Implications of Electronics Technology Trends to Algorithm Design

Daniel Greenfield and Simon Moore
Computer Laboratory, University of Cambridge
JJ Thomson Avenue, Cambridge, CB3 0FD, United Kingdom
daniel.greenfield@cl.cam.ac.uk, simon.moore@cl.cam.ac.uk

Abstract

Scaling of electronics technology has brought us to a pivotal point in the design of computational devices. Technology scaling favours transistors over wires which has led us into an era where communication takes more time and consumes more power than the computation itself. This technology driver inevitably pushes us toward a communication-centric approach to algorithm design. To assess the efficiency of an algorithm we will need to be able to predict data movement both in time and space. We demonstrate that algorithms exhibit fractal like communication behaviour which is likely to help with such an analysis. Moreover, successfully exploiting these fractal properties will allow us to reduce communication, thereby increasing performance and power efficiency.

Keywords: CMP, communication complexity, algorithms, fractal structure, temporal interconnect, networks-on-chip, Rent's rule, tera-scale

1. INTRODUCTION

For many decades, electronics technology has brought us ever faster scalar processors. Recently this trend has almost stopped; we are now offered more processors each with similar scalar performance to the previous generation. As we enter this manycore era, there are many people looking again at parallel algorithm design. Traditionally parallel computing has focused very heavily on keeping the compute engines busy. We argue in this paper that in the near future the efficiency of communication will become far more important. As a consequence, computational complexity will become a second order effect, so we must look more seriously at the complexity of communication for algorithms.

This work is based on an earlier work [17]: The Next Resource War: Computation vs. Communication, in SLIP '08: Proceedings of the 2008 international workshop on System level interconnect prediction, © ACM, 2008. http://doi.acm.org/10.1145/1353610.1353627

2. TECHNOLOGY DRIVER

Since the birth of the microprocessor in the early 1970s, industry has exploited the exponential increase in transistor density (predicted by Moore's law) to integrate more elaborate processors and memory on-chip (often in the form of caches). During this period, chip sizes have changed little, often with commodity parts being around 10mm on a side, and more expensive parts around 20mm on a side. Prior to 1970, greater integration reduced the across-system wire lengths, but with the advent of the microprocessor, this scaling parameter stopped. Initially this scaling issue was insignificant, but the last 35 years (or so) of scaling transistors from 10,000nm (Intel 4004) to 45nm (current Intel and AMD processors) has changed all this.

In the next section we look at wire scaling in a little more detail and at some of the implications from a technology perspective. Then we look at the interconnectedness of algorithms and their implications for the era of massively multicore Chip Multiprocessors (CMPs).

2.1. Wire scaling

Driving a signal along an across-chip wire has become more complicated. As geometries have shrunk, so has the width of the wire. But we still tend to drive a wire in a DC manner, and the RC time constant has not improved much because a shrink of l results in R increasing by a factor l and C decreasing by a factor l with the net effect that little changes (to a first approximation). For old technologies it was sensible to drive an across-chip wire from one end using a buffer which was not much bigger than other logic gates. Whilst logic gates have shrunk, buffers have not scaled well since RC is little changed.

As clock frequencies have risen, so has the demand for wire bandwidth which is limited by the RC time constant. This has resulted in across-chip wires being broken into smaller segments with buffers inserted in between. If a wire is broken into two segments then the RC time constant reduces by a factor of four so the new combined wire delay is half what it was plus the delay through the extra buffer (see Figure 1). The next step is to pipeline the wire to allow more than one data item to be in flight without risk of inter-symbol interference.

In the early days of the microprocessor, clock frequencies were sufficiently low that performing an off-chip memory access completed in a single clock cycle. Today an off-chip memory access takes 100s of processor clock cycles. Moreover, the power consumed by computation is now dwarfed by the power needed for communication [6]. Table 1 illustrates that in just the few years between 130nm and 45nm CMOS, the communication power requirements have shot up in proportion to the power consumed by arithmetic operations. The power required to send 32-bits of data off-chip is equivalent to 1300 32-bit arithmetic operations with a memory access being even greater.

