Comparing Performance and Energy Efficiency of FPGAs and GPUs for High Productivity Computing

Brahim Betkaoui, David B. Thomas, Wayne Luk

Department of Computing, Imperial College London
London, United Kingdom
{bb105,dt10,wl}@imperial.ac.uk

***Abstract*—This paper provides the first comparison of performance and energy efficiency of high productivity computing systems based on FPGA (Field-Programmable Gate Array) and GPU (Graphics Processing Unit) technologies. The search for higher performance compute solutions has recently led to great interest in heterogeneous systems containing FPGA and GPU accelerators. While these accelerators can provide significant performance improvements, they can also require much more design effort than a pure software solution, reducing programmer productivity. The CUDA system has provided a high productivity approach for programming GPUs. This paper evaluates the High-Productivity Reconfigurable Computer (HPRC) approach to FPGA programming, where a commodity CPU instruction set architecture is augmented with instructions which execute on a specialised FPGA co-processor, allowing the CPU and FPGA to co-operate closely while providing a programming model similar to that of traditional software. To compare the GPU and FPGA approaches, we select a set of established benchmarks with different memory access characteristics, and compare their performance and energy efficiency on an FPGA-based Hybrid-Core system with a GPU-based system. Our results show that while GPUs excel at streaming applications, high-productivity reconfigurable computing systems outperform GPUs in applications with poor locality characteristics and low memory bandwidth requirements.**

I. Introduction

In recent years clusters built from commodity processors have been a common choice for High Performance Computing (HPC), as they are cheap to purchase and operate. However, HPC systems constructed from conventional microprocessors now face two key problems: the reduction in year-on-year performance gains for CPUs, and the increasing cost of power supply and cooling as clusters grow larger.

One way of addressing this problem is to use a heterogeneous computer system, where commodity processors are augmented with specialized hardware that can accelerate specific kernels. Examples of specialized hardware include Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). Such hardware accelerators can offer higher performance and energy efficiency compared to commodity processors. However, programming these co-processing architectures, in particular FPGAs, requires software developers to learn a whole new set of skills and hardware design concepts, and accelerated application development takes more time than producing a pure software version.

One potential solution to the problem of programming FPGA-based accelerated systems is High Productivity Reconfigurable Computing (HPRC). We define a HPRC system as a high performance computing system that relies on reconfigurable hardware to boost the performance of commodity general-purpose processors, while providing a programming model similar to that of traditional software. In a HPRC system, problems are described using a more familiar high-level language, which allows software developers to quickly improve the performance of their applications using reconfigurable hardware. Our main contributions are:

- Performance evaluation of benchmarks with different memory characteristics on two high-productivity platforms: an FPGA-based Hybrid-Core system, and a GPU-based system.
- Energy efficiency comparison of these two platforms based on actual power measurements.
- A discussion of the advantages and disadvantages of using FPGAs and GPUs for high-productivity computing.

II. Related Work

Several researchers have explored ways to make FPGA programming easier by giving developers a more familiar C-style language instead of hardware description languages [1], [2]. However, these languages require a significant proportion of an existing application to be rewritten using a programming model specific to the accelerator, which is problematic in an environment where large amounts of legacy code must be maintained.

There is previous work on comparing the performance of FPGAs and GPUs. For example, it is reported that while FPGA and GPU implementations for real-time optical flow have similar performance, the FPGA implementation takes 12 times longer to develop [3]. Another study shows that FPGA can be 15 times faster and 61 times more energy efficient than GPU for uniform random number generation [4]. A third study shows that for many-body simulation, a GPU implementation is 11 times faster than an FPGA implementation, but the FPGA is 15 times better than the GPU in terms of performance per Watt [5].

Prior work on comparing FPGAs and GPUs for high productivity computing used a set of non-standard benchmarks that target different process architectures [6] such as asynchronous pipeline and partially synchronous tree. Using the benchmarks in [6] results in analysis that considers processes at architectural level. However, these benchmarks do not cover

978-1-4244-8983-1/10/$26.00 ©2010 IEEE

applications with different memory access characteristics, and they do not address the performance of GPUs for non-streaming applications in comparison to a HPRC system. In our work, we take a different approach by selecting specific benchmarks with different memory locality characteristics. Our benchmark results lead us to a different conclusion from the one reported in [6] about HPRC systems being marginalised by GPUs. Moreover, we also provide comparison of energy efficiency for FPGA and GPU technologies.

