2010 International Conference on Field Programmable Logic and Applications

GPU versus FPGA for high productivity computing

David H. Jones, Adam Powell, Christos-Savvas Bouganis, Peter Y. K. Cheung
Imperial College London,
Electrical and Electronic Engineering, London

***Abstract*—Heterogeneous or co-processor architectures are becoming an important component of high productivity computing systems (HPCS). In this work the performance of a GPU based HPCS is compared with the performance of a commercially available FPGA based HPC. Contrary to previous approaches that focussed on specific examples, a broader analysis is performed by considering processes at an architectural level. A set of benchmarks is employed that use different process architectures in order to exploit the benefits of each technology. These include the asynchronous pipelines common to "map" tasks, a partially synchronous tree common to "reduce" tasks and a fully synchronous, fully connected mesh. We show that the GPU is more productive than the FPGA architecture for most of the benchmarks and conclude that FPGA-based HPCS is being marginalised by GPUs.**

***Keywords*-High Productivity Computing, FPGA, GPU**

I. Introduction

High-performance computing (HPC) is the use of parallel processing for the fast execution of advanced application programs. In 2004, DARPA replaced "performance" with "productivity" and formed the high-productivity computing system (HPCS) [1]. Instead of comparing solutions with their different execution times, they proposed using "time to solution", a measure that includes the time to develop the solution as well as the time to execute it. Factors important to HPCS include how scalable, portable and reusable the solution is. Another important factor is at what programming level effective solutions can be developed.

FPGAs have been shown to effectively accelerate certain types of computation useful for research and modelling [2], [3], [4]. However their utilization has been restricted by the development time and complexity for fine-grained implementations. A study of the HPCS jobs performed by the San Diego Supercomputing Centre between 2003 and 2006 showed that the longest job required 24 days to execute but the average job length was three hours [5]. Thus developing custom firmware or hardware will severely limit the productivity of most HPCS tasks.

Another restriction on the effective use of FPGAs as HPCS is that there is little open-source, portable firmware for FPGA HPCS. This is due to the fact that no standardised protocols, interfaces or coarse-grain architectures exist. The OpenFPGA [6] standard seeks to correct this, however their API is still in draft form (v0.4 since 2008) and not all FPGA-based HPCS suppliers are members of the alliance.

In 2007 Nvidia made the Compute Unified Device Architecture (CUDA^{TM}) available for the development of graphics processing units (GPU) based HPCS platforms. CUDA made it simple to develop distributed code for a pervasive and standardised coarse-grain platform. The question this paper addresses is whether GPUs are now an effective alternative to FPGA based HPCS systems.

To do this we compare the productivity of two commercially available HPCS platforms from the point of view of a HPCS developer: an Nvidia GPU and a multiple-FPGA supercomputer developed by Convey Computer and released in 2009 [7]. This is the first published evaluation of this HPCS platform.

The objective is to compare the productivity of each platform for a broad range of tasks. This is addressed by selecting benchmarks with different process architectures classified by their algorithmic skeletons [8]. Algorithmic skeletons are tools for the classification of distributed programs according to their synchronisation and communication requirements. Goodeve [9] suggested two skeletons for classifying interdependent threads:

- **Tree computations** are performed using the application of recursive functions that form the nodes of a branch and leaf process architecture. Examples include "farm" and "reduce" operations.
- **Crowd computations** are performed with a set of independent but co-operating processes that communicate and synchronise with each other. A typical crowd computation follows some regular graph such as a ring or mesh.

We add to this list independent threads, each of which forms an asynchronous pipeline for performing some calculation.

In order to best compare the productivity of each platform we consider algorithms from each category. Another consideration is whether our benchmarks are necessarily an optimum implementation for each platform. Thus where possible we use library functions or custom firmware that have been optimised by Nvidia and Convey. The benchmarks we have chosen that satisfy these criteria are:

1) Batch generation of pseudo-random numbers.
2) Dense square matrix multiplication.
3) Sum of large vectors of random numbers.
4) Second order N-body simulation.

