Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 241439, 10 pages
doi:10.1155/2012/241439

Research Article

The "Chimera": An Off-The-Shelf CPU/GPGPU/FPGA Hybrid Computing Platform

Ra Inta, David J. Bowman, and Susan M. Scott

The Centre for Gravitational Physics, Department of Quantum Science, The Australian National University, Canberra, ACT 0200, Australia

Correspondence should be addressed to Ra Inta, ra.inta@anu.edu.au

Received 2 October 2011; Revised 4 January 2012; Accepted 4 January 2012

Academic Editor: Thomas Steinke

The nature of modern astronomy means that a number of interesting problems exhibit a substantial computational bound and this situation is gradually worsening. Scientists, increasingly fighting for valuable resources on conventional high-performance computing (HPC) facilities—often with a limited customizable user environment—are increasingly looking to hardware acceleration solutions. We describe here a heterogeneous CPU/GPGPU/FPGA desktop computing system (the "Chimera"), built with commercial-off-the-shelf components. We show that this platform may be a viable alternative solution to many common computationally bound problems found in astronomy, however, not without significant challenges. The most significant bottleneck in pipelines involving real data is most likely to be the interconnect (in this case the PCI Express bus residing on the CPU motherboard). Finally, we speculate on the merits of our Chimera system on the entire landscape of parallel computing, through the analysis of representative problems from UC Berkeley's "Thirteen Dwarves."

1. Computationally Bound Problems in Astronomical Data Analysis

Many of the great discoveries in astronomy from the last two decades resulted directly from breakthroughs in the processing of data from observatories. For example, the revelation that the Universe is expanding relied directly upon a newly automated supernova detection pipeline [1], and similar cases apply to the homogeneity of the microwave background [2] and strong evidence for the existence of dark matter and dark energy [3]. Most of these discoveries had a significant computational bound and would not have been possible without a breakthrough in data analysis techniques and/or technology. One is led to wonder the astounding discoveries that could be made without such a computational bound.

Many observatories currently have "underanalyzed" datasets that await reduction but languish with a prohibitive computational bound. One solution to this issue is to make use of distributed computing, that is, the idle CPUs of networked participants, such as the SETI@HOME project [4]. It is clear that a number of common data analysis techniques are common across disciplines. For example, LIGO's Einstein@HOME distributed computing project, designed to search gravitational wave data for spinning neutron stars, recently discovered three very unusual binary pulsar systems in Arecibo radio telescope data [5].

These are far from the only "underanalyzed" datasets from existing observatories, and this situation is expected to only compound as we look forward to an ever increasing deluge of data. For example, the Square Kilometer Array is expected to produce about an exa-byte *a day* [6] (to put this into perspective, it is estimated that *all* the stored information created by humanity is roughly 300 EB [7]); just for fairly basic operation, this alone will require to be close to the projected computing power of the world's single most powerful computing system [8] at the expected 2020 date of "first light." There are a number of robotic survey telescopes either already, or scheduled soon to be, on-line to detect transient events, from near-Earth objects such as asteroids [9] to distant GRBs [10].

It should be obvious that many other sectors have exponentially growing appetites for computation, from military [11] through financial [12], even cinematic applications require the most powerful HPC systems [13]. Considering

the common computational requirements of these systems, it is clear that a revolution in HPC technology is required in order to keep pace with projected needs.

2. Problems with Conventional Cluster-Based HPC Systems

Until very recently, the most powerful HPC systems (the "Top 500") were purely CPU based [8], although there is a very recent but significant shift towards the use of general-purpose graphical processor unit (GPU) coprocessing. However, many critics point out that the "Top 500" may not be representative of the true compute power of a cluster [14]. The negative corollary of the efficient compliance of CPUs with LINPACK is that the resulting rigid instruction set can compromise performance when performing operations not heavily dependent on dense linear algebra. As we show in Section 6 below, this is, only one feature of many that are vital to problems involving parallel computation.

Because power consumption, and hence heat generation, is proportional to clock speed, processors have begun to hit the so-called "speed wall" (e.g., Intel cancelled their 4 GHz processor line because of heat dissipation issues [15]). Furthermore, there is a growing awareness of the monetary cost of powering traditional HPC systems: over half the lifetime cost of a modern supercomputer is spent on electrical power [16]. Indeed, many computing clusters are sited near large generation plants in order to save on power transport costs [17].

It is also easy to forget that the concept of a "general purpose" microprocessor is a relatively recent idea—for example, only since the introduction of the 486 processor have Intel not used a dedicated floating point coprocessor. Meanwhile, hardware accelerators have occupied specialized niches, such as video processing and high-speed DSP applications. The most commonly available of these accelerators are the (general-purpose) graphical processor unit (GPU) and the FPGA. Easing or circumventing many of the issues with CPU-based HPC is becoming an attractive prospect to a growing cadre of consumers. Currently promising teraflop/s performance with a low initial capital outlay and without need of a specialized power supply or cooling facility, both the GPU and the FPGA are viable alternatives to purchasing many CPU-based units. Until recently, the development time associated with these platforms was considered rather high, especially for the FPGA. However, the programming interfaces for both have become more user-friendly. Finally, the future appears bright for both accelerator classes, as both have performance growth well exceeding Moore's Law, and consequently, that of CPUs [18].

3. Comparisons between CPUs and Hardware Accelerators

The long-standing workhorse of the vast majority of data analysts is the general-purpose CPU. Because it is expected to perform a range of different tasks, CPU processor designs cannot afford to specialize. Although the processor clock speeds are fairly high, it can take many cycles to perform an intensive computation, because it will be scanning for interrupts, and so forth. CPUs are generally very good at performing a multitude of separate tasks at a moderate speed and are efficient at moderating/coordinating a range of slave devices. The performance and merits of these devices should be fairly familiar to the reader.

Because GPU platforms were designed to efficiently perform linear operations on vectors and matrices, a general rule of thumb is that any operations that require intensive linear calculations are best made on a GPU. Many embarrassingly parallel computations rely on linear algebraic operations that are a perfect match for a GPU. This, in addition to the amount of high level support, such as C for CUDA, means they have become adopted as the hardware accelerator of choice by many data analysts. For example, a comparison of the GPU-based CUDA-FFT against the CPU-based FFTW3 on gravitational wave data analysis showed a 4X speedup for one million, and 8X for four million, points; this exact approach can also be used to detect radio transients with synthetic aperture array radio telescopes [19]. An excellent analysis of a range of algorithms heavily used in astronomy with applications including imaging, gravitational lensing, and pulsar characterization, implemented on GPUs, is given in [20].

