Classification and Performance of Reconfigurable Architectures

Steven A. Guccione and Mario J. Gonzalez

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Abstract. Recently, several systems have been designed which use reconfigurable logic to perform general purpose computation. While the number of these systems being constructed continues to increase, their relationship to conventional architectures is not clear. This paper proposes a model which unifies traditional instruction set architectures with reconfigurable architectures. From this model, four major architectural categories of reconfigurable machines are given. From this classification, issues of performance, programmability and scalability are addressed.

1 Introduction

A promising new approach to computing is currently being explored by researchers. This approach uses reconfigurable logic devices to perform computations previously reserved for either traditional instruction set computers or custom hardware.

Starting from a small handful of research projects in the late 1980s, over 40 systems based on reconfigurable logic have been constructed to date [11]. This rate of growth appears to be increasing rapidly.

Machines based on this technology have taken several diverse architectural approaches. It is the goal of this paper to first define the general features of these machines which make them unique and then to place these machines in a framework which permits comparison to other architectures.

From this general framework, four architectural categories of reconfigurable machines are defined and examined in closer detail. Finally, performance issues concerning these machines are examined. Particular attention is paid to performance limitations, rather than peak performance potential of these systems.

2 A Reconfigurable Model of Computing

With the commercial availability of relatively large reconfigurable logic devices and the evolution of computer aided design tools, it has become feasible to build fairly large and powerful systems based on reconfigurable logic. While it is clear that large gains in performance can be achieved with this approach, little has been reported on architectural issues concerning these systems. Unfortunately,

these machines appear on the surface to be sufficiently different from existing approaches to computation that direct comparison to traditional architectures is difficult and often confusing.

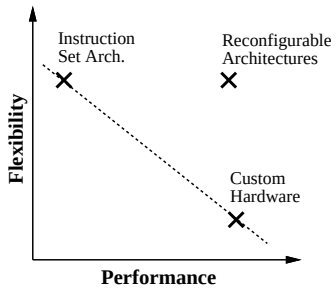

Fig. 1. The traditional tradeoff of flexibility and performance, and the potential of reconfigurable architectures.

Figure 1 gives a general diagram of traditional approaches to computation and their relation to reconfigurable architectures. Until the recent large scale use of reconfigurable logic to perform calculation, there was a generally accepted tradeoff between flexibility and performance in computing systems. In general, the more flexible a machine was, the simpler the programming, but the lower the performance. At one extreme, instruction set architectures can easily implement a wide variety of algorithms, but at only moderate levels of performance. On the other extreme is custom logic, which typically performs a single task very efficiently, but other tasks poorly or not at all. In between are various domain specific architectures which trade performance for flexibility.

The use of reconfigurable logic appears to offer the flexibility of instruction set architectures with the potentially high performance of fully custom hardware. This unique combination has led to difficulties in analyzing reconfigurable machines. Performance comparisons to both custom hardware and to instruction set machines can be found in the literature. While these comparisons are useful for benchmarking, they provide little insight into how performance gains are achieved and what levels of performance can be expected for other algorithms.

2.1 A Hardware Model

On selected algorithms, the performance of reconfigurable machines approaches that of custom logic. This level of performance is typically two to three orders of magnitude greater than that of implementations on instruction set architectures. For this reason, it is tempting to make performance comparisons to a custom hardware reference.

It should, however, be a foregone conclusion that any custom hardware implementation of an algorithm can also be similarly implemented on a suitably large

reconfigurable machine. The custom hardware implementation may be used to determine the maximum achievable performance for a given algorithm, but this is only useful in the cases where a comparable custom hardware solution exists.

Viewing reconfigurable systems as a form of custom logic does nothing to aid in predicting performance for algorithms for which no custom hardware reference platform exists. Neither is it clear that comparing a highly programmable machine to a fixed one is appropriate. Despite the similarities in performance, it appears that comparing reconfigurable machines to fixed custom logic implementations of algorithms can only be useful in providing a rough expectation of performance levels.

