A Survey of CPU-GPU Heterogeneous Computing Techniques

Sparsh Mittal, Oak Ridge National Laboratory
Jeffrey S. Vetter, Oak Ridge National Laboratory and Georgia Tech

As both CPU and GPU become employed in a wide range of applications, it has been acknowledged that both of these processing units (PUs) have their unique features and strengths and hence, CPU-GPU collaboration is inevitable to achieve high-performance computing. This has motivated significant amount of research on heterogeneous computing techniques, along with the design of CPU-GPU fused chips and petascale heterogeneous supercomputers. In this paper, we survey heterogeneous computing techniques (HCTs) such as workload-partitioning which enable utilizing both CPU and GPU to improve performance and/or energy efficiency. We review heterogeneous computing approaches at runtime, algorithm, programming, compiler and application level. Further, we review both discrete and fused CPU-GPU systems; and discuss benchmark suites designed for evaluating heterogeneous computing systems (HCSs). We believe that this paper will provide insights into working and scope of applications of HCTs to researchers and motivate them to further harness the computational powers of CPUs and GPUs to achieve the goal of exascale performance.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; I.3.1 [**COMPUTER GRAPHICS**]: Graphics Processor ; C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems; H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness); C.0 [**Computer Systems Organization**]: System architectures

General Terms: Experimentation, Management, Measurement,Performance, Analysis

Additional Key Words and Phrases: CPU-GPU heterogeneous/hybrid/collaborative computing, workload division/partitioning, dynamic/static load-balancing, pipelining, programming frameworks, fused CPU-GPU chip.

ACM Reference Format:
Sparsh Mittal and Jeffrey S. Vetter, 2013. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comput. Surv.* X, Y, Article 1 (February 2015), 36 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Computer architects, programmers and researchers are now moving away from the CPU *versus* GPU debate [Gregg and Hazelwood 2011; Lee et al. 2010; Mittal and Vetter 2015; Vuduc et al. 2010] towards a CPU *and* GPU paradigm where the best features of both can be intelligently combined to achieve even further computational gains. This paradigm, known as heterogeneous computing (HC), aims to match the requirements

Authors' address: Sparsh Mittal and Jeffrey S. Vetter, 1 Bethel Valley Road, Oak Ridge National Laboratory, Building 5100, MS-6173, Tennessee, USA 37830; email: {mittals,vetter}@ornl.gov.

DOI: http://dx.doi.org/10.1145/0000000.0000000

of each application to the strengths of CPU/GPU architectures and also achieve load-balancing by avoiding idle time for both the PUs. The importance of heterogeneous computing can be seen from the fact that an astonishingly large fraction of TOP500 and Green500 supercomputers now use both CPUs and GPUs [Green500 2014; Top500 2014]. An even closer level of integration between CPU and GPU can be achieved by fabricating them on the same chip and many such processors have already been produced, such as AMD Llano [Branover et al. 2012], Intel Sandy Bridge [Yuffe et al. 2011], Ivy Bridge [Damaraju et al. 2012] etc.

The vastly different architectures and programming models of CPU and GPU, however, also present several challenges in achieving such collaborative computing. Due to the interaction between them in a heterogeneous system, optimizing performance and energy efficiency requires taking into account the characteristics of both the PUs. For this reason, conventional CPU-only or GPU-only optimization techniques may not work well in a heterogeneous system and hence, novel techniques are required to realize the potential and promise of heterogeneous computing and also move towards the goals of exascale performance.

Contributions: In this paper, we provide an extensive survey of techniques and architectures proposed for heterogeneous computing. We first discuss the motivation for and challenges involved in heterogeneous computing and also clarify the terminology used in this research field (Section 2). In Section 2, we also discuss how the hardware architecture of both CPU and GPU have evolved over years to identify important trends. We then analyze the research works from several perspectives to highlight their similarities and differences. From the perspective of researchers and algorithm-designers, we summarize the works related to CPU-GPU workload partitioning and classify them based on their key idea, such as nature of scheduling (dynamic/static), basis of division of work among PUs etc. (Section 3 and Tables I and II). From the point of view of programmers and application developers, we classify the works based on the languages used for programming CPU and/or GPU in those works and also review the techniques proposed which are related to compiler and programming framework/library (Section 4 and Tables III and IV).

We then discuss the techniques for saving energy in HCSs and classify them based on their essential approach (Section 5 and Table V). Further, we review research works dealing with fused (integrated) CPU-GPU chips to show their relative features/limitations compared to the discrete GPUs (Section 6 and Table VI). Furthermore, we classify the research works based on their application (or workload) domain and discuss benchmark suites designed for evaluating HCSs (Section 7 and Tables VII and VIII). We conclude this paper with the discussion of the future trends (Section 8). We hope that this paper will be useful for a wide range of readers, including computer architects, developers, researchers and technical marketing professionals.

Scope of the paper: Since it is practically infeasible to review the broad spectrum of research works, we take the following approach to limit the scope of this survey. We discuss only those heterogeneous computing systems/techniques which use both CPU and GPU, although it is also possible to build heterogeneous systems using FPGAs (field programmable gate arrays) etc. We discuss heterogeneous computing techniques proposed for both discrete and fused CPU/GPU architectures. We do not include research works which obtain performance improvements from using GPU alone or which discuss load-balancing within a single GPU or within multiple GPUs only. In other words, we include works where CPU is used as a processing unit also, in addition to as a host running operating system. We do not include circuit/device/microarchitectural level research works, rather we include research works dealing with runtime and algorithm/application execution, compiler, programming framework/language/library and system-level techniques etc. We mainly focus on the key research idea of the papers to

gain insight, but we also mention the CPU/GPU used in their experiments to get an idea of their evaluation platform.

2. PROMISES AND CHALLENGES OF HETEROGENEOUS COMPUTING

2.1. Clarifying the terminology

Since different research works use different terminologies, we first clarify them and also mention the nomenclature which we use in this paper. In literature, CPU-GPU heterogeneous computing approaches have also been referred to as collaborative, hybrid, co-operative or synergistic execution, co-processing, divide and conquer approach etc. In accordance with its purpose of use, GPU is conventionally referred to as an *accelerator*. However, given that CPU-GPU workload division based techniques on heterogeneous systems use CPU also to get performance gain (and not just as a host), sometimes CPU is also referred to as an accelerator [Sun et al. 2012]. Similarly, while CPU is generally termed as *host* and GPU as a *device*; in context of HCSs, some researchers use the term 'device' to refer to both CPU and GPU [Liu and Luk 2012; Veldema et al. 2011]. To maintain clarity, we do not use the term accelerator to refer to GPU or device as a generic term for CPU or GPU. Instead, in this paper we use *PU* (processing unit) as the generic term for either CPU or GPU, following other researchers (e.g. [Binotto et al. 2010]). A few other equivalent terms used in literature are CU (computing unit) [Verner et al. 2011], CE (computing element) [Li et al. 2012] and PE (processing element) [Tsoi and Luk 2010].

At the architecture level, a chip which has both CPU and GPU integrated on the same chip is referred to as *APU* (accelerated processing unit) or *SCHP* (single-chip heterogeneous processor). This is also referred to as *fused* or *integrated* system, in contrast to a conventional *discrete* system which has CPU and GPU on different chip, connected through PCIe (Peripheral Component Interconnect Express) bus.

2.2. Evolution of hardware architecture of PUs

Before delving into *heterogeneous computing*, it is interesting to note how the hardware architecture and performance of each PU *individually* have evolved over time. For brevity, we only discuss some key parameters and focus on last 7-8 years to see the trends by sampling a few products.

2.2.1. Transistor count. Few years ago, CPUs had nearly 1B transistors [Wendel et al. 2010] while the recently announced Oracle SPARC M7 CPU will have more than 10B transistors on chip [Morgan 2014]. Similarly, the GT200 GPU (2008) had 1.4B transistors, while the recent Geforce GTX TITAN X GPU has 8B transistors [Pirzada 2015].

2.2.2. Core count. Earlier CPUs had only one or two cores, while as recent CPUs have 8 to 32 cores [Fluhr et al. 2014; Morgan 2014] and CPUs with more than 60 cores are likely to be available in near future [Gardner 2014]. Along similar lines, the number of CUDA cores in GeForce GTX 280, GTX 480, GTX 680 and GTX TITAN X GPU is 240, 448, 1536 and 3072, respectively [NVIDIA 2015].

