FCCM '94 --IEEE Workshop on FPGAs for Custom Computing Machines
April 10-13, Napa, CA

DPGA-Coupled Microprocessors: Commodity ICs for the Early 21st Century

André DeHon
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA, 02139

Abstract

During the past decade the microprocessor has become a key commodity component for building all kinds of computational systems. During this time frame large, reconfigurable logic arrays have exploited the same advances in IC fabrication technology to emerge as viable system building blocks. Looking at both the technology prospects and application requirements, there is compelling evidence that microprocessors with integrated reconfigurable logic arrays will be a primary building block for future computing systems. In this paper, we look at the role such components can play in building high-performance and economical systems, as well as the ripe technological outlook. We note how the tight integration of reconfigurable logic into the processor can overcome some of the major limitations of contemporary, attached reconfigurable compute engines. We specifically consider the use of integrated Dynamically Programmable Gate Array structures for the configurable logic and examine the advantages rapid reconfiguration provides in this application.

1 Introduction

Continuing advances in semiconductor processing have allowed the integration of increasing functionality into single-chip microprocessors. Today's high-performance microprocessors sport 2-3 million transistors and include multiple functional units and large on-chip memories. Microprocessors built for low-cost and embedded systems heavily integrate peripheral control to reduce the chip count for complete systems.

As technology continues to advance, room remains to enhance performance with additional, fixed functional units and reduce costs by integrating more of the computing system onto the microprocessor IC. Nonetheless, simply adding fixed functional capacity will not produce the highest performance on the broadest class of applications nor allow the construction of the broadest range of low-cost systems. Much of the economy in the use and production of microprocessors has come from their commoditization. Integration of fixed functional capacity risks overspecialization and reduced volume utilization per design investment.

For broader application, future microprocessors should dedicate a portion of their silicon real-estate to reconfigurable logic. The reconfigurable logic can be specialized in application-specific ways to provide application acceleration and in system-specific ways to serve as support logic implementing system-specific functions. A single reconfigurable microprocessor design can serve as the principal building block for a wide range of applications including personal computers, embedded systems, application-specific computers, and general- and special-purpose multiprocessors. The wide field of application for the reconfigurable microprocessor allow it to draw heavily on economies of scale and volume production.

In this paper, we look at the technology push (Section 2) and application pull (Section 3) which argue compellingly for the tight integration of reconfigurable logic into commodity microprocessors. We review contemporary efforts to accelerate application-specific computing tasks (Section 4) to emphasize the range of application where reconfigurable logic has already proven itself capable of acceleration and to review the typical shortcomings of contemporary systems. We review the Dynamically Programmable Gate Array Architecture (DPGA) (Section 5) in the context of microprocessor integrated reconfigurable logic and show that the DPGA architecture overcomes some of the limitations of contemporary reconfigurable logic systems. With this background in place, we take a broader look at the roles DPGA-coupled processors can play in future computing systems (Section 6). Finally, we look at the acceptance path for this technology (Section 7), take a reprise look at the costs and benefits relative to fixed functional units (Section 8), and look at the challenges ahead (Section 9) before concluding.

2 Technology Trends

Semiconductor processing has continued to improve steadily, allowing the fabrication of smaller and smaller devices. This trend shows little signs of abating. Effective device densities and IC capacity improve at an exponential rate. We are all now quite familiar with the progress of microprocessors where operational performance increases by roughly 60% per year while the number of gates increases by 25% per year. With 3 million gates available for today’s modern microprocessors, we can expect to have over 12 million gates available by the end of the century.

As gate densities have improved, more of the computing system has been integrated onto the microprocessor die and larger processors have been implemented. What started as minimal instruction stream control and a simple, 4-bit ALU, has grown with the available area to include multiple, 64-bit functional units including hardwired floating-point support. The basic microprocessor design has expanded to include large, on-chip memories to prevent off-chip i/o latency from significantly degrading performance. Today’s high-performance microprocessors move towards higher execution rates using aggressive pipelining and superscalar execution utilizing multiple functional units. Caches increase with instruction throughput in an attempt to prevent off-chip i/o latency from limiting effective computational throughput. Today’s cost-conscious microprocessors move to integrate common system and peripheral functions onto the IC die to reduce system cost and power consumption.

