From Silicon to Science: The Long Road to Production Reconfigurable Supercomputing

KEITH D. UNDERWOOD
Intel Corporation
and
K. SCOTT HEMMERT and CRAIG D. ULMER
Sandia National Laboratories

The field of high performance computing (HPC) currently abounds with excitement about the potential of a broad class of things called *accelerators*. And, yet, few accelerator based systems are being deployed in general purpose HPC environments. Why is that? This article explores the challenges that accelerators face in the HPC world, with a specific focus on FPGA based systems. We begin with an overview of the characteristics and challenges of typical HPC systems and applications and discuss why FPGAs have the potential to have a significant impact. The bulk of the article is focused on twelve specific areas where FPGA researchers can make contributions to hasten the adoption of FPGAs in HPC environments.

Categories and Subject Descriptors: B.6.1 [**Logic Design**]: Design Styles—*Logic arrays*

General Terms: Design, Performance, Standardization

Additional Key Words and Phrases: FPGA, HPC, reconfigurable computing

ACM Reference Format:
Underwood, K. D., Hemmert, K. S., and Ulmer, C. D. 2009. From silicon to science: The long road to production reconfigurable supercomputing. ACM Trans. Reconfig. Technol. Syst. 2, 4, Article 26 (September 2009), 15 pages. DOI = 10.1145/1575779.1575786.
http://doi.acm.org/10.1145/1575779.1575786.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
Authors' addresses: K. D. Underwood, Intel Corporation, P.O. Box 5200, MS-1319, Albuquerque, NM 87185-1319; email: kdunder@gmail.com; K. S. Hemmert and C. D. Ulmer, Sandia National Labs., P.O. Box 5800, Albuquerque, NM 87185-1319.

1. INTRODUCTION

In the world of high performance computing (HPC), the last five years have been filled with excitement about the promise of a new wave of accelerator technology. Accelerators ranging from the ClearSpeed processor [ClearSpeed Technology 2008] to the IBM Cell processor [Gschwind et al. 2006] have been introduced alongside the long explored FPGA based systems such as the Cray XD1 [Cray Canada, Inc. 2005a; Fahey et al. 2005] and the SRC-7 [SRC Computers, Inc. 2007]. Despite boundless energy and enthusiasm around such systems, virtually no full-scale systems with accelerators have been deployed at the major supercomputing sites—the RoadRunner system [Crawford et al. 2008] is the only major installation to date. Why do we see this seeming paradox in the field?

The answer lies in the fundamental mismatch between the research demonstrating the system's capabilities and the needs of the production users of the systems. On one hand, accelerators promise exactly the things that the end-user craves: order-of-magnitude performance increases, drastically more memory bandwidth, user managed memory hierarchies, and a willingness to listen to the little guys—how most perceive HPC customers. On the other hand, many things that the HPC user expects from a platform are noticeably absent from accelerator based systems. For example, application portability (see Section 3.2) is critical to the end user, but virtually unheard of in the accelerator community.

We begin with the context from which HPC users see the world along with several reasons why there is hope for FPGAs in HPC systems (Section 2). This is followed by a discussion of the specific gaps between the current state of the art and the expectations of users. We discuss twelve research areas to close those gaps with some context for the current state of the art. Ultimately, it will be the ability of the FPGA research community to meet the needs of the end user that will determine when, and if, FPGAs will be broadly deployed in HPC systems.

2. A PERSPECTIVE ON HIGH PERFORMANCE COMPUTING

High performance computing (HPC) describes a broad field that is driven by the needs of end user applications. The systems are designed around those needs; thus, they tend to have thousands of nodes and are designed with high bandwidth interconnects to enable application scalability. HPC systems also focus on creating highly reliable systems to support long application runs. Here, we give a brief overview of the expectations of applications and application developers and discuss why accelerators scare them, but have great potential for their needs.

2.1 Applications

Scientific applications have large code bases—tens of thousands to millions of lines [Rodrigues et al. 2004]—that are rarely rewritten, but are constantly evolving. This is the not-so-dusty "dusty deck" problem. Application portability is key, as applications last for decades and run on a variety of different platforms over their lifetimes. Given the diversity of system types in production at any given time, the developers generally avoid unique system features that only exist on a small subset of platforms and that may only exist for one generation. Ironically, rather than being tied to the development effort, this aversion is tied to the validation effort of maintaining unique platforms. Additionally, these large conglomerations of code rarely make a large DGEMM [Lawson et al. 1979] or FFT call.

In production, these codes leverage thousands of processors and use many terabytes of memory. Processors communicate extensively using message passing, as the memory is partitioned over many physical nodes. Despite running on thousands of nodes, the applications run for months; thus, they use checkpoint and restart to recover from system failures. To do this successfully, they must know when a failure occurs that could corrupt data. Note that, while the work of the application is primarily floating-point, the processor must execute far more integer instructions than floating-point instructions [Rodrigues et al. 2004] and those instructions are mostly used for address computations [Rupnow et al. 2006].

