

Programming Soft Processors in High Performance Reconfigurable Computing

Andrew W. H. House and Paul Chow
Department of Electrical and Computer
Engineering
University of Toronto
Toronto, ON, Canada M5S 3G4
{ahouse, pc}@eecg.toronto.edu

ABSTRACT

This paper examines the ways in which soft processors can contribute to high performance reconfigurable computing systems, and the challenges this presents. To overcome these challenges, the use of new programming languages and an unconventional intermediate representation is advocated, to support the automatic partitioning of an application to make use of soft processors and other available resources.

Categories and Subject Descriptors

D.1.3 [Programming Techniques]: Concurrent Programming—*parallel programming*; D.3 [Programming Languages]: Miscellaneous

1. INTRODUCTION

Recently, as traditional microprocessors have started plateauing in terms of performance, there has been much interest in the use of accelerators for high performance computing (HPC). While much of this focus has been on the use of graphics processing units (GPUs) for general computation, there has also been renewed interest in the use of reconfigurable hardware for application acceleration.

Reconfigurable hardware has much potential in this arena – significant application speedups are possible, and it still allows flexibility. The difficulty, as always, is in the tools – hardware design is not a skill most HPC users have, and even high level synthesis tools still require deep understanding to get good results. Soft processors can alleviate this difficulty by allowing users to employ reconfigurable hardware with a software programming model.

At first glance, however, soft processors seem incongruous with HPC, since HPC is focused on application acceleration, and soft processors typically pale in direct comparison to their hardwired brethren. However, if we leverage the other great strength of soft processors – their flexibility – we

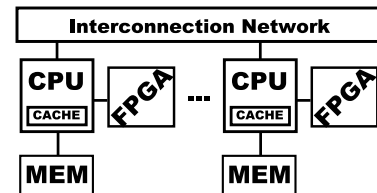


Figure 1: FPGA as co-processor

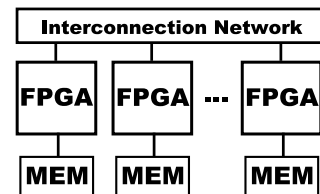


Figure 2: FPGA multiprocessor

can see they have much to offer, both directly (as application accelerators) and indirectly (as control and interfacing systems).

2. HIGH PERFORMANCE RECONFIGURABLE COMPUTING

As discussed in [6], reconfigurable computing elements (typically FPGAs) can be employed in HPC in three general ways: as a co-processor subordinate to a CPU (such as the Cray XD1 [11]), shown in Figure 1; as a primary computing element in a specialized reconfigurable platform (such as the BEE2 [2]), shown in Figure 2; or as a first-class computing element in a heterogeneous platform (facilitated by technology like DRC Computer's RPU110 [4]), shown in Figure 3.

These types of high performance reconfigurable computing

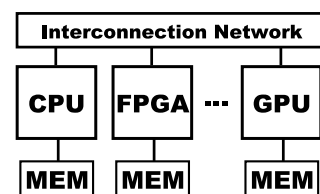


Figure 3: FPGA in heterogeneous multiprocessor

(HPRC) systems reinforce the question of whether soft processors are worthwhile in this environment, since there are high performance CPUs in the heterogeneous systems, and even an FPGA-only system might have embedded hard processor cores. As suggested above, however, taking advantage of the inherent flexibility of soft processors can make them invaluable components in an HPRC application.

Current off-the-shelf soft processors (such as the Xilinx MicroBlaze or Altera Nios-II) are targeted at embedded applications rather than HPC applications. However, more heavily-customized soft processor systems could be leveraged for application acceleration, control, and interfacing in HPRC.

2.1 Application Acceleration

Soft processors can be used to accelerate applications in several ways, most obviously through the use of application-specific soft processors. This could take the form of special instructions added to an otherwise conventional architecture (as is possible with the MicroBlaze or Nios-II, or ASIC solutions such as those from Tensilica and others [13]). Another approach, used by Mitronics, is to build an application-specific soft processor (the Mitrion Virtual Processor) out of a set of predefined elements to implement programs provided in their Mitrion-C language [7]. These approaches can generate significant speedup for many applications.

However, even general soft processors can be used for accelerating applications, by exploiting massive parallelism. There is no need for a soft processor to compete on a one-to-one basis with hard processors, when dozens could be implemented in a large FPGA. This is especially effective for applications that perform a lot of computation on a relatively small amount of data – a soft multiprocessor solution can take advantage of the massive on-chip memory bandwidth of an FPGA (on the order of 0.5 TB/s) to offer superior performance over hard processors with limited off-chip memory bandwidth (approximately 25 GB/s). Such applications are a subset of all HPC, but are important.

