ARMY RESEARCH LABORATORY

FPGAs and HPC

by Daniel M. Pressel

ARL-SR-147 **January 2007**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-SR-147 **January 2007**

FPGAs and HPC

Daniel M. Pressel
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2007	Final	1 October 2005–30 September 2006

4. TITLE AND SUBTITLE
FPGAs and HPC

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Daniel M. Pressel

5d. PROJECT NUMBER
7UH7CC

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
AMSRD-ARL-CI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-SR-147

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report addresses the current uses of field programmable gate arrays (FPGAs) and their potential for use in high performance computing (HPC). FPGAs are devices programmed using languages and methodologies originally developed for describing the circuit layouts used in today's integrated circuits. As such, they are well suited for applications involving bit manipulations performed on a continuous stream of data. However, their general applicability to HPC applications is open to debate. This report details many of the issues that determine the applicability of FPGAs to different classes of problems.

15. SUBJECT TERMS
FPGA, HPC

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			(410) 278-9151
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	22	19b. TELEPHONE NUMBER (*Include area code*) Daniel M. Pressel

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

Acknowledgments

The author would like to thank the U.S. Army Research Laboratory (ARL) and the Department of Defense High Performance Computing Modernization Program for their support in this project. Additionally, thanks go to Raju Namburu, Dale Shires, and Brian Henz from ARL and Vincent Natoli of Stone Ridge Technology for their assistance and guidance in putting together this report.

1. Introduction

This report addresses the current uses of field programmable gate arrays (FPGAs) and their potential for use in high performance computing (HPC). FPGAs are devices programmed using languages and methodologies originally developed for describing the circuit layouts used in today's integrated circuits. As such, they are well suited for applications involving bit manipulations performed on a continuous stream of data. However, their general applicability to HPC applications is open to debate.

An FPGA is a chip designed to emulate a user-specified integrated circuit. As such, FPGAs can be very useful in prototyping efforts. More recently, they have been incorporated into add-on boards which frequently consist of the following:

- One or more FPGAs.
- A small to moderate amount of fast memory (static random access memory [SRAM]).
- Optionally a larger amount of dedicated on board memory (dynamic random access memory [DRAM]).
- An interface for connecting the board to the system that will be hosting it. In most cases this will be based on either the peripheral component interconnect (PCI) or the newer peripheral component interconnect extended (PCI-X) standards.

There are many companies that make these boards. Information for two of them can be accessed at the companies' websites (http://www.nallatech.com and http://www.celoxica.com).

Xilinx and Altera are the major manufacturers of the FPGAs. Many of their products include additional features such as the following:

- A small amount of on chip fast memory.
- A complete embedded PowerPC processor.
- Specialized circuitry such as analog-to-digital converters.
- Digital signal processor (DSP) slices, from which entire DSPs and other types of computational units can be efficiently constructed.

More information can be accessed at the companies' websites (http://www.xilinx.com and http://www.altera.com).

2. FPGA Uses

There are many uses for FPGAs, and currently, most of the following them have nothing to do with HPC. Examples of non-HPC applications for FPGAs include the following:

- Replacing application specific integrated circuits (ASICs) in a design in an attempt to reduce the time to market.
- Replacing ASICs in a low volume product in an attempt to save money (ASICs are cheaper, but only when ordered in quantity).
- Temporarily replacing ASICs (or possibly even the processor itself) during the design phase of a project. This would make it easier to test out the merits of alternative designs before committing to one.
- Accelerating applications that are primarily composed of a large number of bit manipulations, including signal and image processing and cryptography, which can all be HPC applications under the correct circumstances.
- Simulating networks, which would otherwise be an HPC application.
- Simulating ASICs prior to their production.

