Guido Meardi
(meardi@ece.utexas.edu, until May 98)

FPGA-coupled Microprocessors: the rise of Morphware

A Literature Survey

Abstract

Microprocessors have been the dominant devices in general-purpose computing for the last decade. However, there remains a large gap between the computational efficiency of microprocessors and that of specialized computing resources, such as Digital Signal Processors and application-specific processors (ASICs). Reconfigurable devices, such as Field Programmable Gate Arrays (basically, reconfigurable ASICs), have come closer to closing that gap, offering very nearly the performance of ASICs (typically 10x to 100x performance boost), but with more than one application. On highly regular, high throughput computations, reconfigurable architectures are clearly superior to traditional processor architectures. However, in irregular tasks and in those with low throughput requirements, the traditional microprocessor organization is still more efficient than these reconfigurable devices.

The best solution, then, could come from combining the two opposite poles: by coupling a general purpose microprocessor with a reconfigurable logic array, we could clearly exploit the best of each solution. FPGA-coupled processors could swiftly execute computationally-intensive tasks while maintaining the flexibility of a programmable architecture.

The very low reconfiguration times that modern FPGAs feature, in particular, are now seriously opening the possibility of *dynamic* HW reconfiguration during the execution, adapting the system to the actual flow of computation.

This project will focus on the dynamic HW/SW partitioning problem. I will use Ptolemy, SDF graphs and the **MORPH** chip that the *Politecnico di Milano* is developing will be reference model. I will try to develop a suitable heuristic to solve the partitioning problem, considering the FPGA dimension, the instructions that can fit in the FPGA (with relative performances) and the relative reconfiguration times.

1. The Big Picture

1.1 What can we do with all those gates?

Everybody knows that effective device densities and IC capacity grow at an exponential rate. We are quite familiar with the progress of microprocessors, where performance increases roughly by 60% per year and the number of gates increases by 25% per year. At the current rate, we can expect to have over 12 million gates available by the end of the century, and possibly an astounding 1 billion gates in just 10 years. The current trend is in enhancing performance with additional, fixed functional units and reducing costs by integrating more of the system on a single chip (*system-on-a-chip*). It appears doubtful, though, that this is the most interesting use of the silicon real-estate becoming available. Much of the economy found in the production and use of microprocessors comes from their commoditization, while integrating fixed functional capacity could mean over-specialization. Moreover, microprocessors may continue including more memory, more FPUs, more ALUs and more system functionalities, but the fixed functional units simply won't provide a broad-based acceleration of applications in *proportion to the area these fixed units consume* [4]. For almost any application we can figure out solutions or modifications to the architecture that would significantly improve the application's performance, but these modifications would obviously differ from application to application. It's unlikely, then, that including these additions in a microprocessor with a broad application base would be the optimal choice.

Incorporating *reconfigurable logic* into a general-purpose processor appears to be the right choice: every application would be allowed to tailor the hardware to its particular requirements, with huge possible advantages in terms of performance. At the same time, this would allow the microprocessor to maintain its appeal across a broad range of applications, and that would clearly mean commodity economics, high volumes and low prices. This last point probably merits clarification: post-fabrication adaptability had a paramount role in the success of general-purpose microprocessor technology and will no doubt be determinant in the development of reconfigurable architectures. The very same general purpose IC can be used to solve different problems in different times, can be used to run applications that the designer of the chip couldn't conceive and enjoys economies of scale as well.

1.2 Configurable computing

What made the design of configurable computing possible was the design of new FPGAs (*Field Programmable Gate Arrays*) that can be configured extremely quickly.

FPGAs consist of arrays of configurable logic blocks that implement the logical function of gates. Both the logical functions performed within the blocks and the connections between the blocks can be modified by sending suitable configuration signals to the array. Coarser-grain FPGAs also feature fairly complex building blocks, like adders, multipliers, multi-port gates, etc.

The earliest FPGAs required several seconds or more to change their configuration, and that was perfectly acceptable for engineers that wanted to test a circuit design or for companies that wanted upgradable devices. Newer FPGA, instead, can be configured in

less than one millisecond, and in a couple of years configuration times could become as low as 100 microseconds [7].

With this kind of performance, we are allowed to think of processors that configure themselves on the fly, adapting to the software that is actually running and to the resources that it needs. Tasks could be swapped in and out of the processor as needed through reconfiguration of the FPGAs. Since most of the processing time for computationally-intensive tasks is spent in relatively small kernels (it is commonly known that 90% of the time is spent in 10% of the code), this kind of hardware acceleration could considerably improve overall performance, with boosts ranging from one to two orders of magnitude.

1.3 Current results

The field is somewhat new, yet there are a few noteworthy architectures that have already been or are being developed around the world. We often see supercomputer level performance from systems which cost orders of magnitude less, and this should tell us something about the importance of this field of research.

Villasenor and Mangione-Smith [7] developed, using reconfigurable technology, the fastest yet economical DES encryption system in the world (as per 1996). They also developed a single-chip video transmission system that reconfigures itself four times per video frame, and thus requires only a quarter of the hardware necessary for a fixed ASIC.

