Design Space Exploration for Programmable Fabrics

Daniel Bates*, Robert Mullins*,[1]

* *University of Cambridge, Computer Laboratory, William Gates Building, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK*

ABSTRACT

This project aims to use an automated approach to thoroughly explore the design space of a massively-parallel processing fabric, made up of many hundreds or thousands of individual processing elements, and find the best possible architectural design for it. Analysis of results will dictate whether features should be included, and drive the generation of new regions of the design space.

KEYWORDS: design space exploration; reconfigurable; architecture; MPPA

1 Introduction

Computer processor technology has historically enjoyed a prolonged period of steady scaling: as the size of transistors decreased, more could be put onto a single chip and they could be switched on and off faster. Their supply voltage could also be decreased to make them more power-efficient. However, this scaling is now slowing down. It is no longer feasible to increase clock frequencies because of the associated increase in power consumption, and supply voltages are nearing their lower-limit. There is also a pressure to decentralise processor designs as thinner wires with higher resistances make communication increasingly costly. These changes are forcing computer architects to make a step-change in their designs, and many are moving to multiprocessing to maintain improvements in computation throughput. This approach, too, has many obstacles which need to be overcome if scaling is to continue.

Looking towards the future, the huge number of available transistors will force architects into designing systems which only use a small percentage of transistors at a time to remain within power budgets [Hor07], implying that some form of specialisation is required. As the end of CMOS approaches, there are also issues with process variation which need to be surmounted in order to implement reliable systems.

[1]E-mail: {Daniel.Bates, Robert.Mullins}@cl.cam.ac.uk

Embedded systems are constrained in three main dimensions: power, performance and cost. There are several existing approaches to satisfying these constraints, including:

- FPGAs: very flexible, but time-to-market is poor because it is difficult to map applications to them. Energy efficiency and performance depend on the task: FPGAs excel at parallel bit-level operations, but are relatively poor at executing sequential, control-flow intensive code.
- ASICs: very high performance and very low power, but at very high cost. ASICs only make economic sense if a very large number of chips are going to be manufactured.
- General purpose processors: cheap, since the same chip can be used in many different situations, running different code in each case. Time-to-market is also good because programs can often be written quickly in high-level languages. Energy consumption and performance are relatively poor, due to the overheads of flexibility.

This project aims to find a solution between an FPGA and a massively-multicore general purpose processor. Although there is much ongoing research in this area [Har01], the design space around tightly-coupled processors remains relatively unexplored. I aim to discover exactly which features are necessary, and the best possible implementations for them.

2 Architecture

The main aims of the architecture are simplicity and flexibility: the fabric should be robust and usable for any purpose. The network is a central design aspect, with efficient communication between components being key to high performance.

The architecture is designed to be minimal, with a few concessions to increase flexibility and speed up common cases. Having a basic starting point allows us to be results-driven, and only add features when analysis suggests that they are important.

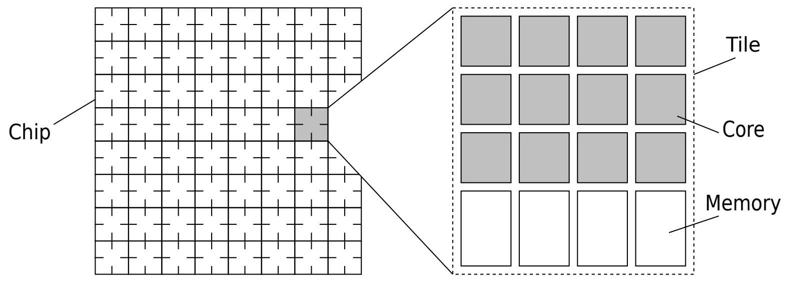


Figure 1: The homogeneous, hierarchical design of the chip.

The chip has a homogeneous and hierarchical design, as shown in Figure 1. It is composed of a number of *tiles*, all interconnected, each of which contains a number of cores and memories and a local interconnect. Any component on the chip can communicate with any other. The distribution of data and computation across the chip reduces the distance that information needs to travel, which in turn reduces the amount of energy consumed.

Each core (Figure 2) is capable of executing any program alone, but the usual approach will be to combine many cores to form a larger *virtual processor*, with each core running highly specialised code. This software specialisation aims to allow performance and energy consumption approaching that of ASICs [DBB+08], whilst retaining FPGA-like flexibility.