2.2 Control and Interfacing

Apart from accelerating applications directly, soft processors can also be used in HPRC to run control and interfacing software. This is especially useful for irregular and control-intensive applications, where soft processors can be used alongside hardware kernels implemented in a reconfigurable device. The soft processor can run control software that coordinates the heavy processing of the tightly-coupled kernels, thus simplifying the hardware design and leaving any hard processors in the system free for other tasks.

Similarly, a soft processor can be used to provide a reconfigurable platform with superior user interaction, host communication, and interfacing between different components and protocols. In these roles, soft processors facilitate the better use of other reconfigurable resources.

3. PROGRAMMING MODELS FOR HPRC

Given that soft processors have a useful role in HPRC, the question then turns to how best to program those soft processors and systems using them. Any useful HPRC system

will have multiple FPGAs, and thus multiple soft processors are likely, both within a single FPGA and spread across the whole system. That system may well incorporate hard processors and other, non-reconfigurable accelerators such as GPUs, and suddenly programming soft processors is no longer a trivial problem.

Existing work on programming HPC systems can be leveraged to a limited extent. Despite extensive research into dataflow and functional languages, stream computing, parallel object models, and data-parallel programming, most HPC applications are written using shared memory or (more commonly on larger systems) message passing for communication between parallel processes. But even these common paradigms run into scalability problems in HPC – though message passing is very flexible, the only style of message passing program that scales easily is the extremely regular, data-parallel, single-program multiple-data (SPMD) approach. More recent work on partitioned global address space (PGAS) languages such as UPC and Co-Array Fortran [3] builds on this, offering a shared memory model for distributed memory systems.

A slimmed-down version of the Message Passing Interface [9] library, called TMD-MPI [12], has been developed for use in HPRC, allowing MicroBlaze soft processors to send messages to each other, the PowerPC hard processors in certain Xilinx FPGAs, and custom hardware kernels running in the FPGA. MPI programs can be compiled to run in soft logic with minimal change.

Unfortunately, the SPMD and PGAS approaches start to lose their effectiveness when deployed in a heterogeneous computing environment where some processing elements may be entirely unable to run general software. For example, if custom hardware kernels are used with a TMD-MPI application, the programmer has to explicitly determine the number of kernels, and explicitly initiate all communication between kernels and processors. While some applications will lend themselves to a repeatable, regularized set of operations and communications, that is not true of all applications, and thus once again scalability becomes a concern.

Tools like RapidMind [8] and Intel's Ct [5] (and the forthcoming Apple-proposed OpenCL) offer solutions to this in a conventional heterogeneous environment, allowing specially-written data-parallel programs to be partitioned by a runtime system over multiple cores (and onto GPUs, in the case of RapidMind). This may work for a desktop computing environment, but in HPRC binary portability is not an issue so much as overall performance, and so this route is not likely to be as effective in programming soft processors in HPRC systems.

Thus, to make the best use of soft processors in an HPRC system, we need a tool that can automatically:

1. partition an application across a heterogeneous platform,
2. identify parts of an application that would best be served by a soft processor,
3. generate necessary application-specific soft processors,

4. generate the software to run on the hard and soft processors in the system,
5. generate hardware kernels where needed,
6. manage communication between parallel elements.

Nothing in the set of existing programming paradigms can satisfy all these requirements. The HPC models that scale well, for example, do not easily map into a heterogeneous computing environment. Tools like Mitrion-C [10], which can generate application-specific soft processors for HPC applications, still require the user to partition the application and manage the design of the overall system. Thus, once again, as in [6] we are forced to conclude that a new programming model is required.

4. A NEW PROGRAMMING APPROACH

In our previous work [6], we investigated programming models for FPGA-based HPC systems. This work included an extensive survey of existing programming models and languages, which were evaluated against nine criteria:

1. heterogeneity,
2. scalability,
3. synthesizability,
4. assumption limited system services,
5. support for varied types of computation,
6. exposure of coarse- and fine-grain parallelism,
7. separation of algorithm and implementation,
8. independence from architecture,
9. execution model transparency.

Based on that survey, we concluded that no existing programming model was appropriate to programming emerging heterogeneous reconfigurable HPC systems. Adding consideration of soft processors into the mix does not improve the situation – while we have seen that existing models *can* work to program soft processor systems, making the best use of such HPRC systems will require a new approach.

