.9 presents the measurements of a setup running with the new clock source frequency of 10MHz. This setup also estimates and controls the impact of the clock drift (Section 6.6). While the increase in time resolution is the main contributor to the improved accuracy, drift correction maintains this accuracy for a longer period of time, thus reducing communication demands. In these measurements, Δ_{max} was set to $1.2\mu s$, resulting in $P = 10^8$ clock cycles, corresponding to an (approximately) ten second period between re-synchronisations.

In Figures 6.8 and 6.9, sudden drops in clock skew result from TSP transmissions required for re-synchronisation. While in simple offset synchronisation (Figure 6.8) re-synchronisations are repeated every second, this would be unnecessarily frequent when clock drifts are estimated and accounted for (Figure 6.9). Measurements in Figure 6.9 show that the rate of TSP transmissions is reduced by approximately ten times when accounting for clock drifts. The TSP packets are rather small and almost negligible when compared to the amount of communication required for chain synchronisation. However, the frequency of offset re-synchronisations determines the frequency of chain re-synchronisations, thus heavily influencing the overall communication requirements of the algorithm. The amount of information communicated during chain synchronisation depends on the size of the chain. Figure 6.10 presents simulation results indicating the average amount of time synchronisation data transmitted by each tile during chain synchronisation – packets carry four bytes of data. For the same simulations, Figure 6.11 presents the amount of setup time taken to complete the chain synchronisation after drift estimation. An examination of the chain synchronisation setup time reveals quadratic increases due to the linear increase of offset data that needs to be processed and communicated on every tile. The chart presented here shows an approximately linear increase of synchronisation time with the increase of chain size, suggesting low coefficients of quadratic scaling.

The advantages of using chain synchronisation are revealed when synchronising clusters of tiles. Simulations of a range of square-shaped clusters were prepared to evaluate the communication load of synchronising clusters. In these simulations, two methods of cluster synchronisations are tested: chain-based synchronisation (Section 6.8) and a *comprehensive cluster synchronisation*. In the comprehensive cluster synchronisation scheme, each tile collects sufficient offset information to allow local transformations of time information coming from any other tile in the cluster. This algorithm also depends on chain synchronisation to collect offset information in rows and columns of the cluster. After forming the chains that correspond to each row and column, each tile contains offset information about the row and column it belongs to. Afterwards, tiles from each row-chain exchange their column offset information with each other, thus collecting offset data for the whole cluster. Figure 6.12 depicts the scaling of communication load as clusters

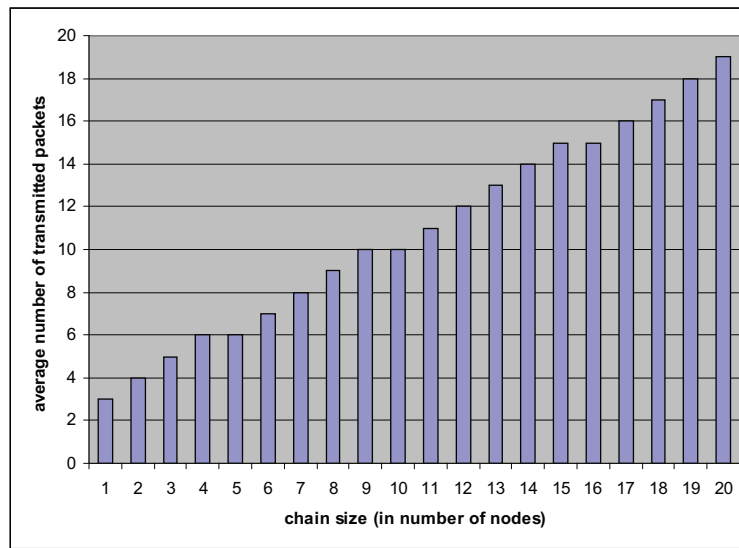


Figure 6.10: Average transmission per tile during chain synchronisation

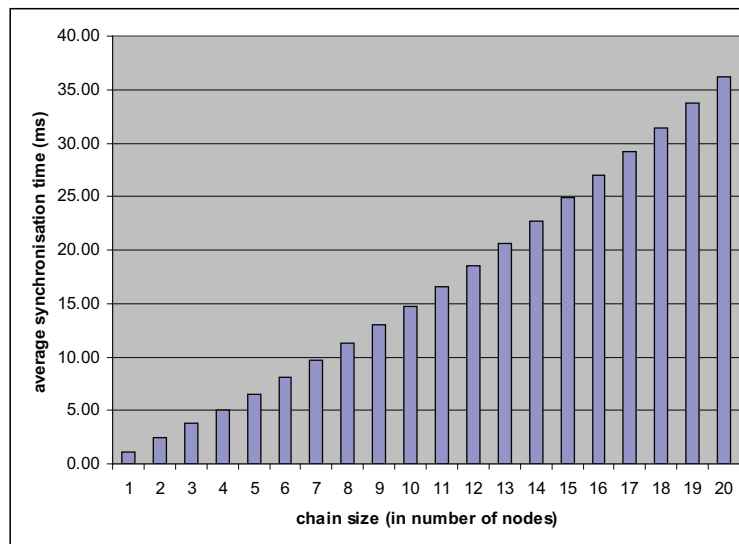


Figure 6.11: Average time taken to achieve chain synchronisation

get larger. The communication load for chain-based cluster synchronisation is doubled compared with individual chain synchronisations. That is due to the fact that during chain-based cluster synchronisation, each tile takes part in two chain synchronisations. On the other hand, the communication load for a comprehensive cluster synchronisation increases quadratically with the increase of the cluster size. In addition, local memory requirements for storing offset information also increase quadratically in the comprehensive cluster synchronisation scheme.