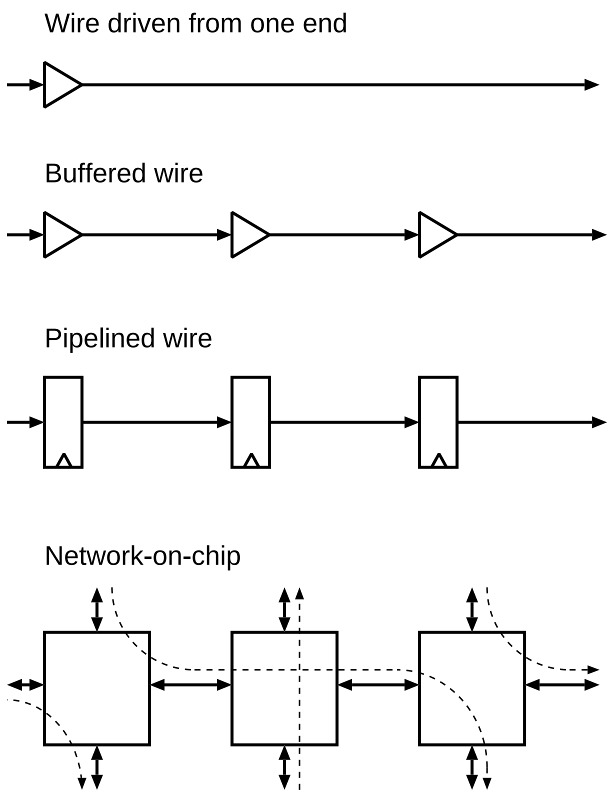

FIGURE 1: Evolution from simple wires to networks-on-chip

TABLE 1: Comparing trends in power consumption

technology node	130nm CMOS	45nm CMOS
transfer 32b across-chip	20 ALU ops	57 ALU ops
transfer 32b off-chip	260 ALU ops	1300 ALU ops

2.2. Toward networks-on-chip

Even on-chip the power needed to communicate across-chip is worrying, particularly now that power is becoming the performance limiter. This leads us to our first observation:

Communication rather than computation now limits performance.

Another wiring issue comes from the complexity of the interconnectedness of the circuits. Rent's rule [12] predicts a slow but exponential increase in the complexity of the interconnect and this has been observed in the increased numbers of layers of metalisation.

As a consequence of Rent's rule, and the silicon area and power of global wiring, increased effort is being placed on managing wires. In particular, there is a great deal of academic and industrial activity looking at networks-on-chip (NoCs), see Figure 1. NoCs are able to manage (schedule and reuse) this precious global wiring resource. At first, the level of control overhead seems excessive, but this turns out to not be the case [2]. By replacing the complex of long interconnects with virtual interconnects running over a regular structure of shorter links, NoCs can reduce the Rent's rule exponent for wiring at the top-level. However, we showed that Rent's rule is still expected to hold for the new virtual wires themselves, transforming into a bandwidth-version of Rent's rule [7].

2.3. Communication off-chip

We noted (see Table 1) that communication power off-chip is not scaling with transistor performance. Figure 2 shows the estimated growth for total chip I/O bandwidth based on ITRS predicted figures [11] for pin-count in their cost-performance balance and the predicted bandwidth growth of high-speed pins. We note that after 2017 the growth changes to a more conservative trend as manufacturable solutions for these are not currently known. A range of Rent's bandwidth-exponents can be estimated for the years up to 2017, yielding exponents between 0.57 to 0.67, depending on pin-count growth rates.

We should note that these numbers may not account for the power and thermal constraints that are also present on the system that may prevent pin counts and pin bandwidth from growing as fast as this. Also, the size of I/O pads and drivers do not tend to change much with each process

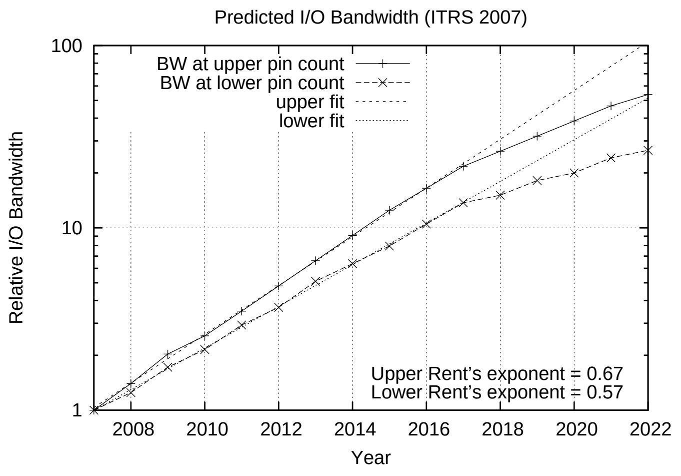

FIGURE 2: Predicted I/O Bandwidth

generation, so the exponential growth in pin count may have other costly repercussions for die size. Optical off-chip communication may help improve available bandwidth, though it is currently unclear what technology will be manufacturable. On-chip power might be usefully reduced by having an external optical power supply (i.e. an external source) and only modulating the photons on-chip. It is unclear just how effective such an approach would be since the technologies are immature, but it is unlikely to be a panacea.

3. LOCALITY OF DATA

Currently we use cache memories to keep state being worked on close to the processor. The concept is an old one and was summed up well by A.W. Burks, H.H. Goldstine and J von Neumann in their *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument* (1945):

> *“Ideally one would desire an indefinitely large memory capacity such that any particular word would be immediately available. We are forced to recognise the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.”*

Caches currently rely on simple statistical properties of data access patterns. Cache hit and miss rates have been very thoroughly studied, and yet their effectiveness is far from perfect. For example, Figure 3 shows the level 2 cache utilisation for a typical modern microprocessor. The utilisation metric refers to cache lines which are still holding “live” data, i.e. data that will be accessed again before the cache line is flushed from the cache. One can see that for many benchmarks over 80% of the cache is holding “dead” data, i.e. data that will never be looked at again.