André DeHon [4][5][6], together with other researchers of the *Massachussets Institute of Technology*, has proposed and fostered the use of DPGAs (*Dynamic Programmable Gate*

Array) in programmable architectures. This evolution of FPGAs should significantly reduce the overheads associated with both processor-*PGA communications and array reconfigurations. Unlike normal FPGAs, where the function of each array element is fixed between relatively slow reconfiguration sequences, the DPGA array elements may switch rapidly among several, pre-programmed configurations. This rapid reconfiguration allows DPGA array elements to be reused in time without significant overhead [4]. Applications could preload multiple, specialized array configurations and then be able to adroitly switch among them: in a single clock cycle, which is on the order of tens of nanoseconds, the chip could swap configuration without erasing previously processed data.

The *BRASS Project*, at the *University of California Berkeley*, is investigating the possible applications of reconfigurable computing and is developing systems that combine microprocessors with FPGAs.

The *Defense Advanced Research Projects Agency* (DARPA) is financing a few applications involving *pattern matching*, which appears to be one of the most promising applications of configurable computing. A target recognition application, in order to function fast enough for military applications, needs to perform comparisons at the shocking rate of several trillion operations per second. This because all the pixels in the input image must be compared with all the pixels in thousands of templates. In a typical template, however, many pixels do not actually contribute to the final matching result, so that a configurable machine could simply omit them from its calculations (the pixels to omit would obviously differ from template to template, rendering reconfigurability is necessary).

Brad L. Hutchings and his research group from *Brigham Young University* developed the *Dynamic Instruction Set Computer* (DISC) [11]. DISC features a fairly simple approach to the partitioning problem and takes advantage of the National Semiconductor Configurable Logic Arrays (CLAys), able to *partially* reconfigure hardware resources. DISC uses partial configuration in order to provide custom-instruction caching, reduce configuration times and implement a global controller that steadily remains on the FPGA (conventional configuration methods would require the saving and restoring of program counter and register values every time a configuration occurs). Before initiating execution of a custom instruction, DISC queries the FPGA for the presence of the custom-instruction configuration . If the custom instruction is on the FPGA, execution is initiated, otherwise program execution pauses while the custom instruction is configured on the FPGA. Some techniques developed for DISC, such as the partial reconfiguration and the global controller, could be useful in addressing problems such as the use of FPGA-coupled processors with multitasking operating systems (see chapter 1.5).

The *HW/SW Codesign Group* of the *Politecnico di Milano* University, led by *Mariagiovanna Sami*, is currently developing architectures and tools aimed at improving the understanding and application of configurable computing. The **MORPH** chip, discussed in Chapter 2, is one of these projects.

The list could go on, but the number of pages is bounded...

1.4 Hw/Sw Codesign issues in configurable computing

1.4.1 The partitioning problem

Configurable computing is surely an exciting opportunity, but there are a few major

hurdles that must be overcome: the software challenges associated with this kind of technology, in particular, may be a lot more troublesome than the hardware issues.

One of the tricky problems associated with FPGA-coupled processors is that the process of mapping algorithms into FPGAs is not fully automated. Typically, programmers take care of identifying the algorithm (or a portion thereof) to be implemented in hardware, and then specialized tools convert the algorithm into an hardware description.

The problem of deciding what to do in HW and what to do in SW is actually a renown and fairly intricate *HW/SW Codesign* issue. In many cases designers would like to define a particular application at task level and only afterwards, considering the various constraints that they may have, choose how to partition the whole system. This approach is called HW/SW Codesign, and opposes the traditional habit of designing the HW first, and then programming the SW. By designing HW and SW in parallel, the final HW/SW split can be made after the evaluation of several alternative structures with respect to performance, programmability, manufacturing (recurring) costs, development (*una tantum*) costs, reliability, maintenance, time to market, etc. [3].

The HW/SW partitioning problem is not limited strictly to making a binary choice between a hardware or software mapping. Within a given mapping, every part can be implemented using different algorithms and synthesis mechanisms, that typically differ in area and delay characteristics. This leads to the joint problem of mapping the various parts of the system to hardware or to software and, within each mapping, selecting a suitable implementation: this problem is called *extended partitioning problem* [1].

Exact solutions to both the binary and the extended partitioning problem are clearly

intractable even for reasonably small problems. If we want to deal with them, we must rely on some kind of heuristic solution.

The need of an HW/SW codesign approach is typically felt during the design of embedded systems, where the use of custom silicon is more likely. These systems normally feature structurally simple (yet computationally-intensive) applications, usually well defined by restricted and non Turing-complete models of computation, such as *Synchronous DataFlow* (thoroughly discussed in [10]). These kind of semantics have the drawback of limited expressiveness (normally suitable for embedded systems applications, though), but in return are much simpler to deal with. Substantial effort has been put into finding polynomial time heuristics that solve the partitioning problem for applications described with restricted models of computation.