III. Characterising Productivity and Locality

Creating a meaningful productivity measure which can be applied to programming models and architectures would require the aggregation of results from many individual projects, and is outside the scope of this paper. Instead we restrict our productivity study to the idea that if the same programmer effort is applied to two systems, then the system providing the highest performance solution is the most productive. This stems from the following equation that was developed in [7]:

$$Productivity = \frac{Relative\ Speedup}{Relative\ Effort} \quad (1)$$

So in our work, we will restrict our productivity comparison to using a common programming model based on familiar high-level programming languages and development tools. The benchmarks are developed using the C language plus platform specific extensions, using comparable development efforts in terms of design time and programmer training. In other words, productivity will be dictated by the relative speedup achieved by a platform:

$$\frac{Productivity(GPU)}{Productivity(FPGA)} = \frac{Relative\ Speedup(GPU)}{Relative\ Speedup(FPGA)} \quad (2)$$

In contrast to previous work [6], we select a number of established benchmarks based on the HPC Challenge (HPCC) Benchmark Suite [8] and the Berkeley technical report on Parallel Computing Research [9] to measure the relative speedup for each platform. Four benchmark programs are used:

- The STREAM benchmark.
- Dense matrix multiplication.
- Fast Fourier Transform (FFT).
- Monte-Carlo methods for pricing Asian options.

The authors in [9] identify the main performance limiting factor for these benchmarks, shown in Table I. Some of these benchmarks are part of the HPPC benchmarks, which aim to explore spatial and temporal locality by looking at streaming and random memory access type benchmarks, as illustrated in Fig.1. Where possible these benchmark programs are implemented using vendor libraries, which are optimised for each platform.

A. STREAM

The STREAM benchmark [8] is a simple synthetic benchmark that is used to measure sustained memory bandwidth to main memory using the following four long vector operations:

TABLE I
Limits to performance of benchmarks

Benchmark Program	Main performance limiting factor
STREAM	Memory Bandwidth limited
Dense matrix multiplication	Computationally limited
Fast Fourier Transform	Memory Latency limited
Monte-Carlo Methods	Parallelism limited

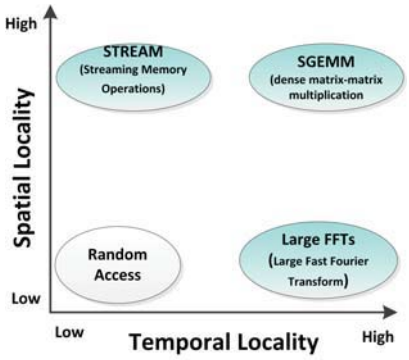

Fig. 1. Benchmark applications as a function of memory access characteristics

$$\texttt{COPY}: c \leftarrow a$$
$$\texttt{SCALE}: b \leftarrow \alpha c$$
$$\texttt{ADD}: c \leftarrow a + b$$
$$\texttt{TRIAD}: a \leftarrow b + \alpha c$$

where $a, b, c \in \mathbf{R}^m; \alpha \in \mathbf{R}$
The STREAM benchmark is designed in such a way that data re-use cannot be achieved. A general rule of this benchmark is that each vector must have about 4 times the size of all the last-level caches used in the run. In our work, we used arrays of 32 Million floating-point elements (4 bytes for each element), which require over 300MB of memory. Each vector kernel is timed separately and the memory bandwidth is estimated by dividing the total number of bytes read and written, by the time it takes to complete the corresponding operation.

B. Dense Matrix Multiplication

Dense floating-point matrix-matrix multiplication is a vital kernel in many scientific applications. It is one of the most important kernels in the LINPACK benchmark, as it is the building block for many higher-level linear algebra kernels. The importance of this benchmark has led HPC system vendors to both optimize their hardware and provide optimised libraries for this benchmark. In our work, we used vendor-provided libraries for the matrix multiplication benchmark, in order to achieve optimum or near optimal performance results for each platform.

The SGEMM routine in the BLAS library performs single precision matrix-matrix multiplication, defined as follows:

$$c \leftarrow \beta C + \alpha AB$$

where $A, B, C \in \mathbf{R}^{n \times n}; \alpha, \beta \in \mathbf{R}^n$

C. Fast Fourier Transform

The Discrete Fourier Transform (DFT) plays an important role for a variety of areas, such as biomedical engineering, mechanical analysis, analysis of stock market data, geophysical analysis, and radar communications [10]. The Fast Fourier Transform [11] is an efficient algorithm for calculating the DFT and its inverse. In particular, the FFT requires $O(N)$ memory accesses versus only $O(N \log N)$ floating-point operations, requiring not only high computation throughput but also high memory bandwidth [12]. In addition, unlike SGEMM, FFT requires non-unit stride memory access, and hence exhibits low spacial locality [13].

D. Monte-Carlo Methods for Asian options

Monte Carlo methods [14] are a class of algorithms that use psuedo-random numbers to perform simulations, allowing the approximate solution of problems which have no tractable closed-form solution. In this work, we examine financial Monte-Carlo simulations for pricing Asian options. Asian options are a form of derivative which provides a payoff related to the arithmetic average of the price of an underlying asset during the option life-time:

$$P_{call} = max(\frac{1}{n+1}\sum_{i=0}^{n} S(t_i) - K, 0) \quad (3)$$

where P_{call} is the payoff of the Asian call option, $S(t_i)$ is the asset price at time t_i, and K is the strike price.

Unlike European options, which can be priced using the Black-Scholes equation [15], there is no closed-form solution for pricing Asian options. Instead, Monte-Carlo methods are used to calculate the payoff of Asian options. Using Monte-Carlo methods leads to accurate results at the expense of a large number of simulations, due to the slow convergence of this method [16].