The L2 cache is the primary mechanism for reducing off-chip data movement. It is also a very large structure (30% to 50% of the chip area) so it’s surprising that it is used so inefficiently.

The trend toward increased communication power in relation to computational power, and the ineffectiveness of caches, leads us to the observation that:

> *We must undertake more computation to reduce communication.*

Since simple statistical properties are insufficient, we will need greater understanding of algorithms and data structures to better manage data movement in systems. One approach is

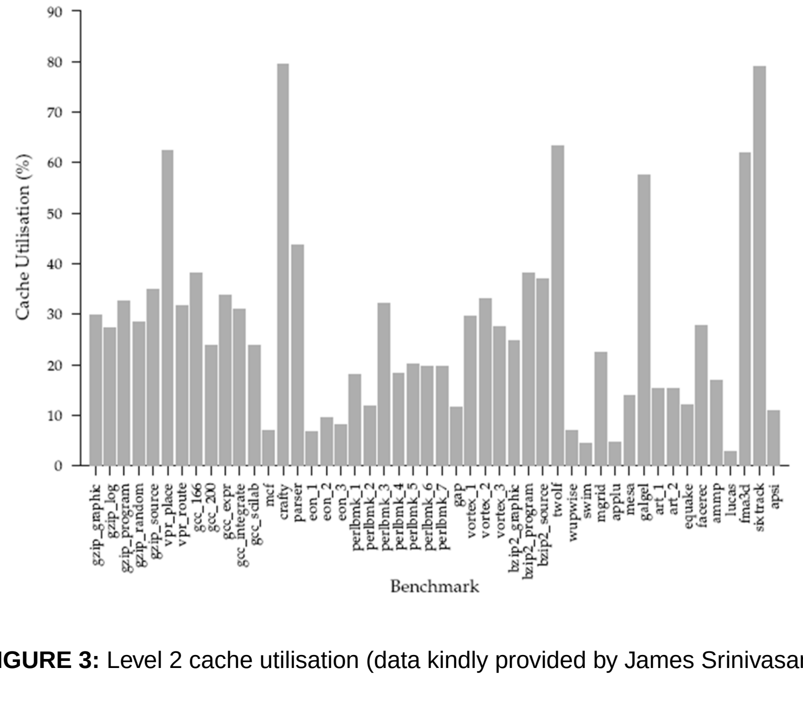

FIGURE 3: Level 2 cache utilisation (data kindly provided by James Srinivasan)

the use of software based prefetching [3] to reduce cache misses. This can be extended by running a dynamic helper thread which uses speculative computation techniques to predict what data needs to be prefetched quite some time in the future [13, 21]. It is interesting that these techniques undertake substantially more computation to improve communication scheduling, but they do not reduce the amount of communication and often result in extra communication fetching extra instructions and miss speculated data. None-the-less, it is an interesting to note that the increase in computation to improve the scheduling of communication does improve performance.

4. EXTERNAL COMMUNICATION CONSTRAINTS

Caches try to keep data local and on-chip, but as was seen in the previous section, cache miss rates exhibit diminishing returns for increases in cache size. As these cache misses lead to external memory I/O, the demand for external memory bandwidth per core may not readily be decreased by increasing cache size alone without exploiting local communication with other cores. Unfortunately, even as the number of cores increases, technology scaling also imposes important limitations on the volume of communication with external memory.

From the predicted I/O bandwidth (Figure 2) we can derive the external memory bandwidth per core (Figure 4), available on average to each core, assuming the number of cores doubles every *two* process generations, which the ITRS deems more representative of current power-limited design trends. We observe the exponential drop in available bandwidth that can impose a significant bottleneck in performance. This shows that cores cannot feed their bandwidth from external I/O and instead communication from other cores on-chip is required to make up the shortfall. We also note that this external bandwidth has considerably larger communication latencies and consumes more power than on-chip communication. Thus, algorithms must exploit communication locality between cores/threads so as to avoid communicating off-chip. These trends are considerably more marked if we allow a doubling of cores every process generation, as normal process scaling permits, instead of every two process generations.

There are interesting implications for systems and software programming. The notion that if we had hundreds of cores that we would merely run hundreds of separate applications is not sensible, even if users wanted to do so, because their external communication needs would be unlikely to scale.