2.2 A Software Model

The ability to dynamically reconfigure hardware is often seen as the unique feature of machines based on reconfigurable logic. However, for any hardware to be used for more than a single purpose, some level of reconfigurability is necessary.

A traditional instruction set processor may be viewed as a reconfigurable processor. At the heart of the system, the arithmetic and logic unit, or *ALU* can be viewed as a *reconfigurable processing unit*, or *RPU*. A dedicated path is provided to the ALU for rapid reconfiguration. Depending on the data sent to this port, the ALU performs various different logical operations on the inputs. At a higher level, this reconfiguration data is viewed as the operation codes which partially define the behavior of the system. Figure 2 shows and ALU with instruction operation codes being used to reconfigure the ALU.

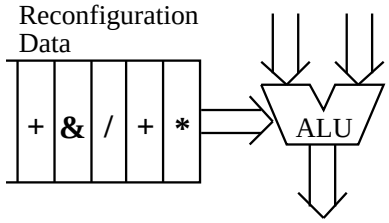

Fig. 2. A reconfigurable model of computing.

From this perspective, the traditional ALU is actually a specific class of RPU. The ALU is characterized by:

- A dedicated port for reconfiguration data
- Few possible configurations
- Rapid but frequent reconfiguration

The number of bits used to configure a typical ALU is less than 10, with 8 being a representative number. This permits at most 2^8 unique configurations. These configurations define the operations available to the machine.

ALU reconfiguration typically takes place on the order of once per clock cycle. While the number of possible functions is limited, sequential combinations of these operations permits a large number of useful functions to be performed. While not particularly efficient for any single task, this approach provides a fairly constant level of performance for a wide variety of algorithms. The primary drawback to this scheme is the bandwidth consumed by constant reconfiguration. Since this bandwidth is limited, operations must be performed in a more or less serial manner. In spite of these limitations, the inherent flexibility of this approach has been extremely successful.

By contrast, RPUs based on reconfigurable logic devices make large scale use of reconfiguration. Instead of the roughly 8 bits used to reconfigure a traditional ALU, thousands of bits are used to reconfigure an RPU. This very large number of bits permits a very large number of possible functions. It is the writing of these bits to the control port of the RPU which is is one of the major limiting factors in the use of larger RPUs.

3 An Architectural Classification

A model has been proposed that considers all general purpose machines to be reconfigurable. What differs is the way in which reconfiguration is managed. As noted, the traditional instruction set architecture opts for frequent reconfiguration with a small number of possible operations. Machines based on reconfigurable logic use a much more flexible RPU, but at the cost of requiring a high reconfiguration overhead. Based on the way in which reconfiguration is managed and utilized to perform computation, these machine can be further classified.

Two relatively independent architectural parameters can be used to subdivide reconfigurable machines into four general categories. These parameters are:

- RPU size
- Dedicated local memory

The first parameter, the RPU size, is the amount of reconfigurable logic used to implement the RPU. This value can be measured more or less by the number of equivalent logic gates in the RPU. This will determine the complexity of the functions which can be implemented by the RPU.

The second parameter is dedicated local memory. This is the memory directly accessible to the RPU. The absence or presence of dedicated memory will effect the system at several levels. Architecturally, dedicated memory implies that the reconfigurable portion of the system may operate independently from the host. From a software perspective, a programming model which supports an independent processor and memory space is indicated. Finally, at the application level, dedicated memory will effect the types of algorithms that can benefit effectively from the use of reconfigurable processing.

Based on these two parameters, reconfigurable machines can be divided into four major categories. These are *Application Specific Architectures (ASA)*, *Reconfigurable Logic Coprocessors (RLC)*, *Custom Instruction Set Architectures (CISA)* and *Reconfigurable Supercomputers (RS)*. Figure 3 shows the four types of reconfigurable architectures and their RPU sizes and presence or absence of dedicated local memory.

