Characterization of FPGA-based High Performance Computers

Karl Savio Pimenta Pereira

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Peter M. Athanas, Chair
Patrick R. Schaumont
Wu-Chun Feng

August 9, 2011
Blacksburg, Virginia

Keywords: HPC, FPGA, GPU, floating-point, integer-point, FFT, molecular dynamics

Characterization of FPGA-based High Performance Computers

Karl Savio Pimenta Pereira

(ABSTRACT)

As CPU clock frequencies plateau and the doubling of CPU cores per processor exacerbate the memory wall, hybrid core computing, utilizing CPUs augmented with FPGAs and/or GPUs holds the promise of addressing high-performance computing demands, particularly with respect to *performance*, *power* and *productivity*. While traditional approaches to benchmark high-performance computers such as SPEC, took an architecture-based approach, they do not completely express the parallelism that exists in FPGA and GPU accelerators. This thesis follows an application-centric approach, by comparing the sustained performance of two key computational idioms, with respect to performance, power and productivity. Specifically, a complex, single precision, floating-point, 1D, Fast Fourier Transform (FFT) and a Molecular Dynamics modeling application, are implemented on state-of-the-art FPGA and GPU accelerators. As results show, FPGA floating-point FFT performance is highly sensitive to a mix of dedicated FPGA resources; DSP48E slices, block RAMs, and FPGA I/O banks in particular. Estimated results show that for the floating-point FFT benchmark on FPGAs, these resources are the performance limiting factor. Fixed-point FFTs are important in a lot of high performance embedded applications. For an integer-point FFT, FPGAs exploit a flexible data path width to trade-off circuit cost and speed of computation, improving performance and resource utilization. GPUs cannot fully take advantage of this, having a fixed data-width architecture. For the molecular dynamics application, FPGAs benefit from the flexibility in creating a custom, tightly-pipelined datapath, and a highly optimized memory subsystem of the accelerator. This can provide a 250-fold improvement over an optimized CPU implementation and 2-fold improvement over an optimized GPU implementation, along with massive power savings. Finally, to extract the maximum performance out of the FPGA, each implementation requires a balance between the formulation of the algorithm on the platform, the optimum use of available external memory bandwidth, and the availability of computational resources; at the expense of a greater programming effort.

This thesis is dedicated to

Papa and Mama,

my grandmother Mãe

&

my loving godchild Ethan J. Mascarenhas

Acknowledgements

Firstly, I would like to express my deep and sincere gratitude to my advisor, Dr. Peter Athanas, for giving me the opportunity to be a part of the CCM team. He has been a source of inspiration throughout my research and it would not have been possible to complete this work without his constant support and personal guidance. His profound knowledge and logical way of thinking have been of great value to me. Above all, besides being an outstanding advisor and professor, he is one of the most caring and approachable persons I have had the privilege to know.

I would also like to thank Dr. Patrick Schaumont and Dr. Wu Feng for serving as members of my committee. I immensely enjoyed learning the fundamentals of hardware-software co-design under Dr.Schaumont. Working with Dr.Feng as a part of CHREC was an extremely rewarding experience.

I would like to express my gratitude to my loving family; brother Keith and his fiancé Mirella; sister Karen, brother-in-law Bosco, aunts Suzette and Maria Lourdes, uncle Ti. Jose Vincent, for being the staunch pillars contributing to my success, for encouraging me to pursue higher studies and being a constant source of support throughout my graduate education. To my grandmother Mae, for all the blessings and prayers. Matu for making it a point to call me without fail. To my cousins in the US and back home in Goa, I cannot thank you guys enough and I love you all a lot.

I am lucky to have made so many friends in Blacksburg, who have always been there

for me through thick and thin. To my best friends Pallavi Bishnoi, Kanchana Surendra, Jatin Thakkar, Nikhil Gupta, Tania Mutmainna, Santoshkumar Sambamoorthy, Lyndsi Calhoun, Rohitesh Rao, Natasha Das, Adrian Pinto, Shalini Ramesh, Nikhil Jain, Sudarshan Gopinath, Navish Wadhwa, Neha Rajora and anyone I might have missed out unintentionally; words cannot describe my appreciation for each and every one of you and for making my journey truly memorable. I would also like to thank my friends and room mates, Ankit Gupta, Vignesh Umapathy, Lloyd Gurjao, Deepanshu Arora, Saikat Chakrabarti, Akshay Gaikwad, Amar Deka, Amit Kalita and Mayank Daga, for accommodating my erratic behavior. To my close friends overseas, thanks for keeping in touch despite the distance. I would also like to thank my CCM lab mates; Abhay Tavaragiri, Rohit Asthana, Umang Parekh, Mrudula Karve, Kavya Shagrithaya for those intense fooseball sessions; Jacob Couch, for being a mentor and a great friend, Tony Frangieh, for the very insightful advice, and the rest of the gang, Xin Xin, Wenwei Zha, Andrew Love, Richard Stroop, Jason Forsyth and Teresa Cervero for making my graduate life a memorable odyssey.

Finally, I thank God for being my source of light and wisdom, and bestowing me with all the blessings from above.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

High-Performance Computing (HPC) is the use of parallel processing for the fast execution of complex computational problems in a variety of critical domains including climate and seismic modelling, medical imaging, oil and gas exploration, bioscience, data security, financial trading and more. Systems that deliver this performance fall under two categories:

- High-Performance Server Computing (HPSC): These systems are typically used by scientists, engineers and analysts to simulate and model applications and analyse large quantities of data, and
- High-Performance Embedded Computing (HPEC): Computers incorporated in equipment to performance specific compute and data intensive tasks.

A summary of the different applications for both server and embedded computing is provided in Table 1.1. It is interesting that these two extremes of system types exhibit different application characteristics and few of them overlap. However, looking ahead, these two ends of the computing spectrum seem to converge concerning the following:

- Power : Battery life limitations for cell phones as well as increasing cost of electricity and cooling in data centers.
- Hardware Utilization : Embedded systems are always sensitive to cost, but efficient use of hardware is also required to justify spending $10M to $100M for high-end servers.
- Software Reuse : As the size of embedded or server software increases over time, code reuse becomes critical to meet shortened time lines.

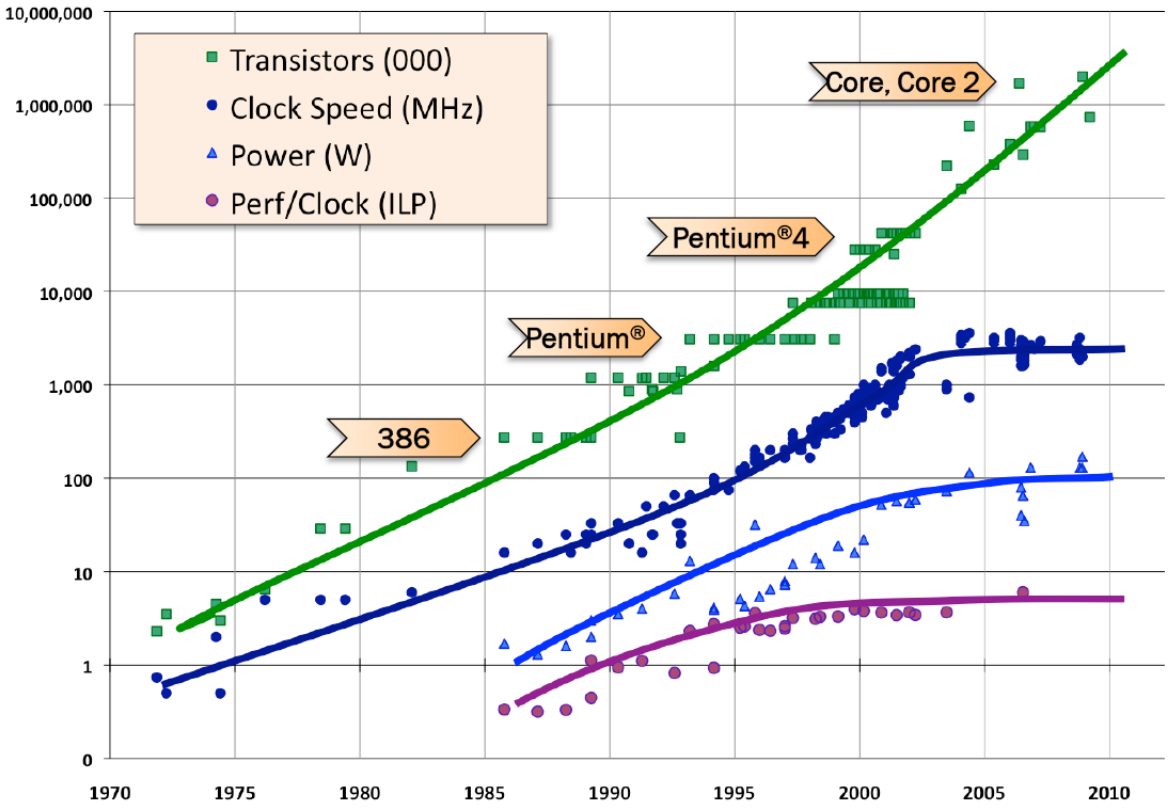


Figure 1.1 Diminishing Performance of CPUs and flattening clock frequencies. Source: H. Sutter, "The Free Lunch is Over: A Fundamental Turn Towards Concurrency in Software". Available online at http://www.gotw.ca/publications/concurrency-ddj.htm. Used under fair use guidelines, 2011.

In recent years, clusters of general-purpose processors (GPPs) were used to perform data intensive tasks by dividing the problem into pieces and distributing them to individual

Industry	Domain	Applications
Government labs	HPSC	Climate modeling, nuclear waste simulation, warfare modeling, disease modeling and research, and aircraft and spacecraft Modeling
Geosciences and engineering	HPSC	Seismic modeling and analysis, and reservoir simulation
Life sciences	HPSC	Gene encoding and matching, and drug modeling and discovery
Defense	HPSC	Video, audio, and data mining and analysis for threat monitoring, pattern matching, and image analysis for target recognition
	HPEC	Beam forming in radar
Financial services	HPSC	Options valuation and risk analysis of assets
	HPEC	Low latency and high throughput data processing in trading solutions
Airborne Electronics	HPEC	Image compression and analysis in payload
Communications	HPEC	Encryption in network routers
Medical Imaging	HPEC	Image rendering in CT and MRI scanners

Table 1.1 Typical applications for server and embedded computing. Source: P. Sundararajan, "High Performance Computing Using FPGAs" Technical Report, 2010. Available online at http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Usi-ng_FPGAs.pdf.

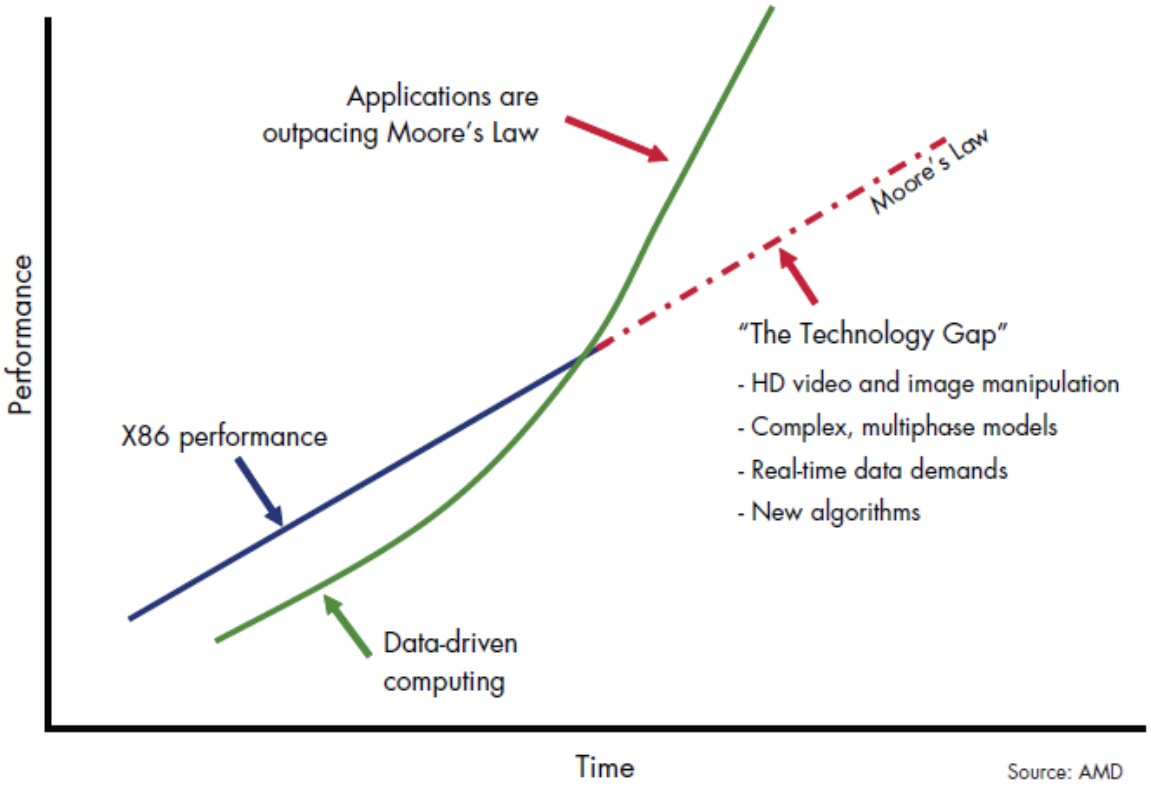


Figure 1.2 Technology gap between CPU performance and HPC application demands. Source: B. Mackin, "Accelerating High-Performance Computing With FPGAs", Altera, White Paper, 2007. Available online at http://www.altera.com/literature/wp/wp-01029.pdf. Used under fair use guidelines, 2011.

processors [1] [2]. One of the most successful use of cluster computing is the Google cluster [3]. Google's web search queries are run on more than 15,000 commodity class CPUs. Google has several clusters located in various locations around the globe. When a search request arrives, it is routed to a cluster closest to the origin of the query. The query is then matched against an index constructed from Google's web crawlers. The results returned from the index search are passed to the document servers, which give the matched results.

However, the computational complexity of current applications in HPC have outpaced the capacity of the GPP to deliver performance despite constant processor improvements, creating a technology gap as seen in Figure 1.2.

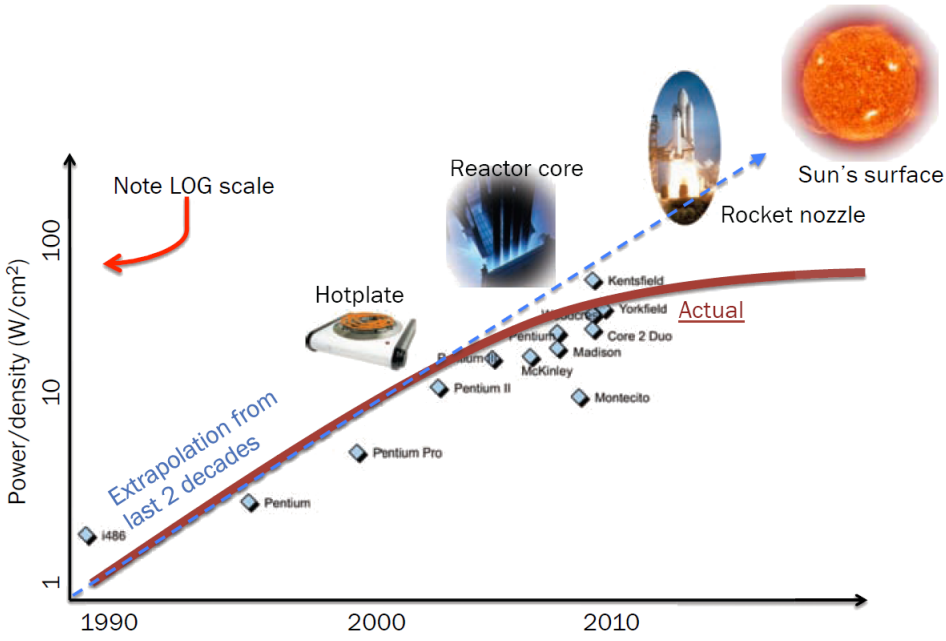


Figure 1.3 CPU Cluster Power Density Wall. Source: F. Pollack, "New Micro-architecture Challenges in the Coming Generations of CMOS Process Technologies", Intel. Available online at http://research.ac.upc.edu/HPCseminar/SEM9900/Pollack1.pdf. Used under fair use guidelines, 2011.

The technique of simply scaling a processor's frequency according to Moore's law for increased performance has run its course due to power dissipation escalating to impractical levels. As

Figure 1.1 depicts, despite transistors being almost free, power is not. In the early years of 2000, this power wall resulted in increasing the number of cores on a single integrated circuit while maintaining clock frequencies, which affected performance and available instruction-level parallelism. Figure 1.3 shows the implication of increasing clock frequencies on the power density per unit area of the chip.

OLD	**NEW**
Power free, **transistors** expensive	**Power** expensive, transistors free (**Power Wall**)
Mainly **Dynamic** Power	**Static** Power large
Reliable, only **pin errors**	Small feature sizes, high **soft and hard errors**
Multiply slow, load/store fast	Multiply fast, **load/store** slow (**Memory Wall**)
Out-of-order + VLIW =**ILP**	ILP returns diminishing (**ILP Wall**)
Performance 2X in 18 months (**Moore's Law**)	**Power Wall** + **Memory Wall** + **ILP Wall**= **Brick Wall**, 2X in 5 years
Performance α **frequency**	Performance α hardware **parallelism**

Table 1.2 Comparing old and new computational problems. ILP here refers to instruction level parallelism.

Table 1.2 succinctly details the problems that have surfaced in recent years. The need to overcome the "brick wall" has given rise to the notion of co-processors and augmenting a CPUs capabilities to perform certain tasks by offloading the most data and compute intensive portions of an application to dedicated hardware ranging from GPUs [4], FPGAs [5] to DSPs [6] and special ASICs [7]. The aim is to deliver unprecedented levels of performance

by extracting the parallelism inherent in HPC workloads at a much lower power using a slower clock.

1.1 Motivation

This thesis compares FPGA and GPU based accelerators; primarily because they have been shown to have great potential in meeting the demands of current and future HPC applications. GPUs, traditionally used for graphical operations, have become attractive to high performance scientific applications due to the combination of high floating-point performance, ease of programming and superior price-performance ratio enabled by large-scale deployment in desktops and in laptops. Their emergence as an attractive high-performance computing platform is also evident from the fact that three out of the top five fastest supercomputers in the Top500 list employ them as accelerators [8]. Also Cray, a long standing supercomputer manufacturer uses NVIDIA Tesla X2090 GPUs in their latest Cray XK6 product line [9].

FPGAs on the other hand, provide massive parallelism and have the flexibility to be tuned to meet the specific needs of an application without the cost or delay of designing a custom coprocessor. In the past, FPGAs were restricted to a narrow set of HPC applications because of their relatively high cost. Over time, however, advancements in process technology have enabled them to have high compute densities and shrinking costs. Architectural enhancements, increased logic cell count, and speed have contributed to an increase in FPGA logic computational performance [5]. For instance, with an average 25% improvement in typical clock frequency for each FPGA generation, the logic compute performance (product of clock frequency increase and logic cell count increase) has improved approximately by 92X over the past decade while the cost of FPGAs has decreased by 90% in the same time period as seen in Figure 1.4. The flexibility in achieving high performance per watt coupled with lower

costs is helping FPGAs make move into the HPC space.

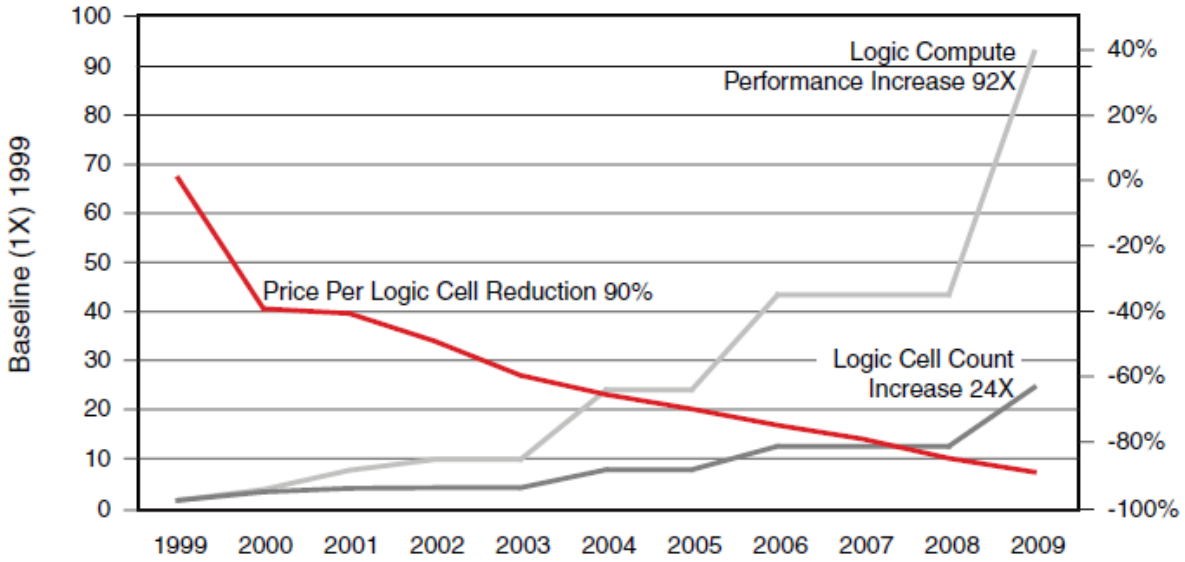


Figure 1.4 Advancement of FPGA density, speed, and reduction in price over time. Source: P. Sundararajan, "High Performance Computing Using FPGAs" Technical Report, 2010. Available online at http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Usi-ng_FPGAs.pdf. Used under fair use guidelines, 2011.

Productivity is another aspect concerning HPC, and can be as important as performance itself. Factors such as the level of programming abstraction translate to ease of programming, scalability, portability and re-usability. In 2004, DARPA replaced the term "performance" with "productivity" and coined the nomenclature High-Productivity Computing Systems (HPCS) [10]. Instead of comparing solutions with their different execution times, it was proposed to use the "time to solution", a measure that includes the time to develop the solution as well as the time to execute it. This thesis throws some light on productivity aspects of FPGA- and GPU-based accelerators and challenges that face FPGAs in HPC.

The need for low power directly impacts both space availability and cost of maintenance. Power reduction translates to lower cost of cooling to keep the space within appropriate temperature limits or the ability to put more computing power within that space. Despite

the importance of price, this thesis focuses on the technical aspects of this computational space; hence, the economic factors are not analysed in the work.

1.1.1 Recent FPGA successes in HPC, June 2011

FPGA's recent popularity in HPC can be exemplified by the fact that the world's largest financial institution, *J.P Morgan Chase* recently invested in a FPGA-based supercomputer to perform real-time risk-analysis of some of its largest and most complex credit derivative portfolios [11]. During the time of recession of 2007-2009, the credit evaluation of the company's underlying assets, mostly mortgages was plummeting. Part of the problem was that analysing the credit portfolio took eight to twelve hours to complete; an overnight run requiring a cluster of thousands of x86 cores. More than complexity, they didn't assess the risk parameters of the various mortgages correctly. Initially, J P Morgan looked at GPUs for acceleration. Porting one of their models to GPUs delivered a respectable 14-to 15-fold performance speedup. With FPGA expertise from Maxeler Technologies [12], a 40-node hybrid HPC cluster was built, each node hosting eight Xeon cores communicating with two Xilinx Virtex-5 (SX240T) FPGAs via PCIe links. After redesigning some of the C++ code to expose parallelism and porting time-critical, compute-intensive section to the new hardware, the execution time shrunk to 238 seconds, roughly a 120X improvement. Of the 238 seconds, the FPGA time was 12 seconds. Fine grained parallelism and deep pipelining coupled with low clock rates delivered an astounding power efficiency [13].

Another trend for FPGA-based supercomputing can be gauged from the fact that just recently, Convey Corporation's HC-1^{ex} [14] (that supersedes Convey HC-1) joined the ranks of the world's most powerful computer systems for data-intensive computing, securing the 25th spot on the Graph500 benchmark [15, 16]. The Graph500 is a set of large-scale benchmarks

for data-intensive applications that have been established by a consortium of over 30 international HPC experts from academia, industry, and national laboratories. The Graph500 gets its name from graph-type problems; algorithms that are a core part of many analytics workloads in applications such as cyber-security, medical informatics, data enrichment, social networks, and symbolic networks. The benchmark code involves a breadth-first search algorithm. The ranking is based on problem size first and then performance. The performance of a *single Convey HC-1^{ex} node* was 773 million traversed edges per second on a problem of scale 27, competitive with large clusters. The scale factor refers to the logarithm base two of the number of vertices in the graph. Lower memory capacity in the Convey-HC1^{ex} compared to large clusters, limits a larger problem size, since each increase in problem size doubles memory footprint.

1.2 Contributions

This thesis provides a multidimensional understanding of how FPGA and GPU based accelerators compare with respect to performance, power, memory bandwidth, productivity, or some combination of these metrics for computationally challenging algorithms such as FFT and molecular dynamics. Ideal performance calculated by "paper-and-pencil" studies using device datasheets give a good idea of device capability and peak theoretical performance. This type of calculation can be useful for raw comparison between devices, but is somewhat misleading as it assumes that the datasets are always available to feed the device, and does not take into account memory interfaces and latencies, place and route constraints, and other aspects of an actual FPGA design. Sustained performance under a workload however, ultimately determines how well the accelerator performs in real-world applications. Application performance on these heterogeneous accelerators is highly dependent upon a balance

of memory volume and bandwidth, input/output bandwidth and the availability of specific compute resources. Results show that for floating-point performance, particularly with respect to FFTs, GPUs fare much better primarily due to immense floating-point capability and high memory bandwidth. The work also reports much better performance over previous efforts to extract floating-point performance from the Convey Hybrid Core computer via a hand-tuned optimized implementation. Mapping the molecular dynamics application to the Convey HC-1 shows a 200-fold improvement over CPUs and a 2-fold speedup over a GPU implementation. The thesis adds value to existing work in this area by analyzing some of the reasons for the performance discrepancy between GPUs and FPGAs, that can help future design of FPGA-based accelerators for HPC.

The results of this thesis does not prove FPGA superiority in HPC. While it is perceived that there is an on-going battle between FPGAs and GPUs for HPC dominance, there's actually plenty of daylight between the two architectures, suggesting that different classes of applications would gravitate towards one or the other. This leads to a possibility of a future heterogeneous system using the specialization of FPGA, the parallelism of GPU, and the scalability of CPU clusters to optimized speed, cost and energy [17]. The work in this thesis would then be beneficial in determining the role of FPGAs, in such converged heterogeneous environment.

1.3 Thesis Organization

The next chapter of this thesis gives some history on FPGA-based supercomputers and cites related efforts to compare them with other accelerator technologies. Chapter 3 gives an overview of the platforms used in the study, and discusses some assumptions that were made. Chapter 4 details the approach for the analysis and the mapping on the accelerators.

Chapter 5 gives a quantitative and qualitative discussion of the results of the implementation on the accelerators. Finally, Chapter 6 will summarize the work, highlighting key limitations facing FPGAs in HPC, and proposing future work.

Chapter 2

Background and Related Work

2.1 History of FPGAs in HPC

The true ingenuity of FPGAs lies in the fact that they can be used to time-multiplex different tasks without the expense and risk involved in custom ASIC development. These *custom-computing machines* meet the true goal of parallel processing; executing algorithms in circuitry with the inherent parallelism of hardware, avoiding the instruction fetch and load/store bottlenecks of traditional Von Neumann architectures.

The concept of high-performance reconfigurable computing using FPGAs as co-processors has been around since the 1960s. It was originally credited to Gerald Estrin [18], who described described a hybrid computer structure consisting of an array of reconfigurable processing elements. His aim was to combine the flexibility of software with the speed of hardware, with a main processor controlling the behavior of the reconfigurable hardware. However, this technology was not realizable during that time due to limitations of the electronics technology. Sometime around the beginning of 1990, one of the world's first recon-

figurable system was born, the Algotronix CHS2x4 [19]. This was an array of CAL1024 processors with up to 8 FPGAs, each with 1024 programmable cells.

Another significant contribution to the use of FPGA-based co-processing traces back to the days of the *Splash*, developed at the Supercomputing Research Center (SRC) by Gokahle et al. Its primary purpose was to help study the Human Genome [20] [21] [22], but soon found itself favorable to other important applications.

2.1.1 The Splash System

The Splash consisted of two boards; the first containing the linear array and the second containing a dual-ported memory card. The two boards resided in two Versa-Modular Eurocard (VME) bus slots of a Sun workstation as detailed in Figure 2.1. The Splash logic-array board held 32 Xilinx XC3090 FPGAs and 32 SRAM memory chips running at a maximum frequency of 32 MHz. Two additional Xilinx chips were used for bus control. The logic array card connected to the Sun VME bus for control and the Sun VME Sub-system Bus (VSB) for data I/O. The associated dual-ported memory card was connected to the Sun VME bus for data initialization and retrieval, and to the Sun VSB bus for data I/O, to and from the logic-array.

2.1.2 Programming the Splash

The Splash was programmed by specifying the logic functions and interconnections of each of the 320 configurable logic blocks (CLBs) and 144 Input/Output blocks (IOBs) on each of the 32 chips. A combination of gate-level Logic Description Generator (LDG) language for programming and the trigger symbolic debugger for debugging (both tools developed at SRC), permitted rapid design turnaround times comparable to software design.

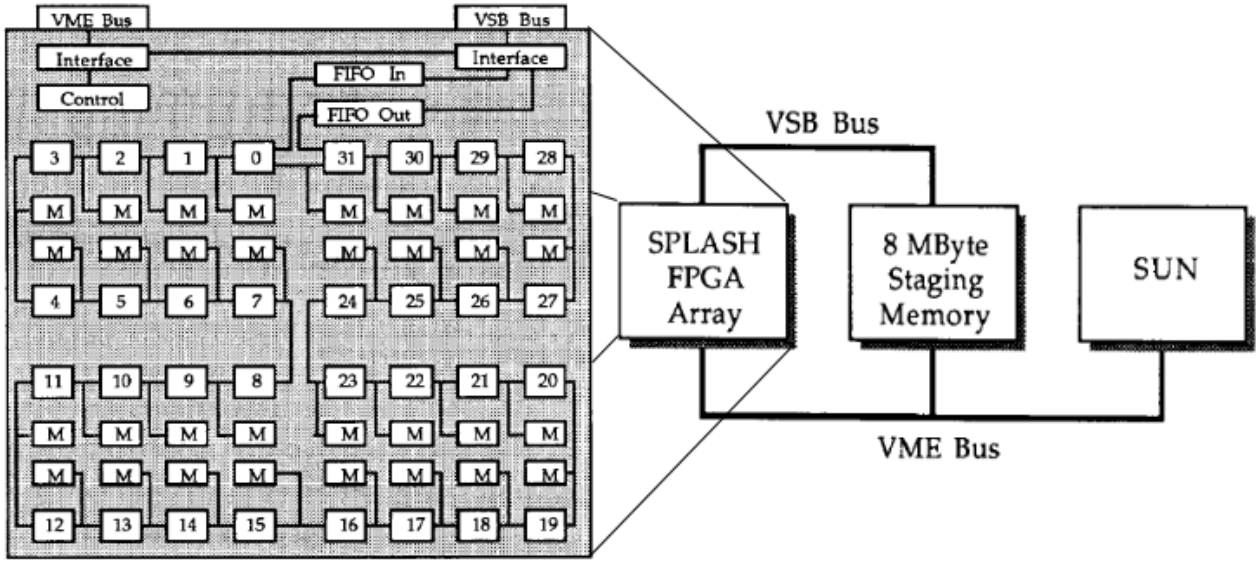


Figure 2.1 The Splash system. Source: T. Waugh, "Field Programmable Gate Array Key to Reconfigurable Array Outperforming supercomputers", In the IEEE Proceedings of Custom Integrated Circuits Conference, 1991, May 1991, pp. 6.6/1-6.6/4. Used under fair use guidelines, 2011.