Just during the past 6-8 years, we have seen reconfigurable logic emerge as a commodity technology comparable to memories and microprocessors. Like memories, reconfigurable logic arrays rapidly adapt to new technology since the design consists of an array of simple structures. Also like memory, the regularity allows designers to focus their time on adapting the key logic structures to extract highest performance from the available technology. Each reconfigurable array can be designed and tested as a single IC design yet gains volume from the myriad of diverse applications to which the general-purpose array is applied.

Our growing microprocessors can continue their current trends by including more memory, more FPUs, more ALUs, and more system functionality, but it is not clear this will be the most judicious use of the silicon real-estate becoming available in the near future. Addition of fixed functional units will not bring about broad-based acceleration of applications *in proportion to the area these fixed units consume*. For a given application, the fixed functional units can be arranged to provide almost proportional improvements. However, each application class will require a different selection and arrangement of units to achieve such improvements. Any fixed collection of functional units, control, and data paths will necessarily be suboptimal for most applications.

Improvement in microprocessor performance and system cost can be achieved through specialization and integration. However, such specialization must be balanced against market size to maintain commodity economics. The danger is that overspecialization will reduce the market base and not allow the resulting microprocessors to benefit from commoditization.

The incorporation of reconfigurable array logic into our growing microprocessor provides an alternative growth path which allows application specialization while benefiting from the full effects of commoditization. Like modern reconfigurable logic arrays, a single microprocessor design can be employed in a wide variety of applications. Application acceleration and system adaptation can be achieved by specializing the reconfigurable logic in the target system or application.

3 Application Outlook

Despite ubiquitous use, contemporary microprocessor architectures are poorly matched to most of the applications they run. For almost any application, one can conjecture additions or modifications to prevalent microprocessor architectures which would significantly enhance the application’s performance. However, the additions differ from application to application, and there is insufficient commonality among applications to merit inclusion of such additions in a microprocessor with a broad application base. The performance advantage gained by employing specialized coprocessors for high-performance graphics, video processing, signal processing, and networking further demonstrates the performance penalty which accompanies using a general-purpose microprocessor over processors specialized to handle more limited application domains.

Incorporating reconfigurable logic into the general-purpose microprocessor, allows applications to specialize the processing hardware to match the application requirements while allowing a single microprocessor design to maintain its appeal across a broad range of application bases. Special-purpose architectures have long been recognized as one path to higher performance, application-specific computing systems. As we will review in the following section, recent research in the area of customizable computing systems demonstrates that reconfigurable logic can be effectively employed to accelerate many computational tasks. By including reconfigurable logic in the microprocessor, we combine the application-oriented benefits of architecture specialization with the economic benefits of commoditization.

4 Contemporary Reconfigurable Computing

4.1 Reconfigurable Compute Engines

Several research groups have built reconfigurable compute engines to extract high application performance at low costs by specializing the computing engine to the computation task.

- Athanas and Silverman introduce the PRISM (Processor Reconfiguration through Instruction-Set Metamorphosis) architecture which couples a programmable element with a microprocessor [21]. From each application program, new processor "instructions" are synthesized in the reconfigurable element which are designed to accelerate the application. PRISM-I and PRISM-II prototypes have been built which couple common microprocessors with Xilinx FPGAs [23].
- Cuccaro and Reese at the Supercomputing Research Center augmented a CM-2 with reconfigurable logic units by using Xilinx FPGAs in place of the CM-2 floating point processors [9]. The reconfigurable logic can be configured to perform common operations which are ill-suited to the fine-grained SIMD processing elements in the CM-2.
- The Supercomputing Research Center has built a series of programmable systolic arrays known as SPLASH [18] [7]. Each SPLASH array attaches to a host workstations and is composed from a number of Xilinx FPGAs in an array fashion along with support memory. SPLASH has seen heavy use in genome sequence matching applications [16].
- DEC Paris's PAM (Programmable Active Memories) is an array of Xilinx FPGAs attached to a host workstation [5]. The DEC team has demonstrated significant performance improvements on many applications by appropriate specialization of the PAM accelerator.
- Algotronix's CH2x4 provides a 2×4 array of CAL 1024 FPGAs along with memory in an ISA, PC-compatible card form factor [2].

