Flexible framework for commodity FPGA cluster computing

Conference Paper · January 2010

DOI: 10.1109/FPT.2009.5377649 · Source: IEEE Xplore

4 authors, including:

Marcin Lukowiak
Rochester Institute of Technology
60 PUBLICATIONS **204** CITATIONS
SEE PROFILE

Gregor von Laszewski
Indiana University Bloomington
246 PUBLICATIONS **8,556** CITATIONS
SEE PROFILE

Flexible Framework for Commodity FPGA Cluster Computing

Jeremy Espenshade [1], Marcin Lukowiak [1], Gregor von Laszewski [2]

[1] *Department of Computer Engineering, Rochester Institute of Technology*
[2] *Service Oriented Cyberinfrastructure Lab, Rochester Institute of Technology*
Rochester, NY USA
[1] {jke2553, mxleec}@rit.edu, [2] laszewski@gmail.com

***Abstract*—With the continued fall of costs and successful demonstrations of performance, FPGAs have become a prime candidate for use in high performance computing. However, the use of FPGA technology in clustered environments has largely been limited to commercial and/or proprietary designs that require developers to learn new programming models and software tools. In this paper, a framework is presented which enables parallel software development across application-specific reconfigured hardware using simple hardware interface abstractions and standard MPI applications. Leveraging embedded Linux running on hardwired PowerPC processors, communication is managed such that each hardware element can function as a fully MPI-2 compatible node. Application case studies are presented and platform characteristics are elucidated through performance analysis.**

I. Introduction

In the domain of high performance computing, several architectural avenues are being explored in the search for maximum performance, optimal cost/benefit ratios, and flexibility of approaching computationally intensive tasks. Although most commercial super computers continue to be built with many homogeneous uniprocessors or chip-multiprocessors [1] [2], a recognizable trend has been toward inclusion of dedicated hardware to assist in computations that are especially intensive or for which a typical general-purpose processor is ill-suited. This acceleration often takes the form of hardware co-processors connected via PCI or some other direct interface using application-specific integrated circuits (ASICs), reconfigurable fabric (FPGAs), or, more recently, the class of streaming architectures including the STI Cell Broadband Engine and programmable graphics processing units (GPGPUs). Examples of each include the D.E. Shaw Research Anton molecular dynamics simulation supercomputer [3], the Cray XT5h [4] and SRC-7 reconfigurable supercomputers [5], and the IBM BlueGene/P hybrid supercomputer [6] and Nvidia Tesla based computing solutions [7] respectively. The motivation for all of these architectures is the set of applications that require acceleration and contain program segments that are not ideally suited to computation on a general purpose processor (GPP). Some examples of such applications are cryptanalysis, molecular dynamics simulations, bioinformatics, and high data-throughput image and video processing [8].

While positive results have been garnered in these and many other application areas using the co-processor model, a recent area of research interest has been placing increased responsibility with the dedicated hardware, especially focused on reconfigurable computing elements. High performance computing research efforts using reconfigurable elements exclusively have been undergone at several universities [9][10] and the Airforce Research Laboratory Rome [11]. These efforts have been largely successful at demonstrating the potential for reconfigurable computing in a massively parallel environment [8].

An avenue that has yet to be explored to any great extent is in the use of commodity off-the-shelf (COTS) FPGAs with multiple custom hardware units each acting as fully functioning and participating nodes in a heterogeneous computing cluster for HPC use. With the advent of high performance, large area FPGAs with cost similar to that of GPP cluster nodes, such an inclusion is both feasible and promising.