We choose Monte Carlo methods as one of the benchmarks for two reasons. First, they are widely used in many applications including finance, physics, and biology. Second, they support highly parallel execution [9] with low memory bandwidth requirements, which would map well onto FPGAs and GPUs.

IV. Productive Reconfigurable Computing

A. Convey HC-1 Architecture

Convey's Hybrid-Core technology is an example of a HPRC system that achieves a compromise between application-specific hardware and architectural integration. In the Hybrid-Core approach, the commodity instruction set, Intel's x86-64, is augmented with application-specific instruction sets (referred to as personalities), which are implemented on an FPGA-based coprocessor to accelerate HPC applications. A key feature of this architecture is the support of multiple instruction sets in a single address space [17]. The Convey Hybrid-Core server, HC-1, has access to two pools of physical memory: the host memory pool with up to 128GB of physical memory located on the x86 motherboard, and the coprocessor memory pool with up to 128GB of memory, located on the coprocessor board. An overview of the architecture of the HC-1 is shown in Fig. 2.

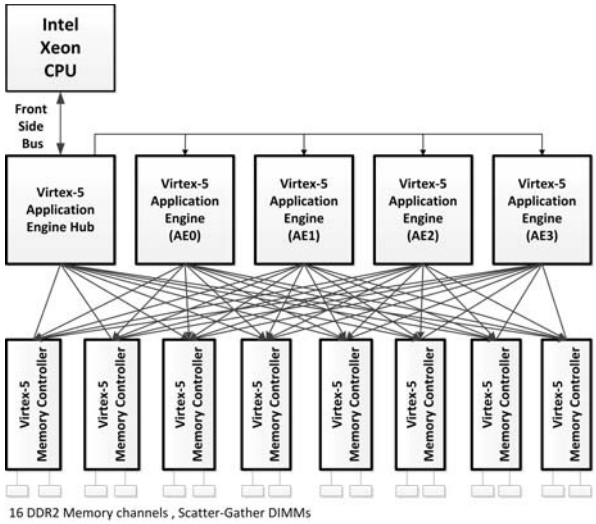

Fig. 2. HC-1 system organisation

The coprocessor contains three main components: the Application Engine Hub (AEH), the Memory Controllers (MCs), and the Application Engines (AEs). The AEH represents the central hub for the coprocessor. It implements the interface to the host processor via a Front-Side Bus (FSB) port, an instruction fetch/decode unit, and a scalar processor to execute scalar instructions. All extended instructions are passed to the application engines. The coprocessor has eight memory controllers that support a total of 16 DDR2 memory channels offering an aggregate of over 80GB/s of bandwidth to ECC protected memory. Finally, the AEs implement the extended instructions that would be responsible for any performance gains. Each Application Engine is connected to all the eight memory controllers via a network of point-to-point links to provide high sustained bandwidth without a cache hierarchy.

The Convey HC-1 used in this work has a single multi-core Intel Xeon 5138 processor running at 2.13GHz with 8GB of RAM. The HC-1 Coprocessor is configured with 16GB of accelerator-local memory. Its AEs consist of four Xilinx V5LX330 FPGAs running at 300MHz. The memory controllers are implemented on eight Xilinx V5LX155 FPGAs, while the the AEH is implemented on two Xilinx V5LX155 FPGAs [18].

B. HC-1 Development Model

The HC-1 programming model is shown in Fig. 3. A number of key features distinguish the Convey programming model from other coprocessor programming models [19]. Applications are coded in standard C, C++, or Fortran. Performance can then be improved by adding directives and pragmas to guide the compiler. In an existing application, a routine or section of code may be compiled to run on the host processor, the coprocessor, or both. A unified compiler is used to generate

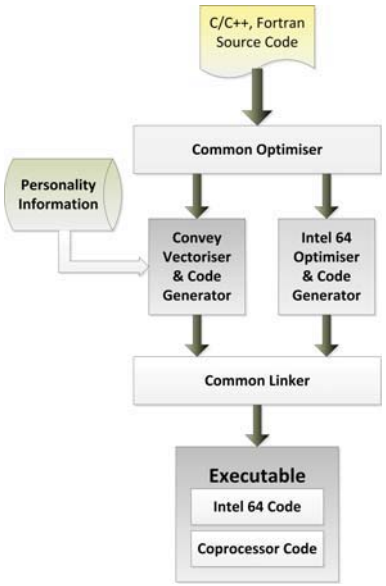

Fig. 3. Programming model of the Convey Hybrid-Core

both x86 and coprocessor instructions, which are integrated into a single executable and can execute on both standard x86 nodes and accelerated Hybrid-Core nodes. The coprocessor's extended instruction set is defined by the personality specified when compiling the application.

