Real-Time, Dynamic Hardware Accelerators for BLAS Computation

Raymond J. Weber[1], Brock J. LaMeres[2*]
[1,2]Dept. of Electrical and Computer Engineering,
Montana State University
Bozeman, MT. USA
[1]*raymond.weber@msu.montana.edu*, [2]*lameres@montana.edu*

Justin A. Hogan[3]
[3]Research & Development Lab
S2 Corporation
Bozeman, MT. USA
[3]*justin.a.hogan@gmail.com*

* *Corresponding author: Brock J. LaMeres*

***Abstract*—**This paper presents an approach to increasing the capability of scientific computing through the use of real-time, partially reconfigurable hardware accelerators that implement basic linear algebra subprograms (BLAS). The use of reconfigurable hardware accelerators for computing linear algebra functions has the potential to increase floating point computation while at the same time providing an architecture that minimizes data movement latency and increase power efficiency. While there has been significant work by the computing community to optimize BLAS routines at the software level, optimizing these routines in hardware using reconfigurable fabrics is in its infancy. This paper begins with a comprehensive overview of the history and evolution of BLAS for use in scientific computing. In the reviews current successes in using reconfigurable computing architectures achieve acceleration. It then presents an investigation of an accelerator approach with a granularity at the logic circuit level through real-time, partial reconfiguration of a programmable fabric with static accelerator cache memory to minimize data movement. Empirical data is presented for a study on a single-FPGA.

Keywords-*Basic linear algebra subprograms; partial reconfiguration; reconfigurable computers*

*****

I. Introduction

Computer-based numerical analysis and visualization enables our science and engineering communities to tackle some of the most complex problems facing society. Performing linear algebra on large sets of data is at the core of scientific computing. Advances in the underlying device technology to perform linear algebra has enabled scientific computing to impact a wide range of fields including biological and physical sciences, geosciences, finance, medicine, energy and defense. Increasing the computational infrastructure available to researchers is a key priority of numerous federal agencies [1], [2], [3], [4]. In a 2012 budget request to congress, the National Science Foundation (NSF) stated that deploying a comprehensive cyberinfrastructure "has the potential to revolutionize every science and engineering discipline as well as education [5]." In order to continue to enhance the capability of the nation's cyberinfrastructure, novel computer architectures and technologies are needed that overcome obstacles identified by numerous researchers [6], [7], [8] that will prevent continued scaling of scientific computing clusters. To overcome these obstacles, innovations must be made that exploit massive parallel computation resources, minimize the latency of large data movement, and maximize power efficiency. The underlying engine of scientific computing is the execution of basic linear algebra subprograms (BLAS). These subprograms represent the most primitive operations needed for numerical linear algebra and are used for the majority of scientific computing algorithms today. Targeting innovations to improve the execution efficiency of BLAS will provide the greatest gain in computing performance.

In this paper, we propose an architecture for an FPGA-based hardware accelerator for BLAS that exploits real-time, partial reconfiguration. We assert that this approach will significantly increase computation of BLAS by reconfiguring the hardware during run-time to implement the BLAS primitive being executed. This provides the most efficient resource utilization by using the FPGA solely for the BLAS primitive being executed and the exact primitive argument size. This is an improvement over implementing large sets of BLAS on an FPGA in which only portions are used at any given time. This is also an improvement over creating oversized BLAS primitives that are not properly matched to the current argument size. The ability to dynamically create the optimized BLAS primitive makes our approach more than a simpler schedule but a true real-time hardware resource manager.

We also assert that this approach will reduce data movement by leaving the data within the memory of the FPGA and reconfiguring the hardware around it. This is an improvement over a complete FPGA reconfiguration in which the data is lost and must be reloaded. The approach of leaving the argument data in FPGA memory promises to scale across multiple FPGAs with a linear performance increase. This is an improvement over fixed size caches that must be swapped and/or reloaded as the design extends beyond a single device.

II. A Brief History of Scientific Computing

A. Basic Linear Algebra Subroutines

The most computationally expensive part of scientific computing is performing numerical linear algebra [9], [10], [11]. These computations include solving linear systems of equations, linear least squares problems, eigenvalue problems and singular value problems. At the core of numerical linear algebra are floating point operations on large sets of data. Increasing the efficiency of these low-level operations has the largest impact on improving performance of scientific computing. In the 1970s a group of researchers (Larson, et al) developed a set of low-level subprograms for the basic operations of numerical linear algebra [12], [13], [14], [15]. This package, known as Basic Linear Algebra Subprograms, has become the underlying engine for the majority of scientific computing algorithms in use today. BLAS are divided into three levels, depending on the type of array argument the operation is performed on. The Level 1 BLAS perform vector-vector operations (e.g., 1D/1D). The Level 2 BLAS perform matrix-vector operations (e.g., 2D/1D). The Level 3 BLAS perform matrix-matrix operations (2D/2D). The rationale for the partitioning of the subprograms into three levels is to give an indication for how much optimization can be accomplished, primarily with regards to minimizing memory access latency. Level 2 and Level 3 BLAS consume the most memory, thus, they have the most room for optimization based on the architecture of the underlying computer system [16]. Numerous packages were developed to provide higher-level numerical functionality based on BLAS. The LINPACK library, based on Level 1 BLAS, was developed in the late 1970s to provide solvers for linear equations and linear least squares [17]. The EISPACK library was also developed in the 1970s to provide numerical computation of eigenvalues and eigenvectors of matrices [18]. These original numerical packages were designed for sequentially executing computers. Thus, as computer technology advanced and parallel and distributed resources became available, these packages became highly inefficient as they ignored multi-layered memory hierarchies. As a result, they spent a considerable amount of time moving data instead of performing floating point operations.

In the 1990s the LAPACK library was developed, which consolidated the operations from LINPACK and EISPACK in addition to adding support for matrix factorizations (e.g., LU, Cholesky, QR, SVD, and Schur) [19]. The original goal of LAPACK was to optimize the operations from LINPACK and EISPACK for use on shared memory and parallel processors. The LAPACK library was designed to exploit the Level 3 BLAS, executed on multiple machines with an inherent memory hierarchy. While the LAPACK library presented the foundation for increasing the performance of scientific computation through scaling resources, it did not efficiently support heterogeneous resource scaling. As a result, the Basic Linear Algebra Communication Subprograms (BLACS) was developed to create a linear algebra oriented message passing interface across a large range of distributed computing platforms [20]. Finally, ScaLAPACK was developed in the late 1990s to accomplish the original goals of LAPACK, but using distributed, heterogeneous computers while overcoming machine dependencies [21]. Any machine with BLAS, BLACS and LAPACK installed can be utilized as a resource for computation using the ScaLAPACK libray. Improvements to the BLAS package have also been released to optimize functions for distributed computation and support for sparse matrix operations [22], [23]. Furthermore, BLAS has been augmented with support for direct parallel operation through the Parallel BLAS (PBLAS) package [24].

