Configurable Computing: The Road Ahead

William H. Mangione-Smith
Electrical Engineering
University of California at Los Angeles
billms@ucla.edu

Brad L. Hutchings
Electrical & Computer Engineering
Brigham Young University
hutch@ee.byu.edu

Abstract

Since at least 1989, FPGA-based computing platforms have demonstrated the potential for achieving extremely high performance for a range of applications: image filtering, convolution, morphology, feature extraction, and object tracking. The potential for high performance is real, and for a number of applications researchers have developed prototype systems that achieve performance that is an order of magnitude higher than conventional approaches [A93]. However, realizing this potential outside of the laboratory has proven extremely difficult because these systems rely on manipulating low level abstractions (e.g. digital circuits) and thus require highly skilled developers. The goal of this paper is to discuss the past, present and future of this important technology and to determine which approaches have worked, which are currently showing promise, and which will likely prove successful in the future.

1 Introduction

Since at least 1989 [BRV89], FPGA-based computing platforms have demonstrated the potential for achieving extremely high performance for a range of applications, including: image filtering [RV93], convolution [AA95,BRV89], morphology [DK95], feature extraction [AA94], and object tracking [QK94,S95]. The potential for high performance is real, and for a number of applications researchers have developed prototype systems that achieve performance that is an order of magnitude higher than conventional approaches [A93]. However, realizing this potential outside of the laboratory has proven extremely difficult because these systems rely on manipulating low level abstractions (e.g. digital circuits) and thus require highly skilled developers. The goal of this paper is to discuss the past, present and future of this important technology and to determine which approaches have worked, which are currently showing promise, and which will likely prove successful in the future.

The term *configurable computing* [VM97] is undergoing redefinition, as often happens with an emerging research field. From a device perspective, there appears to be three major approaches: reconfigurable gates (RG), reconfigurable interconnect (RI), and hybrids. An RG device typically contains thousands of small (i.e. low powered) processing elements. For example, a typical RG processing element may consist of a single flip-flop and a function generator that implements a Boolean function of four variables. The RG devices have programmable interconnect which is manipulated as individual wires. Because of their fine granularity, RG devices are the most flexible configurable component; their elements can be used to implement state machines, datapaths and nearly any digital circuit. This flexibility is purchased with

additional silicon and lowered performance on certain classes of problems, as compared with RI devices. In contrast, an RI device consists of perhaps 100 or less processing elements; each element typically resembles a conventional 8- or 16-bit ALU which implements a rudimentary set of functions. Each ALU typically has a set of registers that have the same width; together they form a fixed width datapath. The routing network exhibits a similar level of granularity, with individual wires grouped into buses. RI devices can achieve high performance on applications that map well to the restricted ALU and interconnect structures. However, RI devices are less flexible in general and provide lower performance for applications that are not well matched to both the ALU and interconnect structures. RI devices also are not capable of implementing high-performance state machines or datapaths of arbitrary width. Finally, hybrid systems couple an RG device with a conventional programmable processor. These systems attempt to combine the best of both worlds: the efficiency and programmability of processors and the high performance of customized digital circuits. Commercial FPGAs are good examples of RG devices. Examples of IC devices include academic projects like Colt [BA97], Matrix [MD96] and Rapid [EC96]. Examples of hybrids include GARP and RSP.

This paper will focus on RG devices, i.e. FPGAs, as configurable-computing devices. We believe that research on RI devices and hybrids is still in the very early stages with no significant reports of actual application or system implementations.

2 Estrin's Early Machine

It now appears likely that the earliest configurable computing machine was proposed, designed, and implemented by Professor Gerald Estrin at UCLA in the early 1960s [E60]. This system is roughly similar to the hybrid devices described above. Estrin proposed the "Fixed plus variable structure computer", where some fixed hardware was dedicated to an inflexible abstraction of a programmable processor and a flexible component which implemented digital logic. The processor would be programmed using conventional compilers and languages, and the flexible hardware component was programmed in a method similar to current FPGAs. Unfortunately, Estrin's architectural concepts were well ahead of the enabling technology, and he was only able to prototype a very crude approximation of his vision. None the less, many of the concepts that are now being "discovered" by the configurable computing community lie quietly unheeded in Estrin's early publications.

It is somewhat ironic that Estrin concluded that program and project management tools were necessary for his architecture to succeed, and the focus of his research shifted to CAD. Many researchers in the configurable computing community are now turning to hardware/software co-design as the silver bullet to help structure, analyze and ultimately synthesize hybrid configurable computing systems.

3 Motivation

We believe that configurable computing generally presents four main benefits:

1. Performance. FPGA enable custom computing systems to be highly specialized to specific data, as well as specific applications. One optimization technique uses a form of partial evaluation, where some of the data are assumed static; the FPGA circuit is optimized to take advantage of this static data. Another typical optimization is to implement highly parallel

architectures that can exploit significant data-level parallelism. By capitalizing on these optimization opportunities, a highly-tuned --yet programmable-- system can be constructed that attains near-ASIC performance.