Porting an application to take advantage of the Convey coprocessor capabilities can be achieved using one or more of the following approaches:

- Use the Convey Mathematical Libraries (CML) [20], which provide a set of functions optimised for the accelerator.
- Compile one or more routines with Convey's compiler. This can be performed by using the Convey auto-vectorization tool to automatically select and vectorize DO/FOR loops for execution on the coprocessor. Directives and pragmas can also be manually inserted in the source code, to explicitly indicate which part of a routine should execute on the coprocessor.
- Hand-code routines in assembly language, using both standard instructions and personality specific accelerator instructions. These routines can then be called from C, C++, or Fortran code.
- Develop a custom personality using Convey's Personality Development Kit (PDK).

C. Convey Personalities

A personality groups together a set of instructions designed for a specific application or class of applications, such as finance analytics, bio-informatics, or signal processing. Personalities are stored as pre-compiled FPGA bit files, and the server dynamically reconfigures between bit-files at run-time to provide applications with the required personality. Convey provides a number of personalities optimized for certain types of algorithms such as floating-point vector personalities, and a financial analytics personality. In our work, we used three Convey personalities: the single-precision vector personality, the double-precision vector personality, and the financial analytics personality.

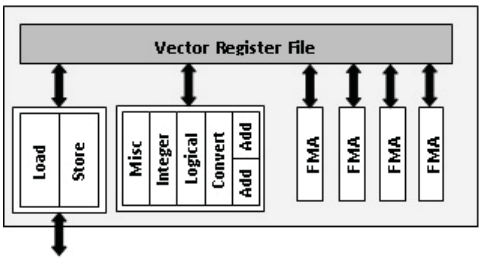

Fig. 4. A function pipe in the Convey single-precision personality

D. Optimisation for the 4 benchmarks

The implementation of the STREAM benchmark on the HC-1 is straight forward, as it only consists of vector operations. For the SGEMM and FFT benchmarks, we use the Convey Mathematical Libraries (CML) [20]. The CML library provides a collection of linear algebra and signal processing routines such as BLAS and FFT. In addition, Convey provides a single-precision *floating-point vector personality* which implements the CML libraries, as well as vector operations for both integer and floating-point data types. This personality is based on a load-store architecture with modern latency-hiding features. Fig. 4 shows a diagram of one of the 32 function pipes that are available in the single precision personality. The double precision version has only two Fused Multiply-Add (FMA) units and one adder unit. We can calculate the peak floating-point performance of the HC-1 for the single and double precision vector personalities as follows:

$$FLOPS_{(peak)} = Number\,of\,FP\,units \times Clock\,Frequency$$

The single precision personality has 32x4 fused multiply-add units, which means it can perform 32x4x2 floating-point operations per clock cycle. The vector personalities are clocked at 300 MHz, and so the floating-point peak performance of the HC-1, when the vector personalities are loaded onto the coprocessor, is 76.8 GFlops and 38.4 GFlops for single and double precision respectively.

Finally, the Asian option pricing benchmark contains two parts: pseudo-random number generation, and Monte-Carlo based simulation paths. The financial analytics personality provides custom hardware for generating random numbers using the Mersenne-Twister algorithm. The second part of the benchmark contains nested loops that can be vectorised for efficient execution on the HC-1 coprocessor.

V. GPU-based Computing

A. CUDA architecture

A CUDA GPU [21] has a massively-parallel many-core architecture that supports numerous fine-grained threads, and has fast shared on-chip memories that allow data exchange

between threads. In this work, we use nVidia's Tesla C1060 GPU which has 240 streaming processors running at 1.3GHz. It has a theoretical peak single-precision floating-point performance of 933 GFlops. It also has access to 4GB of GDDR3 memory at 800MHz, offering up to 102GB/sec of memory bandwidth [22].

B. CUDA Development Model

The NVIDA CUDA SDK environment provides software developers with a set of high-level languages such as C and Fortran. These programming languages are extended to support the key abstractions of CUDA such as hierarchical thread groups, shared memories, and barrier synchronization. CUDA C extends C by allowing the programmer to implement kernels as C functions that are executed in parallel by CUDA threads. These threads can be grouped in thread blocks where they share the processing and memory resources of the same processor core. The blocks are organized into a one, two, or three dimensional grid of thread blocks, with the number of thread blocks chosen according to the size of the processed data and the processing resources. The CUDA programming model assumes that the CUDA threads execute on a physically separate device, operating as a coprocessor to the host running the C program. This is the case, for example, when the kernels execute on a GPU and the rest of the C program executes on a CPU.

C. Development and optimisation of the benchmarks

The STREAM benchmark is implemented as 4 CUDA kernels, with each kernel being executed by a large number of threads to achieve optimum performance. The number of threads is set so that the first vector kernel yields the same performance as the CUBLAS:sscopy routine. For dense matrix multiplication we use the CUBLAS library [23], provided by nVidia for linear algebra kernels. Given the low double-precision floating-point peak performance (78 GFLOPS) of the Tesla C1060 compared to its single precision peak performance (933 GFLOPS), we only consider the single precision version (SGEMM). [1]

The Asian option pricing application is implemented using two kernels: a pseudo-random number generator (PRNG), and a path simulator. The PRNG is implemented using nVidia's Mersenne Twister RNG. For the path simulator kernel, each thread simulates a price movement path and then the results from each thread are summed in a hierarchical fashion starting with threads from the same block using shared memory. The partial results are then stored in the global memory, and final aggregation takes place for the valuation of the option price.