Arithmetic precision is a topic of great tension between the accelerator community and the application community. Accelerator enthusiasts point out that single precision floating-point arithmetic is adequate for many computations, as has been demonstrated with some applications [Scrofano et al. 2006]. However, two key factors make double precision support critical. First, many solver algorithms [Heroux et al. 2005; Balay et al. 1997] already require aggressive preconditioners to converge on real data sets and are beginning to expose the need for better than double precision. Second, and more importantly, each platform a developer supports must be validated with each revision of the application. Validation generally comes in the form of regression testing, such that an answer that is different in any but the least significant position or two poses a major problem for the validation process—even if the quality of the answer is not measurably different. While this challenge can be solved, developers will not take on the effort until a platform is widely deployed. HPC customers will not purchase something that cannot be used at all on some noticeable fraction of their workload. Most accelerators have simply given up and included double precision support [Crawford et al. 2008; ClearSpeed Technology 2008]. Thus, FPGAs will have to do the same for much of the arithmetic in most applications.

2.2 Accelerators are Daunting

There is a mismatch between the long lifetime of HPC applications (with their conservative development methodology) and the nature of modern accelerators. HPC has a multi-billion dollar code base that is growing every day. With tens to hundreds of applications running on any given system, the cost of rewriting the code base dwarfs the cost of a given system. To complicate matters further, there are many accelerators to choose from and each one has a different programming model. Worse still, with many of the current programming methodologies, the architectures (particularly FPGAs) change enough with each generation to require additional application modifications. That is, there is no high-performance forward portability for applications. This becomes a validation nightmare for the developer.

2.3 FPGAs have Potential

While FPGAs face many challenges to entering the broad HPC market, numerous factors are driving customers to seek an alternative to traditional architectures. In many cases, FPGAs have substantial advantages to offer. For example, the next generation of HPC systems is forecast to exceed 10 MW, but FPGAs promise to do more for less in terms of performance/Watt by adapting the architecture to the application. In addition, memory bandwidth has substantial impact on sustained application performance, and FPGAs have the opportunity to provide higher memory bandwidth than commodity systems. Scientific applications also have many integer operations, which FPGAs perform very well. Finally, some HPC users are expressing a desire for ExaFLOPs of performance before 2020. With such a goal, the users may be willing to try something somewhat different.

3. TWELVE STEPS FOR FPGAS TO PENETRATE HPC

There is no single silver bullet to progress reconfigurable computing technology to the point where it can penetrate the broad HPC market segment. Relatively minor flaws are significantly amplified at 10,000 node scales; thus, many areas must be addressed in an incremental way until a compelling solution is reached—presented here as a twelve step program. HPC is a harsh environment where a technology is never secure. The Moore's Law advance of microprocessors will always be putting performance and feature pressures on alternative technologies—just as it did when the killer micros overtook the traditional vector platforms in most HPC centers.

3.1 Step 1: Standardization

One of the most important activities the reconfigurable computing community can engage in is standardization, such as the work that OpenFPGA is doing [Stahlberg 2008]. The ubiquity of parallel processing in HPC only emerged after PVM [Sunderam 1990] and MPI [Message Passing Interface Forum 1997; Hempel and Walker 1999] made *write-once, run-anywhere* a reality. The competition between FPGAs, GPGPUs, and other accelerators is detrimental to the entire community. HPC customers are wary of being locked into any single approach, and today's development model means that not even the programming skills are portable across technologies; thus, standards are needed across the spectrum, from languages to libraries to data transfer [Palaniswamy 2008]. Researchers can have a huge impact in this area by simply engaging and providing an objective viewpoint.

3.2 Step 2: High Performance Forward Portability

For the foreseeable future, every processing technology will need to produce a new iteration every two to three years to remain competitive. For microprocessors, HPC applications running as is reap much of the performance advantage,[1] particularly with recompilation. In contrast, an FPGA application often requires repartitioning between hardware and software and rearchitecting of the hardware components, because the best way to perform many operations changes as the device density changes. There are not enough people skilled at programming FPGAs to port every library, much less every application, in every generation; thus, this discrepancy in high performance portability must be solved.

High performance portability could begin with architectural research to create a virtualized hardware layer (akin to PipeRench [Chou et al. 2000]) that provides high level virtualization of resources. Beyond that, an abstract machine model that has adequate semantics to express the power of reconfigurable hardware (e.g. variable bit widths) would provide a better target for compilers and an optimization point for hardware architectures. Finally, a programming model that allows the developer to express medium granularity parallelism would provide a convenient point to virtualize access to hardware resources and simplify the move to future architectures. Specifically optimizing every bit width to the problem and the platform to maximize efficiency is the antithesis of this objective.