The Splash architecture was suited to applications requiring *non-floating point* one-dimensional vectors, such as pattern matching, speech recognition, data retrieval and genetic analysis. A basic sequence matching algorithm resulted in a 2,700-fold speed-up over a VAX system from Digital Equipment Corporation (DEC) and 325 times faster than the CRAY-2 supercomputer [22]. A two-dimensional implementation with similar FPGAs was built by DEC Paris Research Lab [23].

The Splash-2 was the second generation processor from SRC to aid in creating arbitrary topologies. It was based on Xilinx XC4010 FPGA devices and introduced a crossbar switch network to overcome the low I/O bandwidth of the VME bus [24]. It also had larger memory on-board suited for more complex applications. Many companies used the Splash-2 architecture for their own custom application-specific boards. One example is the Wildfire heterogeneous system [25] from Annapolis Micro Systems. The Splash-2 achieved a perfor-

mance of 180 Million Floating-Point Operations Per Second (MFLOPS) [26].

Since the splash there have been a plethora of reconfigurable architectures spawned from both industry and academia [27] [28] [29].

2.2 FPGA configurations for HPC

2.2.1 Massive Parallelism in FPGA architecture

The SRAM-based FPGA architecture varies slightly across vendors, but the basic architecture layout is the same as shown in Figure 2.2. It comprises of an array of logic blocks, memory elements, DSP units and is surrounded by programmable interconnects that can be configured by an algorithm description in a Hardware Description Language (HDL) such as Verilog or VHDL. This architecture provides the following capabilities:

- *Functional and bit-level parallelism:* Replication of varied bit-width function units as a result of ample logic resources.
- *Data parallelism:* Handling of data arrays or data matrices.
- *Custom instruction parallelism with pipelining:* Stream data into functional units separated with registers to obtain results every clock cycle.
- *Huge primary cache size and bandwidth:* Block RAMs (BRAMs) provide large capacity with internal bandwidth to move the operands and results of application-specific arithmetic units in the order of Tera Bytes/sec (TB/s).
- *Flexible routing of datapath:* Huge crossbar routing transferring data in one clock cycle.

- *Custom off-chip I/Os:* Protocol, bandwidth, and latency flexibility as per needs of the application and budget. For a memory interface, FPGAs can provide support for DDR, DDR2, DDR3, RLDRAM and QDR SRAM [30] [31].

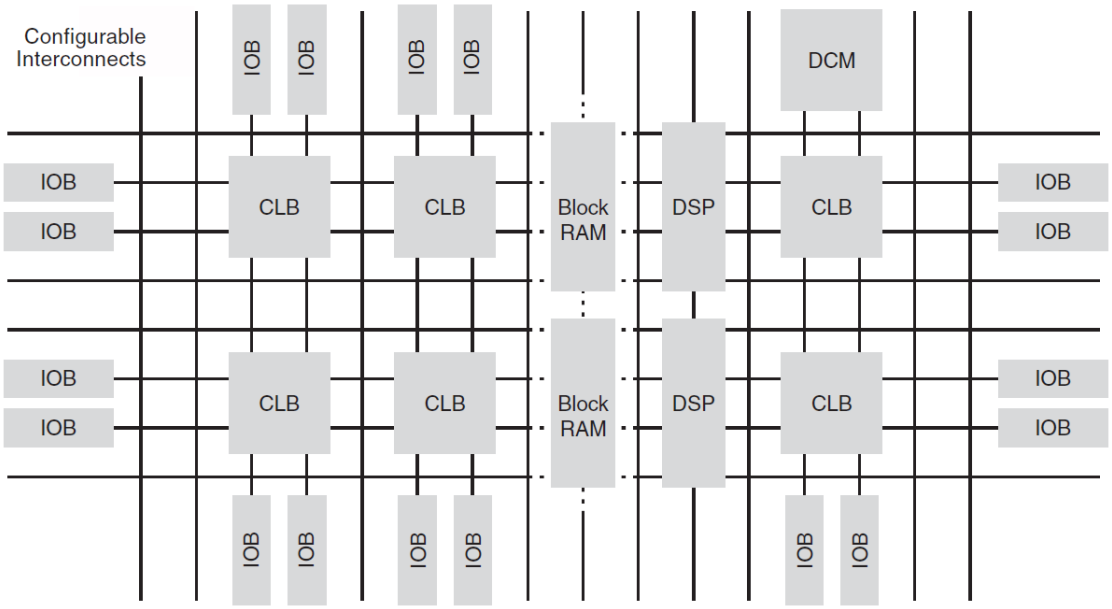


Figure 2.2 High-Level Architecture of FPGAs.

The massive fine-grained parallelism (due to pipelining) enables them to run at low frequencies; as low as 200 MHz (approx 5X slower compared to GPUs and 12X slower than CPUs) due to which they consume a maximum power of 30 W. This offers high power density for energy-constrained environments.

2.2.2 FPGA use model in HPC

Regardless of the HPC application, FPGA-based HPC platforms need to support certain infrastructure. This includes an interface to the host processor, on-board memory and a mechanism to interconnect multiple FPGAs on the same board. This is where commercial off-the-shelf vendors provide users with a design pre-configured with platform infrastructure

elements; helping HPC users to focus on application development rather than the housekeeping. Figure 2.3 illustrates the most common use-model for FPGAs in HPC, which involves accelerating portions of software that are compute intensive and more suited to hardware. Profiling is an important phase early in development that helps making estimates of timelines. This partitioning of data is also critical to achieving end-to-end acceleration of the application.

2.2.3 FPGA interconnect

In 2006, AMD opened its HyperTransport interface via the Torrenza initiative closely followed by Intel, who licensed their Front Side Bus (FSB) as a part of their Geneseo program, to make FPGA co-processing a practical endeavour. This incited companies like XtremeData, DRC Computer, Nallatech and Convey Computer to develop FPGA expansion modules/boards for x86-based platforms [32] [33] [34] [35]. Prior to this, only custom RASC (Reconfigurable Application Specific Computing) technology from Silicon Graphics Inc. (SGI) and Cray's XD1 system were possible [36] [37]. These proprietary means were extremely costly and limited the use of FPGAs in HPC.

The ability to plug FPGAs into the x86 system-bus made it not only cheaper, but kept the data coherent with the processor memory space compared to PCIe-based accelerators. For instance, GPU accelerators are usually PCIe-based and cannot keep the data coherent with the processor memory space. This fact has important implications on both, the type of applications that can be accelerated as well as on accelerator system's programming model. This not only meant lower latency to host memory, but FPGAs could now operate without host intervention.

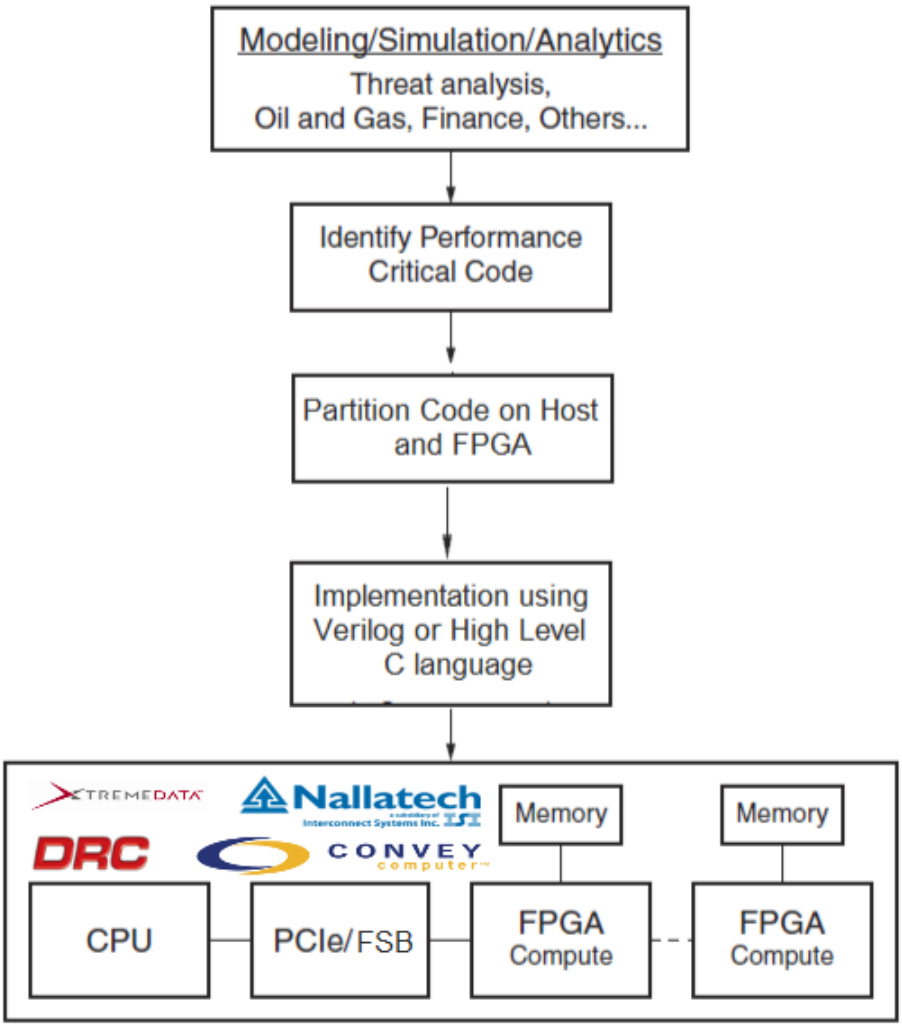


Figure 2.3 FPGA HPC use model. Source: P. Sundararajan, "High Performance Computing Using FPGAs" Technical Report, 2010. Available online at http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Usi-ing_FPGAs.pdf. Used under fair use guidelines, 2011.

2.3 Related Work

Many modern devices ranging from x86 processors to GPUs to FPGAs have been compared by their raw computation density, power consumption, I/O bandwidth and memory bandwidth [38]. While these peak numbers are good to understand qualitative trends, an actual running application gives better quantitative insight and permits better assessment of these advanced architectures. Govindaraju et al. have analyzed GPU memory system behavior by using an FFT as the algorithm for evaluation [39]. A GeForce 7900 GTX performed a 1 M sample FFT in 19 ms and provided a 2X improvement over an Intel processor. An equivalent Virtex-4 FPGA implementation with a Sundance floating-point FFT core, operating at 200 MHz, performed a 1 M sample FFT in 21 ms. The GPU was demonstrated in this case to be superior.

Baker et al. [40] compared device cost, power, and performance of a CPU, GPU and FPGA. It was concluded that the GPU is superior in terms of price per unit speed-up in performance, while an FPGA provided better speed-up per unit of energy. These results suggest that GPUs are beneficial for supercomputing systems while FPGAs could benefit embedded systems more. Owens et al. [41] surveyed the use of GPUs for general-purpose applications. Memory reduction operations and the lack of memory scatter capability are cited as weaknesses of GPUs.

Ben et al. compared GPUs and FPGAs using five different benchmark algorithms [42]. It was found that the FPGAs were superior over a GPUs for algorithms requiring a large number of regular memory accesses, while GPUs were better suited to algorithms with variable data reuse. It was concluded that FPGAs with a customized datapath outperform GPUs at the expense of larger programming effort. Reduced bit-width floating-point units for FPGAs resulted in much higher speed-ups compared to a full floating-point implementation for a

face detection algorithm studied by Yingsoon et al [43]. While still retaining 94% of face detection accuracy, a significant reduction in area utilization by 50% and power by 15%, was recorded. Likewise, Clapp et al [44] showed that using customized number representations can significantly improve performance upto 48 times, reduce area cost and I/O bandwidth in seismic applications, provided the reduced precision provided acceptable seismic accuracy.

Brahim et al. performed extensive comparison of the NVIDIA Tesla C1060 GPU and the Convey Hybrid Core (HC-1) FPGA system using four benchmarks having different computational densities and memory locality characteristics [45]. It was shown that the Convey HC-1 had superior performance and energy efficiency for the FFT and a Monte Carlo simulation. The GPU in [45] achieved a meagre 25 Giga Floating-point Operations Per Second (GFLOPS) for the 1D in-place FFT, possibly due to the unoptimized mapping of NVIDIA's CUDA FFT on the NVIDIA Tesla. The work done by Brahim et al [45] concluded that for applications requiring random memory accesses, the Convey HC-1 with its optimized memory system for non-sequential memory accesses performs better than GPUs, which incur a higher performance penalty for non-contiguous memory block transfers.

A FPGA-based molecular dynamics simulation performed by Guo et al. [46] resulted in a substantial speed-up over a production software. However, it used fixed-point arithmetic to improve scalability. The molecular dynamics implementation in this work uses a single-precision floating-point format, which gives better accuracy, making it computationally intensive to implement on the FPGA.

2.4 Summary

This chapter introduced how the notion of FPGA-based high-performance computing was conceived with the Splash system. It then explained the basic functionality offered by the

FPGA fabric and its most common use model to accelerate software. Finally, some related work is presented that establishes a common ground for comparing results obtained in this thesis, and how this work adds value to the existing base of knowledge, that can help design of future FPGA-based accelerators.

Chapter 3

Architecture Background

This thesis compares two generations of promising accelerators built from FPGAs and GPUs, based on technology size and time of availability as shown in Figure 3.1, including metrics for analysis. The Convey HC-1 node installed at the Bioinformatics Institute at Virginia Tech (VBI) served as the representative Virtex-5 platform [47]. A Pico EX-500 PCIe card (housing the M-503 module) served as a representative Xilinx Virtex-6 platform. Convey HC-1 is the high-performance, high-power server system while Pico M-503 is the low-power PCIe-based system. Similarly, a NVIDIA Tesla C2050 PCIe card based on the latest generation Fermi architecture with large number of compute cores was used in addition to previous a generation NVIDIA GTX 280 PCIe card based on the G80 architecture [48] [49]. This chapter serves to introduce these advanced accelerators with respect to their hardware and programming models, which will help in understanding their mapping and optimization in subsequent chapters.

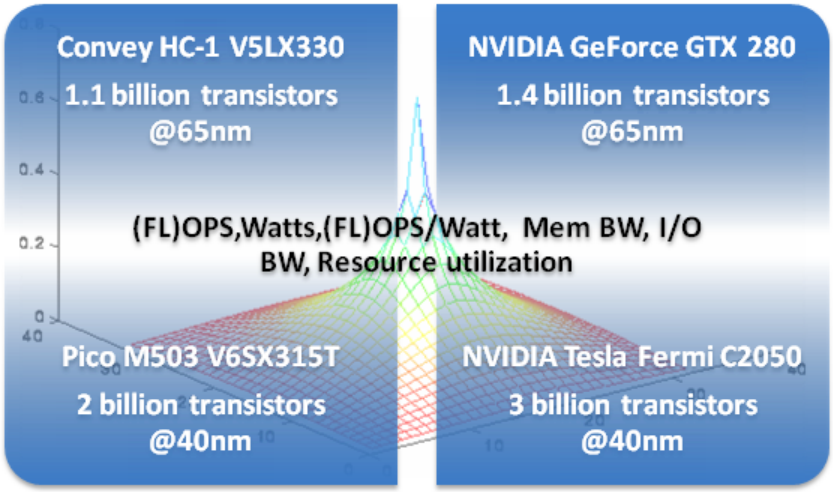


Figure 3.1 Mix of accelerators and corresponding metrics to be evaluated.

3.1 The Convey HC-1 Hybrid Core

A hybrid-core computer improves application performance by combining an x86 processor with hardware that implements application-specific instructions [50]. The Convey HC-1 is one such hybrid-core system that uses a commodity two-socket motherboard to combine a reconfigurable, FPGA-based co-processor with an industry standard Intel Xeon processor. Unlike the in-socket co-processors from Nallatech, DRC and XtremeData [34] [33] [32]; all of which are confined to socked-sized footprints, Convey uses a *mezzanine* connector to extend the Front Side Bus (FSB) interface to a large co-processor board. The assembly is shown in Figure 3.2.

3.1.1 Convey HC-1 System Architecture

The system architecture is shown in Figure 3.3. The co-processor supports multiple instruction sets (referred to as "personalities"), which are optimized for different workloads and dynamically reloaded at application run-time. Each personality includes a base set of

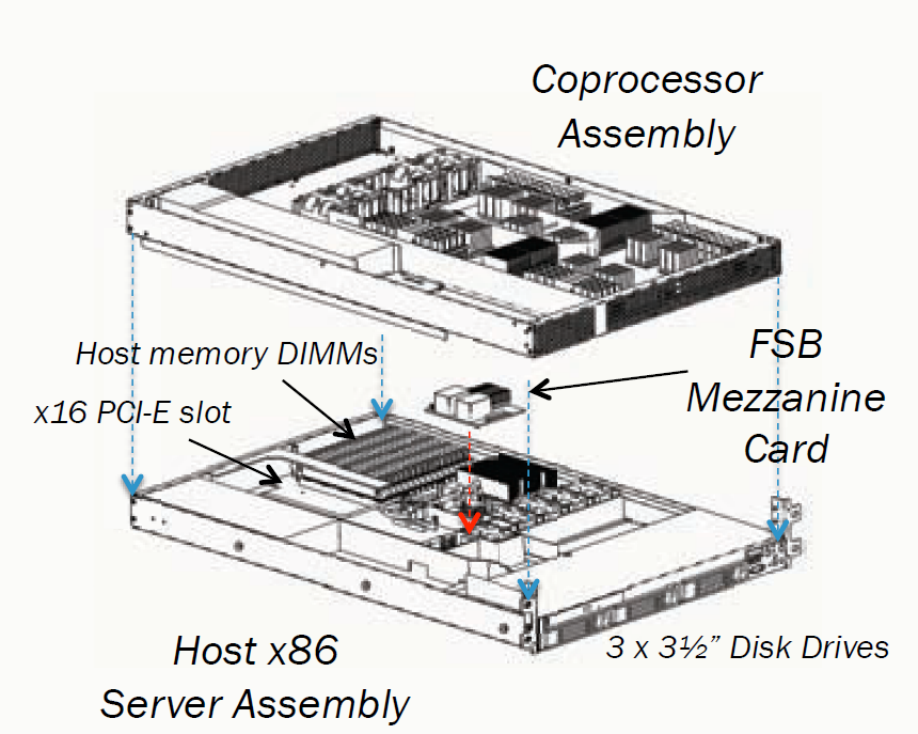


Figure 3.2 Convey HC-1's 2U enclosure. Source: T. M. Brewer, "Instruction Set Innovations for Convey's HC-1 Computer", Presentation at Hot Chips Conference, 2009. Available online at http://www.hotchips.org/archives/hc21/3_tues/HC21.25.500.ComputingAccelerators-Epub/HC21.25.526.Brewer-Convey-HC1-Instruction-Set.pdf. Used under fair use guidelines, 2011.

instructions that are common to all personalities, as well as extended instructions that are designed for a particular workload. The HC-1 host runs a 64-bit 2.6.18 Linux kernel with a Convey modified virtual memory system to be memory-coherent with the co-processor board memory.

There are four user-programmable Virtex-5 XC5VLX330 compute FPGAs, which Convey refers to as *Application Engines*(AEs). Convey refers to a particular configuration of these FPGAs as a "personality". The four AEs each connect to eight memory controllers through a full crossbar. Each memory controller is implemented on its own FPGA and can connect to two standard DDR2 DIMMs or two Convey-designed scatter-gather memory modules (SG-DIMMs), containing 64 banks each and an integrated Stratix-2 FPGA [51]. The AEs themselves are interconnected in a ring configuration with 668 MB/s, full duplex links for AE-to-AE communication. These links are used for multi-FPGA applications and are not used in the implementation in this work.

Each AE has a 2.5 GB/s link to each memory controller, and each SG-DIMM has a 5 GB/s link to its corresponding memory controller. As such, the effective memory bandwidth of the AEs is dependent on their memory access pattern to the eight memory controllers and their two SG-DIMMs. Each AE can achieve a theoretical peak bandwidth of 20 Gbyte/s when striding across eight different memory controllers, but this bandwidth would drop if two other AEs attempt to read from the same set of SG-DIMMs because this would saturate the 5 Gbytes/s DIMM memory controller links [51].

Convey provides two memory addressing modes to partition the co-processor's virtual address space among the SG-DIMMs [52].

- *Binary interleave*, which maps bit-fields of the memory address to a particular controller, DIMM, and bank, and

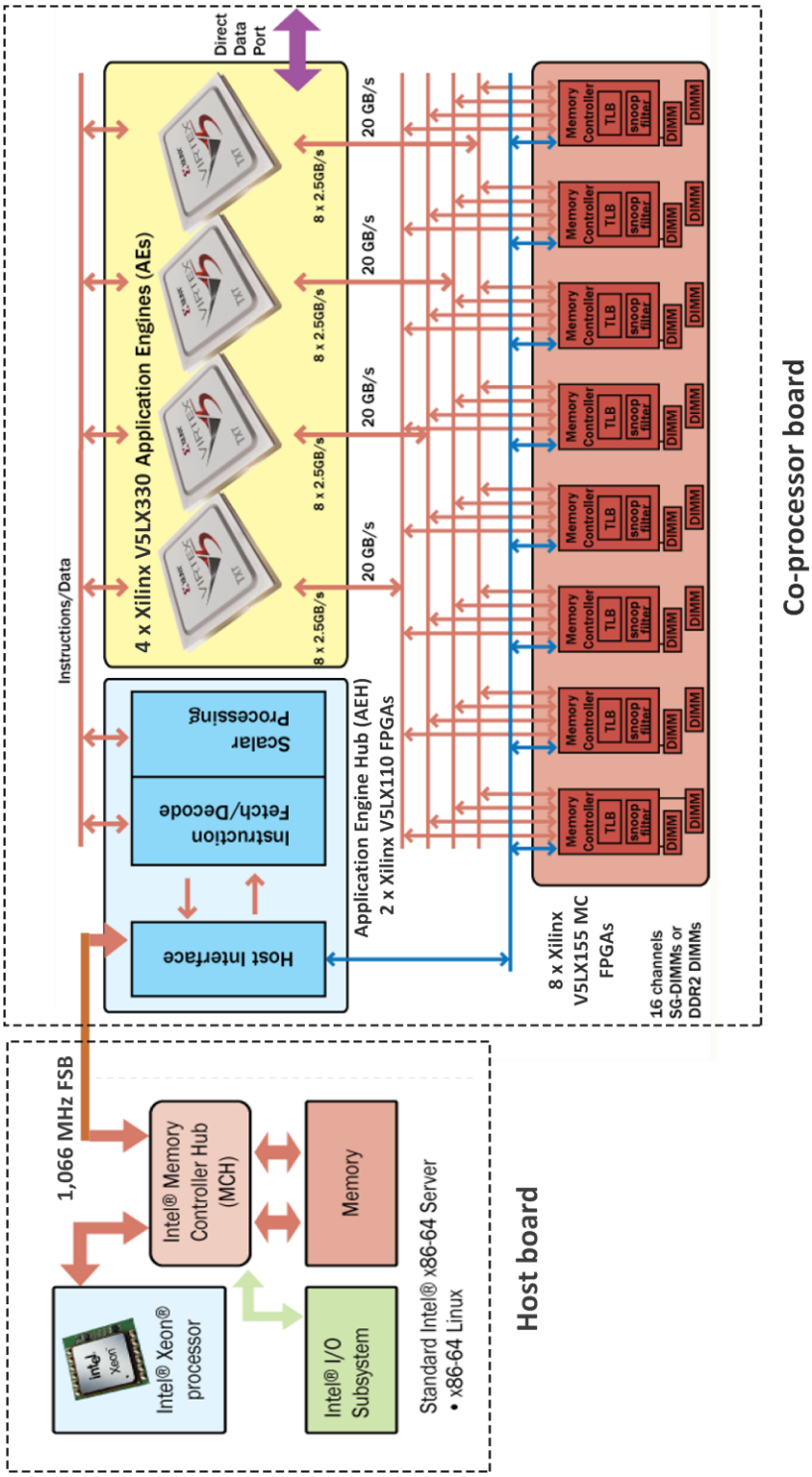


Figure 3.3 Convey HC-1 system block diagram. Source: Convey HC-1 Personality Development Kit Reference Manual, v 4.1, Convey Corporation, 2010, used under fair use guidelines, 2011.

- *31-31 interleave*, a modulo-31 mapping optimized for non-unity memory strides (stride lengths that are a power-of-two are guaranteed to hit all 16 SG-DIMMs for any sequence of 16 consecutive references).

The memory banks are divided into 32 groups of 32 banks each. In 31-31 interleave, one group is not used, and one bank within each of the remaining groups is not used [52]. The number of groups, and banks per group being a prime number, reduces the likelihood of strides aliasing to the same SG-DIMM. Selecting the 31-31 interleave comes at a cost of approximately 1 GB of addressable memory space (6%) and a 6 percent reduction in peak memory bandwidth.

The co-processor board contains two non-user programmable FPGAs, that together form the Application Engine Hub. One FPGA implements the physical interface between the co-processor board and the FSB. The second FPGA contains the scalar soft-core processor, that implements the base Convey instruction-set. More information on the Convey HC-1 can be found at [35, 50, 52–54].

3.1.2 Convey HC-1 Programming Model

The Convey HC-1 programming model is shown in Figure 3.4. In Convey's programming model, the AEs act as co-processors to the scalar processor, while the scalar processor acts as a co-processor to the host CPU. To facilitate this, the binary executable file on the host contains integrated scalar processor code. This is transferred to and executed on the scalar processor when the host code calls a scalar processor routine through one of Convey's runtime library calls (similar to how NVIDIA GPUs execute code meant for the GPU) [51]. The scalar processor code can contain instructions that are dispatched and executed on the AEs. Applications can be coded in standard C, C++, or Fortran. Performance can then be

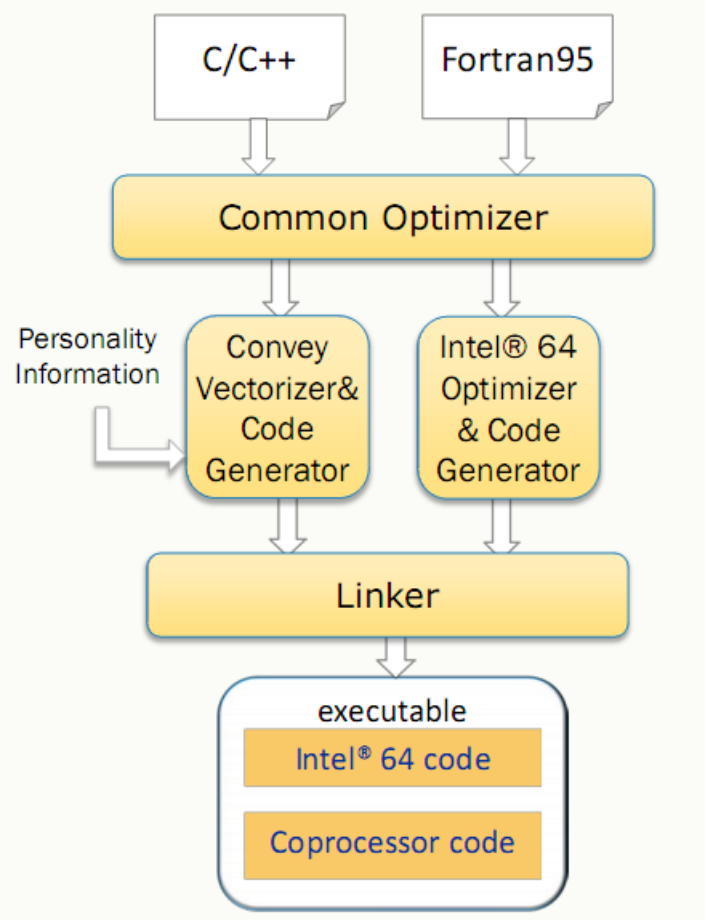


Figure 3.4 Convey HC-1 programming model. Source: T. M. Brewer, "Instruction Set Innovations for Convey's HC-1 Computer", Presentation at Hot Chips Conference, 2009. Available online at http://www.hotchips.org/archives/hc21/3_tues/HC21.25.500.ComputingAccelerators-Epub/HC21.25.526.Brewer-Convey-HC1-Instruction-Set.pdf. Used under fair use guidelines, 2011.

improved by adding directives and pragmas to guide the compiler to performance specific optimizations to the code they encompass. A unified compiler is used to generate both x86 and co-processor instructions, which are integrated into a single executable and can execute on both standard x86 nodes and on the co-processor. The co-processor's extended instruction set is defined by the personality specified when compiling the application.

Seamless debugging environment is also provided for host and co-processor code [54]. A critical design consideration made by Convey was providing a dedicated PCIe link to the Management Processor (MP) on the co-processor board, independent of the application data-path link through FSB. It is this link that configures and monitors the FPGAs. This link also provides visibility, when the AE FPGAs are in bad state and appear hung, a situation that inadvertently happens when debugging custom personalities. Although Convey provides Control and Status Register (CSR) agents to get visibility into the FPGA, using the Chipscope logic analyzer tool was the most convenient way to debug as illustrated in Figure 3.5.