2.2.3. Cache size. With increasing number of cores, the size of last level cache (LLC) on CPU is also increasing [Mittal 2014b]. The 45nm POWER7 processor had 32MB eDRAM (embedded DRAM) LLC [Wendel et al. 2010], the 32nm POWER7+ processor had an 80MB eDRAM LLC [Zyuban et al. 2013], while the 22nm POWER8 processor has a 96MB eDRAM LLC [Fluhr et al. 2014]. Earlier GPUs only provided software-managed caches and not hardware-managed caches, however, as the GPUs move towards general-purpose computing, they are now featuring increasingly larger-sized hardware-managed caches. For example, GT200 architecture GPU did not have an L2

cache, the Fermi GPU has 768KB LLC, the Kepler GPU has 1536KB LLC and the Maxwell GPU has 2048KB LLC [Mittal 2014a].

2.2.4. 3D stacking. 3D stacking facilitates high bandwidth and memory capacity [Poremba et al. 2015] and hence, both CPUs and GPUs are increasingly moving towards use of 3D stacking, for example, Intel's Knights Landing [Gardner 2014] and NVIDIA's Pascal GPU [Gupta 2014] will feature 3D stacked memory.

2.2.5. Interconnect bandwidth. The limited bandwidth of PCIe has conventionally remained a bottleneck in GPU performance, especially for applications which transfer large amount of data between CPU and GPU [Gregg and Hazelwood 2011; Lee et al. 2010]. However, a recently proposed interconnect, called NVLink, promises to offer 5 to 12$\times$ bandwidth compared to PCIe Gen3 interconnect [Gupta 2014] and this is likely to offset the bandwidth limitation of conventional interconnects.

These trends make it evident that the hardware architecture of both CPU and GPU has undergone and still undergoing a process of never-ending evolution. This motivates the research on CPU-GPU heterogeneous computing, which is exactly what we review in this paper.

2.3. Motivation for heterogeneous computing

While use of GPU as a stand-alone device seems a promising idea at first, there are several compelling reasons for moving towards a heterogeneous CPU/GPU computing approach:

2.3.1. Acknowledging and leveraging unique architectural strengths of PUs. Both CPU and GPU possess distinct architectural features. Modern multicore CPUs use upto a few tens of cores, which are typically out-of-order, multi-instruction issue cores. Also, CPU cores run at high-frequency and use large sized caches to minimize the latency of a single-thread. Clearly, CPUs are suited for latency-critical applications. In contrast, GPUs use much larger number of cores, which are in-order cores that share their control unit. Also, GPU cores use lower frequency, and smaller sized caches [Mittal 2014a]. Thus, GPUs are suited for throughput-critical applications. Thus, a heterogeneous system can provide high performance for a much wider variety of applications and usage scenarios than using either of CPU or GPU alone [Vetter and Mittal 2015].

2.3.2. Matching algorithmic requirements to features of PUs. For applications where data transfers dominate execution time, or branch divergence does not allow for the uninterrupted execution on all GPU cores, CPUs can provide better performance than GPUs. Not only different applications, but even different phases of a single application may exhibit properties which make it more suitable for execution on a particular PU [He and Hong 2010; Nere et al. 2012; Shen et al. 2013]). For example, Ding et al. [2009] note that for a query processing application, CPUs are more efficient for queries involving short lists while GPUs are more efficient for those involving long lists.

2.3.3. Improving resource utilization. To meet the worse-case performance requirements, the CPU-only or GPU-only systems are usually over-provisioned, however, their average utilization remains low [Mittal 2012; Mittal and Vetter 2015]. Further, after allocating the task to GPU (i.e. starting the kernel), CPU stays idle which leads to wastage of energy. Similarly, for applications where the GPU memory bandwidth acts as a bottleneck [Daga et al. 2011; Spafford et al. 2012], the computational resources of GPUs remain underutilized. HCTs can address these inefficiencies by intelligently managing the resources of both PUs [Gelado et al. 2010; Hu et al. 2011]. As a definite case in point, the June 2014 Green500 list of most energy efficient supercomputers

shows that *all* the top 15 systems in the list are CPU-GPU heterogeneous systems [Green500 2014].

2.3.4. Reaping the fruits of advancements in CPU design. In systems with GPUs, CPUs have been conventionally used as host for GPU to manage I/O and scheduling, however, as continuing innovations improve CPU performance even further (refer Section 2.2), using their computation capabilities also has become more attractive. Further, while several initial works report that GPUs provide up to 100× to 1000× speedup, other researchers claim that on applying careful optimizations on *both* CPUs and GPUs; CPUs may equal or even outperform the performance of GPUs [Gummaraju et al. 2010; Lee et al. 2010]. Due to this, different amount of work-divisions to CPU and GPU can lead to vastly different performance [Grewe and O'Boyle 2011]. These points highlight the importance of using computational capabilities of CPU also.

2.4. Factors and challenges involved in heterogeneous computing

The vastly different architecture, programming model and performance (for a given program) of CPU and GPU present unique challenges in heterogeneous computing. Several factors relating to both the PU and the application itself and spanning from microarchitecture-level to system-level need to be taken into account for fully leveraging their potential in an HCS. In what follows, we briefly mention these factors/issues.

2.4.1. PU-specific. (1) Architecture of HCS (discrete or fused) (2) Computation power of the PUs (3) Current load on PUs and achieving load balancing between them (4) Memory bandwidth and CPU-GPU data transfer overhead; avoiding and/or amortizing overhead of launching GPU kernel and data transfer (5) Pipelining for overlapping data transfer with computation or CPU computation with GPU computation (6) Taking into account limitations of PUs, e.g. CPU-GPU memory bandwidth, size of GPU and CPU memory, number of GPU threads and CPU cores, reduced performance of GPU for double-precision computations etc. [Vetter and Mittal 2015]. Also, aggressively using CPU for computation affects its ability to act as a host, which harms the performance [Endo et al. 2010; Gregg et al. 2010; Sun et al. 2012].

2.4.2. Application/problem-specific. (1) Nature of algorithms, e.g. amount of parallelism, presence of branch divergence (2) Subdividing the workload and selecting suitable work sizes to be allocated to PUs (3) Accounting for data dependencies, e.g. if a task has data dependencies on previous task, where was the previous task executed?

2.4.3. Objective-specific. Achieving higher level optimization targets such as energy saving, performance and fairness etc.

The HCTs discussed in next several sections account for these factors to avoid their impact on performance.

3. ALGORITHM AND PROGRAM-EXECUTION LEVEL WORKLOAD PARTITIONING TECHNIQUES

Tables I and II categorize the techniques proposed for intelligently partitioning the workload between CPU and GPU at the level of algorithm or during program execution. We classify the works based on two main criteria, which are as follows.

(1) **Dynamic or static scheduling:** This answers *when* the scheduling is done (Table I). In dynamic division, the decision about running the subtasks or program-phases or code-portions on a particular PU is taken at runtime. In static division, the subtasks which are executed on a particular PU are already decided before program execution, in other words, the mapping of subtasks to PUs is fixed.

(2) **Basis of workload partitioning:** This answers *why* a particular scheduling of tasks to PUs is done (Table II). This can be motivated by characteristic/capability of PU itself and/or the subtasks themselves. For example, assuming that the different subtasks are of similar nature and can run on any of the PU, the decision about where a subtask should be mapped depends on which PU provides higher performance and which mapping helps in achieving load-balancing. However, if subtasks differ, it may be more suitable to map a particular subtask to a particular PU, for example, highly-parallel subtasks can be mapped to the GPU while the sequential subtasks can be mapped to the CPU. In some cases, a subtask cannot be mapped on a particular PU, for example, when the memory footprint of a subtask exceeds the memory size of GPU.

In Table II, we also identify the techniques which use pipelining between CPU and GPU, such that different phases or dependent subtasks are handled by CPU and GPU in an overlapped manner, or transfer of data between them is overlapped with computation. We now discuss a few of these techniques.