3.3 Step 3: Enhanced Device Performance

The performance win for an accelerator has to be compelling! Since accelerators will inevitably change the application, developers want a 10× motivation to adopt a new technology, since smaller advantages would disappear to Moore's Law in 18 months. That 10× must include cost and power: performance/Watt and performance/$. Achieving this goal will require new device architectures that target scientific applications; this means providing the right support for floating-point along with the correct balance of memory, memory bandwidth, and integer support. A truly exceptional device architecture will be difficult to achieve, due to the diversity in applications; however, defining good coarse grained architectures can improve device performance by improving clock rates and silicon efficiency.

3.4 Step 4: Enhanced System Architecture

One of the fundamental challenges with all accelerators is defining the system architecture. If the history of parallel computing has taught us anything, it is that vanquishing Amdahl's law is dependent on specifically decomposing the problem to virtually eliminate serial portions of the code. To accomplish this, FPGAs in an HPC environment must be treated as first class citizens, similar to the examples in Figure 1. FPGAs can be placed so that they can process the network data stream [Underwood et al. 2001] (Figure 1(a)), as a true peer on the network (Figure 1(b)), on the processor bus (Figure 1(c)), or in a variety of other locations (see El-Ghazawi et al. [2008] and Gokhale and Graham [2005] for a more complete taxonomy). The only requirement is that the FPGA be

[1]HPC applications are already parallelized and scale well on multicore processors.

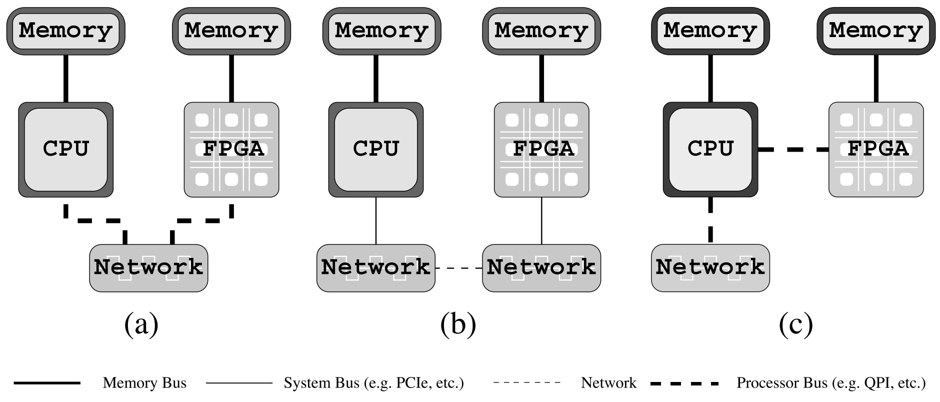

Fig. 1. System architectures: (a) FPGA capable of processing network traffic; (b) FPGA as a network peer; (c) FPGA on processor bus.

allowed to process data concurrently with the processor and that it be allowed to initiate its own network transactions. This places requirements on both the architecture and the programming model and leaves numerous open questions. For example: what is the right memory architecture for an FPGA? How many FPGAs should be on a node? Should the system be a collection of heterogeneous nodes (Figure 1(a) and (c)) or a heterogeneous collection of homogeneous nodes (Figure 1(b))? The answer should be based on system level simulation, but the first step along this path is for researchers to stop treating every interface as if it were a PCI bus. Many systems are more powerful than the "copy data in, do work, copy data out" model that most application examples use.

3.5 Step 5: Simplify Library Usage

Standard libraries make it easy for FPGAs to have an impact. Some accelerators already interface to the standard BLAS [Lawson et al. 1979] libraries (ClearSpeed, GPUs). FPGAs need to extend this trend to capture libraries that are relevant to HPC, such as sparse solvers. The key here is for the community to provide a well maintained (preferably open source) solution that integrates FPGA acceleration with the commonly used libraries (Trilinos [Heroux et al. 2005], PETSc [Balay et al. 1997], ATLAS [Whaley et al. 2001], FFTW [Frigo and Johnson 1998], etc.). It should be performance-portable—it must run well across a variety of platforms. As has been often said, a rising tide lifts all boats. This can be achieved at relatively low effort by adapting the typical application example methodology of many academic efforts to integrate with standard software packages.

3.6 Step 6: Define Concurrent APIs

Standard library interfaces are critical, but few applications use standard libraries in a way that is easy to accelerate with FPGAs. For example, instead of a single 1D FFT call that is 2 million points, LAMMPS [Plimpton et al. 1997] performs a 3D $128 \times 128 \times 128$ double precision FFT, which becomes 2^{14}

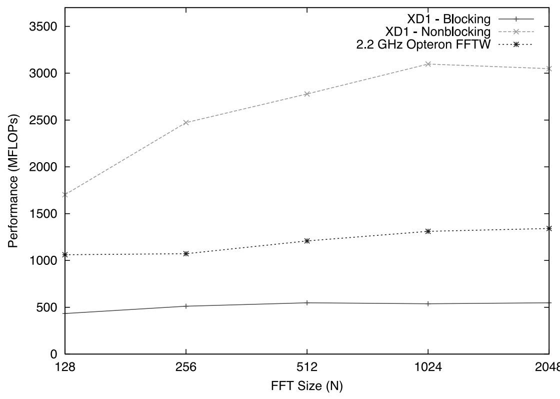

Fig. 2. Comparison of FPGA performance to processor performance for double precision FFTs.