FPGAs may be applied to HPC applications in the following distinct ways:

- The acceleration of bit manipulations (as just mentioned).
- Integer calculations.
- Fixed-point (sometimes called decimal) calculations.
- Floating-point calculations using dedicated adders, multipliers, and/or fused multiply-add units. These may be formed entirely from reconfigurable cells, or they may be formed from DSP slice technology combined with glue logic created out of reconfigurable cells.
- Floating-point calculations using soft implementations of embedded microprocessors (the Xilinx Virtex 2 and Virtex 4 FPGAs currently support a maximum of two such microprocessors per FPGA).

3. Mapping HPC Applications to FPGAs

When discussing the mapping of HPC applications to FPGAs, it is important to gain insights into how this might be done based on historical experience in moving HPC applications to new platforms. In particular, four case studies will be considered:

1. Many early parallel architectures meant for HPC used large numbers of very simple processors (e.g., 1-bit processors in the DAP and early Connection Machine and 4-bit processors in the MasPar MP-1). These processors were well-suited for calculations involving simple bit manipulations and were frequently a good choice for integer and fixed-point calculations. However, experience rapidly demonstrated their limitations when it came to floating-point calculations. Interestingly, it was found for some signal and image processing applications that the number-theoretic transform could be used in place of the floating-point intensive fast fourier transform (FFT) (*1*). Therefore, it can be seen that by significantly altering the algorithm, they were able to change a floating-point intensive application into a bit-manipulation intensive algorithm that was well suited to the new systems.

2. Many of the early parallel architectures used processors that ranged from very weak (1–4-bit processors) to relatively weak (early microprocessors equipped with 32-bit floating point units capable of delivering at most a few MFLOPS[*]/processor). Therefore, it was necessary to use thousands (sometimes tens of thousand) of these processors to solve a single problem. Many applications were poorly suited to these systems. However, rewriting the applications to use embarrassingly parallel applications (e.g., Monte Carlo methods), made it possible to use these large numbers of processors. Unfortunately, some of these new algorithms were computationally less efficient than their predecessors, which complicated efforts to compare the new systems to the benchmark Cray vector processors of the time.

3. As the power of the microprocessors started to catch up with that of the Cray vector processors, it became possible to discuss using significantly less than 1000 processors in an HPC application. However, it was found that codes based on implicit algorithms were difficult to parallelize in a distributed memory environment. As a result, today one finds that many of the most frequently used applications on distributed memory parallel architectures use an explicit algorithm, which supports the concept of scaled speedup (also known as soft scalability, as opposed to the traditional metric of fixed size speedup, also known as hard scalability) (*2*). Again, there were some misgivings about this since the

[*] MFLOPS = million floating point operations per second.

most computationally efficient algorithms on Cray vector processors were generally considered to use an implicit formulation.

4. With the advent of reduced and complex instruction set computer architectures with large caches (1 megabyte [MB] or larger), many researchers noted two things. The first was that vector-tuned codes ran very inefficiently on the new processors, frequently delivering less than 1% of peak (compared to the 20%–50% of peak many applications were reputed to achieve on Cray vector processors). The second thing that was noted was that implementation level tuning aimed at reducing the number of cache and translation look-aside buffer misses could frequently improve performance by a factor of 2–10 or more. Additionally, many of the Cray optimized applications were out-of-core solvers relying on the fast, solid state disk with which the Cray X/MP, Y/MP, etc. were equipped. In fact, some of these applications could have run slightly more efficiently on the Crays as in-core solvers, but they were vastly easier to schedule and therefore provided faster turn around times when run as out-of-core solvers. The new parallel architectures lacked solid state disks but, in general, had significantly more generous amounts of main memory (as did the Cray 2). Therefore, one could frequently improve performance by one or more orders of magnitude if the application could be rewritten as an in-core solver, when using enough processors to support that mode of operation without paging (on distributed memory architectures, the number of processors used determines the amount of available memory).