2. Cost Effectiveness. Configurable computing can be used to reduce system costs through two approaches: hardware reuse and low NRE. A number of research efforts have demonstrated time-shared methods for simulating a large circuit on a smaller FPGA [SJ95,EH94,EH96]. For example, the UCLA image compression work swapped four different circuit configurations onto a National Clay FPGA. Since the clock speed of the FPGA was more than 4X greater than the required circuit speed, the system achieved comparable performance at one quarter of the hardware costs. The DISC project at BYU showed a similar result; runtime reconfiguration was used to implement a set of image processing applications. Furthermore, as the feature size of semiconductor processes shrink, silicon foundries are raising mask charges and minimum fabrication runs. These trends are driving low volume digital designs away from state-of-the-art process technology, and making FPGAs much more cost effective.
3. Custom I/O. FPGAs provide an extremely rich and flexible set of programmable I/O signals. These components give the system designer an opportunity to reuse existing hardware, or commit earlier to a new hardware design. The benefits of custom I/O extend beyond the prototype stage, for example system functionality may change after deployment, and the ability to rewire system I/O through reprogramming the FPGA can be an invaluable advantage.
4. Fault-Tolerance. Some recent efforts have investigated the benefits of using FPGAs to provide fine-grained fault recovery. This approach has the benefit of increasing system reliability for much lower cost than the traditional approach of circuit and subsystem redundancy.

While the benefits of configurable computing may appear obvious to those immersed in the research community, we believe that it is important to always keep the specific goals in mind when beginning a line of investigation. It appears that some of the active research is driven by a desire to simply experiment with the technology, rather than improve performance or reduce the cost of a complete system.

4 Performance = Applications + Hardware + Development Model

The ultimate performance achieved by an application running on a configurable-computing platform is affected by three factors: the characteristics of the application, the hardware the application runs on, and application development model (CAD tool, compiler, etc.). This section will give a brief overview of these three areas and suggest important issues for achieving high performance.

4.1 Applications

A review of successful applications shows that the amount of available parallelism in an algorithm is the primary factor limiting the performance of an FPGA-based implementation. FPGAs typically fare much worse than microprocessors on sequential algorithms. Consider that a state of the art microprocessor contains a few coarse (e.g. 32-bit) computational elements (e.g. floating-point multipliers) controlled by dedicated circuits. Thus, microprocessors excel at the execution of sequential programs with limited parallelism that operate on relatively wide data. In contrast, a state-of-the-art FPGA contains thousands of fine-grained (e.g. 1-bit) computational elements with no dedicated control structure. Because of these fine-grained components, FPGAs do well on applications that exhibit unlimited parallelism on relatively narrow data. The flexible control capabilities work well on simple execution patterns that can be hard-coded into circuits. This approach avoids the instruction-fetch bottleneck. In general, microprocessors outperform an FPGA when executing a single operation or when executing several operations that must be performed sequentially. However, if the algorithm allows hundreds or thousands of operations to occur simultaneously, then even a *single* FPGA can in many cases outperform a microprocessor [WH95,PH95].

4.1.1 Example Application Area: Image Processing

Image processing has proven to be an ideal application area for FPGAs for many of the reasons listed above. First, many image-processing applications are inherently parallel; the amount of exploitable parallelism is limited only by the size of the image and by the available hardware. Most image operations can be performed in parallel; each output pixel is only dependent upon a small number of input pixels and, in most cases, all output pixels can be computed simultaneously. Second, many image-processing applications consist of simple operations that are applied in a fixed sequence to a set of input images. This sequence of operations can be implemented as a deep pipeline where each stage of the pipeline implements some image-processing operation and all stages operate concurrently. Third, image data are low resolution; current sensor technology limits the width of each pixel to 8-12 bits with 8 bits being typical. Fourth, data sets are quite large. A single image can contain several megabytes of information. Often, the data is streaming in real-time from a sensor and so represents an infinitely long sequence. Finally, many image-processing applications can be represented as simple dataflow graphs with simple control and no loop-carried dependencies. The control algorithms usually can be hard-wired into the FPGA as a finite state machine.

While many image-processing applications have been shown to be appropriate for FPGA technology, e.g., convolution, histogram computation, thresholding, etc., there are some exceptions. For example, FFT applications often require floating-point arithmetic, which has not been efficiently implemented on FPGAs.

4.1.2 Problem Specific Integrated Circuits (PSICs)

We believe that at least one important class of applications awaits exploitation by configurable computing. Thus far, there has been a significant amount of effort to build application-specific circuits on programmable hardware. As we have discussed, this approach is effective for many applications. However, ASICs have also proven effective for these tasks. While there are a number of benefits that justify using

programmable parts, each system developer must be prepared to defend the choice (as opposed to ASICs) on some grounds.