1D FFTs of 128 points. Similarly, some sparse solvers will call thousands of small DGEMM operations instead of one large DGEMM call. Unfortunately, standard library APIs are *blocking*—they do not allow concurrency between the calling routine and the device executing the routine. While this makes perfect sense for code running on microprocessors, an accelerator must exploit the application level concurrency. By switching to a *non-blocking* interface, our previous efforts [Underwood et al. 2006] demonstrated a dramatic difference in performance. By exploiting the ability to overlap data transfer with computation on the FPGA, the FPGA moves from a factor of two performance loss to a factor of two performance gain (Figure 2); thus, one of the keys to delivering performance is to extend traditional APIs with non-blocking functionality to enable concurrency. This allows the application to expose the same concurrency as would be achieved by creating an extra thread for each API call, but at a much lower overhead and complexity.

3.7 Step 7: Perform Better Studies

Researchers have provided numerous studies of the potential of FPGAs (and other accelerators), but they often fall into a fatal trap: the study is not ultimately useful to the end user. The assumptions that simplify the study to a manageable level tend to make the study much less representative of real applications. These practices include measuring the performance of application kernels and using input sets that are unrealistically small. Other challenges include the repeated study of applications from domains that are known to be FPGA-friendly as well as hand-tuning the bit width based on a small number of input sets. In contrast, a good demonstration of the capabilities of FPGAs would measure the performance of the full application on multiple realistic inputs. It would also indicate the effort level of porting the application, including how much hand coding of the application had to be performed in HDL versus what could be done using a compiler.

Regrettably, realistic inputs are difficult to quantify. Unrealistic inputs include most input sets that are more than five years old. Applications evolve rapidly and real science grows the problem size along with the memory storage capacity of typical platforms. This problem is not unique to accelerators, but there is not even a SPEC [SPEC 2004][2] style benchmark for accelerators. The research community should therefore create and maintain a set of benchmarks (similar in concept to SPEC) based on input from the end-users. Those benchmarks should have real problems that are updated every two years as inputs, and the benchmarks themselves should be updated every four years. Furthermore, much of the work of porting benchmarks (e.g. factoring the code into pieces that can be independently accelerated) should be done by the organization maintaining the benchmark. This can be achieved at incremental effort for each graduate student doing research, while increasing the reproducibility of published results.

3.8 Step 8: Improve the Programming Environment

While there has been a lot of progress in compiling C code to FPGAs, few have achieved high efficiency and ease of use [El-Araby et al. 2007]. In addition, there has been little work on other languages (except for the SRC compiler [Poznanovic 2005], which should be commended for supporting Fortran). Unfortunately, most scientific applications are written in a mixture of C, C++, and Fortran. Compilers will have to tackle these languages to succeed in HPC. More importantly, perhaps, is the need to give feedback to the user. Traditional vector compilers, for example, give feedback to the users to indicate pieces of source code that inhibit vectorization. Modern microprocessor compilers give similar types of feedback about pieces of code that impact TLB usage or caching behavior. FPGA compilers must give specific feedback on a line-by-line basis to the user to indicate which pieces of code impact performance—to tell the user how good a job the compiler did.

Finally, the programming environment must provide a variety of traditional support functions. Prior work has explored providing some callback capabilities to implement C-library calls [Pozzi et al. 2005], and this is absolutely critical. Most C-library calls are expected by the user (e.g. even `printf()`), but these calls can be proxied as they are in modern HPC environments [Kelly and Brightwell 2005; Adiga et al. 2002].

The biggest step here is to simply provide the users with the features they expect: broad language support, reasonable performance feedback, and standard C-library calls. The next step, however, is to stand back from the problem and recognize the commonality among all modern compiler challenges: the expression of parallelism. Medium grained parallelism must be expressed in a way that is familiar to users (e.g. objects and threads). In this domain, a promising direction is found in Greaves and Singh [2008]. When enough parallelism is expressed, this objective ties back into Step 2 by facilitating the virtualization of the hardware execution resources.

[2] http://www.spec.org

3.9 Step 9: Improve the Infrastructure

Although the compiler is crucial for making FPGAs usable in HPC, it is not the full story. Application developers expect robust debuggers, performance analysis tools (e.g. VTune^{TM} and Intel®Thread Profiler), and correctness checkers (e.g. Intel®Trace Analyzer and Collector and Intel®Thread Checker). These tools must correlate hardware execution directly to source code. While some work has been done in debuggers [Graham et al. 2001; Hemmert et al. 2003], work on performance analysis is just beginning [Koehler et al. 2008]. All of these provide excellent examples of where to start, but a much more robust set of tools is a critical feature as reconfigurable computing attempts to go mainstream. If Step 8 is achieved by allowing the developer to express medium grained concurrency, then work in the parallel computing can be more directly leveraged as a starting point for debugging and performance analysis.