Table I compares their different process architectures and

978-0-7695-4179-2/10 $26.00 © 2010 IEEE
DOI 10.1109/FPL.2010.32

	Benchmarks			
Process Architecture	**1**	**2**	**3**	**4**
Asynchronous Pipeline	X	X		
Tree computation			X	
Crowd computation				X
GPU Process implementation				
Optimised software	X	X	X	
FPGA Process implementation				
Optimised software		X	X	
Application-specific firmware	X			

Table I
BENCHMARKS AND THE CORRESPONDING PROCESS ARCHITECTURES

System	Form	(#) Technology	Performance
Nallertech BenNuey	PCIe Card	(1) Virtex-II	12 GFlop/s [10]
SRC-7 MAPstation	2U Rack	(6) Stratix-II	24 GFlop/s [11]
Cray XD1	2U Rack	(6) Virtex-II	58 GFlop/s [12]
Convey HC-1	2U Rack	(4) Virtex-5	65 GFlop/s [13]

Table II
COMMERCIALLY AVAILABLE FPGA-BASED HPCS SYSTEMS

their different implementations.

We consider two asynchronous pipeline tasks because Convey supply custom firmware for pseudo-random number generation but rely on soft cores for matrix multiplication.

II. COMMERCIAL FPGA-BASED HPC

Till recently, Convey, Cray, SRC and Nallertech all made FPGA-based HPCS. Table II compares their technology and performance.

We chose to investigate the HC-1 as opposed to the alternative devices because it is the newest, most powerful FPGA-based HPCS system that is currently commercially available.

A. Convey HC-1 Architecture

The HC-1 is a 2U server card that uses four Virtex-5's as application engines (AE) to execute the distributed processes. The HC-1 also uses another Virtex-5 for process management and eight Stratix-II's for memory interfaces. Figure 1 shows their arrangement.

The resulting system has 128GB of DDR2 RAM with a maximum bandwidth of 80GB/sec. The memory address space is virtually shared with the host processor, making memory allocation and management simple.

B. The development model

HC-1 applications can be developed for in C, C++ and Fortran. The compiler will, where possible, vectorize the sequential code so that it is executed in parallel on 32 soft core processors implemented across all four AEs and running at 300MHz. However there are restrictions to what can be vectorised. These restrictions include intra-loop dependencies and conditional statements.

To take advantage of its reconfigurable architecture Convey sells various "personalities": firmware optimised for different applications. However these personalities only affect

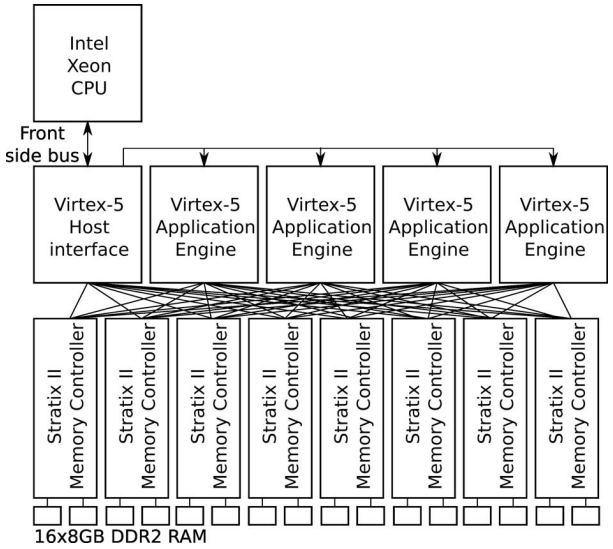

Figure 1. Structure of HC-1

the application engines, not the memory controllers. We compare the performance of three of these personalities, 32-bit soft core processors, 64-bit soft core processors and a collection of functions for financial applications.

The HC-1 also supports the Basic Linear Algebra Subroutines (BLAS) libraries, which include functions for matrix and vector algebra. These functions are implemented on the 32 or 64-bit soft core processors.

Finally the HC-1 can be developed for in Verilog and VHDL however these are intensive development models. We understand Convey are also working to build a C-to-HDL development environment at the moment.