Configurable computing provides the opportunity to apply partial circuit evaluation to hardware design and implement systems that could never be developed with ASICs. Some early steps along this path have been taken, usually involving constant propagation [SHM96,PH95]. Clearly, some of the benefits of this approach can be captured with ASIC devices that use programmable registers. However, when the partial evaluation results in a data-specific structural change in the circuit there is no opportunity for developing a reusable ASIC. We prefer to refer to these systems as *problem-specific integrated circuits (PSICs)*, because the hardware is structurally optimized to a specific instance of a problem. PSICs are generated once to solve a specific problem instance; after they are executed once they produce a result and are then thrown away. A number of researchers have discussed pursuing this approach to attack some classic optimization problems, though thus far no results have been reported.

4.2 Hardware

A variety of FPGA-based computing platforms have been constructed and reported in the literature. These platforms fall into two basic categories: FPGA emulator and custom computer.

The FPGA emulators use *no* a priori knowledge of the characteristics of the applications that will eventually be implemented on the platform once it is constructed. These systems try to create an extremely flexible platform with sufficient generality to implement almost any FPGA-based application. We view this approach as FPGA emulation because the goal is to create a configurable system that resembles an FPGA that is much larger than those commercially available. While some of these systems are augmented with embedded memories and hardware for accelerating specific tasks, their basic characteristic is of a large FPGA.

The custom computer model does assume *some* a priori knowledge of the applications that will be implemented on the platform. With this strategy, the goal is to create a platform that is flexible enough to implement an identified subset of FPGA applications. We refer to these machines as custom computers because the system architectures are customized/optimized for a given class of application architectures.

4.2.1 FPGA Emulators

The foremost consideration when building systems of this type is to ensure that there are sufficient interconnection resources in the platform to support a wide range of applications. These systems are primarily composed of commercial FPGAs and Field Programmable Interconnect Chips (FPICs). The FPICs are used for connections among the FPGAs and to augment the general interconnect resources of the system. FPGAs are distributed throughout the device, with FPICs providing an extremely rich board level network among FPGAs. In most systems of this type, there are far more FPICs than there are FPGAs [LG97].

Other approaches have taken commercial FPGAs and mounted them on an MCM substrate along with FPICs. For example, the FPMCM [DG95] consisted of 12 ATT 3064 FPGAs with an APTIX FPIC. Altera also fabricated an MCM that consisted of

four 8K Flex CPLDs with an Aptix FPIC. A full system would require multiple MCMs, probably interconnected with FPICs.

The main advantage of this approach is that it allows researchers to build larger systems with more configurable gates. However, a number of practical concerns need to be addressed:

1. Short Lifetime. The ultimate lifetime of such a machine is dictated by Moore's law, which says that circuit density (transistor count) doubles every 18 months or so. Unless designers are extremely fast, their FPGA emulators will have about the same capacity as the highest density FPGA device. At best, these systems appear to provide about a 2-3-year advance on present technology. Difficulties in acquiring known good die, scaling the FPGA with system interconnect, and creating the CAD tools to target the device/system makes it very difficult to keep up with technology advances.
2. High Cost. FPGA emulators are extremely expensive to construct. In a typical system, most of the resources must be devoted to the implementation of the general-purpose routing network, and thus most of the ICs are FPICs, which do not produce useful work. Only a small percentage of devices can be devoted to logic for computation.
3. Lowered Clock Rate. System clock rates are typically limited by the large delays encountered in these systems as signals pass through FPICs between FPGAs and system I/O.
4. CAD Tools. The software to partition, place and route multiple chip systems remains an open research area. In addition, device utilization is limited by the need to assign I/O pins (the utilization of many FPGAs is reduced when I/O pins are constrained). Finally, if the system consists of many chips, the time required for CAD place and route can be extremely long, perhaps measured in CPU days.

We believe that these four factors make it difficult to use FPGA emulators as configurable-computing research vehicles. A more prudent approach is to use the highest density component available and then focus on using it efficiently. Any of the techniques developed for efficient use of a small FPGA will likely apply to a larger one. When the research community gets off the main line of commercial development we are taking a large risk because we can no longer leverage commercial activities. This increased risk must be compensated by an increased payoff in case of success. It isn't clear what the payoff is for developing large FPGA emulation systems with conventional FPGAs, since the industry is aggressively moving down that path as well and may arrive there before the researcher.