One of the tradeoffs of chain-based cluster synchronisation is intermediate timestamp transformations. In comparison with the comprehensive cluster synchronisation scheme, chain-based synchronisation exhibits slower timestamp delivery. In the previous simulations, after cluster synchronisation is achieved, each tile communicates timestamps with other, randomly chosen cluster tiles. The transmission times are measured for both types

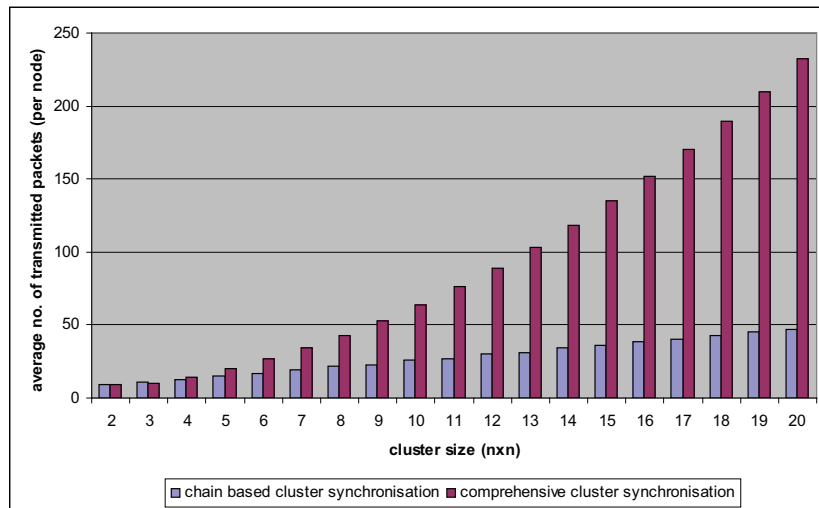


Figure 6.12: Average number of transmitted packets by each node to achieve cluster synchronisation

of cluster synchronisation with no other traffic in the network. For tile-based cluster synchronisation, only timestamps that travel to different chains are considered. Figure 6.13 shows a comparison of average delivery times for a range of hop distances. It is clear from this chart that the overheads are not significant, especially when communicating among longer distances. This is attributed to the fact that network communication requires more time than the processing of timestamps.

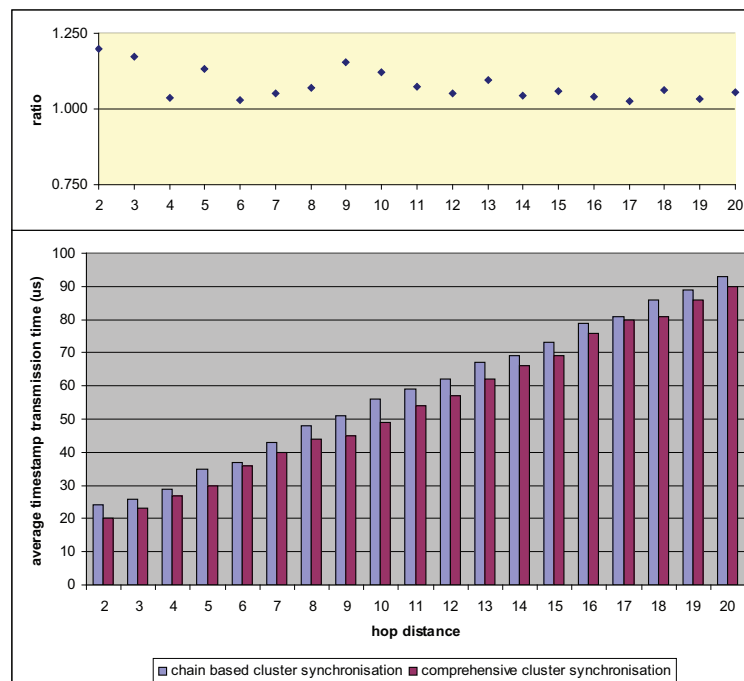


Figure 6.13: Average timestamp delivery time in cluster synchronisation and the corresponding ratio between delivery times

In one of the most comprehensive studies of the NTP to date[165], it is reported that most NTP clocks were within 21ms of their servers and all are within 29ms on average. These measurements were done on 38,722 NTP servers. A direct comparison between this clock skew report and the ones from this work (Figures 6.8 and 6.9) are not entirely just mainly due to the number of nodes involved and the differences in network topology. A slightly more fair comparison would be to measure the time offsets of a computing surface network of 40,000 nodes against the reported NTP results. Although such a network has not been constructed, its performance can be estimated due to the fact that clock skew in chain synchronisation grows linearly (Equation 6.35). If we assume a computing surface of 40,000 nodes organised in a 200×200 mesh, the most distant nodes are 398 hops apart. Offset synchronisation without drift correction would result in maximum of 11ms (assuming $27\mu s$ clock skew in one hop – Figure 6.8), whilst with drift correction and a 10MHz clock counter, the maximum clock skew would be less than 400us (assuming $1\mu s$ clock skew per hop - Figure 6.9). However, due to the tree-like topology of the Internet, clock skew increases much slower in comparison to chain synchronisation.

If incremental time-steps would be broadcasted to all the nodes in the computing surface through a dedicated physical channel, the clock skew of the time reference delivered to every tile would be extremely low (in the range of ns). Yet, this does not guarantee accurate synchronisation in itself due to uncertainties introduced in the software layer. If combined with a mechanism such as the one described in Section 6.4, maximum clock skew below μs range would be achieved for very large networks.

Compared to other hardware-based time synchronisation algorithms such as [64] and [1], the protocol presented here achieves similar accuracy with low communication footprint.

6.12 Summary

Time synchronisation is an essential middleware service for any distributed system, including computing surfaces. Classical time synchronisation algorithms, including NTP are not suitable for computing surfaces. On the other hand, direct adoption of time synchronisation algorithms developed for wireless sensor networks is also inappropriate.

A TSS-based algorithm that provides implicit offset and rate synchronisation is proposed. Using a dedicated Time-Controller module and the introduction of special TSP packets, precise implicit offset synchronisation for immediate neighbouring tiles is achieved. We see that the communication overheads introduced by one-flit TSP packets are negligible. High precision in offset synchronisations also allows for drift corrections, which provide implicit rate synchronisation.

A linear chain of tiles is synchronised through synchronisation of immediate neighbouring tiles. Chain synchronisation is used as a component for cluster synchronisation. Considering the localised nature of applications running in computing surfaces, cluster synchronisation is a suitable form of synchronisation.

Its implementation in the computing surface hardware prototype successfully demonstrated sub-microsecond accuracy in the synchronisation of two neighbouring tiles. When correcting for drift differences, this accuracy was maintained across ten second durations without the need for any intermediate time information updates, thus demonstrating very low communication overhead.

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

Emerging display, sensor and electronic technologies promise flexible and affordable sensitive displays. Their compelling technological and economical advantages inspire a range of exciting applications involving arbitrarily large areas of interactive display surfaces. However, conventional approaches to their design and implementation deliver only limited scalability, thus posing a substantial challenge to many of these applications. This thesis has introduced an alternative approach: distributed computing surfaces as a platform for scalable interactive display systems.

The fully distributed network architecture of computing surfaces provides inherent structural scalability. The embedded 2D mesh interconnection network allows form-factor uniformity. The spatial relationship between individual tiles has a determinant effect on the way input, processing and output are managed. The distribution of processing resources across tiles exposes new design demands. New algorithms related to surface input collection and output management are needed. This thesis has presented suitable algorithms for managing both input and output. The demonstrated efficiency of these algorithms reveals the potential of this platform. These algorithms also reveal the impact of tile-to-tile communication in the performance of the system. Communication-intensive algorithms raise new performance considerations for the interconnection network. Moreover, the distributed nature of computing surfaces necessitates the availability of some essential distributed services for accomplishing advanced operations and interactions. Time synchronisation is a prime component of such essential services. A novel algorithm that is tightly coupled with the computing surface architecture has been presented and evaluated.