1.4.2 The SDF model of computation

Developed by E. A. Lee and D. G. Messerschmitt in the mid 80s, *Synchronous DataFlow* is a restricted form of the dataflow model of computation. In the dataflow model, an application is represented as a directed graph. The nodes of the graph, also called *actors*, represent computation, and the arcs represent data paths between computations. In SDF, each node consumes a fixed number of data items (*tokens*) per invocation and produces a fixed number of output samples per invocation: this is probably the most important characteristic of this model of computation. Thanks to the fixed number of input and output tokens, an SDF graph can be statically scheduled (if the graph is consistent). The drawback is that every kind of non-synchronous or data-dependent behavior outside the implementation of the actors is banned, so that the model is *not* Turing-complete.

A thorough research developed by A. Kalavade showed that, using SDF semantics and

introducing the notion of Global Criticality (a global look-ahead measure that estimates the time criticality of a particular node at each step of the algorithm), reasonable heuristics can be developed that solve either the partitioning problem or the extended partitioning problem [1].

1.5 Other challenges of reconfigurability

HW/SW partitioning is by no means the only technical challenge that reconfigurable architecture designers will have to face. As insightfully pointed out in [4], the space of design options is large. A certain number of tradeoffs will be necessary to deliver a balanced design, allowing effective acceleration and adaptation across a large range of applications. The key challenges include the *interfacing between the processor and the configurable logic*, the *grain-size of the FPGA*, the *area and pin allocation* and the problem of *multitasking and state interaction*. This last hurdle, in particular, appears quite daunting: the FPGA introduces a large amount of state associated with each computation in progress, and the overhead necessary to reconfigure contemporary reconfigurable architectures is such to make time-sharing systems quite simply impractical.

2. Future directions

This project will focus on **MORPH**, an FPGA-coupled architecture currently in development at the *Politecnico di Milano* University. At present, **MORPH** is a Very Long Instruction Word (VLIW) processor with a few standard hard-wired pipelines (load and store, fixed point, floating point) and an FPGA on-chip with space for two more configurable pipelines (Morphware). The maximum number of stages for each of the configurable pipelines is fixed and the latency is the same for both the hard-wired and the configurable pipelines. Every configurable pipeline has access to a set of private registers (as well as to the general register file).

Working with the SDF model of computation and with Ptolemy, I'll try, if possible, to exploit A. Kalavade's HW/SW Codesign results (GCLP and MIBS heuristics, described in [1]), adapting them to a configurable environment and to *dynamic reconfiguration* (i.e., the FPGA might be reconfigured on-the-fly during the computation). The key issues of the study, in particular, will probably include finding suitable *metrics* to address the problem and a suitable *goal function* to drive the heuristics.

For the sake of simplicity, I will initially focus on fine-grained SDF graphs, but if possible I will try to deal with a coarse granularity of computation as well.

Since the project is evolving rapidly, though, the actual aim of this paper could be slightly altered during the research.

References

[1] Asawaree Kalavade, *System Level Codesign of Mixed Hardware-Software Systems*, Tech. Report UCB/ERL 95/88, Ph.D. Dissertation, Dept. Of EECS, University of California, Berkeley, CA 94720, September, 1995.

[2] W.-T. Chang, A. Kalavade, and E. A. Lee, *Effective Heterogeneous Design and Cosimulation*, chapter in Hardware/Software Co-design, G. DeMicheli and M.G. Sami, eds., NATO ASI Series Vol. 310, Kluwer Academic Publishers,1996.

[3] A. Kalavade and E. A. Lee, *Hardware/Software Co-Design Using Ptolemy - A Case Study*, Proc. of the IFIP International Workshop on Hardware/Software CoDesign, Grassau, Germany, May 19-21, 1992.

[4] André DeHon, *DPGA-coupled Microprocessors: coomdity ICs for the early 21st Century*, Ph.D. Dissertation, Massachussets Institute of Technology, January 1994.

[5] André DeHon, *The Next Generation for General-Purpose Computing Machines*, http://www.ai.mit.edu/projects/transit/reconfig.com/next_generation.html, October 1995.

[6] André DeHon, *Directions in General-Purpose Computing Architectures*, http://www.ai.mit.edu/projects/transit/reconfig.com/rc-viewpoint.html, 1995.

[7] J.Villasenor, W.H. Mongione-Smith, *Configurable Computing*, Scientific

American, June 1997.

[8] Hw/Sw Codesign Group, *POLIS web site*, University of California, Berkeley, http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html.

[9] *BRASS Research Group web site*, University of California, Berkeley, http://HTTP.CS.Berkeley.EDU/projects/brass.

[10] S.S. Bhattacharyya, P.K. Murthy, E.A. Lee, *SW Synthesis from DataFlow Graphs*, Kluwer.

[11] M. J. Wirthlin, B. L. Hutchings, *DISC: The dynamic instruction set computer*, in Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, John Schewel, Editor, Proc. SPIE 2607, pp. 92-103 (1995).