III. GPU-BASED HPC

The GTX285 is a member for the GeForce 200b family of GPUs made by Nvidia. It has 240 core processors running at 1.4GHz and supports up to 4GB of external DDR3 RAM (we use a 1GB version) with a maximum bandwidth of 159GB/sec. Each core is grouped with seven others and two small but low latency memories: shared memory and a texture cache. In contrast to the shared memory, the texture cache is optimised for 2D spatial locality and can access the global memory directly. Figure 2 shows its component arrangement and interfaces.

A. The development model

CUDA is a set of extensions for C. Unlike the sequential development model for the HC-1, a CUDA developer will write a "kernel" of code that is run by each thread in parallel. The host program will dispatch these kernels asynchronously as threads to run on the multiprocessors of the GPU. Message passing between threads is performed using shared and texture memories within each multiprocessor, or global memory when the message must pass between different

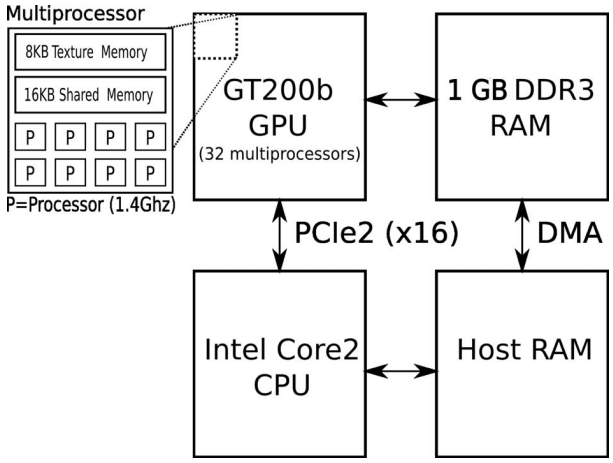

Figure 2. Structure of GTX285

multiprocessors. Threads can be locally synchronised with other threads running on the same multiprocessor, however global synchronisation can only be performed by stopping and re-dispatching the kernel.

The BLAS libraries have also been ported for GPU implementation.

A set of bindings for CUDA have been developed to enable CUDA development in higher-level languages such as Python (PyCuda, PyCublas), Fortran, Java and Matlab. These make the development of distributed code simple. For instance, using PyCublas and the numerical python libraries two matrices can be declared and multiplied using four lines of code:

```
1 include pycublas, numpy
2 A = pycublas.CUBLASMatrix(numpy.
      matrix(numpy.random.random
      ((100,100)),dtype=numpy.float32))
3 B = pycublas.CUBLASMatrix(numpy.
      matrix(numpy.random.random
      ((100,100)),dtype=numpy.float32))
4 C = (A*B).np_mat()
```

B. Initial comparisons and planned benchmarks

The HC-1 is at a significant disadvantage in terms of cost, power and floating-point operations per second. The GTX285 is a significantly less expensive, uses half the power during operation and can theoretically perform at 1062 GFlop/s. However the GPU has a fixed, 32-bit architecture and the only way to synchronise all of its threads is by stopping and restarting its kernel externally. Thus we expect the GPU to fare worse for 64-bit operations and operations requiring synchronisation.

In order to compare the productivity of both platforms we first developed code for the devices using a development model common to both platforms. This is the development of solutions in C using the libraries, firmware and compilers supplied by the manufacturers. Thus, the development time should be approximately equivalent. Then we benchmarked the performance of both devices against the performance of a single core of an Intel 2GHz Quad (Core2) Xeon with 4GB DDR3 RAM.

We see the smaller RAM available to the GPU as a device-specific limitation not representative of the broader technology. Thus we only consider benchmarks that can be executed within the 4GB RAM available to the GTX285. Also these benchmarks do not consider the time required to transfer data to and from the devices for the following reasons:

- Some of the benchmark times are overwhelmed by the memory transfer rates. This is particularly true for the smaller problems.
- We consider the tasks as likely components of larger simulations for which the memory transfer rates are negligible.
- The GTX285 supports overlapping of data transfers with kernel executions. That the HC-1 currently does not is a limitation of the Convey compilers not the device itself. In order for the benchmarks to best represent the FPGA family of HPC devices we need to ignore this software limitation.
- The bandwidth of a PCIe2(x16) port and the front side bus are both approximately 8GB/s, so should not bias the benchmarks.