The work presented in this thesis demonstrates that computing surfaces are a compelling candidate for the implementation of thin, scalable, interactive display surfaces.

7.2 Summary

The notion of computing surfaces was introduced in Chapter 1. Computing surfaces address the growing gap between the bright technological prospects for providing large sensitive display areas on the one hand, and the inappropriate existing driving architectures on the other.

Scalability limitations of existing computing and display architectures were described in Chapter 2. In addition, this chapter also reviewed the potentials of emerging technologies to realise thin, conformable, sensitive displays of any size – organic electronics, OLED based displays and appropriate sensor technologies for integrating into large sensitive displays. The usability of large sensitive displays was reviewed on the basis of studies conducted by the HCI research community. As an architectural framework for the sensitive displays that are envisaged in such studies, computing surfaces need to efficiently support the proposed applications and interactions.

The architectural details of computing surfaces were presented in Chapter 3. The tiled, 2D mesh architecture provides the required architectural scalability. Additional compelling advantages of this architecture such as homogeneity and autonomy are revealed. Finally, a hardware prototype and a test platform for computing surfaces were described.

Direct user input with touch-based gestures is essential in novel interaction techniques with sensitive display surfaces. The distributed nature of computing surfaces necessitates novel algorithms that take into account the architectural constructs of the tiled surface in order to provide seamless interaction. Chapter 4 proposed a universal middleware platform for managing stroke input. This middleware service enables off-the-shelf gesture recognition algorithms to be used in computing surfaces. Additionally, a novel, communication-efficient distributed algorithm for recognising unistroke gestures was developed and tested on the proposed middleware, further demonstrating its feasibility.

Direct manipulation with digital content is yet another critical aspect of novel interactions with sensitive displays. Inspired by the idea that during image transformations computing surfaces require pixels to travel from one location/tile to another, Chapter 5 described a novel approach to 2D-transformations of images. A suite of algorithms that facilitates a distributed approach to these transformations was proposed and evaluated. Thorough evaluation of the traffic patterns generated by image transformations reveals significant performance implications of the networking infrastructure. Traditional router architectures that employ locally fair arbitration exhibit poor network utilisation due to global unfairness in the allocation of network resources. It was demonstrated that controlled injection rates improve network utilisation in deterministic routing algorithms and therefore improve the overall performance of the network in executing distributed 2D-transformations.

Chapter 6 proposed a novel time synchronisation algorithm for computing surfaces. This algorithm provides communication-efficient time synchronisation without compromising on high accuracy. Its high accuracy is made possible by minimising message delay uncertainty with hardware mechanisms implemented in the network interface. The low message delay uncertainty allows for controlling the effects of clock drifts. Synchronisation chains and clusters efficiently provide this service to any application running in clusters of neighbouring tiles.

The contributions of this dissertation come in two categories: those that are tangible and verifiable, and those that are suggestive and need time to develop. Tangible contributions presented in each chapter were summarised above. The larger contribution which this thesis hopes to make is a matter of consciousness-raising within the design and engineering community. Although scalability limitations in designing large format sensitive displays have been encountered and addressed in isolated situations, an all-encompassing treatment has been missing. By adopting the tiling approach in all architectural levels, the computing surface framework provides one such treatment.

The architecture that has been suggested here, together with the novel algorithms that have been presented, demonstrate that computing surfaces are both flexible and practical.

7.3 Future work

In essence, computing surfaces reveal a powerful concept whose full materialisation of potential remains for future work. Chapter 2 emphasised that robust, thin, conformable sensitive displays remain work in progress. Only after overcoming the aforementioned precursors can implementations that encompass all features of computing surfaces be considered. Nevertheless, a range of system issues can be addressed without waiting for final implementations.

An immediate project that would allow more thorough investigation of computing surfaces is the development of a prototype that encapsulates the spatial and architectural features of computing surfaces. Multi-touch sensitive LCD displays embedded in minimal form factors are gradually becoming more commoditised. These off-the-shelf products, in conjunction with customised systems that provide appropriate communication and computation, allow for more compelling tile prototypes. Such advanced prototypes would allow the execution of integrated applications, making possible further explorations of the potential of computing surfaces.

Computing surfaces also open a wide range of interesting research topics. While the underlying networked platform is sound, the implementation details may gradually evolve towards an architecture that is optimised for ubiquitous interactive surfaces. In addition, the spatial relationship between individual tiles in this distributed system requires further investigation of suitable, communication-centric algorithms.

More specifically, the following avenues should provide promising research with practical and theoretical significance:

- *Software architectures* – developing software applications for computing surfaces requires an underlying software architecture. As discussed in Section 3.5, software architectures for table-top displays can serve as a starting point.
- *Architectural/networking* – the tight collaboration of tiles for enabling smooth interaction necessitates low communication latency. Some form of Quality-of-Service may be necessary for the networking infrastructure. In addition, a better understanding of collective communication needs among applications running in computing surfaces would be beneficial to performance. The collective communication requirements are strongly differentiated from point-to-point requirements. Broadcast and multicast types of communications involve rather small messages that are primarily

latency-bound. Since latency is likely to improve much more slowly than bandwidth, the separation of concerns may suggest adding a separate latency-oriented network dedicated to such collectives.

The evaluation of network traffic exhibited by distributed transformations reveals the importance of efficient networking infrastructure. Switch allocation schemes that implicitly regulate fair injection rates may constitute a compelling solution for the poor network resource utilisation witnessed in Chapter 5.