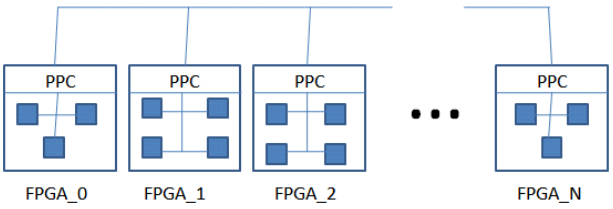

Fig. 1. Physical Cluster Structure

To enable FPGA inclusion in such a cluster, one must first consider the communication method used to coordinate processes and pass data between nodes. A commonly accepted standard API for communication on everything from traditional COTS clusters to the BlueGene/P supercomputer is the Message Passing Interface (MPI). The MPI standard supports a send/receive paradigm for interprocess communication with many additional features for collective communication, process organization, etc [12]. As such, retaining this common interface has the benefits of allowing reuse of existing code structure and function, building upon a well-established communication infrastructure, and, most importantly, presenting a

familiar and understood API for development targeted toward an FPGA cluster.

To realize the inclusion of FPGA hardware units as fully functioning cluster nodes, the following paper presents a framework for hardware and software development that allows multiple, specialized computation elements to exist on each FPGA and exploits a Linux operating system environment running on embedded PowerPC processors to abstract communication such that each computation element exists as a node supporting MPI communication and synchronization in a common manner. The rest of the paper is organized as follows: Section 2 will discuss related and supporting work. Section 3 will present the system description including hardware, software, and programming model considerations while walking through testbed applications. Section 4 will discuss the results collected. And section 5 will conclude the paper with discussion of future research and development directions.

II. Related Work

In general computation, the use of MPI communication has been investigated extensively and demonstrated to be a high quality method for organizing processes and communication across clusters ranging in size from small groups of workstations to supercomputers to grids using MPI as a layer underlying Condor and/or Globus system coordination. The current standard is MPI-2.1, an extension to the original MPI-1.X that adds one-sided communication, dynamic processes, and parallel I/O among other features to the original set of features supporting point-to-point and collective communication and synchronization. One of the most prevalent implementations of the MPI API is OpenMPI [13], a community driven open source implementation merging and building on the older FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI implementations.

On the hardware side, Field Programmable Gate Arrays (FPGAs) have been the target of vigorous research that attempts to make effective use of the potential for performance improvement in appropriate applications. Most a propos is the body of work targeting System-on-Chip (SoC) designs. SoC is generally defined as any disparate set of computing elements existing on a single chip, and this design strategy is common practice for FPGA usage in embedded systems. An extension on SoC is MPSoC, or multiple processor System-on-Chip, design where multiple processors are combined with other elements on a single chip and interface buses provide communication. The use of MPI for MPSoC integration has been discussed for several years [14] [15] but has been largely confined to single platforms.

Single-FPGA MPSoC research using Linux and MPI has been performed with good results on Xilinx FPGAs for image processing[16]. This work builds on an earlier MPI implementation using the Xilinx FSL (Fast-Simplex Link) bus to provide FIFO communication links between MicroBlaze processors [15]. A parallel image processing library is subsequently built on top of that MPI layer and scaling similar to multiple desktops is demonstrated. As mentioned, this work is limited to a single FPGA and implements only a small subset of the MPI API.

Probably the closest in terms of scope to the effort detailed in this paper is the TMD-MPI project work done at the University of Toronto [17]. TMD-MPI was developed as a programming model for the TMD multi-fpga computing machine built for molecular dynamics simulations. Like the previous work, a stripped down MPI implementation was designed that also used the FSL bus for on-chip communication [18]. This design was extended to make use of Xilinx Multi-Gigabit Transceivers that connected multiple FPGAs on a single PCB and multiple PCBs to form the full system. Besides the required custom construction of such a hardware configuration, the TMD-MPI implementation suffered from some limitations resulting primarily from the absence of an operating system. Only a small subset of the MPI API was implemented and without dynamic management, ranks had to be statically assigned, processes must be statically started, and only synchronous sends and receives were implemented [19]. Despite these issues, TMD-MPI serves as a valuable effort showing the validity of a MPSoC approach across multiple FPGAs.

Finally, there is a body of work considering integration of hardware acceleration units into a Linux Operating System environment running on embedded PowerPC processors [20][21][22]. These efforts demonstrate the validity of using a Linux device driver to control software/hardware communication. FIFO queues in particular were suggested as a preferred SW/HW communication abstraction by Williams et al [23].