Table I. A classification of algorithm/runtime level HCTs based on nature of scheduling/mapping

Classification	References
Dynamic	[Acosta et al. 2010; Agulleiro et al. 2012; Agullo et al. 2011; Albayrak et al. 2012; Álvarez-Melcón et al. 2013; Becchi et al. 2010; Belviranli et al. 2013; Bernabé et al. 2013; Bhaskaran-Nair et al. 2013; Binotto et al. 2011; Boyer et al. 2013; Breß et al. 2013; Chen et al. 2012; Choi et al. 2013; Clarke et al. 2012; Delorme 2013; Deshpande et al. 2011; Diamos and Yalamanchili 2008; Gao et al. 2012; Garba and González-vélez 2012; Gregg et al. 2011, 2010; Hamano et al. 2009; Hartley et al. 2010; Hawick and Playne 2013; He and Hong 2010; Hermann et al. 2010; Horton et al. 2011; Humphrey et al. 2012; Huo et al. 2011; Jiang and Agrawal 2012; Jiménez et al. 2009; Joselli et al. 2008; Kofler et al. 2013; Kothapalli et al. 2013; Lang and Rünger 2013; Lecron et al. 2011; Lee et al. 2012b; Li et al. 2012, 2011; Liu et al. 2012; Ma et al. 2012, 2013; Mariano et al. 2012; Munguia et al. 2012; Muramatsu et al. 2011; Murarașu et al. 2012; Odajima et al. 2012; Papadrakakis et al. 2011; Pienaar et al. 2011; Ravi and Agrawal 2011; Ravi et al. 2012; Scogland et al. 2012; Shen et al. 2010; Shirahata et al. 2010; Siegel et al. 2010; Silberstein and Maruyama 2011; Stefanski 2013; Su et al. 2013; Tan et al. 2012; Teodoro et al. 2012, 2013, 2009; Udupa et al. 2009; Vömel et al. 2012; Wang et al. 2013a,c; Wu et al. 2012; Yang et al. 2010; Yao et al. 2010]
Static	[Agullo et al. 2011; Balevic and Kienhuis 2011; Banerjee and Kothapalli 2011; Benner et al. 2011, 2010; Binotto et al. 2010; Boratto et al. 2012; Boyer et al. 2013; Chai et al. 2013; Chen et al. 2010, 2012; Choi et al. 2013; da S Junior et al. 2010; Delorme 2013; Dziekonski et al. 2011; Endo et al. 2010; Gao et al. 2012; Gregg et al. 2011; Grewe and O'Boyle 2011; Hamano et al. 2009; Hampton et al. 2010; Hardy et al. 2009; Hu et al. 2011; Jetley et al. 2010; Korwar et al. 2013; Kothapalli et al. 2013; Liu et al. 2012, 2011; Liu and Luk 2012; Liu et al. 2009; Lu et al. 2012a,b; Luo et al. 2011; Mariano et al. 2012; Matam et al. 2012; Munguia et al. 2012; Murarașu et al. 2012; Nakasato et al. 2012; Nigam et al. 2012; Ogata et al. 2008; Ohshima et al. 2007; Pajot et al. 2011; Panetta et al. 2009; Park et al. 2011; Phothilimthana et al. 2013; Pienaar et al. 2011; Pirk et al. 2012; Rahimian et al. 2010; Ravi and Agrawal 2011; Scogland et al. 2012; Shen et al. 2013; Shukla and Bhuyan 2013; Singh and Aruni 2011; So et al. 2011; Sottilea et al. 2013; Stpiczynski and Potiopa 2010; Sun et al. 2012; Takizawa et al. 2008; Toharia et al. 2012; Tsoi and Luk 2010; Tsuda and Nakamura 2011; Venkatasubramanian and Vuduc 2009; Verner et al. 2011; Wang and Song 2011; Wang et al. 2013b; Xiao et al. 2011; Xu et al. 2012; Yang et al. 2013; Zhong et al. 2012]

3.1. Scheduling based on relative performance of PUs

Ogata et al. [2008] present a library for 2D fast Fourier transform (FFT) that automatically uses both PUs to achieve optimal performance. Using a performance model, it evaluates the respective contributions of each PU and then makes an estimation of the total execution time of FFT problem for arbitrary work distribution and problem

Table II. A classification of algorithm/runtime level HCTs based on criterion used for workload division (and other aspects)

Classification	References
Based on relative performance of PUs for load-balancing	[Acosta et al. 2010; Agulleiro et al. 2012; Agullo et al. 2011; Albayrak et al. 2012; Becchi et al. 2010; Belviranli et al. 2013; Bernabé et al. 2013; Bhaskaran-Nair et al. 2013; Binotto et al. 2011; Boratto et al. 2012; Boyer et al. 2013; Chai et al. 2013; Chen et al. 2012; Choi et al. 2013; Clarke et al. 2012; Endo et al. 2010; Gao et al. 2012; Garba and González-vélez 2012; Gharaibeh et al. 2012; Gregg et al. 2011, 2010; Grewe and O'Boyle 2011; Hardy et al. 2009; Hartley et al. 2008, 2010; Hawick and Playne 2013; He and Hong 2010; Hermann et al. 2010; Hu et al. 2011; Humphrey et al. 2012; Huo et al. 2011; Jiang and Agrawal 2012; Jiménez et al. 2009; Joselli et al. 2008; Kofler et al. 2013; Kothapalli et al. 2013; Lang and Rünger 2013; Lecron et al. 2011; Lee et al. 2012b; Li et al. 2012, 2011; Liu et al. 2012; Lu et al. 2012a; Ma et al. 2012, 2013; Matam et al. 2012; Murarașu et al. 2012; Nakasato et al. 2012; Nigam et al. 2012; Odajima et al. 2012; Ogata et al. 2008; Ohshima et al. 2007; Papadrakakis et al. 2011; Phothilimthana et al. 2013; Pienaar et al. 2012, 2011; Rahimian et al. 2010; Ravi and Agrawal 2011; Ravi et al. 2012; Rofouei et al. 2008; Scogland et al. 2012; Shen et al. 2010; Shimokawabe et al. 2011; Shirahata et al. 2010; Shukla and Bhuyan 2013; Siegel et al. 2010; Singh and Aruni 2011; Sottilea et al. 2013; Su et al. 2013; Sun et al. 2012; Teodoro et al. 2012, 2013; Udupa et al. 2009; Venkatasubramanian and Vuduc 2009; Verner et al. 2011; Vömel et al. 2012; Wang et al. 2013a,c; Wen et al. 2012; Wu et al. 2012, 2013; Yang et al. 2010, 2013; Yao et al. 2010; Zhong et al. 2012]
Based on low/high parallelism or other characteristics of subtasks/phases	[Álvarez-Melcón et al. 2013; Balevic and Kienhuis 2011; Banerjee et al. 2012; Banerjee and Kothapalli 2011; Benner et al. 2011, 2010; Binotto et al. 2010; Breß et al. 2013; Chen et al. 2010; Choudhary et al. 2012; Deshpande et al. 2011; Diamos and Yalamanchili 2008; Dziekonski et al. 2011; Garba and González-vélez 2012; Hampton et al. 2010; He and Hong 2010; Hong et al. 2011; Horton et al. 2011; Hu et al. 2011; Jetley et al. 2010; Joselli et al. 2008; Kothapalli et al. 2013; Liu et al. 2011, 2009; Ltaief et al. 2011; Lu et al. 2012b; Luo et al. 2011; Mariano et al. 2012; Munguia et al. 2012; Muramatsu et al. 2011; Pajot et al. 2011; Panetta et al. 2009; Pirk et al. 2012; Shen et al. 2013; Shimokawabe et al. 2011; So et al. 2011; Stefanski 2013; Stpiczynski 2011; Stpiczynski and Potiopa 2010; Su et al. 2013; Teodoro et al. 2009; Toharia et al. 2012; Tomov et al. 2010; Tsoi and Luk 2010; Tsuda and Nakamura 2011; Wang and Song 2011; Wang et al. 2013b; Xu et al. 2012; Yang et al. 2013]
Based on floating-point precision of PUs	[Benner et al. 2011; Gao et al. 2012; Stpiczynski and Potiopa 2010]
Based on limitation in size of GPU memory	[Huo et al. 2011; Ogata et al. 2008; Teodoro et al. 2013; Yao et al. 2010]
Other aspects	
Pipelining	[Balevic and Kienhuis 2011; Banerjee et al. 2012; Banerjee and Kothapalli 2011; Benner et al. 2011; Chen et al. 2012; Choudhary et al. 2012; Conti et al. 2012; Deshpande et al. 2011; Hampton et al. 2010; Huo et al. 2011; Jetley et al. 2010; Korwar et al. 2013; Li et al. 2012; Liu et al. 2011; Ltaief et al. 2011; Pajot et al. 2011; Panetta et al. 2009; Park et al. 2011; Pienaar et al. 2012; Shimokawabe et al. 2011; Su et al. 2013; Tan et al. 2012; Udupa et al. 2009; Wen et al. 2012; Wu et al. 2012; Xiao et al. 2011; Yang et al. 2010, 2013]
Use of MapReduce framework	[Chen et al. 2012; Hong et al. 2010; Jiang and Agrawal 2012; Ravi et al. 2012; Shirahata et al. 2010; Tan et al. 2012; Tsoi and Luk 2010; Wu et al. 2013]

sizes. It decomposes the computation of FFT into multiple sub-steps, and uses profiling along with data transfer time to estimate the execution time of each step. Using these estimates, the optimal workload division between PUs is found for achieving load-balancing. Their experimental system uses Core 2 Duo E6400 CPU and GeForce 8800 GTX GPU.