IV. Random number generation

We used a Mersenne twister [14] pseudo-random number generator (PRNG) to create batches of 32-bit random numbers. Both Nvidia and Convey provide a Mersenne-twister as library functions. Whereas the Nvidia PRNG is implemented as custom software on a fixed architecture, the Convey PRNG uses a pipeline shift-register architecture in custom firmware as part of their financial applications personality.

Figure 3 shows the performance of the Nvidia GTX285 and the Convey HC-1 when generating increasingly large batches of random numbers. This performance is measured as an improvement over a single core CPU implementation of a Mersenne Twister.

The HC-1 performs, on average, 88.9 times better than the CPU. The GTX285 performs 89.3 times better than the CPU. However because the GPU uses a batch processing architecture it is much more sensitive to the size of the batch than the HC-1's pipeline architecture. Also the memory available to the FPGAs on the HC-1 is 128 times greater than that available to the GTX285, so larger batches can be generated.

A similar comparison by Tian and Benkrid [15] showed that an FPGA implementation using on-chip RAM of the Mersenne Twister algorithm could outperform this GPU implementation by a factor of $\sim 8\times$. This suggests that the overhead required for off-chip RAM and host-processor

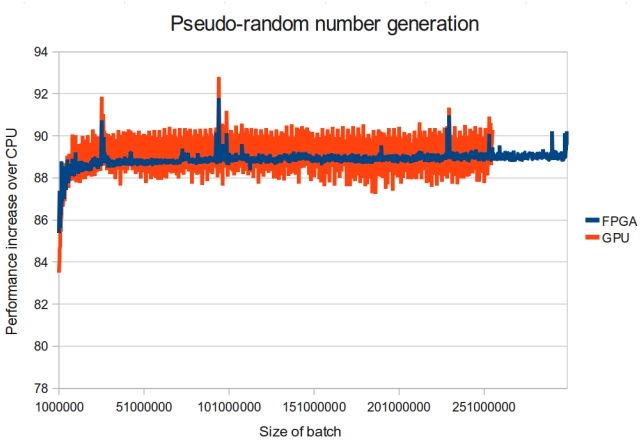

Figure 3. Performance of GPU and FPGA architectures generating random numbers

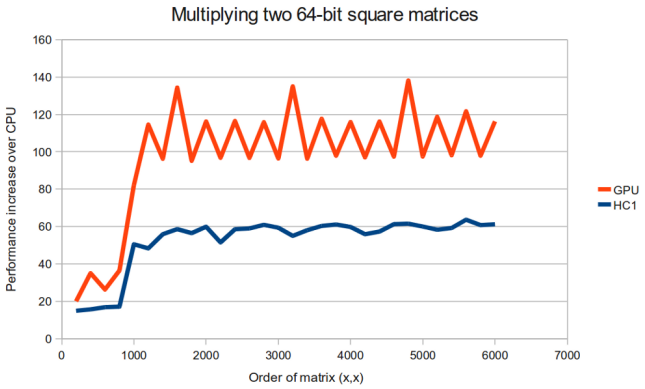

Figure 5. Performance of GPU and FPGA architectures calculating product of two 64-bit matrices

interfaces are reducing the effective performance of the HC-1.

V. Matrix Multiplication

Here we tested the performance of each architecture when multiplying two large square matrices. We repeated this experiment with 32-bit and 64-bit floating-point arithmetic.

We used the BLAS routines blas:sgemm and blas:dgemm available to each device[1]. The Nvidia card performed the calculations on 240 single-precision hardware cores. The HC-1 used 32 soft cores, however they were optimised for either 32-bit or 64-bit calculations, depending on the experiment. Figure 4 shows the performance improvement over the reference CPU implementation of the same BLAS routines for 32-bit matrices. Figure 5 shows the performance improvements for 64-bit matrices.