Porting applications to the Convey HC-1 can use one or a combination of the following approaches:

- Use the Convey Mathematical Libraries (CML) [55], which provide a set of functions optimized for the co-processor. They use pre-defined Convey-supplied personalities, for example Convey's CML-based FFT uses the single-precision personality.
- Compile one or more routines with Convey's compiler. This uses the Convey auto-vectorization tool to automatically select and vectorize do/for loops for execution on the co-processor. Directives and pragmas can also be manually inserted in the source code, to explicitly indicate which part of a routine should execute on the co-processor [54]. Recently, there is on-going efforts to using third-party High Level Synthesis (HLS)

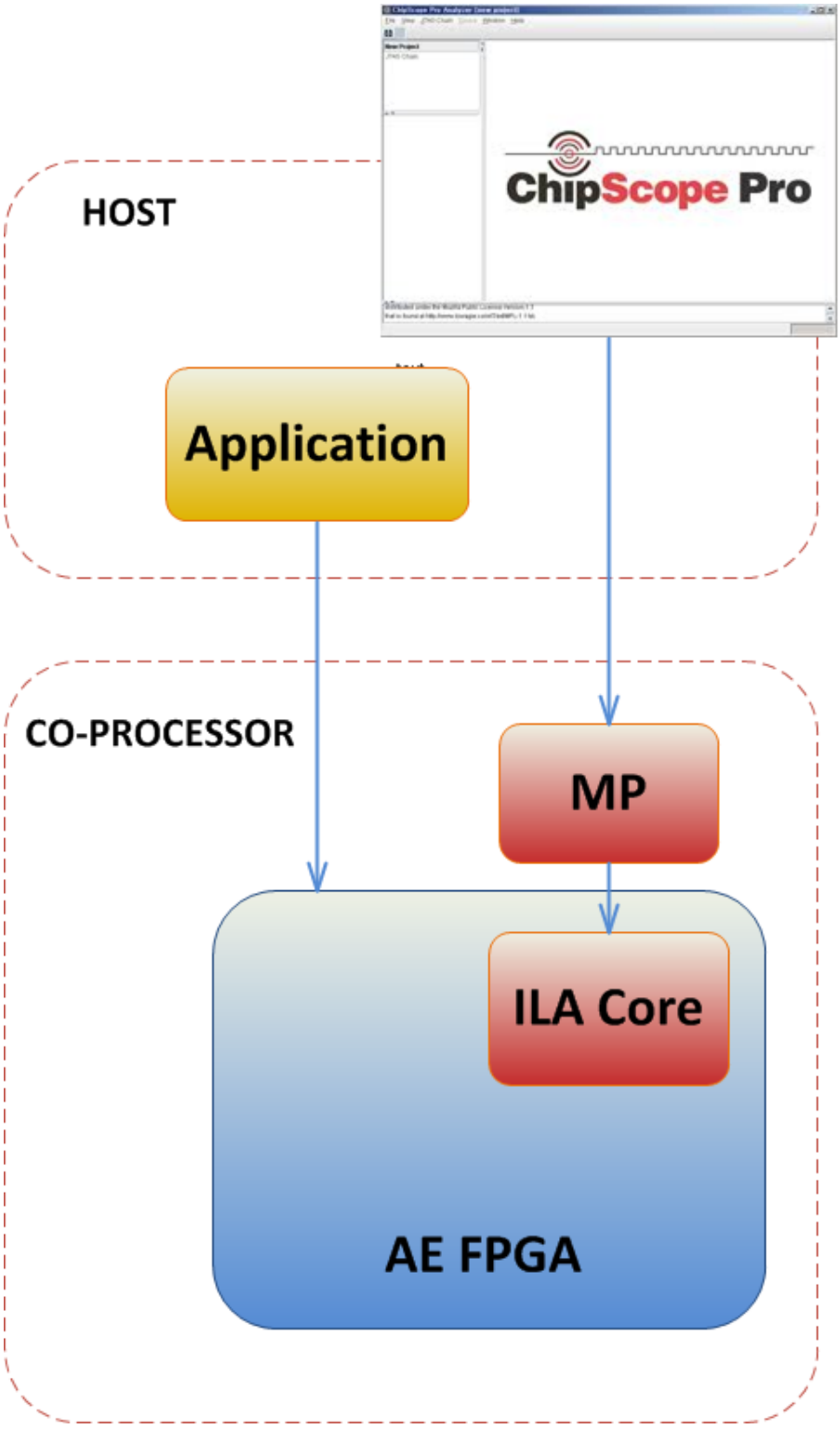


Figure 3.5 Debugging personalities on the Convey HC-1 compute FPGA. Independent visibility path to the compute FPGA through the management processor for debugging.

tools such as ImpulseC and ROCCC to translate a C specified code to HDL [56]. The productivity section in Chapter 5 will address the use of Impulse for Convey HC-1.

- 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 [54].
- Develop a custom personality using Convey's Personality Development Kit (PDK), to give the ultimate in performance using a hardware description language such as Verilog or VHDL.

The Convey Personality Development Kit

The Personality Development Kit is a set of tools and infrastructure that enables development of a custom personality for the Convey HC-1 system. A set of generic instructions and defined machine state in the Convey instruction-set architecture allows the user to define the behavior of the personality. Logic libraries included in the PDK provide the interfaces to the scalar processor, memory controllers and to the management processor for debug as shown in Figure 3.6. The user develops custom logic that connects to these interfaces. Functional verification of the personality is done using the Convey architectural simulator, which allows users to verify the design using the application to drive the simulation [52]. The PDK also includes a tool-flow to ease the process of generating the FPGA image to run on the system. From experience in this thesis, working with the Convey's memory model is much easier than having to co-ordinate with the host to explicitly set-up DMA transfers, greatly simplifying host interfacing.

The Convey PDK provides the following set of features as a part of the kit:

- Makefiles to support simulation and synthesis design flows,

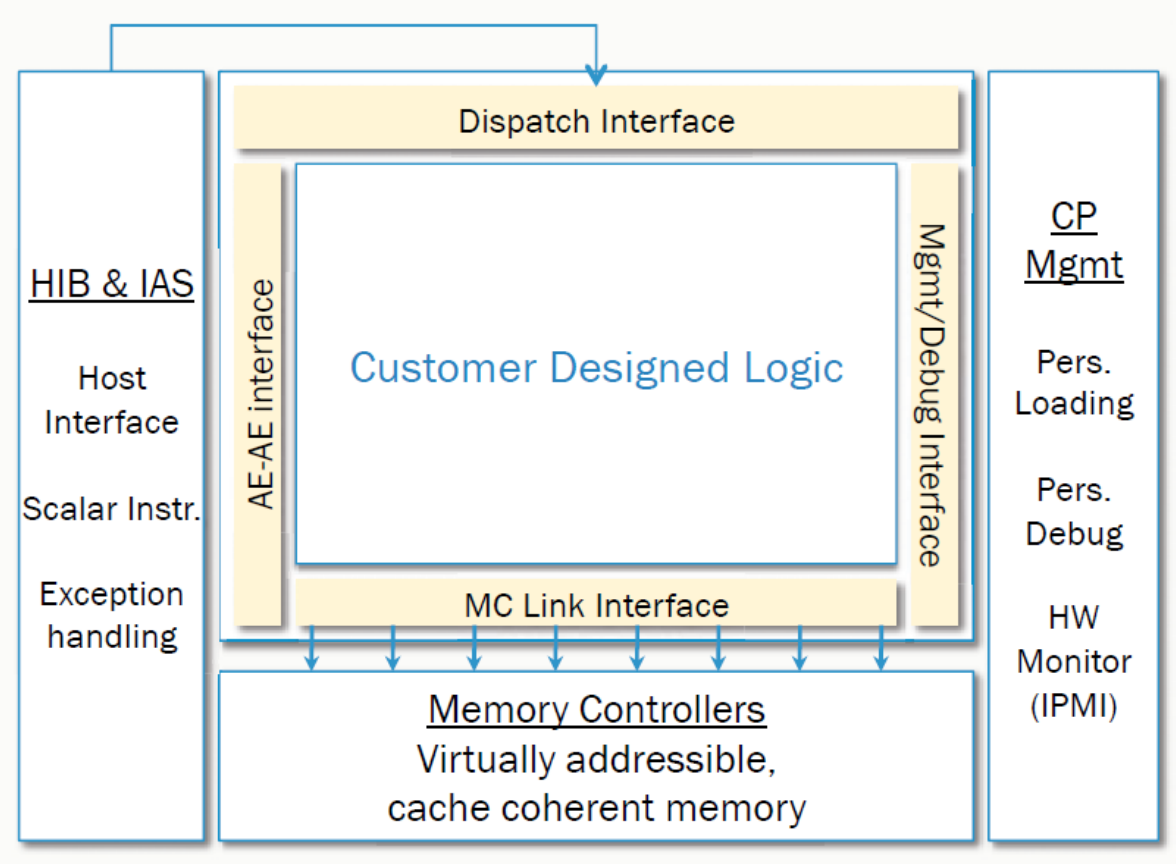


Figure 3.6 Convey HC-1 PDK Architecture. Source: T. M. Brewer, "Instruction Set Innovations for Convey's HC-1 Computer", Presentation at Hot Chips Conference, 2009. Available online at http://www.hotchips.org/archives/hc21/3_tues/HC21.25.500.ComputingAccelerators-Epub/HC21.25.526.Brewer-Convey-HC1-Instruction-Set.pdf. Used under fair use guidelines, 2011.

- Verilog support and interface files,
- Simulation models for all of the co-processor board's non-programmable components (such as the memory controllers and memory modules), and
- A *Programming-Language Interface* (PLI) to let the host code interface with a behavioral HDL simulator such as Modelsim or Synopsys [57, 58].

The PDK's simulation framework is easy to use and allows users to switch between a simulated co-processor mode and an actual co-processor, by changing a single environment variable. Developing with the PDK involves working within a Convey-supplied wrapper that gives the user logic access to instruction dispatches from the scalar processor, access to all eight memory controllers, access to the co-processor's management processor for debugging, and access to the inter-FPGA links. However, these Convey-supplied interfaces require fairly substantial resource overheads: 184 out of the 576 18-KB block random access memory (BRAMS) and approximately 10% of each FPGA's slices. Convey supplies a fixed 150-MHz clock to each FPGA's user logic.

3.2 The Pico M-503

Pico Computing, a vendor that specializes in delivering FPGA-based accelerator cards, supported this thesis by loaning a EX-500 PCIe x16 board capable of housing three Xilin Virtex-6 based M-503 modules. The EX-500 provides electrical and mechanical connectivity to the host system. The M-503 module addresses the ever increasing demands for memory and I/O bandwidth for HPC applications in the bioinformatics, image processing and signal processing domains [59]. The next two sections detail architectural features and the programming model of the Pico M-503.

3.2.1 Pico System Architecture

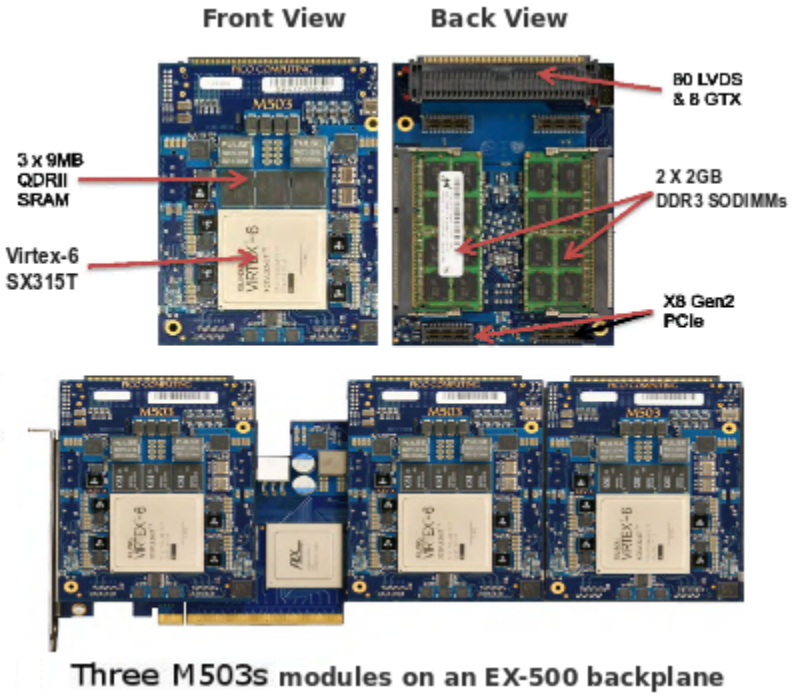


Figure 3.7 Pico M-503 Modules and EX-500 backplane. Source: M-503 Hardware Technical Manual, Rev B, Pico Computing Inc., 150 Nickerson Street Suite, 311, Seattle, WA 98109, used under fair use guidelines, 2011.

The M-503 features a Xilinx Virtex-6 XC6VSX315T FPGA with two independent banks of DDR3 SODIMM providing 15.9 GB/s of sustained local memory bandwidth to the FPGA. In addition to the DDR3, there are three independent QDRII SRAM banks capable of 7.5 GB/s of sustained random access memory bandwidth [60]. There are two x8 Gen2 PCIe links provided via a Xilinx PCIe-endpoint on the Virtex-6. Eighty LVDS and eight GTX transceivers are available via a high speed high density connector to create intermodule connections between multipler M-503 modules or direct access to other external peripherals. The specifications for the M-503 are tabulated in Table 3.1. Figure 3.7 shows the modules and the EX-500 board.

FPGA type	Xilinx Virtex-6 XC6VSX315T
Dimensions	4.9 in x 3.5 in
DRAM	2 X 2 GB DDR3 SODIMM(64-bit @ 533MHz), Peak: 17 GB/s Sustained:15.9 GB/s
SRAM	3 X 9MB QDRII SRAM (18-bit @ 400MHz), Peak: 10.8 GB/s Sustained: 7.5 GB/s
FLASH	4MB
PCIe	x8 Gen2 PCIe
I/O	80 LVDS and 8 GTX Transceivers
Host OS support	Linux (Ubuntu 10.04)
DC Input Voltage	12 V Nominal
DC Input Current	12.5 A @ 12V input voltage
Power Consumption	12 W @ idle
Ambient Temperature	10 deg nominal and 50 deg max

Table 3.1 Pico M-503 module specifications. Source: M-503 Hardware Technical Manual, Rev B, Pico Computing Inc., 150 Nickerson Street Suite, 311, Seattle, WA 98109, used under fair use guidelines, 2011.

3.2.2 Pico Programming Model

Pico provides a set of Application Programming Interfaces (APIs) for the Linux platform. These APIs provide the link between the application software running on the host computer, and the hardware algorithm, or firmware, implemented in the FPGA. The hardware-software interfaces provided by Pico include a number of software library functions and corresponding firmware components enabling FPGA control and processing. These interfaces aim to abstract away the details of communication and provide a platform-portable method of programming the Pico M-503 modules.

PicoClk	250 MHz Free Running Clock
PicoRst	Active High Reset Signal asserted on system power-on and PCIe bus reset
PicoRdAddr[31:0]	32-bit read address that host is reading from
PicoDataOut[127:0]	128-bit data output returned from FPGA user logic at PicoRdAddr
PicoRd	Active high read signal asserted when host is performing a read operation
PicoWrAddr[31:0]	32-bit write address that host is writing to
PicoDataIn[127:0]	128-bit input data being written to FPGA user logic at PicoWrAddr
PicoWr	Active high write signal asserted when host is performing a write operation

Table 3.2 Signals corresponding to the memory mapped PicoBus.

Pico provides a *PicoDrv* class that is the primary point of contact between an application

running on Linux, and the Pico FPGA card. A PicoDrv object encapsulates and abstracts a single Pico card or FPGA device. It handles the configuration of the FPGA, sends and receives data, and gets status information. To communicate between the Pico card and the host PC, a memory-mapped 128-bit synchronous bus model called *PicoBus* is provided. In order to connect to the board, an address range is chosen and then corresponding read and write signals are used to transfer data on the PicoBus. Memory mapped, bus-oriented communications are best for passing data of fixed sizes, ranging from a single-word status message to a large amount of data, representing an entire 2D frame of data for example. This is the technique used in this work. The handshaking signals associated with the PicoBus are listed in Table 3.2.

On the FPGA side, DDR3 memory controllers need to be generated using Xilinx Memory Interface Generator (MIG). The MIG from Xilinx handles the physical interface from the FPGA to the DDR3 providing a easier to use user-interface to the user. A PCIe-endpoint is provided by Pico along with hardware driver code to communicate with the host side PicoBus. The code listing in Appendix A shows how a simple read and write from the hardware side will look like.

3.3 Introduction to GPUs

GPU's massively parallel many-core architecture has landed itself huge market penetration due to deployment in desktops and workstations and gaining a lot of attention in the high-performance computing community due to the low cost and ease of programming [61]. In GPUs, caching and flow control is sacrificed for computations. This implies that GPU computations are essentially free with memory accesses and divergent branching instructions being expensive [62]. Thus, a key requirement to harnessing GPU computational power is

to successfully hide the latency of memory accesses with computations. The GPU does this by performing massive multi-threading. By having thousands of threads simultaneously in-flight, such that, when one thread is awaiting data from memory, other threads perform computations, increases overall utilization and hides memory latency. GPUs are designed to exploit the thread-level parallelism in the single program multiple data paradigm. Threads on GPU are organized in a grid of thread blocks, and all threads of a block run the same instruction. This makes the GPU suitable for applications that exhibit data parallelism, i.e., the operation on one data element is independent of the operations on other data elements.

GPUs are primarily developed by two vendors, AMD and NVIDIA having significantly different architectures. This thesis focuses its comparison with primarily NVIDIA GPUs, hence they are briefly introduced in the next section.

3.3.1 NVIDIA GPU System Architecture

NVIDIA GPUs consist of a large number of compute units known as Streaming Multiprocessors (SMs), and each SM consists of a certain number of scalar streaming processor cores. Each SM, can run up to a thousand threads, thus realizing a massively parallel architecture. The minimum unit of execution on a NVIDIA GPU is called a warp, which is a group of 32 threads. If the threads of a warp diverge and follow different execution paths, then the execution of these threads would be serialized, thereby increasing execution time. Therefore, optimal performance is obtained when all the threads in a warp execute the same instruction.

On the NVIDIA GT200 architecture, each SM has on-chip shared memory. The shared memory enables extensive reuse of on-chip data, thereby greatly reducing off-chip traffic and improving application performance. On the latest NVIDIA Fermi architecture, each SM owns a configurable on-chip memory that can act as either shared memory or as a L1 cache. The

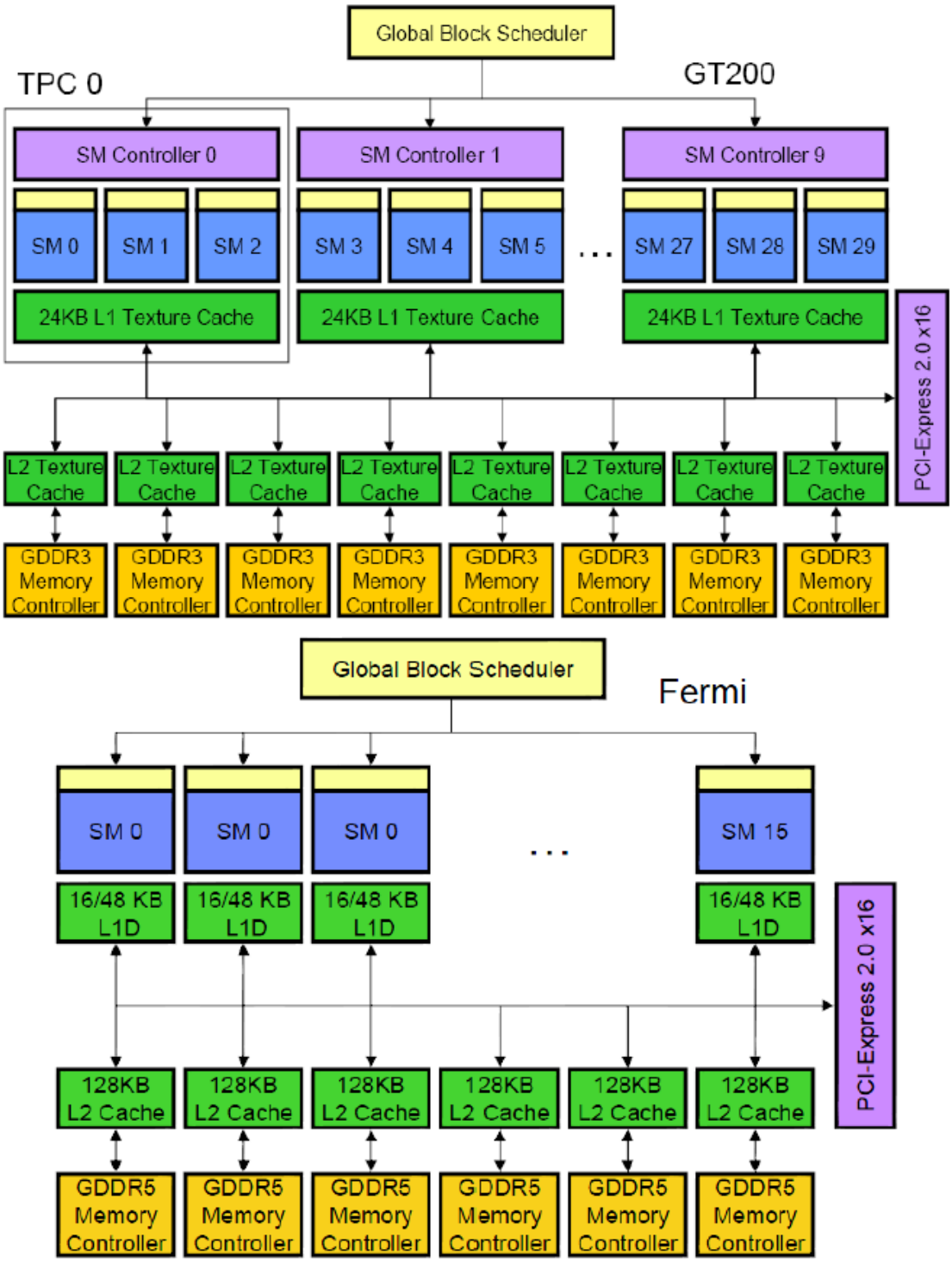


Figure 3.8 Overview of NVIDIA GPU architectures. Source: M. Daga, "Architecture-Aware Mapping and Optimization on Heterogeneous Computing Systems", Master's thesis, Virginia Tech, 2011, used under fair use guidelines, 2011.

device memory consists of thread local and global memory, both of which reside off-chip. On the GT200 architecture, the global memory has been divided into eight partitions of 256-byte width. If all active warps on the GPU try to access the same global memory partition then, their accesses are serialized, which in turn, adversely affects performance. NVIDIA Fermi GPUs also consist of a L2 cache which was missing on the previous architectures. Figure 3.8 depicts the architectural difference between NVIDIA's Fermi and the GT200 architectures.

3.3.2 NVIDIA GPU Programming model

Implementing applications on the GPUs has been assisted by the development of frameworks like OpenCL and CUDA [63,64]. Programs written in OpenCL can execute on a multitude of platforms including multicore CPUs, GPUs and even the Cell Broadband Engine, whereas programs written in CUDA can execute currently only on NVIDIA GPUs. OpenCL and CUDA are similar; the main difference is in the terminology. Figure 3.9 portrays the differences, and the following two sections describe the OpenCL and CUDA programming environments in brief.

OpenCL

OpenCL is an open standard language for programming GPUs and is supported by all major manufacturers of GPUs and some manufacturers of CPUs, including AMD, Intel, and most recently by ARM with its Mali-T604 GPU chip [65]. An OpenCL application is made up of two parts, C/C++ code that is run on the CPU and OpenCL code in a C-like language on the GPU. Code on the CPU allocates memory, stages the data, and runs it on the GPU. The OpenCL code consists of kernels, which are essentially functions designated for the GPU when invoked by the CPU. Each kernel is in turn made up of a one- to three-dimensional

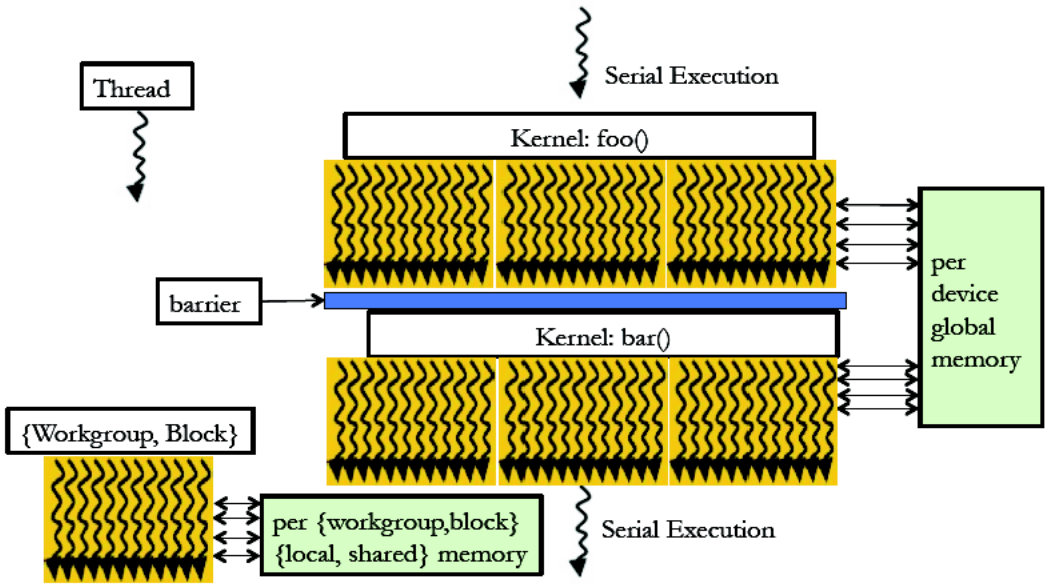


Figure 3.9 GPU Programming environments: OpenCL and CUDA. Source: M. Daga, "Architecture-Aware Mapping and Optimization on Heterogeneous Computing Systems", Master's thesis, Virginia Tech, 2011, used under fair use guidelines, 2011.

matrix of work groups, which consist of one- to three-dimensional matrices of worker threads. The kernel is the atomic unit of execution as far as the CPU is concerned, but other levels become necessary on the GPU [62]. The work groups within a kernel are not designed to communicate with each other directly. Only threads within a work group can synchronize with each another.

CUDA

CUDA is a set of extensions for C provided by NVIDIA. Unlike the sequential development model for the HC-1, a CUDA developer will write a *kernel* code that is run by each thread in parallel. The host program dispatches kernels asynchronously as threads to run on the multiprocessors of the GPU. CUDA logically arranges the threads in up to three-dimensional blocks, which are further grouped into two-dimensional grids. Each thread on the GPU has

its own identifier, which provides a one-to-one mapping. Threads pass messages to other threads via shared memory within each multiprocessor, or global memory when the message must pass between different multiprocessors. Threads can be locally synchronized with other threads running on the same multiprocessor, however global synchronization can only be performed by stopping and re-dispatching the kernel.

3.4 Summary

This chapter introduced the basic architectural features of the accelerators used in the work. Programming models for each of them are mentioned to highlight how they differ from conventional programming flows. Moreover, understanding underlying architectural characteristics helps in mapping computations and optimizing memory access patterns on these devices; the focus of Chapter 4.

Chapter 4

Approach and Implementation

Traditional benchmarks like the the Standard Performance Evaluation Corporation (SPEC), the Embedded Microprocessor Benchmark Consortium (EEMBC), and the Stanford Parallel Applications for Shared memory (SPLASH) do not completely express the parallelism available in current heterogeneous architectures. These are also confined to a narrow domain. For example, SPEC is used for benchmarking desktops and servers primarily. EEMBC, is widely known benchmark for embedded computing. These benchmarking efforts primarily took a "bottom-up" to compare different systems with the sole purpose of stressing hardware components and identifying bottlenecks, that can help architectural designers explore better designs. Current high-performance computing applications require immense computing power and span multiple domains making them vital to be considered early in exploration.

4.1 Berkeley Dwarfs

Phil Colella of Lawrence Berkeley national laboratory coined the term *dwarf* [66]; a core algorithmic kernel that captures a pattern of computation and communication. He initially

defined seven dwarfs that constitute equivalence classes, where association to a class is defined by similarity in computation and data movement. The dwarfs were specified at a high level of abstraction to allow reasoning about their behavior across a broad range of applications. Programs that are members of a particular class can be implemented differently and the numerical methods change over time, but the claim was that the underlying patterns have persisted through generations of changes and become important into the future. Six more dwarfs were added later by Berkeley parallel computing lab after data mining modern application characteristics [66].

Seeking a "top-down" approach to delineate application requirements can help draw broader conclusions about hardware requirements for emerging applications. Figure 4.1 is a temperature chart summarizing twelve of the dwarfs recurrently seen in a diverse set of applications along with their limits on performance. Compared to SPEC, EEMBC and other traditional benchmarks, the dwarfs span a much large application space. As Figure 4.1 illustrates, many of the dwarfs feature in the domain of SPEC, EEMBC and other older domain-specific benchmarks. The different colors indicate the intensity of the presence of that computational pattern in a specific application. It is evident that memory bandwidth and latency can become a major bottleneck. The dwarf that was left out is the *Map Reduce*, since it is problem dependent and variable across application domains. The Monte Carlo method is an example of a Map Reduce dwarf. It has very little communication between computation units and is considered embarrassingly parallel.

This work draws inspiration from two diverse kernels having different computation and memory requirements, as highlighted in the dwarf chart. Latter sections of this chapter will give a more detailed background about them and their implementations on the accelerators.

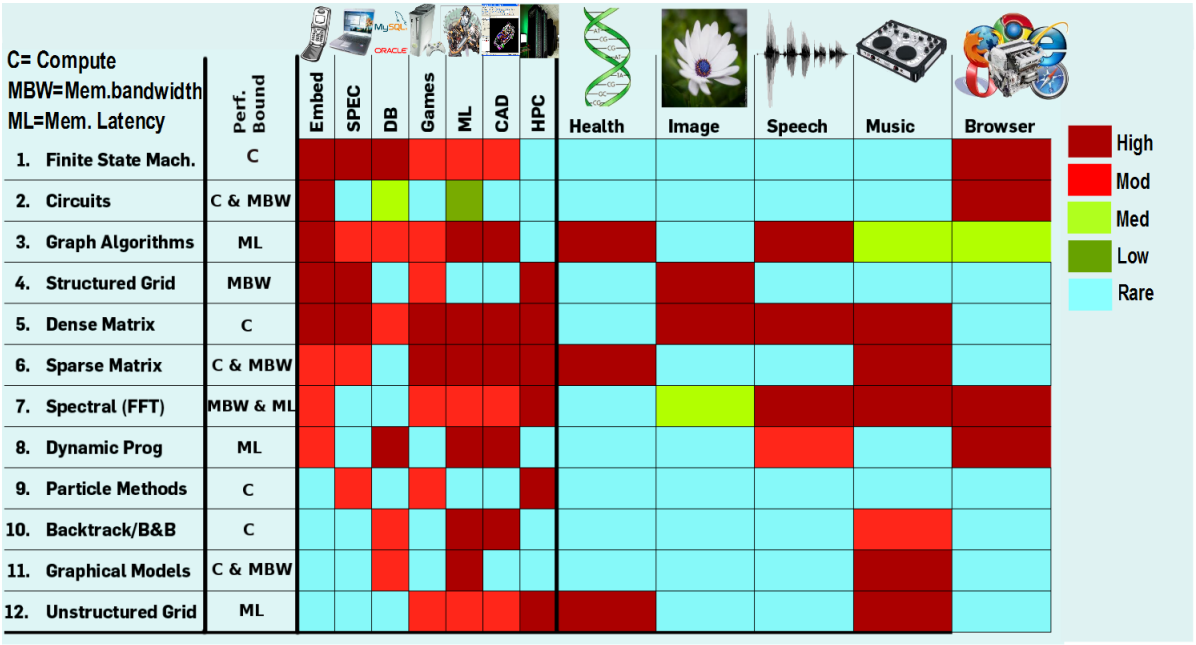


Figure 4.1 Dwarf presence in myriad applications and their performance bounds. Modified from source: K. Asanovic et al., "A View of The Parallel Computing Landscape", In the Communications of the ACM, vol. 52, pp. 56-67, October 2009, and used under fair use guidelines, 2011.