It should be noted that one research effort, Teramac [AC96], actually achieved significant success in two of the above areas: lifetime and CAD tools. Teramac was a success primarily because the system --both the individual devices and system architecture--was crafted to accommodate the compiler and not vice versa. Constructed from custom FPGA devices designed specifically for this project, Teramac achieved a level of automation and mapping speed that, to date, has not been rivaled. On Teramac, a single netlist of 1 million gates can be automatically partitioned, placed, and routed in less than 2 hours on a modern workstation. Teramac still suffers from clock-speed limitations (1 Mhz or less), a problem mitigated for massively parallel problems because of the massive size of the platform and, because

of its high cost, will remain strictly a research vehicle. However, Because of its extremely high partition/place/route speeds, Teramac remains (and is likely to remain for some time), the only truly ``rapid'' prototyping system.

4.2.2 4.2.2 Custom Computers

The alternative to FPGA emulators is to design systems based on some prior knowledge of the applications that will be implemented. This section will overview two of the best known custom computers: Splash and DecPerle.

From a conceptual point of view, the Splash system consists of a linear array of processing elements. This makes Splash a good candidate for linear-systolic applications, which stress neighbor-to-neighbor communications. Because of limited routing resources Splash has not proven as effective at implementing multi-chip applications that are not linear systolic, though some progress has been made [GN95]. The actual Splash platform consists of a board with 16 Xilinx 4010 chips for computation arranged in a linear systolic array. One additional 4010 is used for system control. Each computation FPGA has a 36-bit connection to its two nearest neighbors and to a local 512 Kbyte memory (16 bit word size). There is also a crossbar connected to all of the chips that enables a limited amount of general communications. Up to 16 boards can be daisy-chained together to provide a linear-systolic array of 256 FPGAs.

The DecPerle-1 is organized as a 2D mesh and consists of a 4X4 array of FPGAs. Each FPGA has connections to its nearest neighbors as well as to a column bus and a row bus. One megabyte memories surround the FPGA array. Both Splash and the DecPerle machines were designed to be used as attached accelerators alongside workstations.

Unlike the FPGA emulators, Splash and DecPerle do not provide a general routing network. Because of this constraint, these platforms require the designer to manually partition the circuit during the design phase ensuring that the available interconnect is used as efficiently as possible. This approach typically implies that the best implementation styles for these types of systems rely primarily on local interconnect, e.g., systolic-like implementations.

The custom-computer approach has the following advantages.

1. Higher Performance. Because the routing network provides restricted connectivity and can be optimized for a certain class of problems, delays between chips can be reduced; this optimization allows the system to achieve high performance. Splash-II and DecPerle have reported the highest performance of all systems described in the literature.

2. Lower Cost. Again, because the routing network is limited, an equivalent amount of programmable hardware can be assembled for less cost than the FPGA emulator approach. The majority of devices can be used to implement computation.

3. Size. It can be feasible to construct custom computers that contain many FPGAs. This is possible primarily because the routing network can be greatly simplified in a custom computer. Splash-II, for example, can be expanded to a maximum of 16 boards, for 256 FPGAs, all of which can be used for computation.

4. Lifetime. Because larger systems become feasible under this approach, systems remain useful longer because commercial components not close the gap quite as quickly.

There are significant disadvantages as well. The designer must be concerned with the size and utilization of the FPGA devices, the size and usage of the memories and finally, the overall interconnection of all devices on the platform during all phases of the design process. This makes the design process more difficult and time consuming. In general, custom computers make extremely poor rapid-prototyping platforms. The design process is slow and modifications require too much design time to complete. However, if the goal is to construct a *computing platform*, this approach will achieve the best performance.

4.3 Development Model

There have been a number of research efforts that have studied compilation strategies and CAD tools for FPGA-based computing systems. Two basic strategies have arisen: automated compilation tools that attempt to automatically map a behavioral description to a netlist, and manual layout tools that describe the structure and physical layout of a circuit.

4.3.1 Automated Compilation

PRISM [AS93,WA93] was perhaps the earliest report of a compilation system that accepted generic 'C' code and generated FPGA configurations in semiautomatic fashion. PRISM analyzed C code to identify functions that could be implemented with combinational logic. The programmer was able to indicate which functions were speed critical and thus good candidates to synthesize to netlists. PRISM targeted a hybrid system composed of a microprocessor and FPGAs; the speed critical functions were mapped to the FPGAs while the remaining non-critical code executed on the microprocessor.

PRISC [RS94] was an attempt to quantify the benefit of adding a small amount of reconfigurable logic, which was called a programmable function unit (PFU), directly to a microprocessor datapath. The PFU could be used to implement a series of Boolean functions as well as optimized versions of if-then-elses and computed GOTOs. These instructions take multiple cycles on a traditional microprocessor but can be executed in a single cycle in the PFU. All of the results were simulated, and no hardware was ever built.