Ding et al. [2009] propose an approach for accelerating query processing. Based on the length of the query, their technique decides whether a query can be more efficiently

analyzed on CPU or GPU. Based on this, the tasks are put in one of the two task-queues. A third task-queue is made with the queries which may be efficiently analyzed on both CPU and GPU. A PU first processes queries from its own task-queue and then from the third task-queue. If it is still idle, it can steal a task from the task-queue of another PU, following the idea of work-stealing scheduling. Their evaluation platform uses Core 2 Duo CPU and GeForce 8800 GTS GPU.

Gregg et al. [2010] propose a workload division technique which schedules works on devices based on several factors such as the contention of devices, historical performance data, number of cores, processor speed, problem-size, and device status, e.g., busy or free. If the CPU is busy with running many threads, the process is run on GPU; and if GPU is significantly faster than CPU for a process, their technique waits for the GPU even if there is contention on the GPU. Moreover, if the problem size is such that the data associated with it cannot fit into GPU memory, the process is run on CPU. They perform experiments using Core 2 Duo CPU and Radeon HD 4350 GPU.

Becchi et al. [2010] propose a technique for automatically scheduling computation tasks over an HCS and managing data placement. Their technique intercepts function calls to kernels and schedules them on a PU based on their argument size, historical profile and location of data. Their technique accounts for both computation time and data transfer time. They observe that if the data are already available on a GPU (say), scheduling a task on GPU may provide advantage even if GPU provides less performance than the CPU. Thus, their technique defers all data transfers between PUs until necessary. Their experiments are conducted using a quad-core Xeon CPU and a Tesla C870 GPU.

Agullo et al. [2011] propose a method for accelerating QR factorization using CPU/GPU heterogeneous system. Their method works in three steps. In the first step, the QR factorization problem is expressed in terms of sequence of tasks of desired granularity such that they can be executed on a suitable PU. In the second step, CPU or GPU functions (or kernels) are designed for executing those tasks. Finally, in the third step, static or dynamic scheduling is used for scheduling these tasks on CPU or GPU. Static scheduling utilizes a priori knowledge of the schedule and provides high performance but does not offer portability. Dynamic scheduling uses StarPU runtime system [Augonnet et al. 2011] and thus provides high productivity and ability to schedule complex algorithms on heterogeneous systems. StarPU is a tasking API (application programming interface) that facalitates execution of parallel tasks on heterogeneous computing platforms, and incorporates multiple scheduling policies. They perform experiments using two HCSs, one with Xeon X5550 CPU and Quadro FX5800 GPU and another with Opteron 8358 SE CPU and Tesla S1070 GPU.

Lecron et al. [2011] present an HCT for detecting and segmenting vertebra in X-ray images. The most computation intensive part of vertebra segmentation is edge detection. After initial loading of images, their technique uses StarPU system to map edge detection function on GPU and CPU to effectively utilize both PUs. StarPU also provides functionality to transfer the GPU results to the CPU at the end of computation. Using their approach, multiple CPU cores and GPUs can be simultaneously used for achieving further speedup. Their experiments use Core 2 Duo 6600 CPU along with Tesla C1060 GPU.

Li et al. [2011] propose an HCT for Cryo-EM 3D reconstruction, where tasks (in this case, individual images) are assigned to CPU and GPU based on their relative performance. The estimates of performance of PUs is updated during each iteration of execution of algorithm. They also propose techniques to exploit hardware parallelism provided by each PU by leveraging thread-level and data-level parallelism. Their experiments are performed on a supercomputer whose nodes are composed of two 6-core Xeon X5650 CPUs and a Tesla C2050 GPU.

Deshpande et al. [2011] present two techniques for accelerating image dithering operation. In image dithering, the amount of parallelism available changes for different regions of the image. The parallelism is low towards the beginning and end; and is high in the middle of dithering. Their first technique, called "CPU-GPU handover" uses CPU for executing initial part of the algorithm (step 1), then transfers the data to GPU for doing middle part of the algorithm (step 2) and finally hands over the data back to GPU for doing last part of the algorithm (step 3). In this technique, although both CPU and GPU are used for the task for which they are most efficient, they are not used together at the same time. Their second technique called "CPU-GPU hybrid" changes the step 2 above, such that the work of step 2 is divided between CPU and GPU. Thus, CPU performs step 1 and 3 alone; and also shares the work in step 2. They have shown that the second technique provides higher performance than the first technique. Their experiments use Core 2 Duo P8600 CPU along with GeForce 8600M GT.

Verner et al. [2011] propose an algorithm for implementing hard real-time stream scheduling in heterogeneous systems. Their algorithm partitions the incoming streams into two subsets, one for processing by the GPU and the other for the CPU. Using a schedulability criteria, it is ensured that both subsets are schedulable. The algorithm works to find an assignment which satisfies both, the deadline constraint of each stream alone and the aggregate throughput requirements of all the streams. A Core 2 Quad CPU and a GeForce GTX 285 GPU was used for their experiments.

Grewe and O'Boyle [2011] propose a static partitioning technique for OpenCL programs on HCSs. Their technique conducts static analysis on OpenCL programs to extract code features. Using this information, their technique first determines the best work-division ratio across the PUs in a system. Afterwards, it divides the workload into suitable sized chunks for each PU using machine learning approach. Use of machine learning makes their technique portable across different implementation platforms and different implementations of OpenCL. They evaluate their technique using a system with Xeon E5530 CPU and Radeon HD 5970 GPU.

Agulleiro et al. [2012] present an HCT for 3D tomographic reconstruction. In electron tomography, single-tilt axis geometry is used to decompose a 3D reconstruction problem into several independent 2D reconstruction tasks. These 2D slices of the volume can be computed in parallel using a reconstruction method, on either CPU or GPU. Their technique starts a CPU thread and one thread for each GPU, which is responsible for sending data to GPU and receiving the slices after reconstruction. When the threads become idle, more work (slices) is assigned to them and parallel execution takes place on both PUs. The amount of slices allocated to a PU depends on its performance, specifically GPU is allocated larger number of slices. Thus, their technique uses on-demand allocation for achieving load-balancing. They use two HCSs for evaluation purposes, one with Quad-core Xeon E5640 CPU and Tesla C2050 GPU and another with Quad-core Xeon W3520 CPU and GeForce GTX 285 GPU.

Teodoro et al. [2012] use CPU-GPU on-demand allocation on an HCS for accelerating an image processing application. To amortize the overhead of GPU kernel call and data transfer, they group large piece of data to be processed by each kernel and ensure that each kernel performs maximum amount of possible computations (i.e. steps of the image processing algorithm) with the data. Their experiments use 6-core Xeon X5660 CPU along with Tesla M2070 GPU.

Boratto et al. [2012] use static scheduling to divide the workload of matrix computation on CPU and GPU for solving the problem of landform attributes representation. Their evaluation platform uses 6-core Xeon X5680 CPUs and Tesla C2070 GPUs.

Matam et al. [2012] present a workload-division technique for generalized sparse matrix-matrix multiplication (SPGEMM). Due to the irregular computations in SPGEMM, GPU does not provide large speedup for this problem. Their technique ob-

serves the speedup of GPU over CPU and accordingly assigns a fraction of work to PUs such that their finish times become equal. They perform experiments using a system with Core i7 920 CPU and Tesla C2050 GPU.