B. *Software Tuning of BLAS Kernels*

The history of numerical linear algebra packages just described would lead some to believe that most of the technical issues of scaling scientific computing capability have been solved and all that remains is adding computing resources. However, the ability to exploit massive amounts of parallel computation and storage resources has introduced some of the most complicated issues in computer science. The rapid evolution of computer hardware has further complicated the problem by continually adding more sophisticated, yet heterogeneous resources to growing computing clusters. In the past decade, a considerable amount of research has been focused on tuning the BLAS software for the computer architecture it is deployed on. In 2008, Goto et al. [25] presented the details of how to hand-tune Level-3 BLAS matrix-matrix operations for a variety of existing computing architectures. The authors showed how proper tuning can achieve increased performance across a variety of platforms. This work demonstrated the potential for performance improvement through proper kernel optimization, but also highlighted the complexity and time consuming nature of hand-tuning. Compilers provide an inherent level of optimization, but rely on simple analytical models of the hardware to compute machine-dependent parameters such as tile sizes and loop unrolling factors [16]. These model-driven optimizations often do not capture all of the complexities of modern architectures [26]. In 2008, a performance study by Soliman [27] of BLAS executed on an Intel Xeon multi-core system demonstrated how complex this problem is. The performance of BLAS varied widely depending on input argument sizes and how they mapped into the available processor cores and memory hierarchy. This study highlighted how having parallel, multi-core resources often does not improve performance if the software cannot exploit them efficiently. An example was shown where a two-core system outperformed a four-core system even when using fewer threads for the computation due to L2 cache latency [27].

As an improvement to model-driven optimization, researchers have been exploring the use of empirical hardware searches to determine machine-dependent parameters and then use them to automatically generate optimal BLAS libraries. In 2005, Demmel et al. [16] presented a comprehensive overview of the work in this area and indicated that the primary factor dominating BLAS kernel performance is the effective use of the machine's memory hierarchy. Other factors also contribute to performance and must be considered such as functional unit structure, the number of registers, and pipeline topologies. Due to the rapid advancement in computer hardware, it is impractical to optimize the machine-dependent kernel implementation through hand-coded programming efforts. Further, the effort to create machine-tuned compilers for such a narrowly focused application was deemed too large of a task to be justified, particularly with new hardware architectures continually being introduced. Instead, the authors proposed automatic tuning systems to empirically determine key operating parameters of the underlying hardware. These systems effectively generate a large set of BLAS kernels with different operating parameters and then measure the performance on the actual hardware. Once key parameters are determined, the system then generates the most optimal BLAS kernel code to be compiled. Demmel et al. presented two systems. The first is called Automatically Tuned Linear Algebra Software (ATLAS) and is targeted for dense matrix operations. ATLAS performs a comprehensive, empirical analysis of the computer hardware to select the optimal operating parameters. The second is called Optimized Sparse Kernel Interface (OSKI). OSKI uses a similar search algorithm as ATLAS but takes advantage of the regularity of sparse data structures to reduce the tuning time. Both systems, and their variants, provide an improvement over static heuristics or profile counts that are often stated with metrics that do not directly represent the actual architecture performance.

Research into tuning the BLAS kernels to optimize for the underlying computer hardware continues to this day. In 2008, Reddy et al. [28] proposed a new package specifically for tuning across heterogeneous, parallel hardware called HeteroPBLAS. In 2008/09, Seik et al. [29] and Belter et al. [30] presented optimization that considers a sequence of BLAS operations instead of just a single BLAS computation. In 2010, Jessup et al. [31] presented a graphical interface in order to set tuning parameters based on either heuristics or the results of a machine search. In 2013, Duchateau et al. [32] proposed generating the kernel code directly from linear algebra equations in order to create the most efficient implementation. While each of these recent contributions produced incremental advancement to BLAS kernel optimization, they are still focused on extracting machine-dependent parameters for the generation of the BLAS kernel software.

C. *Hardware Optimization for BLAS Computation*

In the last decade, there has been a parallel thread of research investigating how to best create computer hardware to execute the BLAS kernels. In this approach, the hardware is architected to accommodate the low-level BLAS. Bell et al. [7] and Szalay et al. [33] described in 2006 and 2008 respectively how Petascale and Exascale computing systems for use in numerical analysis need to be architected with balance between computational units, memory hierarchy, and I/O bandwidth if they are to be realistically scaled. In 2012, Pedram et al [34] presented a design approach for multi-core systems based specifically on computing Level-3 BLAS that promised to deliver increased computation while conserving energy. In 2012, Intel released a high-performance computing (HPC) accelerator [35] with a similar architecture to those discussed in [7], [33], [34] that provides abundant cores (up to 80 on a single chip) that are dedicated to only computation. All of this work has been focused on developing architectures with abundant, general-purpose, multi-core processors and ignores the use of more customized, heterogeneous computing resources.

Graphics Processing Units have received recent interest for use in scientific computing as they offer increased amounts of parallel computation over multi-core processors and have already been optimized for low-level arithmetic operations [36]. GPU accelerator cards have been developed for use in clusters to provide heterogeneous computing resources with promising results. In 2008, Volkov and Demmel [37] presented a benchmarking study of dense, Level-3 BLAS executed on a variety of NVIDIA GPUs. In particular, they demonstrated LU, QR and Cholesky factorization rates at over 300 GFLOPS. Numerous other studies in the last 3 years have shown how GPUs can be used to accelerate scientific computing [38], [39], [40], [41], [42]. The primary drawback of GPU acceleration is the difficulty in the programming model, which borrows much of its abstraction from graphics applications [37]. To achieve the maximum efficiency from a GPU, it requires the developer to have an intimate knowledge of the underlying architecture and how the libraries exploit the parallel resources.

D. *Reconfigurable Computing Platforms for BLAS Computation*

Reconfigurable computing is an area that promises to provide the most improvement in scientific computing, not only in terms of computation, but also in power efficiency. The theory of reconfigurable computing is that the hardware can be changed at run-time to implement the exact algorithm being executed. This is as opposed to mapping the software into fixed hardware. The primary technology in use today for reconfigurable computing is the Field Programmable Gate Array. An FPGA contains abundant, programmable logic elements that are connected through a programmable interconnect system. While the overhead associated with the programmability of an FPGA does impact system performance, FPGAs are extremely attractive for scientific computing due to the promise of massive parallelism. Figures 1 and 2 show the theoretical computation rates of a single FPGA device (64-bit and 32-bit) compared to state-of-the-art, multi-core system released during the same year [43]. These rates are accomplished by exploiting the abundant parallel resources on the FPGA without the need for cache swapping as in a multi-core CPU. Theoretical computation rates for a Virtex-6 FPGA (in 2010) reach 116 GFLOPS compared to the only 72 GFLOPS and 110 GFLOPS from the latest Intel and AMD multi-core CPUs respectively [44].