From this it can be seen that, when trying to take advantage of a new architecture with dramatically different characteristics, it may be necessary to adapt significantly different (and even entirely new) approaches to solving the problem. In particular, it would be interesting to investigate the substitution of the number-theoretic transform for a 64-bit FFT in one or more appropriate HPC applications. While there are multiple ways in which one can view an FPGA, two of the most productive, in terms of HPC, are likely to be the following:

1. A huge number of 1-bit processors.
2. A significant number of computational units that need to be combined as some sort of systolic array so that the memory bandwidth/requirements are well-matched to the limitations of the hardware (*3*).

4. Peak Performance of FPGAs vs. Microprocessors

There are a variety of ways in which one might compare the performance of an FPGA to that of a microprocessor. This report considers four such comparisons to demonstrate that it is possible for FPGAs to be the right tool for one job, a reasonable tool for a second job, and a questionable tool for other jobs. For all four of these cases, a comparison will be made between a 200–500-MHz FPGA based on the Xilinx Virtex 4 FPGA with 200,000 cells (with at least one usable gate

per cell) and the 3.6-GHz Intel Pentium 4 processor. Both of these are state-of-the-art chips. The Pentium 4 is equipped with 1 MB of cache. The Virtex 4 has some on-chip SRAM, up to 16 MB of off-chip SRAM, and optionally hundreds of megabytes/several gigabytes of dedicated on-board DRAM. The four comparisons of an FPGA's performance to that of a microprocessor are as follows:

1. The simplest use of an FPGA is to perform a series of bit manipulations. Most frequently, this would be done in place of an ASIC. Since the usable clock rate of an FPGA can decrease as the percentage of gates in use increases, it will be assumed that one can construct a 200-MHz system using 200,000 usable gates, producing 40-trillion bit operations per second. In contrast, the Pentium 4 can carry out one SSE2/SSE3 operation on 128 bits per cycle. This translates into 461-billion bit operations per second. Both of these numbers are, of course, peak speeds. Clearly, the FPGA has a significant performance advantage over the Pentium 4.

2. One can also use the FPGA to emulate up to 256 DSPs interconnected in a systolic array. In the case of the Virtex 4, these DSPs would be clocked at 500 MHz and implement 18-bit fixed-point arithmetic. This comes out to 128–256 billion operations per second. The closest comparison for the Pentium 4 would be to use SSE2/SSE3 operations on eight 16-bit operations per cycle. This comes out to 29-billion operations per second. The Virtex 4 is still better, but not by as much. Furthermore, depending on what exactly it is you want to do and which FPGAs you will be using, either the FPGA or the Pentium 4 might be better for fixed-point arithmetic.

3. Alternatively, it is also possible to use a combination of two DSPs and some glue logic to perform 32-bit floating-point operations. Presumably, one can use four DSPs with glue logic to perform 64-bit floating-point additions (multiplications require a larger number [at least eight] of DSPs). This would come to 64 GFLOPS for 32-bit operations, or roughly 20 GFLOPS for 64-bit operations. The peak speed for a Pentium 4 when using SSE2/SSE3 is 14.4 GFLOPS for 32-bit operations and 7.2 GFLOPS for 64-bit operations. Clearly, if one can combine a large fraction of the DSPs on a single Virtex 4 (or multiple Virtex 2s) into a systolic array, one has a reasonable chance of beating the Pentium 4.[*]

4. Finally, one can implement a complete embedded microprocessor on the FPGA. The Vextex 4 supports a maximum of two 450-MHz PowerPC microprocessors. If they are each capable of carrying out one fused floating-point multiply-add operation per cycle, this gives a peak speed of 1.8 GFLOPS. The peak speed of the Pentium 4 when using SSE2/SSE3 is 14.4 GFLOPS for 32-bit operations and 7.2 GFLOPS for 64-bit operations. Therefore, one would expect the Pentium 4 to be the clear winner in this case. In some cases, access to the large amounts of off-chip SRAM may significantly close the gap in

[*] On systems lacking a large number of hardwired DSPs, one could do this entirely in reconfigurable logic, but it would probably be less efficient (in terms of space) and would probably operate at a lower frequency.

terms of delivered performance. However, this is not the end of the story. The following are at least three ways in which FPGAs can catch up with, and possibly even overtake, microprocessors when performing floating-point operations (*4*):

(1) Many systems use more than one FPGA per microprocessor.