Handel [P93] uses a syntax-directed approach for translation. Specific language constructs (e.g. if-then-else and for-loops) and operators (e.g. "+" and "*") are mapped directly to a library of predefined hardware blocks. Handel is based on a version of OCCAM, and relies on the parallel constructs in the language to model concurrent hardware resources.

The speedups reported by these various systems are modest, typical averaging no more than 2x for a complete application. Modest speedups are obtained because the problem is inherently difficult: sequential descriptions are difficult to convert into high-performance parallel implementations, automated place and route systems are relatively inefficient, etc. Note that although this short survey is not all-inclusive, we believe that it is very representative of the complete body of results.

4.3.2 Structural and Physical Design Tools

To achieve better FPGA utilization and performance, a number of systems have incorporated structural and physical placement information. The most successful of these was the PAM project [BT94], which used a tool based on a C++ variant. Although most of the reported results emphasized physical representations, PAM was also able to input structural descriptions. The PAM project resulted in some of the highest performance FPGA systems ever reported. They reported 10x or better performance (over any known solution, including supercomputers) for a variety of problems in image processing, heat-transfer simulation, RSA decryption, etc. Others research efforts have used similar approaches to achieve equally promising results [GK90].

All previously discussed efforts were academically inclined studies that developed special-purpose languages and CAD tools. SPLASH [AB92,A93] took a different approach and used a commercially available synthesis tool (SYNOPSYS). The designer partitioned the application across the available FPGAs and used VHDL to capture the behavior of each chip individually. Synthesis, followed by standard place and route, was used to do the final mapping. The second generation SPLASH achieved a speedup of more than 40,000X over a SPARC-1 on the genome sequencing problem [H93]. These speeds are obtained because the overall organization is carefully constrained by the designer, similar to the manual design systems already discussed.

4.4 What Works Best

What "works best" would depend upon the user's requirements. However, it is important to remember that the primary reason for using FPGAs is often to *achieve significant performance improvements.* Systems that are manually designed, i.e. those defined structurally and incorporating information regarding physical placement, achieve the highest performance. The next highest performance is achieved with RTL-level VHDL and Verilog using commercial synthesis tools. Compilers based on higher level abstractions, such as C or behavioral HDL, achieve significantly lower performance than either of these two other approaches.

We expect that systems defined with very low level abstractions will continue to exhibit a performance advantage over those using higher level abstractions. Fully automated approaches will continue to lag significantly behind manual design approaches in performance for two reasons. First, automated place and route cannot achieve the same performance as carefully optimized custom layout. Second, the translation of a behavioral description into a structural circuit is an extremely difficult problem that is still an active research area. It is very difficult to compete with an experienced designer who carefully partitions, structures, and optimizes a system.

We are well of the fact that this argument parallels an earlier argument regarding compilers and assembly language coding for general purpose computing. We are not claiming that automatic compilation will never be important for application development on configurable computers; we hope that it will. However, it is important to remember that configurable computing systems are the realm of ultra-high performance today. The development effort required to get any application to produce even modest performance is high, and any seemingly small detail can easily result in a significant performance reduction. Thus, those engineers who work with this technology tend to continue to optimize their systems until they are convinced

that all of the performance potential has been captured. Automatic compilation tools have a very long road ahead of them before they achieve the performance demands of these engineers.

Note that what has been pursued thus far represents only two possible extremes of this field: manual layout and fully automatic compilation. For the future, a *hybrid* approach that combines libraries of custom layout with automated compilation has the potential to achieve higher performance at lower design cost. DISC [WH95] was an early example of such a system and it demonstrated that a compiler could be combined with a domain-specific library (e.g. image processing) to achieve high performance with relatively little FPGA hardware. This hybrid approach works best for applications that can be composed from a set of high-level operators that have been carefully optimized. The responsibility of the compiler is to select the correct module (often using a syntax-driven approach) and then to introduce any additional circuitry to ensure correct communication between and sequencing of the selected operators. Many important domains are amenable to this approach, for example image and signal processing as well as sonar data processing.

5 Cost Effectiveness

It is well known that FPGA implementations can be cost effective because they avoid expensive Non-Recurring Engineering (NRE) costs. This makes them especially useful for prototyping. Because SRAM-based devices can be reconfigured in-circuit any number of times, designers can easily perform in-system tests, modify circuit configurations and incrementally debug their designs. However, post-fabrication (both IC and PCB) reconfiguration can be exploited to implement designs that achieve high-performance with less silicon than non-reconfigured solutions. Configuring systems rapidly at run-time is referred to as run-time reconfiguration (RTR).

RTR is an implementation technique where an application is divided into temporally exclusive partitions; each partition is implemented as a single circuit configuration and is loaded onto the device as necessary during execution. With RTR, configurations are cooperative; configurations interact with one configuration typically producing a result that is consumed by some succeeding configuration. There have been several demonstrations of RTR, for example DISC, the Mojave project at UCLA [V96], and RRANN.