- *‘Not-so-thin’ client* – the primary purpose of computation and communication resources present in each tile is to provide the desired scalability. Yet the aggregate resources of this platform may result in enormous processing infrastructure. Efficient utilisation of this infrastructure to perform significant computational tasks (such as those in traditional desktop computers) is hindered by our poor understanding of the design, implementation, evaluation and verification of distributed algorithms. In these circumstances, the most suitable mode of operation for computing surfaces is to serve as a thin client platform. However, as demonstrated with the input and output algorithms presented in Chapters 4 and 5, the large number of processors can be efficiently used in many scenarios. The use of such distributed algorithms pushes the boundaries of the ‘thin client’ concept. Future work could reveal more algorithms that would benefit from such a homogeneous processing structure in computing surfaces. Most importantly, future work needs to determine the tradeoffs between this shift from host to computing surface.
- *Fault tolerance* – the fully distributed architecture of computing surfaces offers the ability to provide fault-tolerant mechanisms at many architectural levels. By being autonomous and homogeneous, computing surface tiles provide a compelling base for implementing fault tolerance by replication. In addition, the availability of processing units in each tile allows easier detection and containment of points of failure. The close relationship between physical and topological locality opens up some new research avenues for fault tolerance in computing surfaces.
- *Hexagonal systems* – the advantages of hexagonal representation of visual information are well recognised among the specialised research community [103]. In Cartesian coordinate systems, the distance between neighbouring points/pixels differs, depending on whether the neighbour is adjacent among lattices (1 unit) or among diagonals ($\sqrt{2}$ units). In the hexagonal system, hexagons are surrounded by six other hexagons, which are equidistant (1 unit). This property, along with the perception that hexagons are ‘rounder’ than rectangles, makes the presentation of visually curved features in images more consistent than under normal mappings. In computing surfaces, communication with diagonally adjacent nodes comes with a latency penalty caused by the additional hop. A computing surface that uses hexagons as tiles, connected in a hexagonal mesh, can not only bring latency uniformity among adjacent tiles, but it can also bring a higher degree of inter-connectivity in the overall network.

While not exhaustive, this list of future research possibilities demonstrates the potential number of novel features that computing surfaces could introduce to interactive

display surfaces. The work presented in this thesis, in conjunction with many promising technologies currently being developed, may take scalable, thin, interactive display surfaces one step further towards ubiquity.

BIBLIOGRAPHY

- [1] B. Abali, C. B. Stunkel, and C. Benveniste. Clock synchronization on a multicomputer. *Journal of Parallel and Distributed Computing*, 40(1):118–130, 1997.
- [2] J. A. Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill Education, 1989.
- [3] Allen, K. J. Reel to real: prospects for flexible displays. *Proceedings of the IEEE*, 93(8):1394–1399, 2005.
- [4] P. Alt and P. Pleshko. Scanning limitations of liquid-crystal displays. *IEEE Transactions on Electron Devices*, 21(2):146 – 155, 1974.
- [5] Altera Corporation. *Nios II Processor Reference Handbook*, 2008. <http://www.altera.com/literature/lit-nio2.jsp>.
- [6] K. Amundson. *Electrophoretic Imaging Films for Electronic Paper Displays*, pages 369–391. John Wiley and Sons, 2005.
- [7] A. C. Arias, S. E. Ready, R. Lujan, W. S. Wong, K. E. Paul, A. Salleo, M. L. Chabinye, R. Apte, R. A. Street, Y. Wu, P. Liu, and B. Ong. All jet-printed polymer thin-film transistor active-matrix backplanes. *Applied Physics Letters*, 85(15):3304–3306, 2004.
- [8] J. Arvo. *Graphics Gems*, volume 2. Academic Press, 1991.
- [9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.
- [10] A. Bernanose. The mechanism of organic electroluminescence. *Journal of Chemical Physics*, 52:396–400, 1955.
- [11] A. Bernanose, M. Comte, and P. Vouaux. A new method of emission of light by certain organic compounds. *Journal of Chemical Physics*, 50:64–68, 1953.
- [12] Bluespec, 2008. <http://www.bluespec.com/>.
- [13] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.
- [14] B. Bollobas. *Modern Graph Theory*. Springer, 1998.

- [15] L. Burgi, R. Pfeiffer, M. Mucklich, P. Metzler, M. Kiy, and C. Winnewisser. Optical proximity and touch sensors based on monolithically integrated polymer photodiodes and polymer LEDs. *Organic Electronics*, 7(2):114–120, 2006.
- [16] J. H. Burroughes, D. D. C. Bradley, A. R. Brown, R. N. Marks, K. Mackay, R. H. Friend, P. L. Burns, and A. B. Holmes. Light-emitting diodes based on conjugated polymers. *Nature*, 347:539–541, 1990.
- [17] B. Buxton. Multi-Touch Systems that I Have Known and Loved. <http://www.billbuxton.com/multitouch0verview.html>.
- [18] B. Buxton. Living in augmented reality: Ubiquitous Media and Reactive Environments. *Video Mediated Communication*, pages 363–384, 1997.
- [19] H. Chen, Y. Chen, A. Finkelstein, T. Funkhouser, K. Li, Z. Liu, R. Samanta, and G. Wallace. Data distribution strategies for high-resolution displays. *Computers and Graphics*, 25(5):811–818, 2001.
- [20] H. Chen, Y. Chen, A. Finkelstein, T. Funkhouser, K. Li, Z. Liu, R. Samanta, and G. Wallace. Data distribution strategies for high-resolution displays. *Computers and Graphics*, 25:811–818, 2001.
- [21] H. Chen, D. W. Clark, Z. Liu, G. Wallace, K. Li, and Y. Chen. Software environments for cluster-based display systems. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 202, Washington, DC, USA, 2001. IEEE Computer Society.
- [22] H. Chen, R. Sukthankar, G. Wallace, and T. Cham. Calibrating scalable multi-projector displays using camera homography trees. In *Computer Vision and Pattern Recognition*, 2001.
- [23] H. Chen, R. Sukthankar, G. Wallace, and K. Li. Scalable alignment of large-format multi-projector displays using camera homography trees. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 339–346, Washington, DC, USA, 2002. IEEE Computer Society.
- [24] H. Chen, G. Wallace, A. Gupta, K. Li, T. Funkhouser, and P. Cook. Experiences with scalability of display walls. In *Proceedings of the Immersive Projection Technology (IPT) Workshop*, 2002.
- [25] Y. Chen, D. W. Clark, A. Finkelstein, T. Housel, and K. Li. Automatic Alignment Of High-Resolution Multi-Projector Displays Using An Un-Calibrated Camera. In *VISUALIZATION '00: Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*, Washington, DC, USA, 2000. IEEE Computer Society.
- [26] J. H. Cheon, J. H. Bae, and J. Jang. P-28: Flexibility Study of High-Performance LTPS-TFT on Flexible Metal Foil. *SID Symposium Digest of Technical Papers*, 38(1):272–275, 2007.