(2) FPGAs, with their large amounts of tightly coupled SRAM, will usually get a very high percentage of peak on well-structured loops/loop nests. Please note, in many cases, it is possible to tune the code for these loops to achieve a high percentage of peak on the microprocessors as well.

(3) If one uses dedicated floating-point adders/multipliers/multiply-add circuitry in conjunction with "nonstandard" floating-point representations, one can significantly increase the number of floating-point operations that can fit onto a single FPGA.

As another point, the growth in the peak speed of mainstream microprocessors has significantly slowed and, in some cases, shifted into reverse while the industry adopts the concept of chip multiprocessors. In contrast, the clock rate for FPGAs is still increasing. At the same time, the number of circuit elements per FPGA is rapidly increasing with each new generation of process technology (chip-manufacturing technology). This observation has come to be known as DeHon's Law (*5*). Unfortunately, as was pointed out by Underwood and Hemmert (*4*), as the number of gates per FPGA increases, most vendors of reconfigurable systems are decreasing the number of FPGAs per system, therefore keeping the gate count per system nearly constant.

5. Metrics: Understanding the Problem

In developing viable metrics for proposed FPGA programs, it is important to first determine the most important characteristics of the program. Our office is just getting into this field, and in our opinion, the most important characteristics of the program are the following (*6*):

- Ability of the program to produce the correct answers
- Performance of the FPGA implementation of the program
- Portability of the FPGA implementation of the program
- Time and effort required to produce the FPGA implementation of the program

The first metric is self evident. From the standpoint of HPC, it only makes sense to consider using exotic hardware, such as FPGAs and general purpose graphical processing units, as application accelerators if they provide a significant boost to performance. It should be noted that for other applications (such as embedded processing) other considerations (such as time to market and energy consumption) may be more important. The requirement for a significant

boost to performance is based on the assumption that there is significant pain as well as added expense in using exotic hardware. It should also be noted that HPC is not about making a run take a millisecond instead of a second. For serial runs, HPC usually is considered to start around 24 hr of wall-clock time when one has at least one CPU month's worth of runs to make. For parallel runs, HPC runs probably start around 3 hr of wall-clock time with at least 1 CPU month's worth of runs to make. However, given the current resources throughout the U.S. Department of Defense High Performance Computing Modernization Program (HPCMP), most of these runs would be considered to be poor candidates for using FPGAs. Challenge-type runs are much more likely to be the best candidates (in terms of justifying the time, cost, and effort/pain that goes into programming an FPGA). More information can be found at the HPCMP's website (*7*). Therefore, it is suggested that in order to obtain a good tradeoff, the prime candidates should have allocations in excess of 100-million CPU seconds (30,000 CPU hours) per year, and 1 or 2 orders of magnitude above that would be even better.

When tuning/parallelizing programs, it is not uncommon for the effort to be measured in man months if not man years. It is expected that, initially, the effort involved in using FPGAs will be at least as great, and probably greater, until one has put together a fully trained staff with experience using these tools. Therefore, it is probably inappropriate to use these tools on a project that is expected to last less than 1 year. Furthermore, it is important that the project continues to use existing technology until the FPGA solution is up and running and has been fully tested.