FPGAs, on the other hand, of course, represent an entirely different approach to computing altogether. Because of their unrivalled computing flexibility, it can be difficult for the data analyst, used to a rigid instruction set-based processor, to be entirely comfortable with the low level required to construct an analysis pipeline. The majority of the applications in astronomy have been in instrumentation and data capture, such as the FPGA-based digital cross-correlator for synthetic aperture array radio telescopes [21, 22] or spectrometers [23]. However, there is a small but growing base of analysts willing to adopt an FPGA-based hardware acceleration solution, with applications including detection of gamma-ray pulsars [24]. There are a number of FPGA-based HPC facilities such as Janus [25] and Maxwell [26], and companies such as Starbridge, Inc. and Pico Computing, Inc. [11] offering FPGA computing solutions. A good survey of the state of FPGA-based HPC is given in [27].

Determining the relative strengths of each hardware acceleration class is highly algorithm (and data-type) dependent. There are many comparisons in the literature between FPGA, GPU and CPU, implementations of the same algorithms, ranging from random number generation [28] (where at 260 Gsample/s, FPGAs were found to be faster by a factor of 15 and 60 over a contemporaneous GPU and CPU resp.), video processing [29] (where FPGAs may have a significant advantage over GPUs when multiplying arrays of rank four or higher), convolution [30] (FPGAs are advantageous because of their pipelining and streaming abilities) to MapReduce [26, 31] (where the former show that GPUs considerably out-perform FPGAs and the latter show that an FPGA implementation of Quasi-Random Monte Carlo outperforms a CPU version by two orders of magnitude and beats a contemporaneous GPU by a factor of three), and least squares applications [32] (an FPGA implementation is

slightly worse than that for a GPU, which is in turn slightly worse than a CPU, for large matrices). Some more general overviews are given in [33, 34]. However, one must be careful not to generalize performance without consideration of the detailed technical specifications of the components. Both FPGA and GPU platform designs are in a state of unprecedented flux, and hence relative performance benchmarks are likely to change also. This caveat notwithstanding there are a number of distinctions amongst each hardware platform intrinsic to the underlying design features.

With the above considerations in mind, we present here a system that attempts to exploit the innate advantages of all three hardware platforms, in order to attack problems with a computational bound that would benefit from a mixed hardware subsystem "heterogeneous" approach.

4. The "Chimera" Heterogeneous CPU/GPU/FPGA Computing Platform

We originally conceived a platform that would exploit the advantages of both FPGA and GPU accelerations *via* a high-speed backplane interconnect system [35]. A schematic is shown in Figure 1.

There are a number of platforms that implement a heterogeneous FPGA/GPU/CPU system. The "Quadro-Plex Cluster" [36] is a sophisticated sixteen node cluster with two 2.4 GHz AMD Opteron CPUs, four nVidia Quadro FX5600 GPUs, and one Nallatech H101-PCIX FPGA in each node, with a thread management design matching that of the GPUs. However, it does not yet appear to implement a combination of FPGAs *and* GPGPUs within an algorithmic pipeline, which is essential for our applications. Also, the FPGA architecture was designed to mirror that of the microprocessor, including full double precision support, which in general will not optimize the potential performance of the FPGA. The "Axel" [37] system is a configuration of sixteen nodes in a Nonuniform Node Uniform System (NNUS) cluster, each node comprising an AMD Phenom Quad-core CPU, an nVidia Tesla C1060, and a Xilinx Virtex-5 LX330 FPGA. From the perspective of Axel, our proposed platform would conform to the Uniform Node Nonuniform System (UNNS) configuration, or perhaps an optimized version of each node within an Axel-type cluster. There is a "desktop supercomputer" comprising a single CPU, GPU, and FPGA [38], used to model coupled resonator optical waveguides, although unfortunately the architecture and configuration of this system is unclear. Finally, there is a fledgling system that proposes to use a combination of GPUs and FPGAs for cryptoanalytic problems [39], although to date the applications have only been tested on GPUs.

We also would like the system to be scalable if possible, although the focus of this paper is the combined heterogeneity of the hardware accelerators in a single-node architecture. The main problems we are concerned with here feature embarrassingly parallel analysis pipelines, and so extrapolation to a cluster system ought to be relatively straight forward. The granularity of this system is dictated by the particular algorithm being used; the most coarse-grained pipelines will be those with a large reliance on GPU-based resources.

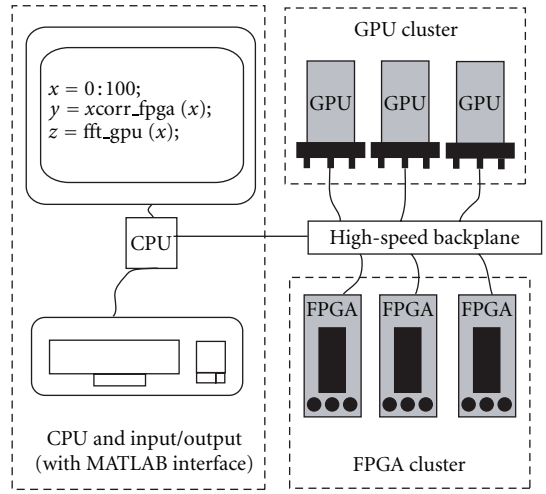

Figure 1: A schematic of our original concept for a heterogeneous CPU/GPGPU/FPGA system.

Table 1: Description of subsystem hardware configuration for the "Chimera" heterogeneous CPU/GPGPU/FPGA computing platform used here.

Subsystem	Vendor	Model
CPU	Intel	i7 Hexacore
GPGPU	nVidia	Tesla C2070
FPGA	Altera	Stratix-IV

Finally, a significant constraint was an inexpensive initial outlay, which immediately restricted us to commercial-off-the-shelf (COTS) components. Table 1 lists the components of the heterogeneous system we describe here, which we call "Chimera," after the mythical Greek beast with a head of a goat, a snake, and a lion on the same body.