VI. Power Measurement

Full system power is measured using the Olson remote power monitoring meter [24]. A diagram of the power measurement system is shown in Fig. 5, which consists of three main components: the system under test (SUT), the remote power meter, and the computer that receives the power readings over the network. The SUT is connected to the mains supply via the remote power meter, which captures and logs the power measurements once every second through an Ethernet connection. The SUT is either the HC-1 coprocessor, the CPU (using one or multiple threads), or the Tesla C1060 GPU. Power is measured by calculating the difference between idle power and active power, so we only measure the dynamic power. The reason for only measuring dynamic power is that the GPU and the HC-1 coprocessor have different base configurations, such as power supplies and CPUs. In addition, since the resolution of the power meter is one second, we make sure that each accelerated kernel has a minimum execution time of 5-10 seconds, so that fluctuations are averaged out.

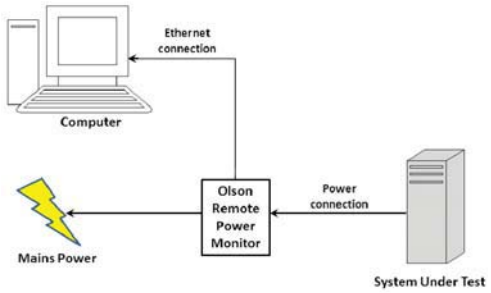

Fig. 5. Power measurement setup

TABLE II
Dynamic Power Measurements

Component	SGEMM	FFT	Monte-Carlo methods
CPU - single-threaded	27W	31W	20W
CPU - multi-threaded	55W	65W	31W
HC-1 Coprocessor	79W	103W	170W
Tesla C1060	71W	85W	138W

Table II shows the measured dynamic power of each platform for three benchmarks. The HC-1 coprocessor has the highest power consumption for all three benchmarks – this is not unexpected, as the HC-1 coprocessor contains 13 Virtex-5 FPGAs and 1024 memory banks. Given the average power consumption and the execution time of a benchmark, we can calculate its energy efficiency, measured as either the number of floating-point operations per second per Watt (Flops/Watt), or the number of Monte-Carlo simulation paths per Joule.

VII. Performance and Energy Efficiency Comparison

In the following, the results for performance and energy efficiency comparison are provided.

A. STREAM

Tables III and IV show the results for the STREAM benchmark on the HC-1 server and the Tesla C1060 GPU respectively. The size of the vectors used in our work is 32 million elements. From these results, we can see that the sequential memory bandwidth sustained by the GPU is about

[1]For double-precision comparison, it would be more appropriate to use the new Tesla C20x series, which has a double-precision floating-point peak performance of 515 GFlops, but these cards are still unavailable at the time of writing.

TABLE III
STREAM BENCHMARK RESULTS FOR HC-1

Function	Average Rate (GB/s)	Min Rate (GB/s)	Max Rate (GB/s)
COPY	35.82	35.77	35.88
SCALE	35.10	35.06	35.16
ADD	42.09	42.06	42.12
TRIAD	44.75	44.61	44.86

TABLE IV
STREAM BENCHMARK RESULTS FOR TELSA C1060

Function	Average Rate (GB/s)	Min Rate (GB/s)	Max Rate (GB/s)
COPY	73.84	73.24	74.34
SCALE	74.00	73.52	74.34
ADD	71.14	70.87	71.46
TRIAD	71.23	71.05	71.43

two times larger than that of the HC-1. These figures, when combined with the peak floating-point performance of GPUs and the HC-1, suggest that the GPU is likely to outperform the HC-1 for streaming programs with intensive computation and bandwidth requirements.

B. Dense Matrix Multiplication

In our work, we use the Intel MKL [25] for software matrix multiplication. The matrix-matrix multiplication benchmark results are shown in Fig. 6 and Fig. 7. The GPU is a clear winner in terms of both performance (up to 370 GFLOPS) and power efficiency (over 5GFLOPS/Watt). This is expected, since the performance of the matrix multiplication benchmark is computation bound, and the Tesla C1060 has over 10 times more floating-point peak performance than the HC-1. In addition, the HC-1 implementation offers no significant speed-up over a multi-threaded MKL implementation running on an Intel Xeon E5420 Quad-Core CPU.

The GPU is about 5 times faster than both the CPU and the Convey Coprocessor. This speed-up decreases to about 2.5 to 4.2 times if we include data transfer from the main memory to the GPU memory, while the HC-1 coprocessor can be slower than the CPU when data transfers from the host processor memory to the coprocessor memory are taken into account.