Algorithms which assume simple independent data-parallel operations distributed across cores also effectively operate like entirely separate cores that need their own allocation of I/O bandwidth. While a lot of focus in parallelism so far has been on finding or specifying *independence* between data, especially in loops, and exploiting this as data-level parallelism, we see that this strategy

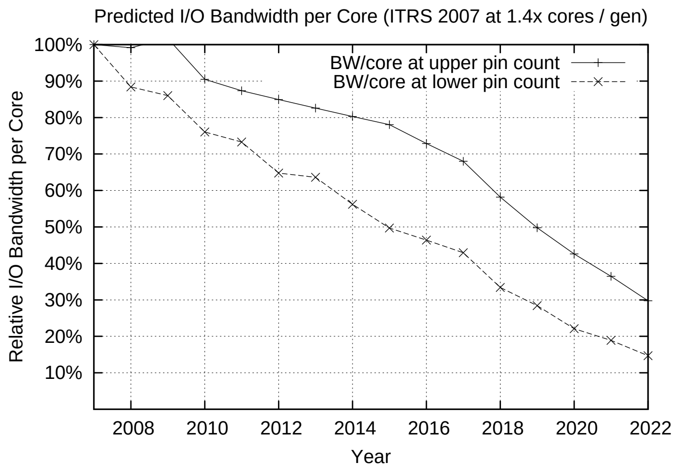

FIGURE 4: Predicted I/O Bandwidth per Core

may not be scalable to large numbers of cores due to these external I/O constraints. In traditional multicore architectures, external memory is effectively employed as a large communication crossbar. Indeed, by using such an approach, many algorithms can be factored into simple data-parallel operations with external memory taking care of communication complexities. For example, each computation stage of the Fast Fourier Transform is independent and data-parallel, however the communication between stages is complex and (external) memory is often used to act as the communication medium. Similar approaches are used for matrix operations and many other algorithms. This factoring of algorithms into batch jobs of independent computation which are glued together by (external) interdependent memory operations is not scalable. Instead, to achieve scalability and locality, we believe fine-grain local computation and communication between cores is required.

Stream processing goes part-way to addressing this by factoring stream-processing stages into cores with explicit communication between them. That is, their interdependence is explicitly mapped into a linear (one dimensional) or near-linear inter-core communication graph. However, with a growing number of cores, this block level partitioning only goes so far. Additional techniques are needed for further utilisation of cores with efficient inter-core communication.

For loops, software pipelining [18, 14] is an approach that essentially takes the dependency graph of operations in a loop and distributes them across cores to form a pipeline across multiple tiles. It is an effective technique for fine-grain parallelism.

Affine Partitioning [15, 16] is another approach that partitions algorithms with affine dependencies, typically within nested loops. It does so by converting the affine dependency structure into a set of linear equations and finding provably minimal communication solutions. The solution results in an instruction sequence in space (across a mesh of tiles) and in time (by cycle count) with a set of communication patterns. Software pipelining can then further extend the parallelisation.

Not all parallelisation approaches, however, result in partitionings with better internal versus external communication utilisation. Thread Level Speculation (TLS) [20] techniques on loops rely on very high independence between blocks of loop iterations for speedup, and thus appear very close to data-parallel in nature. They then allow for some interdependencies by aborting and restarting threads upon encountering dependency conflicts, with obvious penalties. Thus when operating on large datasets, such TLS techniques may also be significantly limited by external I/O bandwidth.

As external bandwidth, power and latency constraints motivate us to further exploit internal bandwidth, we envision that many more techniques will be developed for interdependent fine-grain communication and parallelism.

5. VIEWS OF COMMUNICATION

In this section we look at several different abstractions of interconnect, starting with the virtual interconnect of the NoC, and moving onto temporal and spatio-temporal extensions.

5.1. NoC — Virtual Interconnect

Just as logic placement tries to minimize delay and congestion, especially for critical paths, so too will *mapping* of software to cores on-chip. In fact, similar to how random placement of logic would result in untenable wiring, so too would random mapping of software result in unacceptable communication costs. Thus, exploiting locality is essential for software running on a CMP, supporting the emergence of Rentian behaviour. We note an important difference to traditional Rent's behaviour — the Rent's *bandwidth-exponent* is dependent on the software application that is running, as well as the logical topology of the NoC connecting cores. Indeed, Rentian behaviour cannot emerge unless the software, the NoC topology and the mapping process allow it — but for scalability reasons, we believe that this behaviour will emerge. Indeed, recent work by Heirman et al. [10], using the SPLASH-2 parallel benchmark, has demonstrated that such Rentian statistics can emerge on a CMP system of 16 to 64 cores, even though the communication is *implicit* through the cache-coherence mechanism of shared memory.

We already demonstrated some uses for a Rent's rule approach — first by predicting the scaling requirements of NoC, and then by generating a hop-length distribution [7] — the equivalent of a wire-length distribution. We focused on characterising and comparing routing approaches for fault-tolerance. We noted that the probability of a packet reaching its destination depends not just on the routing algorithm, and the reliability of the router, but also how far it needs to travel — the greater the distance, the greater the chance of encountering a fault. This gave us some interesting insights into fault-tolerant design. We also used the distribution to compare the congestion impact of handling a fault under two different schemes. We expect many more uses of hop-length distributions, such as for power, delay and perhaps even critical path estimation, all aiding in the evaluation and design of NoC.