3.10 Step 10: Deal with Communications

Communications is perhaps the single biggest issue for reconfigurable computing. The performance of the interconnect between the FPGA and the processor dominates the performance of many applications of the technology, but the communications issue goes beyond the processor-to-FPGA link. Device manufactures could help to address the bandwidth challenges of current systems and the large logic area required by many bus interfaces by incorporating a hard macro for a high speed bus interface. Similarly, more high speed SERDES blocks would certainly help; however, it remains for the research community to deal with issues around the right integration model for FPGAs in the HPC communications infrastructure. For example, what is the right communication model? Should it be a subset of a message passing library like MPI [Patel et al. 2006], a lighter weight message passing system like SHMEM [Cray Research, Inc. 1994], or something entirely new? A good model would be one that is lightweight and has low overhead (i.e. small amounts of hardware and software to support it), and semantically matches the needs of applications.

3.11 Step 11: Enhance Reliability

One of the largest challenges of the HPC environment is the extreme scale of systems. With 10,000 to 100,000 nodes and applications that run across all of the nodes in the system, the issue of both silent data corruption and detected, but uncorrected, errors becomes enormous. Generally, procurements carry a specification that a single application running on the entire system can only fail once every 2 days. This yields an aggregate FIT (failure in time) rate for all components on the node (including system software) of 2100 at 10,000 nodes and 210 at 100,000 nodes. As observed by others [Quinn and Graham 2005], at these scales, relatively modern FPGAs perform poorly and techniques like triple modular redundancy (TMR) are both expensive and relatively easy to defeat [Quinn et al. 2007]. Unfortunately, FPGAs introduce new modes of architectural vulnerabilities that have not been extensively considered. For example, a configuration bit change can add a routing stub to a critical path and change timing, can connect two signals that cause contention, or can introduce cross-coupling between signals where none existed before. This creates two large issues for researchers to address. First, how can the error rate be reduced to an acceptable level without seriously impacting application performance? And, second, how can such errors be detected in time to prevent the corruption of the application state?

To satisfy the needs of HPC, researchers will need to find a way to localize the impact of errors in the configuration state, determine when the errors actually impact the application logic, communicate those errors to the application logic, recover the application state associated with that error, and roll back while the configuration is recovered. The solution is likely to begin with an interaction between the high level compiler and the place-and-route tools, to analyze the architecture vulnerability factor (AVF). The tools and architecture will need to set aside logic that can then be added/configured after the place-and-route for the application logic. The application logic can then leverage techniques such as those used by out-of-order execution engines to save and roll back the state.

3.12 Step 12: Provide OS Support

An FPGA that is promoted to first class citizen will require some form of operating system (OS) support. Some work has begun for general purpose OS support [Fu and Compton 2005], but HPC does not require all of the same OS support needed for the general market. For example, one user at a time on a node is fine. However, HPC does require certain types of OS functionality, such as user authentication over the network and file access protection enforcement. The fundamental challenge for FPGAs is that the user is supposed to edit the hardware itself, and it becomes difficult to trust anything loaded into the FPGA. This can even go so far as needing to protect the hardware from malicious (or just naive) users. Conceptually, these goals are low-hanging fruit, but it may require direct support from the architecture. The accelerator device must support the notion of protected state, or else a cryptographic technique to guarantee that the authentication portion of the loaded bitstream has not been corrupted. The advent of IO memory management units (IOMMUs) [Ben-Yehuda et al. 2007] may help to solve parts of this problem by providing appropriate protection of memory resources.

4. A REPORT CARD

At the request of the reviewers, we have attempted to assess the state of FPGA research in Table I. Many caveats apply here, as we are clearly not familiar with all of the work in both academia and industry. That said, after a brief review of recent activity in major FPGA conferences, we rate the current status of each step (poor, fair, good, excellent), the activity level in that area (none, low, moderate, high), and the difficultly of raising the status of the area to excellent.

There is virtually no standardization (Step 1) across the accelerator community, but the problem is not conceptually hard. The academic community simply needs to engage with existing efforts [Stahlberg 2008] and provide the

Table I. The State of FPGA Research Toward HPC

Area	Status	Activity	Difficulty
Step 1: Standardization	poor	moderate	low
Step 2: High Performance Forward Portability	poor	low	high
Step 3: Enhanced Device Performance	good	low	high
Step 4: Enhanced System Architecture	fair	none	moderate
Step 5: Simplified Library Usage	fair	low	low
Step 6: Concurrent APIs	poor	low	low
Step 7: Better Performance Studies	fair	moderate	moderate
Step 8: Improved Programming Environment	good	high	high
Step 9: Improved Infrastructure	poor	low	moderate
Step 10: Enhanced Communications	good	moderate	moderate
Step 11: Enhanced Reliability	poor	low	high
Step 12: Provide OS Support	poor	low	low

objective viewpoint that only it can. Step 2 (high performance forward portability) is in similarly bad shape, and is a much harder problem. There is relatively little activity in this area beyond work on high level language compilers, and the compilers still require far too much target specific tuning.