Numerous other reconfigurable computing engines have been and continue to be built due to their favorable cost/performance ratios.

4.2 Application Sampling

Reconfigurable computing engines such as these have been effectively employed in a wide-rage of applications, including:

1. **Binary operations** – Reconfigurable logic can perform binary operations very efficiently and with substantial parallelism. When applications require the evaluation of large, regular binary operations, reconfigurable compute engines can offer significant advantage over the fixed logic available in typical processors. The PRISM research project demonstrates the kinds of advantages provided by this class of specialization [22].
2. **Arithmetic** – When the arithmetic operations required by a computation do not exactly match those provided by the ALU/FPU in conventional microprocessors or when the arithmetic operations admit to substantial bit-level parallelism, there is ample room to specialize reconfigurable logic to provide higher performance [11] [20] [4] [9].
3. **Encryption/Decryption/Compression** – Encryption/decryption and compression applications require application of simple sequences of arithmetic and logic operations to large datasets. The operations required are often not native to typical microprocessors. Specialized computing engines can provide the appropriate operators and make use of the parallelism and regularity in the application to extract high performance [20] [4].
4. **Sequence and string matching** – By recognizing the application's natural structure and specializing a configurable compute engine to take advantage of the structure, researchers have managed to achieve very high performance at modest costs [14] [12] [16].
5. **Sorting** – Sorting tasks exhibit natural, fine-grained parallelism. By exploiting this parallelism, large sorting tasks can be performed efficiently with sorting networks built from configurable logic [17].
6. **Physical system simulation** – Simulating physical phenomena often require repeated evaluation of state-variables using very regular computations, often of limited precision. Compute engines specialized to evaluating particular systems of equations can achieve significant advantage over general-purpose processors [4].
7. **Video and image processing** – The fine-grained, bit-level parallelism available in most image processing applications make it highly amenable to acceleration using fine-grained processing arrays (*e.g.* [4] [11]).

A common theme in these applications is that application-specific specialization of a reconfigurable computing system provides orders of magnitude higher performance than general-purpose microprocessors or workstations. The costs of the reconfigurable logic are typically less than that of the base processor/workstations being augmented. We often see "supercomputer" level performance from systems which cost orders of magnitude less to assemble. For several applications, the specialized, reconfigurable compute engine demonstrates the best performance reported on any machine.

The applications reviewed here were reported on a widely varying collection of reconfigurable architectures of various sizes. We cannot, fairly, conclude that a single reconfigurable architecture will necessarily exhibit directly comparable performance gains. Nonetheless, the ensemble demonstrates a clear potential for application acceleration via specialized reconfigurable logic across a large range of application.

4.3 Some Common Themes

One theme we see exploited in many of these application is the advantage of exploiting bit-level parallel computation. The reconfigurable array can manipulate a large number of bits in parallel, whereas conventional processor technology focusses on fixed, word-wide data manipulation using word sizes of at most 64 bits in today's microprocessors.

When ALUs are implemented in reconfigurable logic, they can be specialized to perform exactly the function required by the application. This specialization allows the use of more economical functional units both in terms of area and operational speed. This economy can enable the reconfigurable logic to support numerous functional units simultaneously to extract high performance through parallelism.

Another recurring theme is application-specific data flow between functional elements. In a conventional processor it is necessary to spend computational cycles moving data around to match the fixed functional resources and data paths provided. This weakness is often aggravated by limited data-transfer bandwidth within the processing system. In a reconfigurable compute engine, it is possible to customize the data paths between compute elements, as well as the compute elements themselves, to match the application. Rather than juggling intermediate results in and out of registers and memory, the compute engine can be customized so data flows directly from producer to consumer. The specialized data movement is often localized between portions of reconfigurable logic avoiding bandwidth bottlenecks on shared data paths.