Aside from the actual CPU, FPGA, and GPU components, perhaps the most important element in this system is the high-speed backplane or interconnect. Because of COTS constraints, we eventually settled on the simplest solution, that is, that already residing as the northbridge system on the CPU motherboard. In this case, the interconnect protocol is Peripheral Component Interconnect express (PCIe) Gen 2.0, on an Intel DX58SO2 motherboard. Although the board has 2X PCIe 2.0 $\times$ 16 ports, only one 16 lane port is dedicated, the other is multiplexed with a third 8 lane slot. The maximum theoretical PCIe throughput, is therefore, 2 $\times$ 16 lane devices or 32 GB/s. These 32 PCIe lanes are routed to the I/O hub processor (the 82X58IOH, which we loosely term here a "northbridge") which implements Intel's Quick Peripheral Interconnect (QPI) protocol to the CPU. The QPI has a 25.6 GB/s point-to-point link to the CPU. In spite of the impressive performance of the motherboard we found, as expected, the PCIe bottleneck presented the most significant limitation to our computing model. We choose to ignore this limitation in what follows for several reasons.

(1) The limitation is algorithm dependant, in some cases (e.g., generation of pseudorandom numbers) large-data sets are developed and processed solely on-chip. In other cases, processing pipelines may be organized to avoid this bottleneck.

(2) FPGA devices, in particular, are provided with very high speed I/O connections allowing multiple FPGAs to process and reduce data-sets before passing them to the final, PCIe limited device.

(3) The purpose of the Chimera is to prove the concept of the hybrid computing model using low-cost COTS devices. Having established that a hybrid design is limited only by interconnect speed the way is open for faster and more expensive interconnect solutions.

The Altera Stratix-IV FPGAs reside on DE-530 development boards, with an 8-lane PCIe interface, while the Tesla C2070 has a 16-lane PCIe interface. The development environment for the FPGAs was Verilog and ModelSim, while that for the GPUs was C/C++ for CUDA and nVidia's SDK libraries. Because there is not yet a widely available communication protocol that allows FPGA-GPU communication without the mediation of a CPU, we are currently developing kernel modules for the PCIe bus. A primary goal of the Chimera system is to provide access to high-performance computing hardware for novice users. This inherently means providing an operating system (OS) with familiar signal processing tools (e.g., MATLAB). Running the OS is naturally a task exclusively handled by the CPU, but it presents some difficulties because the security layers of the OS will usually deny direct access to the FPGA and GPU hardware. In the case of the GPU, this problem is solved by the vendor (nVidia) provided drivers, but, for the FPGA, an alternative approach is necessary: custom driver development is necessary. We opted for a Linux-based OS because we feel it is much simpler to develop drivers than for proprietary OSs. Like all modern OSs, GNU/Linux implements a virtual memory environment that prevents "user" code from directly accessing memory and memory-mapped peripherals. In order to directly control system hardware, such as a PCIe device, it is necessary to write code that runs in "kernel" mode. Linux permits these "kernel modules" to be loaded and unloaded into the running kernel *via* the `insmod` and `rmmod` commands. This provides a straightforward means to share data between the FPGA and GPU devices on the PCIe bus, as well as the system memory. A schematic of this stack design is given in Figure 2.

It is also possible to rebuild the Linux kernel, thus incorporating the module into a custom Linux kernel. The kernel code then provides a bridge between the user code space and the FPGA hardware. As these kernel modules are still under development, and the bottleneck from the PCIe transfer is simple to calculate for the simple examples below, we consider here the subsystem performance only. Hence, the run-time and data-transfer systems are currently fairly primitive. In order to optimize performance, the parallel programming models are algorithm dependent, including the number of threads and how data is shared. Data transfer between the subsystem components currently has only limited automation, and pipelines are mediated by the CPU, which is highly undesirable. However, we are in the process of building data transfer protocols (*via* the kernel modules) that depend upon the particular application but are implemented using a coherent level of abstraction.

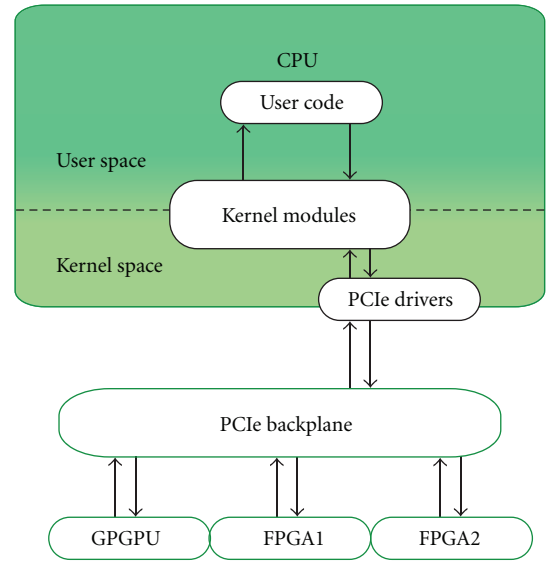

Figure 2: The software stack hierarchy of the Chimera system. The Linux kernel modules provide an interface between user-generated code and the PCIe bus and hardware components.

Finally, this system was conceived with the explicit consideration of the significant trade-off between development time (especially associated with development of the FPGA related pipeline and PCIe modules) and performance. Hence, we consider only computing solutions that are likely to have a high reuse within the data analysis community.

5. Appropriate Algorithms for a Heterogeneous Computing System

As with most compute intensive problems, the implementation is extremely dependent on the underlying hardware and the most significant bottlenecks. Much consideration is required in the "technology mapping" from one system to another. For example, an important consideration for FPGA HPC applications is whether fixed or floating point calculations are required. Data analysts have become accustomed to assume that their pipeline requires single precision or more, without considering that this may actually *decrease* the absolute error of the calculation. It is also not entirely true that FPGAs perform poorly on floating point performance [40] (note that this is not necessarily the same as conforming exactly to the IEEE-754 single or double precision definition [41]; GPUs do not natively support denormals but now perform linear operations very well with most double precision calculations). Considering around 50% of FPGA logic can be consumed by tracking denormals, there are recent ingenious developments, optimizing floating point pipelines on FPGAs, such as Altera's "fused datapath" approach [42].

We consider here three important algorithms to attempt to illustrate the considerations required, and relative benefits of, mapping common problems to our heterogeneous computing system.