Lu et al. [2012a] present an HCT for accelerating RRTM (Rapid Radiation Transfer Model) application used in radiation physics. RRTM involves loop over 2-dimensional grid and thus, presents opportunity for workload-division by splitting along x or y direction. Their technique divides the workload between CPU and GPU by taking into account their processing power. They use MPI + OpenMP/CUDA programming model, where communication between different nodes takes place using MPI; CPU parallelization is implemented using OpenMP and GPU is programmed using CUDA. They perform experiments using Tianhe-1A supercomputer whose compute node has two 6-core Xeon X5670 CPUs and a Tesla M2050 GPU.

Hawick and Playne [2013] present an HCT for cluster component-labeling analysis in critical phase modeling. Their technique performs simulation of Potts model on GPU, since Potts system has regular data structure and memory access locality. It also uses random number generation, which can be efficiently implemented on a GPU. For connected component labeling, their technique uses CPU if at least one CPU core is available, otherwise, this labeling is performed on the GPU itself. This approach indirectly takes the different performance values of GPU/CPU into account and avoids CPU idling. They perform experiments using multiple CPU-GPU HCSs, viz. an i7-2700K with a GTX590 (utilizing only one of its GPUs), an i7-970 with a GTX580, two Xeon E5-2640 with an M2090, two Xeon X5675 with an M2075 and finally two Opteron 6274 with a GTX680. They note that in all cases, the hybrid algorithm provided higher performance than using just the GPU. They also observed that based on decreasing order of performance, GPUs can be ranked as GTX680, GTX580, M2090, GTX590 and M2075.

Choi et al. [2013] present a workload division based scheduling technique for HCS. They also discuss simple scheduling policies such as the first-free scheduling policy, which schedules an incoming task on the first available PU; alternate-assignment scheduling policy, which schedules the tasks alternately on CPU and GPU and performance-history scheduling policy, which computes the ratio of performance of GPU over CPU for a task based on performance-history and schedules the task on GPU if the ratio is more than a threshold. Their technique provides enhancements to performance-history scheduling policy by also taking into account the information about the time when a PU will become available. Using this, along with actual estimated execution time of the task, the completion time of a task on both CPU and GPU can be estimated and the best PU can be chosen to minimize the makespan. Their evaluation platform is a system with Core 2 Quad Q9400 CPU and Geforce 8500GT GPU.

Bernabé et al. [2013] present an HCT for accelerating 3D-Fast Wavelet Transform. For GPU, they use kernels implemented in CUDA or OpenCL and for CPU, they use kernels implemented in Pthread. Using these kernels, they first profile the performance of GPU and CPU and then allocate the workload to the PUs in proportion to their performance. They experiment using two systems. The first system has a Xeon E5620 CPU, a Tesla C2050 GPU and an ATI FirePro V5800 GPU. The second system has Core 2 Quad Q6700 CPU and a Tesla C870 GPU. Of the two, the first system provides higher performance.

Belviranli et al. [2013] present a dynamic load-balancing technique for loop iterations on HCS. Their technique works in two phases. In first phase, the relative performance of PUs is estimated by experimenting with different task size allocations to the PU. Small task sizes provide better load-balancing, while large task sizes provide better resource utilization and an optimal value of task sizes achieves a trade-off between

these two factors. In second phase, the remainder and majority of computations are performed based on the relative performance values obtained in the first phase. The second phase utilizes a modified version of self-scheduling algorithm [Belviranli et al. 2013] to achieve load-balancing. Their experiments use a system with 16-core Opteron 6200 CPU and Tesla C2050 GPU.

3.2. Scheduling based on nature of subtasks

Liu et al. [2009] present a method to parallelize non-rigid registration (NRR) of medical images. They first divide the original serial algorithm into regular and irregular parts. Then, the regular part is mapped to GPU since it offers high parallelism. The irregular part is mapped to multicore CPU since it requires synchronization and communication and hence, cannot benefit from GPU implementation. A system with dual-core Opteron 2218 CPU and GeForce 8800 GT GPU is used for their experiments.

Yao et al. [2010] propose an HCT for HMMER application which is used for biosequence analysis. Their technique spawns one thread each for CPU and GPU; and when a sequence is retrieved from the database it can be assigned to either CPU or GPU. HMMER algorithm has large memory requirement and thus, for large sized sequences, shared memory of GPU becomes unsuitable while global memory incurs large performance penalty. Hence, their technique maps sequences larger than a threshold on CPU and smaller sequences on GPU. Their experiments are conducted on a system with Core 2 Duo E7200 CPU and GeForce 8800 GTX GPU.

Hermann et al. [2010] propose a workload-division technique for interactive physics simulations. Their technique divides the work of time integration between PUs using conventional graph partitioner and then uses work-stealing scheduling to achieve load-balancing. Their work-stealing algorithm takes spatial and temporal locality into account. For leveraging spatial locality, the task using same data are likely to be mapped to the same PU. For leveraging temporal locality, at the time of starting a new time integration step, preference is given to a PU which executed the previous iteration. Also, during work-stealing, their technique takes into account the fact that smaller tasks are more efficient on CPU and vice-versa. Their experiments use quad-core Nehalem CPU and GeForce GTX 295 GPU.

Benner et al. [2010] propose an HCT for accelerating computation of matrix sign function using Gauss-Jordan elimination. In each iteration, the factorization of the current column panel is performed on the CPU. It is because this step involves smaller data size, BLAS-1 operations and pivoting which are not suitable for parallelization. On GPU, matrix multiplication and pivoting of the columns outside the current column panel are performed, since they can be easily parallelized. Their evaluations are performed using a system with Xeon quad-core E5405 CPU and Tesla C1060 GPU.

Hu et al. [2011] present an approach for accelerating fast multipole method (FMM) on an HCS. They study the cost of different phases of FMM and the communication involved. Then, they distribute the work by considering the characteristics of both CPU and GPU, such that each PU is given the work which it can do in most efficient manner. This approach also provides load-balance and minimizes the data transfer between them. Their evaluations are performed over three CPU-GPU HCSs, one with Xeon X5560 CPU and Tesla S1070 GPU, second with quad-core Xeon Harpertown 5300 CPU and Tesla S1070 GPU and third with Xeon E5504 CPU and Tesla C2050 GPU.

Muramatsu et al. [2011] present an HCT for accelerating Hessenberg reduction for nonsymmetric eigenvalue problems. BLAS (Basic Linear Algebra Subprograms) operations form the majority of operations in this algorithm. Their technique assigns level-1 and level-2 BLAS operations on CPU, since parallelism present in these operations is limited and hence, the data transfer cost outweighs the performance gain obtained by using GPU. Further, for level-3 BLAS operations, the data are divided between CPU

and GPU, and they both perform operations on the data. This enables leveraging the performance of GPU, while avoiding CPU idling. They perform evaluations using Core i7 920 CPU and Tesla C1060 GPU.

Hong et al. [2011] propose a technique for accelerating breadth-first search (BFS) application. For each level of BFS algorithm execution, their technique dynamically selects the most suitable implementation from multiple choices, viz. a sequential execution and two possible parallel execution methods on CPU and a GPU execution. This decision is taken based on the number of nodes to be traversed in each level. For small number of nodes, the parallelism present is small, due to which GPU resources cannot be fully utilized and overhead of data transfer to GPU cannot be amortized. By choosing the suitable PU, their approach optimizes the performance achieved for each graph size. The experiments are conducted using an Xeon X5550 CPU and a Tesla C2050 GPU.

Stpiczynski [2011] implements linear recurrence systems involving constant coefficients on an HCS. The algorithm for solution of this problem can be expressed in terms of BLAS-2 and BLAS-3 operations. His heterogeneous implementation uses CPU for sequential steps and GPU for parallel steps to leverage the benefits of both. An HCS with Core i7 950 CPU and Tesla C2050 GPU is used for the experiments.

Stefanski [2013] proposes a workload-division technique for discrete Green's function (DFG), where short length DFGs are assigned to CPU and long length DFGs are assigned to GPU. They conduct experiments on a system with a quad-core Core i7 and a GeForce GTX 680 GPU.