RRANN [EH96, EH94] was perhaps the earliest demonstration of an RTR system. It divided the popular backpropagation algorithm used to train neural networks into 3 temporally exclusive circuits (configurations), each implementing part of the overall algorithm. For neural networks of moderate size (150 neurons per layer), RRANN achieved significantly higher performance (500%) using the same number of FPGAs as statically configured networks. In addition, once the network was fully trained, only one of the configurations was necessary (the other configurations were used to train the network) and reconfiguration overhead was completely eliminated while maintaining the increased density provided by RTR.

Another early example of RTR involved a video communications system implemented using CLAy FPGAs from National Semiconductor. This system delivers real-time video at a rate of eight frames per second, and includes the steps of image transformation, quantization and run-length coding, along with BPSK modulation and demodulation. These functions are implemented using a single 5000 gate CLAy, with rapid swapping of designs used to time-share the gate array

hardware. This approach represents a 75% reduction in the necessary amount of hardware.

The Brigham Young group has also investigated the use of partial reconfiguration as a means to construct a computer with a dynamic instruction set [WH95]. This idea, which has also been discussed by Athanas and Silverman [AS93], achieves increased efficiency by using FPGA resources to hold the instructions that are needed on an application-specific basis. For example, for a neural net application partial reconfiguration enabled a 25% reduction in configuration time and a 50% reduction in hardware when compared to a system based on complete reconfiguration.

6 Fault-Tolerance

Current technology is moving in directions that tend to make FPGAs less reliable. FPGA vendors have been moving down the same path of smaller device size as the rest of the semiconductor industry. Current density in metal traces will only increase as device feature size shrinks from 0.5 um to 0.35 um and smaller, which results in a greater threat of electromigration. As transistors are physically shrunk, the amount of charge required to turn a transistor on reduces, which also makes the components more susceptible to gamma particle radiation. At the same time, FPGA vendors are moving to larger and larger dies in order to deliver more logic gates to their customers. The larger dies introduce more opportunities for failure and bigger targets for gamma particles.

The traditional response to these threats is to increase reliability through redundancy. This approach has been used by replicating components (e.g. microprocessors and ASICs) or replicating logic internal to a component (e.g. built-in self-repair (BISR)). Duplication is a particularly unattractive approach for FPGA systems given the common customer complaint that devices cost too much and don't provide enough equivalent logic gates. A better approach would be to leverage the flexible nature of FPGA devices to provide duplication at a much finer level. Conceptually, if a single logic block failed it is often possible to find an alternate placement and route for the circuit that avoids the fault. Most vendor place and route tools provide an option for reserving resources, and in the face of a fault the tool could be invoked (either automatically or manually) to search for a new placement which would be functional. The resulting system could provide reliability with very low overhead, i.e. by reserving only a few percent of the resources as spares for fault recovery. Of course, this approach results in significant system downtime. Thus, the technique will not be sufficient for mission-critical applications with hard real-time constraints. This technique also requires that the end user have the vendor place and route tools, which is usually not possible. It seems unlikely that the end consumer will wish to even know about an embedded FPGA, let alone worry about generating a new configuration for one. Finally, because each fault is distinct, each component would possibly require a unique circuit placement. These three factors combine to make the approach impractical.

Researchers at UCLA have refined these concepts into a set of techniques for achieving highly reliable FPGA systems. The essential formulation involves breaking the FPGA and circuit into tiles, which have fixed interfaces. Each tile has multiple instances, any of which can be used to produce a functional system. Intuitively, if each tile has at least one instance that does not use each of the contained resources, then any single failure can be tolerated. This approach was evaluated by considering

the PREP benchmarks as well as extrapolation to a very large FPGA. Although PREP is an industrial benchmark, the designs are rather small. The area overhead in terms of CLBs on a Xilinx 4000 is always less than 14%. For example, for PREP2, with the probability of a CLB failure set at 10%, the probability of the initial design being functional is 15%, while the probability of the tiled design being functional is 82%. For the case of a 5000 CLB design, with the probability of a CLB failure at 0.1%, the probability of the initial design being functional is less than 1%, while the probability of the fault-tolerant design being functional is 98%.

Related approaches have been developed by Hanchek and Dutt [HD96], who considered a much more restricted form of resource covering, and Liu who investigated the impact of run-time place-and-route on circuit performance [ML96].

7 Benchmarks

We don't hold out much hope for that a meaningful broad-based benchmark suite will be developed for configurable computing systems. There are a number of reasons for this.

First, high performance is typically achieved by carefully fine-tuning and tailoring a circuit description to a specific application. Most of the time, these circuits are never used anywhere else; circuits may even be optimized for a specific data set. It is unlikely that any general benchmark will apply to these circuits in a meaningful way. When implementations are this highly optimized, basic system designs (such as netlists) are not even truly portable across different FPGA architectures. Often times, performance measures are evaluating the skill of the designers rather than the system architecture or technical approach.