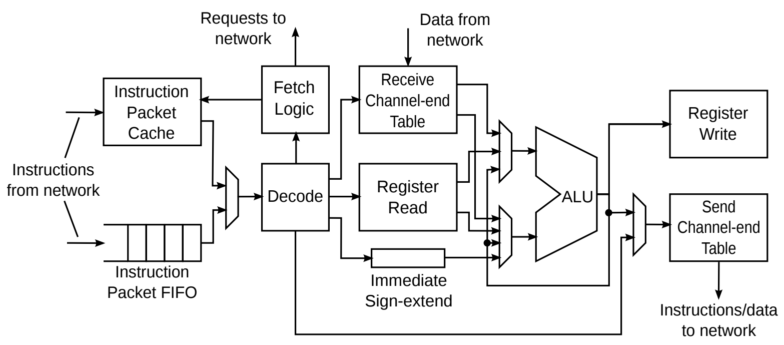
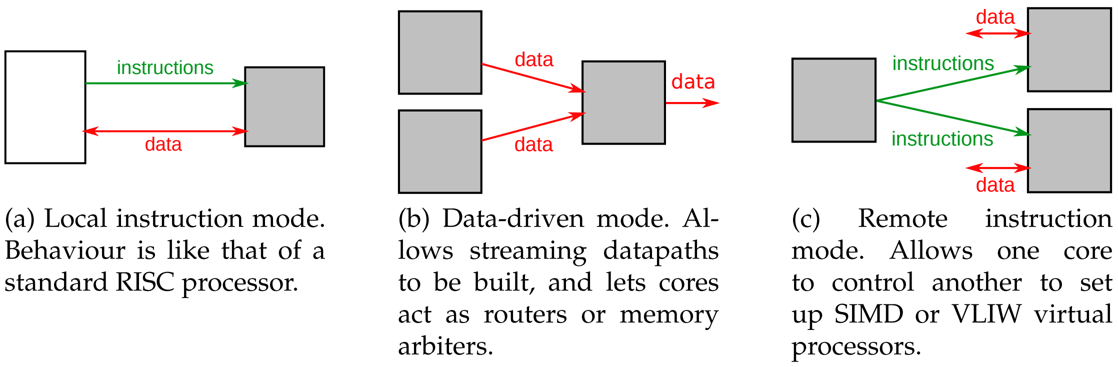


Figure 2: Main features of the four-stage pipeline. There is no dedicated Memory Access stage: data is put onto the network in the same way as if it was being sent to another core.

Some features initially added to the design are listed below. The features were introduced to increase flexibility, and allow interesting ways of mapping applications to the hardware. However, if testing suggests that one or more features are poorly-utilised, it is possible that they may be changed or removed to keep the design simple.

- Instruction packets: instructions are grouped according to their basic blocks, and the instruction cache has been modified to deal efficiently with variable-length packets.
- Register-mapped channel-ends: the network is directly connected to the pipeline, and it is possible for instructions to read data from or write data to the network.
- Support for indirect register access: allows a core to behave as a small memory on behalf of another core, and also increases the number of addressable registers.
- Specialisation of the memory system: a memory (or group of memories) could behave as a basic scratchpad or a more complex cache.
- End-to-end flow control: allows guarantees that there will be no deadlock in the network. If information is put onto the network, it is guaranteed to reach its destination.

The deep-interconnection between cores and the ability for one core to send instructions to another allows for three main execution patterns, as shown in Figure 3.



(a) Local instruction mode. Behaviour is like that of a standard RISC processor.

(b) Data-driven mode. Allows streaming datapaths to be built, and lets cores act as routers or memory arbiters.

(c) Remote instruction mode. Allows one core to control another to set up SIMD or VLIW virtual processors.

Figure 3: The three main execution patterns. The white rectangle represents a memory, and the grey squares represent cores.

These patterns aim to allow increased performance, even in sequential code sections, for example through implementation of small *helper engines*. Helper engines are cores or groups of cores dedicated to providing services to the main virtual processor, such as routing or prefetching data, allowing the main processor to work more effectively.

In summary, the architecture aims to achieve high performance and low power through specialisation of its many cores. Power is saved by bypassing functionality (*e.g.* cache tag checks) and reducing switching. The specialisation is done through software, however, so it is possible to use the same chip for many different purposes, reducing costs. The architecture will be a target for high-level languages, decreasing time-to-market, further reducing costs.

3 Plans

The results of initial design space exploration would dictate the direction of future research: it may be the case that an individual component becomes particularly interesting and worthy of further detailed study.

The network may be such a component: there are many ways that it can be partitioned, such as different levels of its hierarchy, or by separating control and data traffic. The network encompasses features such as flow control, buffering, channel width and topology, each of which can be varied for each partition.

Alternatively, there may be so many parameters to optimise that the exploration tool runs too slowly to reach an optimal solution in a reasonable amount of time. In this case, spending some time investigating new exploration approaches or optimisations would be useful. These could include additional heuristics to guide the search away from sub-optimal regions, combinations of different exploration algorithms, or investigations into faster simulation [EEDB06] or predictions [DJO08].

Automating the extension of the design space could be investigated, perhaps by going beyond parameterisation and implementing a technique for applying architectural transformations to the chip. These transformations would generally be more subtle than the effects of parameters, but could help find more-optimal, non-obvious configurations. This approach is particularly suited to the detailed investigation of an individual component.

References

[DBB$^+$08] William Dally, James Balfour, David Black-Schaffer, James Chen, Curtis Harting, Vishal Parikh, JongSoo Park, and David Sheffield. Efficient Embedded Computing, July 2008.

[DJO08] Christophe Dubach, Timothy M. Jones, and Michael F.P. O'Boyle. Exploring and Predicting the Architecture/Optimising Compiler Co-Design Space. In *CASES '08: Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 31–40, New York, NY, USA, 2008. ACM.

[EEDB06] Stijn Eyerman, Lieven Eeckhout, and Koen De Bosschere. Efficient Design Space Exploration of High Performance Embedded Out-of-order Processors. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 351–356, 3001 Leuven, Belgium, 2006. European Design and Automation Association.

[Har01] R. Hartenstein. A Decade of Reconfigurable Computing: a Visionary Retrospective. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 642–649. IEEE Press, 2001.

[Hor07] Mark Horowitz. Scaling, Power and the Future of CMOS. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, page 23, Washington, DC, USA, 2007. IEEE Computer Society.