- [27] D. S. Clouse, C. L. Giles, B. G. Horne, and G. W. Cottrell. Time-Delay Neural Networks: Representation and Induction of Finite-State Machines. *IEEE Transactions on Neural Networks*, 8(5):1065–1070, 1997.
- [28] B. Comiskey, J. D. Albert, H. Yoshizawa, and J. Jacobson. An electrophoretic ink for all-printed reflective electronic displays. *Nature*, 394(6690):253–255, 1998.
- [29] G. P. Crawford. *Flexible Flat Panel Displays*. John Wiley and Sons, 2005.
- [30] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, New York, NY, USA, 1993. ACM.
- [31] M. Czerwinski, G. Robertson, B. Meyers, G. Smith, D. Robbins, and D. Tan. Large display research overview. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 69–74, New York, NY, USA, 2006. ACM.
- [32] H. Dai and R. Han. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(1):125–139, 2004.
- [33] W. J. Dally, P. Carvey, and L. Dennison. Architecture of the Avici terabit switch/router. In *Proceedings of Hot Interconnects Symposium*, volume 6, pages 41–50. IEEE, 1998.
- [34] W. J. Dally and B. P. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.
- [35] J. H. Daniel, A. C. Arias, W. S. Wong, R. Lujan, B. S. Krusor, R. B. Apte, M. L. Chabiny, A. Salleo, R. A. Street, N. Chopra, G. Iftime, and P. M. Kazmaier. 54.1: Flexible Electrophoretic Displays with Jet-Printed Active-Matrix Backplanes. *SID Symposium Digest of Technical Papers*, 36(1):1630–1633, 2005.
- [36] P. Dietz and D. Leigh. DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 219–226, New York, NY, USA, 2001. ACM.
- [37] C. D. Dimitrakopoulos and D. J. Mascaró. Organic thin-film transistors: A review of recent advances. *IBM Journal of Research and Development*, 45(1):11–27, 2001.
- [38] L. M. Do, E. M. Han, Y. Niidome, M. Fujihira, T. Kanno, S. Yoshida, A. Maeda, and A. J. Ikushima. Observation of degradation processes of Al electrodes in organic electroluminescence devices by electroluminescence microscopy, atomic force microscopy, scanning electron microscopy, and Auger electron spectroscopy. *Journal of Applied Physics*, 76(9):5118–5121, 1994.
- [39] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

- [40] J. E. Elson. *Time Synchronisation in Wireless Sensor Networks*. PhD thesis, University of California, Los Angeles, 2003.
- [41] R. Fan and N. Lynch. Gradient clock synchronization. *Distrib. Comput.*, 18(4):255–266, 2006.
- [42] W. F. Feehery. Solution Processing of Small-Molecule OLEDs. In *SID Symposium Digest of Technical Papers*, pages 1834–1837. Society for Information Display, 2007.
- [43] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
- [44] G. W. Fitzmaurice, H. Ishii, and W. A. S. Buxton. Bricks: laying the foundations for graspable user interfaces. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 442–449, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [45] S. Forrest. Ultrathin Organic Films Grown by Organic Molecular Beam Deposition and Related Techniques. *Chemical Reviews*, 97(6):1793–1896, 1997.
- [46] S. R. Forrest. The path to ubiquitous and low-cost organic electronic appliances on plastic. *Nature*, 428:911–918, 2004.
- [47] R. H. Friend. Conjugated polymers. New materials for optoelectronic devices. *Pure and Applied Chemistry*, 73(3):425–430, 2001.
- [48] M. Fukuhara, Y. Igahama, T. Suzuki, and H. Yamaguchi. . *Proceedings of 15th International Display Research Conference*, page 197, 1995.
- [49] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM Press.
- [50] G. H. Gelinck, H. E. A. Huitema, E. van Veenendaal, E. Cantatore, L. Schrijnemakers, J. B. P. H. van der Putten, T. C. T. Geuns, and M. Beenhakkers. Flexible active-matrix displays and shift registers based on solution-processed organic transistors. *Nature Materials*, 3(2):106–110, 2004.
- [51] D. Germans, H. J. W. Spoelder, L. Renambot, and H. E. Bal. VIRPI: A High-Level toolkit for interactive scientific visualization in virtual reality. In *IPT/EGVE*, pages 109–120, 2001.
- [52] A. Glassner. *Graphics Gems*, volume 1. Academic Press, 1990.
- [53] D. Goldberg. Unistrokes for computerized interpretation of handwriting. United States Patent No. 5,596,656, 1997.
- [54] D. Goldberg and C. Richardson. Touch-typing with a stylus. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 80–87, New York, NY, USA, 1993. ACM.

- [55] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*, chapter 3. Addison-Wesley, 1994.
- [56] R. G. Greene, J. P. Krusius, D. P. Seraphim, D. Skinner, and B. Yost. 30.3: Seamless Tiling Technology for Large Direct-View Color AMLCD's. *SID Symposium Digest of Technical Papers*, 31(1):461–463, 2000.
- [57] G. Greiner and K. Hormann. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics*, 17(2):71–83, 1998.
- [58] J. Y. Halpern, N. Megiddo, and A. A. Munshi. Optimal precision in the presence of uncertainty. *Journal of Complexity*, 1:170–196, 1985.
- [59] J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 115–118, New York, NY, USA, 2005. ACM.
- [60] P. Heckbert. *Graphics Gems*, volume 4. Academic Press, 1994.
- [61] K. Hinckley. Synchronous gestures for multiple persons and computers. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 149–158, New York, NY, USA, 2003. ACM.
- [62] S. Hodges, S. Izadi, A. Butler, A. Rrustemi, and B. Buxton. ThinSight: versatile multi-touch sensing for thin form-factor displays. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 259–268, New York, NY, USA, 2007. ACM.
- [63] G. Hollemans, T. Bergman, V. Buil, K. van Gelder, M. Groten, J. Hoonhout, and T. Lashina. Entertaible: Multi-user multi-object concurrent input. *UIST 2006, Adjunct Proceedings: Demonstrations*, pages 55–56, 2006.
- [64] M. Horauer, U. Schmid, and K. Schossmaier. NTI: A network time interface module for high-accuracy clock-synchronization. In *IPPS/SPDP Workshops*, pages 1067–1076, 1998.
- [65] M. Horauer, U. Schmid, and K. Schossmaier. PSynUTC - Evaluation of a High Precision Time Synchronization Prototype System for Ethernet LANs. In *PTTI*. IEEE, 2002.
- [66] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.
- [67] S. E. Hudson. Using light emitting diode arrays as touch-sensitive input and output devices. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 287–290, New York, NY, USA, 2004. ACM.
- [68] H. E. A. Huitema, G. H. Gelinck, E. van Veenendal, E. Cantatore, F. J. Touwslager, and L. R. R. Schrijnemakers. A Flexible QVGA Display With Organic Transistors. *Proc Int Disp Workshops*, 10:1663–1664, 2003.