	No Local Memory	Local Memory
Small RPU	CISA	RLC
Large RPU	ASA	RS

Fig. 3. An architectural classification of reconfigurable machines.

In this table, a small RPU is defined to be less than 10^5 equivalent gates and a large RPU is assumed to be greater 10^6 equivalent gates. This boundary is somewhat arbitrary and leaves a "grey area" for machines between 10^5–10^6 equivalent gates. Machines which have RPUs whose gate count is somewhere in this region may have features of two classes of machines.

3.1 Application Specific Architectures

The first class of reconfigurable systems are *Application Specific Architectures (ASA)*. These machines were some of the earliest to exploit the advantages of reconfigurable logic. They have no dedicated memory and have relatively large RPUs. These machines are primarily characterized by a very narrow area of application.

One popular use of such application specific architectures is in the acceleration of logic simulation. Here, reconfigurable logic is used to prototype custom hardware. This approach has resulted in dramatic speedups over more traditional software simulations. A good overview of this area can be found in [15].

Another example of an application specific machine is *GANGLION* [6]. This machine was used to implement a fixed size three-layer neural network. This system made use of reconfiguration to provide a dramatic speedup over established software techniques. This hardware was, however, only useful to simulate a single neural network configuration. Modifying the number of neurons in the system was not possible.

While perhaps the earliest large-scale use of reconfigurable logic, these machines function much like custom hardware. While they may take advantage of reconfiguration to accomplish their tasks, they are typically used for a single application. In this sense these machines are more closely related to traditional fixed custom hardware than more general purpose reconfigurable machines.

3.2 Reconfigurable Logic Coprocessors

The second class of machines based on reconfigurable logic are called *Reconfigurable Logic Coprocessors (RLC)*. These machines are relatively small, with only a few thousand equivalent gates in the RPU. They contain dedicated memory directly coupled to the RPU. Since the RPU is relatively small, the memory on these systems is similarly limited. Typically on the order of 1 megabyte or less is provided.

Figure 4 gives a high-level diagram of the RLC approach. Some examples of this approach to reconfigurable computing are the *Algotronix 2x4* [1] [14], the *AnyBoard* system [8], the Xputer [12] and the *BORG* system [5].

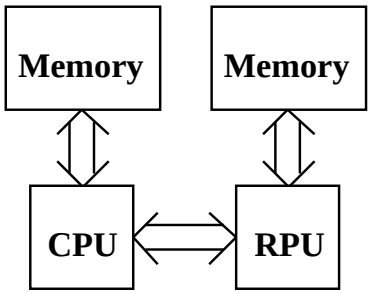

Fig. 4. The coprocessor approach.

Because of the relatively small RPU, these systems are used primarily as small custom logic prototyping systems and are programmed using circuit design tools and methodologies. They may be used effectively to perform tasks of low computational complexity requiring high throughput. Digital signal processing is one fertile area of application for this class of machine.

3.3 Custom Instruction Set Architectures

The third type of reconfigurable system is the *Custom Instruction Set Architectures*, or *CISA*. These machines trace their roots to earlier custom microcode machines. They attempt to increase performance by providing customized instructions typically unavailable in traditional instruction set architectures.

Figure 5 gives a diagram of this approach to reconfigurable computing. These machines differ from reconfigurable logic coprocessors in that they are typically more tightly coupled to the host CPU and have no dedicated memory. Some examples of CISA machines are the *PRISM* systems [2] [17]. the *flexible processor* [18], *Spyder* [13], the *ArMen* machine [16] and the *CM-2X* [7].