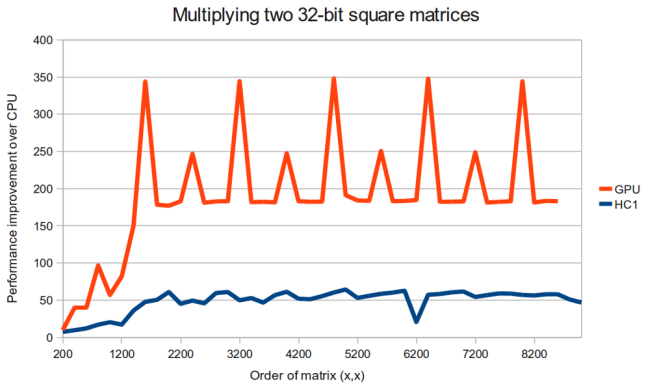

Figure 4. Performance of GPU and FPGA architectures calculating product of two 32-bit matrices

[1]All code available at http://cas.ee.ic.ac.uk/people/dhjones

The HC-1 performs, on average, 48.8 times better than the CPU on 32-bit matrices and 52.5 times better on 64-bit matrices. The GPU performs 190.4 times better on 32-bit matrices and 98.0 times better on 64-bit matrices. The GPU performance peaks occur when the width of the matrix is a multiple of the size of the available shared memory (16kb for every group of eight cores), allowing the use of a more efficient full tile matrix multiplication function.

Altera benchmarked a Stratix-II with custom firmware for double precision matrix multiplication [16] and concluded it could sustain 14.25GFlop/s performance. This compares to the 66.4GFlop/s ($2x^3$ flops for an x by x matrix) average performance of the GTX285 and 23.4GFLop/s average performance of the HC-1 for double precision matrix multiplication.

VI. Scalar Sum of Vector

Determining the scalar sum of a vector is a "reduce" operation requiring a partially synchronous tree process architecture (see figure 6).

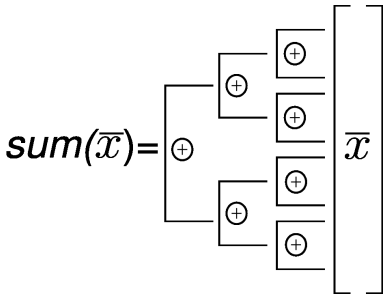

Figure 6. Process architecture for a "Reduce" operation

We performed this experiment using the BLAS routines blas:sasum and blas:dasum. Figure 7 shows the performance improvement over the reference CPU implementation of the same BLAS routines for 32-bit vectors. Figure 8 shows the performance improvements for 64-bit vectors.

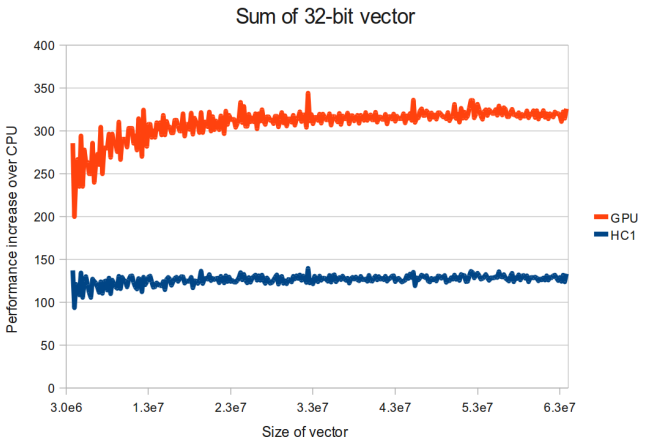

Figure 7. Performance of GPU and FPGA architectures calculating sum of 32-bit Vector