Step 3 (enhanced device performance) and Step 4 (enhanced system architecture) are in much better shape, though there is little academic activity attempting to improve either for HPC applications. Truly excellent architectures will be difficult to define due to the diversity of applications. Step 5 (simplified library usage) is progressing much better than the related Step 6 (concurrent APIs). Although vendors ship BLAS and FFT libraries, they have not extended the APIs to expose the concurrency. Furthermore, they do not provide higher level libraries (e.g. solvers like Trilinos [Heroux et al. 2005] and PETSc [Balay et al. 1997]). This leads into Step 7 (better performance studies). Accelerator research stands in striking contrast to high performance computing and general microprocessor optimization work. In the latter, optimization work often goes into widely available libraries (e.g. ATLAS [Whaley et al. 2001] and FFTW [Frigo and Johnson 1998]). In contrast, accelerator research tends to be a single proof of concept effort that never makes it outside the lab—despite the fact that it targets widely used core algorithms [Zhuo and Prasanna 2005; deLorimier and DeHon 2005]—and the authors of this work are no different [Underwood and Hemmert 2004; Underwood et al. 2007]. It is time for accelerator researchers to invest the extra effort and make their work applicable to Step 5 and Step 6.

Step 8 (improved programming environment) and Step 9 (improved infrastructure) go hand in hand from the perspective of an application developer, but all of the research community's attention has been focused on compilers. Thus, compilers are in relatively good shape (though much remains to be done), but research into other key components is extremely rare. The problem of communications with an FPGA (Step 10) has improved dramatically with recent parts that include hard cores for PCIExpress and 8+ Gb/s SERDES [Alfke 2008; Mansur 2008]. Thus, the hardest part of the problem is largely solved; however, these blocks remain difficult to use, because the community still needs to define semantically useful, but generally applicable, interfaces between the support for high speed signaling and the application logic.

Providing reliability levels that are commensurate with the highest end HPC systems (Step 11) is a major challenge. Traditional techniques and most current work are designed for space based systems, where things like TMR are appropriate. In HPC, TMR is far too expensive. Finally, we come to the issue of operating system support (Step 12). Although the requirements here are not dramatic, the tendency with modern system architectures has been to completely isolate the hardware to provide this protection; thus, there has been relatively little work on supporting the right things. As FPGAs become first class citizens in the system, that will have to change.

5. ALTERNATE PATHS TO SUCCESS

So far, we have discussed the advancement of FPGA technology needed to succeed in general purpose scientific computing on a broad scale. An alternate view of success would be broadly deploying FPGAs for HPC applications, without penetrating general purpose HPC. Here, we briefly discuss two such models for success and how they interact with the twelve steps.

5.1 The Appliance Model

One potential model for FPGA based accelerators is as a *computing appliance*—a system sold to perform a single application. Vendors are already finding success with this model in different niche applications. For example, TimeLogic's [2008] CodeQuest (http://www.timelogic.com/codequest.html) product utilizes FPGA hardware to perform pattern matching operations that are common in many bioinformatics applications. For financial customers, both Cray [Cray Canada, Inc. 2005b] and XtremeData [Woods 2008] have developed FPGA-based pseudo random number generators to accelerate Monte Carlo simulations for options trading. Finally, Exegy (http://www.exegy.com) [Davidson et al. 2006], and XtremeData (http://www.extremedatainc.com) have all developed data mining appliances using FPGA accelerators to process large amounts of streaming data.

Many of the characteristics discussed in Section 3 are less relevant in the appliance model, while others are still critical. At one end of the spectrum, the performance of the devices and the enhancement of system architectures remain critical, because the vendor for the appliance will need to target multiple applications while leveraging the same hardware. At the other end of the spectrum, any focus on HPC style libraries or concurrent APIs is irrelevant. Similarly, such appliances are unlikely to need novel OS support. In between, emphasis on items like standards, forward portability, programming, and development infrastructure will make it easier for companies to develop and deploy such appliances.

5.2 The Data Ingest Problem

Another area that FPGAs can readily impact is data ingest engines, where streaming data can be processed inline by FPGAs as it arrives from an external device [Gokhale et al. 2008]. This is similar to the appliance model; however, the highest priority will be the system architecture to enable high rate data flow through the FPGA and into the primary system. Reliability will also be critical, as it will be extremely difficult to detect processing errors by examining the resulting data stream.

6. CONCLUSIONS

Despite the excitement in the HPC community around accelerators, there remains a large gap between what HPC system buyers, users, and administrators expect and what accelerators (particularly FPGAs) provide. Even though FPGAs frequently have a substantial performance advantage to offer, the gap between expectations and capabilities can create a tremendous barrier to entry. Here, we have presented a collection of research and development objectives that have the potential to bridge the gap and broaden the applicability of FPGA-based systems. The twelve steps presented here cover four overarching themes that require action on the part of the FPGA community. First, make it easy to create portable applications that last for decades. Second, enhance the technology to provide a large performance win across a wider range of applications—and prove it! Third, provide the user with something that looks like a typical development infrastructure. And finally, deal with the system-level issues that are unique to HPC and that are addressed by most of the existing microprocessor-based systems targeted specifically to HPC.