III. Framework Description

The framework described throughout this work is intended to be portable across any Xilinx FPGAs that include an embedded PowerPC processor. For demonstration purposes a network of two Virtex-5 FX70T, two Virtex-5 FX130T, and one Virtex-4 FX60 evaluation platforms was used as the testbed. These platforms include Gigabit Ethernet controllers, 256-512 MB of DDR2 RAM, and 512MB compact flash cards. Each FPGA has a single PowerPC hardwired into the reconfigurable fabric. The Processor Local Bus (PLB) connects the 400 MHz PowerPC 440 (Virtex-5) or 300 MHz PowerPC 405 (Virtex-4) processors to all on-chip peripherals running at 100 MHz.

A. Software Environment

The core of the software environment is a Linux operating system built on the 2.6 kernel. Xilinx provides a standard configuration along with drivers for a minimal set of peripheral hardware. Subsequent configuration changes and driver development was done using the DENX Embedded System Development Kit cross-compilation environment. The root file system was built starting from a base Xilinx ramdisk and BusyBox utilities.

This effort builds upon OpenMPI, as it is constructed in a consistent layered model that provides for the possibility of

custom physical layer communication methods, builds in support for limited heterogeneity, and is implemented in standard C/C++ rather than using Python scripts to implement the MPI command line interface. Given the use of embedded Linux, each additional application that must be ported and/or configured within the root file system requires additional design effort and is preferably avoided. OpenMPI utilizes OpenSSH and in turn, OpenSSL, to provide secure communication. Both to save space and ensure maximum interoperability, all programs were built dynamically, requiring the manual identification and installation of shared libraries. SSH was also configured across the nodes for seamless certificate-based security. While initially time consuming, the same root file system can now be deployed to the compact flash drive of each FPGA or shared via NFS.

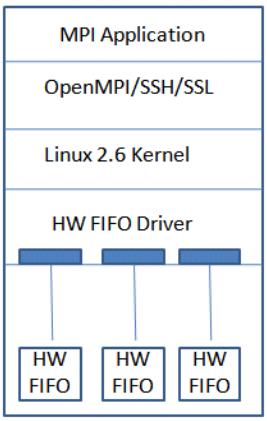

Fig. 2. Application Layers

B. Hardware Interface

The hardware acceleration units expose a common FIFO interface through the use of a read/write FIFO pair attached to the PLB. Application specific hardware may then be attached, using data arriving in the Write FIFO and reporting results to the Read FIFO. Optional interrupt generation hardware is also supported. The recommended hardware design process utilizes existing Xilinx tools within their Embedded Development Kit (EDK). The Base System Builder is used to generate an overall design including the PowerPC core, DDR interface, hardwired Ethernet MAC, and SystemACE compact flash components connected via the PLB. The Create/Import Peripheral wizard is then used to generate PLB interface wrappers and include the FIFO, interrupt generation, and supporting logic. Example interaction code is provided allowing developers to quickly and easily develop or integrate application-specific logic. Further tools allow intuitive connection of new hardware accelerators to the PLB and global interrupt controller and also assignment of unique physical addresses.

Interaction with this hardware interface is supported through a common device driver developed for the purpose. This driver registers Linux character drivers for each compatible hardware accelerator included in the hardware design. Physical addresses and IRQs are remapped into Linux-managed virtual space and file operations implementing *open*, *release*, *read*, and *write* behaviors are implemented. Through the use of special device files, standard C-language *fopen* operations are thereby mapped to the *open* behavior including hardware initialization and lock-based arbitration ensuring singular software process access. Similarly, *fclose*, *fwrite*, and *fread* operations can be used to invoke the defined *release*, *read*, and *write* behaviors.

Special device files are created and managed dynamically allowing for seamless hardware design flexibility. At boot time, the character device registry exposed in */proc/devices* is checked for user-created hardware accelerators identified by a "user-" prefix prepended to device names by the driver. Special device files with the original device name and appropriate major number are then created in the */dev* directory such that a possible configuration of two *addition* and three *multiplication* accelerators would appear as shown in Figure 3.