CISA machines typically offer a more traditional programming environment than other types of systems. This is primarily because the architecture is based on the instruction set model of computation. This shared programming model

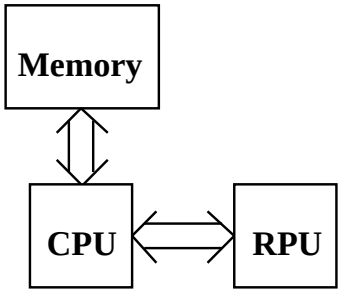

Fig. 5. The CISA approach.

permits the host and RPU to cooperate closely. The function configured into the RPU is viewed by the host as another instruction available to the processor. This permits a simple interface for existing tools and languages. It is likely that these systems will continue to be used for research in high level language programming of reconfigurable machines.

While these machines tend to be easier to program, the use of the instruction set model of computing makes this approach more or less serial. While the RPU can effectively implement complex bit operations not found in traditional architectures, it is not possible to further exploit parallelism within the RPU via pipelining.

Additionally, scaling to larger RPUs permit more complex functions, but the increase in the amount of logic will tend to slow the speed of the RPU. Depending on the coupling to the host processor, this may require a decrease in the system clock speed.

Except for special cases, this approach to reconfigurable computing offers relatively modest gains in performance. It is interesting to note that of the five machines cited above, three are multiprocessor systems. This multiprocessor approach should further boost performance by exploiting data parallelism, but at the cost of replicated hardware.

3.4 Reconfigurable Supercomputers

The final class of reconfigurable machines are *Reconfigurable Supercomputers (RS)*. These machines have large RPUs, on the order of one million equivalent gates. They typically have large amounts of dedicated memory and a high bandwidth link to a powerful host processor.

Architecturally, reconfigurable supercomputers resemble reconfigurable logic coprocessors. The difference is primarily one of scale. Reconfigurable supercomputers are several times larger, both in RPU and memory size, than reconfigurable logic computers. This permits these machines to perform larger and more complex algorithms.

Some examples of reconfigurable supercomputers are the *PAM* systems [3], the *Splash* systems [9] [10] and the *Virtual Computer* [4]. All of these systems

tend to be on the low end of the scale, with none having an RPU with one million equivalent logic gates. These systems are perhaps better referred to as reconfigurable mini-supercomputers. They are, however, distinguished by their large memory and I/O bandwidth, as well as their fairly powerful host machines.

Like the smaller reconfigurable logic coprocessors, these systems currently tend to be programmed using hardware design tools and methodologies. Because of their larger size, however, they can be used to implement larger and more complex algorithms, often involving more general arithmetic operations. Several applications have been implemented on these machines achieving speeds surpassing that of large vector supercomputers.

4 Performance Issues

Based on the reconfigurable model of computation, all reconfigurable machines can be viewed as coprocessors which implement custom instructions. A portion of the increase in performance comes from replacing a sequence of instructions normally executed on the host with a single complex instruction implemented in the RPU.

If N cycles of processing on a traditional machine are replaced by reconfigurable logic and R cycles are used to configure the hardware, the reconfigurable logic must be used at least M times to amortize the overhead of reconfiguration. Assuming a single cycle operation for the reconfigurable logic processor, M is given by Equation 1. This represents the *break even* point where reconfigurable logic may be profitably used.

$$M = \frac{R}{N-1} \tag{1}$$

In order to make effective use of the reconfigurable logic, M operations must be performed to cover the cost of reconfiguration. More operation using the reconfigurable logic will server to save $(N-1)$ cycles in the overall execution of the algorithm.

This simple analysis indicates that in order to minimize the break even point, reconfiguration should be rapid and the number of instructions replaced large. Unfortunately, these parameters are not independent. Typically, large numbers of instructions will require large amounts of logic to implement, this will, in turn, require longer reconfiguration. While the exact value of M will vary, it should be closely tied to the type of reconfigurable device used to implement the hardware.

This implies that more complex functions will require more repetitive operations to be competitive.

5 Conclusions

A model of computing which bridges the existing gap between traditional instruction set architectures and evolving reconfigurable architectures has been

described. Rather than attempting to define reconfigurable computers in terms of instruction set machines, instruction set machines have been demonstrated to be a special type of reconfigurable machine.