In context of electromagnetic scattering, Gao et al. [2012] present a technique for implementing "shooting and bouncing ray" (SBR) method in conjunction with truncated wedge "incremental length diffraction coefficients" (TW-ILDCs) on HCS. They map SBR on GPU since numerous independent ray tubes can fully utilize the massively parallel resources on the GPU. TW-ILDCs are mapped to CPU since they require high-precision, complex numerical calculations to get the accurate result. Evaluations are performed using a system with Core i5 CPU and GeForce 470 GTX GPU.

Wang et al. [2013b] present an HCT for accelerating mapping of high-resolution human brain connectome. Out of different steps of the algorithm, their technique maps computation of Pearson's correlation, modular detection and all-pairs shortest path to GPU (based on available parallelism), and the other steps such as computation of clustering coefficients to CPU. Their experiments are performed on a system with quad-core Core i7-3770 CPU and Geforce GTX 580 GPU.

Yang et al. [2013] present an HCT for global atmospheric simulations using cubed-sphere shallow-water model. This simulation involves processing 2D data arranged in the form of rectangular patches. Their technique divides the patch into sub-blocks, where the computation of a sub-block also depends on its neighbors. Further, the sub-block is divided into an inner part, the computation for which does not depend on neighboring sub-blocks, and an outer part consisting of four boundaries. Their technique schedules the inner part on GPU to leverage parallelism and outer part on CPU since this part does not present large parallelism. By adjusting the size of inner part, their technique adjusts the workload division between PUs to achieve best possible performance. They conduct evaluations on Tianhe-1A supercomputer (configuration shown above) and achieve 0.8 PetaFLOP performance.

3.3. Pipelining

Park et al. [2011] present an HCT for ultra-wideband signal processing applications. Their technique divides the imaging component of this application, such that the back-projection step is carried out on GPU and data interpolation step is carried out on CPU. Moreover, using asynchronous kernel launch, the CPU starts processing the next

frame while GPU is completing the current frame. Using this approach, the latency of GPU backprojection processing time is effectively hidden. They use three CPU-GPU HCSs, which, in decreasing order of performance are 1) Xeon 5570 + Tesla 1060 2) Xeon 5160 + GeForce 8800 GTX and 3) Core 2 Duo T9500 + Quadro 3600M.

Xiao et al. [2011] propose an HCT for accelerating protein sequence search using BLASTP (Basic Local Alignment Search Tool for Protein sequence search). BLASTP has four steps (viz. hit detection, ungapped extension, gapped alignment, gapped alignment with traceback), of which the first three stages consume 99% of the execution time. Also, execution of first two stages is performed together, while that of the third stage is done independently. In their technique, the first two stages are executed on GPU and the third stage is executed on CPU. Also, the database is separated into several chunks. GPU executes first stages for a given chunk, transfers the output to CPU and then CPU starts stage three. Meanwhile, GPU starts processing the next chunk. Thus, they improve resource utilization using pipelining. Their CPU-GPU hybrid experimental system uses Core 2 Duo CPU and one of the two following GPUs: Tesla C1060 and Tesla C2050. Of the two GPUs, Tesla C2050 provides higher performance.

Banerjee et al. [2012] present a work-division technique for on-demand pseudo random number generation. Their random number generator uses parallel random walks on an expander graph. In this problem, few random bits are required to select a neighbor on the graph. In their technique, these bits are generated on CPU and are then transferred to GPU in asynchronous manner and then the GPU generates the random number. Since each walk can be performed independent of each other, massive parallelism can be obtained and using asynchronous transfer, latency of communication can be overlapped with that of computation. Their technique performs better than both CPU-only and GPU-only implementations. Their evaluation platform has Core i7 980 CPU and Tesla C1060 GPU.

Li et al. [2012] propose use of heterogeneous computing to accelerate DGEMM (double-precision general matrix multiplication) algorithm taking into account the case where the size of matrices is too large to fit in GPU memory. Their technique divides the matrices in sub-matrices and multiplies them block by block. Also it separates the GPU computation from write-back of results. Thus, both data transfer to GPU and from GPU are effectively overlapped with computation on GPU, so that CPU, GPU and PCIe can work in pipelined manner, which hides the latency of computation. Their experimental system has Xeon X5650 CPU and Radeon HD5970 GPU and achieves up to 844 GFLOPS performance.

3.4. Use of MapReduce Framework

Shirahata et al. [2010] propose a map task scheduling technique for HCSs. If a MapReduce job, whose tasks can be executed on both CPU and GPU, is submitted, a task-scheduler uses profiles collected from dynamic monitoring of map task's behavior to assign the map task to a suitable PU with a view to minimize the overall MapReduce task execution time. They implement their technique using Hadoop system. The experimental system uses dual-core Opteron 880 CPUs and Tesla S1070 GPUs.

Tsoi and Luk [2010] describe a heterogeneous compute cluster named Axel. Axel uses NNUS (non-uniform node, uniform system) architecture, where heterogeneous PUs (viz. CPU, GPU and FPGA) are hosted in a single node, and all nodes are connected through system bus. Axel uses MapReduce framework where GPU and FPGA work on the Map part in parallel and CPU works on the Reduce part. Further, it accounts for processing capability, local memory and communication capability for processing units to assign works to them and thus, leverage the parallelism of GPU, the specialization of FPGA, and the scalability of CPU clusters. The experimental system uses quad-core Phenom X4 9650 CPUs and Tesla C1060 GPUs.

Chen et al. [2012] present two approaches for accelerating MapReduce applications on an HCS. The first approach, called *map-dividing* dynamically schedules workload to each scheduling unit on both CPU and GPU, and each PU conducts map and reduce simultaneously. The second approach called *pipelining* runs the map and reduce stages on different PUs, i.e. each PU only executes one stage of MapReduce. They use one core of CPU as the scheduler; and the remaining CPU cores and each streaming multiprocessor of GPU as one scheduling unit. To achieve load balance while keeping scheduling costs low, they also propose a runtime tuning method to adjust task block sizes for each scheduling unit. The experiments are done using AMD Fusion APU (A8-3850) which integrates a quad-core CPU and an HD6550D GPU.

4. PROGRAMMING LANGUAGES, FRAMEWORKS AND RELATED DEVELOPMENT TOOLS

4.1. A classification of research works based on programming languages used

Table III classifies the works discussed in this paper based on the programming language used. Different languages offer different tradeoffs between ease of programming, ability to write optimized code, ability to target multiple PUs and products from different vendors etc. For example, OpenCL can be used for both CPU and GPU, and hence, applications written in OpenCL can be easily scheduled on any PU. By comparison, CUDA works only on NVIDIA GPUs and ACML (AMD Core Math Library) and Brook+ can be used to program AMD GPUs only. Similarly OpenMP, Pthread, TBB (thread building block) and MKL (math kernel library) are used to write parallel programs on CPUs. From Table III, it is clear that for GPU programming, CUDA is most widely used, which is due to its similarity with C/C++ and ability to write optimized code. For CPU programming, OpenMP is most widely used due to its portability and ease of programming by virtue of use of compiler directives. Note that MKL is internally parallelized by OpenMP threading; for sake of clarity, we mention it separately in Table III. In Section 4.4, we briefly discuss about OpenCL due to its capability to provide heterogeneous computing and omit the discussion of other languages/libraries mentioned in Table III for sake of brevity.

Table IV summarizes the works on programming frameworks, compiler techniques and runtime systems which facilitate heterogeneous computing. We now discuss a few of them.

4.2. Programming Frameworks

Diamos and Yalamanchili [2008] propose Harmony which utilizes performance estimates to schedule applications in an HCS. They propose online monitoring of kernels and describe a dependence-driven scheduling which analyzes how applications share data and decides on PU selection based on which applications can run without blocking. Their evaluations are performed on a system with Athlon64 CPU and GeForce 8800 GT GPU.

Hong et al. [2010] propose a framework called MapCG that facilitates portability between CPU and GPU at the level of source code. Without requiring modification, a program can be compiled and executed on either CPU or GPU using a MapReduce programming model. Use of OpenCL enables using a single kernel code version for both PUs in place of providing separate kernel versions for them. Their experimental platform uses 6-core Opteron CPU and GeForce GTX 280 GPU.