5.1. Monte Carlo Integration. The simplest illustration of the advantages of this system is the well-known Monte Carlo calculation of π (the ratio of the circumference of a circle to its diameter) [43]. This proceeds as follows. A large set of points (x, y) are created from pairs of random numbers, uniformly distributed between -1 and $+1$, such that they fall within a square of area $2 \times 2 = 4$ units^2. Each point is then checked to see if it lies within a circle of unit radius (equivalently solving the inequality $x^2 + y^2 < 1$). The ratio of the number of points satisfying this inequality to the total number of points then determines the ratio of the total area to the area of the circle. Yet the area of a circle of unit radius is π, so the ratio will yield a numerical estimate of $\pi/4$.

Because FPGAs have the unrivalled capacity to generate large quantities of uniformly distributed random numbers, and GPUs are vastly superior at taking squares (or square roots) over any other commonly available general purpose platform, it would be sensible if an FPGA were to generate the randomly placed points (in parallel), which were in turn directly given to a GPU to determine the placement of each point (i.e., within the circle or not). This calculation and how we anticipate performing it using a hybrid approach is depicted in Figure 3. We could implement a more sophisticated quasirandom version of this algorithm, that is, using a Halton, Sobol' or other low-discrepancy sequence [26, 43], but considering this a rather poor way to calculate π, we restrict ourselves here to uniform pseudorandom distributions for illustrative purposes.

The pseudorandom calculation of π was implemented entirely in the GPU and entirely in the FPGA. In both cases, two 32 bit unsigned integers ((x, y) pairs) were generated using a pseudorandom generator. These were squared and added and the result compared to unity. The limiting factor in the StratixIV530 FGPA was the available number of multiplier blocks which were necessary for the square operation. Our design required 5 DSP18 blocks per sampling module, allowing a total of $\approx$200 parallel units. Applying a conservative clock speed of 120 MHz, we achieved 24 giga-samples per second. Surprisingly, this is approximately an order of magnitude greater than our results for the GPU, which performed 100,000 trials in 47 μsec, giving $\approx$2.13 giga-samples per second. These results agree broadly with those in [26], in that the FPGA calculation of the entire pipeline is about an order of magnitude faster than the GPU implementation, and many many times faster than that by a comparable CPU.

This result ought to be surprising, *prima facie*, considering that the multiplication intensive calculations involved in testing whether the points lie within the unit circle ought to favor the GPU. However, one must remember that a considerable amount of effort went into an implementation optimized for the FPGA, including the avoidance of the costly square root operation. We expect the GPU to perform a lot better, in relative terms, when the function to be integrated is more complicated than this extremely simple model. This argument, of course, ignores the fact that copying memory to devices takes far longer than these simple calculations (e.g., in the case of the GPU, about 42 times as long, at 2,118 μsec).

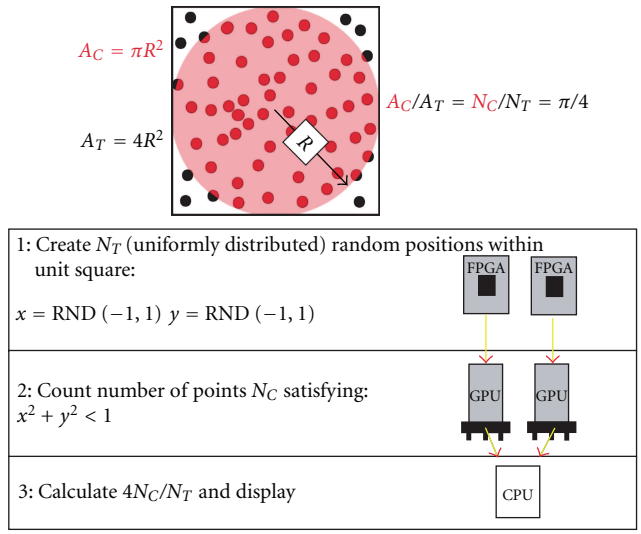

FIGURE 3: A schematic of Monte Carlo calculation of π, and how we expect to perform this calculation on a hybrid GPU/FPGA system.

5.2. Normalized Cross-Correlation. Template matching is one of the most important tasks in signal processing and is often achieved by computing the cross-correlation (or "sliding-dot product") between the template and a search signal. For example, cross-correlation is a crucial operation for resolving images in synthetic aperture array-based observatories such as the Very Long Baseline Array or the proposed Square Kilometre Array (SKA) [44].

The point in the search signal with maximum correlation is considered the best match to the template. For discrete one-dimensional signals, we seek to find $\max(A \times B[t])$, where $A \times B[t] = \sum_T A[T] * B[T + t]$. Essentially, we are treating the template, and a template-sized window of the search signal, as vectors in N-dimensional space (where N is the length of the template) and computing their dot-product. The Pythagorean relationship shows that, for vectors of equal length, this is simply a measure of the angular relationship between them, since $\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}| \cdot |\mathbf{B}| \cos\theta$.

Normalized cross-correlation (NCC) is so called because it divides the cross-correlation by the product of the magnitude of the vectors, thus calculating $\cos\theta$, regardless of the length of the vectors. NCC is widely applied in image processing, such as video compression, pattern recognition, and motion estimation where the signals are, of course, two-dimensional, real, and nonnegative. For image processing applications, the 2DNCC is given by (1):

$$\sum_X \sum_Y \frac{A[x, y] \cdot B[x + X, y + Y]}{\sqrt{A^2 B[X, Y]^2}}. \quad (1)$$

Assuming the pixel data in question is always an integer and, since $\sqrt{A^2B^2} \leq A^2B^2$, we can equally express (1) as (2):

$$\sum_X \sum_Y \frac{A[x,y] \cdot B[x+X, y+Y]}{A^2 B[X,Y]^2}, \tag{2}$$

provided we are only interested in finding the peak of the NCC surface.

Computationally, there are many options to accelerate (2) [45]. For most real-world datasets, calculation of the numerator is faster if the Fourier shift theorem is applied and the (unnormalized) cross-correlation computed via the usual transform multiply and inverse transform approach (although it is important to remember that in order to prevent wrap-around pollution it is necessary to zero-pad both images to be size(A) + size(B) − 1). The multiply-intensive nature of this approach, combined with the all-but-essential use of floating-point data types, leads us to conclude that the numerator will be best computed with the GPU.