IV. Programming Model Example

To provide a simple hardware acceleration unit for initial framework verification and analysis, a matrix multiplication unit was implemented on the FPGA. This unit accepts 32-bit integer operands and expects to be initialized with the number of rows and columns. After initialization, the unit stores a set of rows from the first matrix and performs concurrent dot products with subsequently received columns of the second matrix. Once all columns are received, it waits on a new set of rows. This is explained in detail to provide the basis for a working example of the programming model in the following section.

As desired, the programming model will look very accessible to anyone familiar with MPI and standard C-language file operations. With FIFO queues acting as a buffering abstraction, little consideration needs to be paid to the hardware implementation by a software application developer. Using the description of the matrix multiplication hardware unit in the last section, a distributed algorithm can be easily written as shown in figure 4. With such a direct and familiar programming model, hardware/software co-design is effectively disentangled and development effort can be spent directly on software algorithms and hardware implementations.

In this case, a choice was made to increase access locality in the hardware by reusing a each row of Matrix A M_DIM times rather than receiving pairs of operands and performing individual multiply-accumulates. This decision and others like it will impact the specification for what data is being sent back and forth across the FIFOs, but this is similar to any software-only programming model with interface specifications. As a result, $MatB$ was formatted in column-major order in this case to allow contiguous blocks of memory to be written to the FIFO. A matrix multiplication engine performing 32 concurrent dot products was synthesized on each FPGA, and an alternative 64 dot product implementation was synthesized for the largest Virtex-5 (FX130T).

```
# ls -la /dev
...
crw------- 1 root root 249, 0 Jan 1 1970 addition0
crw------- 1 root root 250, 0 Jan 1 1970 addition1
...
crw------- 1 root root 251, 0 Jan 1 1970 multiplication0
crw------- 1 root root 252, 0 Jan 1 1970 multiplication1
crw------- 1 root root 253, 0 Jan 1 1970 multiplication2
...
```

Fig. 3. Hardware Accelerator Special Device Files in /dev Directory

```
Input:
  MatA, MatB, MatC: MatA * MatB = MatC
  HW_FIFO: Device name (ie /dev/hwfifo0)
  N_DIM, M_DIM: MatA is an NxM matrix
  NUM_ROWS: Number of Rows locally stored

  MPI_Init();
  MPI_Rank(&rank);
  MPI_Size(&size);
  // Send/Receive data
  ...
  file *HWFIFO = fopen{HW_FIFO, "r+"}
  setvbuf(HWFIFO, NULL,_IONBF, 4)
  for (i=rank to N_Dim+ = size * NUM_ROWS){
    fwrite(&(MatA[i * M_DIM]),
        4,M_DIM * NUM_ROWS, HWFIFO);
    for (j= 0  to M_Dim+ = 1){
      fwrite(&(MatB[j * M_DIM]),
        4,M_DIM, HWFIFO);
    }
    fread(&(MatC[i * N_DIM]),4,NUM_ROWS, HWFIFO);
  }
  // Send/Receive Results
  ...
  fclose(HWFIFO);
  MPI_Finalize();
```

Fig. 4. Matrix Multiplication Program

A. Matrix Multiplication Outcomes

While exceptional performance should not be the expectation when targetting FPGAs for dense linear algebra [24], application development and deployment did shed light on some important characteristics of the platform. Acting as a general purpose comparison, a cluster of 32 Intel Xeon 5140 2.33 GHz dual-core processors connected over Gigabit ethernet was also targeted with similar algorithmic organization. Table I shows the execution times for the three FPGA models with 32 multiply accumulators (MACs) performing dot products, the Virtex-5 FX130T with 64 MACs, and a single core Xeon performing multiplications across matricies of varying dimensions.