C. Fast Fourier Transform

In this work, we use the popular FFTW [26] for the CPU implementation, as FFTW supports multi-threading and is reported to be more efficient than its Intel MKL counterpart. Fig. 8 shows the performance of a one-dimensional in-place single-precision complex-to-complex FFT on the three platforms. The HC-1 outperforms the GPU (CUFFT) by up to a factor of three for large FFT sizes, and is about 16 times faster than the single-threaded FFTW, or about 4 times faster than the multi-threaded version.

The HC-1 implementation of the FFT achieves over 65 GFlops, thanks to its coprocessor memory subsystem that is optimized for non-unit stride and random memory accesses [17]. The Tesla C1060 uses GDDR memories which are optimised for sequential memory access operations and stream programming for graphics applications. This leads to a

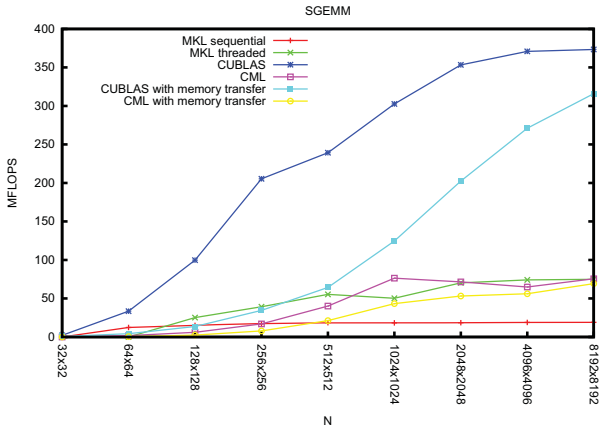

Fig. 6. SGEMM - floating-point performance

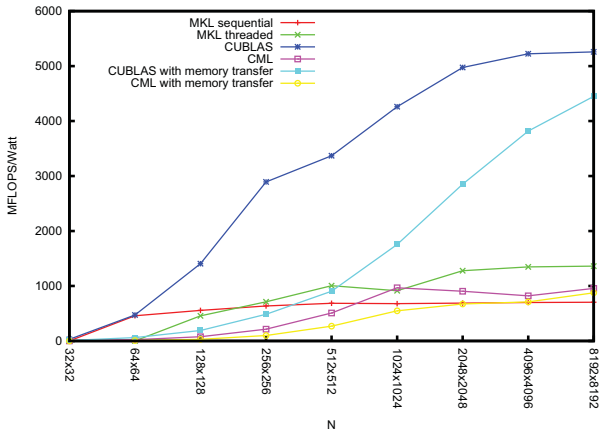

Fig. 7. SGEMM - energy efficiency

performance penalty if GPU applications, such as FFT, involve non-sequential memory accesses [12].

To further investigate the effect of strided memory access on the performance of the HC-1 and Tesla C1060 GPU, we conduct the following experiment. We measure the effective bandwidth that can be achieved with strided memory access,

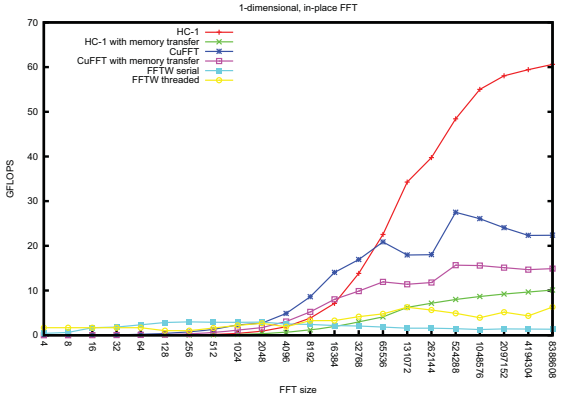

Fig. 8. 1-dimensional in-place single-precision complex-to-complex FFT

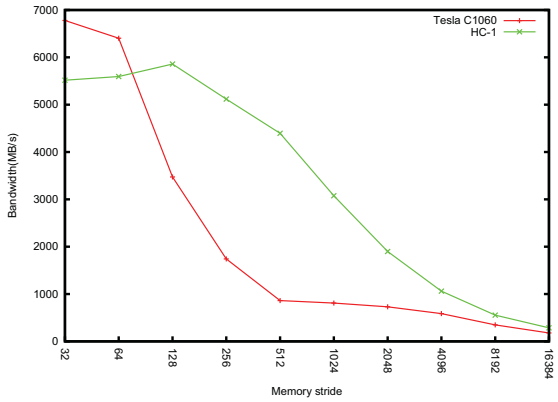

Fig. 9. Memory bandwidth for stride memory access

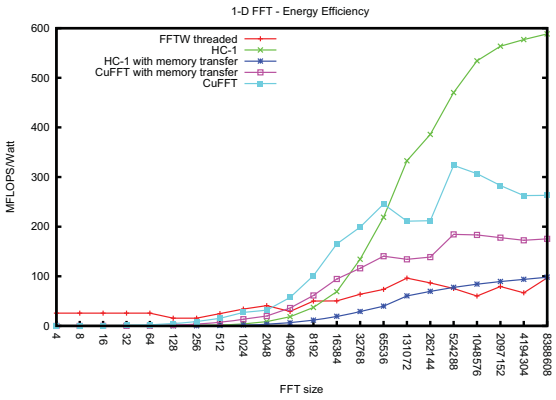

Fig. 10. Energy efficiency for 1-dimensional in-place single-precision complex-to-complex FFT

and how it compares to a baseline sequential memory access bandwidth. We used the BLAS routine blas:sscopy available to each platform. This routine copies a real vector into another real vector. The increment between two consecutive elements in each vector can be specified, i.e. the stride parameter. The results of this experiment for vectors of 32 million elements are shown in Fig. 9. As the stride parameter gets larger, the data transfer rate of the GPU becomes slower than that of HC-1.