The ideal platform to calculate the denominator, however, is not so easily assured. As the image data is integral and the only operators are the sum and square, which are both closed under the set of integers, this would seem an ideal candidate for an FPGA. However, because $\mathbf{B}^2 \equiv \mathbf{B} \cdot \mathbf{B}$, it is also a task which might be well handled by a second GPU.

We evaluated the 2D NCC using a common image matching task. We exhaustively searched a 1024×768 pixel search image for an 8×8 template. In both cases, the pixels used were 16 bit unsigned greyscale integers. The numerator of (2) was calculated in the Fourier domain on the C2070 GPU and found to take 6.343 ms (≈158 frame/s). We then investigated two approaches to calculating the denominator: FPGA based, or using a second GPU. The FPGA implementation of the dot-product relied on the multiply-accumulate pipeline built into the DSP18 blocks in the StratixIV FPGA. Four DSP18s were required for each 8×8 dot-product block and, because very few of the FPGA's other resources were required, this was the limiting factor in determining the number of parallel units. For the StratixIV530 device, this allowed 256 dot-product modules to operate in parallel. As the DSP18 is a hard silicon block, the maximum clock speed was relatively high, at slightly over 400 MHz, giving a total value of roughly 10 giga-ops. For the search image size quoted, this would achieve ≈12.5 kframes/s. The GPU dot-product was profiled at one operation in about 1.4 nsec, or 715 Mop/s, corresponding to a frame rate of ≈894 frames/s.

Thus, in this implementation, ignoring the impact of the PCIe bottleneck, a hybrid system comprising a GPU working in tandem with an FPGA was found to achieve a better result than a system consisting of two GPUs. We would like to apply a similarly optimized algorithm to the correlation problem required by a synthetic aperture array observatory such as the VLBA or SKA [44].

5.3. Continuous Gravitational Wave Data Analysis. A much more complicated, and consequently useful, example is that related to the computationally bound problems found in gravitational wave data analysis. A number of mechanisms may cause rotating neutron stars to emit periodic distortions of space time (gravitational waves), which may be detected by ground-based gravitational wave observatories such as LIGO [46] or Virgo [47]. The data analysis pipeline is as follows: the data coming from the gravitational wave observatories is filtered, then template matching is applied. If no candidate is found, in order to determine the statistical upper limits, intensive Monte Carlo calculation of injections is required. This stage is so computationally intensive that it is often not fully implemented because of the prohibitive computational cost [48]. We estimate the Chimera platform will be able to provide these limits for approximately a dozen potential neutron star targets, including central objects in supernova remnants of unknown pulsation frequency. There are already GPU-based acceleration of similar pipelines used in gravitational wave data analysis; many FFTW3-based routines have been replaced with CUDA FFT, to enable low-latency detection of gravitational waves from coalescing neutron stars [19]. We see a similar pipeline as a natural application for the Chimera system we describe here, and we are in the process of implementing this.

5.4. Other Promising Algorithms. Many promising data analysis applications that are considered computationally expensive may be implemented rather simply using this type of heterogeneous hardware acceleration. For example, digital filter application is efficiently implemented on FPGA devices [49]. A number of promising analysis techniques based on compressed sensing [50] are considered computationally expensive. However, the most compute-intensive component, namely, the least-squares minimization routine, may be efficiently implemented on an FPGA via Cholesky decomposition [51–53].

6. Potential for Other Data Analysis Applications: Analysis via Berkeley's "Thirteen Dwarves"

Probably the most labor-intensive process involved in choosing the most appropriate platform weighting between the hardware accelerators in a heterogeneous system is that of identifying the most appropriate algorithms—especially their most efficient implementations—for a given pipeline and input/output constraints. Although we have identified a small number of algorithms here, it would be an interesting and valuable exercise to consider the possible classifications to determine which hardware acceleration combinations would be most appropriate.

In practice, there is a natural classification of problems merely by virtue of the similarity in computation and data movement through usage of the same software packages. Take, for example, FFTW/FFTW3 for spectral methods, or the dense linear algebra LAPACK/ScaLAPACK libraries. Indeed, the latter forms an important benchmark providing some measure of CPU performance (and possible entry into the popular "Top 500" list). However, we would like a more systematic approach to benchmark performance on parallel algorithms that are not necessarily strongly dependent on linear algebra.

In order to identify the likely future requirements of software and hardware design, Phil Colella identified seven parallel numerical methods important for science and engineering [54], known as "dwarves" (a reference to the Snow White fairy tale). This concept of a dwarf, as an "algorithmic method encapsulating a pattern of computation and/or communication," was further extended by the Department of Computer Science of UC Berkeley, to thirteen [55]. It is intended these "dwarves" comprise an algorithm classification system spanning the entire landscape of parallel computing for the foreseeable future ([55]; see Table 2). We see this approach as a promising means of determining the relative (dis)advantages of our Chimera system on other scientific problems. To our knowledge, this is the first attempt at the analysis of dwarf performance on systems heterogeneous at the hardware accelerator level.

The example of the Monte Carlo calculation of π above falls under the "MapReduce" Dwarf: the Mapper function is each trial point, while the reducer just aggregates the counts. From the Chimera perspective, the mapper functions are either performed on the FPGA (the simple example above) or split between the FPGAs and the GPU, while the reducer runs on the CPU. The normalized cross-correlation would be classified within the "Dense Matrix" dwarf. The much more complicated case of the full gravitational wave analysis pipeline largely falls under the "Spectral Methods" jurisdiction, along with that of "MapReduce."

It is clear that many dwarves naturally map to each of the separate hardware accelerators. For example, the "Dense Matrix" dwarf equivalently relates to LAPACK/ScaLAPACK performance on a problem such as principal component analysis of a dense structure. Here, we should expect different performance for floating and fixed point operations, and hence we expect the GPU to excel alone on floating point (at least for matrices of up to rank 4 [29]), while the FPGA will be extremely competitive for fixed point versions. The same argument applies for naïve implementations of the "Sparse Matrix" dwarf, we expect GPUs to have superior performance calculating a sparse PCA problem in floating point, while the FPGA ought to well on fixed point. The "Spectral" dwarf generally comprises an FFT-based computation, such as wavelet decomposition. It is difficult for any platform to beat the GPU CUFFT library, and hence the GPU will be superior in most implementations, although for large numbers of FFT points, FPGAs may be more appropriate [40].