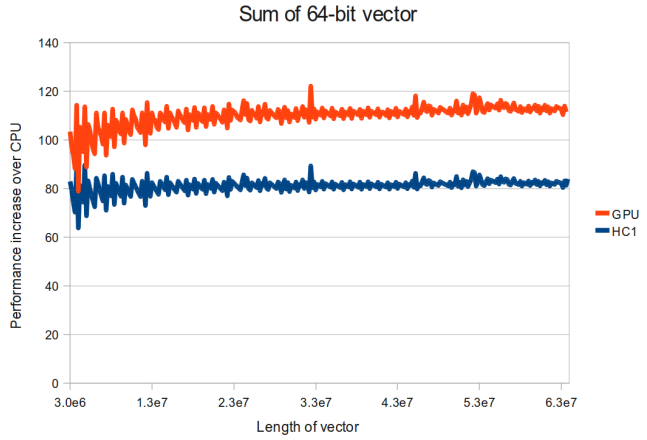

Figure 8. Performance of GPU and FPGA architectures calculating sum of 32-bit Vector

The HC-1 performs, on average, 125.62 times faster than the CPU for 32-bit vectors and 81.1 times better for 64-bit vectors. The GPU performs on average 306.4 times better than the CPU for 32-bit vectors and 109.3 times better than the CPU for 64-bit vectors. Again we see the predicted performance improvement for 32-bit operations but here has been no obvious performance reduction due to the requirement for synchronisation.

VII. N-body simulation

As a final benchmark a two-dimensional, second-order simulation of the gravitational forces between a number of bodies of different mass was employed. The execution cycle is:

1) For each body, i, calculate the vector gravitational force, $F_{i,j}$, that is acting on it from every other body, $j, j \neq i$ using the following equation:

$$F_{i,j} = \frac{m_j(\overline{x_i} - \overline{x_j})}{|\overline{x_i} - \overline{x_j}|^2} \quad (1)$$

2) For each body, i, sum the vector gravitational forces, $F_{i,j}$ acting on it using the blas:sasum routine.

$$F_i = m_i G \sum_{j \neq i} F_{i,j} \quad (2)$$

Where m is the mass of each body, x is the position of each body and G is a scaling factor (we use $G = 0.001$).

3) Synchronise each thread
4) Calculate the new velocities of each body using the following equation:

$$\overline{v}_{t+1,i} = \overline{v}_{t,i} + \frac{\overline{F}_i}{m_i}.t \quad (3)$$

5) Calculate the new positions of each body using the following equation:

$$\overline{x}_{t+1,i} = \overline{x}_{t,i} + \overline{v}_{t+1,i}.t \quad (4)$$

Where t is the unit time, (we use $t = 1$).

6) Synchronise each thread
7) Repeat to (1) 100 times.

This task has been chosen in part because it is a common scientific model and in part because it requires a fully synchronised mesh process architecture. All the calculations were performed to 32-bit precision. Figure 9 shows the performance improvement over a reference CPU implementation.

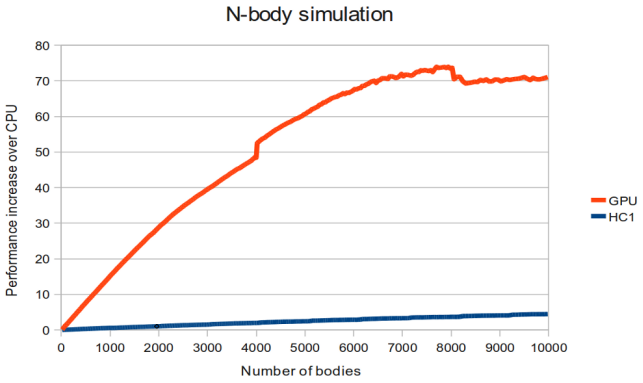

Figure 9. Performance of GPU and FPGA architectures doing N-body simulations

The only way it is possible to globally synchronise the threads of the GPU was by stopping the kernel and restarting it. For this benchmark, the performance of the GPU is on average 43.2 times greater than the CPU. The cost of completely synchronising the GPU has reduced its performance relative to the other benchmarks. However,

it still significantly outperforms the HC-1, which ran the benchmarks on average 1.9 times faster than the CPU. The improved performance on the GPU of systems between 4800 and 9600 bodies is due to the model using a more efficient allocation of threads between the 240 cores.