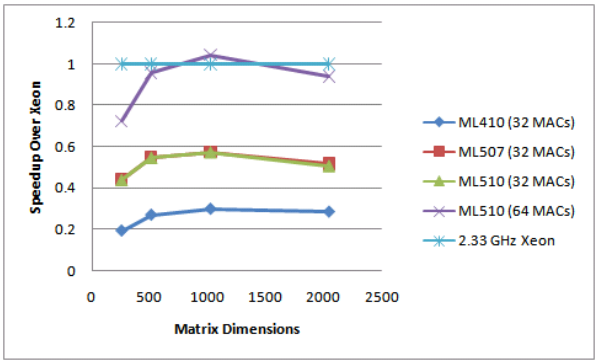

Fig. 5. Single Node Matrix Multiplication Speedup Over Xeon

Several interesting results are immediately apparent when analyzing these results along with the speedup graphically shown in Figure 5. First, while the two similarly configured Virtex-5 devices perform nearly identically, the same configuration on the Virtex-4 suffers a performance drop. The 64 MAC version of the FX130T also provides speedups ranging from 1.64x to 1.85x over the 32 MAC version. Most importantly, however, is the lack-luster performance of the FPGA solutions, most notibly the Virtex-4. Analysis of the hardware design and performance results shows inefficient use of arithmetic hardware resulting from a bandwidth bottleneck between the PowerPC and hardware accelerator. Table II shows the observed bandwidth peaking at around 23 MBps for the Virtex-4 and 42 MBps for the Virtex-5 devices. This motivates the need for DMA data transfer to hardware accelerators which is one of the top priorities for future development of this framework.

Scaling here?? It would be good if I have room. I have room! Will add ASAP

V. DES Cryptanalysis

A. Introduction and Design

While the bandwidth characteristics observed in the previous section limit the range of applications that would experience speedup when deployed with this framework, many applications exhibit a sufficiently low communication to computation ratio such that the computational capabilities of FPGAs are appropriately exploited. One such application area is cryptanalysis. Ageing encryption standards such

Matrix Dimensions	256 x 256 (s)	512 x 512 (s)	1024 x 1024 (s)	2048 x 2048 (s)
Virtex-4 FX60 (32 MACs)	0.162	0.957	6.562	47.139
Virtex-5 FX70T (32 MACs)	0.071	0.470	3.427	26.015
Virtex-5 FX130T (32 MACs)	0.071	0.470	3.410	26.403
Virtex-5 FX130T (64 MACs)	0.043	0.269	1.875	14.269
2.33 GHz Xeon	0.031	0.258	1.956	13.405

TABLE I
SINGLE NODE MATRIX MULTIPLICATION EXECUTION TIMES

Matrix Dimensions	256 x 256 (MBps)	512 x 512 (MBps)	1024 x 1024 (MBps)	2048 x 2048 (MBps)
Virtex-4 FX60 (32 MACs)	16.150	19.732	21.731	23.490
Virtex-5 FX70T (32 MACs)	37.031	40.183	41.610	42.563
Virtex-5 FX130T (32 MACs)	36.924	40.078	41.815	41.938
Virtex-5 FX130T (64 MACs)	36.492	38.990	40.272	39.976

TABLE II
OBSERVED POWERPC TO HARDWARE ACCELERATOR BANDWIDTH

as the Data Encryption Standard (DES) often were created with insufficient key lengths that have since been subject of successful brute force attacks. Block ciphers like DES are generally well-suited to FPGA implementation because of the bit-level operations and explicitly staged operation that allows the included bitwise logic and highly pipelined architecture to be fully engaged.

As a practical example of such an attack, DES has previously been cracked using a known-plaintext attack where a block of unencrypted data (plaintext) and the cooresponding encrypted ciphertext is known a priori. DES uses a secret key to encrypt blocks of 64-bits at a time and decryption is trivial if the key used during encryption is known. Therefore, the plaintext can be encrypted with each possible secret key and once the result matches the known ciphertext, the secret key will have been found and the encryption is broken. DES keys have a cryptographic strength of 56 bits, so 2^56 keys must be checked in the worst case to break the encryption.

From a parallel software development standpoint, a DES cracking application is concerned with the coordination of available processors such that each processing element stays busy until the key is found or a chosen subspace of possible keys is completely searched. The later approach is particularly helpful during development and performance testing, as searching the entire key space remains prohibitive regardless of platform. To partition the key space into work units that can be distributed to processing elements, the method used by the COPACOBANA [25] team in their custom code-cracking FPGA platform was applied such that the most significant 24 bits are used to indicate a group of 2^32 possible keys that must be searched. A master-slave paradigm was then applied with the master maintaining a dynamic queue of remaining work to be distributed to the slaves.