4.2 Floating-point Considerations

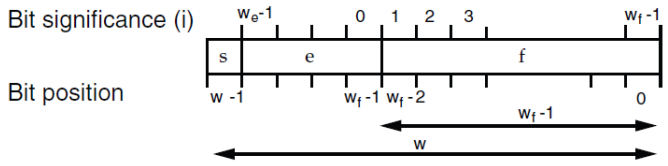


Figure 4.2 Bit fields in the IEEE-754 floating-point standard.

Scientific computing typically involves both single precision and double precision floating-point operations. Both the applications considered in this work use the IEEE-754 floating-point standard [67] whose representation is shown in Figure 4.2. A floating-point number is represented using 32-bits, consisting of a 1-bit sign component, a 7-bit exponent(w_e) and 24-bit fraction field (w_f).

There is a clear-cut advantage for GPUs, which already support IEEE-754 single precision and double precision built into the architecture for native floating-point graphic rendering operations. FPGAs, on the other hand, usually have no native support for floating-point arithmetic and many applications use fixed-point implementations for ease of development. FPGA developers use on-chip programmable resources to implement floating-point logic for higher precisions [68], but these implementations consume significant programmable resources and tend to require deep pipelining to get acceptable performance [69]. In this work, floating-point cores from Xilinx were selected for the Molecular Dynamics application [70], after considering other floating-point libraries and generators available out there [69, 71, 72]. The Xilinx floating-point cores are also used internally by the Xilinx FFT core implementation.

Dave Strenski's three part article [73] extensively analyzes floating-point performance on contemporary CPUs and FPGAs. In the case of GPUs peak performance is also easy to calculate. The peak performance for both CPUs and GPUs can be defined by the product

Device	**Absolute Peak**	**Predicted Peak**
Virtex-5 XC5VLX330	135 [73]	110 [73]
NVIDIA Geforce GTX 280	933 [49]	N/A
Virtex-6 XC5VSX315T	380 [73]	355 [73]
NVIDIA TESLA FERMI C2050	1030 [48]	N/A

Table 4.1 Device performance in GFLOPS.

of floating-point operations per clock, the number of floating cores and the clock frequency. For FPGAs, calculating the peak is slightly more complex given the different combinations of functional units that can be generated and the level of maturity of the floating point cores. The peak floating-point performance of the accelerators studied in this work is shown in Table 4.1 considering the symmetry of the butterfly operation (1-to-1 add/mult ratio) in the FFT. The peak predicted in case of FPGAs is the estimate made in [73], after factoring overhead logic due to the I/O interface, place and route overhead, and the reduced clock for timing considerations. Its easily seen that GPUs generically have a much higher floating-point performance compared to FPGAs, sometimes even 3X-4X times. The question remains as to how much of that peak can be sustained.

4.3 Custom Personality

None of the personalities, libraries and auto-code generation constructs provided by Convey were used for the FFT implementation in this work. For example, Convey provides a single-precision vector personality with 32 function pipes. Each pipe provides four Fused-Multiply Add (FMA) operations. The peak performance is thus 76.8 GFLOPS (32 pipes * 4 FMA/pipe * 2 FLOP/FMA * 300 MHz). The Convey-supplied Math Library (CML) for a 1-D FFT, that

internally uses the single-precision vector personality also peaks at 65 GFLOPS [50]. Single precision floating-point performance for a matrix multiply algorithm on the Convey HC-1 gave a sustained performance of 76 GFLOPS [51]. These numbers are low considering the abundant compute resources available in these high-end architectures. The design in this thesis is a custom developed personality using Convey's Personality Development Toolkit (PDK) [52], and is hand-tuned for maximum efficiency by balancing resource utilization with available memory bandwidth.

The host CPU communicates with the co-processor via the Front Side Bus (FSB), that creates an I/O bottleneck for streaming applications; therefore, for experiments in this study, the datasets are appropriately allocated in device memory and the communication overhead is ignored. The bandwidth of both PCIe (16-lane) and FSB is roughly 8.5 GB/s, so the assumption does not bias either platform. The performance of a streaming FFT illustrates how important inter-processor bandwidth is for accelerators. Consider a 1024-pt FFT where the real and imaginary components are represented with 32-bit floating-point numbers. Assuming the sustained FSB bandwidth is an optimistic 4.8 GB/s (roughly half the peak of 8.5 GB/s) and that the accelerator is infinitely fast, such that no time is spent performing the calculations, this operation requires 8 KB (1024 X 8 Bytes) of data to be transferred 629 K times (4.8 GB/s / 8KB) per second. The resulting performance is only around 32 GFLOPS as shown in Equation 4.1.

$$(629\text{k} \times 1024 \times 5 \times \log_2 1024) \,/\, 1\,\text{s} = 32 \text{ GFLOPS} \tag{4.1}$$

Clearly, the I/O bandwidth is the bottleneck since FPGAs are capable of higher floating-point performance. However, the advantage of the FSB is that it keeps the data coherent with the processor memory space compared to the PCIe-based accelerators. Emerging heterogeneous

computing architectures that *fuse* the CPU and GPU on the same die, e.g. AMD's Fusion Accelerated Processing Unit (APU) and Intel's Knights Ferry, hold the promise of addressing the PCIe bottleneck in GPU accelerators [74, 75]. Clearly, higher I/O bandwidth is required if sustained performance is to be achieved in streaming applications for FPGA accelerators.

4.4 The Fast Fourier Transform

4.4.1 Introduction

The *Fast Fourier Transform* is a spectral method used to efficiently compute the Discrete Fourier Transform (DFT) and its inverse [76]. The number of complex multiplications and additions required to implement an N-point DFT is proportional to N^2, with N being a power-of-2. The FFT algorithm achieves its computational efficiency through a divide-and-conquer strategy. The 1D DFT can be decomposed so that the number of multiply and add operations is proportional to $N \log_2 N$. The idea is to group the time and frequency samples in such a way that the DFT summation over N values can be expressed as a combination of two-point DFTs. The two-point DFTs are called *butterfly* computations and require one complex multiply, and two complex additions to compute. The notation used for a butterfly structure is shown in Figure 4.3, for the Decimation-In-Time (DIT) FFT. A Decimation-In-Frequency (DIF) FFT also exists, and has the same number of operations taking place as the DIT.

By using the FFT partitioning scheme, a 8-point 1D DIT FFT can be computed as shown in Figure 4.3. Each stage of the N-point FFT is composed of $(N/2)$ radix-2 butterflies and there are a total of $\log_2 N$ stages. Therefore there are a total of $(N/2) \log_2 N$ butterfly structures per FFT. In addition, the input is in bit-reverse order and the output is in linear

order. A DIF FFT on the other hand has the input in linear order and output in bit-reverse order.

The floating-point data samples stored in the external memory are complex, with real and imaginary parts each of 32 bits. The use of floating-point allows a large dynamic range of real numbers to be represented, and helps to alleviate the underflow and overflow problems often seen in fixed-point formats, however, at the cost of higher computational complexity. As introduced earlier, the metric for floating-point performance used is Floating-point Operations Per Second (FLOPS), given by Equation 4.2.

$$FLOPS = \frac{N_c * 5\, N\, log_2(N)}{T_t} \tag{4.2}$$

where N_c = Number of FFT cores, N = FFT transform size ,T_t = Transform time. The "5" from the equation comes from the number of floating-point operations in a single butterfly stage of the FFT as shown in Figure 4.3. There are a total of ten floating-point operations taking place per butterfly and $(N/2)$ butterflies per stage. A and B are complex floating-point numbers with real and imaginary components, and W_N^k is the phase factor.

The FFT was the spectral method chosen to characterize the accelerators, being both memory and computation intensive and a recurrently seen kernel in myriad signal processing applications in both high performance servers and high performance embedded systems [77]. The FFT manifests itself not only in the signal processing domain, but also in many math oriented problems. The convolution taking place in J.P Morgan's credit risk evaluation discussed earlier, uses the FFT as a critical transformation step [11]. A 2D FFT can be decomposed into two arrays of 1D FFTs, making the performance of 1D FFT highly important.

The FFT was implemented using the FFT IP core from Xilinx [78] that provides the ability

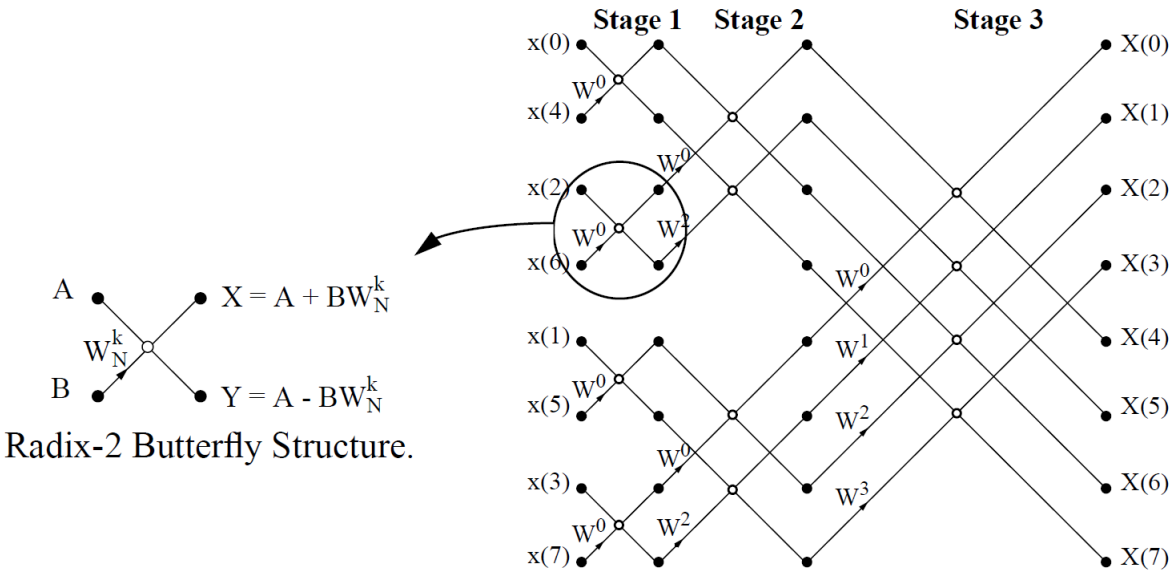


Figure 4.3 Decimation-In-Time 8-point FFT.

to make all the necessary algorithmic and implementation specific trade-offs, to select the most resource- and performance-efficient solution for any point size. Both pipelined and burst mode implementations were considered to balance resource utilization (BRAMs and DSP48E slices), transform time and available external memory bandwidth. The FPGA implementation for the Convey HC-1 used the Verilog hardware description language, the Xilinx FFT IP, Xilinx ISE 11.5 and the Convey-supplied Personality Development Kit (PDK). The Pico implementation used Verilog, Xilinx's Memory Interface Generator [30, 79], a Pico-designed PCI-endpoint module and a driver API [60]. The GPU used the FFT that is a part of the Scalable Heterogeneous Computing (SHOC) benchmark suite written in OpenCL [80]. Both the implementations use the Cooley-Tukey algorithm [76].

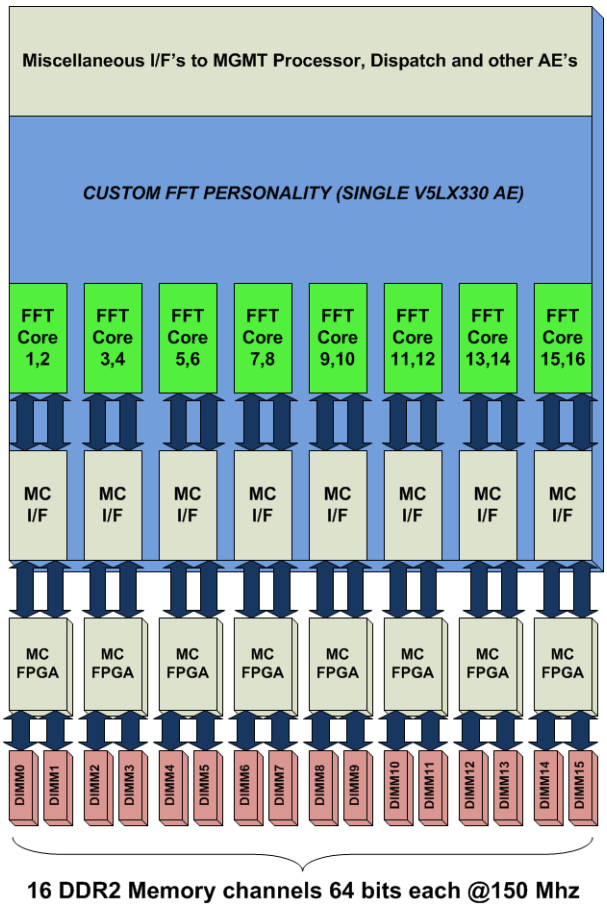
4.4.2 Mapping and Optimizing FFT on the Convey HC-1

The Mapping of the FFT on the Convey HC-1 was carried out in two phases. The first phase was an aggressive approach, aiming to maximize the utilization of the Virtex-5 die area by

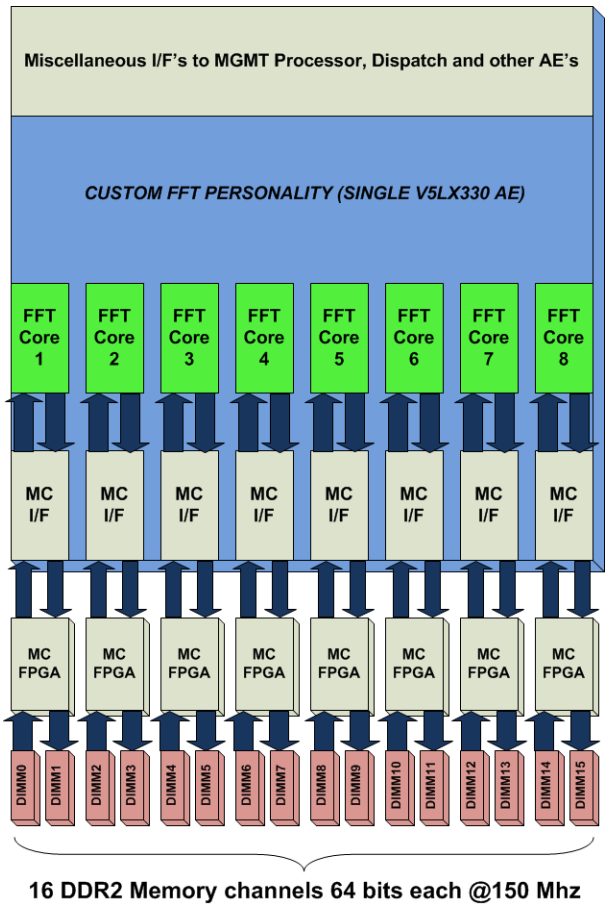
populating it with as many Xilinx FFT cores as possible. The second, was an optimized hand-tuned implementation after gathering metrics calculated from the results of the first phase.

The Xilinx core provides four architecture options (differing in memory types and DSP48E utilization) to offer a trade-off between core size and transform time. The pipelined/streaming I/O solution pipelines several radix-2 butterfly processing engines to offer continuous data processing (overlapped load, store and unload), at the expense of greater resources; the block RAMs for storing input and intermediate data; and the DSP48E slices for the butterfly arithmetic [78]. The radix-2 burst I/O core uses fewer resources at the expense of a larger transform time (load, compute and unload happen sequentially). The transform time for burst I/O architectures is increased by approximately N, the number of points in the transform, for satisfying input normalization requirements. For the pipelined/streaming I/O architecture, the initial latency to fill the pipeline is substantial, however, after the initial phase data streams through the core with no gaps, resulting in a single output transform point per clock cycle.

Initially, using all the available DSP48E resources, sixteen FFT radix-2 burst cores were mapped to each compute FPGA as shown in Figure 4.4a. The radix-2 burst cores use the decimation-in-time algorithm. The number of cores correspond to the sixteen 150 MHz memory ports available to the user. Using the streaming, pipelined cores was not feasible due to the limited DSP48Es in the Virtex-5 LX330, and using fewer memory ports would result in an unbalanced implementation, affecting performance and memory utilization. Using the 16 radix-2 burst FFT cores balanced available memory ports and utilized all the DSP48E slices (see Figure 5.6a). The performance achieved was similar to that of the Convey Math library FFT implementation and was deemed low. With the burst mode, the core does not load the next data frame until it finishes processing the current data frame. Pre-fetching the



(a) FFT Mapping on a single FPGA using radix-2 burst cores. Note the direction of data flow. Each FFT pipe has access to its own DDR2 DIMM for reading and writing. There are 16 cores.



(b) FFT Mapping on a single FPGA using radix-2 streaming/pipelined cores. Note the direction of data flow. Each FFT pipe reads from the first DIMM and writes results to the second of a pair. There are 8 pipes.

Figure 4.4 FFT mapping on Convey HC-1.

next frame while computing the current frame was a possible optimization, however superior performance was not guaranteed to justify the greater design effort.

A comparative analysis between the radix-2 burst cores and the radix-2 streaming cores favored the streaming cores in terms of performance and resource utilization efficiency. Further exploration of the Xilinx core generator, indicated a possible streaming FFT core implementation with reduced DSP48E utilization and an increased usage of the Virtex-5 Look-Up Tables (LUTs). This permitted an implementation with eight streaming/pipelined cores using fewer DSP48Es and more LUTs. The streaming implementation is based on the decimation-in-frequency FFT algorithm, having the same number of floating-point operations per butterfly as the DIT. For a pair of memory ports, the first port was used for streaming data in and the second port was used to write back results as shown in Figure 4.4b. The dedicated read/write data-paths eliminated bus turn around penalties and improved performance.

Both the approaches needed additional infrastructure to have a working implementation. Each of the FFT cores needed an input re-order queue to buffer the responses from memory matching request transaction IDs, and an output FIFO to buffer the result data to be written back to memory. Finite-state machine controllers were provided to perform the read/write requests to memory as well as to control the operation of the core as shown in Figure 4.5.

The input data was staged in such a way that the FFT units can independently read the data residing in the DIMM it is attached to, so that all the 8 or 16 pipes can operate in parallel, in a true single-instruction multiple-data manner, and have no conflicting memory accesses. This is true even when all the four AEs are in operation. Each AE accessed different banks of the DIMM to reduce the contention with other AEs accessing the same DIMM.

The host-side application involved creating data structures for the complex floating-point data values, allocating, aligning the data to the memory controllers and copying the sample

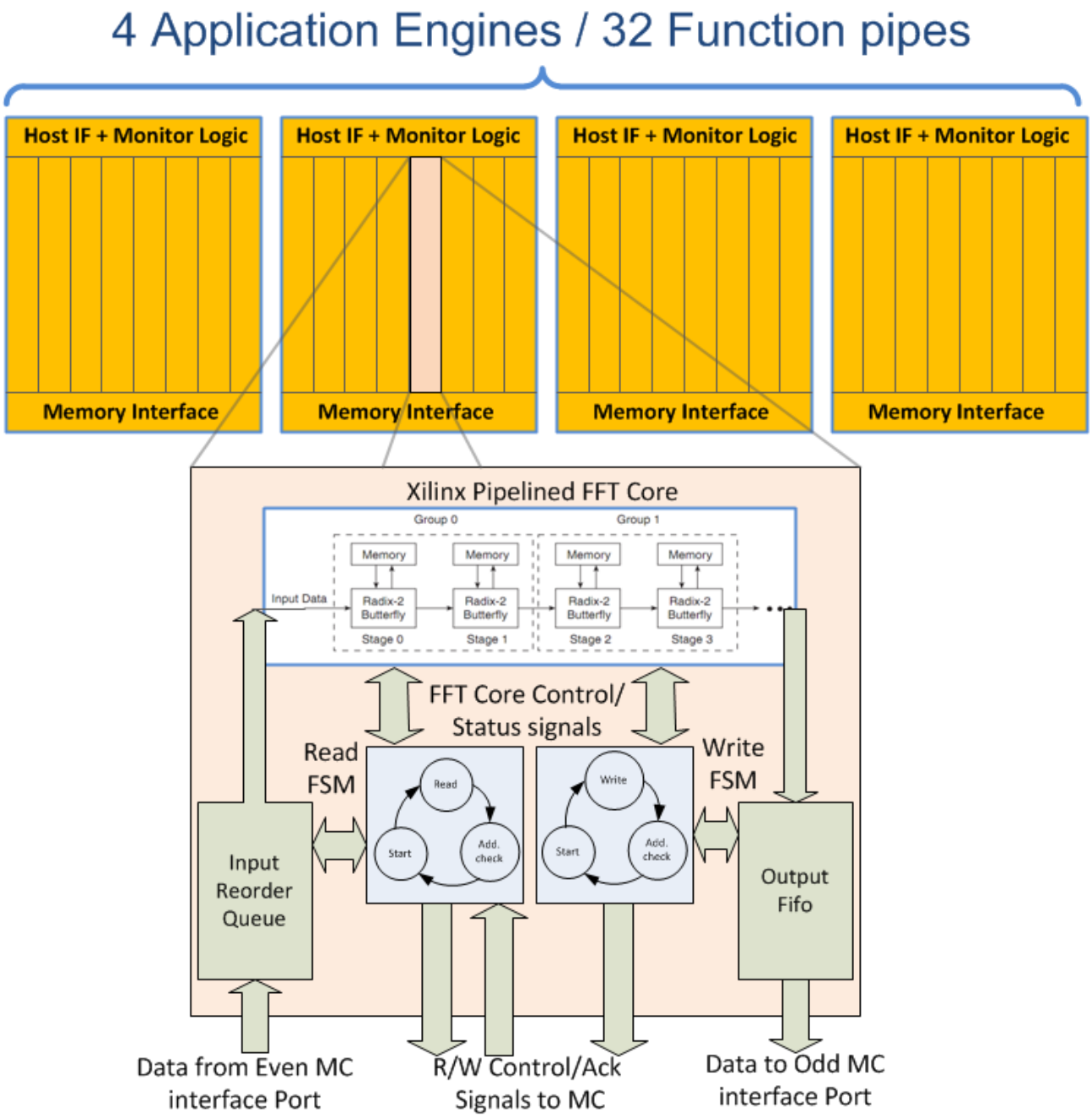


Figure 4.5 Macroscopic view of the Convey FFT Personality. It shows the internals of each function pipes on the Convey co-processor board.

data to the device memory so that the computational pipes across the four AEs have their own independent dataset. An assembly program with both scalar move and co-processor instructions provides the start/end addresses, the array length and the trigger to start the FPGA execution. On completion, the transform data was stored in a file. Finally, the execution time and memory bandwidth were measured using the co-processor interval timer that increments every 300 MHz clock cycle.

4.4.3 Mapping and Optimizing FFT on the Pico M-503

Unlike the Convey HC-1, where each memory controller is implemented on its on separate FPGA, DDR3 memory controllers had to be generated for the Virtex-6 SX315T using the Memory Interface Generator (MIG) from Xilinx [79]. The Virtex-6 memory interface generator simplifies the generation of the DDR3 controller as shown in Figure 4.6 and provides the following features:

- Initializes and calibrates the DDR3 SDRAM.
- Presents a flat address space to the user-logic and translates it to the addressing required by the DDR3 SDRAM.
- Buffers the read and write data and re-orders the data read to match the order of read requests. The re-ordering is done to optimize memory interface throughput.
- Provides a simpler user-friendly interface to the user instead of the more complex native interface.
- Manages the house-keeping activities of DDR3 SDRAM such as the row/bank configuration and the refresh/pre-charge phases.

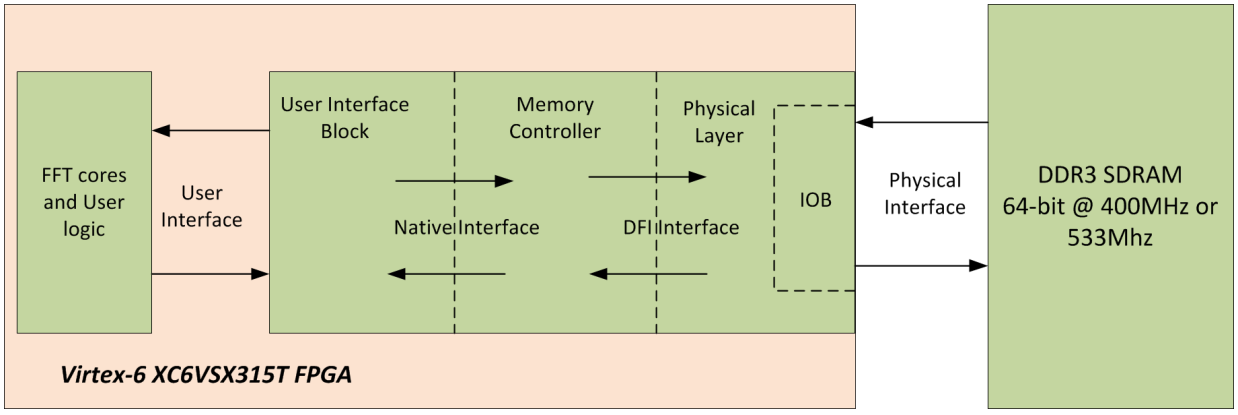


Figure 4.6 DDR3 SDRAM memory controller in the Pico M-503. User design directly talks to the user interface without having to deal with the low-level DDR3 specific physical details.

Initially, the DDR3 memory controllers were configured for 400 MHz operation, to ease meeting the timing constraints. Based on the width of the user interface provided by the memory controllers (256 bits), four FFT cores were arranged working in a lock-step manner. The memory interface requires that all user requests be made at half the memory clock, thus operating the FFT cores at 200 MHz. However, the core achieves its high throughput by using a burst of four 64-bit data words, thus producing the 256-bit interface. The SRAM banks were not used for now as they provide only 18 bits at a time, whereas the FFT core requires 64 bits of real and imaginary data in a single clock cycle. As in the Convey HC-1 implementation, the first DDR3 memory port was used for reading samples and the second for writing the results, thus, permitting continuous streaming as shown in Figure 4.7.

As assumed, the overhead of moving data from the host to the external DDR3 device memory is ignored. Since the DDR3 memory peak specified frequency is 533 MHz, higher performance could only be achieved by increasing the frequency of the memory controllers. Timing was met with additional stages being added to the datapath. Some other critical optimizations that were applied to get the maximum performance out of the Pico M-503 are as follows:

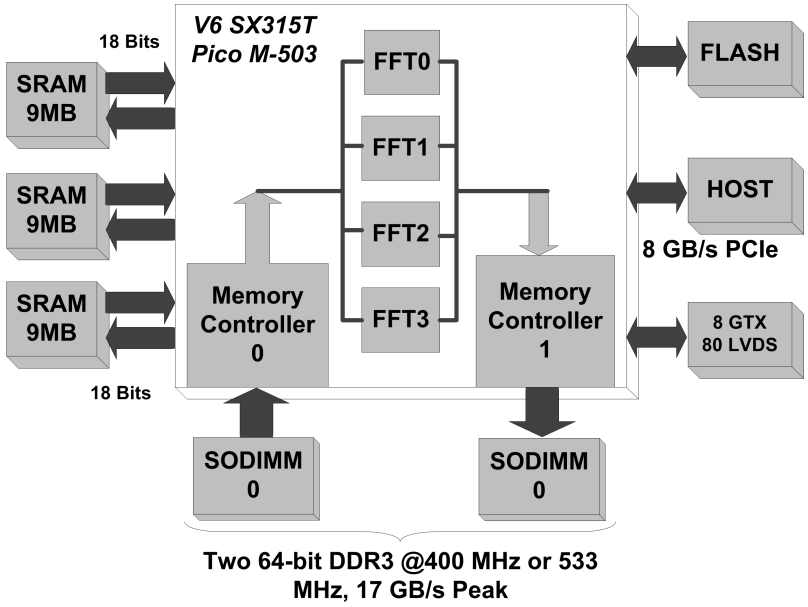


Figure 4.7 FFT mapping to the Pico M-503. It shows the four complex floating-point FFT cores and memory controllers to the DDR3 memory.

- Optimal memory access pattern for reads and writes for a burst length of 8. The code Listing in Appendix B details how the memory address is incremented. Any other step size results in poor performance. The always block shows how the write to the DDR3 occurs in two phases.

- Increasing the number of bank machines and rank machines to cater multiple simultaneous read and write requests. The bank machines are the core logic of the memory controller and manage a single DDR3 SDRAM bank by accepting/completing requests at any given time. the rank machines, on the other hand correspond to the SDRAM ranks that monitor the activity of the bank machines and track the device-specific timing parameters [79]. A fixed configuration of both bank and rank machines are provided to trade-off area and performance. Since area is not a limitation here, the number of bank and rank machines were increased to maximize performance.

- Minimized intermediate buffering. The cores were stalled when there was no data

available so there was no need to pre-fetch and store data.

These optimizations resulted in achieving almost 82% of the estimated performance without considering the effect of memory (i.e. data is available every clock cycle). Results for the Convey HC-1 and the Pico M-503 for various point sizes are discussed in Chapter 5.

4.4.4 Power Measurement Setup

The system or device under test was plugged into a Watts Up power meter [81], that measured the power drawn from the power supply as shown in Figure 4.8. The meter has an ethernet connection to a computer that captures and logs the readings once every second. The Convey HC-1 power was measured using current sensors on-board the co-processor, that measured current drawn from the bulk power supply fed to the co-processor board and does not include power consumed by the host. However, it was still treated as system power due to the large number of components on-board the co-processor, in addition to the compute FPGAs.

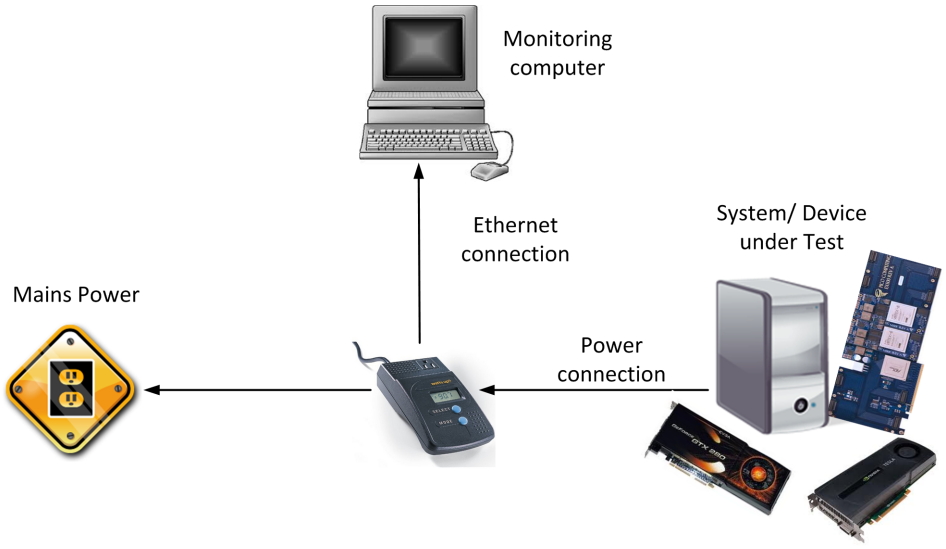


Figure 4.8 Experimental power measurement setup.