On the other hand, "Combinational Logic" problems (dwarf 8) such as hashing, DES encryption, or simulated annealing, are extremely well suited for the logic-intensive FPGA. It is also clear the "Finite State Machine" dwarf (13), such as control systems, compression (e.g., the bzip2 function), or cellular automata, can be most easily optimized by an FPGA. For example, consider a simple implementation of a 4-bit TTL (Transistor-Transistor Logic) counter, requiring 4 XOR gates, 3 AND gates, and 4 1-bit registers, our Stratix-4 could produce $\approx$120 k operations per clock cycle, or roughly 36 peta-op/s.

What are not so clear are problems requiring conditional elements or communication between hardware accelerators that would require prohibitively costly transfers across the back plane. The "N-Body" dwarf generally consists of calculations such as particle-particle interactions. A Barnes-Hut-based particle-particle N-Body model, as used for modelling astrophysical gravitating systems [56], would be able to calculate the changes in interaction on a GPU, while the memory-intensive cell (spatial position) data would be optimally handled by an FPGA. Although there is an implementation of an Ising "spin-flip" model on the Janus FPGA cluster [25], an example of a "Structured Grid" dwarf, a simple Ising model with a limited number of FPGAs would be better optimized using a GPU in addition.

The "Unstructured Grid" dwarf involves Adaptive Mesh Refinement, where calculations may be simplified by considering that only salient points in a space need be calculated, such as for adaptive finite element modelling (FEM) or computational fluid dynamics (CFD). The CPU will be useful in this case to tally changes in the mesh and co-ordinate calculations on salient mesh points using the FPGA and/or GPU.

Because of its conditional nature, "graph traversal" (including selection (Section 3.4, [43]), searching (Section 8.5, [43]) or decision trees) requires coordination from a CPU, and hardware acceleration is dependent on the particular application. "Dynamic Programming," such as the famous "Tower of Hanoi" problem, also requires CPU coordination, and also memory-intensive routines that are likely to benefit from an FPGA.

"Backtrack/Branch-and-Bound" problems, including search and global optimization problems, also depend on coordination from a CPU and again are generally memory intensive. "Graphical Models," such as Hidden Markov, probabilistic, neural, and Bayesian networks are also heavily dependent on coordination.

In light of these arguments, we summarize the most promising subsystem combination to apply for each dwarf

Table 2: The "Thirteen Dwarves" of Berkeley. Each dwarf represents an "algorithmic method encapsulating a pattern of computation and/or communication," and this intended to be a comprehensive list of the major requirements of parallel computational problems for now into the short term.

	Dwarf	Examples/Applications
1	Dense Matrix	Linear algebra (dense matrices)
2	Sparse Matrix	Linear algebra (sparse matrices)
3	Spectral	FFT-based methods
4	N-Body	Particle-particle interactions
5	Structured Grid	Fluid dynamics, meteorology
6	Unstructured Grid	Adaptive mesh FEM
7	MapReduce	Monte Carlo integration
8	Combinational Logic	Logic gates (e.g., Toffoli gates)
9	Graph traversal	Searching, selection
10	Dynamic Programming	Tower of Hanoi problem
11	Backtrack/ Branch-and-Bound	Global optimization
12	Graphical Models	Probabilistic networks
13	Finite State Machine	TTL counter

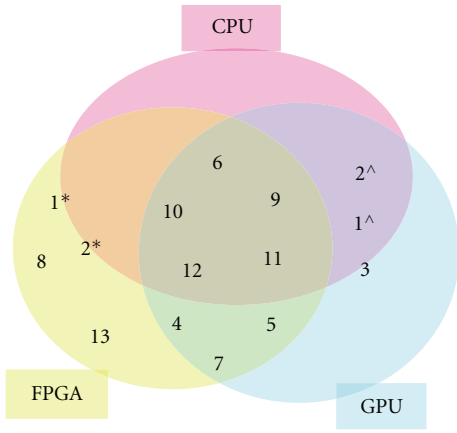

Figure 4: The most appropriate hardware acceleration subsystem combination for representative problems from the "Thirteen Dwarves" (Table 2). The * refers to fixed point, while ^ represents floating point calculations.

as a Venn diagram in Figure 4. Of course, this should be understood that the performance is heavily dependent on the particular implementation, generation of subsystem (including on-board memory, number of LUTs, etc.), and interconnect speed.

To characterize the performance of the Chimera system, we would like to analyze representative problems in each dwarf in the future using appropriate benchmark packages such as, for example, the "DwarfBench" package [57] or "Rodinia" [58]. Such a benchmark would be of use to researchers considering the benefits of this approach for their own work.

7. Discussion

We see a heterogeneous CPU/GPU/FPGA solution as a viable future platform for ameliorating some of the computationally bound problems in astronomy based on performance, initial outlay, and power consumption considerations. Because of the outstanding progress made in microprocessor technologies, most members of the astronomical and broader scientific community will not have considered the possibility of including an FPGA-based accelerator to their analysis pipelines; indeed the adoption of the GPU is relatively new within the data analysis community.

However, this solution is not without challenges. There is a significant investment required in development time. This begins with the "technology mapping" stage, which can require many hours for a modest pipeline, such as in Sections 5.1 and 5.2, each of which required considerable thought of how to distribute compute resources, including small-scale trials using MATLAB as a development environment, comprising one to two hours each. The GPU programming for these two examples took a total of about five hours, largely thanks to nVidia's comprehensive CUDA SDK support. However, because the degree of optimization of the FPGA pipelines meant the total amount of time spent "programming" was approximately forty to fifty hours, the majority of which was spent in the synthesis/debugging phase using ModelSim. Although we have not yet implemented the example in Section 5.3—which requires the dynamic range given by floating point types—considerable development time was saved using Altera's DSP Builder, which interfaces easily with Mathworks' Simulink. Another issue to be aware of is that a high degree of hardware optimization generally means a trade-off in problem-size scalability. For example, in Section 5.1 example, the pairs of numbers (x, y) are each 32-bit and the cases under-/overflow are well known; a naïve scaling to, say, single precision floating point pairs would quickly reduce the number of parallel units on the FPGA. In example Section 5.2, if instead of an 8×8 template, we were to use 32×32 template, performance would scale very poorly because the limiting factor in performance is the number of DSP18 blocks on the StratixIV530 FPGA.