However, the floating-point performance of the HC-1 drops significantly if data transfer times between host memory and coprocessor memory are taken into account. Fortunately, there are various applications that execute many FFT operations between memory transfers from CPU memory to coprocessor memory. An example of such an application is the protein-protein docking simulator ZDock [27], which uses a scoring scheme to determine the best docking positions. At the heart of ZDock is an FFT used for computing convolutions, and once the host processor has sent the initial input data, the coprocessor can perform multiple FFTs without significant data transfer.

In terms of energy efficiency, the HC-1 generally fares better than the GPU and the CPU, as shown in Fig. 10. It is twice as energy efficient as the GPU, and about 6 times more energy efficient than a multi-threaded CPU implementation for large FFT sizes.

D. Monte-Carlo Methods: Asian Option Pricing

Table V shows the performance of an HC-1 implementation of an Asian option pricing application, compared to the performance of CPU and GPU implementations. For the Asian option pricing benchmark, we select the same parameters as in [16], i.e. one million simulations over a time period of 356 steps. We use an Intel Xeon 5138 dual-core processor running at 2.13GHz with 8GB of memory for the CPU implementation. We also include performance results of an optimised FPGA implementation [16] coded in a hardware description language (HDL) targeting a Xilinx xc5lx330t FPGA clocked at 200MHz.

These results show that using the HC-1 coprocessor yields a performance improvement of up to 18 times over an optimized multi-threaded software implementation. The performance of the HC-1 is about the same as a single precision GPU implementation, and is twice as fast as the double precision version. The major reason for this performance improvement is the vectorization of the FOR loops, which form the bottleneck in the option pricing benchmark. Moreover, the random number generator is implemented in the HC-1 as a custom hardware library, whereas the CUDA GPU must use an instruction based approach. Note that currently the finance analytics personality for HC-1 does not support single-precision floating-point operations, so it cannot be compared directly with a hand-crafted single-FPGA version in single-precision floating-point arithmetic [16] also shown in Table V; however, there is no doubt that the hand-crafted version involves much more development effort.

In terms of energy efficiency, the GPU and the HC-1 coprocessor are only about 2 to 4 times more energy efficient than the CPU as shown in Table VI, with the HC-1 about two times more energy efficient than the GPU. The measured GPU and HC-1 power consumption is relatively higher than the other benchmarks. This can be explained by the fact that Monte-Carlo methods are ‘embarrassingly parallel’, which leads to near full utilisation of the hardware resources on HC-1 and the GPU.

TABLE V
PERFORMANCE RESULTS FOR ASIAN OPTION PRICING

Implementation	Execution time		Speed-up	
	Single	Double	Single	Double
CPU single-threaded	8,333 ms	14,727ms	0.53x	0.57x
CPU multi-threaded	4,446 ms	8,378 ms	1x	1x
Convey HC-1	-	471 ms	-	17.8x
Tesla C1060 [16]	440 ms	1,078 ms	10x	7.7x
HDL-coded FPGA [16]	115 ms	-	38.6x	-

TABLE VI
ENERGY EFFICIENCY FOR ASIAN OPTION PRICING

Implementation	Sim. Paths per Joule		Comparison vs CPU	
	Single	Double	Single	Double
CPU single-threaded	6,002	3,396	1x	1x
CPU multi-threaded	7,255	3,886	1.2x	1.14x
Convey HC-1	-	12,489	-	3.7x
Tesla C1060 [16]	16,469	6,722	2.74x	2x

VIII. CONCLUSIONS

This paper provides a comparison of FPGAs and GPUs for high productivity computing. Using a number of established benchmarks and a common development model, we evaluate the performance of a commercially-available high-productivity reconfigurable computing system (HPRCS), the Convey HC-1 server. We also compare its performance and energy efficiency to those of a multi-core CPU as well as a GPU-based HPC system. The selected benchmarks for our work have different memory access patterns ranging from high locality memory characteristics to low locality characteristics. The HC-1 and the GPU outperform the CPU for all of our benchmark programs. From our results, we can summarize the following about some of the pros and cons of the HPRCS and the GPU:

- GPUs often perform faster than FPGAs for streaming applications. GPUs usually enjoy a higher floating-point performance and memory bandwidth than FPGA-based HPC systems. The streaming performance of the HC-1 is often limited by the floating-point computational performance achieved by FPGAs.
- The HC-1 employs a memory system optimised for non-sequential memory accesses, which makes them faster and more energy efficient than GPUs for applications such as Fast Fourier Transform. GPUs, on the other hand, use GDDR memories which are optimized for sequential memory access operations, incurring a higher performance penalty for non-contiguous memory block transfers.
- The HC-1 demonstrates superior performance and energy efficiency for highly parallel applications requiring low memory bandwidth, as illustrated by the Monte Carlo benchmark for pricing Asian options.