4.4 Limitations of Current Systems

Despite the high performance contemporary reconfigurable computing systems provide, they do exhibit a couple of common limitations:

1. **Low Bandwidth and High Latency Interface** – Since contemporary reconfigurable compute engines are attached to host processors as peripherals, they are generally attached to a peripheral bus with limited bandwidth and high latency to the processor. The high overhead of communicating with the the attached reconfigurable engine limits the kinds of acceleration possible and prevents close cooperation between the fixed and reconfigurable logic units. The low bandwidth similarly makes communication expensive and limits the throughput achievable with the reconfigurable compute engine.

2. **High Reconfiguration Overhead** – Since the reconfiguration costs are high in almost all contemporary reconfigurable logic technologies, the reconfiguration time must be amortized over a large amount of processing to justify reconfiguring the compute engine. Often, this means that a single configuration must be maintained throughout an application even when different portions of the application might be accelerated with different kinds of specialized logic. This high-reconfiguration costs also makes it impossible to employ contemporary reconfigurable compute engines productively in multitasking or time-shared systems.

Tight integration of the reconfigurable logic into the base microprocessor can significantly decrease communication latency between the fixed functional units and the reconfigurable logic. Similarly, when the processor and reconfigurable logic share the same die, higher bandwidth is easily available for high throughput processing employing reconfigurable logic. By reducing the communication overhead between the base microprocessor and the reconfigurable logic, we can greatly increase the kinds of application-specific specialization which provide significant acceleration.

5 Dynamically Programmed Gate Arrays

The Dynamically Programmable Gate Array (DPGA) architecture [6] is particularly well-suited for reconfigurable computing. Unlike normal Field-Programmable Gate Arrays (FPGAs) where the function of each array element is fixed between relatively slow reconfiguration sequences, the DPGA array elements may switch rapidly among several, pre-programmed configurations. This rapid

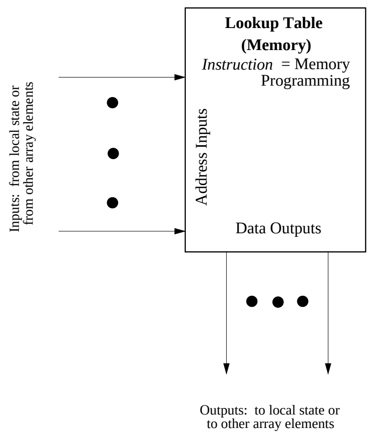
Lookup Table
(Memory)
Instruction = Memory Programming
Address Inputs
Data Outputs
Inputs: from local state or from other array elements
Outputs: to local state or to other array elements

Figure 1: Computational Unit for FPGA AE

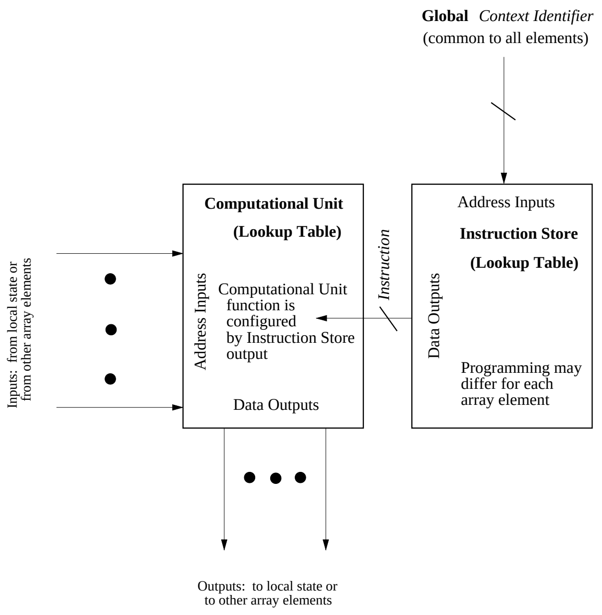
Global Context Identifier
(common to all elements)
Computational Unit
(Lookup Table)
Computational Unit function is configured by Instruction Store output
Address Inputs
Data Outputs
Instruction
Address Inputs
Instruction Store
(Lookup Table)
Data Outputs
Programming may differ for each array element
Inputs: from local state or from other array elements
Outputs: to local state or to other array elements

Figure 2: DPGA Array Element

reconfiguration allows DPGA array elements to be reused in time without significant overhead. Applications can preload multiple, specialized array personalities and switch among configurations rapidly.