- [69] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. Wiregl: a scalable graphics system for clusters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 129–140, New York, NY, USA, 2001. ACM.
- [70] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.
- [71] T. Isenberg, A. Miede, and S. Carpendale. A buffer framework for supporting responsive interaction in information visualization interfaces. In *C5 '06: Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing*, pages 262–269, Washington, DC, USA, 2006. IEEE Computer Society.
- [72] H. Ishii and B. Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1997. ACM.
- [73] ITRS report. <http://public.itrs.net>.
- [74] B. Jeong, R. Jagodic, L. Renambot, R. Singh, A. Johnson, and J. Leigh. Scalable Graphics Architecture for High-Resolution Displays. *Proceedings of IEEE Information Visualization Workshop*, 2005.
- [75] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 139–146, New York, NY, USA, 2007. ACM.
- [76] S. Ju, A. Facchetti, Y. Xuan, J. Liu, F. Ishikawa, P. Ye, C. Zhou, T. J. Marks, and D. B. Janes. Fabrication of fully transparent nanowire transistors for transparent and flexible electronics. *Nature Nanotechnology*, 2:378–384, 2007.
- [77] C. R. Kagan, D. B. Mitzi, and C. D. Dimitrakopoulos. Organic-Inorganic Hybrid Materials as Semiconducting Channels in Thin-Film Field-Effect Transistors. *Science*, 286(5441):945–947, 1999.
- [78] M.-K. Kang, K. Uh, and H. Kim. 35-1: Invited Paper: Advanced Technologies Based on a-Si or LTPS (Low Temperature Poly Si) TFT (Thin Film Transistor) for High Performance Mobile Display. *SID Symposium Digest of Technical Papers*, 38(1):1262–1265, 2007.
- [79] K. Kincade. Flexible displays open new windows of opportunity. *Laser Focus World*, 40:65–69, 2004.
- [80] D. Kirk. *Graphics Gems*, volume 3. Academic Press, 1992.

- [81] N. K. Krishnaprasad, V. Vishwanath, S. Venkataraman, A. G. Rao, L. Renambot, J. Leigh, A. E. Johnson, and B. Davis. Juxtaview - a tool for interactive visualization of large imagery on scalable tiled displays. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 411–420, Washington, DC, USA, 2004. IEEE Computer Society.
- [82] J. P. Krusius, D. P. Seraphim, R. G. Greene, and B. Yost. 8.1: Seamless Tiling of AMLCDs for Large Area Displays. *SID Symposium Digest of Technical Papers*, 33(1):78–81, 2002.
- [83] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. Jha. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *International Conference on Computer Design (ICCD)*, October 2007.
- [84] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler. Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services. *International Journal of Ad Hoc and Ubiquitous Computing*, 1:239–251(13), 2006.
- [85] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [86] G. Largillier. Developing the First Commercial Product that Uses Multi-Touch Technology. *Information Display Magazine*, 23(12), 2007.
- [87] S. Lee, W. Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. In *CHI '85: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 21–25, New York, NY, USA, 1985. ACM.
- [88] M.-Q. Li and M.-D. Rus. Global clock synchronization in sensor networks. *IEEE Transactions on Computing*, 55(2):214–226, 2006.
- [89] Z. Li and H. Meng. *Organic Light-Emitting Materials and Devices (Optical Science and Engineering Series)*. CRC, 2006.
- [90] R. C. Liang, J. Hou, and H. M. Zang. Microcup Electrophoretic Displays by Roll-to-Roll Manufacturing Processes. *Proceedings of International Display Workshops*, 9:1337–1340, 2002.
- [91] C. H. Lin, W. Wolf, A. Dixon, X. Koutsoukos, and J. Sztipanovits. Design and implementation of ubiquitous smart cameras. In *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing - Vol 1 (SUTC'06)*, pages 32–39, Washington, DC, USA, 2006. IEEE Computer Society.
- [92] Y.-Y. Lin, D. Gundlach, S. Nelson, and T. Jackson. Stacked pentacene layer organic thin-film transistors with improved characteristics. *Electron Device Letters*, 18(12):606 – 608, 1997.
- [93] A. C. Lowe, P. A. Bayley, N. A. Gallen, M. Huang, and B. Needham. A Novel Approach to Tiled Displays. *SID Symposium Digest of Technical Papers*, 34(1):180–183, 2003.

- [94] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62:190–204, 1984.
- [95] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.
- [96] S. Malik and J. Laszlo. Visual touchpad: a two-handed gestural input device. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 289–296, New York, NY, USA, 2004. ACM.
- [97] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM Press.
- [98] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 295–305, New York, NY, USA, 1983. ACM Press.
- [99] N. Matsushita and J. Rekimoto. HoloWall: designing a finger, hand, body, and object sensitive wall. In *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 209–210, New York, NY, USA, 1997. ACM.
- [100] T. Mayer. New options and considerations for creating enhanced viewing experiences. *Computer Graphics*, 1997.
- [101] N. McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*, 7(2):188–201, 1999.
- [102] Microchip Technology Inc. *PIC16F818/819 Data Sheet*, 2004. <http://ww1.microchip.com/downloads/en/DeviceDoc/39598e.pdf>.
- [103] L. Middleton and J. Sivaswamy. *Hexagonal Image Processing*. Springer-Verlag, 2005.
- [104] D. L. Mills. Adaptive hybrid clock discipline algorithm for the network time protocol. *IEEE/ACM Trans. Netw.*, 6(5):505–514, 1998.
- [105] D. B. Mitzi. Synthesis, Structure, and Properties of Organic-Inorganic Perovskites and Related Materials. *Progress in Inorganic Chemistry*, 1:1–121, 1999.
- [106] D. B. Mitzi, C. A. Feild, Z. Schlesinger, and R. B. Laibowitz. Transport, Optical, and Magnetic Properties of the Conducting Halide Perovskite. *Journal of Solid State Chemistry*, 114(1):159–163, 1995.
- [107] M. Kobayashi. A 17 in. WXGA Full-Color OLED Display by Using Polymer Ink-Jet Technology. In *International Display Workshop*, page 231. Society for Information Display, 2002.
- [108] D. Morton and E. Forsythe. Flexible-Display Development for Army Applications. *Information Display*, 23(10):18–23, 2007.