This new approach would likely be a new language (or an adaptation of an existing language) designed to meet the requirements presented above. Such a new language might include some of the beneficial features identified in [6], such as data-parallel operations, region-based array management, or a global view of memory. It would be extremely high level, meant for writing simulations rather than systems or applications programming. To ensure portability and a high level of abstraction, communication and synchronization will be implicit, and there will be an emphasis on libraries and the use of functions to allow programmers to write algorithms, not platform-specific implementations.

The new language would also have to be compatible with the requirements for effectively using soft processors in HPRC systems that we developed in section 3.

4.1 The Armada Programming Language

The Armada programming language is currently under development to meet this need. The goal of Armada is to abstract away from platform-specific knowledge of the system, which will improve programmer productivity and code portability. It is a data-parallel PGAS-style language that provides a number of higher-level operators, functions that are free of side effects (allowing immediate parallelization), a dataflow interpretation of the program, no pointers or direct memory manipulation, and region-based array management.

The Armada language is designed to describe algorithms, and expose all the possible parallelism in an application. A set of back-end compilation tools are then used to determine which available parallelism can actually be exploited on the target platform. Thus, the same source code can be compiled for a conventional multiprocessor and be implemented in a SPMD fashion, or it can target a heterogeneous platform and have parts of the computation spread over hard processors, specialized accelerators, soft processors, and hardware kernels as appropriate. (Note that Armada was designed to facilitate high-level synthesis, and thus the back-end tools can determine which parts of the application to synthesize, and which to run as software.)

Therefore Armada is a single, unified description of the computation, and depends heavily on the power and capability of the back-end infrastructure to compile to different target platforms. Careful consideration of this back-end framework is therefore crucial.

4.2 Modeling HPRC Systems

To map a given algorithm to a target system, the Armada back-end naturally needs information about that system. In a conventional system, this might simply be the number of CPUs, how much memory each has, and how they are interconnected. In a modern heterogeneous system, it must include not only CPUs, but also reconfigurable elements, GPUs, Cell broadband engines (BEs), and other accelerators, while still describing the capacity of each and the interconnection between them.

To capture this complexity, approximated system models are represented as a weighted directed graph. The graph edges represent communication links between processing elements (PEs), and are weighted with their bandwidth. Bidirectional edges are used to represent half-duplex links, with the implication that bandwidth used to communicate in one direction is unavailable to the other. A simplified example of this can be seen in Figure 4, which models a system similar to the BEE2 [2].

The PEs can be of several types: CPUs, GPUs, FPGAs, Cell BEs, and the like. They are annotated with their available resources, (which in different cases can be the number of processor cores, rendering pipelines, hard processors, hard multipliers, and so on) as well as their “computational capacity”.

Computational capacity is a heuristic approximation of how much computation a PE can provide. For a processor, this might be provided as a MIPS or MFLOPS number; for an FPGA, it might be based on how much datapath logic it

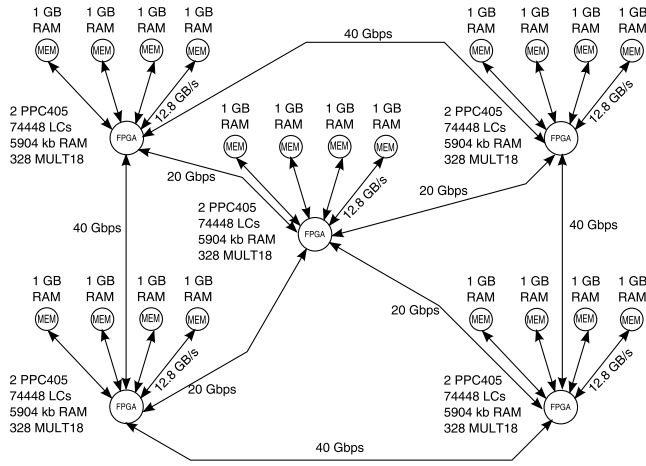


Figure 4: Example model of a BEE2-like system.

can implement. It is basically intended as a guide for the mapping of parts of an algorithm to parts of the system architecture.

There are also non-computational PEs that are used to simplify the relations between elements in a parallel system. These include switch nodes, which allow routing of data but have a finite switching capacity, as well as memory nodes, which have simply a fixed amount of memory, and a number of read and write ports. Thus, a distributed memory system would be modeled with each PE connected to a memory node. A shared memory system, on the other hand, would be represented as several computational PEs connected to a switch node, which is then connected to a memory node. This is done to model contention over memory access in a shared system.