The two remaining characteristics, performance and portability, with respect to FPGAs, are somewhat mutually exclusive. If one programs to the strengths of an FPGA and the chips (e.g., SRAM) it is packaged with, one is likely to see improved performance at the expense of losing some degree of portability. Therefore, if one is starting out with a program written in a higher level language, one will probably want the compiler/programming tools to produce a new layout for each of the prospective platforms. Anything less is equivalent to programming to some common subset of the equipment (e.g., in terms of conventional microprocessors, using software emulation of floating-point since some early microprocessors lacked floating-point hardware). It is also important to note that it is not always desirable to use all of the FPGA for a single loop nest, since that would require the FPGA to be reprogrammed (a very slow task—milliseconds to possibly as long as a second in duration) when transitioning between loop nests.

Experience has shown that, depending on the requirement, FPGAs can frequently produce speedups in the range of a factor of 2–100 (*8*). In most cases, speedups of less than a factor of 10 do not appear to justify, at this time, the effort of switching to a system augmented with FPGAs. Ideally, for parallelized applications, this improvement in performance should be achieved without significantly interfering with the scalability of the code up to at least 1000 processors.

Prior examples that justified the effort required to design and build custom HPC systems include the following:

- Signal- and image-processing for satellite reconnaissance data/sonar net data
- Quantum chromodynamics (1- and 2-year runs on IBM's GF11, as well as systems from Japan, Columbia University/Brookhaven National Lab, etc.)
- Cryptography dating back to Turing's work during World War II
- Astronomy for problems that do not lend themselves to tree-based approximations (the various incarnations of the GRAPE project)
- A few high-end chemistry projects (MD-GRAPE)

It is important to consider what types of programs are the best candidates for using a FPGA-based solution. The ideal program should have a single loop that iterates forever and is simple enough that it can be translated into VHDL (VHSIC hardware description language, where VHSIC stands for very high speed integrated circuit) that will easily fit onto the available hardware, while at the same time meeting the performance objectives previously mentioned. In reality, most HPC applications will have multiple loops that need to be placed on the hardware. This will require tradeoffs between the benefits of using a larger percentage of the hardware for a particular loop versus the cost of reprogramming the hardware when transitioning between loops. In particular, it is unlikely that one will want to place the entire basic linear algebra subprograms (BLAS) library on an FPGA at once, since at any given time nearly all of the hardware would be idle. On the other hand, if one does not place the most frequently used routines on the FPGA at the same time, then one will be spending nearly all of the time waiting for the FPGA to be reprogrammed.

Another consideration is the time required to pass data/control between the processor and the FPGA. This is an expensive process to be avoided whenever possible, as it is with virtually all approaches to using attached processors. However, this raises the question of Amdahl's Law. If we assume that the portion of the code that runs on the FPGA produces a speedup of a factor of 20, at least 95% of the work would now have to be done on the FPGA in order to achieve an overall factor of 10 speedup. If one includes reasonable assumptions concerning the cost of transitioning between the processor and the FPGA and back again, then these transitions will need to be few and far between, or one will need better than a factor of 20 speedup for the hardware assisted code and/or close to 100% of the work will need to be done on the FPGA.

Vincent Natoli, Ph.D. of Stone Ridge Technology has given presentations in which he recommends sticking to dense computational kernel and/or iterative algorithms whenever possible. His slides stress the importance of keeping the calculation entirely within the FPGA for as long a period as possible (ideally, this would be forever or at least until the end of the run).

6. Metrics: Practical Guidelines

Based on these and some other considerations which will be briefly touched on, it is suggested that the following metrics are used in judging the effectiveness of a VHDL program for use with an HPC application:

1. To what extent is the program a good candidate, based on the expected number of runs, time required per run, allocation, and number of years over which one expects to be using the program?

2. On a loop by loop basis, what are the estimated speedups ignoring overhead costs, the costs of data motion, reprogramming costs, etc? If you are not seeing a significant speedup (e.g., a factor of 10) when combining these speedups on a weighted basis, then you should probably stop or consider running some of the less efficient loops on the node's processor.