A promising advantage of this platform is the low power consumption. Taking the peak rated single precision performance of each subsystem gives 1.02 Tflop/s for the Tesla C2070, drawing a power of 228 W. This gives 4.3 Gflop/s/W, or 4.3 Gflop/J. Compare to the StratixIV, with 500 Mflop performance, drawing around 20 W at this performance, yielding roughly 25 Mflop/s/W. Perhaps more importantly, the FPGA draws virtually no power when not performing intensive calculations, unlike both the GPU and CPU systems.

Although we have shown the merits and challenges of a mixed system, the advantages are obvious for a diverse range of parallel computing tasks, as shown by analysis of Berkeley's "Thirteen Dwarves." This paper hopefully provides a blueprint for future researchers intending to perform computationally intensive investigations and are willing to embrace a heterogeneous computing platform.

Acknowledgments

The authors wish to thank the Australian National University and Altera Corporation for their generous support. R. Inta is supported by an Australian Research Council "Discovery" project and D. J. Bowman is supported by an Australian Research Council "Super Science" Project. They would like also to thank Martin Langhammer for helpful discussion, and the anonymous reviewers, whose suggestions greatly improved the manuscript.

References

[1] B. P. Schmidt, N. B. Suntzeff, M. M. Phillips et al., "The high-Z supernova search: measuring cosmic deceleration and global curvature of the universe using type Ia supernovae," *Astrophysical Journal*, vol. 507, no. 1, pp. 46–63, 1998.

[2] G. F. Smoot, C. L. Bennett, A. Kogut et al., "Structure in the COBE differential microwave radiometer first-year maps," *Astrophysical Journal*, vol. 396, no. 1, pp. L1–L5, 1992.

[3] D. N. Spergel, L. Verde, H. V. Peiris et al., "First-year Wilkinson Microwave Anisotropy Probe (WMAP) observations: determination of cosmological parameters," *Astrophysical Journal, Supplement Series*, vol. 148, no. 1, pp. 175–194, 2003.

[4] University of California, 2011, http://setiathome.berkeley.edu/.

[5] B. Knispel, B. Allen, J. M. Cordes et al., "Pulsar discovery by global volunteer computing," *Science*, vol. 329, no. 5997, p. 1305, 2010.

[6] T. Cornwell and B. Humphreys, "Data processing for ASKAP and SKA," 2010, http://www.atnf.csiro.au/people/tim.cornwell/presentations/nzpathwaysfeb2010.pdf.

[7] M. Hilbert and P. López, "The world's technological capacity to store, communicate, and compute information," *Science*, vol. 332, no. 6025, pp. 60–65, 2011.

[8] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "Top 500 supercomputers," 2011, http://www.top500.org/.

[9] LSST Corporation, "Large synoptic survey telescope," 2011, http://www.lsst.org/lsst/.

[10] The Australian National University, "The SkyMapper survey telescope," 2010, http://msowww.anu.edu.au/skymapper/.

[11] Pico Computing, Inc., "Using FPGA clusters for fast password recovery," 2010, http://www.scribd.com/doc/26191199/Using-FPGA-Clusters-for-Fast-Password-Recovery.

[12] C. Duhigg, "Stock traders find speed pays, in milliseconds," 2009, http://www.nytimes.com/2009/07/24/business/24trading.html.

[13] J. Ericson, "Processing Avatar," 2009, http://www.information-management.com/newsletters/avatar_data_processing-10016774-1.html.

[14] J. Jackson, "Supercomputing top500 brews discontent," 2010, http://www.pcworld.idg.com.au/article/368598/supercomputing_top500_brews_discontent/.

[15] N. Hasasneh, I. Bell, C. Jesshope, W. Grass, B. Sick, and K. Waldschmidt, "Scalable and partitionable asynchronous arbiter for micro-threaded chip multiprocessors," in *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS '06)*, W. Grass, B. Sick, and K. Waldschmidt, Eds., vol. 3894 of *Lecture Notes in Computer Science*, pp. 252–267, Frankfurt, Germany, 2006.

[16] P. E. Ross, "A computer for the clouds," 2008, http://spectrum.ieee.org/computing/hardware/a-computer-for-the-clouds.

[17] M. Wehner, L. Oliker, and J. Shalf, "Low-power supercomputers (ieee spectrum)," 2009, http://spectrum.ieee.org/computing/embedded-systems/lowpower-supercomputers.

[18] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in *Proceedings of the 12th International Symposium on Field-Programmable Gate Arrays (FPGA '04)*, pp. 171–180, New York, NY, USA, February 2004.

[19] S. K. Chung, L. Wen, D. Blair, K. Cannon, and A. Datta, "Application of graphics processing units to search pipelines for gravitational waves from coalescing binaries of compact objects," *Classical and Quantum Gravity*, vol. 27, no. 13, Article ID 135009, 2010.

[20] B. R. Barsdell, D. G. Barnes, and C. J. Fluke, "Analysing astronomy algorithms for graphics processing units and beyond," *Monthly Notices of the Royal Astronomical Society*, vol. 408, no. 3, pp. 1936–1944, 2010.

[21] M. Bergano, F. Fernandes, L. Cupido et al., "Digital complex correlator for a C-band polarimetry survey," *Experimental Astronomy*, vol. 30, no. 1, pp. 23–37, 2011.

[22] L. De Souza, J. D. Bunton, D. Campbell-Wilson, R. J. Cappallo, and B. Kincaid, "A radio astronomy correlator optimized for the Xilinx Virtex-4 SX FPGA," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 62–67, August 2007.

[23] B. Klein, S. D. Philipp, I. Krämer, C. Kasemann, R. Güsten, and K. M. Menten, "The APEX digital fast fourier transform spectrometer," *Astronomy and Astrophysics*, vol. 454, no. 2, pp. L29–L32, 2006.

[24] J. Frigo, D. Palmer, M. Gokhale, and M. Popkin Paine, "Gamma-ray pulsar detection using reconfigurable computing hardware," in *Proceedings of the 11th IEEE Symposium Field-Programmable Custom Computing Machines*, pp. 155–161, Washington, DC, USA, 2003.