We believe this simple graph model can be used to broadly describe different target platforms for use by a high level algorithmic language. The intent is to provide enough architectural information to guide general partitioning of the program, and then use more specialized mapping techniques to optimize for each PE.

4.3 The Armada Intermediate Representation

The irregular nature of a heterogeneous computing platform means that load balancing is most easily implemented at compile time, since that is when reconfigurable components can be configured and the computation can be partitioned. Given a description of the target platform, it is still necessary to map the algorithm onto the platform.

Most compilers map the program provided by the user to an intermediate representation (IR) to simplify analysis and optimization, and Armada follows this pattern. A common approach is to use a three-address code as an IR, in which the program is encoded as a sequence of instructions for an imaginary processor [1]. This does not map so well to a heterogeneous environment, however, since it already makes assumptions about the architecture it will run on, and binds the algorithm to specific memory transactions.

To mitigate this problem, the Armada Intermediate Repre-

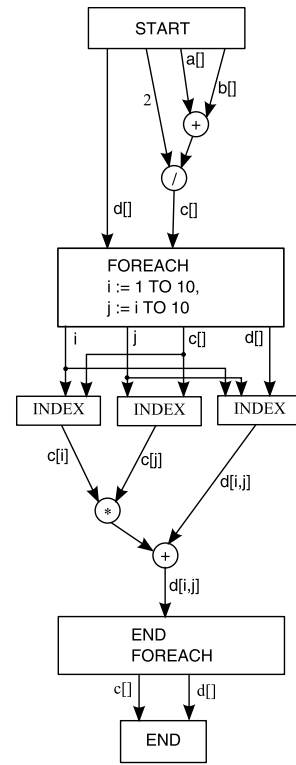


Figure 5: Example AIR dataflow graph.

sentation (AIR) takes another common approach – using a tree-like dataflow graph as the IR – and adapts it by removing memory load and store operations. Instead, the edges of the graph are annotated to indicate which variables the data belongs to, and so the edges capture not only the data dependences, but also all movement of data between operations. A small example of an AIR dataflow graph is shown in Figure 5, representing the following code snippet:

```

c[] := a[] + b[] / 2;
foreach(i := 1 TO 10 : j := i TO 10) {
  d[i,j] := d[i,j] + c[i] * c[j];
}

```

Thus, the AIR dataflow graph consists of a set of nodes – which are purely computational elements, and include high level operations such as matrix multiplication – connected by a set of edges, which represent data flowing between operations. This allows the back-end tools a greater degree of freedom in mapping computation to different PEs. For example, the first time an array is read, it might have to be brought in from main memory. But, based on the graph, the back-end tools might use the local memories in an FPGA to pass data to the next node, until eventually it is written back to main memory. This degree of ambiguity in the IR should allow easier mapping of algorithms to different target architectures.

Naturally, the nodes and edges of the AIR dataflow graph are also heavily annotated, describing the size and types of operations and the data flowing between them. This infor-

mation is used to generate a metric known as “computational density”. It is determined for each node, and attempts to approximate the amount of computation that the node entails. The value is normalized against a simple operation (say, 16-bit integer addition), and each node’s density is calculated based on the word size, array size, and data type of the operation.

The notion is that computational density and data edges can be used as guides to map portions of the AIR dataflow graph onto portions of the system model that have both the computational capacity and communication bandwidth to effectively implement that portion of the program. Such a partitioning will be very conservative, since it is only an approximation. Once partitioned, each of the subgraphs is compiled and optimized for its target PE, and any necessary top-level control code and hardware is generated.

4.4 Application to Soft Processors

The Armada programming model is designed to facilitate the programming of heterogeneous HPC systems, including those with reconfigurable elements. It does this by providing a high level, abstract way to describe algorithms, and represents them with a platform-agnostic IR. The back-end decision-making algorithm is where soft processors need to be taken into consideration.

The intent of the Armada back-end is to first do a high level partitioning of the application, clustering operations together to map roughly onto the PEs in such a way that data movement between clusters matches the capacity of communication links between the PEs. Thus, once the AIR dataflow graph is partitioned, each subgraph will have code generated for its target platform, including any necessary invocation of communication between PEs.

Each type of PE will have its own optimal coding style. While the code generated for CPUs, GPUs, and Cell BEs will all be different, all are relatively straightforward. When considering reconfigurable PEs, however, the notion of code generation becomes much more complicated, as the “code” has to be a configuration file.