3. Consider the cost of moving data between the FPGA and main memory (or, for FPGAs with dedicated DRAM, the FPGA's DRAM and main memory) and/or between the FPGA and the microprocessor. In particular, consider the portion of this cost that ordinarily would not be overlapped with useful work by either the FPGA or the microprocessor. According to Natoli (*9*), when using an FPGA on a PCI card to execute the BLAS-1 routine for Dot Product, the FPGA will spend nearly all of its time waiting for data to transfer through the rather slow PCI bus, allowing the CPU to run the standard library routine in a fraction of the time required by the FPGA. For FPGAs with more efficient access to memory (e.g., Cray XD-1, SGI Altix, or newer cards using PCI-express) and/or their own dedicated DRAM, this might be less of a problem.

4. Estimate the number of times that all or portions of the FPGA would need to be reprogrammed during the life of an average run, and estimate the expected costs associated with that reprogramming. Adjust this number when appropriate so it only reflects the portion of time required that is not overlapped with useful work.

5. Recalculate on a loop-by-loop basis what the estimated speedups are, but include the data from steps 3 and 4. If you are still showing any speedup less than a factor of 2, you should stop. There are still too many other factors to consider, and the actual speedup will probably be even less than you have estimated.

6. If you are determined to move ahead, and your estimated speedup was on the low side, you might want to see where all the time is going. If you are spending too much time switching the work between the microprocessor and the FPGA and back again, look into the possibility of moving all of the work to the FPGA, even if some of the loops are inefficient. On the other hand, if you are spending an excessive amount of time reprogramming the

FPGA, you might want to consider approaches to reducing this time (e.g., using simpler instantiations of the loops so that they require fewer gates).

7. As was previously mentioned, most FPGAs have on-chip SRAM. Many are also equipped with off-chip SRAM and/or large amounts of dedicated DRAM. In general, some data needs to be staged in the SRAM, while other data will be streamed through the FPGA directly from main memory (or in the case of signal and image processing applications, directly from the satellite/sonar array/radar system, etc.). So far, only the cost of staging the data in the SRAM has been considered. Consider the cost of moving the data into and out of the FPGA as well. In cases where data flows directly from the gates associated with one loop into the gates associated with another loop, the cost of this also needs to be considered. An implementation is considered well balanced if the required memory bandwidth (in terms of bandwidth to/from the SRAM and to/from any other relevant units, each considered separately) is less than or equal to 100% of the peak bandwidth. If it isn't, the memory system will not be able to keep up with the demands of one or more loops, and these loops will need to be redesigned.

8. If you have made it this far and everything still looks promising, you may want to consider the possibility that things will not fit on the FPGA quite as easily as hoped. In Byoungro et al. (*10*), it is suggested that one should use "behavioral synthesis tools" to estimate the actual requirements for placing the VHDL on the FPGA. They can be used in a relatively short amount of time to check the fit and, when necessary, consider the fit of alternate implementations. Once the most promising implementation(s) have been identified, you may proceed to using the placement tool—a process that may take hours/days per implementation.

7. Metrics: Non-HPC Applications

It is nearly impossible to come up with metrics for non-HPC applications without additional guidelines. Why does one want to use FPGAs? Perhaps for one of the following reasons:

- For the fun of it
- To get your feet wet
- To save money
- To get the product to market faster
- To create a test bed for ideas before creating dedicated silicon

Some applications of interest to the military have traditionally used some combination of custom/semicustom processors and custom ASICS. Examples of these include the following:

- Real time encryption/decryption (not code breaking) of message traffic
- Other aspects of communication
- Guidance systems (especially for missiles, and antiaircraft artillery)
- Low to moderate resolution radar/sonar (high resolution [used in missile defense or for the distant early warning (DEW) line where an entire border needs to be protected] is more properly considered an embedded HPC application)
- Almost any other use of embedded electronics

Many of these areas could use FPGAs and/or other types of specialized hardware. In general, only the proponent for those systems can supply any but the most general of metrics/ requirements. It should be noted, however, that in many cases, metrics/requirements such as the following list will be just as important as getting the correct answer (and may be more important than the precise speed of the system):

- Size
- Weight
- Durability
- Energy requirements
- Compatibility with current practices
- Time to market

In many cases, the major contractor will ignore issues such as code portability or the long-term availability of system components. They may buy some of the components in large batches in order to assure the availability of those components during the initial phases of the contracts. In other cases, they may work with small specialized companies to continue the production of components long after the original supplier discontinued the parts (this is increasingly a problem with designs that use discreet components and/or components with a low scale of integration).