FPGA technology also promises to deliver improved energy efficiency. FPGAs are a widely-used technology, thus they have sufficient volume to warrant the most recent process

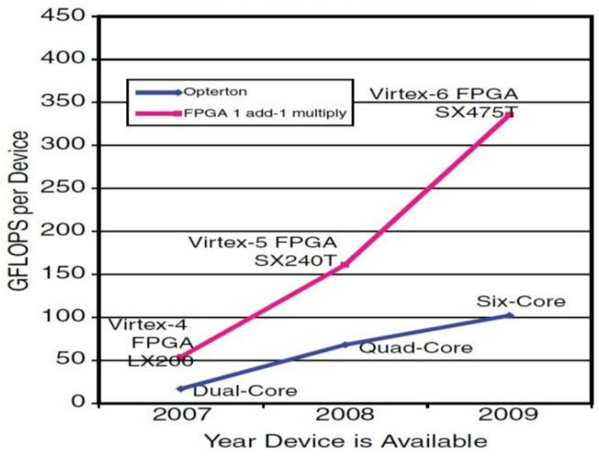

Figure 1. Predicted Opereron Processor vs. Virtex FPGA 64-Bit Performance [44].

nodes. Modern FPGAs are being fabricated in the 20 nm process node resulting in over 4M logic resources on a single device [45]. Using the latest process node and having the ability to optimize the hardware for the exact algorithm provides significant improvements in energy efficiency. FPGAs have been shown to consume 84% less energy per computation than GPUs implemented on the same node [46] and as much as an order of magnitude less power consumption compared to CPUs implemented on the same node [47]. The parallelism and energy efficiency of FPGAs has made them highly competitive with general purpose processors in fields such as scientific computing [48], [49], [50], imaging applications [51], [52], cryptology [53], and communication [54], [55].

Recently, FPGA technology has advanced to the point where their use as hardware accelerators within traditional CPU-based system is feasible. In this use-model, FPGAs are programmed to execute some of the most computationally expensive operations of an algorithm. The host CPU off-loads the computation to the FPGA when these operations are needed. In scientific computing applications, the FPGAs are most often programmed to compute linear algebra, particularly BLAS. In 2008, Zhuo & Prasanna [56] presented a design trade-off study of FPGAs as hardware accelerators for performing linear algebra based on the state-of-the-art at the time (e.g., a Xilinx Virtex-II). The authors showed how FPGAs could outperform general-purpose processors (e.g., 2.2 GHz AMD Operton) for operations such as matrix multiplication and matrix factorization, achieving computation nearing 4 GFLOPS. They also showed how FPGAs promised to scale more efficiently than increasing general-purpose CPU nodes because they do not suffer from the memory hierarchy issues. Instead, multiple FPGAs can be connected together to scale the computation resources directly.

Several high-performance computers were introduced circa 2008 by commercial vendors that used FPGAs as hardware accelerators. These systems represented the first steps by industry into reconfigurable computing. In 2008, El-Ghazawi et al. [57] presented an overview of three early commercial reconfiguration computer systems, the Cray XD1 [58], the SRC-6 [59], and the SGI Altrix/RASC [60]. Each of these systems contains general-purpose processors with FPGA hardware accelerators. These systems were benchmarked using a variety of scientific computing applications such as molecular dynamics, bioinformatics, and cryptanalysis. In each application, these systems showed significant speedup compared to a general purpose AMD Opteron processor implementation. In some cases, as much as four orders of magnitude improvement in performance, up to three orders of magnitude reduction in power consumption, and two orders of magnitude savings in cost and size were achieved by performing the computation on the FPGA hardware accelerator [57].

The study of FPGA performance for accelerating linear algebra also continues to this day. In 2010, Kestur et al. [61] presented a new comparison of the performance of BLAS

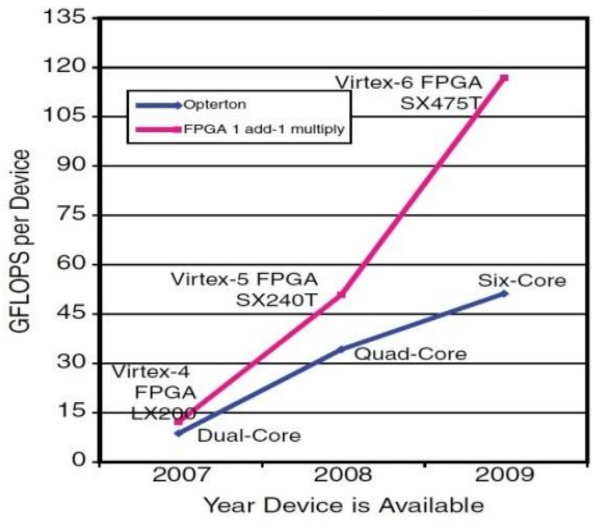

Figure 2. Predicted Opereron Processor vs. Virtex FPGA 32-Bit Performance [44].

between FPGAs, CPUs, and GPUs. This compared the performance of Level-2 BLAS on a Xilinx Virtex-5 FPGA, a 3.15 GHz Intel Core 2, and a Nvidia 9500 GT. The results showed that the FPGA was able to achieve performance on par with the other systems (>3 GFLOPS) while achieving significantly better power efficiency (>2k iterations per Joule). In 2012, Chungz et al. [62] demonstrated 6.4 GFLOPS performance on a single Xilinx Virtex-6 FPGA for a matrix multiplication. Also in 2012, Jovanovic et al. [63] demonstrated 4.5x better performance on a matrix multiplication compared to state-of-the-art 4-core processors (Intel Core2Quad and AMD Phenom X4, both at 2.8GHz). And in 2013, Cappello & Strenksi [64] presented a performance evaluation on a Xilinx Virtex-7 FPGA, which demonstrated a matrix multiplication at 180 GFLOPS when using the built-in DSP slices within the FPGA.

E. Our Contribution

While the promise of exploiting FPGAs as hardware accelerators for scientific computing is immense, one of the practical barriers to implementation is in creating a real-time, reconfigurable system that dynamically brings accelerators online when needed. The first component of such a system is abstracting this reconfiguration from the user through a hardware operating system [Agne 2014 and Andrews 2014]. The second component is understanding how the hardware reconfiguration impacts system efficiency both in terms of latency and power consumption. Our work aims to provide insight into the second component of such a system. This paper presents empirical data on the impact of real-time reconfiguration of an FPGA on both computation and power efficiency when bringing BLAS accelerators online.

III. EVALUATION OF A PROTOTYPE RC BLAS SYSTEM

In order to prove the viability of a reconfigurable system that can dynamically bring on FPGA-based BLAS accelerators, our team implemented a prototype system to compute level 1 BLAS operations for a variety of vector sizes. The system was designed to mimic an HPC consisting of a general-purpose processor augmented with hardware accelerators. In order to provide a fair performance comparison between different processing nodes often encounter in a typical HPC (i.e., the GP processor and accelerator hardware are implemented in different fabrication processes), the system was designed to contain both the host processor and the accelerators on a single FPGA. Figure 3 shows a graphical depiction of the potential improvements that can be gained by using a hardware accelerated system. The improvement in power efficiency comes from behavior that while the instantaneous power consumption is higher in the accelerated system, the time of the computation is much less than a GP system. This means the overall energy used is less. The improvement in performance comes from the behavior that the computation in the accelerated system takes less time compared to the GP system, even after considering the latency of dynamically bringing on the accelerator tile. The latency in this system is labeled “PR Operation”, referring to the partial reconfiguration of the FPGA to instantiate the accelerator.

