Summary of FLEET 2006

Greg Gibeling
Friday December 15th, 2006
GDG07 – Summary of FLEET 2006

1.0 Introduction

In the past year FLEET has progressed from an intellectual exercise among a small group of graduate students and researchers into a full blown architectural specification for which real code can and has been written. In this memo, I intend to sum up particularly those changes which have occurred in the last 4 months to bring this about, along with my commentary on this progress, some suggestions for the future and concerns over these developments.

2.0 Progress

The most significant progress to FLEET since 2005 have been the creation of in and outboxes, which were originally invented to provide an abstraction for the simple flow control boundary between SHIPs and the switch fabric.

As a result of these developments it became possible to implement a standardized FLEET in a Xilinx FPGA using RDL. In addition, an integrated assembler for FLEET programs and some basic debugging hardware led to the creation of several running example programs for the first hardware implementation of FLEET.

One of the primary concerns in building FLEET has been the cost of switch fabric traversals, and instruction distribution. This has spawned a number of design decisions starting with standing and counted moves, progressing through records in the middle of the year, and ending with the design of more advanced boxes, with support for ZOMA moves, and token triggering.

With the introduction of these boxes, FLEET, a year later is a fully programmable, and reasonably usable architecture, to the point that I find it no harder to write FLEET assembly than x86.

Best of all, with the passing of two semesters and the hard work of a number of students and researchers there are now a total of 4 implementations of FLEET, two in hardware, two in software, a wide range of running example code and best yet, a stable instruction set architecture definition.

3.0 Concerns

Despite all of the major progress made in the last 12 months, there are still a number of concerns associated with the current design of FLEET.

First, as we have spent more time designing and perfecting the switch fabric to support more complex move instructions, like standing moves, we have increased the cost of switch fabric traversals. To me, this raises the concern the we may be in the middle of a design spiral whereby our attempts to compensate for the cost of traversing the switch fabric are actually increasing this cost. Without hard performance data, this is hard to judge, but it is still a very real concern that we must be mindful of.

This leads to the second clear problem with our work so far, as yet we have no hard performance data for any one of the four FLEET implementations. This is unsurprising, as we had no stable ISA until very recently, and yet now that we do, benchmarks must become an important part of our work on FLEET.

One of the key problems going forward is the current trend in computer architecture to use realistic programs to benchmark a design. While this is highly commendable in that it tends to produce more believable and relevant results, it has the downside that a high level programming abstraction must be available to implement these benchmarks.

Given that we still have no available high level programming languages, and not even a matching model, this may become a serious problem moving forward. The alternative of course is to implement these complicated benchmarks in FLEET assembly, a painful prospect.

4.0 Further Ideas

During the past year we have adopted a number of very clever ideas to improve FLEET, there remain several which have not been fully examined.

First of all, among the hardware implementations there is a relatively high cost to separate implementations of the instruction and data switch fabrics. This has led to the relatively simple suggestion that the two should be merged into a single simple packet switched fabric.

While the most recent hardware implementation of FLEET by Adam Megacz makes use of this as a design, we have, as yet, not fully evaluated the consequences of this decision. For example the memo GDG01 makes the suggestion that first class instructions would provide a better mechanism for everything from literals to data dependant branches to the interaction between control and dataflow, a subject which has been somewhat neglected with the advent of boxes from AM11. More important, 1st class instructions were intended to provide a way to allow the programmer to decide the sequencing guarantees they need from the fetch SHIP.

While first class instructions remain somewhat of a mystery to most, there is a clear need for virtualization, some kind of control transfer mechanism for function calls, and the design of Chip Multi-Processors, referred to as Flotillas.

While these three ideas have been the subject of memos in the last 4 months, they have received almost no serious investigation, as the ISA question remained open.

5.0 Conclusion

The largest and clearest conclusions from the above sections are that, one, we have completed a major step by finishing the ISA specification to a usable state, and two, testing and concrete performance measurements are now very much in order.

This means that the next step in our work must be to produce a series of working implementations of FLEET with realistic performance, in order to evaluate our design decisions against working hardware.

6.0 FLEET in 2007

Building on the conclusion of this memo, above, and the progress made on the RDL Compiler v3, I believe it is time to update the RDL implementation of FLEET to match the new ISA specification.

By using RDL, I believe we can more easily separate the performance of the design from its functionality allowing us to experiment with a wide range of designs for SHIPs, switch fabrics and more advanced concepts like virtualization and boxes.

Finally, in conjunction with larger FPGA platforms such as the BEE2, I believe we will be able to run concrete experiments with Flotillas fast enough that it is quite possible we may be able to produce a FLEET capable of being used as an actual computer terminal.