5.2. Temporal Interconnect

It is sometimes helpful to take a step back and re-examine familiar objects with a new perspective. Using our communication-centric focus, we note that data doesn't just traverse space, it also traverses time. An application merely utilises memory to transport data from one moment in time to another. So we observe that:

> *Memory serves as a form of temporal interconnect, linking sources with destinations.*

We note that at one extreme, *register files* act like wires, since they typically connect a fixed source to fixed destination(s). However, at another extreme, memory can be used as a look-up table (LUT), replacing entire functions. Both extremes can be united by the view of memory acting more like a high-radix switch, routing multiple sources to multiple destinations. Thus what may seem like fundamentally different concepts — memory and NoC, may not be so different after all.

Temporal communication has many similarities to spatial communication. The greater the distance of communication, the greater the cost, as storing values takes up resources (and energy) for how ever long it is needed. We note that memory is divided into multiple tiers of service, with different area, power and performance penalties, of register-file, L1/L2/L3 cache, external memory and even virtual memory. Communication is automatically routed between these tiers using statistical techniques to minimize costs, and when there is too much congestion in one tier, it overflows into the next tier. Indeed, the properties of this traffic are so important that most on-chip area is currently taken up by memory — our temporal interconnect.

5.3. Spatio-Temporal Interconnect

We observe that once we start viewing memory as an additional interconnect, or as another form of network:

> *We can take the separate concepts of NoC and memory and unite them into a single spatio-temporal view of interconnect.*

We believe that this intuition allows us to see new ways of doing things, or leveraging analytical tools in one domain for the other. For example, it may be beneficial to relieve local temporal congestion by routing data spatially, depending on neighbouring temporal congestion. In the spatio-temporal domain we expect to see a trade-off between temporal locality and spatial locality. As a very simple example, looking at Figure 5, if we take computation that naturally has a 2D graph embedding among a chain of processors (on the left), and restrict it to only a 1D embedding, then the local spatio-temporal communication is replaced by longer temporal-only links.

It may even be possible to develop analytical tools for deciding how much area to allocate to memories versus NoC for power or performance concerns. We believe that being able to predict the properties of such spatio-temporal communication, and exploiting it, will be important as we scale to massively multicore CMPs.

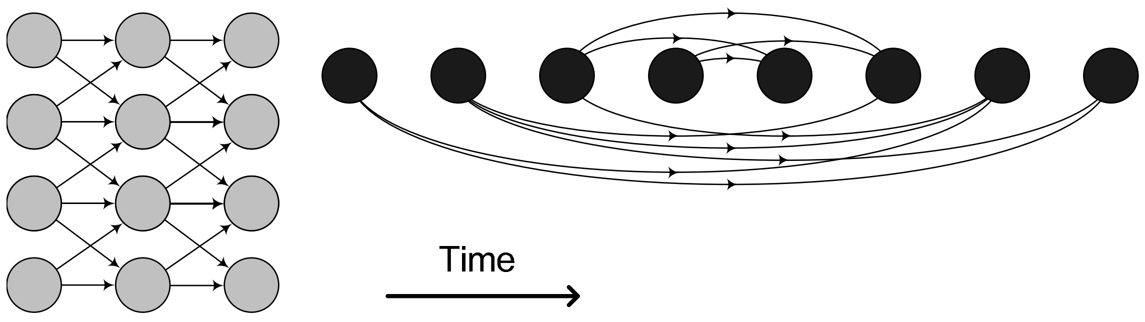

FIGURE 5: Simple example illustrating the trade-off between spatial and temporal locality. A graph is mapped to four processors on the left (grey), and one layer of its communication is mapped to a single processor on the right (black). The spatio-temporal distances are clearly affected by the mapping.

6. COMMUNICATION CONSTRAINTS IN SOFTWARE

Chip MultiProcessors (CMPs) are a somewhat different domain to traditional multi-chip multiprocessors due to their relatively much greater on-chip bandwidth and lower latencies of communication between cores, enabling fine-grain parallelism to be exploited across the tiles. We have seen the introduction of many such multi-core solutions already. This trend will only continue and it has been argued that we may even expect CMPs with over a thousand simple cores at the 30nm technology node [1].

In this section we examine the complexity of communication in software, noting some self-similar behaviour, and explore what this might mean in the massively multicore CMP era.

6.1. Communication vs Computation

The on-chip position of where computation is performed and data is located can no longer be ignored in the multi-core domain. The particular physical embedding of computation and data, and their physical locality to one another, can directly impact the time and energy taken due to communication.