Like FPGAs, DPGAs are composed of a tesselation of simple computational array elements. Each array element can perform a simple logical function on several input bits producing one or more output bits. Many modern FPGAs are best modelled as programmable lookup tables. The lookup table programming constitutes the configuration of each array-element (See Figure 1). Between array elements programmable interconnect allows the array elements to be linked up as required by the application. The interconnect in FPGAs is typically configured by programming pass gates and multiplexors. Each DPGA array-element uses a second lookup table to map a broadcast configuration selection into a local configuration (See Figure 2). The broadcast configuration behaves like the broadcast instruction in a SIMD array, telling each array-element which function to perform on the next clock cycle. Unlike a SIMD array, the indirection through a pre-programmed, context lookup table allows each DPGA array element to perform a different function in response to the broadcast configuration identifier. In a similar manner, the configurable interconnect in a DPGA array has a table of loaded configurations and selects between configurations based on the current array context identifier.

At the cost of somewhat larger array elements and interconnect, the DPGA serves as a multiple-context FPGA. Within the space of preloaded contexts the DPGA can switch personalities completely from one clock cycle to the next. The DPGA configuration lookup table effectively serves as a cache of array element configurations. By giving each array element a small configuration cache, we effectively get very high reconfiguration bandwidth by performing lookups locally at all array elements in parallel. By pipelining the context lookup, DPGA cells need run no slower than comparable complexity FPGA or SIMD logic units.

The multiple loaded context now allow us to utilize the array elements more efficiently. In the most straightforward application, this rapid reconfiguration allows a single DPGA array to be loaded simultaneously with multiple configurations. The DPGA can switch between configurations to accelerate different portions of an application. If the configurable array is too small to hold an entire acceleration logic block, the block can be partitioned across multiple context and evaluated in stages. An application can also make use of the DPGA's capacity to perform computations where the processing elements vary both spatial and temporally. This allows computational functions to be placed where intermediate data resides in the reconfigurable array, taking further advantage of application-specific locality and data flow.

Finally, DPGAs are naturally suited to conventional multitasking or fine-grained multithreading (*e.g.* April [1], *T [19]). In these applications, each thread or context may require a different array personality. By partitioning the DPGA contexts and assigning a different context, or sets of contexts, to each thread or task, array reconfiguration will not complicate sharing the processor between tasks or threads.

6 DPGA-Coupled Processor Applications

The DPGA-coupled processor can be efficiently and economically employed as the key computational building block in almost all kinds of computing systems. The computational power and flexibility this component provides will allow it to subsume the role of more traditional microprocessors as well as many processors which have been specialized for application-specific domains. Following is a sampling of DPGA-coupled microprocessor applications:

- **General-Purpose Workstations and Personal Computers** – The DPGA logic can be employed for application-specific acceleration. Each application program can configure the reconfigurable array in the manner most beneficial to the application. The operating system and applications can also specialize the DPGA logic to serve in coprocessing roles for compute-intensive functions such as compression, encryption, signal processing, or protocol processing. For many of these compute-intensive tasks, high performance is desirable in bursts, but not continually. Configuring DPGA logic to perform these tasks as needed is much more economical than employing dedicated, application-specific coprocessors. The reconfigurable logic can also be employed for minimally intrusive profiling or statistics collection. Information on the dynamics of program execution and resource utilization is becoming increasingly important for high-quality program optimization.

- **Special-Purpose Computing Machines** – The attached DPGA logic can be specialized to configure the microprocessor to the special-purpose application. The designer building the special purpose computer provides added value by specializing both the configured architecture and the application software. In this scenario, the designer can use all the techniques developed to accelerate applications in general-purpose computing setting. Further, knowing that his application is dedicated to a particular application, he can often realize greater specialization.

- **Embedded Systems** – For embedded systems, the DPGA-coupled processor can provide intelligent, integrated control with no external components for interfacing. Part of the reconfigurable logic can be employed as system-specific glue logic.