From a hardware development perspective, internal counters can be used to generate the least significant 32 bits of each key in the search space designated by the 24 most significant bits. After initialization with the known plaintext and ciphertext, 24-bit key space indicators are received and either the correct key or key-not-found condition is returned upon exhaustive search completion. FPGA hardware allows DES to be pipelined easily into sixteen stages, one for each identical DES round, and with an additional input and output registration stage, an 18-stage pipeline results. Furthermore, multiple concurrent encryptions are possible with duplicate encryption engines while reusing some peripheral hardware including the PLB interface and interaction logic.

To make use of the available hardware parallelism, the 32-bit search space can be further partitioned with each encryption engine retaining a unique identifier such that the actual search space is reduced to 32 minus $\log_2$(Number of Encryption Engines). For example, with four encryption engines 2-bit identifiers of “00”, “01”, “10”, and “11” would prepend the lower 30 bits of the remaining search space. Eight encryption engines were synthesized in the final accelerator design. This resulted in very dense usage of the available fabric in each FPGA type with one DES accelerator on the Virtex-4, two on the Virtex-5 FX70T, and three on the FX130T. This configuration implies eleven total accelerators across the five FPGAs, each guessing eight keys concurrently at 100 MHz, which ideally results in 8.8 billion keys guessed per second.

To act as a basis for comparison, the same cluster of Xeon GPPs was targeted for the development of a similar software-only implementation using the GNU Crypt library. The only change in structure was an increase in the key space indicator from 24 bits to 32 bits, as a 32-bit key space search required prohibitively lengthy computation in software. To combat the inherent increase in communication that results from this change, all timing comparisons are between application configurations with identical numbers of interprocess communication instructions rather than total key space searched.

B. Results and Analysis

The vast majority of the computation time involved in cracking DES is searching key spaces that do not contain the correct key. Performance analysis was therefore performed across subspaces known not to contain the correct key so as to

ensure the entire space was searched. The target search space was scaled linearly with the addition of processing nodes such that each node would check sixteen key spaces on average. Figure 6 shows how the application scales across the eleven processing nodes available across the five FPGAs.

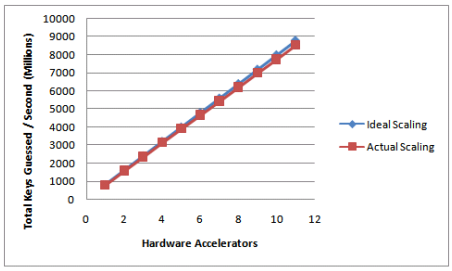

Fig. 6. DES application scalability with reference to ideal hardware speed

The listed data and efficiency included in Table III demonstrate remarkable application and framework scalability. Individual node hardware access from user software exceeds 99% of the ideal hardware performance, while deploying the application across distributed resources retains efficiency in excess of 97%. This result implies that additional FPGA resources would exhibit similar performance scaling and while efficiency would be expected to degrade slowly, a small number of additional FPGAs would quickly reduce the total time required to search the entire key space and break DES. The current configuration guessing 8.548 billion keys per second would require an average of approximately 48.78 days of continuous computation to break DES encryption, but hardware design improvements and additional resources could quickly reduce that to a very reasonable period of time.