Second, configurable computing systems try to balance I/O and computation by carefully overlapping the two to increase throughput. It is difficult to incorporate I/O requirements in a broad benchmark. Furthermore, I/O characteristics can change dramatically based on the available hardware ports. This factor can greatly complicate the task of performing a deep and thorough performance evaluation.

We are not suggesting that it is impossible to measure performance and compare results. On the contrary, application-specific benchmarks (benchmarks that are associated with a specific challenge problem) are an effective means of comparing results, and ultimately they are the only meaningful way to compare systems.

Of course, application-specific benchmarks are no panacea. They tend to tell us only what is possible and don't always shed a lot of light on how the performance was achieved—this tends to be true for any benchmark activity. However, in these early stages as everyone experiments with a variety of implementation techniques and makes the details of these experiments available to the community at large, application-specific benchmarks can be useful both as a means of comparison (for the same challenge problem) and as a demonstration of what is feasible.

8 Conclusions And Summary

FPGA based configurable computing systems have proven effective at a range of applications, including image analysis, signal processing, and cryptography. Thus far, however, these successes have not been transferred from the research community into commercial products. In particular, we find it somewhat surprising that no commercial image analysis product uses configurable computing, as this application

area is extremely well suited to the strengths of current components and is well studied. We believe that the first commercial success will come in this area, and until this is achieved the commercial relevance of configurable computing technology will likely remain in doubt.

The most successful research systems built thus far manipulate very low level abstractions: digital circuits. We believe that this trend will continue and that it has two very important consequences. First, FPGA configurable computers cannot be classified as general-purpose systems. General purpose computing (i.e. MS Word and X-Windows) is largely sequential, dominated by control dependencies, and tends to rely on dynamic data structures. None of these characteristics maps well to hardware circuits or FPGAs. Parallelizing compilers provide the promise of highly concurrent data paths and speculative control structures, but these techniques are not likely to provide enough boost to general purpose programming languages, such as C and C++. Second, the most effective configurable computing systems will continue to be based on FPGA technologies, rather than programmable interconnect. The programmable interconnect systems use higher level abstractions than FPGA based systems, thus sacrificing efficiency and flexibility for reduced development time. This is a risky path to pursue, however, because it leads towards general-purpose programmable microprocessors. Ultimately, these devices will win any competition that sacrifices too much efficiency for reduced development time.

Based on the premise that FPGA based configurable computing will remain an important technology, but still removed from the mainstream of computing, we have reached certain conclusions about the future of commercial devices. Mainstream microprocessor vendors will not adopt FPGA blocks into their products, and barring some dramatic discovery programmable logic will not be widely available on the main datapath of high performance processors. However, it does appear likely that FPGA blocks will soon be incorporated in embedded processor devices. The main commercial benefit of this approach is to integrate system ASICs with processors for prototyping and low volume products. None the less, these devices may prove to be valuable and effective components to the configurable computing community.

Progress will continue to move forward, through heroic acts of system design and optimization from a small set of highly skilled designers. This level of effort and commitment is necessary early on in the development of any new computing technology, when abstractions and tools lag behind vision and ideas. We happen to believe that forces from the commercial markets and properties of the technology will force the configurable computing community to live with this development model for a longer time than some other communities.

References

[A93] J. M. Arnold, “The Splash 2 Software Environment”, *Proceedings of FCCM 1993.*

[AA94] A. L. Abbot, P. M. Athanas, L. Chen and R. L. Elliot, “Finding Lines and Building Pyramids with Splash 2”, *Proceedings of FCCM*, April 1994.

[AA95] P. Athanas and A. Abbot, “Real-Time Image Processing on a Custom Computing Platform”, *IEEE Computer*, vol. 28, no. 2, pp 16-24, Feb 1995.

[AB92] J. M. Arnold, D. A. Buell, and E. G. Davis, “Splash 2”, in *Proceedings of the 4’th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 316-324, June 1992.

[AC96] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider and Albertson L, “Plasma: An FPGA for Million Gate Systems”, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Feb 1996.

[AS93] P. Athanas and H. Silverman, “Processor Reconfiguration Through Instruction-Set Metamorphosis,” *IEEE Computer*, Vol. 26, No. 3, pp. 11-18, March 1993.

[BA97] R. Bittner and P. Athanas, “Wormhole Run-Time Reconfiguration,” FPGA’97, ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Feb 1997

[BRV89] P. Bertin, D. Roncin and J. Vuillemin, “Introduction to Programmable Active Memories”, *DEC Paris Research Laboratory Report #3*, 1989

[BT94] P. Bertin and H. Touati, “PAM Programming Environments: Practice and Experience,” *Proceedings of the 1993 Symposium on Research on Integrated Systems*, 1993.