It is important to note the difference between the system power and the device power. The

system power (in the Convey HC-1 and the NVIDIA GTX 280) is the power measured when both the host processor and the device under test share the same power supply. The device power (in the Pico M-503 and the NVIDIA Tesla Fermi) is the power drawn by the device alone from a dedicated power supply. The idle power in the case of the system power is the difference in power when the card or co-processor is attached to host, and when disconnected. This is done because the GPU and the Convey HC-1 have different base CPU configurations and power supplies. For the device power, idle power is when the GPU is not running any algorithm and the Pico M-503 does not have a bitstream loaded. The load in all the devices is a 1024-point FFT run long enough (around 10s) to average out any fluctuations and startup delays. Thus, the load power indicates the power level when the system or device is running the FFT in addition to the idle power.

4.5 Molecular Dynamics

4.5.1 Introduction

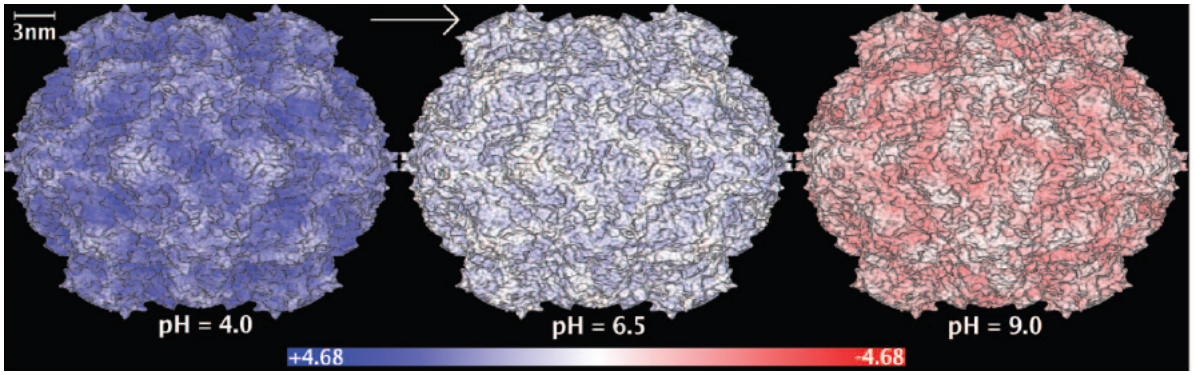


Figure 4.9 Visual output of GEM. Shows the outer surface of a virus capsid color-coded according to the electrostatic potential computed. Source: J. C. Gordon, A. T. Fenley, and A. Onufriev, "An Analytical Approach to Computing Biomolecular Electrostatic Potential. II. Validation and Applications",The Journal of Chemical Physics, vol. 129, no. 7, pp. 075 102, 2008, used under fair use guidelines, 2011.

GEM is a production-level, molecular modeling application that allows the visualization of the electrostatic potential along the surface of a macromolecule, as shown in Figure 4.9. It is an N-body computation, and a core kernel in popular molecular dynamics packages such as AMBER, that is used in rational drug design assisting AIDS research. Virginia Tech's Bioinformatics Institute (VBI) has developed two ground-breaking algorithms to this end, the Analytical Linearized Poisson-Boltzmann (ALPB) and the Hierarchical Charge Partitioning (HCP). The latter is an approximation made to speed up the algorithm [82]. The ALPB method is used for computing the electrostatic surface potential produced by a molecule. Though ALPB computes the potential on an atomic scale, it is highly computationally expensive taking days or even longer on a typical desktop [83]. The computational complexity of these force calculations can be reduced either by improving the computational algorithm or by improving the underlying hardware on which the computational algorithm runs. This

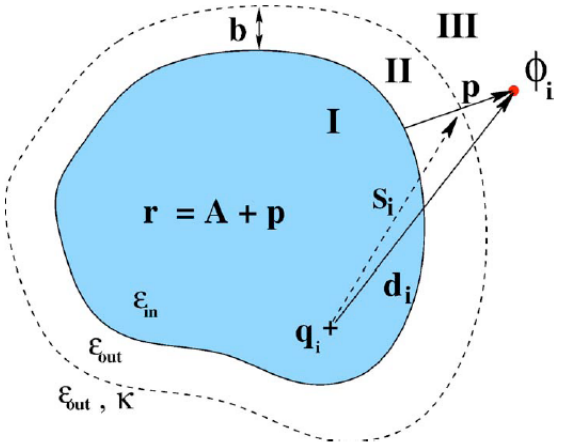
work does not apply any algorithmic optimizations such as the HCP [84], which has been known to produce very good results. The aim is to reduce the execution time by exploiting the massive parallelism in the FPGA.

Figure 4.10a shows the high-level overview of the GEM computation and the parameters involved. The Equations shown in Figure 4.10b represent the ALPB model to compute the electrostatic potential,ϕ_i, at a point on the surface of a molecule due to a single point charge, q_i. d_i is the distance between the source charge and the point where the potential is being calculated. The distance from the point of observation to the molecular surface is p, and the distance between the point of observation and the center of the surface is given as $r = A + p$, where A is the electrostatic radius of the molecule. The potential at each vertex on the surface is computed as the summation of the potentials generated by all atom charges in the system. If there are P vertex points on the surface and N is the number of atoms in the molecule, the time complexity of the potential computation is given by $O(NP)$.

The potentials at all the vertex points determine the electrostatic potential of the surface. The points are represented in 3D, using a single-precision floating-point format. While most N-Body method are all-pairs computations given a single set of points, GEM is an all-pairs computation between two sets. The problem is accentuated by the fact that both the vertices and atoms run into millions, for large molecules, making the calculation computationally challenging as well as memory intensive. The characteristics of some representative molecules used in this work are shown in Table 4.2.

4.5.2 Mapping and Optimizing GEM on the Convey HC-1

The first step in accelerating the application was to find out which functions would benefit most from hardware acceleration. The Listing in Appendix C shows a piece of C code



(a) Geometrical parameters involved in GEM.

$$\phi_i^{\mathrm{I}} \approx \frac{1}{\epsilon_{\mathrm{in}}}\frac{q_i}{d_i} - \frac{q_i}{A}\left(\frac{1}{\epsilon_{\mathrm{in}}} - \frac{1}{\epsilon_{\mathrm{out}}}\right)\frac{1}{1+\alpha\beta} \times\left[\frac{A^2}{\sqrt{(A^2-r_i^2)(A^2-r^2)+A^2d_i^2}} + \alpha\beta\right] + \frac{q_i}{\epsilon_{\mathrm{out}}}\frac{1}{1+\alpha\beta}\left[\frac{1+\alpha}{s_i}\left(\frac{1}{1+\kappa s_i} - 1\right) - \frac{\alpha(1-\beta)}{(A+b)}\left(\frac{1}{1+\kappa(A+b)} - 1\right)\right],$$

$$\phi_i^{\mathrm{II}} \approx \frac{q_i}{\epsilon_{\mathrm{out}}}\frac{1}{1+\alpha\beta}\left[\frac{1+\alpha}{d_i} - \frac{\alpha(1-\beta)}{r}\right] + F,$$

$$\phi_i^{\mathrm{III}} \approx \frac{q_i}{\epsilon_{\mathrm{out}}}\left(D\frac{e^{-\kappa r}}{r} + E\frac{e^{-\kappa d_i}}{d_i}\right).$$

(b) Equations involved with the GEM computation.

Figure 4.10 Dynamics of the GEM computation. (Figure from [82])

Structure	**# Atoms**	**# Vertices**
H helix myoglobin, 1MBO	382	5,884
nuclesome core particle, 1KX5	25,086	258,797
chaperonin GroEL, 2EU1	109,802	898,584
virus capsid, 1A6C	476,040	593,615

Table 4.2 Representative molecular structures and their characteristics.

that was filtered out from the original GEM code, after running the GNU profiler tool on the source code, indicating the most time-consuming portion, that will benefit most from acceleration according to Amdahl's law. As obvious from the code, the three loops if unrolled one by one have no dependence on each other. That is, the potential at one surface point can be computed independently of the computation of potential at other surface points, making GEM an embarrassingly parallel application. The variables in the code represent the terms in the original linearized equations in 4.10b.

The optimal way to accelerate this computation on the FPGA is to pipeline all the operators needed for a single vertex computation and stream in the atom elements one after the other. The Figure 4.11 shows this arrangement for a single vertex computation unit, after carefully analyzing the loop operations and adjusting the data dependencies.

The next step in the design was choosing the floating-point operators. An open-source floating-point arithmetic core generator suited for FPGAs particularly, called FloPoCo was studied [72]. It is a C++-based core generator framework that produces VHDL code and automatically generates pipelines from the input specifications. The Xilinx floating-point IP cores were also considered. After comparing the floating-point operators from both FloPoCo and Xilinx, and making area-speed trade-offs, the Xilinx floating-point operators were found

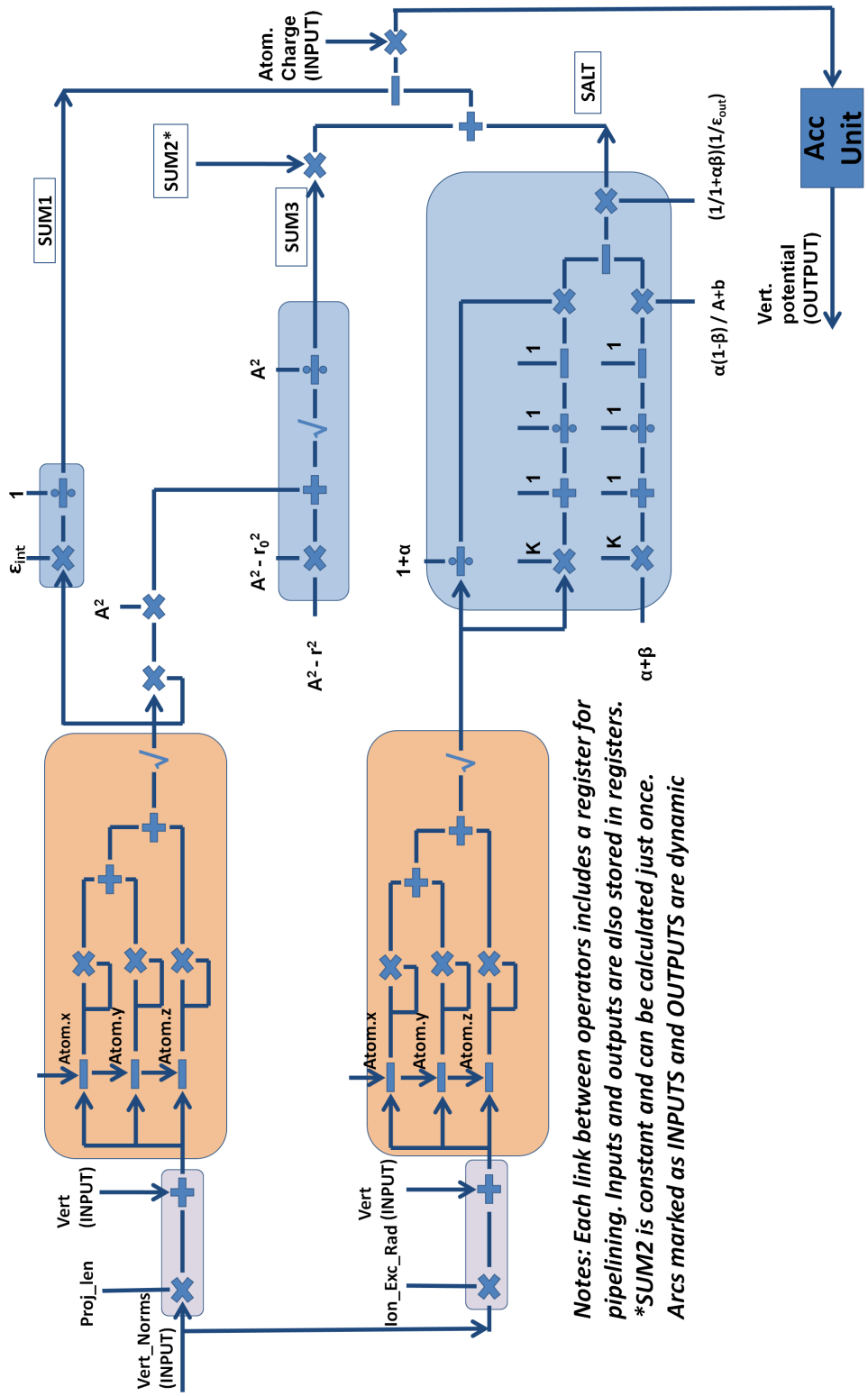


Figure 4.11 Pipeline for single vertex calculation.

superior as they have lower resource utilization and still maintain a high operating frequency. The Tables 4.3 and 4.4 summarize the findings, based on the operators needed for the pipeline. The accumulator and post-normalization units however, were adapted from the FloPoCo generator since Xilinx does not provide a floating-point accumulator. The floating-point operators in both the cases were generated for a operating frequency in excess of 300 MHz, for a possible 300 MHz pipeline operation in the Convey HC-1. Designing the pipeline involved a trade-off between latency of the core and area. For example, in the first section of the pipeline, the three dimensions of vertices and normals could be fed to the pipeline in a single clock cycle by replicating operators. However, resources were saved by serializing the inputs, since the vertex loading phase is relatively less frequent compared to atoms.

Operator	Latency (cycles)	Number in a single vertex pipe	Slice usage on V5LX330	Total Usage by operator
$+/-$	9	18	242	4,356
$*$	7	15	88	1,320
$\div$	17	4	371	1,484
$\surd$	25	3	204	612
Accumulator	2	1	70	70
Normalization	1	1	112	112
Total				**7,954**

Table 4.3 FloPoCo operator slice usage for a single vertex pipe on the V5LX330.

Using the FloPoCo operators would result in roughly five vertex pipes per FPGA, considering the total slices in the LX330 part available after factoring the contribution from the Convey HC-1 interfaces and user interfacing logic. The Xilinx operators on the other hand, resulted in eight pipes fitting on a single FPGA. Since each vertex calculation could run inde-

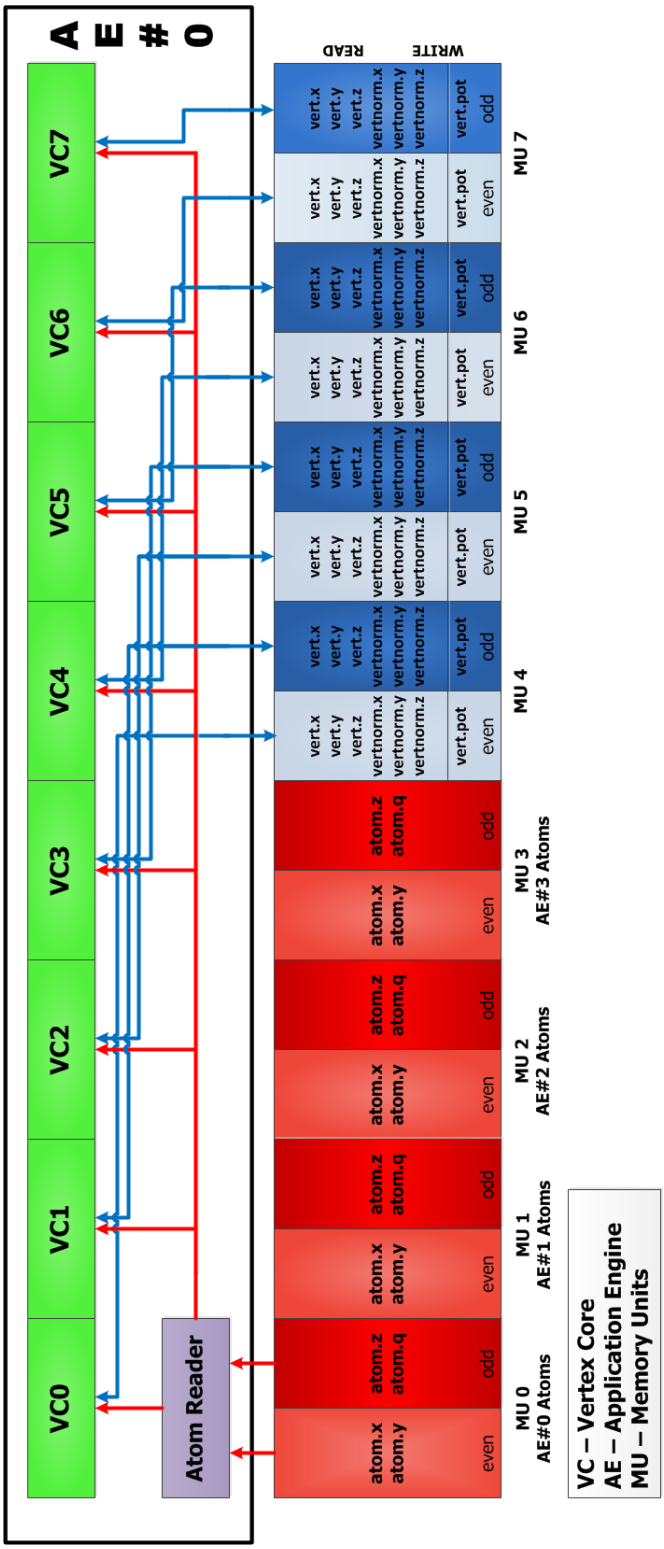


Figure 4.12 GEM Mapping on a single Convey HC-1 FPGA.

Operator	Latency (cycles)	Number in a single vertex pipe	Slice usage on V5LX330	Total Usage by operator
$+/-$	7	18	142	2,556
$*$	5	15	43	645
$\div$	20	4	278	1,112
$\surd$	20	3	196	588
Accumulator	2	1	70	70
Normalization	1	1	112	112
Total				**5,083**

Table 4.4 Xilinx floating-point operator slice usage for a single vertex pipe on the V5LX330.

pendently, multiple pipeline units were instantiated to maximize available FPGA real-estate. Figure 4.12 shows the multiple units operating in parallel. The big performance factor in the implementation is how quickly the atoms can be made available to the vertex computation units. This required careful arrangement of the data in the co-processor memory, to make sure there is minimal contention for the atoms, when all four FPGAs are accessing memory simultaneously. Hence, each FPGA (AE0-AE3) retrieves its set of atoms from an independent memory port, reserved for that FPGA, with the first 150 MHz port of a memory unit providing the x and y co-ordinates of an atom, and the second 150 MHz port providing the z co-ordinate and the associated charge.

Each vertex unit first reads the vertex (x,y,z and corresponding normals) at which the potential is being calculated. The vertex units in each FPGA, and across FPGAs share the vertex load evenly, such that each unit computes the potential of a fraction (1/32, since there are 32 total cores) of the total number of vertices in a molecule. After all the units

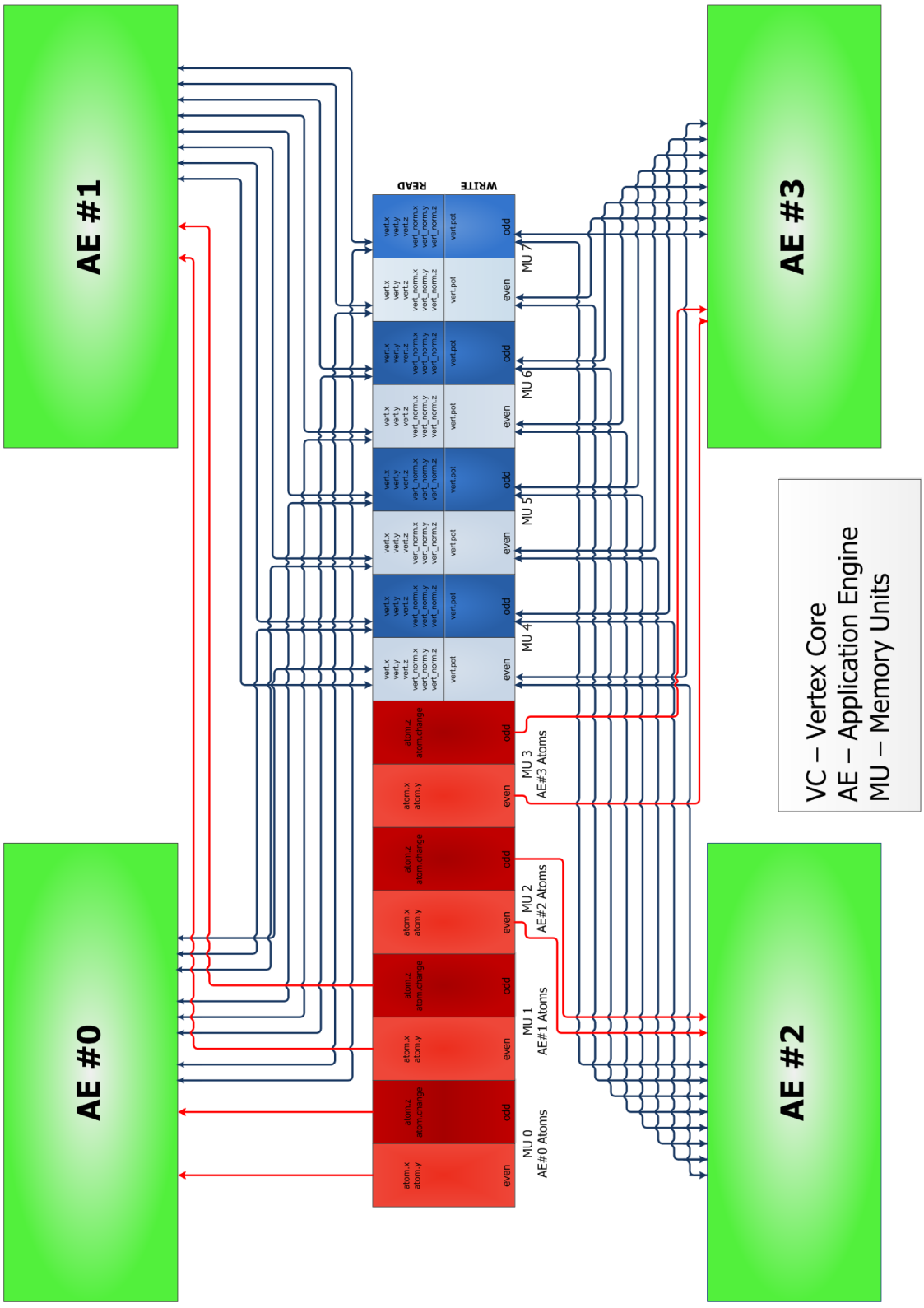


Figure 4.13 GEM mapping on all four compute FPGAs in the Convey HC-1. Optimized interaction with the memory sub-system.

are synchronized, an atom reader unit, corresponding to each FPGA fetches atoms from its allocated memory port and scatters them to the vertex computation units that operate in lock-step. The vertices are low-priority in terms of memory bandwidth requirements and hence vertex memory (memory unit 4 to 8) are shared by all the FPGAs. The write phase for the electrostatic potentials is also relatively low-bandwidth and is time-multiplexed with reads from the vertex memory units.

The constants required by the pipeline were provided via arguments to the co-processor call on the host side and passed to the vertex units through the FSB. As in the case of the FFT, the time to allocate and arrange data on the various DIMMs is ignored.

4.6 Summary

This chapter discussed the approach used in comparing GPU and FPGA architectures, focusing on FPGA implementations with respect to two key kernels, a FFT spectral method and a molecular dynamics N-body method. Key assumptions were made that will affect results on these architectures. The next chapter will discuss results and the effectiveness of the mapping.

Chapter 5

Results and Discussion

This chapter will focus on providing detailed qualitative and quantitative discussion of the results of the mapping of the FFT and GEM, on the FPGA accelerators. The metrics for comparison are performance, memory bandwidth, resource utilization, power and productivity.

5.1 FFT Results

This section will compare the 1D, complex, floating-point, FFT results on the FPGA and GPU accelerators considered. It also provides detailed characterization of the results with respect to the metrics selected and explain why the results vary across devices.

5.1.1 Performance

Figure 5.1 shows the floating-point performance for the devices considered, including integer-point numbers for the FPGA, having real and imaginary parts. The GPUs inherently support

floating-point computation and the datapath width is fixed, so a reduced precision is not efficient. For a long integer FFT (32-bits) implementation, the GPU performance would be similar to the floating-point performance, and this analysis makes that assumption. For all practical purposes, the performance would be worse due to the additional datapath scaling needed after each stage of the transform.

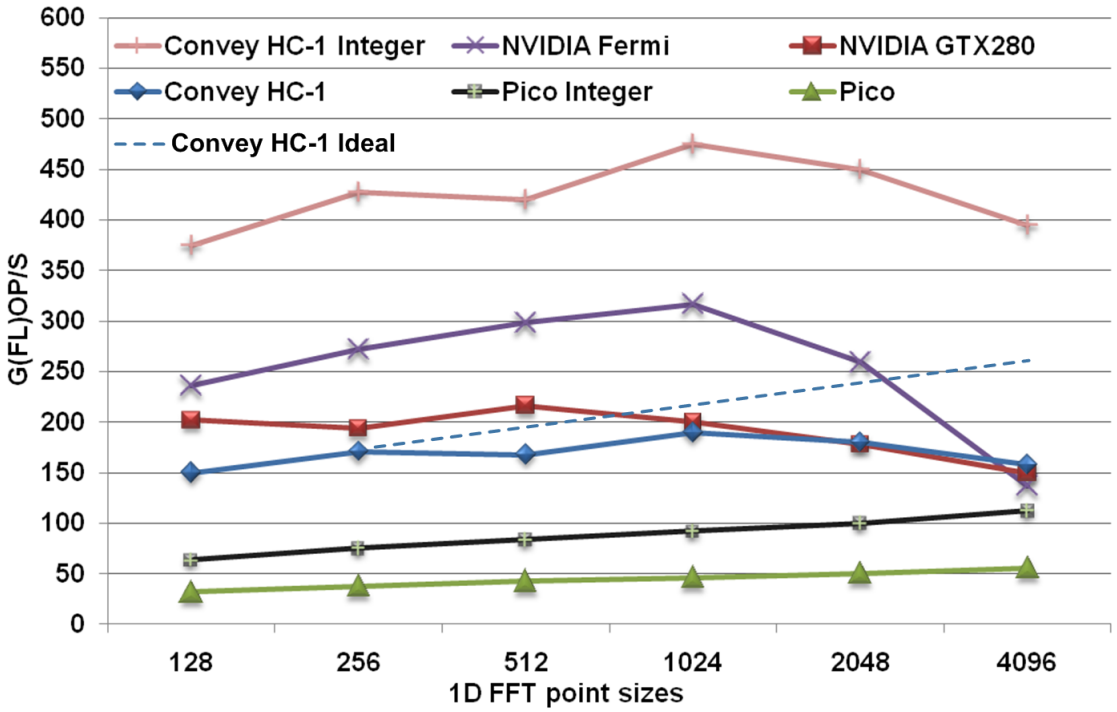


Figure 5.1 1D floating-point FFT performance on FPGA and GPU architectures.

For the Convey HC-1, the performance of all four compute FPGAs is considered against single GPU. This is done for three reasons:

1. All FPGAs being confined to the same co-processor board, makes it hard to isolate a single FPGA, specially when measuring aggregate power of the co-processor board.

2. Each FPGA can achieve a theoretical peak bandwidth of 19.2 GB/s (8 B * 300 MHz * 8 controllers) when striding across eight memory controllers. Comparing external

memory bandwidth in excess of 140 GB/s in GPUs, against a single FPGA having around 20 GB/s of memory bandwidth, makes the results biased for the GPU.

3. Finally, using all the FPGAs helps evaluate the affect of multiple FPGAs accessing the same memory controller, link saturation and its effect on performance.

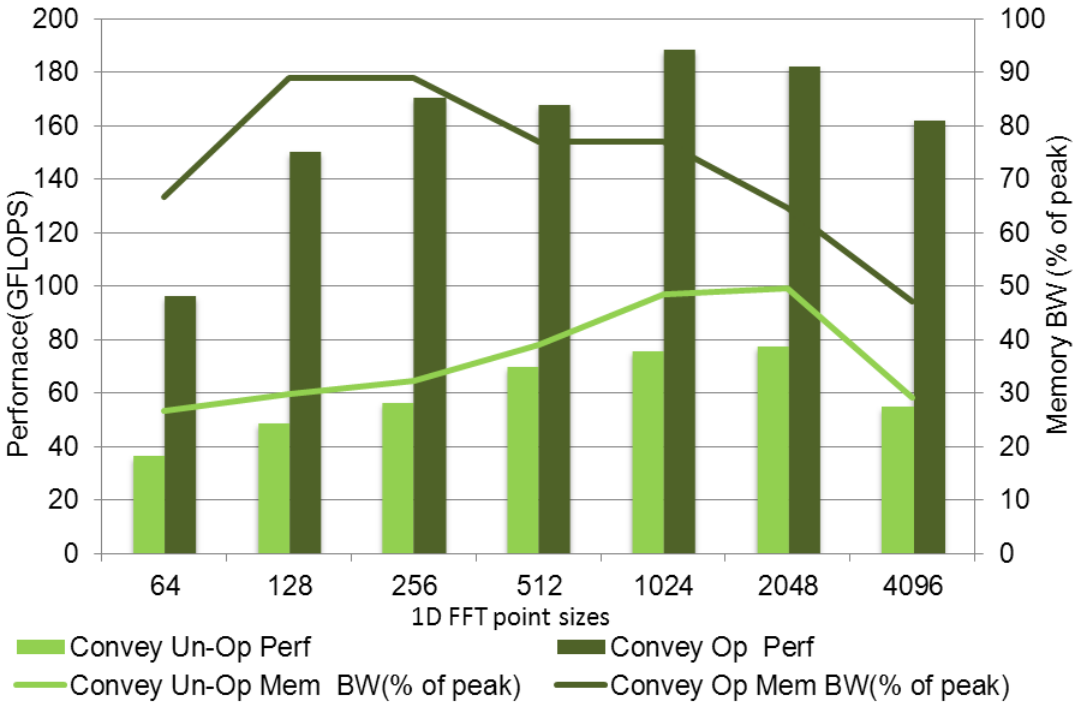


Figure 5.2 Comparing the FFT unoptimized and optimized implementations on the Convey HC-1.