According to West et al. (*11*), MIT Lincoln Laboratory has a benchmark written in serial C called RASSP (rapid prototyping of application specific signal processors). They adapted it for use in benchmarking a hybrid system consisting of FPGAs, DSPs, and general purpose processors. They also used a program called RT_STAP (real time STAP) that was originally developed by the MITRE Corporation to benchmark this hardware for space-time adaptive processing.

8. Conclusions

This report has discussed the potential for using FPGAs to accelerate HPC applications. While there are clear potential benefits for certain classes of applications, one needs to be aware of the many obstacles before one can realize even a fraction of the anticipated benefits. Even so, several vendors (Cray, SRC, SGI, and Linux Networx) have developed HPC architectures that incorporate FPGAs.

Currently, one of the most serious limitations in using FPGA-based/augmented systems is the steep learning curve associated with VHDL. Therefore, a number of groups have developed proprietary tools that sit between the programmer and the VHDL compiler. These tools fall into two main categories: graphical-user-interface-based tools and compilers based on an augmented version of the C programming language.

Only time will tell the degree to which this approach to HPC is generally applicable. However, there is already strong evidence to believe that for many integer and bit-oriented applications, FPGAs are already the correct solution.

9. References

1. Hockney, R. W.; Jesshope, C. R. *Parallel Computers 2*: *Architecture, Programming and Algorithms*; Adam Hilger: Bristol, U.K. and Philadelphia, PA, 1988.

2. Gustafson, J. L. Reevaluating Amdahl's Law. *Communications of the ACM* **1998**, *31* (5), 532–533.

3. Almasi, G. S.; Gottlieb, A. *Highly Parallel Computing, second edition*; Benjamin/Cummings Publishing Company, Inc.: Redwood City, CA, 1994.

4. Underwood, K. D.; Hemmert, K. S. Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance. *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE Computer Society: Washington, DC, 2004.

5. DeHon, A. The Density Advantage of Configurable Computing. *IEEE Computer* **2000**, 41–49.

6. Namburu, R. Ph.D.; Shires, D.; Pressel, D. U.S. Army Research Laboratory: Aberdeen Proving Ground, MD. Private meeting, 2006.

7. Department of Defense High Performance Computing Modernization Program. http://www.hpcmo.hpc.mil/Htdocs/Challenge/index.html.

8. Harr, J., Ph.D. A Case Study of Application Acceleration Using Specialized Processors. Presented at the 2006 Commodity Cluster Symposium, sponsored by the Army Research Laboratory and Ohio Supercomputer Center, Baltimore, MD, July 2006.

9. Natoli, V., Ph.D. Timing Comparison: CPU vs. FPGA. Stone Ridge Technology, internal unpublished document.

10. Byoungro, S.; Diniz, P. C.; Hall, M. W. Using Estimates From Behavioral Synthesis Tools in Compiler-Directed Design Space Exploration. *Proceedings of the 40th Conference on Design Automation*; ACM Press: New York, NY, 2003.

11. West, J. M., et al. A Hybrid FPGA/DSP/GPP Prototype Architecture for SAR and STAP. Presented at HPEC 2000, The Fourth Annual Workshop on High-Performance Embedded Computing. http://www.mitre.org/tech/hpc/benchmarking.html.