- **Multiprocessor Systems** – DPGA-coupled processors can be employed in a variety of ways to produce high-performance multiprocessors. Part of the DPGA logic array could be used to build a tightly-integrated network interface (*e.g.* [10] [19]) to adapt the microprocessor for multiprocessing. Part of the DPGA logic could be used to construct application-specific synchronization primitives or communication networks. Each node may employ the DPGA logic for task acceleration in the same manner as single processor special- and general-purpose computers. The fine-grained, bit-level parallelism provided by the DPGAs can compliment the coarse-grained parallelism which is easiest to extract from microprocessor-based multiprocessors.

7 Acceptance Path

Integration can evolve smoothly from today's research-oriented, reconfigurable compute engines to commoditized, DPGA-coupled microprocessors. Contemporary, reconfigurable compute engines (Section 4) are attached as peripherals to host workstations and personal computers and serve as a first step in the direction of processor integrated, reconfigurable logic. The next step will be to provide reconfigurable logic coprocessors which can be more tightly integrated with the fixed-logic processor in general-purpose computing systems. With minor design effort, the reconfigurable logic can be integrated onto the same die as a core processor. ASIC vendors (*e.g.* LSI Logic, VLSI Technology Inc., Texas Instruments), which provide core-processors and compiled memories could provide first-generation, FPGA- or DPGA-coupled microprocessors in the current or upcoming ASIC technologies with minor up-front design overhead. Native integration of DPGA arrays into microprocessor designs will require the processor to be designed with the attached reconfigurable logic in mind. This may entail additional processor instructions for computational cooperation between the fixed and reconfigurable logic as well as some rethink of which portions of the processor should be implemented as hardwired functional resources.

Similarly, the "programming" necessary to efficiently employ these machines can evolve towards native integration with conventional compiler technology. The earliest reconfigurable compute engines were configured through explicit, human-crafted gate-designs. Increasingly, the behavior for these reconfigurable computing engines is described at a behavioral level in hardware description languages (*e.g.* VHDL, Verilog). In contemporary cases, experts familiar with the reconfigurable architecture develop the configurations for accelerating each application. PRISM [21] and dbC [13] demonstrate that conventional programming languages can be restricted or extended to allow programmers more comfortable with programming languages to express computations in a way which can be readily compiled for implementation on reconfigurable logic. For the near future, experts and more sophisticated users can write subroutines for the reconfigurable logic and provide them as library routines. General programmers can make use of the provided library routines for program acceleration simply by accessing them like any other library routine from their high-level programming language. If we make a large library of parameterizable, hardware subroutines known to a profiling compiler, we can employ the compiler to profile applications and select the hardware configurations and library routines which best accelerate each application. The compiler can treat the replacement of sequences of instructions on the processor's fixed logic with known, hardware subroutines as a potential optimization transform and evaluate the relative merits in a manner similar to conventional compiler transforms. Eventually, hardware synthesis technology and conventional compiler technology will converge and the compiler will manage both the reconfigurable and fixed resources on the microprocessor.

8 Costs and Benefits of Reconfiguration

For fixed functionality, we will always be able to achieve higher performance or produce smaller IC dies by "fixing" the logic functionality rather than customizing a reconfigurable array. However, when we specialize a design for fixed logic functionality, we limit the range of application. The reduced market volume of the specialized die may offset any savings due to decreased die size. Further, for each advance in processing technology, design and engineering effort are required for the specialized design to track processing technology. Just as it is often more prudent to ride the commodity microprocessor technology wave and take advantage of the regular advances in process technology, it will often be more beneficial to stay with programmed, reconfigurable logic and track process technology by moving to the newest DPGA-coupled microprocessors.

We may also see the emergence of hybrid solutions like the Xilinx HardWire Gate Arrays [24]. In systems where the configured logic is employed in a fixed manner, stable designs can migrate from the reconfigurable microprocessor to a microprocessor with an equivalent gate array which can be programmed at the mask level. This provides some of the advantages of fixed, specialized logic by reducing die size while retaining many commodity advantages. The tooling cost is low since few masks are required to personalize the generic design. Further, the specialized logic can follow the hard-wired array through processing advances.