A simple example can be seen if we consider the traversal of a balanced binary tree data-structure which we ordinarily think of as taking time $O(\log n)$. Let us suppose that the binary tree data structure is too large to fit on one processor tile, and so overflows to a 2D neighbourhood of on-chip tiles. For simplicity we will also assume that each leaf node is equally likely to be accessed, and the communication time is proportional to traversal distance. If each tile can perform the traversal computation, then we don't need to go back and forth with the root node and an embedding maximally employing locality leads at best to $O(\sqrt{n})$, whilst a random embedding leads to $O(\sqrt{n}\log n)$. We can do even better, on average, if the access patterns are not uniform by trading off the physical locality of frequent accesses to those of infrequent ones.

What is important to note here is the large asymptotic gap between the traditional computational complexity of $O(\log n)$ which implicitly assumes an $O(1)$ communication cost, to the considerably larger $O(\sqrt{n})$ that is optimal for a two dimensional embedding with communication costs factored.

6.2. Fractal Dimensionality

It is interesting to examine the communication characteristics of software, and in particular, whether or not software exhibits self-similar or *fractal* behaviour [8]. Prior work by Concas et al. [5] investigated whether the relationships of *objects* was fractal. In such a view, nodes represent classes, and edges represent their object-oriented relationships. They analysed several large object-oriented Java projects, showed fractal scaling behaviour and measured the dimensionality. However, we are more interested in the actual communication between instructions/functions/blocks, rather than the programmer's view of object relationships.

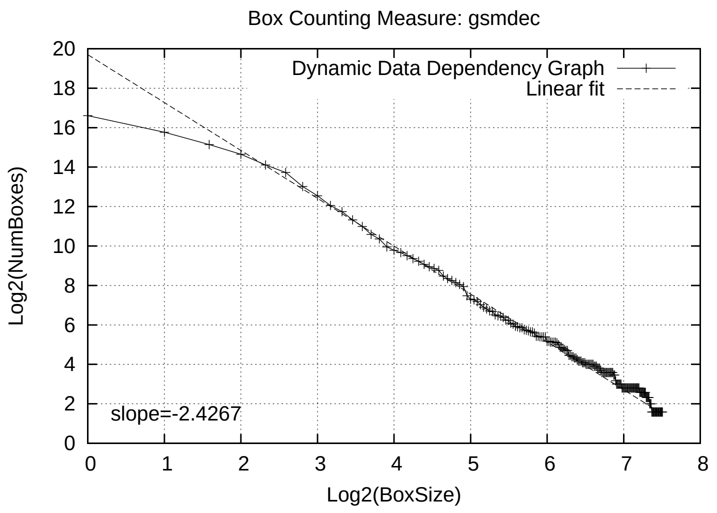

FIGURE 6: Box-counting measure of a *GSM-decoder* benchmark exhibits a power-law relationship between box-size and box-count. This indicates that the dynamic data-dependency graph for this algorithm exhibits fractal communication with a dimensionality, equal to its slope, of approximately 2.4.

We can take x86 instruction traces from benchmarks to generate dynamic data dependency graphs. The edges in such graphs correspond to communication of operands from one instruction to another. We can then do a *box-counting* analysis similar to the technique by Concas et al. [5], where we tile the graph with boxes of maximum length l, and then count the number of boxes. For example, in a 3-D mesh, the number of nodes inside each box will tend to grow by the cube, and so the number of boxes will tend to shrink by l^3. For fractal graphs we would expect to see linearity, on a log-log plot, across a continuum of box lengths, whereas if we look at random graphs such as of the Erdös-Rènyi variety, their slopes exhibit a rapidly accelerating decrease in box-counts with box-length.

By extracting graphs from x86 instruction traces of benchmarks, we have shown that many indeed exhibit fractal communication. Looking at Figure 6 we see an example of this behaviour for the *GSM-decoder* benchmark. The slope here indicates a communication dimensionality of approximately 2.4.

We note that the fractal behaviour of communication holds across multiple levels of abstraction — whether between instructions, functions or higher assemblies of code. Thus, in a CMP setting, we believe that regardless of the level of instruction mapping, communication between cores will probably also involve fractal communication patterns. The fractal properties of these graphs, impose constraints on the nature of the complexity of communication and locality that we can expect. Prior work has shown that Rent's rule emerges when higher dimensional graphs are embedded into lower dimensional surfaces, and that the Rent's exponent can also be directly related to the dimensionality [19]. As we enter the massively multicore CMP era, we expect that the VLSI community's work in modelling equivalent constraints and self-similarity with Rent's rule, may be leveraged in characterising communication in software as well.

6.3. Reducing Complexity

It is quite easy to increase the quantity and complexity of communication by the insertion of new links, or by mapping communicating blocks to disparate regions. However, it is an altogether more difficult task to reduce these. We acknowledge that the connectivity within current software does not specifically optimise for communication so there is considerable room for improvement from such automated tools. Nonetheless it is encouraging that even without optimisation, communication exhibits fractal locality.