REFERENCES

ADIGA, N., ALMASI, G., AND ARIDOR, Y., ET AL. 2002. An overview of the BlueGene/L supercomputer. In *Proceedings of the SC Conference on High Performance Networking and Computing*.

ALFKE, P. 2008. Virtex-5 FXT: A new FPGA platform. *Hot Chips 20*.

BALAY, S., GROPP, W. D., MCINNES, L. C., AND SMITH, B. F. 1997. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 163–202.

BEN-YEHUDA, M., XENIDIS, J., OSTROWSKI, M., RISTER, K., BRUEMMER, A., AND VAN DOORN, L. 2007. The price of safety: Evaluating iommu performance. In *Proceedings of the Ottawa Linux Symposium (OLS'07)*. 9–20.

CHOU, Y., PILLAI, P., SCHMIT, H., AND SHEN, J. P. 2000. Piperench implementation of the instruction path coprocessor. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO'33)*. ACM, New York, 147–158.

CLEARSPEED TECHNOLOGY 2008. *CSX700 Processor Product Brief*.

CRAWFORD, C. H., HENNING, P., KISTLER, M., AND WRIGHT, C. 2008. Accelerating computing with the Cell broadband engine processor. In *Proceedings of the Conference on Computing Frontiers (CF)*. ACM, New York, 3–12.

CRAY CANADA, INC. 2005a. Cray XD1 technical specifications - Release 1.4. http://www.clearspeed.com/products/document/CSX700_Product_Brief.pdf.

CRAY CANADA, INC. 2005b. *Mersenne Twister Application: Cray XD1 FPGA Programming Manual* release 1.2.1. http://www.cmf.nrl.navy.mil/CCS/help/pdfs/SHMEM/131_CrayXD1FPGADevelopment.pdf. Cray Canada, Inc.

CRAY RESEARCH, INC. 1994. *SHMEM Technical Note for C, SG-2516 2.3*.

DAVIDSON, G., COWIE, J., HELMREICH, S., ZACHARSKI, R., AND BOYACK, K. 2006. Data-centric computing with the Netezza architecture. Tech. rep., SAND2006-1853, Sandia National Laboratories.

DELORIMIER, M. AND DEHON, A. 2005. Floating point sparse matrix-vector multiply for FPGAs. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*.

EL-ARABY, E., NOSUM, P., AND EL-GHAZAWI, T. 2007. Productivity of high-level languages on reconfigurable computers: An HPC perspective. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'07)*. 257–260.

EL-GHAZAWI, T. A., EL-ARABY, E., HUANG, M., GAJ, K., KINDRATENKO, V. V., AND BUELL, D. A. 2008. The promise of high-performance reconfigurable computing. *IEEE Comput. 41*, 2, 69–76.

EXEGY 2008. http://www.exegy.com.

FAHEY, M. R., ALAM, S., DUNIGAN, T. H., VETTER, J. S., AND WORLEY, P. H. 2005. Early evaluation of the Cray XD1. In *Proceedings of the Cray User Group Annual Technical Conference*.

FRIGO, M. AND JOHNSON, S. G. 1998. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*. Vol. 3. 1381–1384.

FU, W. AND COMPTON, K. 2005. An execution environment for reconfigurable computing. In *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*. 149–158.

GOKHALE, M., COHEN, J., YOO, A., MILLER, W. M., JACOB, A., ULMER, C., AND PEARCE, R. 2008. Hardware technologies for high-performance data-intensive computing. *IEEE Comput. 41*, 60–68.

GOKHALE, M. AND GRAHAM, P. S. 2005. *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Springer.

GRAHAM, P., NELSON, B., AND HUTCHINGS, B. 2001. Instrumenting bitstreams for debugging FPGA circuits. In *Proceedings of the 9th Annual Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*. 41–50.

GREAVES, D. AND SINGH, S. 2008. Kiwi: Synthesis of FPGA circuits from parallel programs. In *Proceedings of the 16th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'08)*.

GSCHWIND, M., HOFSTEE, H. P., FLACHS, B., HOPKINS, M., WATANABE, Y., AND YAMAZAKI, T. 2006. Synergistic processing in Cell's multicore architecture. *IEEE Micro 26*, 2, 10–24.

HEMMERT, K., TRIPP, J. L., HUTCHINGS, B. L., AND JACKSON, P. A. 2003. Source level debugger for the sea cucumber synthesizing compiler. In *Proceedings of the 11th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'03)*. 228–237.

HEMPEL, R. AND WALKER, D. W. 1999. The emergence of the MPI message passing standard for parallel computing. *Comput. Stand. Interfaces 21*, 1, 51–62.