Reconfigurable machines can be grouped into four categories based on the size of their reconfigurable logic units and the presence or absence of dedicated memory. Ignoring application specific architectures, machines with no dedicated memory may be more easily programmed using the traditional instruction set model of computation. This model, while simplifying programming, is inherently serial and will limit performance.

Systems with dedicated memory promise higher performance, but require a more complex programming model. This model must manage control and communication between the host and the RPU as well as recognize and make effective use of the dedicated memory.

Finally, reconfigurable supercomputers with over one million equivalent logic gates are on the horizon. Indications are that these machines will be competitive with existing parallel and vector supercomputers, with orders of magnitude less hardware. The ability to configure and interconnect several complex arithmetic and logic blocks is likely to make the use of high level languages a practicality, if not a necessity.

References

1. Algotronix, Ltd. *CAL1024 Datasheet*, 1990.
2. Peter M. Athanas and Harvey F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *IEEE Computer*, 26(3):11–18, March 1993.
3. Patrice Bertin, Didier Roncin, and Jean Vuillemin. Introduction to programmable active memories. Technical Report 3, DEC Paris Research Laboratory, 1989.
4. Steven Casselman. Virtual computing and the virtual computer. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 43–48, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
5. Pak K. Chan, Martine D. F. Schlag, and Marcelo Martin. BORG: A reconfigurable prototyping board using field-programmable gate arrays. In *First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pages 47–51, 1992.
6. Charles E. Cox and W. Ekkehard Blanz. GANGLION – a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27(3):288–299, March 1992.
7. Steven A. Cuccaro and Craig F. Reese. The CM-2X: A hybrid CM-2 / xilinx prototype. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 121–130, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
8. David E. Van den Bout. The anyboard: Programming and enhancements. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 68–77, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
9. Maya Gokhale, William Holmes, Andrew Kosper, Dick Kunze, Dan Lopresti, Sara Lucas, Ronald Minnich, and Peter Olsen. SPLASH: A reconfigurable linear logic array. In *International Conference on Parallel Processing*, pages I–526–I–532, 1990.

10. Maya Gokhale, William Holmes, Andrew Kosper, Sara Lucas, Ronald Minnich, and Douglas Sweely. Building and using a highly parallel programmable logic array. *IEEE Computer*, pages 81–89, January 1991.
11. Steven A. Guccione. List of FPGA-based computing machines. World Wide Web page http://www.utexas.edu/~ guccione/HW_list.html, 1994.
12. Reiner W. Hartenstein, Alexander G. Hirschbiel, Michael Reidmüller, Karin Schmidt, and Michael Weber. A novel ASIC design approach based on a new machine paradigm. *IEEE Journal of Solid-State Circuits*, 26(7):975–989, July 1991.
13. Christian Iseli and Edwardo Sanchez. Spyder: A reconfigurable VLIW processor using FPGAs. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 17–24, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
14. Thomas Andrew Kean. *Configurable Logic: A Dynamically Programmable Cellular Architecture and its VLSI Implementation*. PhD thesis, University of Edinburgh, Department of Computer Science, January 1989.
15. Henry L. Owen, Ubaid R. Khan, and Joseph L. A. Hughes. FPGA-based emulator architectures. In Will Moore and Wayne Luk, editors, *More FPGAs*, pages 398–409. Abingdon EE&CS Books, Abingdon, England, 1993.
16. F. Raimbault, D. Lavenier, S. Rubini, and B. Pottier. Fine grain parallelism on a MIMD machine using FPGAs. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 2–8, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
17. M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh. PRISM-II compiler and architecture. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 9–16, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
18. Andrew Wolfe and John P. Shen. Flexible processors: A promising application-specific processor design approach. In *Proceedings of the 21st Annual Workshop on Microprogramming and Microarchitecture*, pages 30–39. IEEE Press, 1988.

This article was processed using the LaTeX macro package with LLNCS style