A similar comparison by Tsoi and Luk [17] using customised hardware and firmware concluded that an FPGA-based n-body simulation can run $\sim 2\times$ faster than a GPU. We adapted this simulation to better imitate his work (simulating 81920 bodies for one iteration in 3D) and reran our simulations. Our GPU simulation ran slightly faster (7.8s versus 9.2s) and our compiled Convey code ran much slower than their custom hardware and firmware (37.9s versus 5.62s). Thus if the development of custom hardware and firmware do not significantly reduce the productivity of a simulation, FPGA based HPCS can still outperform ($1.4\times$ faster) our GPU software.

VIII. Conclusions

We have evaluated the performance of the Convey HC-1 and the Nvidia GTX285 against a range of tasks common to HPCS-based scientific research. In all cases, both platforms outperformed an equivalent CPU implementation. For most of these the GPU significantly outperformed the FPGA architecture. The one exception, the generation of pseudo-random numbers, used closed-source firmware customised for both the task and the platform. We suggest that without a standardised FPGA HPCS platform about which open-source firmware could be developed, the future for FPGA-based HPCS will be increasingly marginalised to specialist applications. Further, the only people both sufficiently equipped and capable to develop the necessary firmware for FPGA-based HPCS will be the hardware developers. Supporting this conclusion is the fact that Cray no longer sell FPGA-based supercomputers and their latest product line (CX1) instead uses Nvidia GPUs.

IX. Acknowledgements

The authors acknowledge the support received from EPSRC on grants EP/C549481 and EP/E045472. The authors also thank Nvidia and Convey Computer for their assistance.

References

[1] J. Kepner, "HPC productivity: An overarching view," *International Journal of High Performance Computing Architectures*, vol. 18, no. 4, pp. 393, 2004.

[2] S. Craven and P. Athanas, "Examining the viability of FPGA supercomputing," *EURASIP Journal on Embedded systems*, vol. 2007, no. 1, pp. 13, 2007.

[3] T. Takagi and T. Maruyama, "Accelerating HMMER search using FPGA," *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 332–337, 2009.

[4] M. Chiu and M. C. Herbordt, "Efficient particle-pair filtering for acceleration of molecular dynamics simulation," *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009.

[5] N. Wolter, M. O. McCracken, A. Snavely, L. Hochstein, T. Nakamura, and V. Basili, "What's working in HPC: Investigating HPC user behavior and productivity," *CTWatch Quarterly*, November 2006.

[6] Open FPGA Alliance, "OpenFPGA general API specification 0.4," *www.openfpga.org*, 2008.

[7] Convey computer, "The Convey HC-1: The world's first hybrid-core computer," *www.conveycomputer.com*, 2009.

[8] D. K. G. Campbell, "Towards the classification of algorithmic skeletons," *Technical Report YCS 276, Department of Computer Science, University of York*, 1996.

[9] D. M. Goodeve, "Performance of multiprocessor communications networks," *PhD Thesis, University of York*, 1994.

[10] W. D. Smith and A. R. Schnore, "Towards an RCC-based accelerator for computational fluid dynamics applications," *The Journal of Supercomputing*, vol. 30, no. 3, pp. 239–261, 2004.

[11] SRC computers, "SRC-7 MAPstation product sheet," *src-comp.com*.

[12] A. J. van der Steen, "Overview of recent supercomputers," *phys.uu.nl*, 2005.

[13] Tony Brewer, "Instruction set innovations for the Convey HC-1 computer," *conveycomputer.com*.

[14] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[15] X. Tian and K. Benkrid, "Mersenne twister random number generation on FPGA, CPU and GPU," *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems*, 2009.

[16] Altera, "Designing and using FPGAs for double-precision floating-point math," *Altera white paper*, 2007.

[17] K. Tsoi and W. Luk, "Axel: A heterogeneous cluster with FPGAs and GPUs," *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 115–124, 2010.