[25] F. Belletti, M. Guidetti, A. Maiorano et al., "Janus: an FPGA-based system for high-performance scientific computing," *Computing in Science and Engineering*, vol. 11, no. 1, Article ID 4720223, pp. 48–58, 2009.

[26] T. Xiang and B. Khaled, "High-performance quasi-Monte Carlo financial simulation: FPGA vs. GPP vs. GPU," in *Proceedings of the ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, pp. 26:1–26:22, New York, NY, USA, 2010.

[27] M. Awad, "FPGA supercomputing platforms: a survey," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 564–568, Prague, Czech Republic, 2009.

[28] D. B. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively processor arrays for random number generation," in *Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '09)*, pp. 63–72, Monterey, Calif, USA, 2009.

[29] B. Cope, P. Y. K. Cheung, W. Luk, and S. Witt, "Have GPUs made FPGAs redundant in the field of video processing?" in *Proceeding of the IEEE International Conference on Field Programmable Technology*, vol. 1, pp. 111–118, December 2005.

[30] B. Cope, "Implementation of 2D convolution on FPGA, GPU and CPU," Tech. Rep., Imperial College, London, UK, 2006.

[31] D. H. Jones, A. Powell, C. -S. Bouganis, and P. Y.K. Cheung, "GPU versus FPGA for high productivity computing," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '10)*, pp. 119–124, 2010.

[32] D. Yang, J. Sun, J. Lee et al., "Performance comparison of cholesky decomposition on GPUs and FPGAs," in *Proceedings of the Symposium Application Accelerators in High Performance Computing (SAAHPC '10)*, Knoxville, Tenn, USA, 2010.

[33] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *Proceedings of the Symposium on Application Specific Processors (SASP '08)*, pp. 101–107, Anaheim, Calif, USA, 2008.

[34] S. J. Park, D. R. Shires, and B. J. Henz, "Coprocessor computing with FPGA and GPU," in *Proceedings of the Department of Defense High Performance Computing Modernization Program: Users Group Conference—Solving the Hard Problems*, pp. 366–370, Seattle, Wash, USA, 2008.

[35] R. Inta and D. J. Bowman, "An FPGA/GPU/CPU hybrid platform for solving hard computational problems," in *Proceedings of the eResearch Australasia*, Gold Coast, Australia, 2010.

[36] M. Showerman, J. Enos, A. Pant et al., "QP: a heterogeneous multi-accelerator cluster," in *Proceedings of the 10th LCI International Conference on High-Performance Cluster Computing*, vol. 7800, pp. 1–8, Boulder, Colo, USA, 2009.

[37] K. H. Tsoi and W. Luk, "Axel: a heterogeneous cluster with FPGAs and GPUs," in *Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA '01)*, pp. 115–124, Monterey, Calif, USA, 2010.

[38] E. J. Kelmelis, J. P. Durbano, J. R. Humphrey, F. E. Ortiz, and P. F. Curt, "Modeling and simulation of nanoscale devices with a desktop supercomputer," in *Proceedings of the Nanomodeling II*, vol. 6328, p. 62270N, 2006.

[39] W. Kastl and T. Loimayr, "A parallel computing system with specialized coprocessors for cryptanalytic algorithms," in *Proceedings of the Sicherheit*, pp. 73–83, Berlin, Germany, 2010.

[40] K. D. Underwood and K. S. Hemmert, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*, chapter 31, Morgan Kaufmann Publishers, Burlington, Mass, USA, 2008.

[41] K. Asanovic, "IEEE standard for binary floating-Point Arithmetic," Tech. Rep. ANSI/IEEE Std., IEEE Standards Board, The Institute of Electrical and Electronics, 1985.

[42] M. Langhammer, "Floating point datapath synthesis for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 355–360, Heidelberg, Germany, 2008.

[43] S. A. T. W. H. Press, W. T. Vettering, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA, 2nd edition, 1997.

[44] P. J. Napier, D. S. Bagri, B. G. Clark et al., "Very long baseline array," *Proceedings of the IEEE*, vol. 82, no. 5, pp. 658–672, 1994.

[45] D. Bowman, M. Tahtali, and A. Lambert, "Rethinking image registration on customizable hardware," in *Image Reconstruction from Incomplete Data VI*, vol. 7800 of *Proceedings of SPIE*, San Diego, Calif, USA, 2010.

[46] LIGO Scientific Collaboration, 2010, http://www.ligo.caltech.edu/.

[47] Virgo Scientific Collaboration, 2010, http://www.virgo.infn.it/.

[48] P. K. Patel, *Search for gravitational waves from a nearby neutron star using barycentric resampling*, Ph.D. thesis, California Institute of Technology, Pasadena, Calif, USA, 2011.

[49] D. Llamocca, M. Pattichis, and G. A. Vera, "Partial reconfigurable FIR filtering system using distributed arithmetic," *International Journal of Reconfigurable Computing*, vol. 2010, Article ID 357978, 14 pages, 2010.

[50] E. J. Candès, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[51] O. Maslennikow, V. Lepekha, A. Sergiyenko, A. Tomas, and R. Wyrzykowski, *Parallel Implementation of Cholesky LL^T—Algorithm in FPGA-Based Processor*, Springer, Berlin, Germany, 2008.

[52] D. Yang, H. Li, G. D. Peterson, and A. Fathy, "Compressed sensing based UWB receiver: hardware compressing and FPGA reconstruction," in *Proceedings of the 43rd Annual Conference on Information Sciences and Systems (CISS '09)*, pp. 198–201, Baltimore, Md, USA, 2009.

[53] A. Septimus and R. Steinberg, "Compressive sampling hardware reconstruction," in *Proceedings of the IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems (ISCAS '10)*, pp. 3116–3119, Paris, France, 2010.

[54] P. Colella, *Defining Software Requirements for Scientific Computing*, DARPA HPCS, 2004.

[55] K. Asanovic and U C Berkeley Computer Science Deptartment, "The landscape of parallel computing research: a view from Berkeley," Tech. Rep. UCB/EECS-2006-183, UC Berkeley, 2005.

[56] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[57] R. Palmer et al., "Parallel dwarfs," 2011, http://paralleldwarfs.codeplex.com/.

[58] S. Che, M. Boyer, J. Meng et al., "Rodinia: a benchmark suite for heterogeneous computing," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC '09)*, pp. 44–54, Austin, Tex, USA, October 2009.