[DG95] J. Darnauer, P. Garay, T. Isshiki, J. Ramirez and W. W.-M. Dai, “Silicon-on-Silicon Field Programmable Multichip Module (FPMCM) – Integrating Programmable MCM Technologies,” *IEEE Transactions on Components, Packaging, and Manufacturing Technology, part B: Advanced Packaging, Nov 1995.*

[DK95] T. H. Drayer, W. E. King, J. G. Tront, and R. W. Conners, “A MOdular Reprogrammable Real-time Processing Hardware, MORRPH”, *Proceedings of FCCM*, April 1995.

[E60] G. Estrin, “Organization of Computer Systems: The Fixed-plus Variable Structure Computer”, Proceedings of the Western Joint Computer Conference, pp. 33-40, 1960.

[EC96] C. Ebeling, D. C. Cronquist, and P. Franklin, “Rapid - Reconfigurable Pipelined Datapath,” FPL’96: 6th Annual Workshop on Field Programmable Logic and Applications, September 1996.

[EH94] J. G. Eldredge and B. L. “Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration”, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, April 1994.

[EH96] J. G. Eldredge and B. L. Hutchings, “Run-Time Reconfiguration: A Method for Enhancing the Functional Density of SRAM-Based FPGAs”, Journal of VLSI Signal Processing”, Vol. 12, 1996.

[GK90] M. Gokhale et al., “SPLASH: A Reconfigurable Linear Logic Array”, *Proceedings of the International Conference on Parallel Processing*, August 1990, pp. 526-532.

[GN95] P. Graham and B. Nelson, “A Hardware Genetic Algorithm for the Traveling Salesman Problem on Splash 2”, in W. Moore and W. Luk, editors, *Field-Programmable Logic and Applications*, pages 352-361, Oxford England, August 1995, Springer Verlag.

[H93] D. T. Hoang and D. P. Lopresti, “FPGA Implementations of Systolic Sequence Alignment”, in *Field Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping,* Springer-Verlag LNCS 705, 1993.

[HD96] F. Hanchek and S. Dutt, “Design Methodologies for Tolerating Cell and Interconnect Faults in FPGAs,” *Proceedings of ICCD*, 1996.

[LG97] D. Lewis, D. Galloway, M. van Ierssel, J. Rose, and P. Chow, “The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System,” *ACM Symposium on FPGAs*, Feb 1997.

[MD96] E. Mirsky and A.. DeHon, “MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources”, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, April 1996.

[ML96] A. Mathur and C. L. Liu, “Timing Driven Placement Reconfiguration for Fault-Tolerance and Yield Enhancement in FPGAs,” *Proceedings of the European Design and Test Conference*, 1996.

[P93] I. Page, “Parameterized Processor Generation”, *International Workshop on Field Programmable Logic and Applications*, 1993.

[PH95] R. J. Peterson and B. L. Hutchings, “An Assessment of the Suitability of FPGA-Based Systems for Use in Digital Signal Processing,” *Proceedings of the 5’th International Workshop on Field-Programmable logic and Applications*, FPL’95.

[QK94] G. M. Quenot, I. Kraljic, J. Serot and B. Zavidovique, “A Reconfigurable Compute Engine For Real-Time Vision Automata Prototyping,” *FCCM 1994.*

[RS94] R. Razdan and M. D. Smith, “A High-Performance Microarchitecture with Hardware-Programmable Functional Units”, *Micro 27*, November 1994, pp. 172-180.

[RV93] D. Ross, O. Vellacott and M. Turner, “An FPGA-based Hardware Accelerator for Image Processing,” *Proceedings of the 1993 International Workshop on Field-Programmable Logic and Applications,* 1993.

[S95] M. Shand, “Flexible Image Acquisition Using Reconfigurable Hardware,” *FCCM 1995*.

[SHM96] S. Singh, J. Hogg and D. McAuley, “Expressing Dynamic Reconfiguration by Partial Evaluation”, *Proceedings of FCCM 1996*, April 1996.

[SJ95] B. Schoner, C. Jones and J. Villasenor, “Issues in Wireless Coding Using Run-Time-Reconfigurable FPGAs,” *FCCM 1995.*

[V96] J. Villasenor et al., “Configurable Computing Solutions for Automatic Target Recognition,” *FCCM 1996.*

[VM97] J. Villasenor and W. H. Mangione-Smith, “Configurable Computing”, Scientific American, May 1997.

[WA93] M. Wazlowski et. al, “PRISM-II Compiler and Architecture,” *Proceedings of FCCM,* April 1993.

[WH95] M. J. Wirthlin and B. L. Hutchings, “A Dynamic Instruction Set Computer”, In D. A. Buell and K. L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 99-107, April 1995.