A. Prototype Architecture

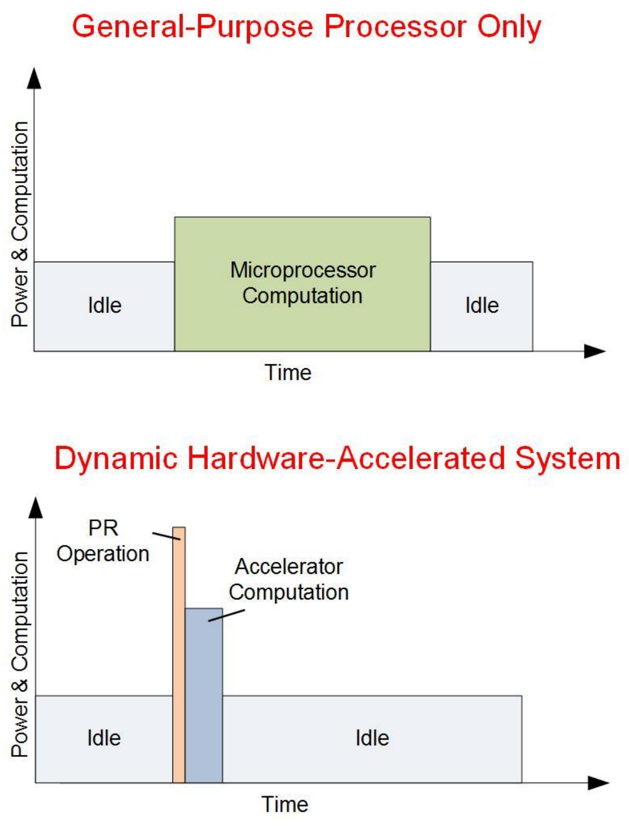

Figure 3. Graphical Depiction of Potential Performance Improvement of Accelerator.

The prototype developed consisted of a Virtex-6 (LX130T) FPGA that contained the reconfigurable fabric. The Virtex-6 contains a MicroBlaze soft processor (32-bit RISC) that represents the primary host processor. This processor takes approximately 15% of the available resources of the Virtex-6 leaving the rest for reconfigurable accelerator tiles. A separate Xilinx Spartan-6 FPGA controlled the partial reconfiguration of the accelerator tiles on the Virtex-6 FPGA.

Two types of hardware accelerator tiles were implemented for the preliminary study. The first was a simple floating point unit (FPU) that can compute the individual operations of L1 BLAS one at a time. The FPU accelerator allows faster computation than running the operations on the MicroBlaze by itself, but is not optimized for vector operations. This represents a very basic approach to hardware acceleration. The second accelerator that was implemented is a true L1 BLAS tile that implements a basic set of vector operations. In our study, the MicroBlaze processor performing the vector operations (one at a time) sets the baseline for the analysis for both computation and power consumption. The FPU was then dynamically brought online and the operations were performed again using the accelerator. Finally, the FPU was disabled and the L1 BLAS accelerator was brought online to perform the same computations for a third time. Key parameters such as configuration latency, computation speedup, and power consumption were recorded for this experiment. Figure 4 shows the prototype system developed for this experiment and the associated floorplan. For this experiment, the accelerators were developed prior to run time (instead of dynamically during run time). The accelerators were brought online by partially reconfiguring a portion of the Virtex-6 FPGA. The accelerators were disabled by programming the corresponding region of the Virtex-6 FPGA with a bitstream containing data corresponding to an un-configured FPGA state.

The L1 BLAS vector operations that were implemented for this study were double scalar vector product (DAXPY),

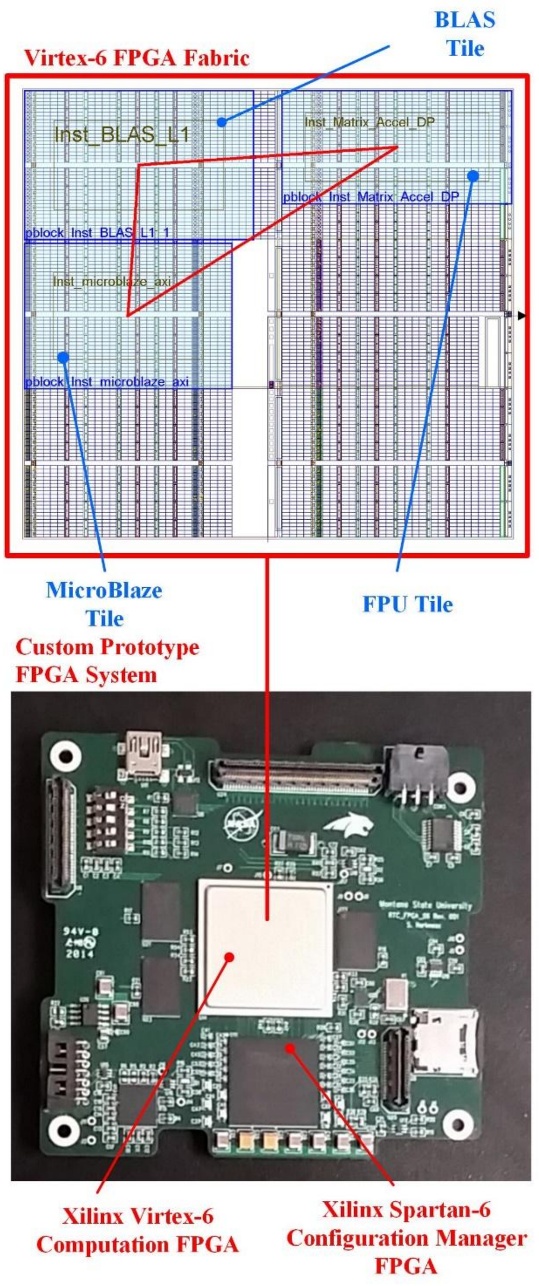
BLAS Tile
Virtex-6 FPGA Fabric
Inst_BLAS_L1
Inst_Matrix_Accel_DP
pblock_Inst_Matrix_Accel_DP
pblock_Inst_BLAS_L1_1
Inst_microblaze_axi
pblock_Inst_microblaze_axi
MicroBlaze Tile
FPU Tile
Custom Prototype FPGA System
Xilinx Virtex-6 Computation FPGA
Xilinx Spartan-6 Configuration Manager FPGA

Figure 4. Prototype System Developed to Evaluate FPGA-Based BLAS Accelerators.

double product of magnitudes (DASUM), double dot product (DDOT), double vector scalar product (DSCL), index of vector maximum (IDAMAX), and index of vector minimum (IDAMIN). On the MicroBlaze processor, these BLAS tasks were performed using routines written using basic double arithmetic instructions inherent to the processor instruction set. The instructions used were double addition (ADDD), double subtraction (SUBD), double multiplication (MULD), double division (DIVD), and compare (CMP). Each of these scalar operations were then implemented in hardware for the FPU accelerator.