The maximum performance achieved using the unoptimized implementation was close to the performance of the FFT routine in Convey's math library, of nearly 75 GFLOPS. However, measuring memory bandwidth indicated that the memory controllers were not fully utilized. Figure 5.2 compares the performance and memory bandwidth utilization for the unoptimized and optimized implementations in the Convey HC-1 for various FFT transform sizes. The unoptimized implementation results in less than 50% memory bandwidth utilization, despite having 16 burst FFT cores and availing all DSP48Es. The loss of performance is because

	Un-Optimized	Optimized			
Parameters	*Burst - All sizes*	*Upto 256-pt Streaming*	*512-pt Streaming*	*1024-pt Streaming_dsp*	*1024-pt Streaming_lut*
Target_clock_frequency	150	150	150	150	150
Transform_length	All	256	512	1024	1024
Implementation_options	radix_2_burst_io	pipelined_streaming_io	pipelined_streaming_io	pipelined_streaming_io	pipelined_streaming_io
Data_format	floating_point	floating_point	floating_point	floating_point	floating_point
Input_width	32	32	32	32	32
Phase_factor_width	24	24	24	24	24
Butterfly_type	use_xtremedsp_slices	use_luts	use_xtremedsp_slices	use_xtremedsp_slices	use_luts
Complex_mult_type	use_4_mult_performance	use_4_mult_performance	use_luts	use_luts	use_luts
Memory_options_phase_factors	block_ram	block_ram	block_ram	block_ram	block_ram
Memory_options_reorder	block_ram	block_ram	block_ram	block_ram	block_ram
Input_data_offset	no_offset	no_offset	no_offset	no_offset	no_offset
Memory_options_data	block_ram	block_ram	block_ram	block_ram	block_ram
Cyclic_prefix_insertion	FALSE	FALSE	FALSE	FALSE	FALSE
Memory_options_hybrid	FALSE	FALSE	FALSE	FALSE	FALSE
Stages_using_block_ram	0	3	4	5	5
Clock Enable	TRUE	TRUE	TRUE	TRUE	TRUE
Channels	1	1	1	1	1
Output_ordering	natural_order	natural_order	natural_order	natural_order	natural_order
Overflow_flag	TRUE	TRUE	TRUE	TRUE	TRUE
Rounding_modes	truncation	truncation	truncation	truncation	truncation
Configurable_transform_length	FALSE	FALSE	FALSE	FALSE	FALSE
Scaling_options	scaled	scaled	scaled	scaled	scaled
Synchronous_clear	TRUE	TRUE	TRUE	TRUE	TRUE

Figure 5.3 Comparing FFT coregen parameters for the Convey HC-1 implementations.

the radix-2 burst FFT cores load and process data in separate phases using an iterative approach, while sharing a single butterfly processing engine to save resources. This results in a reduction in overall raw computations per unit area-time.

The Convey HC-1 optimized implementation using streaming FFT cores however, achieves much higher performance and memory bandwidth utilization due to the pipelining of several radix-2 butterfly processing engines, giving a higher computational density per unit time. Prior to a 512-point FFT transform, all memory ports are fully utilized by the eight computational FFT cores per FPGA, that are implemented with available DSP48Es. Performance is primarily limited by memory ports. At the 512-point FFT and higher, performance is limited by the availability DSP48Es. This results in mapping fewer FFT cores and requiring a subset of the cores to use Virtex-5 Look-Up Tables (LUTs), instead of the DSP48Es, to implement the complex multipliers. Table 5.3 shows this trade-off of resources made by selecting suitable parameters during core generation.

The performance peaks at 190 GFLOPS for a 1024-point FFT transform. Six 1024-point FFT cores were synthesized using available DSP48Es and one core used the LUTs for the butterfly implementation; thus leaving one memory port remaining idle. The performance curve after the 1024-point FFT keeps falling off primarily because of resource limitations and reduced memory bandwidth utilization. The performance beyond a 4096-point FFT transform is bound by the availability of block RAMs. The numbers were obtained using the standard DDR2 memory modules and the binary interleave memory-addressing scheme. Using the Convey-designed scatter-gather memory modules and binary interleaving scheme gave approximately 6% superior performance, despite being designed for non-unity stride memory accesses. The dotted line for the Convey HC-1 indicates ideal performance, that could be achieved if DSP48E slices and block RAMs were available abundantly. The FFT performance on the Convey HC-1 would potentially reach 250 GFLOPS for a 4096-pt FFT.

The NVIDIA GTX 280 suffers after the 512-pt transform due to the capacity limits of shared memory, resulting in a high latency to the external memory.

For the Pico M-503, the performance is limited entirely by external memory ports. A linear increase in performance gives the Pico M-503 an advantage for a large FFT transform due to the abundant DSP48E slices, block RAMs, and LUTs available in the Xilinx Virtex-6. The DDR3 memory clocked at 533 MHz improved performance linearly from 400 MHz. Unlike the Convey HC-1, the Pico is not bound by BRAMs giving results upto a 32 K FFT transform. The Fermi on the other hand, performs fairly well till 1024-pt, after which it drops mainly due to the thrashing of the L1 cache. In Fermi, the 64 KB L1 cache is configurable to support 48 KB of shared memory and 16KB of L1 cache or 16 KB of shared memory and 48 KB of L1 cache. The benchmark run used the former configuration. Reversing the configuration would improve performance but its effectiveness is yet to be determined.

Part	I/O banks	Max. usable DDR3 MCs	Max FFTs	DSP48Es	Performance in GFLOPS (transform size)	Bound
XC6VSX315T	18	4	8	1344	98 (16384)	Memory & BRAMs
XC6VSX475T	21	4	8	2016	104.4 (32768)	Memory & BRAMs
XC6VLX760	30	6	12	864	192 (65536)	DSPs
XC7VX1140T	22	4	8	3360	112 (65536)	Memory

Table 5.1 Estimating maximum FFT performance on recent FPGAs.

The performance limitation in the Virtex-6 is its I/O banking architecture, and number of

available banks for implementing the memory controllers. It was found that a minimum of four I/O banks are needed to implement a single memory controller in the Virtex-6 XC6VSX315T device. There are 18 I/O banks in the XC6VSX315T; thus, permitting a maximum of four DDR3 memory controllers, since one I/O bank is reserved for the system clock. Virtex-6 I/O columns are such that only the center columns can run at the fastest possible frequency (i.e. 533 MHz). To instantiate all four controllers, the outer columns (i.e. non-center columns) would have to be used, limiting the highest frequency to 400 MHz. Thus, the speculated maximum performance on the Virtex-6 XC6VSX315T is 98 GFLOPS; extrapolated from the current configuration. The resulting memory bandwidth needed is 34 GB/s (4 * 8.5 GB/s). A different part of the Virtex-6 family, the XC6VLX760 with nearly 30 I/O banks can implement upto six memory controllers, resulting in a maximum performance of 192 GFLOPS, when running at 400 MHz. Table 5.1 presents estimated results for recent FPGAs. Even the newest Xilinx Virtex-7 series FPGAs do not have enough I/O banks to provide the high memory bandwidth needed by the FFT.

An integer-point FFT was also implemented, since FPGAs naturally favor integer precision. Double the number of integer-point FFT cores were possible, with a 16-bit data part and a 16-bit phase factor. This resulted in more than a two-fold speedup against the floating-point implementation in the Pico M-503, due to shorter pipelines. The performance reached 470 GOPS in the Convey HC-1, as seen in Figure 5.1.

The performance of the Pico M-503 optimized implementation edged a single FPGA optimized implementation in the Convey HC-1 after a 2048-point FFT transform, due to the abundant DSP48E slices in the Virtex-6. As seen in Figure 5.4, despite having just two memory ports and half the number of FFT cores, the Pico M-503 performance almost equals that of the Convey HC-1. This is mainly due to the low-latency, high-bandwidth DDR3 memory clocked at 533 MHz against the 300 MHz DDR2 memory in the Convey HC-1. This

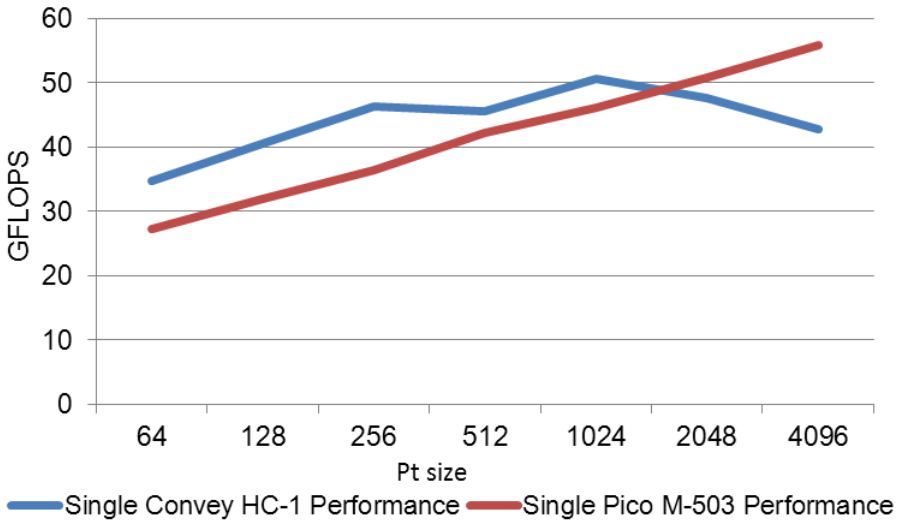


Figure 5.4 Comparing Convey HC-1 and Pico M-503 single FPGA performance.

means higher clock rates for the FFT cores in the Pico M-503. The NVIDIA GTX 280 has eight 64-bit memory channels to the GDDR3 memory, with a clock rate of 1.1 GHz, almost 3.5X faster than the Convey HC-1 memory frequency and 2X times faster than the Pico M-503 memory. The high clock rate and the abundant floating-point multiply-accumulate units in GPUs, is what makes GPU performance for the floating-point 1D FFT impressive. The FFT core IP from Xilinx is slated to run upto 400 MHz, which could potentially increase performance by nearly 2.5X, closing the performance gap with GPUs. The NVIDIA Fermi reduces the width of the memory interface by having six memory controllers. It compensates this reduced memory interface width by operating the newer GDDR5 memory at 1.5 GHz. Approximately double the number of computational cores in the NVIDIA Fermi, as compared to the NVIDIA GTX 280, justifies its superior performance.

5.1.2 Memory Bandwidth and Resource Utilization

The Convey HC-1 and the Pico M-503 sustain 88% of the estimated performance without considering the effects of memory (i.e. data is available every clock cycle), demonstrating

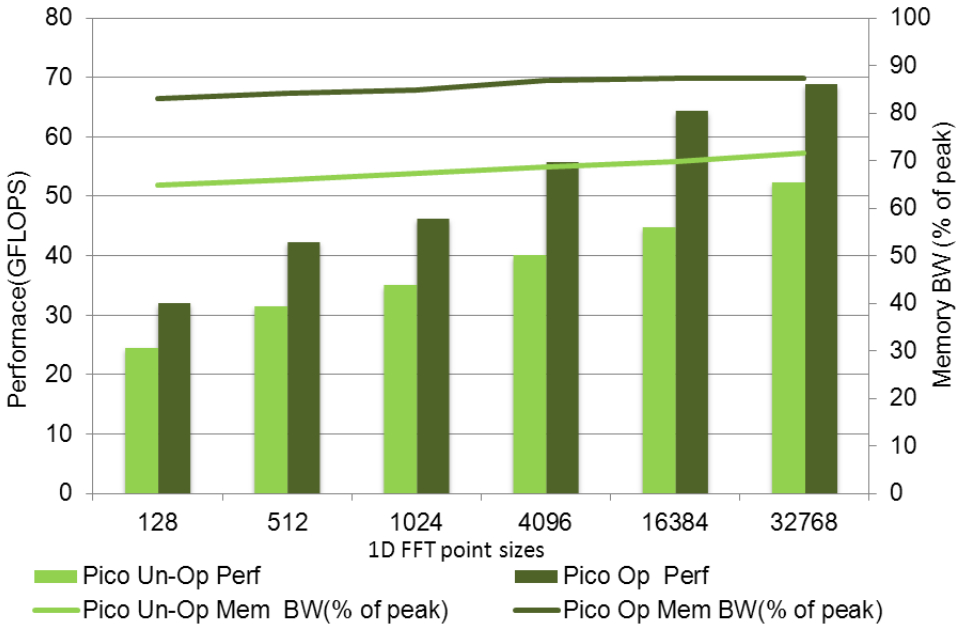
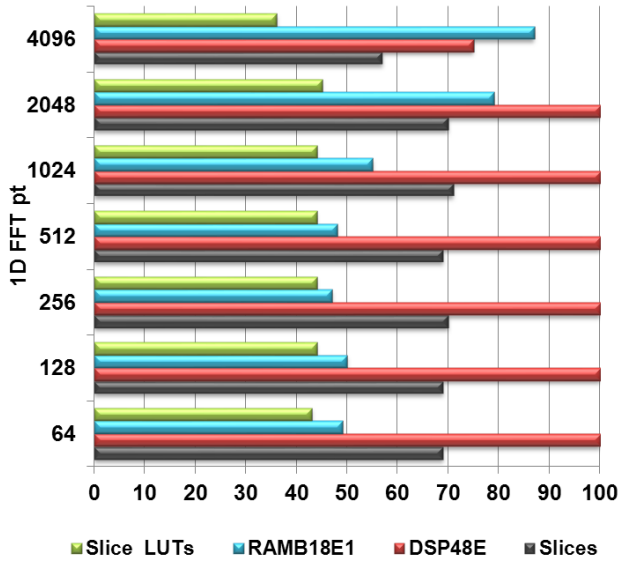
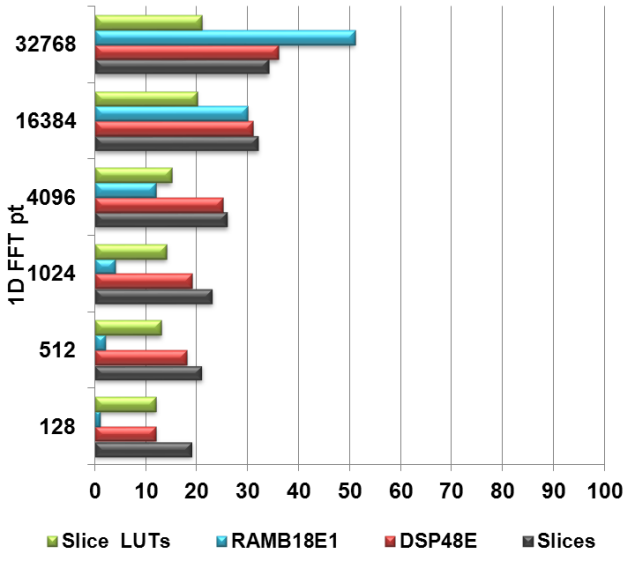


Figure 5.5 Comparing the FFT implementations on the Pico M-503.

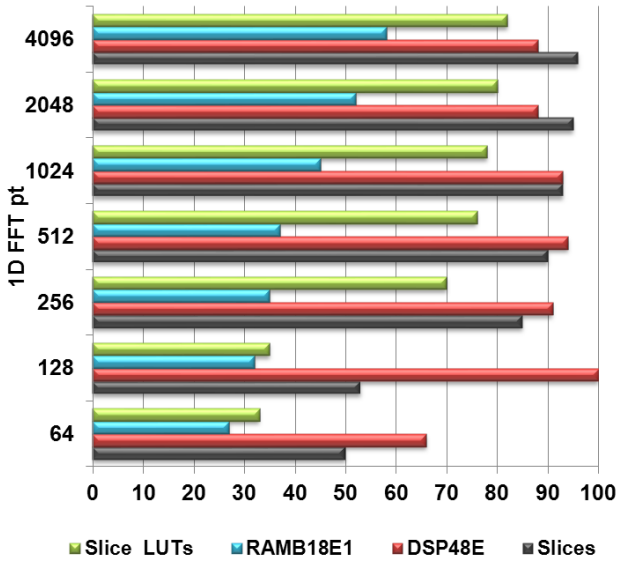
the efficient utilization of the memory subsystem by the FPGAs. In the Convey HC-1's best case, a memory bandwidth of 62 GB/s of the available 76.8 GB/s was recorded; a 80% utilization, as seen in Figure 5.2. In the Pico M-503, best performance availed 85% of available memory bandwidth, as seen in Figure 5.5. Figure 5.6 shows device resource utilization as a percentage of the peak available, in both the unoptimized and the optimized implementations on the Convey HC-1 and the Pico M-503. There is a small increase in device utilization in the optimized implementation. In the Convey HC-1, almost 90% of the slice logic and DSP48Es are used. A substantial utilization is also attributed to the Convey-designed interfaces, which occupy 10% of the FPGA area and 25% of block RAMs. The two DDR3 memory controllers, a PCIe interface, the four FFT cores and the interfacing logic utilize hardly 30% of the Pico M-503.



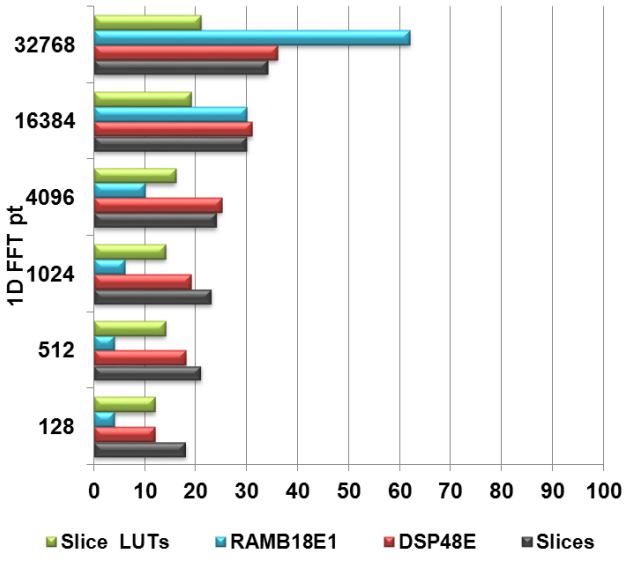
(a) Unoptimized Convey HC-1 resource utilization.



(b) Unoptimized Pico M-503 resource utilization.



(c) Optimized Convey HC-1 resource utilization.



(d) Optimized Pico M-503 resource utilization.

Figure 5.6 Resource utilization % on the FPGA architectures for various 1D FFT point sizes.

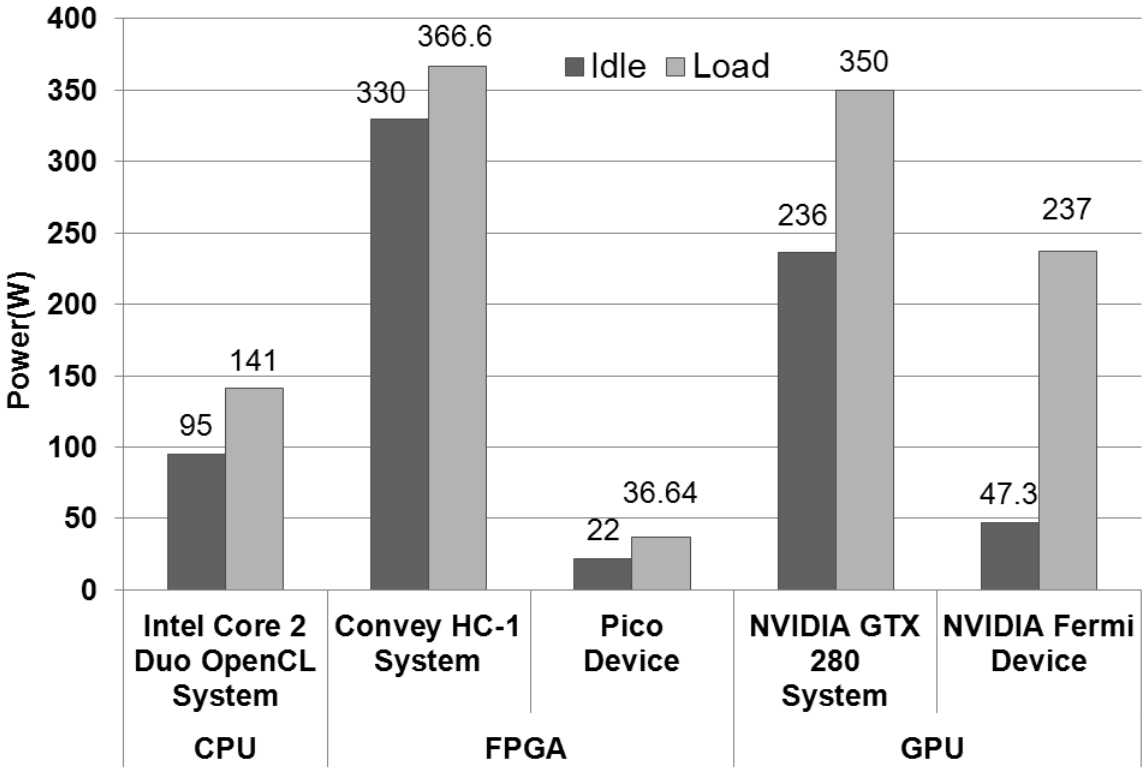
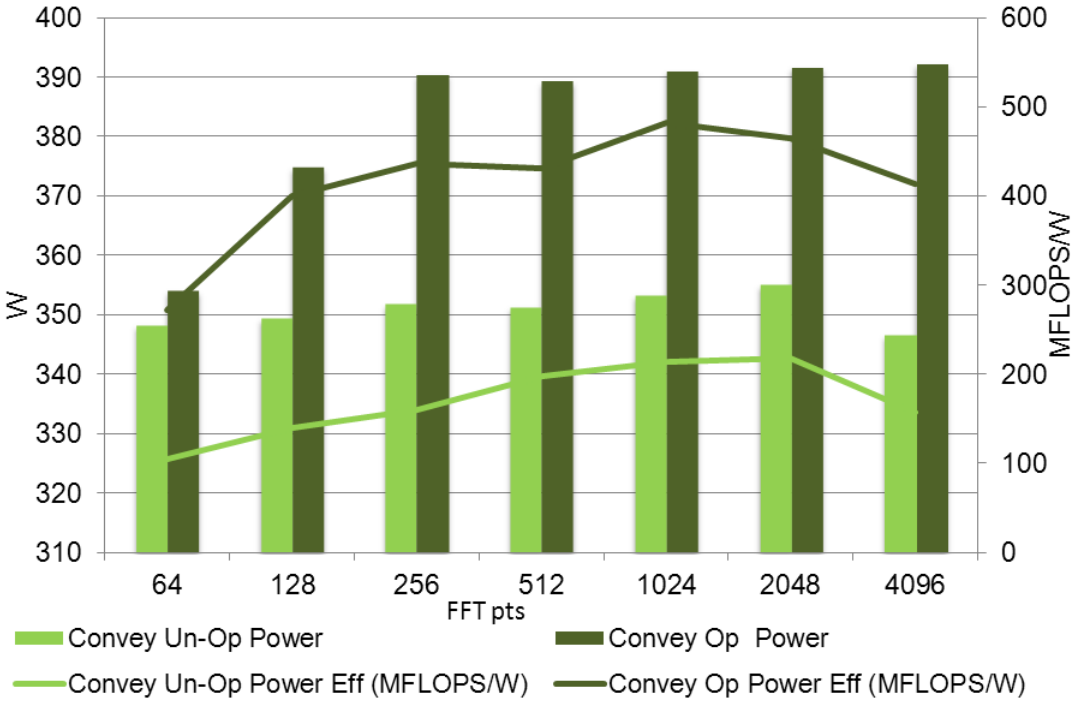
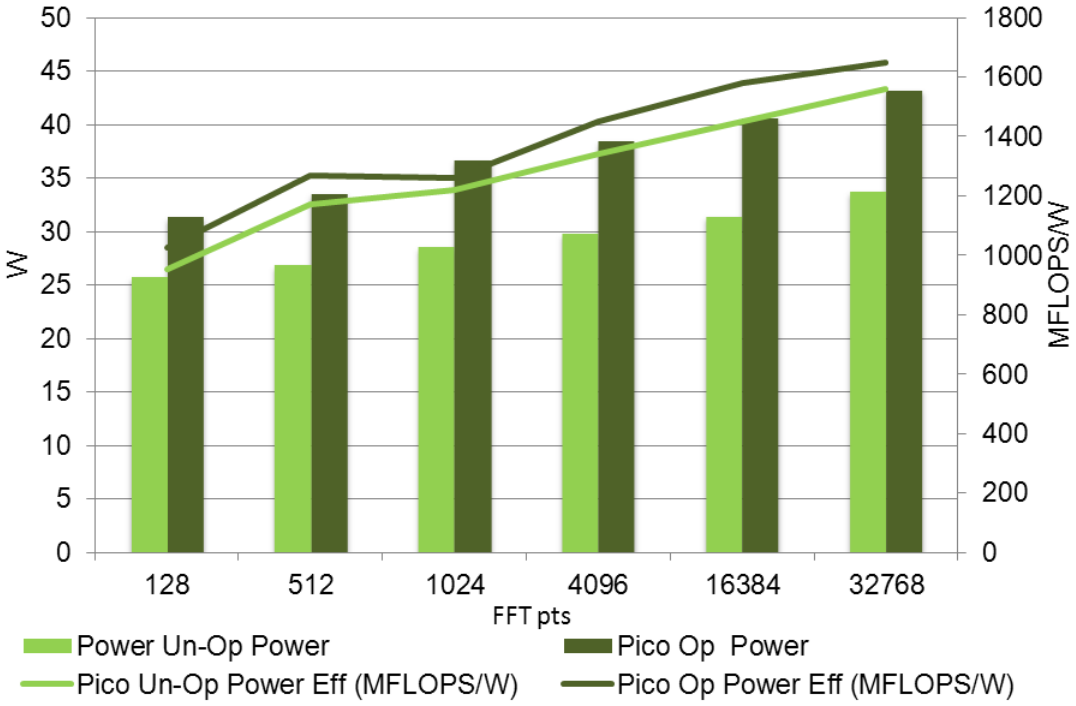


Figure 5.7 Comparing system and device power consumption of FPGAs and GPUs. Power measured at idle and at load, when running a 1024-pt 1D floating-point FFT.



(a) Comparing Convey HC-1 power for various pt sizes.



(b) Comparing Pico M-503 power for various pt sizes.

Figure 5.8 Comparing power consumption on the Convey HC-1 and the Pico M-503.

5.1.3 Power

Figure 5.8a compares the power consumption and power efficiency of the unoptimized and optimized FFT implementations. There is a 12% increase in power consumption across the different point sizes for the two versions due to greater resource utilization by the streaming, pipelined FFT cores. However, the power efficiency is much higher (almost 2.5x) due to the higher performance of the optimized implementation. Similarly, Figure 5.8b shows a 30% increase in power consumption, due to linear increase in both memory and core clock frequency for the Pico M-503 implementations. A smaller delta increase in performance of the Pico M-503 optimized implementation, results in a lower increase in power efficiency, compared to the un-optimized version.

The high core and memory frequencies increase power consumption in GPUs, as seen in Figure 5.7. The large power dissipation in the Convey HC-1 server system is attributed to the presence of nearly 16 FPGAs and supporting peripherals, making the floating-point FFT solution the least power efficient.

Configuration	**Power (W)**
Idle	6
Base PCIe Image	14.4
PCIe + 2 SO-DIMMs	23.88
PCIe + 2 SO-DIMMs + 2 DDR3 MCs + 4 1024-pt FFTs	30.2

Table 5.2 Power Dissipation in a single M-503 module only.

Pico draws the least power at idle and at load. As Figure 5.9 depicts, this results in a higher power efficiency over the NVIDIA Fermi, that performs almost 6X faster. Despite its low performance, the Pico M-503 maintains good power efficiency for large floating-point

FFT transforms compared to GPUs which suffer after a 2048-point FFT. The Pico power numbers shown here are of the entire EX-500 card. The actual powers consumed by the Pico M-503 module alone at 533 MHz are shown in Table 5.2. Since the module cannot operate in isolation, numbers reported include power due to the EX-500 circuitry. As seen in the table, the Pico M-503 draws minimal power and is ideally suited for power-constrained scenarios.

The FPGA devices also exhibit a lower dynamic power consumption from idle to load. The integer-point FFT on the Pico M-503 is the most power efficient solution, as apparent from its superior performance and lower power dissipation.

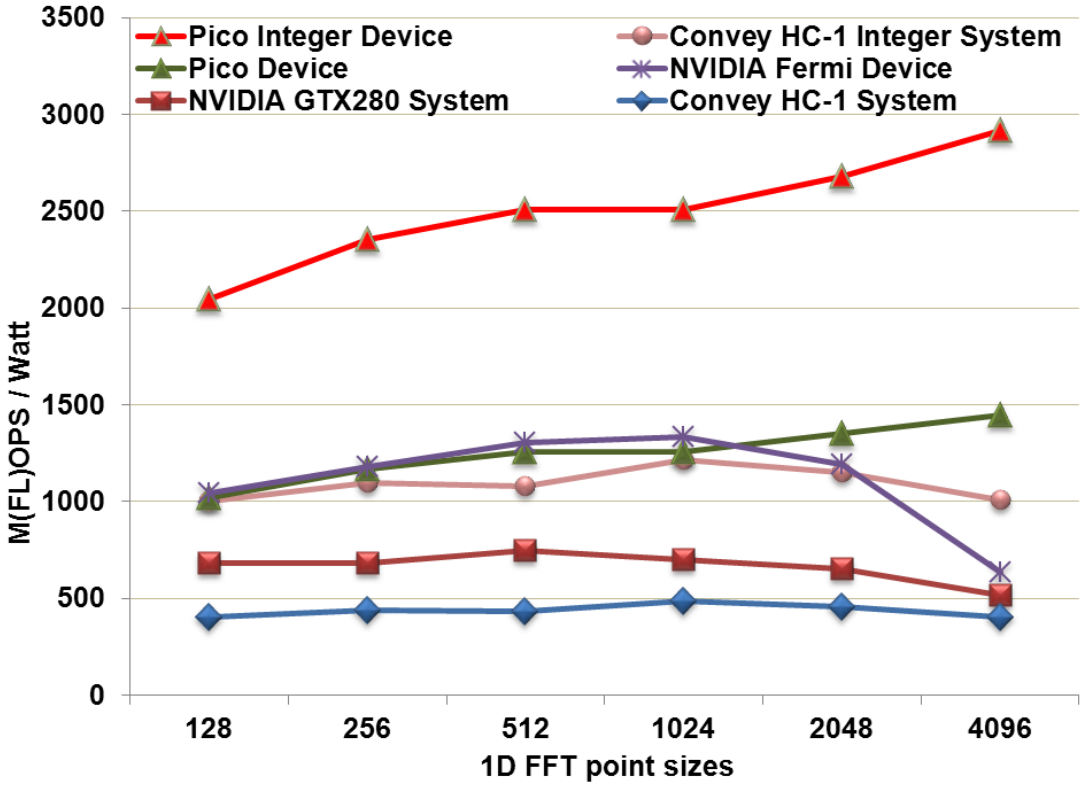


Figure 5.9 System and device power efficiency of FPGAs and GPUs. Load used when running a 1024-pt 1D floating-point FFT.

5.2 GEM Preliminary Results

This section presents preliminary GEM results that were collected for the four production-level molecular structures introduced earlier. The compilation times (shown later) for the large GEM implementation severely limited a full implementation; thus the numbers presented here are the result of a smaller implementation on the Convey HC-1.