Compression, of course, is one technique to reduce the quantity of communication, but typically at the cost of latency. Increased intelligence in NoC or the memory hierarchy may improve the utilisation of communication resources. Then there are techniques that involve modifying the program itself. Given the much greater costs of communication versus computation, we also believe that we can trade communication within software for more computation, also within software. One idea is to replicate the computation in SW if the cost of replication is less than communicating the results. Transformations may also exist that reduce effective dimensionality, or that may be applied to reduce the degree of high-degree nodes, for example, by building virtual Steiner trees.

In our fractal-dimensional analysis of the *tiff2bw* benchmark, which converts images to black and white, we expected each pixel calculation to be independent of the others, and thus have a relatively simple communication structure. We were surprised to find very high-degree nodes, which further investigation revealed to be due to lookup-tables (LUTs) being used to speed up computation. In adapting this for a scalable CMP implementation, we would expect that either the LUT would be replicated across the cores, or that computation would replace the LUT instead.

We also note that mapping can be used to alleviate the impact of communication complexity in *some* parts of the graph — by having highly-interconnected parts mapped within a single core, thus utilising the much higher-radix routing of memory compared to that of spatial routing. This, however, comes at the expense of parallelism.

6.4. Length Distributions

We believe that the fractal dimensionality of software imposes constraints on the minimum achievable Rent's exponent and of the length distributions. We have already suggested a Rentian model for NoC spatial locality, but we have also argued that there are trade-offs between spatial and temporal locality. So let us return to examining temporal locality.

If temporal communication is expensive, then there are possibly some incentives for it to be minimised — might we expect some statistical similarities for temporal distributions as we do for spatial ones? There are some indications that the distribution of interconnections may roughly follow a power-law distribution. Recent work by Hartstein et al. [9], has observed such a distribution for cache-line accesses. Caches exhibit a long-observed but unsatisfactorily explained scaling law, where doubling the cache size reduces the cache misses by roughly $\sqrt{2}$. Hartstein et al., derived analytical cache models and showed that this scaling behaviour appears under a power-law distribution, and is directly related to the power-law exponent. Of course, in the VLSI domain, we are more familiar with approximate power-law length distributions as being a feature of Rentian behaviour — could implicit temporal constraints be producing this behaviour?

Although Hartstein et al. [9], examined the distribution for cache-lines, their view involves address-locality as well as temporal locality. We were primarily interested in the more-fundamental communication graph requirements of instructions, whereas theirs relates to a particular embedding and clustering in address-space. Thus, using the same traces as in Section 6.2, and with instruction count as a measure of distance, we looked at the distribution of temporal distances for a suite of benchmarks. Temporal distances were measured independent of the means of communication — be they via register files, stacks, caches, or external memory. Interestingly, we found that many benchmarks exhibited evidence of approximately power-law behaviour. Even the innocuous Unix command *ls* seems to do so, as seen by its frequency-length relationship on a log-log scale (Figure 7).

7. CONCLUSION

With the growing costs of communication compared to computation we believe that the complexity of communication for an algorithm will become a far more accurate predictor of performance than computational complexity. As a consequence, we see a trend toward more intelligent, managed use of communication resources.

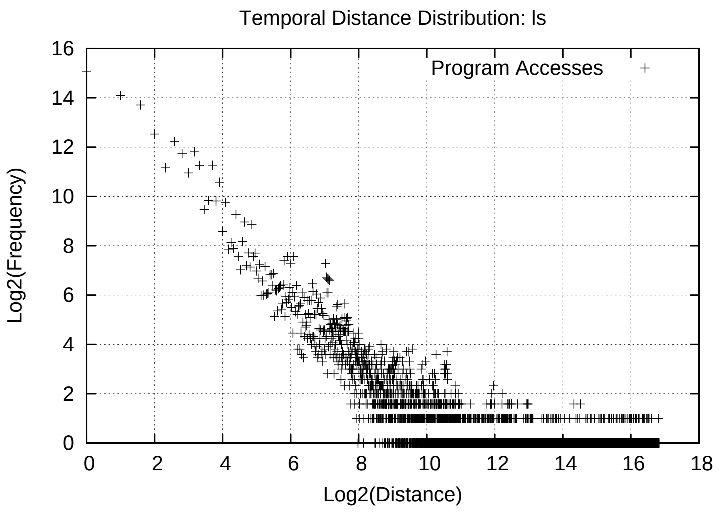

FIGURE 7: Temporal distance distribution of a trace for the Unix command *ls*, seems to exhibit approximately power-law behaviour.

At a computer architecture level we see the use of Networks-on-Chip (NoCs), which transform physical interconnections between cores into virtual ones. Nonetheless, communication requirements will grow in this virtual domain. It is important, then, to understand and characterise how these virtual interconnects are used. In a CMP setting, such interconnect is due to the communication of software between multiple cores, and external memory. Due to the slower growth rate of external I/O and its higher latency and energy costs, keeping communication on-chip is critical to power and performance. To this end, we observe that some parallelisation techniques utilise internal communication whereas others merely exacerbate the problem. The limits of such internal communication are thus worth exploring.