HEROUX, M. A., BARTLETT, R. A., HOWLE, V. E., HOEKSTRA, R. J., HU, J. J., KOLDA, T. G., LEHOUCQ, R. B., LONG, K. R., PAWLOWSKI, R. P., PHIPPS, E. T., SALINGER, A. G., THORNQUIST, H. K., TUMINARO, R. S., WILLENBRING, J. M., WILLIAMS, A., AND STANLEY, K. S. 2005. An overview of the Trilinos Project. *ACM Trans. Math. Softw. 31*, 3, 397–423.

KELLY, S. M. AND BRIGHTWELL, R. 2005. Software architecture of the light weight kernel, Catamount. In *Proceedings of the Cray User Group Annual Technical Conference*.

KOEHLER, S., CURRERI, J., AND GEORGE, A. D. 2008. Performance analysis challenges and framework for high-performance reconfigurable computing. *Para. Comput. 34*, 4-5, 217–230.

LAWSON, C., HANSON, R., KINCAID, D., AND KROUGH, F. 1979. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw. 5*, 3, 308–323.

MANSUR, D. 2008. Stratix IV FPGA and HardCopy IV ASIC @ 40 nm. *Hot Chips 20*.

Message Passing Interface Forum 1997. *MPI-2: Extensions to the Message-Passing Interface*. Message Passing Interface Forum.
http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.

PALANISWAMY, N. 2008. Intel quickassist. http://www.intel.com/go/quickassist.

PATEL, A., MADILL, C. A., SALDANA, M., COMIS, C., POMES, R., AND CHOW, P. 2006. A scalable FPGA-based multiprocessor. In *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)*. 111–120.

PLIMPTON, S. J., POLLOCK, R., AND STEVENS, M. 1997. Particle-mesh ewald and rRESPA for parallel molecular dynamics. In *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing*.

POZNANOVIC, D. 2005. Application development on the SRC computers, Inc. systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*. 78a–78a.

POZZI, L., IENNE, P., DUBACH, C., AND VULETIC, M. 2005. Enabling unrestricted automated synthesis of portable hardware accelerators for virtual machines. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'05)*. 243–248.

QUINN, H. AND GRAHAM, P. 2005. Terrestrial-based radiation: A cautionary tale. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.

QUINN, H., MORGAN, K., GRAHAM, P., KRONE, J., CAFFREY, M., AND LUNDGREEN, K. 2007. Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs. *IEEE Trans. Nucl. Sci. 54*, 6, 2037–2043.

RODRIGUES, A., MURPHY, R., KOGGE, P., AND UNDERWOOD, K. 2004. Characterizing a new class of threads in scientific applications for high end supercomputers. In *Proceedings of the International Conference on Supercomputing (ICS'04)*.

RUPNOW, K., RODRIGUES, A., UNDERWOOD, K., AND COMPTON, K. 2006. Scientific applications vs. SPEC-FP: A comparison of program behavior. In *Proceedings of the International Conference on Supercomputing (ICS'06)*.

SCROFANO, R., GOKHALE, M., TROUW, F., AND PRASANNA, V. K. 2006. A hardware/software approach to molecular dynamics on reconfigurable computers. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.

SPEC 2004. http://www.spec.org.

SRC COMPUTERS, INC. 2007. *Introduction to the SRC-7 MAPstation*. SPC Computers Inc. Colorado Springs, CO.

STAHLBERG, E. 2008. OpenFPGA website. http://www.openfpga.org/.

SUNDERAM, V. S. 1990. PVM: A framework for parallel distributed computing. *Concurr. Pract. Exper. 2*, 4, 315–339.

TimeLogic 2008. http://www.timelogic.com/codequest.html.

UNDERWOOD, K. D. AND HEMMERT, K. S. 2004. Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.

UNDERWOOD, K. D., HEMMERT, K. S., AND ULMER, C. 2006. Architectures and APIs: Assessing requirements for delivering FPGA performance to applications. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'06)*. ACM, New York, 1–10.

UNDERWOOD, K. D., LEVENHAGEN, M. J., AND BRIGHTWELL, R. 2007. Evaluating NIC hardware requirements to achieve high message rate PGAS support on multi-core processors. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07)*. ACM, New York, 1–10.

UNDERWOOD, K. D., SASS, R. R., AND LIGON, W. B. 2001. A reconfigurable extension to the network interface of Beowulf clusters. In *Proceedings of the Conference on Cluster Computing*. 212–221.

WHALEY, R. C., PETITET, A., AND DONGARRA, J. J. 2001. Automated empirical optimizations of software and the ATLAS project. *Para. Comput. 27*, 1–2, 3–35.

WOODS, N. 2008. FPGA acceleration of European options pricing.

XtremeData 2008. http://www.xtremedatainc.com.

ZHUO, L. AND PRASANNA, V. K. 2005. Sparse matrix-vector multiplication on FPGAs. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*.

Received May 2008; revised September 2008; accepted October 2008