5.2.1 Performance

Operator	Latency(cycles)	Normalized per clock	Number in vertex pipe	Operations per clock
$+/-$	7	0.143	18	2.57
$*$	5	0.200	15	3.00
$\div$	20	0.050	4	0.20
$\surd$	20	0.050	3	0.15
Accumulator	2	0.500	1	0.50
Normalization	1	1.000	1	1.00
Total per pipe				7.42
Entire Convey FPGA (32 pipes)				237.49
Total GFLOPS				**35.6**

Table 5.3 GEM FLOPS calculation.

Table 5.3 shows the FLOPS calculation for the existing pipelined operators. Since each operator is pipelined internally, it takes multiple cycles to perform a single floating-point operation. To ease calculation of FLOPS, the operations are normalized to a single clock cycle. Deep pipelining of the operators helps improve the maximum operating frequency,

but reduces the total FLOPS. As seen in the table, the total FLOPS is only 35.6 GFLOPS. The existing implementation is unoptimized since the floating-point core frequency can be lowered to a 150 MHz clock (i.e. the memory frequency of the 150 MHz Convey HC-1 memory ports), increasing the number of FLOPS sustained. As noted earlier, the operators were generated to sustain a possible 300 MHz clock frequency.

Table 5.4 shows the execution times in seconds of four implementations; a basic serial unoptimized CPU version, a CPU serial version optimized with a set of Streaming SIMD Extension (SSE) instructions, a CUDA optimized version on the NVIDIA GTX 280 GPU, and finally the Convey HC-1 implementation. The execution times featured here do not use the HCP approximation algorithm. The basic CPU version was the result of running the original GEM code on an Intel E8200 quad-core processor having a 2.33 GHz clock frequency and 4 GB of DDR2 SDRAM memory [85].

Molecule and number of atoms	**Unoptimized CPU**	**Optimized CPU (-O3+SSE)**	**Optimized NVIDIA GTX 280 GPU**	**Convey HC-1**
Helix-382	0.43	0.09	0.06	0.00078
Nucleosome-25,068	1376	371	3.3	1.58
2eu1-109,802	21395	5752	53	24
Virus Capsid-476,040	62499	16921	150	69.1

Table 5.4 GEM execution times in seconds. All execution times besides the Convey HC-1 from source : M. Daga, W. Feng, and T. Scogland, "Towards Accelerating Molecular Modeling via Multi-scale Approximation on a GPU", In Proceedings of the 2011 IEEE 1st International Conference on Computational Advances in Bio and Medical Sciences, ser. ICCABS 2011. Washington, DC, USA, IEEE Computer Society, 2011, pp. 75-80.

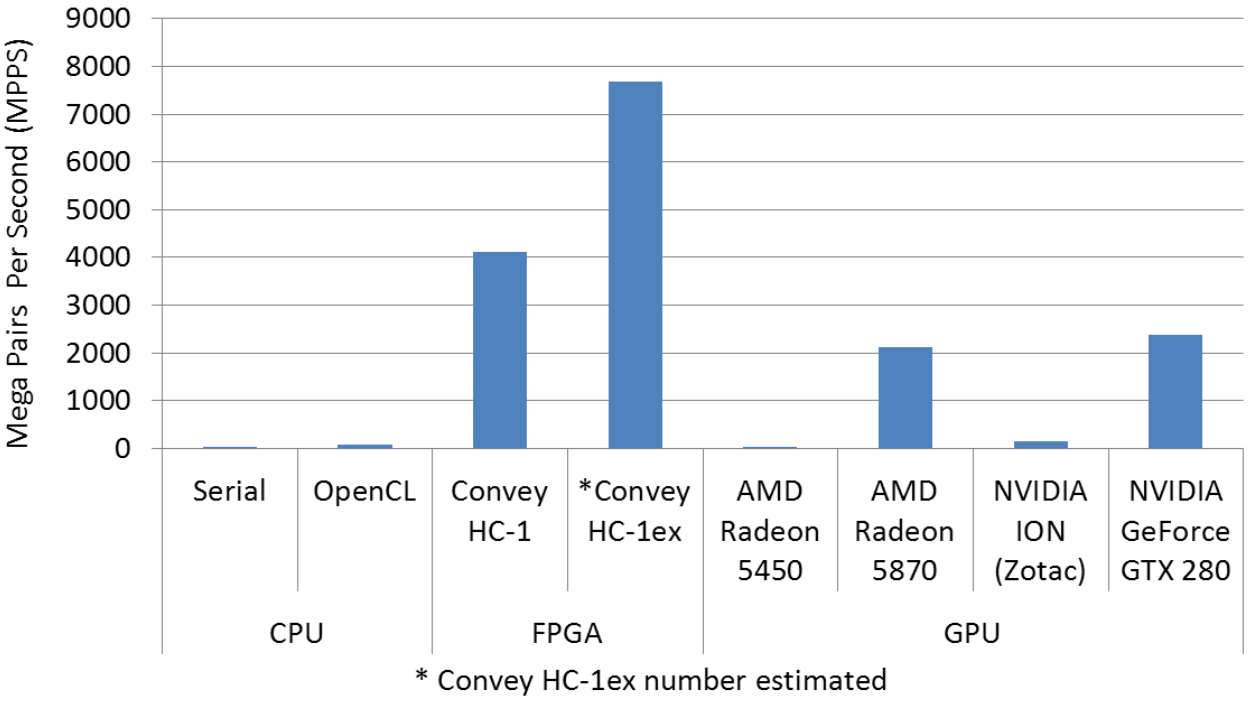
Table 5.5 shows the speed-up the Convey HC-1 achieves over the different implementations.

A speed-up of almost 900x is recorded over the basic CPU implementation. For example, modeling a virus capsid molecule using the GEM code takes around 17 hours. The same code after mapping on a single node of the Convey HC-1, takes a little over a minute. The SSE uses a specialized 128-bit vectorized float data structure consisting of four 32-bit floats, making it possible to perform the same operation on all four floats in parallel. However, a 4X speed-up over the basic version is not seen due to a hidden cost of vectorizing the code [85]. The Convey HC-1 GEM implementation roughly maintains a 200-fold speed-up against the CPU version with SSE instructions and the -O3 compiler optimization.

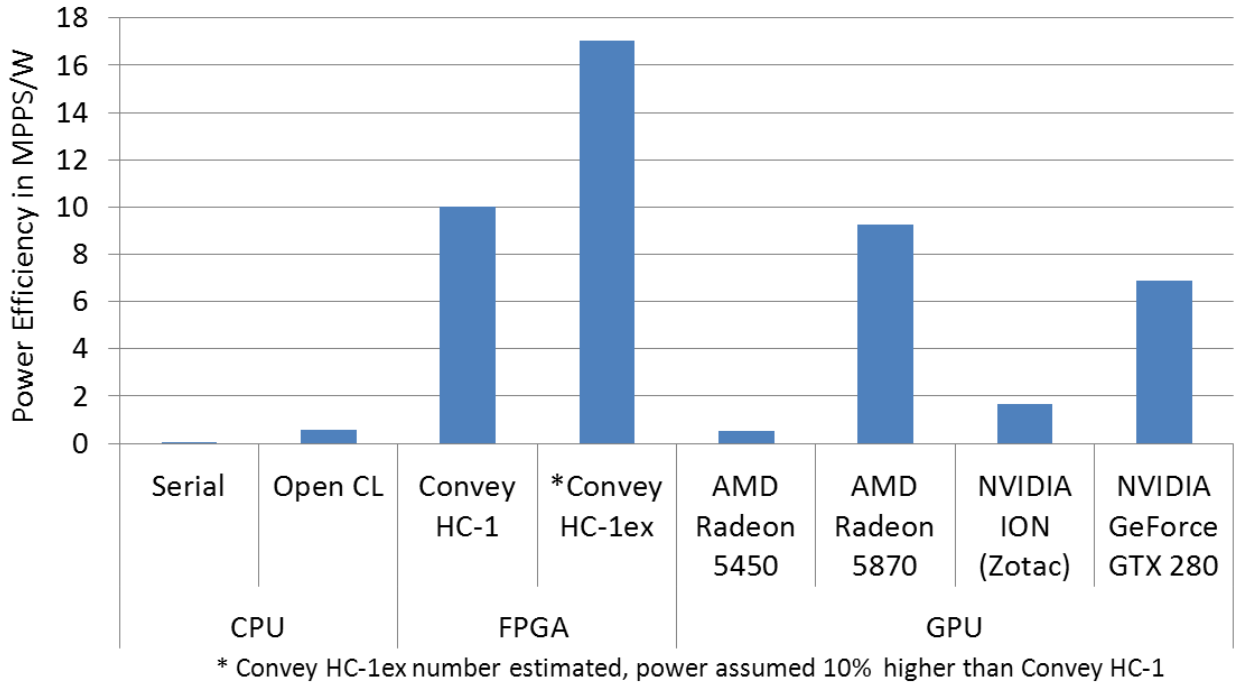
Molecule and number of atoms	**Basic serial CPU**	**Optimized CPU (-O3+SSE)**	**Optimized NVIDIA GTX 280 GPU**
Helix-382	551	115	76.9
Nucleosome-25,068	871	235	2.1
2eu1-109,802	891	240	2.2
Virus Capsid-476,040	904	245	2.2

Table 5.5 Convey HC-1 speed-ups over various implementations.

The NVIDIA GTX 280 GPU suffers for the small Helix structure due to the fewer warps possible, leaving some of the symmetric multi-processors idle; lowering the overall utilization of the GPU. The Convey HC-1 takes advantage of the effect of the larger coarse-grained parallel vertex computation units consisting of fine-grained pipelines, that amortize the cost for small datasets, resulting in almost 79-fold speed-up for the Helix molecule. However, for the large structures, the threshold of the minimum number of threads to be executed is met by the GPUs, resulting in improved performance. The Convey HC-1 still maintains a 2X speed-up over the GPU for the large molecular structures.



(a) Comparing performance in terms of MPPS.



(b) Comparing power efficiency in terms of MPPS/W.

Figure 5.10 Performance and power efficiency of the Convey HC-1 GEM implementation.

Million Pairs Per Second (MPPS) is a commonly used metric for particle methods, and is the product of vertices and atoms interacting in a medium. Figure 5.10 compares the performance in MPPS of the Convey HC-1 with other systems, for the *2eu1* molecule. The Convey HC-1 performs better than the the NVIDIA GTX 280, and the large AMD Radeon 5870 that has nearly 1600 cores. The limited number of DSP48E slices in the Convey HC-1 can be over come by using a larger Virtex-6 part, as in the Convey HC-1^{ex}. The figure gives estimated performance of the same GEM implementation on the Virtex-6, that has nearly 3X more gates compared to the FPGA in the Convey HC-1. This translates to double the number of vertex computational pipelines. The memory bandwidth requirement for the vertices is increased as a result of the doubling of the number of pipes. However, vertices being lower priority than atoms, the sequential overhead is minimal and is factored in the estimate. The bandwidth for the atoms remains unchanged due to scatter operation of the atom reader unit.

5.2.2 Resource Utilization

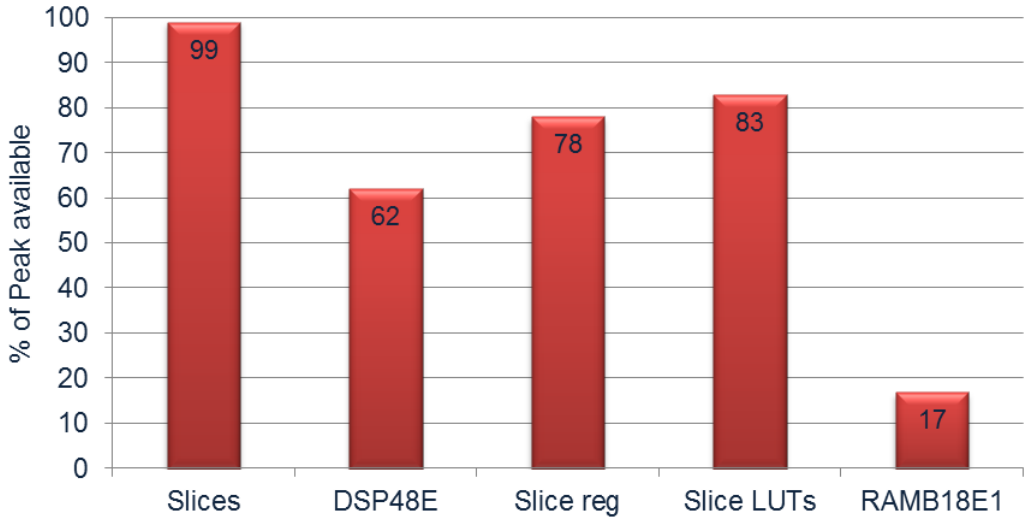


Figure 5.11 GEM resource utilization as a percentage of the peak available in a single V5LX330.

Figure 5.11 shows the device resource utilization for the Convey HC-1 GEM implementation for a single FPGA. As seen, the design just about fits on a single FPGA. As noted earlier, the operators were designed to run at a higher frequency. Fine tuning the implementation could lower resource utilization, save power and potentially reduce build time. The DSP48E slice allocation for the floating-point operators was carefully crafted using the best permutation of DSP48Es across the operator, to maximize utilization of available DSP48Es.

5.2.3 Power

Figure 5.10b compares the power efficiency of the various implementations in MPPS/Watt. The large power consumption of the Convey HC-1 overshadows its performance gains, resulting in slightly higher power efficiency than the AMD Radeon 5870. The Convey HC-1ex estimates indicate the power savings possible, being the most power-efficient.

5.3 Productivity

Productivity is one of the major issues concerning high-performance computing and can be one of the influential factors in determining the adoption of one or the other technology. Programming GPUs and FPGAs is inherently harder than conventional CPUs due to the many levels of parallelism exposed. The development tools and languages available for them are in a state of continuous flux, incurring a steeper learning curve compared to conventional CPUs. This section will investigate some of the productivity aspects of FPGAs and GPUs, based on experiences with these accelerators in this work.

Figure 5.12 gives a high-level overview of productivity in CPUs, GPUs and FPGAs. Productivity is not only measured by the time to arrive to complete a *new* design, but also

includes time and ease to iterate over an *existing* design, so that maximum performance can be attained. A number of factors influence productivity, including,

- The level of abstraction of the programming language,
- The ability to reuse pre-designed components, and
- The ratio of turns-per-day, that determines how many times a design cycle can be iterated in a single day.

In CPUs, a combination of High-Level Languages (HLLs) such as C/C++, available software static and dynamic libraries, debugging tools and standard OS platforms, have enabled fast and easy deployment of solutions. The performance is minimally controlled by compiler optimizations and is limited. GPUs are easily programmable via high-level programming languages like CUDA and OpenCL. Compared to CPUs, they require paying more careful attention to the underlying architecture details and programming model. For example, a sufficient number of threads are needed to keep the symmetric multi-processors occupied and judicious use of shared memory and L1 cache are needed to maximize performance. Thus, the initial time invested in analysis of the algorithm on the GPU architecture, is paid off of with much higher performance compared to CPUs. Finally, programming FPGAs using HDLs such as Verilog and VHDL requires close attention to intricate hardware details such as timing and delays, that the conventional CPU programmer or even a GPU programmer have nothing to do with. The performance that can be achieved is, however, the closest to the theoretical peak achievable, due to hardware acceleration.

The HLLs also improve designer productivity by increasing the number of turns-per-day, enabling faster debug. OpenCL, introduced earlier, is a platform-independent, generic method to split a computation across a heterogeneous mix of accelerators (CPUs and GPUs). CUDA,

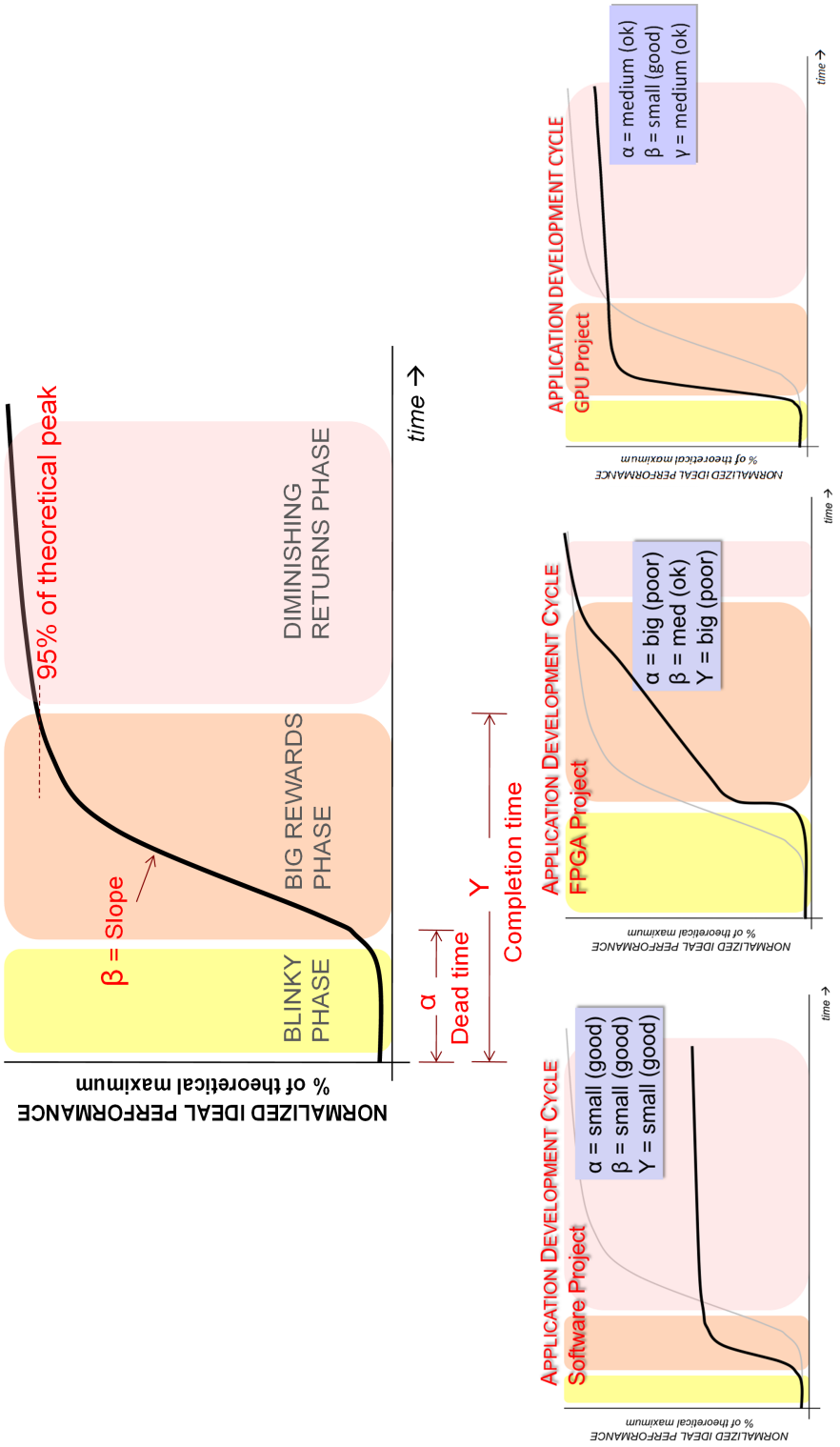


Figure 5.12 Measuring productivity across CPUs, GPUs and FPGAs.

specific to NVIDIA GPUs is the resultant code from the conversion of a high-level OpenCL specification, performed by the OpenCL compiler for NVIDIA GPUs. The key understanding here is the OpenCL framework that lends itself the flexibility to be used in a heterogeneous computing environment, agnostic to underlying hardware or operating system. Figure 5.13 illustrates the different layers of the OpenCL framework.

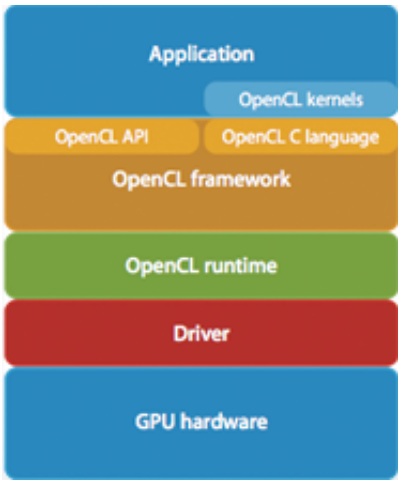


Figure 5.13 The OpenCL architecture standard. Source: B. Alun-Jones, "A Quick Introduction to OpenCL". Available online:http://www.mat.ucsb.edu/594cm/2010/benalunjones-rp1/index.html, used under fair use guidelines, 2011.

The OpenCL kernels are functions that are launched by the host program and are defined by the programmer, to be executed on the OpenCL devices. The OpenCL framework layer is responsible for identifying the devices attached and creating device contexts. The runtime layer manages the created contexts (command queues, memory objects etc). The compiler then provides the executable for the program and the kernel codes to be scheduled for the devices. The programmer specifies the core logic at a much higher level of abstraction, while the well-defined intermediate layer perform the task of converting the high-level specification to a format compatible for execution in the device. There are on-going efforts to adapt OpenCL to the FPGA reconfigurable logic [86] [87] [88], that convert OpenCL code to HDL.

The primary approach followed by [86,88] is to design the FPGA hardware so as to overlay a GPU-like architecture. This involves structuring the FPGA fabric to consist of parameterizable algorithmic cores that execute the OpenCL kernel threads in a SIMD or SPMD manner. A control core and a kernel scheduler controls core execution and memory management. The memory model corresponds to the structure in the OpenCL and CUDA specification, where LUTs, BRAMs and external SDRAM memory serve as the private memory, shared memory and global memory, respectively. [87] follows a slightly different approach, making use of an architectural template for hardware generation. The template consists of a customizable datapath unit and a streaming unit for handling memory transfers. The only difference is the slightly higher flexibility offered by [87] as compared to [86, 88]. The relevance of OpenCL for FPGAs is still questionable, since by raising the abstraction level, the flexibility in designing the hardware is taken away from the FPGA developer and an appreciable loss of performance.

5.3.1 FPGA Productivity Challenges

Programming FPGAs using HDLs is time consuming and requires low-level hardware design experience. In FPGAs, the developer effort is higher that CPUs and GPUs, since it requires creating a custom datapath and optimizing memory accesses. This drawback however, lends the programmer flexibility for certain custom applications.

Secondly, FPGA compilation tool run-times have unfortunately not kept pace with the advances in FPGA logic density. This creates a major productivity bottleneck for large implementations, common in HPC applications. For example, the 1D FFT implementation in this work uses several large-size FFT cores that can take hours or even days to compile; worsening debug time. Table 5.6 gives an idea of the compile times experienced in the designs in

this work. While GPU and CPU programmers have a few seconds of build-time, resulting in many turns-per-day, FPGA designers can usually afford one or two turns a day. Almost 70% of the time in a design is spent on verification and debug, making functional simulation crucial. The architectural simulator in the Convey HC-1 provides a useful tool for the initial phases of the design. However, as the design cycle progresses and designs get larger, simulation can take upto an hour or consume too much host memory. Another drawback of the simulator is the fact that it is not cycle-accurate and cannot model co-processor memory effectively. Figure 5.14 shows the iterative nature of the design process in the Convey HC-1.

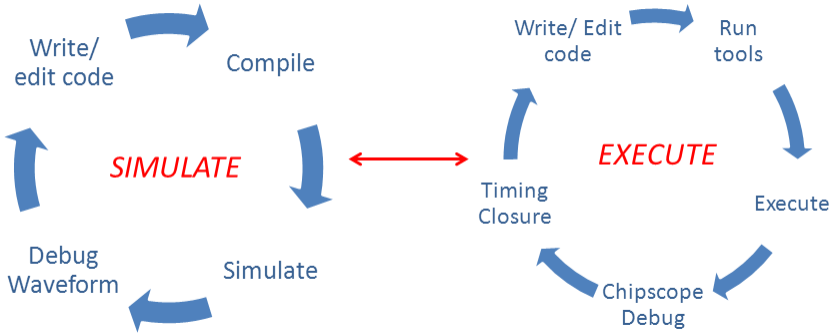


Figure 5.14 Iterative nature of developing personalities for the Convey HC-1.

5.3.2 Enhancing FPGA Productivity for HPC

IP cores and open-source cores are the alternatives for software libraries in the FPGA world. The FFT IP core used in this work can substantially reduce development time, allowing the programmer to focus on optimizing other portions of the design, primarily memory accesses. These IP cores are optimized for each generation of FPGA architectures unlike CUDA or OpenCL code, that would run on a new GPU architecture, but may not be optimal [62]. Systems such as the Convey HC-1 help speed-up development time by supplying the necessary interfaces, providing accelerated mathematical kernels via Convey Math Libraries

Implementation	Platform	Slice utilization (%)	Build Times(hrs)
16 1024-pt burst cores	Convey HC-1*	72	4-5
8 1024-pt streaming cores	Convey HC-1*	85	7-9
8 Vertex pipe units + 1 atom reader for GEM	Convey HC-1*	99	13-14
4 32768-pt 4 pt streaming cores	Pico M-503**	50	18-20
Notes: * With 8 Memory controller I/Fs, Dispatch I/F and Debug I/F ** With PCIe I/F, 2 DDR3 Memory controllers			

Table 5.6 Compilation times for designs in this work

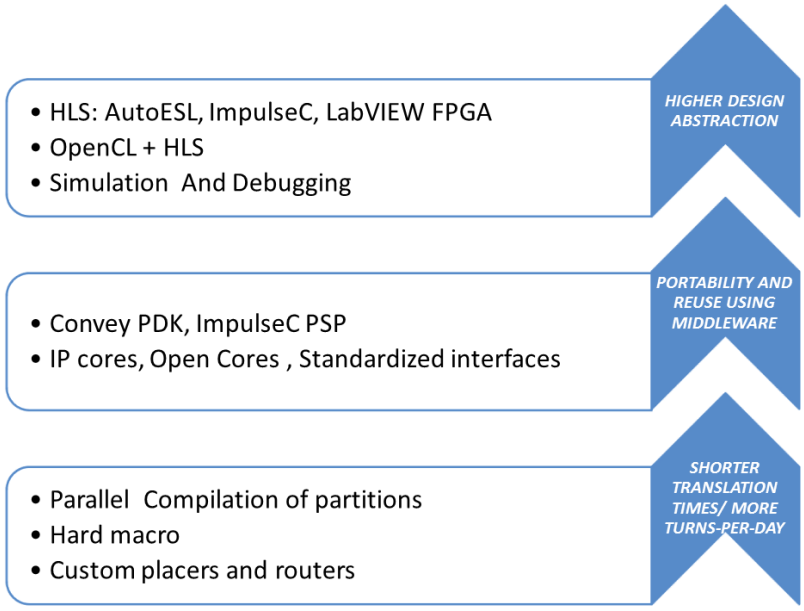


Figure 5.15 FPGA front-end and back-end productivity enhancements for HPC.

(CML) and abstracting away communication details from the developer. A combination of front-end High Level Synthesis (HLS) tools, reusable cores, pre-configured interfaces and back-end tool enhancements are needed to increase designer productivity and turns-per-day; closing the productivity gap with respect to GPUs. Figure 5.15 lists the approaches.

Electronic system-level tools provide higher levels of abstraction than traditional HDLs. There are several HLS tools available today. Most of them are based on a subset of C/C++, extended with a set of specific data types or I/O capabilities. This section will provide a brief overview of the experience with the ImpulseC environment and efforts to use ImpulseC to develop personalities for the Convey HC-1.

ImpulseC is primarily based on the ANSI C standard. ImpulseC provides two levels of parallelism,

1. Coarse-grained parallelism via ImpulseC processes that communicate through streams, signals or registers, and

2. Fine-grained parallelism within a process by automatically pipelining instructions and data-flow parallelism.

ImpulseC eases hardware-software integration by providing the option to configure an ImpulseC process as a hardware process, ie. a module hard-wired in the FPGA, or a software process, that can be executed on the host processor. ImpulseC also provides Platform Support Packages (PSPs), specific to a particular hardware platform, that abstract away communication details between hardware and software processes.

Impulse Accelerated Technologies just released a beta version of a Convey HC-1 PSP that can potentially speed-up development time for Convey personalities. Figure 5.16 shows the process involved in converting a high-level specification of an application into a bitstream,

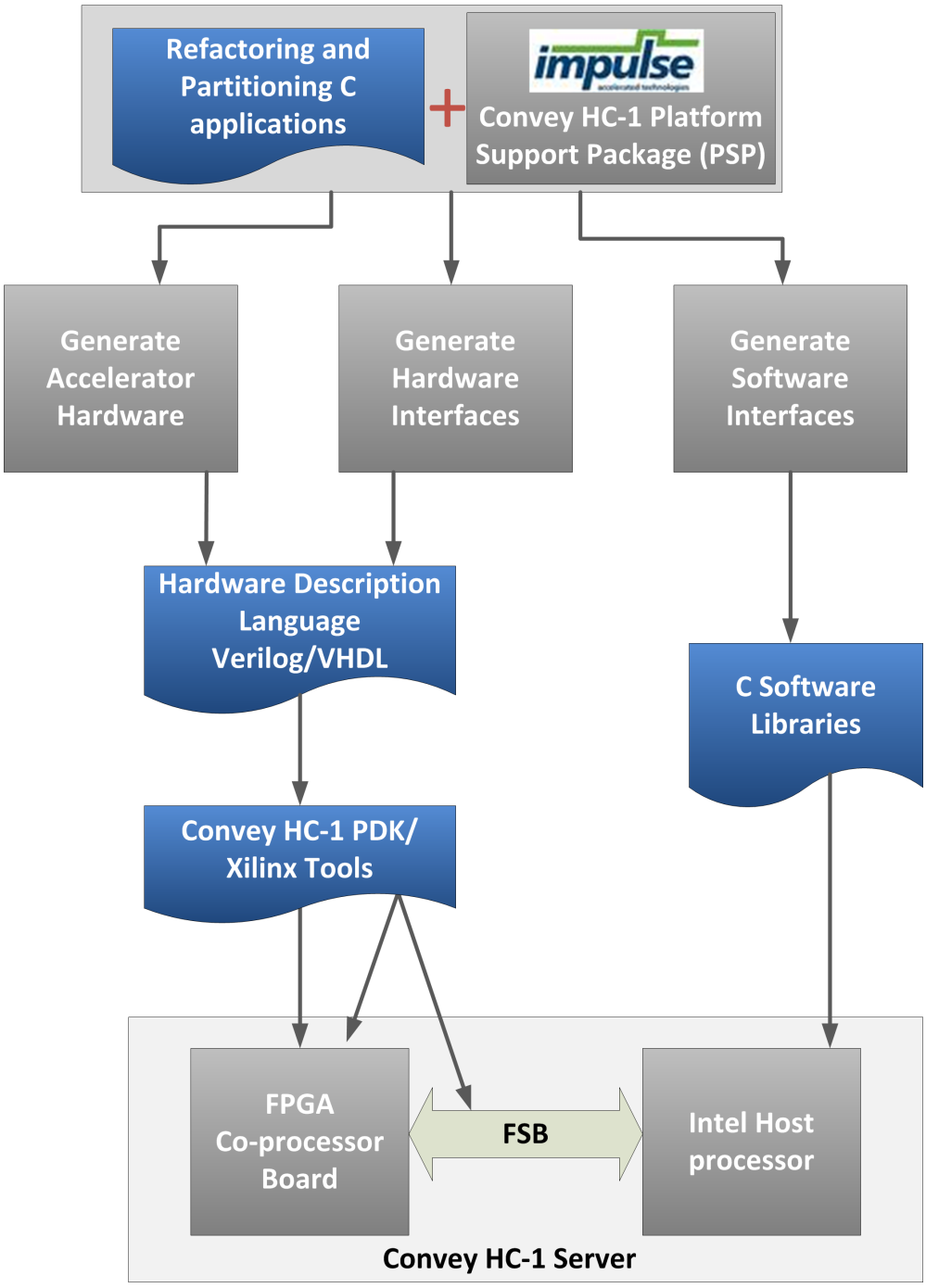


Figure 5.16 Developing personalties for the Convey HC-1 using ImpulseC.

to be loaded onto Convey HC-1's application engines. A basic memory block-copy and a streaming example was tested successfully using the PSP.