B. *Experimental Results*

The first experiment performed was to evaluate the reconfiguration time, reconfiguration power consumption, and computation power consumption for each of the three systems in this work (MicroBlaze, MicroBlaze+FPU, and MicroBlaze+BLAS). Figure 5 shows the results of this experiment.In this plot, the power of the Virtex-6 FPGA was measured as it ran the BLAS operations continuously in each of the three test conditions (labeled 1, 2, and 3 in the plot) and also as the FPGA was reconfigured. The reconfigurations included a full FPGA configuration in addition to every possible accelerator configuration. In this plot the “1.0V V6 Power” is the internal core power for the Virtex-6 while the “3.3 V Power” is the Virtex-6 I/O power. During a reconfiguration (full or partial), the core power drops to its quiescent state while the I/O power increases as the bitstream is driven in. This experiment showed a variety of both intuitive and non-intuitive items. First, the operating power consumption was as expected with the MicrBlaze configuration consuming the least (1) and the MicroBlaze+FPA (2) and MicroBlaze+BLAS (3) configurations consuming the most. Also shown is a MicroBlaze+FPU+BLAS configuration. Note that the MicroBlaze+FPU+BLAS was not a valid operating mode but just measured as a validity check of the experiment setup. The second, less intuitive item of note was that the power consumption during partial reconfiguration was significantly higher (10% to 15%) than during operation. This indicates that there will be a breakeven point with respect to power efficiency for the proposed architecture in order to overcome the increased power consumption associated with partial reconfiguration. The final item of interest was the reconfiguration time of the accelerator tiles. This was measured nominally at 233 ms. Again, this non-negligible amount indicates that there will be a breakeven point with respect to computation performance for the proposed architecture in order to overcome this latency.

Table 1 shows the time and the power consumption during the reconfiguration procedures for the Virtex-6 FPGA. The size of the PR tiles in this experiment are all approximately 33% of the fabric. The time to perform PR consists of the time to read from an SD card that contains the PR bitstream files plus the time to write to the FPGA. The time to perform a full configuration of the FPGA consists of the time to read from the Xilinx platform flash memory device plus the time to write to the FPGA. There is not a linear mapping between the PR size and the percentage of the FPGA that is programmed. For example, the PR of ~33% of the FPGA takes 233ms; however, the full configuration of the FPGA does not take 3*233ms (i.e., 100%). This is due to the programming for PR being accomplished using a parallel interface in our system while the full configuration uses a serial interface.

TABLE1. FPGA Configuration Time and Power.

	Time (sec)	Power (W)
Full Configuration	2.139	0.119
Partial Reconfiguration (PR)	0.233	0.089

Table 2 lists the power consumption while computing the BLAS operations for each of the three architectures studied in this work. Also provided for reference is the power

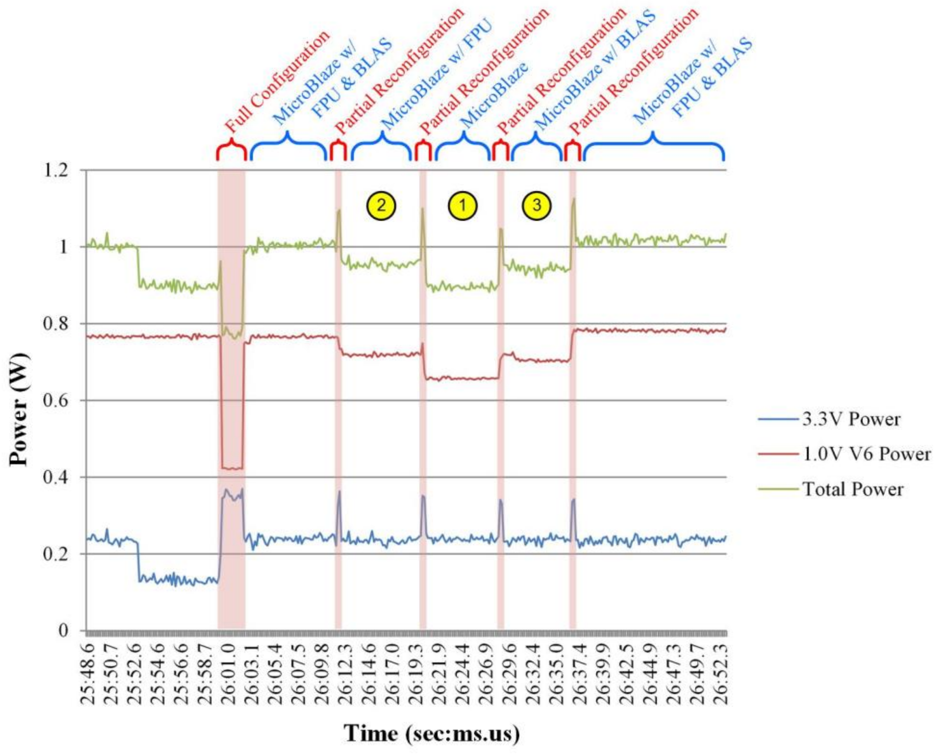

Figure 5. Prototype Empirical Results for Power Consumption and Reconfiguration Latency.

consumption of the FPGA when unprogrammed. Note that the FPU and BLAS tiles are mutually exclusive under normal operation. When not in use, the tile resources are configured back to their original, *unconfigured* state to reduce power.

TABLE 2. Computation Power Consumption

	Power (W)	Δ Power (W)
Un-programmed FPGA	410 m	-
MicroBlaze only	900 m	+ 490 m
MicroBlaze + FPU Tile	950 m	+ 50 m
MicroBlaze + BLAS Tile	970 m	+ 70 m

The primary analysis of interest when considering power efficiency is comparing how much energy it takes for each of the systems to compute the same number of BLAS operations. We setup an experiment to perform the BLAS operations mentioned above on a set of data that was swept in size. We defined the variable *N* as the number of operations computed in order to average the computation time across a set of different BLAS primitives. We then measured the time for each system to complete the computation. The first set of computations was performed by the MicroBlaze using its inherent instruction set. Next, the FPGA was partially reconfigured to bring on the FPU accelerator and the same operations were performed but with the assistance of the accelerator. At the end of the operations, the accelerator tile was *unprogrammed* and the time for the computation was recorded. Using this empirical approach, the partial reconfiguration time in addition to the communication time with the accelerator was accounted for. Finally, the L1 BLAS accelerator was brought online to perform the same operations. This allowed a comparison between the three configurations to be recorded in a single run. This was repeated for a sweep of vector sizes.

The computation time for the general purpose, MicroBlaze system is denoted as t_{GP} and the power usage is denoted by P_{GP}(taken from table II). The total energy used by the MicroBlaze system to complete N operations is then found by multiplying the power (W=J/s) by the computation time (s) to find the total number of Joules used. Equation 1 gives the calculation of MicroBlaze energy based on the power consumption and computation time.

$$E_{GP} = P_{GP} \cdot t_{GP} \qquad (1)$$

The energy used by the accelerated systems needs to consider additional procedures. First, the movement of data from the host processor to the accelerator after instantiated is included in the computation time measurement (t_{ACC}). Second, the energy required for partial reconfiguration is simply the power used (P_{PR}) multiplied by the reconfiguration time (t_{PR}). This quantity is a constant and independent of N. Equation 2 gives the calculation of the accelerated energy usage.

$$E_{ACC} = P_{ACC} \cdot t_{ACC} + P_{PR} \cdot t_{PR} \qquad (2)$$

Figure 6 shows the energy usage comparison of the three systems as the number of operations is swept. This plot illustrates that for a small number of operations, the MicroBlaze system has better energy efficiency than the accelerated configurations. This is because for a small number of operations, the higher power consumed by the accelerators, plus the additional power used during partial reconfiguration and data movement, dominates their overall energy usage.This plot shows an inflection point around 90,000 operations where the accelerated systems become more energy efficient than the MicroBlaze system. This occurs when the reduced time of the accelerated computation outweighs the cost of using a higher power computational element even after including PR power.