ACKNOWLEDGEMENT.

The support of Imperial College London Research Excellence Award, the FP7 REFELCT (Rendering FPGAs for Multi-Core Embedded Computing) Project, the UK Engineering and Physical Sciences Research Council, HiPEAC, Convey Computer Corporation, and Xilinx, Inc. is gratefully acknowledged.

REFERENCES

[1] J. Xu, N. Subramanian, S. Hauck, and A. Alessio, "Impulse C vs. VHDL for accelerating tomographic reconstruction," 2009, University of Washington. [Online]. Available: http://ee.washington.edu/faculty/hauck/publications/ImpulsePET.pdf

[2] A. Putnam, D. Bennett, E. Dellinger, J. Mason, P. Sundararajan, and S. J. Eggers, "CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures," in *FPL'08*. IEEE, 2008, pp. 173–178.

[3] J. Chase, B. Nelson, J. Bodily, Z. Wei, and D. Lee, "Real-time optical flow calculations on FPGA and GPU architectures: A comparison study," in *16th International Symposium on Field-Programmable Custom Computing Machines*, 2008, pp. 173–182.

[4] D. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation," in *Proceeding of the ACM/SIGDA international symposium on field programmable gate arrays*. ACM, 2009, pp. 63–72.

[5] T. Hamada, K. Benkrid, K. Nitadori, and M. Taiji, "A comparative study on ASIC, FPGAs, GPUs and general purpose processors in the O(N^2) gravitational N-body simulation," *NASA/ESA Conference on Adaptive Hardware and Systems*, vol. 0, pp. 447–452, 2009.

[6] D. H. Jones, A. Powell, C.-S. Bouganis, and P. Y. Cheung, "GPU versus FPGA for high productivity computing," in *FPL'10*, 2010.

[7] M. V. Zelkowitz, V. R. Basili, S. Asgari, L. Hochstein, J. K. Hollingsworth, and T. Nakamura, "Measuring productivity on high performance computers." in *IEEE METRICS*. IEEE Computer Society, 2005, pp. 19–22.

[8] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. Mccalpin, D. Bailey, and D. Takahashi, "Introduction to the HPC challenge benchmark suite," Lawrence Berkeley National Laboratory, Tech. Rep., 2005.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: a view from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006.

[10] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and Y. Kaneda, "16.4-Tflops direct numerical simulation of turbulence by a Fourier spectral method on the earth simulator," in *SC'02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–17.

[11] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[12] A. Nukada, Y. Ogata, T. Endo, and S. Matsuoka, "Bandwidth intensive 3-D FFT kernel for GPUs using CUDA," in *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–11.

[13] V. Volkov and J. W. Demmel, "Benchmarking GPUs to tune dense linear algebra," in *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–11.

[14] N. Meteopolis and S. Ulam, "The Monte Carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[15] B. Fischer and S. Scholes Myron, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–54, May-June 1973.

[16] A. H. Tse, D. B. Thomas, K. Tsoi, and W. Luk, "Efficient reconfigurable design for pricing Asian options," in *HEART'10*, 2010.

[17] *Convey Reference Manual*, Convey Computer Corporation, 2010.

[18] T. M. Brewer, "Instruction set innovations for the Convey HC-1 computer," *IEEE Micro*, vol. 30, pp. 70–79, 2010.

[19] *Convey Programmers Guide*, Convey Computer Corporation, 2010.

[20] *Convey Mathematical Libraries User's Guide*, Convey Computer Corporation, 2010.

[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro-Institute of Electrical and Electronics Engineers*, vol. 28, no. 2, pp. 39–55, 2008.

[22] NVIDIA CUDA. Compute Unified Device Architecture. [Online]. Available: http://www.developer.nvidia.com/object/cuda.html.

[23] *CUDA CUBLAS Library*, NVIDIA, 2007.

[24] Olson Electronics Ltd. Olson Remote Power Monitoring. [Online]. Available: http://www.olson.co.uk/remote_monitor_range.htm

[25] Intel Math Kernel Library (MKL). [Online]. Available: http://software.intel.com/en-us/intel-mkl/

[26] FFTW, "Fastest Fourier Transform in the West – Homepage," 2003. [Online]. Available: http://www.fftw.org/

[27] R. Chen, L. Li, and Z. Weng, "ZDOCK: an initial-stage protein-docking algorithm," *Proteins: Structure, Function, and Bioinformatics*, vol. 52, no. 1, pp. 80–87, 2003.