When designing native microprocessors with integrated, configurable logic, we have the opportunity to migrate functionality which has traditionally been implemented as processor fixed logic into reconfigurable logic. In particular, we can consider moving some of the more complicated control structures, especially those dealing with exception handling, into reconfigurable logic. This migration can provide considerable flexibility and a number of practical advantages, including:

- Feature interaction, particularly with exception and error conditions, have a history of being one of the hardest parts of a processor design to get correct (*e.g.* [8] [15]). Careful relocation of this interaction logic into the reconfigurable logic allows later binding and post-fabrication modification of interaction behavior.

- The fault, interrupt, and system call behavior of many, modern microprocessors is often quite mismatched with the needs of the operating system [3]. Migrating some critical control over the management of fixed resources in situation such as these into reconfigurable logic will allow the operating-system designer the freedom to specialize the control handling to better match operating-system requirements.

9 Challenges

Designing efficient and widely applicable DPGA-coupled microprocessors will provide several engineering and architectural challenges. The space of design options is large and a number of tradeoffs will be necessary to extract a balanced design allowing efficient acceleration and adaptation across a large range of applications. The key tradeoffs which must be considered when designing a DPGA-coupled microprocessor include:

- **Processor↔reconfigurable logic interfacing** – The interface between the fixed and reconfigurable logic will be one of the most important aspects of the composite architecture. The interface will determine the kinds of functions to which the reconfigurable logic can be efficiently employed. This interface will have a critical effect on the communication overhead between fixed and reconfigurable logic.

- **Grain-size** – There is no consensus among the contemporary FPGA community as to the appropriate grain-size for primitive reconfigurable elements. To a large extent, optimal array-element grain size depends on available silicon implementation technology and reconfiguration implementation techniques. Grain-size will remain an important parameter to re-evaluate with advances in processing technology, array architecture, and array application.

- **Area and pin allocation** – Microprocessor designers must partition available silicon real-estate among on-chip memories (*e.g.* caches), fixed-functional units (*e.g.* ALUs, FPUs), fixed control, and reconfigurable logic. In addition to any dedicated, microprocessor bus-interface pins, some pins should be dedicated to the reconfigurable logic for use in system-specific interfacing, reconfigurable array expansion, and specialized control and communications.

- **Multitasking and state interaction** – The reconfigurable array introduces a large amount of state associated with each computation in progress. Both the array configuration and the intermediate data values within the array are necessary to describe the computation at any point in time. In general-purpose computing systems which must share computational resources (*e.g.* time-shared or multitasking workstations, personal-computers, or multiprocessors), it is generally necessary to snapshot and restore computational state so that tasks can be interleaved in time on the same computational hardware. As noted in Section 4.4, the overhead necessary to reconfigure contemporary reconfigurable engines is quite large making such time-sharing impractical.

  The DPGA architecture can mitigate this drawback by providing on-chip, context caches for array configurations and memory for computed intermediate values. However, the number of on-chip contexts each DPGA instance provides is fixed. Sharing reconfigurable resources among a larger number of users or tasks will require provisions for offloading and restoring computations. To further mitigate the expense of unloading and reloading a context, reconfigurable-array savvy compilers can explicitly identify points in the computation where the state inside the reconfigurable array is minimal. By specializing context loading and unloading code to store and retrieve only the necessary state and reconfiguration, the compiler can produce lower overhead code for context swapping.

10 Conclusion

Microprocessors with tightly-integrated, rapidly reconfigurable logic promise to be a prime commodity building block for computing systems during the early part of the next century. We have seen that we can achieve high application performance by specializing the computational resources to the application. A microprocessor with integrated reconfigurable logic allows us to take advantage of application-specific specialization for extracting high performance while maintaining broad-based, high-volume appeal to reap the benefits commoditization. Tightly coupled DPGA processing arrays overcome the primary limitations of contemporary reconfigurable compute engines by significantly reducing the overheads associated with both processor↔array communications and array reconfiguration.

Acknowledgements

This work was supported in part under ARPA contracts N00014-91-J-1698 and DABT63-92-C-0039. Thomas Knight, Jr., Michael Bolotski, and Ian Eslick provided valuable input to the development and presentation of these ideas.

References

[1] Anant Agarwal, Beng-Hong Lim, David Kranz, and John Kubiatowicz. APRIL: A Processor Architecture for Multiprocessing. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 104–114. IEEE, May 1990.