The next analysis that was performed was calculating the *speedup* that was achieved by the accelerators. The speedup is the ratio of the baseline system's computation time ($t_{comp-old}$) over the accelerated system's computation time ($t_{comp-new}$). A speedup less than 1 indicates a loss of performance and a speedup ratio greater than 1 is an improvement in system performance. The baseline system's computation time is t_{GP}. The accelerated computation time is the actual computation and data movement time (t_{ACC}) plus the PR time to bring on the accelerator (t_{PR}). Equation 3 gives the calculation of speedup.

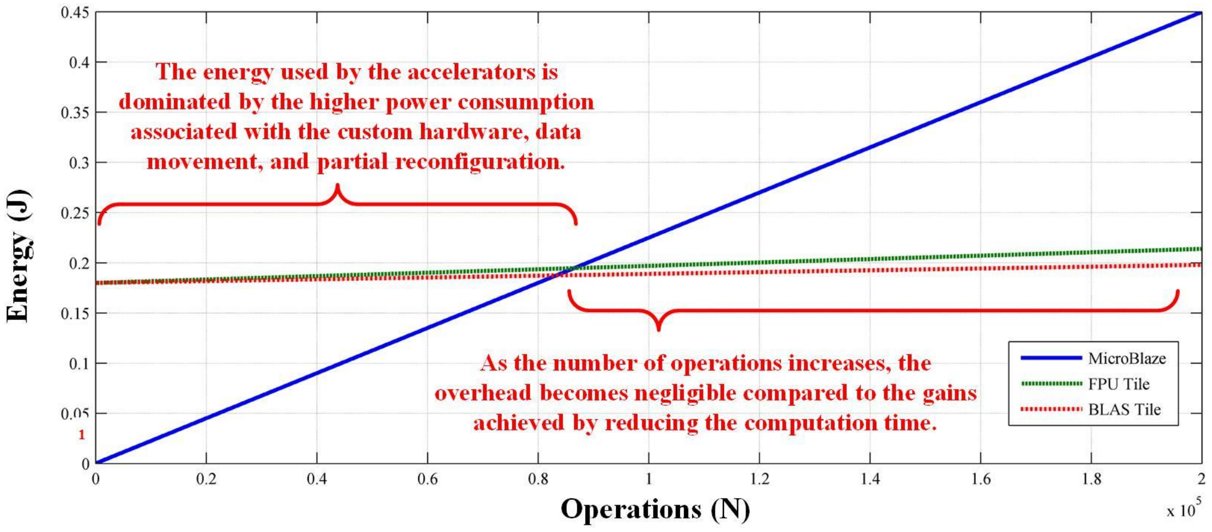

Figure 6. Energy Usage vs. the Number of Operations between the MicroBlaze baseline and Two Accelerator Configurations.

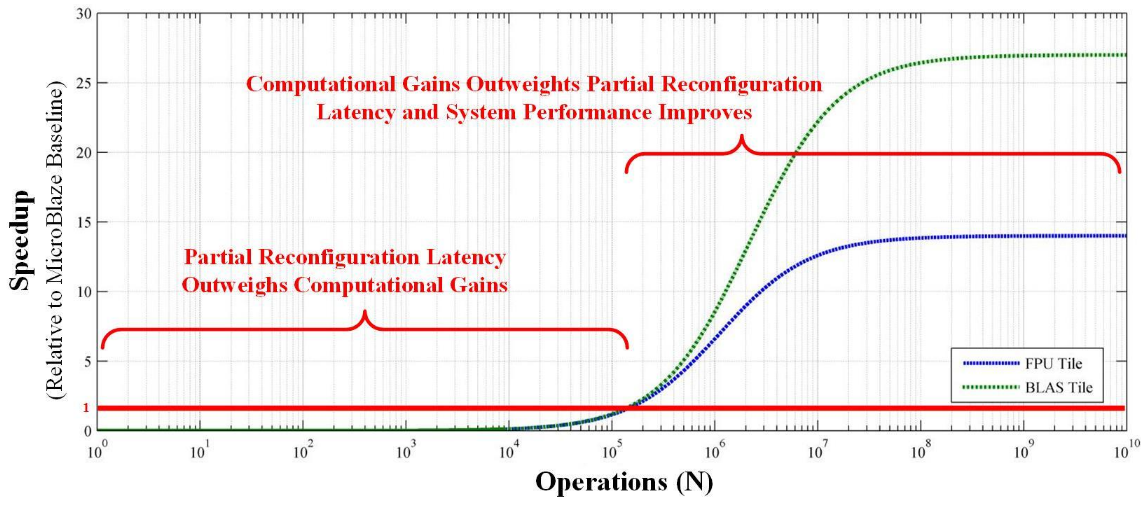

Figure 7. Speedup vs. the Number of Operations between the MicroBlaze baseline and Two Accelerator Configurations.

$$Speedup = \frac{t_{comp\ -old}}{t_{comp\ -new}} = \frac{t_{GP}}{t_{ACC}\ + t_{PR}} \quad (3)$$

Figure 7 shows the speedup achieved by the accelerators versus the total number of operations.This plot clearly shows the inflection point between where the accelerator performance is dominated by data movement and the partial reconfiguration time and where the increased computational ability of the accelerators becomes dominant. This inflection point occurs around 100,000 operations. This plot also shows how the BLAS accelerator achieves a higher speedup compared to a simple scalar accelerator (i.e., the FPU). This improved performance is due to the optimal hardware configuration of the BLAS tile when performing the vector operations. This improved performance also only occurs for large enough vectors in which the improvement outweighs the reconfiguration latency.

IV. Future Work

One of the pressing issues in modern computation is the latency of data movement between memory and the actual processing hardware. In the study presented in this paper, all data was transferred to memory within a single FPGA device. This set an upper bound on the size of the maximum dataset that could be evaluated in addition to only providing latency numbers for on-chip data movement. In order to more fully understand the potential impact that real-time, reconfigurable, BLAS accelerators can have on scientific computing, experiments must be conducted on datasets that span large numbers of FPGA-based accelerator cards. This will provide more insight into how chip-to-chip and card-to-card data movement latency impacts the computation. It is also important that this study be conducted empirically since theoretical calculations often don't consider all of the implementation details that impact performance.

V. Conclusion

This paper presented the motivation for creating BLAS hardware accelerators as real-time, reconfigurable tiles on Field Programmable Gate Arrays. By dynamically bringing on custom BLAS accelerators, the optimal hardware configuration can be obtained for the computation and the massive parallelism of FPGA hardware can be exploited. We presented a prototype system that implemented BLAS accelerators as on-chip, partially reconfigurable tiles. For this system, we measured and presented information on the reconfiguration time and power consumption, power efficiency, and speedup compared to a traditional, general-purpose computation. Our empirical data showed that the power consumption of the partial reconfiguration was considerable and impacted the inflection point for the power efficiency comparison. Our experiment also demonstrated that BLAS accelerator approach achieved a significant speedup compared to a general-purpose system, even when augmented with a hardware floating-point-unit.