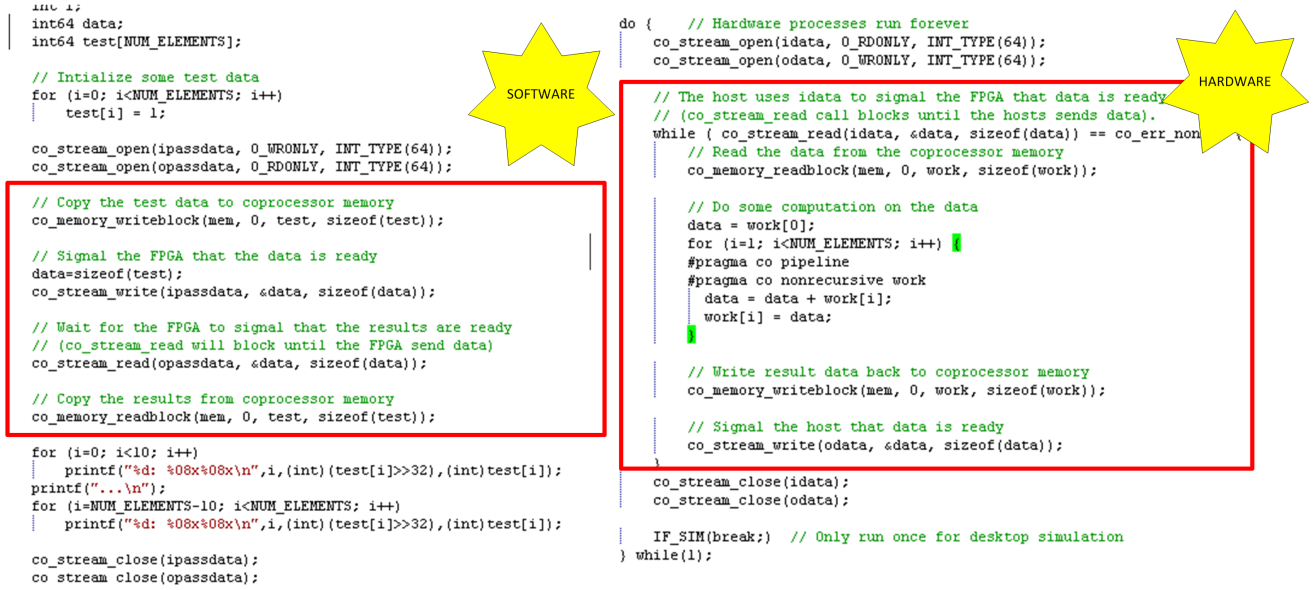


Figure 5.17 A snapshot of a ImpulseC code written for a memory block copy.

Figure 5.17 shows a snapshot of the hardware and software code written to transfer a basic block of data from the host to co-processor memory, perform some computation on the data, and return the result back to the host. It is interesting to see the minimal lines of code needed to implement this operation. This would probably take a few tens of lines of Verilog code to implement, including the complexity in understanding Convey's interfaces. The PSP in addition, handles the memory interfacing with co-processor memory. A simple *co_stream_read()* and *co_stream_write()* call enables streaming data between the host and FPGA. The *co_memory_readblock()* and *co_memory_writeblock()* are used on the hardware and software side to read and write data from/to co-processor or host memory. Intermediate buffering, memory addressing and hardware details are effectively abstracted away from the programmer.

5.4 Summary

This chapter focused on providing a detailed discussion of the results of the mapping of the FFT and the molecular dynamics application on various FPGA and GPU accelerators. GPUs clearly excel in the 1D floating-point FFT. FPGAs, on the other hand do well for the integer-point FFT. Speculations show that floating-point FFT performance on FPGAs can be improved if an appropriately balanced amount of key device resources are available. For the GEM application, however, the FPGAs map well compared to FPGAs at the expense of a large programming effort and compilation times. Finally, the last section addresses the productivity challenges of GPUs and FPGAs and efforts to alleviate them. The next chapter concludes the work, citing key challenges surfacing the adoption of FPGAs in HPC and proposing future work.

Chapter 6

Conclusions

The performance and power demands of applications in High-Performance Computing (HPC) have made *hybrid-core computing* inevitable. This thesis has provided a qualitative and quantitative comparison between FPGAs and GPUs, with respect to performance, power and programmability. Two key algorithms, the Fast Fourier Transform and a molecular dynamics application, were used to benchmark these high-end accelerators. This chapter presents a summary of the results of this work, followed by a brief mention of some of the challenges facing FPGAs in HPC, and finally proposing future work that can be built upon this thesis.

6.1 Summary of Results

The sustained floating-point FPGA performance for the FFT benchmark is less than the 50% of the estimated peak floating-point performance for the Virtex part, limited by "key" device resources. For instance, the DSP48Es are pivotal to floating-point butterfly computations and attaining higher frequencies using fewer programmable device resources, however, this

problem is solved on the latest FPGA devices. In addition, block RAMs are needed to cache reused data (i.e phase factors) for large FFT transforms. Finally, I/O banks are required to achieve greater external memory bandwidth. GPUs attain the same percentage of the peak performance and need architecture-specific optimizations to get higher performance [62]. The Convey HC-1 performs worse than the NVIDIA GTX 280, despite having four FPGAs and the same number of memory ports. This is primarily due to the high latency to external memory, arising from a low clock frequency, that limits the operation of the FFT cores. As the Pico M-503 system demonstrates, this low frequency translates to a high power efficiency, that makes it suited for power-constrained environments. In the Virtex-6 device, hardly 30% of the FPGA logic is utilized.

FPGA floating-point FFT performance can be improved by having an *appropriately balanced* number of DSP48Es, BRAMs, I/O banks and integrating faster memory parts. Estimates show that the Convey HC-1^{ex} [14], with a large Virtex-6 part can potentially deliver upto 350 GFLOPS for the 1D, floating-point FFT, owing to the abundant DSP48Es. GPUs having the highest external memory bandwidth and abundant floating-point units excel at the floating-point 1D FFT, as seen in the Fermi.

Integer-point FFT results for the FPGA show that reduced-precision can deliver almost 2-2.5X speed-up over the floating-point FFT, making it more power-efficient compared to GPUs. FPGAs also deliver much better performance for applications requiring bit-level operations. These become highly complex in GPUs, that use high-level language constructs and lack fine-grained control. This results in thread divergence and serialization of the various execution paths.

The performance of an unoptimized GEM implementation on the Convey HC-1 has shown roughly *200X* improvement over an optimized CPU implementation and an almost *2X* speed-up over an optimized GPU implementation. The ability to carefully lay out data in memory

and deeply pipeline operations have helped the Convey HC-1 maintain an advantage over GPUs for the N-body method. Despite its large power consumption, the Convey HC-1 server manages a higher power efficiency against GPUs.

6.2 Challenges facing FPGAs in HPC

Experience with two generations of FPGAs in this work has enabled speculation of key challenges facing FPGAs in HPC:

- *Design Productivity*: While high-level synthesis languages such as ImpulseC show promise in addressing front-end aspect of FPGA programmability, compilation times of large designs, common to HPC workloads, is the largest bottleneck to FPGA productivity. It reduces turns-per-day and aggravates debugging. Simulation is one way to get around initial runs of a design, but in-situ debugging is severely limited.

- *External bandwidth*: The percentage of peak performance that can be achieved in FPGAs, is highly dependent on a balance of device resources. For scientific workloads common in HPC applications, a sufficient number of DSP48E slices are needed to extract floating-point performance. DSP48Es are abundantly available in the latest Xilinx Virtex-7 FPGAs, but I/O banks are limited, severely affecting off-chip memory bandwidth. FPGAs also need a high-bandwidth interconnect when coupled with a host for streaming applications.

- *Price-performance ratio*: While FPGAs are cost-competitive with CPUs for floating-point performance [89], they still fall short to the price offered by GPUs. It is a well known fact that the big win for FPGAs over GPUs is not so much the FLOPS/dollar but the FLOPS/Watt or Watts/dollar. However, gaining wide acceptance in the HPC

community requires a reduction in procurement costs for FPGAs.

6.3 Future Work

Some future work that can be build upon this thesis includes the following:

- This thesis focused on two important applications in HPC. However, broader comparisons between FPGA and GPU accelerators can be made by studying the behavior of the other dwarfs of the benchmark suite having different compute and memory requirements. This will also help to determine the role of FPGAs in a possible future amalgamation of CPUs, GPUs and FPGAs.

- With a range of new FPGAs and GPUs being available, such as the Xilinx Virtex-7 family, Altera's Stratix-5, NVIDIA's GeForce 500M series; it would be worth revisiting the metrics for the new products. Finally, looking beyond the Convey HC-1, large-scale multi-FPGA configurations such as the Novo-G FPGA-based cluster [90], are possible candidates to realize even higher levels of computing power.

- The power consumption of high-performance architectures has raised economic and environment-related concerns. Thus, there is a potential need to explore HPC in power-constrained environments, in an effort to sustain green computing.

- This thesis used hand-optimized Verilog language for the implementations. Comparing an implementation using a high-level synthesis language (e.g ImpulseC) with the results in this work can help evaluate the effectiveness of the compilers and can provide performance and programmability trade-offs for time-sensitive designs.

Bibliography

[1] "The Beowulf Project." [Online]. Available: http://www.beowulf.org/overview/index.html

[2] B. Zagrovic, C. D. Snow, M. R. Shirts, and V. S. P, "Simulation of Folding of a Small Alpha-helical Protein in Atomistic Detail using Worldwide-distributed Computing," in *CODEN: JMOBAK ISSN*, 2002, pp. 323–927.

[3] L. Barroso, J. Dean, and U. Holzle, "Web Search for a Planet: The Google Cluster Architecture," *Micro, IEEE*, vol. 23, no. 2, pp. 22 – 28, march-april 2003.

[4] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879 –899, may 2008.

[5] P. Sundararajan, "High Performance Computing Using FPGAs," Tech. Rep., 2010. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp375_HPC_Using_FPGAs.pdf

[6] H.-J. Lee, J.-B. Lee, and H. Byun, "Comparisons of Fast 2D-DCT Algorithms for Parallel Programmable Digital Signal Processors," *High Performance Computing and Grid in Asia Pacific Region, International Conference on*, vol. 0, p. 209, 1997.

[7] M. Gschwind, F. Gustavson, and J. F. Prins, "High Performance Computing with the Cell Broadband Engine," *Sci. Program.*, vol. 17, pp. 1–2, January 2009.

[8] "The Top 500 Supercomputer Sites." [Online]. Available: http://www.top500.org/

[9] "Redefining Supercomputing," Cray Inc. [Online]. Available: http://www.cray.com/Assets/PDF/products/xk/CrayXK6Brochure.pdf

[10] J. Kepner, "HPC Productivity: An Overarching View," *International Journal on High Performance Computing Applications*, vol. 18, pp. 393–397, November 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1057159.1057161

[11] H. Michael Fieldman, "JP Morgan Buys Into FPGA Supercomputing." [Online]. Available: http://www.hpcwire.com/hpcwire/2011-07-13/jp_morgan_buys_into_fpga_supercomputing.html?featured=top

[12] Maxeler Technologies. [Online]. Available: http://www.maxeler.com/content/frontpage/

[13] S. Weston, J.-T. Marin, J. Spooner, O. Pell, and O. Mencer, "Accelerating the computation of portfolios of tranched credit derivatives," in *2010 IEEE Workshop on High Performance Computational Finance (WHPCF)*, nov. 2010, pp. 1 –8.

[14] *Convey's Hybrid-Core Technology: The HC-1 and the HC-1^{ex}*, Convey Corporation, November 2010.

[15] "The Graph500 List." [Online]. Available: http://www.graph500.org/june2011.html

[16] HPCWire, "Convey Computer Announces Graph 500 Performance." [Online]. Available: http://www.hpcwire.com/hpcwire/2011-06-16/convey_computer_announces_graph_500_performance.html

[17] K. H. Tsoi and W. Luk, "Axel: A Heterogeneous Cluster with FPGAs and GPUs," in *Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 115–124. [Online]. Available: http://doi.acm.org/10.1145/1723112.1723134

[18] G. Estrin, "Organization of Computer Systems-the Fixed Plus Variable Structure Computer," *International Workshop on Managing Requirements Knowledge*, vol. 0, p. 33, 1960.

[19] Algotronix, "The algotronix hybrid system." [Online]. Available: http://www.algotronix.com/company/Background.html

[20] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and Using a Highly Parallel Programmable Logic Array," *Computer*, vol. 24, no. 1, pp. 81 –89, jan 1991.

[21] M. Gokhale et al., "Splash: A Reconfigufable Linear Logic Array," in *Proceedings Conference Application-Specific Array Processing*, September 1990.

[22] T. Waugh, "Field Programmable Gate Array Key to Reconfigurable Array Outperforming supercomputers," in *Custom Integrated Circuits Conference, 1991., Proceedings of the IEEE 1991*, may 1991, pp. 6.6/1 –6.6/4.

[23] P. Bertin, D. Roncin, and J. Vuillemin, "Systolic Array Processors," J. McCanny, J. McWhirter, and E. Swartzlander, Jr., Eds. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989, ch. Introduction to programmable active memories, pp. 301–309. [Online]. Available: http://portal.acm.org/citation.cfm?id=88722.88789

[24] P. Athanas and A. Abbott, "Real-time Image Processing on a Custom Computing Platform," *Computer*, vol. 28, no. 2, pp. 16 –25, feb 1995.

[25] B. Fross, D. Hawver, and J. Peterson, "WILDFIRE(TM) heterogeneous adaptive parallel processing systems," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International and Symposium on Parallel and Distributed Processing, 1998*, mar-3 apr 1998, pp. 611 –615.

[26] N. Shirazi, P. M. Athanas, and A. L. Abbott, "Implementation of a 2-D Fast Fourier Transform on an FPGA-Based Custom Computing Machine," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*, ser. FPL '95. London, UK: Springer-Verlag, 1995, pp. 282–292. [Online]. Available: http://portal.acm.org/citation.cfm?id=647922.741016

[27] Xilinx, "CRAY XD1 Supercomputer Delivers Three Times More Power to Reconfigurable Computing Applications." [Online]. Available: http://www.xilinx.com/prs_rls/design_win/0591cray05.htm

[28] SGI, "SGI RASC Technology." [Online]. Available: http://www.ustar.ua/f/files/2496rasc_data.pdf

[29] "SRC IMPLICIT+EXPLICIT: Delivering Programmer-friendly Reconfigurable Processor-based Computers," SRC Computers. [Online]. Available: http://www.srccomp.com/techpubs/implicitexplicit.asp

[30] *Memory Interface Solutions User Guide, UG086*, Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf

[31] *Virtex-6 FPGA Memory Interface Solutions Datasheet, DS186(v1.7)*, Xilinx, March 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ds186.pdf

[32] "In-Socket Accelerators," Xtremedata. [Online]. Available: http://www.xtremedata.com/products/accelerators/in-socket-accelerator

[33] DRC Computers. [Online]. Available: http://www.drccomputer.com/pdfs/DRC_Accelium_Coprocessors.pdf

[34] Nallatech. [Online]. Available: http://www.nallatech.com/images/stories/product/facs/fsb-accelerator-module/FSB_Accelerators_v1-5.pdf

[35] Convey Corporation. [Online]. Available: http://www.conveycomputer.com/Resources/Convey_HC1_Family.pdf

[36] "SGI Extends High-Performance Computing Leadership with Reconfigurable Computing Technology," SGI. [Online]. Available: http://www.sgi.com/company_info/newsroom/press_releases/2005/september/rasc.html

[37] *Cray XD1 Datasheet*, Cray Inc. [Online]. Available: http://www.hpc.unm.edu/~tlthomas/buildout/Cray_XD1_Datasheet.pdf

[38] J. Richardson, S. Fingulin, D. Raghunathan, C. Massie, A. George, and H. Lam, "Comparative Analysis of HPC and Accelerator Devices: Computation, Memory, I/O, and Power," in *2010 Fourth International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)*, nov. 2010, pp. 1 –10.

[39] N. K. Govindaraju, S. Larsen, J. Gray, and D. Manocha, "A Memory Model for Scientific Algorithms on Graphics Processors," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[40] Z. Baker, M. Gokhale, and J. Tripp, "Matched Filter Computation on FPGA, Cell and GPU," in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2007.*, april 2007, pp. 207 –218.

[41] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. KrÃijger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*.

[42] B. Cope, P. Cheung, W. Luk, and L. Howes, "Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study," *IEEE Transactions on Computers*, vol. 59, no. 4, pp. 433 –448, april 2010.

[43] Y. Lee, Y. Choi, S.-B. Ko, and M. H. Lee, "Performance Analysis of Bit-width Reduced Floating-Point Arithmetic Units in FPGAs: A Case Study of Neural Network-Based Face Detector," *EURASIP Journal on Embedded Systems*, vol. 2009, pp. 4:1–4:11, January 2009. [Online]. Available: http://dx.doi.org/10.1155/2009/258921

[44] H. Fu, W. Osborne, R. G. Clapp, O. Mencer, and W. Luk, "Accelerating Seismic Computations Using Customized Number Representations on FPGAs," *EURASIP Journal on Embedded Systems*, vol. 2009, pp. 3:1–3:13, January 2009.

[45] B. Betkaoui, D. Thomas, and W. Luk, "Comparing Performance and Energy Efficiency of FPGAs and GPUs for High Productivity Computing," in *2010 International Conference on Field-Programmable Technology (FPT)*, dec. 2010, pp. 94 –101.

[46] H. Guo, L. Su, Y. Wang, and Z. Long, "FPGA-Accelerated Molecular Dynamics Simulations System," in *International Conference on Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09*, sept. 2009, pp. 360 –365.

[47] B. Whyte, "Virginia Bioinformatics Institute installs Convey's Hybrid-Core Computing Platform," May 2010. [Online]. Available: http://www.vbi.vt.edu/marketing_and_communications/press_releases_view/virginia_bioinformatics_institute_installs_conveys_hybrid_core_computing_pl

[48] *Tesla C2050 and C2070: GPU Computing Processor Supercomputing at 1/10th the Cost*, NVIDIA Corporation. [Online]. Available: http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf

[49] *NVIDIA GeForce GTX 200 GPU datasheet*, NVIDIA Corporation. [Online]. Available: http://www.nvidia.com/docs/IO/55506/GPU_Datasheet.pdf

[50] T. M. Brewer, "Instruction Set Innovations for the Convey HC-1 Computer," *IEEE Micro*, vol. 30, pp. 70–79, March 2010.

[51] J. D. Bakos, "High-Performance Heterogeneous Computing with the Convey HC-1," *Computing in Science Engineering*, vol. 12, no. 6, pp. 80 –87, nov.-dec. 2010.

[52] *Convey HC-1 Personality Development Kit Reference Manual, v4.1*, Convey Corporation, 2010.

[53] *Convey HC-1 Reference Manual, v2.0*, Convey Corporation, 2010.

[54] *Convey HC-1 Programmers Guide, v1.7*, Convey Corporation, 2010.

[55] *Convey HC-1 Mathematical Libraries User's Guide, v1.2.2*, Convey Corporation, 2010.

[56] J. Villareal, A. Park, R. Atader, W. Najjar, and G. Edwards, "Programming the Convey HC-1 with ROCCC 2.0," in *First Workshop on the Intersections of Computer Architecture and Reconfigurable Logic (CARL 2010)*, December 2010.

[57] "ModelSim- Advanced Simulation and Debugging." [Online]. Available: http://model.com/

[58] "VCS, Multicore-enabled Functional Verification solution," Synopsys. [Online]. Available: http://www.synopsys.com/tools/verification/functionalverification/pages/vcs.aspx

[59] "Pico M-503 Series modules." [Online]. Available: http://www.picocomputing.com/m_series.html

[60] *M-503 Hardware Technical Manual, Rev B*, Pico Computing Inc, 150 Nickerson Street Suite, 311, Seattle, WA 98109.

[61] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream Computing on Graphics Hardware," *ACM TRANSACTIONS ON GRAPHICS*, vol. 23, pp. 777–786, 2004.

[62] M. Daga, "Architecture-Aware Mapping and Optimization on Heterogeneous Computing Systems," Master's thesis, Virginia Tech, 2011.

[63] K. Group, "OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems." [Online]. Available: http://www.khronos.org/opencl/

[64] N. Corporation, "CUDA Parallel Programming Made Easy." [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[65] "ARM Introduces New Graphics Processor with OpenCL Support," November 2010. [Online]. Available: http://www.xbitlabs.com/news/graphics/display/20101110145839_ARM_Introduces_New_Graphics_Processor_with_OpenCL_Support.html

[66] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A View of The Parallel Computing Landscape," *Communications of the ACM*, vol. 52, pp. 56–67, October 2009.

[67] "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, 1985.

[68] R. Scrofano and V. Prasanna, "A Hierarchical Performance Model for Reconfigurable Computers," *Handbook of Parallel Computing: Models, Algorithms and Applications*, 2008.

[69] R. Scrofano, G. Govindu, and V. K. Prasanna, "A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing," in *Engineering of Reconfigurable Systems and Algorithms*, pp. 137–148.

[70] *LogiCORE IP Floating-Point Operator v5.0 Datasheet*, Xilinx Inc., March 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

[71] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for Floating-Point Double Precision and Beyond on FPGAs," in *International Workshop on Higly-Efficient Accelerators and Reconfigurable Technologies (HEART)*. ACM, Jun. 2010.

[72] "Floating Point Core (FloPoCo) Generator Project." [Online]. Available: http://flopoco.gforge.inria.fr/

[73] "The Expanding Floating-Point Performance Gap Between FPGAs and Microprocessors," HPCwire, November 2010.

[74] "The AMD Fusion Family of APUs," AMD Corporation. [Online]. Available: http://sites.amd.com/us/fusion/apu/Pages/fusion.aspx

[75] "Intel Unveils New Product Plans for High-Performance Computing," Intel Corporation. [Online]. Available: http://www.intel.com/pressroom/archive/releases/2010/20100531comp.htm

[76] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://dx.doi.org/10.2307/2003354

[77] 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 *Supercomputing, ACM/IEEE 2002 Conference*, nov. 2002, p. 50.

[78] Xilinx, *Xilinx LogicCore IP FFT*, 7th ed., March 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf

[79] *Virtex-6 FPGA Memory Interface Solutions User Guide, UG406*, Xilinx, June 2011.

[80] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) Benchmark Suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[81] "Watts Up? Power Meters," Wattsup Power Meters. [Online]. Available: https://www.wattsupmeters.com/secure/products.php?pn=0

[82] J. C. Gordon, A. T. Fenley, and A. Onufriev, "An Analytical Approach to Computing Biomolecular Electrostatic Potential. II. Validation and Applications," *The Journal of Chemical Physics*, vol. 129, no. 7, pp. 075 102+, 2008. [Online]. Available: http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=JCPSA6000129000007075102000001&idtype=cvips&gifs=yes

[83] J. G. Kirkwood, "Theory of Solutions of Molecules Containing Widely Separated Charges with Special Application to Zwitterions," *J. Chem. Phys.*, vol. 2, no. 7, pp. 351–361, 1934. [Online]. Available: http://dx.doi.org/10.1063/1.1749489

[84] R. Anandakrishnan and A. V. Onufriev, "An $N \log N$ Approximation Based on the Natural Organization of Biomolecules for Speeding up the Computation of Long Range Interactions," *Journal of computational chemistry*, vol. 31, no. 4, pp. 691–706, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1002/jcc.21357

[85] M. Daga, W. Feng, and T. Scogland, "Towards Accelerating Molecular Modeling via Multi-scale Approximation on a GPU," in *Proceedings of the 2011 IEEE 1st International Conference on Computational Advances in Bio and Medical Sciences*, ser. ICCABS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 75–80. [Online]. Available: http://dx.doi.org/10.1109/ICCABS.2011.5729946

[86] I. Lebedev, S. Cheng, A. Doupnik, J. Martin, C. Fletcher, D. Burke, M. Lin, and J. Wawrzynek, "MARC: A Many-Core Approach to Reconfigurable Computing," in *Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs*, ser. RECONFIG '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 7–12. [Online]. Available: http://dx.doi.org/10.1109/ReConFig.2010.49

[87] M. Owaida, N. Bellas, K. Daloukas, and C. Antonopoulos, "Synthesis of Platform Architectures from OpenCL Programs," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2011, pp. 186 –193.

[88] M. Lin, I. Lebedev, and J. Wawrzynek, "OpenRCL: Low-Power High-Performance Computing with Reconfigurable Devices," in *2010 International Conference on Field Programmable Logic and Applications (FPL)*, 31 2010-sept. 2 2010, pp. 458 –463.

[89] S. Craven and P. Athanas, "Examining the Viability of FPGA Supercomputing," *EURASIP J. Embedded Syst.*, vol. 2007, pp. 13–13, January 2007. [Online]. Available: http://dx.doi.org/10.1155/2007/93652

[90] "NOVO-G Overview." [Online]. Available: http://www.chrec.org/~george/Novo-G.pdf

Appendix A : Code listing to Read and write to Pico Bus from hardware

```
module PicoBus_ReadWrite(
input PicoClk,
input PicoRst,
input [31:0]     PicoRdAddr,
input [31:0]     PicoWrAddr,
input [127:0]    PicoDataIn,
input            PicoRd,
input            PicoWr,
output  reg [127:0] PicoDataOut = 0
);
wire RdAddrValid = PicoRdAddr[31:0] == 32'h10A00000;
wire WrAddrValid = PicoWrAddr[31:0] == 32'h10A00000;


reg [127:0] TheReg = 0;


always @(posedge PicoClk) begin
```

```
        if (PicoRst)
                        TheReg <= 128'h0;
        else if (PicoWr && WrAddrValid)
                        // The data being written is in PicoDataIn.
                        // Flip the bits and store for example.
                        TheReg <= ~PicoDataIn;
        if (PicoRd && RdAddrValid)
                        // Place the requested data on
                        // PicoDataOut on the next cycle.
                        PicoDataOut <= TheReg;
        else
                        PicoDataOut <= 128'h0;
end
endmodule
```

Listing 6.1 Code listing to Read and write to Pico Bus from hardware.

Appendix B : Verilog code listing for a BL=8 read and write in the Pico M-503

```
//Setting up Controller 0 to be for reading
assign c0_app_cmd = 3'b001;

//Setting up Controller 0 addresses
assign c0_tg_addr = q_c0_app_addr ;

//Read burst of 8. Increment address by same amount.
assign d_c0_app_addr = c0_app_rdy?q_c0_app_addr+8:q_c0_app_addr;

//Setting up Controller 1 to be for writing
assign c1_app_cmd = 3'b000;

//Setting up Controller 1 addresses
assign c1_tg_addr = q_c1_app_addr;

//Write burst of 8. Increment address by same amount.
assign d_c1_app_addr = (c1_app_rdy && dv)?q_c1_app_addr+8:q_c1_app_addr;

always @(*)
begin
```

```
nxt_write_state = r_write_state;
case(r_write_state)
    first:begin //c1_app_wdf_end state 0, first phase of the BL 8
            if(dv & w_c1_app_wdf_rdy)begin
                c1_app_wdf_wren = 1'b1;
                c1_app_wdf_end = 1'b0;
                nxt_write_state = 1;
            end
            else begin
                c1_app_wdf_wren = 1'b0;
                c1_app_wdf_end = 1'b0;
                nxt_write_state = 0;
            end
     end
     second:begin //c1_app_wdf_end state 1, second phase of the BL 8
            if(dv & w_c1_app_wdf_rdy)begin
                c1_app_wdf_wren = 1'b1;
                c1_app_wdf_end = 1'b1;
                nxt_write_state = 0;
            end
            else begin
                c1_app_wdf_wren = 1'b1;
                c1_app_wdf_end = 1'b1;
                nxt_write_state = 1;
            end
      end
endcase
end
```

Listing 6.2 Burst-length 8 read and write in the Pico M-503

Appendix C : Code listing for time-consuming section of GEM code

```
for (; eye < bound; eye++)
{
    /* offset point for calculation of d */
    vert_x = vert[eye].x + proj_len * vert[eye].xNorm;
    vert_y = vert[eye].y + proj_len * vert[eye].yNorm;
    vert_z = vert[eye].z + proj_len * vert[eye].zNorm;

    /* offset point for calculation of d' only */
    vert_xprime = vert[eye].x + ion_exc_rad * vert[eye].xNorm;
    vert_yprime = vert[eye].y + ion_exc_rad * vert[eye].yNorm;
    vert_zprime = vert[eye].z + ion_exc_rad * vert[eye].zNorm;
    vert[eye].potential = 0;

    for (k = 0; k < nres; k++)
    {
        natoms = residues[k].natoms;
        for (j = 0; j < natoms; j++)
        {
```

```
                /* distance from point to charge */
                d = sqrt((vert_x - residues[k].atoms[j].x)^2 +  (vert_y -
                    residues[k].atoms[j].y)^2 + (vert_z - residues[k].atoms[j
                    ].z)^2);
                /* distance from point to charge */
                dprime = sqrt((vert_xprime - residues[k].atoms[j].x)^2 +  (
                    vert_yprime - residues[k].atoms[j].y)^2 + (vert_zprime -
                    residues[k].atoms[j].z)^2);
                defs.Asq_by_dsq = A*A * d*d;
                //Potential Contribution calculation
                one_over_one_plus_kappa_rprime = 1./(1. + (defs->kappa *
                    rprime));
                sum1 =  1. / (d * diel_int);
                sum2 =  (1./diel_int - 1./diel_ext) / (defs->
                    one_plus_alpha_beta * A);
                sum3 =  defs->Asq / sqrt(defs->Asq_minus_rnautsq*defs->
                    Asq_minus_rsq + defs->Asq_by_dsq);
                salt =  defs->inverse_one_plus_ab_by_diel_ext* (defs->
                    one_plus_alpha/dprime*(1./(1.+defs->kappa*dprime)-1.)
                        +defs->alpha_by_one_minus_beta/rprime*(1.-
                            one_over_one_plus_kappa_rprime));
                vert[eye].potential += (sum1 - sum2*sum3 + salt) * charge;
            }
        }
    }
```

Listing 6.3 Time-consuming section of GEM code after running through GPROF