# of Hardware Accelerators	Execution Time (seconds)	Key Throughput (keys/second)	Efficiency
1	86.835	791.38 M	0.989
2	87.077	1578.35 M	0.986
3	87.461	2357.16 M	0.982
4	87.366	3146.28 M	0.983
5	87.777	3914.45 M	0.979
6	88.334	4667.70 M	0.972
7	88.451	5438.46 M	0.971
8	88.454	6215.16 M	0.971
9	88.303	7004.00 M	0.973
10	88.490	7765.75 M	0.971
11	88.426	8548.55 M	0.971

TABLE III
DES APPLICATION PERFORMANCE ACROSS FPGA CLUSTER

Placed in contrast with the stellar performance observed on the FPGA cluster, the Intel Xeon GPPs executing software-only key searches performed poorly. Application scalability followed similar trends with 95% of single-node efficiency observed across eleven independent processors resulting in three orders of magnitude speedup exhibited by the FPGA platform. To provide further framing, reference is made to a previous study [8] of DES cryptanalysis applied to commercial FPGA supercomputers by Cray and SRC. The cluster of commodity FPGAs organized with the developed framework fairs quite well against these solutions. Furthermore, while a full cost/performance analysis is impossible given private pricing policy, Cray's XD1 debuted in 2005 at nearly $100,000 U.S. Dollars [26], an order of magnitude more than the commodity FPGA cluster.

Platform	Key Throughput (keys/second)	Speedup Over Software Solution
Commodity FPGAs	8,548 Million	1107 x
SRC-6	4,000 Million	518 x
Cray XD1	7,200 Million	930 x
2.33 GHz Xeon (11 cores)	7.722 Million	1 x

TABLE IV
FPGA SUPERCOMPUTER DES PERFORMANCE COMPARISON

VI. CONCLUSION

In this paper, a flexible framework for scalable commodity FPGA cluster computing was presented and demonstrated. A direct matrix multiplication application was deployed across multiple FPGAs demonstrating functionality while platform characteristics were discovered through subsequent performance analysis. A DES cryptanalysis application was then presented, demonstrating speedup in excess of 1100x over a cluster of general purpose processors with a cost/performance improvement of 371x. This work serves as a valuable contribution to the literature building off the contemporary body of research and provides the foundation for future work with significant research potential in the field of high-performance distributed computing leveraging FPGA technology.

A. Future Work

This paper marks the first report of an ongoing effort to develop a flexible framework for commodity FPGA cluster computing with considerations for software programmability, ease of hardware implementation, and hardware/software abstraction. As such, there are myriad approaches to be considered and optimizations to be attempted. The following outlines the expected continuation flow building on this work but is less than comprehensive.

Given the limited speed of I/O operations on the PLB, investigation into supplementary communication abstractions is warranted. Communicating control signals and DMA addresses via the FIFO interface working in tandem with a DMA interface between hardware accelerators and main memory could provide enough bandwidth to expand application suitability to higher communication to computation ratios. More suitable applications will be deployed in future work to better examine the potential for productive HPC use regardless, however bandwidth improvements are a high priority. Likely areas of investigation are in the fields of astrophysics and neural simulations.

Further framework improvements increasing the cluster robustness with fault tolerance and actively managed job queues are expected. And investigation into dynamic reconfiguration,

integrated design tool flows, and improved developer experience round out the most likely avenues of exploration and development.

REFERENCES

[1] T. Domany and et al, "An Overview of the BlueGene/L Supercomputer," in *SC '02: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Baltimore, MD, USA, 2002.

[2] "Exploring Science Frontiers at Petascale," Oak Ridge National Laboratory, Tech. Rep., 2008.

[3] D. E. Shaw and et al, "Anton, a special-purpose machine for molecular dynamics simulation," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*. New York, NY, USA: ACM Press, 2007, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1145/1250662.1250664

[4] "Cray XT5h Supercomputer," Cray Inc., Tech. Rep., 2008.

[5] S. Computers. (2009) SRC-7 Overview. SRC Computers. Http://srccomputers.com/products/src7.asp. [Online]. Available: http://srccomputers.com/products/src7.asp

[6] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the petaflop era: the architecture and performance of Roadrunner." in *SC*. IEEE/ACM, 2008, p. 1. [Online]. Available: http://dblp.uni-trier.de/db/conf/sc/sc2008.html#BarkerDHKLPS08

[7] N. Corp. (2009) Tesla S1070. Nvidia Corp. Http://www.nvidia.com/object/product_tesla_s1070_ us.html. [Online]. Available: http://www.nvidia.com/object/product_tesla_s1070_us.html

[8] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The Promise of High-Performance Reconfigurable Computing," *IEEE Computer Magazine*, vol. 41, no. 2, pp. 69–76, 2008.