Acknowledgments

The authors wish to thank NASA for supporting this research. This work was supported in part by grants (NNX10AN32A, NNX10AN91A, NNX12AM50G, NNX13AR03A, and NNX14AL03A) from NASA.

References

[1] "eXtreme Scale Computing Initiative," Pacific Northwest National Laboratory, [Online]. Available: http://xsci.pnnl.gov/pdf/XSCI_brochure.pdf
[2] B. Bishop, "Lawrence Livermore researchers awarded a billion supercomputer core hours," Lawrence Livermore National Laboratory Press Release, Nov. 2013, [Online], Available: https://www.llnl.gov/news/newsreleases/2013/Nov/NR-13-11-05.html#.UsGC0PRDtyS.
[3] "TeraGrid-to-XSEDE Transition," Extreme Science and Engineering Discovery Environment (ESEDE), Nov. 2011, [Online], Available: https://www.xsede.org/documents/10157/169907/TG-XSEDE-transition.pdf.
[4] "Getting Up To Speed, The Future of Supercomputing," Graham, S.L. Snir, M., Patterson, C.A., (eds), National Research Council of the National Academies, National Academies Press, 2004, ISBN 0-309-09502-6

[5] "Cyberinfrastructure Framework for the 21st Century - Science and Engineering (CIF21)," FY 2012 NSF Budget Request to Congress, National Science Foundation, [Online], Available: http://www.nsf.gov/about/budget/fy2012/pdf/40_fy2012.pdf.

[6] M. Gokhale, "Extreme Scale Challenges - Can Reconfigurable Computing come to the rescue?," in Proc. ReConFig 2013, International Conf. on, Dec 9-11, 2013.

[7] G. Bell, J. Gray, A. Szalay, "Petascale Computational Systems," Computer, pp. 110-112, Jan. 2006.

[8] W. Gropp, D. Kaushik, D. Keyes, and B. Smith, "Toward realistic performance bounds for implicit CFD codes," Proceedings of Parallel CFD'99. Elsevier, 1999.

[9] V. Eijkhout, "Introduction to High Performance Scientific Computing," Creative Commons Attribution 3.0, USA, ISBN: 9781257992546, June 2013.

[10] L.D. Foskick, E.R. Jessup, C. Schauble, G.Domik, "Introduction to High Performance Scientific Computing: 1st Edition," MIT Press, ISBN: 9780262061810, April 1996.

[11] J. W. Demmel, Applied Numerical Linear Algebra. Philadelphia, PA: SIAM, 1997.

[12] R. Hanson, F. Krogh, and C. Lawson, "A proposal for standard linear algebra subprograms," ACMSIGNUM Newslett., vol. 8, no. 16, p. 16, 1973.

[13] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic linear algebra subprograms for Fortran usage," ACMTrans. Math. Softw., vol. 5, no. 3, pp. 308–323, 1979.

[14] J. Dongarra, J. D. Croz, S. Hammarling, and R. Hanson, "Algorithm 656: An extended set of basic linear algebra subprograms: Model implementation and test programs," ACM Trans. Math. Softw., vol. 14, no. 1, pp. 18–32, 1988.

[15] J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling, "A set of level 3 basic linear algebra subprograms," ACM Trans. Math. Software., vol. 16, no. 1, pp. 1–17, 1990.

[16] J. Demmel, et. al., "Self-Adapting Linear Algebra Algorithms and Software," Proceedings of the IEEE , vol.93, no.2, pp.293,312, Feb. 2005.

[17] J.J. Dongarra, J.R. Bunch, C.B. Moler, G.W. Stewart, "LINPACK Users' Guide," SIAM Philadelphia, 1979.

[18] B. Smith, J. Boyle, J. Dongarra, B. Garbow, Y. Ikebe, V. Klema, C. Moler, "Matrix Eigensystem Routines, EISPACK Guide," Lecture Notes in Computer Science, Volume 6, Springer Verlag, 1976.

[19] E. Anderson, et. al., "LAPACK User's Guide," 3rd Edition, Aug. 1999.

[20] D.W. Walker, "An MPI version of the BLACS," Scalable Parallel Libraries Conference, Proceedings of the, pp.129-146, Oct 12-14, 1994.

[21] S.L. Blackford, et al, "ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance," Supercomputing, 1996.

[22] "Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard," [Online]. Available: http://www.netlib.org/blas/blast-forum/, January 25, 2001.

[23] S. Blackford, et. al., "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," ACM Transactions on Mathematical Software, Vol. 28, No. 2, June 2002.

[24] J. Choi, J.J. Dongarra, S. Ostrouchov, A. Petitet, D.W. Walker and R.C. Whaley, "A Proposal for a Set of Parallel Basic Linear Algebra Subprograms," Technical Report CS-95-292, Department of Computer Science, University of Tennessee, Knoxville, May 1995.

[25] K. Goto, et al, "High-performance implementation of the level-3 BLAS," ACM Trans. Math. Softw., vol. 35, no. 1, pp. 1–14, 2008.

[26] K. Yotov, L. Xiaoming, G. Ren, M.J. Garzaran, D. Padua, K. Pingali, P. Stodghill, "Is Search Really Necessary to Generate High-Performance BLAS?," Proceedings of the IEEE , vol.93, no.2, pp.358-386, Feb. 2005.

[27] M.I. Soliman, "Performance evaluation of multi-core intel xeon processors on basic linear algebra subprograms," Computer Engineering & Systems, 2008. ICCES 2008. International Conference on, pp.3-9, Nov. 25-27, 2008.

[28] Reddy, A. Lastovetsky, P. Alonso, "Heterogeneous PBLAS: Optimization of PBLAS for Heterogeneous Computational Clusters," Parallel and Distributed Computing, 2008. ISPDC '08. International Symposium on pp.73-80, July 1-5, 2008.

[29] J.G. Siek, I. Karlin, E.R. Jessup, "Build to order linear algebra kernels," Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, pp.1,8, April 14-18, 2008.

[30] G. Belter, E.R. Jessup, I. Karlin, J.G. Siek, "Automating the generation of composed linear algebra kernels," High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on, pp.1,12, Nov. 14-20, 2009.

[31] E.R. Jessup, S.C.Bernstein, B. Norris, J. Hossain, "Lighthouse: A User-Centered Web Interface to Matrix Algebra Software," [Online]. Available: http://lighthouse-taxonomy.googlecode.com/files/lighthouse.pdf

[32] A.X. Duchateau,D. Padua, D. Barthou, "Hydra: Automatic algorithm exploration from linear algebra equations," Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on, pp.1-10, Feb. 23-27, 2013.

[33] A.S. Szalay, et al, "GrayWulf: Scalable Clustered Architecture for Data Intensive Computing," Microsoft Research, Microsoft Corporation, Tech. Rep. MSR-TR-2008-187, Sept. 2008, [Online]. Available: http://research.microsoft.com/pubs/79429/GrayWulf_Hardware_FINAL.doc

[34] A. Pedram, S.Z. Gilani, S.K. Nam, R. van de Geijn, M. Schulte, A. Gerstlauer, "A Linear Algebra Core Design for Efficient Level-3 BLAS," Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on, pp.149-152, July 9-11, 2012.