[2] Algotronix Ltd., Edinburgh, UK. *The Configurable Logic Data Book*, 1990.

[3] Thomas Anderson, Henry Levy, Brian Bershad, and Edward Lazowska. The Interaction of Architectures and Operating System Design. In *Fourth International Conference on Architectural Support for Programming Languages*, pages 108–120. ACM, April 1991.

[4] P. Bertin, D. Roncin, and J. Vuillemin. Programmable Active Memories: A Performance Assessment. Prl report, DEC Paris Reserch Laboratory, 85, Av. Victor Hugo, 92563 Rueil-Malmaison Cedex, France, June 1992.

[5] Patrice Bertin, Didier Roncin, and Jean Vuillemin. Introduction to Programmable Active Memories. PRL Report 3, DEC Paris Reserch Laboratory, 85, Av. Victor Hugo, 92563 Rueil-Malmaison Cedex, France, June 1989.

[6] Michael Bolotski, André DeHon, and Thomas F. Knight Jr. Unifying FPGAs and SIMD Arrays. Transit Note 95, MIT Artificial Intelligence Laboratory, September 1993.

[7] D. A. Buell. A Splash 2 Tutorial. Technical Report SRC-TR-92-087, Supercomputing Research Center, Bowie, Maryland, December 1992.

[8] Robert P. Colwell. Latent Design Faults in the Development of Multiflow's TRACE/200. In *Proceedings*

of The Twenty-Second International Symposium on Fault Tolerant Computing Systems, pages 468–474. IEEE Computer Society, July 1992.

[9] 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, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 121–130, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.

[10] William J. Dally et al. The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms. *IEEE Micro*, pages 23–39, April 1992.

[11] Frederick Furtek. Arithmetic Benchmarks for the CLi6000. In Duncan A. Buell and Kenneth L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press. Paper does not appear in printed proceedings; contact author `fred@clogic.com`.

[12] Maya Gokhale, William Holmes, Andrew Kopser, Sara Lucas, Ronald Minnich, Douglas Sweely, and Daniel Lopresti. Building and Using a Highly Programmable Logic Array. *IEEE Computer*, 24(1):81–89, January 1991.

[13] Maya Gokhale and Ron Minnich. FPGA Computing in a Data Parallel C. In Duncan A. Buell and Kenneth L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 94–101, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.

[14] Dzung T. Hoang. Searching Genetic Databases on Splash 2. In Duncan A. Buell and Kenneth L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.

[15] Intel Corporation. *A90960CA16,S V594 A-4 Stepping, Errors and Exceptions*, rev. 2 edition, April 1990.

[16] Daniel Lopresti. Rapid Implementation of a Genetic Sequence Comparator Using Field-Programmable Logic Arrays. In Carlo H. Séquin, editor, *Advanced Research in VLSI*, pages 138–152, Cambridge, MA, April 1991. MIT Press.

[17] Wayne Luk, Vincent Lok, and Ian Page. Hardware Acceleration of Divide-and-Conquer Paradigms: a Case Study. In Duncan A. Buell and Kenneth L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 192–1201, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.

[18] et. al. M. Gokhale. SPLASH: A Reconfigurable Linear Logic Array. In *Proceedings of the International Conference on Parallel Processing*, pages 526–532, August 1990.

[19] Rishiyur S. Nikhil, Gregory M. Papadopoulous, and Arvind. *T: A Multithreaded Massively Parallel Architecture. In *Proceedings of the 19th International Symposium on Computer Architecture*. ACM, May 1992.

[20] M. Shand and J. Vuillemin. Fast Implementations of RSA Cryptography. In Earl Swartzlander Jr., Mary Jane Irwin, and Graham Julien, editors, *Proceedings of the 11th Symposium on Computer Arithmetic*, pages 252–259, Los Alamitos, California, June 1993. IEEE Computer Society, IEEE Computer Society Press.

[21] Harvey F. Silverman. Processor Reconfiguration Through Instruction-Set Metamorphosis. *IEEE Computer*, 26(3):11–18, March 1993.

[22] 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, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 9–16, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.

[23] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *The Programmable Gate Array Databook*, 1989.

[24] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *Product Description and Selection Guide*, April 1993.