[9] C. Patterson, B. Martin, S. Ellingson, J. Simonetti, and S. Cutchin, "FPGA Cluster Computing in the ETA Radio Telescope," *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, pp. 25–32, Dec. 2007.

[10] R. Sass, W. Kritikos, A. Schmidt, S. Beeravolu, and P. Beeraka, "Reconfigurable Computing Cluster (RCC) Project: Investigating the Feasibility of FPGA-Based Petascale Computing," *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pp. 127–140, April 2007.

[11] G. D. Peterson, "Programming High Performance Reconfigurable Computers," Airforce Research Lab Rome Research Site, Rome, NY, Final Technical Report AFRL-IF-RS-TR-2003-2, January 2003.

[12] *MPI: A Message Passing Interface Standard*, Message Passing Interface Forum Std. 2.1, June 2008.

[13] R. L. Graham, T. S. Woodall, and J. M. Squyres, "OpenMPI: A Flexible High Performance MPI," in *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, Poznan, Poland, September 2005.

[14] M. Youssef, S. Yoo, A. Sasongko, Y. Paviot, and A. Jerraya, "Debugging HW/SW interface for MPSoC: video encoder system design case study," *Design Automation Conference, 2004. Proceedings. 41st*, pp. 908–913, 2004.

[15] J. A. Williams, I. Syed, J. Wu, and N. W. Bergmann, "A Reconfigurable Cluster-on-Chip Architecture with MPI Communication Layer," in *FCCM '06: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 350–352.

[16] I. Syed, J. Williams, and N. Bergmann, "A Hybrid Reconfigurable Cluster-on-Chip Architecture with Message Passing Interface for Image Processing Applications," *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pp. 609–612, Aug. 2007.

[17] A. Patel, C. Madill, M. Saldana, C. Comis, R. Pomes, and P. Chow, "A Scalable FPGA-based Multiprocessor," *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, pp. 111–120, April 2006.

[18] M. Saldana and P. Chow, "TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs," *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pp. 1–6, Aug. 2006.

[19] M. Saldana, "A Parallel Programming Model for a Multi-FPGA Multiprocessor Machine," Master's thesis, University of Toronto, 2006.

[20] T. Mehlan, J. Strunk, T. Hoefler, F. Mietke, and W. Rehm, "IRS - A Portable Interface for Reconfigurable Systems," in *PARELEC '06: Proceedings of the international symposium on Parallel Computing in Electrical Engineering*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 187–191.

[21] M. Liu, W. Kuehn, Z. Lu, A. Jantsch, S. Yang, T. Perez, and Z. Liu, "Hardware/Software Co-design of a General-Purpose Computation Platform in Particle Physics," *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, pp. 177–183, Dec. 2007.

[22] S. D. Breijer, "Memory Organization of the Molen Prototype," Master's thesis, Delft University of Technology, 2007. [Online]. Available: http://ce.et.tudelft.nl/publicationfiles/1367_700_thesis.pdf

[23] J. A. Williams, N. W. Bergmann, and X. Xie, "FIFO Communication Models in Operating Systems for Reconfigurable Computing," in *FCCM '05: Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 277–278.

[24] J. P. Morrison, P. O'Dowd, and P. D. Healy, *High Performance Computing: Paradigm and Infrastructure*. John Wiley and Sons, 2006, ch. Chapter 14 - An Investigation of the Applicability of Distributed FPGAs to High-Performance Computing.

[25] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, A. Rupp, and M. Schimmler, "How to break des for 8,980 euro," in *SHARCS'06 - Special-purpose Hardware Attacking Cryptographic Systems*, Cologne, Germany, April 2006.

[26] (2005, Feb 15) Cray XD1 Supercomputer Outscores Competition in HPC Challenge Benchmark Tests. Business Wire. http://investors.cray.com/phoenix.zhtml?c=98390&p=irol-newsArticle&ID=674199&highlight=.