- [109] R. Mullins. Netmaker interconnection networks. wiki, 2007. http://www-dyn.cl.cam.ac.uk/~rdm34/wiki/index.php?title=Main_Page.
- [110] T. Nelson and J. R. Wullert II. *Electronic Information Display Technologies*. World Scientific, 1997.
- [111] T. Nesson and S. L. Johnsson. Romm routing on mesh and torus networks. In *SPAA '95: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 275–287, New York, NY, USA, 1995. ACM.
- [112] H. Ohshima and M. Fuhren. 46.2: Invited Paper: High-Performance LTPS Technologies for Advanced Mobile Display Applications. *SID Symposium Digest of Technical Papers*, 38(1):1482–1485, 2007.
- [113] A. Paeth. *Graphics Gems*, volume 5. Academic Press, 1995.
- [114] P. page. Silicon Graphics, Inc.: SGI Onyx. <http://www.sgi.com/>.
- [115] W. H. Paul Horowitz. *The Art of Electronics*. Cambridge University Press, 1989.
- [116] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 255, Washington, DC, USA, 2001. IEEE Computer Society.
- [117] K. A. Perrine and D. R. Jones. Parallel graphics and interactivity with the scaleable graphics engine. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 5–5, New York, NY, USA, 2001. ACM.
- [118] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.
- [119] Product concept, 2007. Plastic Logic <http://www.plasticlogic.com/>.
- [120] M. Pope, H. Kallmann, and P. Magnante. Electroluminescence in organic crystals. *J. Chem. Phys*, 38:2042, 1963.
- [121] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.
- [122] J. Rekimoto. SmartSkin: an infrastructure for freehand manipulation on interactive surfaces. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 113–120, New York, NY, USA, 2002. ACM.
- [123] P. release. Toshiba Matsushita: LCD with Finger Shadow Sensing. http://www3.toshiba.co.jp/tm_dsp/press/2005/05-09-29.htm.
- [124] P. Release. EPSON: Epson Creates World's First Large Full-Color OLED Display Using Original Inkjet Technology. http://www.epson.co.jp/e/newsroom/news_2004_05_18.htm, 2004.

- [125] P. Release. CDT: Cambridge Display Technology and Sumation Announce Improved Performance Characteristics of Green and Blue P-OLED Materials. http://www.cdtltd.co.uk/press/current_press_releases/619.asp, 2007.
- [126] P. Release. SONY: Sony Launches World's First OLED TV. <http://www.sony.net/SonyInfo/News/Press/200710/07-1001E/index.html>, 2007.
- [127] K. Reynolds, S. Burns, M. Banach, T. Brown, K. Chalmers, N. Cousins, L. Creswell, M. Etchells, C. Hayton, K. Jacobs, A. Menon, S. Siddique, C. Ramsdale, W. Reeves, J. Watts, T. V. Werne, J. Mills, C. Curling, H. Siringhaus, K. Amundson, and M. D. McCreary. . 2004.
- [128] D. Risteski, A. Kulakov, and D. Davcev. Single exponential smoothing method and neural network in one method for time series prediction. *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, 2:741–745, 2004.
- [129] J. A. Rogers, Z. Bao, K. Baldwin, A. Dodabalapur, B. Crone, V. R. Raju, V. Kuck, H. Katz, K. Amundson, J. Ewing, and P. Drzaic. From the Cover: Paper-like electronic displays: Large-area rubber-stamped plastic sheets of electronics and microencapsulated electrophoretic inks. *Proceedings of the National Academy of Sciences*, 98(9):4835–4840, 2001.
- [130] K. Romer. Time synchronization in ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 173–182, New York, NY, USA, 2001. ACM Press.
- [131] K. Römer, P. Blum, and L. Meier. Time synchronization and calibration in wireless sensor networks. In I. Stojmenovic, editor, *Handbook of Sensor Networks: Algorithms and Architectures*, pages 199–237. John Wiley & Sons, Sept. 2005.
- [132] M. S. Active-matrix displays. *SID Seminar Lecture Notes*, II:10.1–10.30, 1989.
- [133] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2005.
- [134] R. W. Scheifler and J. Gettys. The x window system. *ACM Trans. Graph.*, 5(2):79–109, 1986.
- [135] S. Scott and S. Carpendale. Investigating Tabletop Territoriality in Digital Tabletop Workspaces. Technical report, Department of Computer Science, University of Calgary, 2006.
- [136] S. D. Scott, M. Sheelagh, T. Carpendale, and K. M. Inkpen. Territoriality in collaborative tabletop workspaces. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 294–303, New York, NY, USA, 2004. ACM.
- [137] D. Seo, A. Ali, W.-T. Lim, N. Rafique, and M. Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 432–443, Washington, DC, USA, 2005. IEEE Computer Society.

- [138] J. M. Shaw and P. F. Seidler. Organic electronics: Introduction. *IBM Journal of Research and Development*, 45(1):3–9, 2001.
- [139] C. Shen, N. B. Lesh, F. Vernier, C. Forlines, and J. Frost. Sharing and building digital group histories. In *CSCW: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 324–333, New York, NY, USA, 2002. ACM.
- [140] T. Shimoda, K. Morii, S. Seki, and H. Kiguchi. Inkjet Printing of Light-Emitting Polymer Displays. *MRS Bulletin*, 28(11):821–827, 2003.
- [141] T. Shinomiya, M. Kawabata, N. Nagae, Y. Izumi, K. Fujimori, S. Fujiwara, M. Shiota, Y. Ishii, and F. Funada. . *Proceedings of SID International Symposium*, page 497, 1997.
- [142] M. Shtein, P. Peumans, J. Benziger, and S. Forrest. Direct, Mask- and Solvent-Free Printing of Molecular Organic Semiconductors. *Advanced Materials*, 16(18):1615–1620, 2004.
- [143] M. Shtein, P. Peumans, J. B. Benziger, and S. R. Forrest. Micropatterning of small molecular weight organic semiconductor thin films using organic vapor phase deposition. *Journal of Applied Physics*, 93(7):4005–4016, 2003.
- [144] M. Sichitiu and C. Veerarittiphan. Simple, Accurate Time Synchronization for Wireless Sensor Networks. In *Wireless Communications and Networking*, pages 1266–1273. IEEE Press, 2003.
- [145] B. Simons, J. L. Welch, and N. A. Lynch. An Overview of Clock Synchronization. In *Proceedings of the Asilomar Workshop on Fault-Tolerant Distributed Computing*, pages 84–96, London, UK, 1990. Springer-Verlag.
- [146] H. Sirringhaus. Device Physics of Solution-Processed Organic Field-Effect Transistors. *Advanced Materials*, 17(20):2411–2425, 2005.
- [147] H. Sirringhaus, T. Kawase, R. H. Friend, T. Shimoda, M. Inbasekaran, W. Wu, and E. P. Woo. High-Resolution Inkjet Printing of All-Polymer Transistor Circuits. *Science*, 290(5499):2123–2126, 2000.
- [148] H. Sirringhaus, N. Tessler, and R. H. Friend. Integrated Optoelectronic Devices Based on Conjugated Polymers. *Science*, 280(5370):1741–1744, 1998.
- [149] D. Smith. Designing the Star User Interface. *Byte*, pages 242–282, April 1982.
- [150] E. C. Smith. 8.3: Total Matrix Addressing (TMA). *SID Symposium Digest of Technical Papers*, 38(1):93–96, 2007.
- [151] T. Someya, T. Sekitani, S. Iba, Y. Kato, H. Kawaguchi, and T. Sakurai. A large-area, flexible pressure sensor matrix with organic field-effect transistors for artificial skin applications. *Proceedings of the National Academy of Sciences*, 101(27):9966–9970, 2004.