Pai et al. [2010] present a programming framework, named PLASMA, that enables writing of portable SIMD (single-instruction, multiple-data) programs. PLASMA uses an intermediate representation (IR), which provides succinct and clean abstractions (i.e. free from details of any particular SIMD architecture) to enable programs to be compiled on different PUs. Then, using a runtime, these programs can be automati-

Table III. Programming languages used for GPU and/or CPU in different research works

Classification	References
OpenCL	[Albayrak et al. 2012; Bernabé et al. 2013; Binotto et al. 2011; Boyer et al. 2013; Conti et al. 2012; Daga et al. 2011; Danalis et al. 2010; Delorme 2013; Gregg et al. 2011, 2010; Grewe and O'Boyle 2011; Hetherington et al. 2012; Kofler et al. 2013; Lee et al. 2012a; Liu et al. 2012; Nakasato et al. 2012; Phothilimthana et al. 2013; Shen et al. 2013; Spafford et al. 2012; Ukidave et al. 2013; Veldema et al. 2011]
CUDA (and its libraries) on GPU	[Acosta et al. 2010; Agulleiro et al. 2012; Agullo et al. 2011; Álvarez-Melcón et al. 2013; Anzt et al. 2011; Augonnet et al. 2011; Balevic and Kienhuis 2011; Banerjee et al. 2012; Banerjee and Kothapalli 2011; Becchi et al. 2010; Belviranli et al. 2013; Benner et al. 2011, 2010; Bernabé et al. 2013; Bhaskaran-Nair et al. 2013; Binotto et al. 2010; Boratto et al. 2012; Breß et al. 2013; Chai et al. 2013; Chen et al. 2010; Choudhary et al. 2012; Clarke et al. 2012; da S Junior et al. 2010; Deshpande et al. 2011; Diamos and Yalamanchili 2008; Dziekonski et al. 2011; Endo et al. 2010; Gao et al. 2012; Gharaibeh et al. 2012; Hampton et al. 2010; Hardy et al. 2009; Hartley et al. 2008, 2010; Hawick and Playne 2013; He and Hong 2010; Hermann et al. 2010; Hong et al. 2011; Hu et al. 2011; Humphrey et al. 2012; Huo et al. 2011; Jetley et al. 2010; Jiang and Agrawal 2012; Jiménez et al. 2009; Korwar et al. 2013; Kothapalli et al. 2013; Lang and Rünger 2013; Lecron et al. 2011; Lee et al. 2012b; Li et al. 2011; Liu et al. 2011; Liu and Luk 2012; Liu et al. 2009; Ltaief et al. 2011; Lu et al. 2012a,b; Luo et al. 2011; Ma et al. 2012, 2013; Mariano et al. 2012; Matam et al. 2012; Meredith et al. 2011; Munguia et al. 2012; Muramatsu et al. 2011; Murarașu et al. 2012; Nakasato et al. 2012; Nigam et al. 2012; Ogata et al. 2008; Ohshima et al. 2007; Padoin et al. 2012; Pajot et al. 2011; Panetta et al. 2009; Papadrakakis et al. 2011; Park et al. 2011; Phothilimthana et al. 2013; Pienaar et al. 2012, 2011; Rahimian et al. 2010; Ravi et al. 2012; Rofouei et al. 2008; Shen et al. 2010; Shimokawabe et al. 2011; Shirahata et al. 2010; Shukla and Bhuyan 2013; Siegel et al. 2010; Silberstein and Maruyama 2011; Singh and Aruni 2011; So et al. 2011; Sottilea et al. 2013; Stefanski 2013; Stpiczynski 2011; Stpiczynski and Potiopa 2010; Su et al. 2013; Sun et al. 2012; Takizawa et al. 2008; Tan et al. 2012; Teodoro et al. 2012, 2013, 2009; Toharia et al. 2012; Tomov et al. 2010; Tsoi and Luk 2010; Tsuda and Nakamura 2011; Udupa et al. 2009; Venkatasubramanian and Vuduc 2009; Verner et al. 2011; Wang et al. 2013a,b,c; Wen et al. 2012; Wu et al. 2012, 2013; Xiao et al. 2011; Xu et al. 2012; Yang et al. 2013; Yao et al. 2010; Zhong et al. 2012]
Brook+ on GPU	[Park et al. 2011; Wang and Song 2011]
ACML on GPU	[Li et al. 2012; Yang et al. 2010]
OpenMP on CPU	[Álvarez-Melcón et al. 2013; Ayguade et al. 2009; Banerjee and Kothapalli 2011; Boratto et al. 2012; Chai et al. 2013; Clarke et al. 2012; Conti et al. 2012; Daga et al. 2011; Deshpande et al. 2011; Hong et al. 2010; Korwar et al. 2013; Kothapalli et al. 2013; Lang and Rünger 2013; Li et al. 2013, 2011; Liu and Luk 2012; Lu et al. 2012a,b; Mariano et al. 2012; Nigam et al. 2012; Park et al. 2011; Rahimian et al. 2010; Shen et al. 2010; Sottilea et al. 2013; Teodoro et al. 2013; Veldema et al. 2011; Wen et al. 2012; Wu et al. 2012; Yang et al. 2013]
Pthread on CPU	[Balevic and Kienhuis 2011; Becchi et al. 2010; Bernabé et al. 2013; Humphrey et al. 2012; Singh and Aruni 2011; Su et al. 2013; Wang et al. 2013c,c; Xiao et al. 2011]
Intel MKL on CPU	[Agullo et al. 2011; Benner et al. 2011, 2010; Boratto et al. 2012; Clarke et al. 2012; Dziekonski et al. 2011; Horton et al. 2011; Ltaief et al. 2011; Matam et al. 2012; Meredith et al. 2011; Spafford et al. 2012; Tomov et al. 2010; Yang et al. 2010]
Intel TBB on CPU	[Becchi et al. 2010; Breß et al. 2013; Luk et al. 2009; Pienaar et al. 2011]

cally multithreaded and executed on different PUs, such as GPU, CPU and Cell BE, for example a `for` loop in a kernel can be split across CPU and GPU. The runtime takes care of load-balancing and distributed memory and also ensures that before any computation, data are moved from CPU to GPU or vice versa. Their experiments show that performance of portable PLASMA code compares well to the platform-specific codes. They perform experiments using a system with quad-core Xeon E5440 CPU and GeForce 8800 GTS GPU.

Table IV. A classification of development tools for HCSs

Classification	References
Programming frameworks, languages or libraries	[Augonnet et al. 2011; Ayguade et al. 2009; Diamos and Yalamanchili 2008; Gelado et al. 2010; Hong et al. 2010; Humphrey et al. 2012; Jablin et al. 2012; Kim et al. 2012; Lee et al. 2013; Li et al. 2013; Insieme Compiler 2014; OpenACC Standard 2014; OpenMP 4.0 2014; Mistry et al. 2013a; Odajima et al. 2012; Pai et al. 2010; Pandit and Govindarajan 2014; Pienaar et al. 2011; Saha et al. 2009; Scogland et al. 2014, 2012; Spafford et al. 2010; Stone et al. 2010; Udupa et al. 2009; Veldema et al. 2011]
Compiler-level techniques for mapping to CPU/GPU	[Kofler et al. 2013; Luk et al. 2009; Phothilimthana et al. 2013; Pienaar et al. 2012; Prasad et al. 2011; Ravi et al. 2010; Takizawa et al. 2008]
Design of OpenCL and related languages, frameworks or extensions	[Gummaraju et al. 2010; Kim et al. 2012; Mistry et al. 2013a; Phothilimthana et al. 2013; Shen et al. 2013; Spafford et al. 2010; Stone et al. 2010; Sun et al. 2012]

Pienaar et al. [2011] present a runtime framework for the execution of workloads represented as parallel-operator directed acyclic graphs (PO-DAGs) on HCSs. They identify four criteria, viz. suitability, locality, availability and criticality, which are important to consider while performing workload division for achieving good performance. Suitability shows which PU is most suited, i.e. provides best performance for a task. Locality shows whether the data required for a task is present in the memory of a PU and if so, by scheduling the task on that PU, data transfer cost can be avoided. Availability shows when a PU will become available for scheduling, thus it might be advantageous to wait for a suitable PU to become free rather than to schedule in a greedy manner. Criticality shows how the execution of a task affects the overall execution time, thus by scheduling the kernels which are on critical path on most suitable PU, the makespan can be reduced. Their method uses analytical models for estimating communication time and online history information for estimating kernel execution time; and based on these schedules the tasks on different PUs to achieve best possible performance. The experiments have been conducted for three classes of processors: netbook (Intel Atom 330 + NVIDIA ION), laptop (Core 2 Duo + GeForce 320M) and server (Xeon 5500 + Tesla C1060).