By looking at the graphs of communication between instructions, we demonstrated that dynamic data-dependency graphs can exhibit fractal behaviour. This self-similar communication lends credence to the idea that Rentian statistics may hold for software running on massively multicore CMP.

We also introduced the concept of temporal communication whereby memory serves as a temporal switch routing data from one moment in time to another. Thus our separate views of NoC and memory can be unified into one of spatio-temporal routing. We found that many benchmark algorithms exhibited an approximately power-law distribution of temporal distances, further lending credence to fractal behaviour and also suggesting a possible Rentian-like characterisation.

We thus believe that the considerable work done in the VLSI domain to characterise and predict interconnect, might be leveraged to help understand and predict spatio-temporal communication in massively multicore CMPs. Indeed, we envision that with the growing dominance of communication-concerns, interconnect prediction naturally evolves into system-level prediction.

8. ACKNOWLEDGEMENTS

This work was funded in part by a grant from EPSRC (EP/D036895) and a studentship from the Gates Cambridge Trust.

REFERENCES

[1] K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gerbis, 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*, December 18 2006.

[2] A. Banerjee, R.D. Mullins and S.W. Moore. A Power and Energy Exploration of Network-on-Chip Architectures. In proceedings of the *First International Symposium on Networks-on-Chips*, May 2007.

[3] D. Callahan, K. Kennedy and A. Porterfield. Software prefetching. In proceedings of the *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 1991

[4] P. Christie and D. Stroobandt. The interpretation and application of Rent's rule. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* , 8(6):639–648, Dec. 2000.

[5] G. Concas, M. F. Locci, M. Marchesi, S. Pinna, and I. Turnu. Fractal dimension in software networks. *EPL (Europhysics Letters)*, 76(6):1221–1227, 2006.

[6] W.J. Dally. Computer architecture is all about interconnect. *HPCA Panel*, February 2002.

[7] D. Greenfield, A. Banerjee, J.G. Lee and S.W. Moore. Implications of Rent's rule for NoC design and its fault-tolerance, In proceedings of the *First International Symposium on Networks-on-Chips*, May 2007.

[8] D. Greenfield and S.W. Moore. Brief Announcement: Fractal communication in software data dependency graphs, In proceedings of the *Twentieth Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, June 2008.

[9] A. Hartstein, V. Srinivasan, T.R. Puzak, and P.G. Emma. Cache miss behaviour: is it sqrt 2? In proceedings of the *3rd Conference on Computing frontiers (CF06)*, pages 313–320, New York, NY, USA, 2006. ACM.

[10] W. Heirman, J. Dambre, D. Strooband, and J. Campenhout. Rent's rule and parallel programs: Characterising network traffic behaviour. In *SLIP*, pages 87–94, 2008.

[11] ITRS. *International technology roadmap for semiconductors — 2007 edition: Assembly and packaging.* Technical report, 2007.

[12] B.S. Landman and R.L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. Comput.*, 20(12):1469–1479, Dec. 1971.

[13] J. Lu, et al. Dynamic helper threaded prefetching on the Sun UltraSPARC/spl reg/CMP processor, In proceedings of *38th IEEE/ACM International Symposium on Microarchitecture (MICRO-38)*, 2005.

[14] F. Li, M. Kandemir, and I. Kolcu. Exploiting software pipelining for network-on-chip architectures. In proceedings of the *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006.*, 2-3 March 2006.

[15] S.-W. Liao, Z. Du, G. Wu, and G.-Y. Lueh. A code generation algorithm for affine partitioning framework. In proceedings of the *11th International Conference on Parallel and Distributed Systems*, volume 2, pages 17–21, 20-22 July 2005.

[16] A.W. Lim, G.I. Cheong, and M.S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. In proceedings of *International Conference on Supercomputing*, pages 228–237, 1999.

[17] S.W. Moore and D. Greenfield. The Next Resource War: Computation vs. Communication. In proceedings of the *International Workshop on System Level Interconnect Prediction (SLIP08)*, pages 81–86, 2008.

[18] G. Ottoni, R. Rangan, A. Stoler, and D.I. August. Automatic thread extraction with decoupled software pipelining. In proceedings of the *38th IEEE/ACM International Symposium on Microarchitecture (MICRO-38)*, 2005.

[19] H.M. Ozaktas. Paradigms of connectivity for computer circuits and networks. *Optical Engineering*, 31(7):1563–1567, 1992.

[20] J.G. Steffan, J.G. Steffan, and T.C. Mowry. The potential for using thread-level data speculation to facilitate automatic parallelization. In T.C. Mowry, editor, *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pages 2–13, 1998.

[21] W. Zhang, D.M. Tullsen and B. Calder. Accelerating and Adapting Precomputation Threads for Efficient Prefetching, In proceedings of the *IEEE 13th International Symposium on High Performance Computer Architecture*, pages 85-95, 2007.