From the point of view of the back-end system, there are several options for using the reconfigurable fabric: just hardware kernels, just soft processors, or a mixture of hardware kernels and soft processors. It is therefore necessary to automatically determine the best solution – or, at the very least, a solution that is good enough.

The easiest approach is to use only soft processors, but this is unlikely to provide worthwhile performance across a large enough range of applications to be an adequate general solution. The generation of optimized, application-specific soft processors may be a better general solution, if the automatic tools can identify significant chunks of the application to accelerate with custom instructions.

Generating only hardware kernels for the reconfigurable fabric would likely offer the best performance, but is also the most difficult option. Synthesizing a large amount of hardware from a high level description is still a challenging problem, and while HPRC systems have a lot of reconfigurable

logic available, we are still limited by trying to map a portion of the application to a specific PE.

Thus, a hybrid solution might offer the best mixture of performance and ease, but this introduces a new partitioning problem in deciding which parts of the program are implemented in a soft processor, and which parts as hardware kernels. There are a few obvious routes to explore to solve this problem: adapting techniques from hardware-software codesign; adapting the partitioning used at the top level to partition the application within the reconfigurable PE; or using the dataflow graph to determine the critical path of the application so that it can be implemented in hardware, while non-critical elements are relegated to one or more supporting soft processors. Which – if any – of these approaches will work best is still an open topic.

5. CONCLUSIONS

Even though general-purpose soft processors may be limited in their performance, they can still have a vital role to play in the HPC landscape, providing supporting control and interfacing on reconfigurable platforms, or by serving as application accelerators in some systems. Supporting such uses will be difficult, however, unless high level languages and sophisticated back-end technologies are built to automate their inclusion in HPRC systems. The key to leveraging soft processors and reconfigurable logic – and, indeed, any other kind of accelerator – is to provide a high enough level of abstraction to allow application experts to describe their algorithm, and then relying on new back-end tools to provide an overall performance improvement via these new systems. The Armada platform – currently under development – is being designed to demonstrate this.

6. ACKNOWLEDGMENTS

This work was supported by grants from NSERC, Xilinx, and the Walter C. Sumner Memorial Fellowship.

7. REFERENCES

- [1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson, second edition, 2007.
- [2] C. Chang, J. Wawrzynek, and R. W. Brodersen. BEE2: A high-end reconfigurable computing system. *IEEE Design and Test of Computers*, 22(2):114–125, March–April 2005.
- [3] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, F. Cantonnet, T. El-Ghazawi, A. Mohanti, Y. Yao, and D. Chavarría-Miranda. An evaluation of global address space languages: Co-array fortran and unified parallel c. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 36–47, 2005.
- [4] DRC Computer Corporation. RPU110 DRC reconfigurable processor unit, 2007.
- [5] A. Ghuloum, T. Smith, G. Wu, X. Zhou, J. Fang, P. Guo, B. So, M. Rajagopalan, Y. Chen, and B. Chen. Future-proof data parallel algorithms and software on intel multi-core architecture. *Intel Technology Journal*, 11(4):333–347, 15 November 2007.
- [6] A. W. H. House and P. Chow. Investigation of programming models for emerging FPGA-based high

- performance computing systems. In *Proceedings of FCCM 2008, the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2008.
- [7] V. V. Kindratenko, R. J. Brunner, and A. D. Myers. Mitrion-c application development on SGI altix 350/rc100. In *Proceedings of the 2007 International Symposium on Field-Programmable Custom Computing Machines (FCCM'07)*, pages 239–250, 23–25 April 2007.
- [8] M. D. McCool. Data-parallel programming on the Cell BE and GPU using the RapidMind development platform. In *GSPx Multicore Applications Conference*, November 2006.
- [9] Message Passing Interface Forum. MPI-2: Extensions to the message-passing interface. Online at <http://www.mpi-forum.org/docs/mpi-20.ps.Z>, 15 November 2003.
- [10] Mitronics AB. Mitrion users' guide. Technical report, Mitronics, 2008.
- [11] J. Osburn, W. Anderson, R. Rosenberg, and M. Lanzagorta. Early experiences on the NRL Cray XD1. In *Proceedings of the HPCMP Users Group Conference*, pages 347–353, June 2006.
- [12] M. Saldaña and P. Chow. TMD-MPI: An MPI implementation for multiple processors across multiple FPGAs. In *IEEE International Conference on Field-Programmable Logic and Applications (FPL 2006)*, pages 329–334, August 2006.
- [13] P. Yiannacouras, J. G. Steffan, and J. Rose. Application-specific customization of soft processor microarchitecture. In *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, 2006.