NO. OF COPIES	ORGANIZATION
1 (PDF ONLY)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	US ARMY RSRCH DEV & ENGRG CMD SYSTEMS OF SYSTEMS INTEGRATION AMSRD SS T 6000 6TH ST STE 100 FORT BELVOIR VA 22060-5608
1	DIRECTOR US ARMY RESEARCH LAB IMNE ALC IMS 2800 POWDER MILL RD ADELPHI MD 20783-1197
3	DIRECTOR US ARMY RESEARCH LAB AMSRD ARL CI OK TL 2800 POWDER MILL RD ADELPHI MD 20783-1197

ABERDEEN PROVING GROUND

1	DIR USARL AMSRD ARL CI OK TP (BLDG 4600)

NO. OF COPIES	ORGANIZATION
1	PROG DIR C HENRY 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	DEP PROG DIR L DAVIS 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	DEP PROG DIR S SCHNELLER 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	HPC CTRS PROJ MGR J BAIRD 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	CHSSI PROJ MGR L PERKINS 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	NAVAL RSRCH LAB J OSBURN CODE 5594 BLDG A49 RM 15 WASHINGTON DC 20375-5340
1	NAVAL RSRCH LAB R RAMAMURTI CODE 6410 OVERLOOK AVE SW WASHINGTON DC 20375-5344
2	US AIR FORCE WRIGHT LAB J SHANG WL FIM 2645 FIFTH ST STE 6 WRIGHT PATTERSON AFB OH 45433-7912

NO. OF COPIES	ORGANIZATION
1	USAF PHILIPS LAB S WIERSCHKE OLAC PL/RKFE 10 E SATURN BLVD EDWARDS AFB CA 93524-7680
1	NVL RSRCH LAB D PAPCONSTANTOPOULOS WASHINGTON DC 20375-5000
1	AIR FORCE RSRCH LAB/DEHE R PETERKIN 3550 ABERDEEN AVE SE KIRTLAND AFB NM 87117-5776
1	NAVAL RSRCH LAB G HEBURN RSCH OCEANOGRAPHER CNMOC BLDG A491020 RM 178 STENNIS SPACE CTR MS 39529
1	AIR FORCE RSRCH LAB INFORMATION DIR R LINDERMAN 26 ELECTRONIC PKWY ROME NY 13441-4514
1	SPAWARSYSCEN (D4402) R WASILAUSKY BLDG 33 RM 0071A 53560 HULL ST SAN DIEGO CA 92152-5001
1	USAE WATERWAYS EXPERIMENT ST CEWES HV C J HOLLAND 3909 HALLS FERRY RD VICKSBURG MS 39180-6199
1	US ARMY CRD&ED AMSEL RD C2 B PERLMAN FT MONMOUTH NJ 07703
1	SPACE & NVL WRFR SYS CTR K BROMLEY CODE D7305 BLDG 606 RM 325 53140 SYSTEMS ST SAN DIEGO CA 92152-5001

NO. OF
COPIES ORGANIZATION

1 US ARMY HIGH PERFORMANCE
COMPUTING RSRCH CTR
B BRYAN
1200 WASHINGTON AVE
S MINNEAPOLIS MN 55415

1 NAVAL CMD CNTRL &
OCEAN SURVEILLANCE CTR
L PARNELL
HPC COORDINATOR 7 DIR
NCCOSC RDTE DIV D3603
49590 LASSING RD
SAN DIEGO CA 92152-6148

1 ASSOCIATE DIR
S MOORE
INNOVATIVE CMPTG LAB
CMPTR SCI DEPT
1122 VOLUNTEER BLVD STE 203
KNOXVILLE TN 37996-3450

ABERDEEN PROVING GROUND

11 DIR USARL
AMSRD ARL CI HC
J CLARKE
P CHUNG
J GOWENS
B HENZ
R NAMBURU
C NIETUBICZ
D PRESSEL
D SHIRES
R VALISETTY
C ZOLTANI
S BOGGAN