- [152] J. Souk. TFT-LCD for Large Size TV Applications. In *International Display Manufacturing Conference (IDMC)*, pages 147–148, 2000.
- [153] N. A. Streitz, J. Geissler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reichl, P. Rexroth, P. Seitz, and R. Steinmetz. i-LAND: an interactive landscape for creativity and innovation. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 120–127, New York, NY, USA, 1999. ACM.
- [154] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 13(2):384–397, 2005.
- [155] A. Sugimoto, H. Ochi, S. Fujimura, A. Yoshida, T. Miyadera, and M. Tsuchida. Flexible oled displays using plastic substrates. *Selected Topics in Quantum Electronics, IEEE Journal of*, 10(1):107–114, Jan.-Feb. 2004.
- [156] I. E. Sutherland and G. W. Hodgman. Reentrant polygon clipping. *Commun. ACM*, 17(1):32–42, 1974.
- [157] SystemVerilog, 2008. <http://www.systemverilog.org/>.
- [158] Synopsys VCS: Comprehensive RTL Verification Solution, 2008. <http://www.synopsys.com/>.
- [159] SystemC, 2008. <http://www.systemc.org/>.
- [160] Y. Tamir and H. C. Chi. Symmetric Crossbar Arbiters for VLSI Communication Switches. *IEEE Trans. Parallel Distrib. Syst.*, 4(1):13–27, 1993.
- [161] P. Tandler, T. Prante, C. Müller-Tomfelde, N. Streitz, and R. Steinmetz. Connectables: dynamic coupling of displays for the flexible creation of shared workspaces. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 11–20, New York, NY, USA, 2001. ACM.
- [162] C. W. Tang and S. A. VanSlyke. Organic electroluminescent diodes. *Applied Physics Letters*, 51(12):913–915, 1987.
- [163] C. Tappert, C. Suen, and T. Wakahara. The state of the art in online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.
- [164] S. Terada, G. Izumi, Y. Sato, M. Takahashi, M. Tada, K. Kawase, K. Shimotoku, H. Tamashiro, N. Ozawa, T. Shibasaki, C. Sato, T. Nakadaira, Y. Iwase, T. Sasaoka, and T. Urabe. 54.5L: Late-News Paper: A 24-inch AM-OLED Display with XGA Resolution by Novel Seamless Tiling Technology. *SID Symposium Digest of Technical Papers*, 34(1):1463–1465, 2003.
- [165] I. timekeeping around the globe. D. l. mills and a. thyagarajan and b. c. huffman. In *Precision Time and Time Interval (PTTI)*, pages 365–371, 1997.
- [166] Tiled Image Viewer (TimV). <http://graphics.ucsd.edu/~cdonner/timv/>.

- [167] Image Magick FAQ. www.imagemagick.com/www/FAQ.html#C9.
- [168] P. Tuddenham and P. Robinson. T3: Rapid prototyping of high-resolution and mixed-presence tabletop applications. *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on*, pages 11–18, Oct. 2007.
- [169] U. University of Minnesota: PowerWall. <http://www.lcse.umn.edu/research/powerwall/powerwall.html>.
- [170] B. Ullmer and H. Ishii. The metaDESK: models and prototypes for tangible user interfaces. In *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 223–232, New York, NY, USA, 1997. ACM.
- [171] J. van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 11–19, New York, NY, USA, 2003. ACM Press.
- [172] J. R. Vig. Introduction to quartz frequency standards. Technical report, Army Research Laboratory, 1992. SLCET-TR-92-1: Available at <http://www.ieee-uffc.org/freqcontrol/quartz/vig/vigtoc.htm>.
- [173] G. Wallace, H. Chen, and K. Li. Color gamut matching for tiled display walls. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pages 293–302, New York, NY, USA, 2003. ACM.
- [174] G. Wallace, H. Chen, and K. Li. DeskAlign: Automatically Aligning a Tiled Windows Desktop. *Proceedings of IEEE International Workshop on Projector-Camera Systems*, 2003.
- [175] X. Wang, V. Vishwanath, B. Jeong, R. Jagodic, E. He, L. Renambot, A. Johnson, and J. Leigh. LambdaBridge: A Scalable Architecture for Future Generation Terabit Applications. In *International Conference on Broadband Communications, Networks and Systems*, volume 3, pages 1–10. IEEE, 2006.
- [176] A. R. Webb. *Statistical Pattern Recognition, 2nd Ed.* John Wiley and Sons, 2002.
- [177] M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–10, September 1991.
- [178] E. W. Weisstein. Cartesian Product. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/CartesianProduct.html>.
- [179] E. W. Weisstein. Sign. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/Sign.html>.
- [180] P. Wellner. Interacting with paper on the DigitalDesk. *Commun. ACM*, 36(7):87–96, 1993.

- [181] A. D. Wilson. TouchLight: an imaging touch screen and display for gesture-based interaction. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 69–76, New York, NY, USA, 2004. ACM.
- [182] A. D. Wilson. PlayAnywhere: a compact interactive tabletop projection-vision system. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 83–92, New York, NY, USA, 2005. ACM.
- [183] K. R. Wood, T. Richardson, F. Bennett, A. Harter, and A. Hopper. Global Teleporting with Java: Toward Ubiquitous Personalized Computing. *Computer*, 30(2):53–59, 1997.
- [184] M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 193–202, New York, NY, USA, 2003. ACM.
- [185] J.-H. Yu, K. Choo, H. Kang, Y. Kim, D. Lee, I. Kang, and I. Chung. P-7: 4 Inch a-Si TFT-LCD with an Embedded Color Image Scanner. *SID Symposium Digest of Technical Papers*, 38(1):196–198, 2007.
- [186] H. Zang and Liang. Microcup electronic paper by roll-to-roll manufacturing processes. *IEEE Spectrum*, 16(2):16–21, 2003.
- [187] H. M. Zang, X. Wang, and R. C. Liang. Composition and process for the sealing of microcups in roll-to-roll display manufacturing. United States Patent No. 7,005,468, 2006.
- [188] C. Zhang, J. Leigh, T. A. DeFanti, M. Mazzucco, and R. Grossman. TeraScope: distributed visual data mining of terascale data sets over photonic networks. *Future Generation Computer Systems*, 19:935–943(9), 2003.