[35] Lawrence Latif, " Intel reveals Xeon Phi architecture details at Hotchips", The Inquirer, [Available], Online: www.theinquirer.net, August 31, 2012.

[36] "What is GPU Accelerated Computing", NVIDIA Corp., [Available], Online: http://www.nvidia.com/object/What-is-GPU-Computing.html.

[37] V. Volkov and J. Demmel, "Benchmarking GPUs to tune dense linear algebra," SC 2008, 2008.

[38] P. Hursky, M.B. Porter, "Accelerating underwater acoustic propagation modeling using general purpose graphic processing units," OCEANS 2011, pp.1-6, Sept. 19-22, 2011.

[39] J. Muramatsu, S. Zhang, Y. Yamamoto, "Acceleration of Hessenberg Reduction for Nonsymmetric Eigenvalue Problems Using GPU," Networking and Computing (ICNC), 2010 First International Conference on, pp.215-219, Nov. 17-19, 2010.

[40] V. Allada, T. Benjegerdes, B. Bode, "Performance analysis of memory transfers and GEMM subroutines on NVIDIA Tesla GPU cluster," Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, pp.1-9, Aug. 31 2009.

[41] M. Nakata, Y. Takao, S. Noda, R. Himeno, "A Fast Implementation of Matrix-matrix Product in Double-double Precision on NVIDIA C2050 and Application to Semidefinite Programming," Networking and Computing (ICNC), 2012 Third International Conference on, pp.68-75, Dec. 5-7, 2012

[42] D. Mukunoki, D. Takahashi, "Implementation and Evaluation of Triple Precision BLAS Subroutines on GPUs," Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pp.1378-1386, May 21-25, 2012.

[43] D. Strenski, J. Simkins, R. Walke, R. Wittig, “Revaluation FPGAs for 64-bit Floating-Point Calculations," HPC Wire, May 14, 2008.
[44] R. Sundararajan, “High Performance Computing Using FPGAs,” Xilinx Corporation, White Paper, No. WP375, Sept. 10, 2010.
[45] N. Mehta, “Xilinx UltraScale Architecture for High-Performance Smarter Systems”, Xilinx Corporation, White Paper, No. WP434, Dec. 10, 2013.
[46] K. K. Matam, L. Hoang Le; V.K. Prasanna, "Evaluating energy efficiency of floating point matrix multiplication on FPGAs," High Performance Extreme Computing Conference (HPEC), 2013 IEEE, pp.1,6, Sept. 10-12, 2013.
[47] J. Fowers, G. Brown, P. Cooke, and G. Stitt, “A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA ’12. New York, NY, USA: ACM, 2012, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145704
[48] O. Storaasli, R.C. Singleterry, and S. Brown, "Scientific Computations on a NASA Reconfigurable Hypercomputer," Proc. Fifth Ann. Int’l Conf. Military and Aerospace Programmable Logic Devices, Sept. 2002.
[49] K.D. Underwood and K.S. Hemmert, "Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance," Proc. 12th Ann. IEEE Symp. Field-Programmable Custom Computing Machines, Apr. 2004.
[50] M. Smith, J. Vetter, and X. Liang, "Accelerating Scientific Applications with the SRC-6 Reconfigurable Computer: Methodologies and Analysis," Proc. 19th IEEE Int’l Parallel and Distributed Processing Symp., Apr. 2005.
[51] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A Quantitative Analysis of the Speedup Factors of FPGAs over Processors," Proc. 12th ACM/SIGDA Int’l Symp. Field Programmable Gate Arrays, pp. 162-170, Feb. 2004.
[52] V. Aggarwal, A. George, and K. Slatton, "Reconfigurable Computing with Multiscale Data Fusion for Remote Sensing," Proc. 14th ACM/SIGDA Int’l Symp. Field Programmable Gate Arrays, p. 235, Feb. 2006.
[53] S. Bajracharya, C. Shu, K. Gaj, and T. El-Ghazawi, "Implementation of Elliptic Curve Cryptosystems over gfð2nÞ in Optimal Normal Basis on a Reconfigurable Computer," Proc. 12th ACM/SIGDA Int’l Symp. Field Programmable Gate Arrays, Feb. 2004.
[54] D.A. Buell and J.P. Davis, "Reconfigurable Computing Applied to Problems in Communications Security," Proc. Fifth Ann. Int’l Conf. Military and Aerospace Programmable Logic Devices, Sept. 2002.
[55] A. Koohi, N. Bagherzadeh, and C. Pan, "A Fast Parallel Reed-Solomon Decoder on a Reconfigurable Architecture," Proc. First IEEE/ACM/IFIP Int’l Conf. Hardware/Software Codesign and System Synthesis, Oct. 2003.
[56] L. Zhuo and V. Prasanna, “High-performance designs for linear algebra operations on reconfigurable hardware," IEEE Trans. on Computers, vol. 57, no. 8, 2008.
[57] T. El-Ghazawi, E. El-Araby, H. Miaoqing, K. Gaj, V. Kindratenko, D. Buell, D., "The Promise of High-Performance Reconfigurable Computing," Computer , vol.41, no.2, pp.69-76, Feb. 2008.
[58] Cray Inc., http://www.cray.com/, 2008.
[59] SRC Computers, Inc., http://www.srccomp.com/, 2008.
[60] Silicon Graphics, Inc., http://www.sgi.com/, 2008.
[61] S. Kestur, J. D. Davis, and O. Williams, “Blas comparison on fpga, cpu and gpu," in Proceedings of the 2010 IEEE Annual Symposium on VLSI, ser. ISVLSI ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 288–293. [Online]. Available: http://dx.doi.org/10.1109/ISVLSI.2010.84.
[62] E. S. Chungz, J. D. Davisz, and S. Kestury, “An fpga drop-in replacement for universal matrix-vector multiplication," in Workshop on the Intersections of Computer Architecture and Reconfigurable Logic. Microsoft Research Silicon Valley Dept. of Computer Science and Engineering, The Pennsylvania State University, 2012.
[63] Z. Jovanovic and V. Milutinovic, “Fpga accelerator for floating-point matrix multiplication," Computers Digital Techniques, IET, vol. 6, no. 4, pp. 249–256, 2012.
[64] W. Zhang, V. Betz, and J. Rose, “Portable and scalable fpga-based acceleration of a direct linear system solver," ACM Trans. Reconfigurable Technol. Syst., vol. 5, no. 1, Mar. 2012.
[65] A. Agne, et al. 2014. ReconOS: An Operating System Approach for Reconfigurable Computing," *IEEE Micro*. vol. 34. no. 1. pp. 60-71. Jan.-Feb. 2014. DOI: 10.1109/MM.2013.110.
[66] D. Andrews. 2014. Operating Systems Research for Reconfigurable Computing. *IEEE Micro*. vol. 34. no. 1. pp. 54-58. Jan.-Feb. 2014. DOI: 10.1109/MM.2014.1.