Veldema et al. [2011] propose a framework for HC in context of OpenMP adapted to Java, called ClusterJaMP. They propose an array package which provides replicated and partitioned arrays, using which a parallel-for loop can be distributed over PUs. At the beginning of a program, they execute a microbenchmark and a few bandwidth tests, and use this information to estimate relative performance of PUs. Using this, their technique dynamically changes the number and type (GPU or CPU) of PUs to achieve optimal communication/computation ratio for achieving the best possible performance. They perform experiments using an HCS with Xeon 5550 CPU and Tesla M1060 GPU.

Scogland et al. [2012] propose a runtime system for automatically dividing an Accelerated OpenMP region across different PUs. The runtime also handles ensuring data transfer to PUs for input and output. Accelerated OpenMP uses `hetero` clause as a compiler directive, using which a programmer can control whether heterogeneous computing is used; and if so, how many loop iterations are assigned to CPU. They propose enhancing this directive by also providing information on the type of scheduler (static or dynamic), ratio of performance of PUs, and a factor termed as *div* (explained shortly). The static scheduler uses a pre-determined work-division based on specified ratio, while the dynamic scheduler updates this ratio after first execution of parallel region and uses this to make better decisions in future executions. To get even better estimate of value of the ratio; the *div* parameter is used to control the number of it-

erations to be used for getting the estimate of ratio. Evaluations are performed using 12-core Opteron 6174 CPU and Tesla C2050 GPU.

Jiang and Agrawal [2012] present an approach called MATE-CG for accelerating MapReduce applications on parallel heterogeneous environments. Apart from allowing CPU-only and GPU-only execution, MATE-CG runtime also supports dividing the work between CPU and GPU. The amount of data to be processed by CPU and GPU is decided by a partitioning parameter, which can be decided at runtime using an auto-tuning approach based on the iterative characteristic of data-intensive applications. By collecting profiling data over first few phases, the value of this parameter can be found with small overhead. The experimental system uses quad-core Xeon E5520 CPUs and Tesla C2050 GPUs.

Humphrey et al. [2012] present Uintah runtime system, which provides a scheduler for heterogeneous computing. Uintah works on an abstract task-graph representation of parallel computation and communication to take into account data dependencies between tasks. Each task has a C++ method associated with it, which performs the actual computation. A GPU task is represented by an additional C++ method, which is used for setting up GPU kernel and invocation. Uintah's heterogeneous task-scheduler accounts for the dependencies of tasks. It also determines the order of execution and ensures correct inter-process communication. Further, it ensures that any variable is not simultaneously used by multiple running tasks. Further, by using CUDA asynchronous API, Uintah scheduler overlaps data transfer with computation on PUs, and also ensures that before a GPU task is run, the required data are present in the GPU memory. The experiments are performed on two systems, one of which has 6-core Xeon X5660 CPUs with Tesla M2090 GPUs and another has 16-core Opteron 6200 CPUs with Tesla 20-series GPUs.

Odajima et al. [2012] propose a framework for heterogeneous computing using StarPU with XcalableMP-dev parallel programming language. XcalableMP is a a directive-based language extension that supports parallel computing over different nodes in a distributed memory system. XcalableMP-dev extends XcalableMP for clusters equipped with heterogeneous nodes. StarPU is used as the execution engine of XcalableMP-dev. They demonstrate the use of their framework for N-body problem, where the loops are distributed between PUs in proportion to their performance. Their experimental system uses Opteron 6134 CPUs and Tesla C2050 GPUs.

4.3. Compiler level techniques

Luk et al. [2009] propose Qilin framework for mapping computations on an HCS. It provides an API for writing parallelizable programs. Qilin uses a training phase, in which a performance model is created for each task on each PU. Each kernel version is executed with different inputs and a linear model is fitted to the observed runtimes. Using this information, optimal workload division is computed and dynamic compilation is done to instantiate the chosen distribution. Similar to OpenMP, Qilin is built on top of C/C++; however, unlike OpenMP which can exploit parallelism only on CPU, Qilin can exploit parallelism on both CPU and GPU. Their experimental machine uses Core 2 Quad CPU along with GeForce 8800 GTX GPU.

Ravi et al. [2010] propose a compiler and runtime framework for mapping MapReduce class of applications to a system composed of multi-core CPU and GPU. Their framework starts with simple C functions which has annotations added to it. Using this, their framework automatically generates the middleware API code for the CPU and GPU simultaneously. Afterwards, the runtime system uses work-sharing based approach to dynamically partition the work between CPU cores and the GPU. In work-sharing, the scheduler adds the work in a global work list. An idle processor can obtain works from this list. They also observe that since GPUs incur large overhead of kernel

invocation and data transfer, the chunk size of work allocated to them should be large. In contrast, CPU cores are relatively slower but also have smaller latency and hence, to achieve better load-balancing, the chunk size allocated to them should be small. Their evaluation system uses Opteron 8350 CPU with GeForce 9800 GTX GPU.

Prasad et al. [2011] propose a compiler for compiling MATLAB programs to achieve execution on heterogeneous processors. After identifying the data parallel regions of the program, the compiler composes them into kernels. Further, the identified kernels are mapped to a suitable PU so that the execution of kernels happens synergistically and the data transfer overhead is minimized. Their compiler does not require the users to perform manual annotation to specify the arrays whose operations are mapped to the GPU. Use of their compiler provides performance advantage over native execution of MATLAB. Their experimental system uses quad-core Xeon processor and one of the following GPUs: GeForce 8800 GTS and Tesla S1070. Their results show that, in general, Tesla S1070 provides higher speedup than GeForce 8800 GTS. They also note that a few large-sized problems could not be run on GeForce 8800 GTS due to its limited memory capacity.

Pienaar et al. [2012] propose an Automatic Heterogeneous Pipelining (AHP) framework for HCSs. AHP takes a C++ program which has been annotated with compiler directives by the programmer regarding targets of pipelining. Using program annotations, AHP extracts a task graph for each program region. It also profiles the tasks on PUs of HCS to estimate task execution time and inter-task communication time. Then AHP works to iteratively identify pipeline stages, transforms the task graph to reflect the pipeline structure. Afterwards, it maps the pipelined task graph to the HCS and generates code for the optimized heterogeneous pipeline that uses frameworks such as Intel TBB, Cilk, OpenMP and CUDA. Their evaluation platform uses quad-core Xeon E5520 CPU and NVIDIA Tesla C1060 GPU.

Kofler et al. [2013] propose a workload-division technique that uses Insieme source-to-source compiler [Insieme Compiler 2014] to translate an OpenCL program for single-device into an OpenCL program for multiple-devices. Afterwards, workload partitioning is performed using on an offline-generated model, which is based on both static program features (e.g. number of branches, floating point operations etc.) and dynamic program features (e.g. data transfer size and overhead etc.), capabilities of PU and the input data. To achieve best-possible workload-division, they use principle component analysis (PCA) method. The use two HCSs used in their experiments have 1) Opteron 6168 CPU with Radeon HD5870 GPU and 2) Xeon X5650 CPU with GeForce GTX480 GPU.

4.4. Design of OpenCL and related frameworks

OpenCL [Stone et al. 2010] is an open standard and has been adopted by several vendors. It facilitates both task-parallelism and data-parallelism. For each PU, a command queue can be created to which tasks can be submitted. Use of parallel programming can be achieved on each PU individually and on all PUs collectively. It provides a higher abstraction programming framework and hence, it enables the programmer to write programs which are portable across wide variety of processing units, although it may not be able to achieve the highest possible performance on a PU.

Spafford et al. [2010] present a library called Maestro for data orchestration on multiple OpenCL-enabled PUs. The OpenCL task queue model requires the programmer to manage separate task-queues for each PU. This, in turn, requires the programmer to have detailed knowledge of each PU. Maestro uses a single high-level task queue and based on the runtime information, it can automatically transfer data and divide work among PUs. Maestro uses autotuning approach to optimize OpenCL code on each PU and offline profiling to measure relative performance of each PU to decide optimal

work division. Maestro also facilitates overlapping data transfer with computation to achieve pipelining. Their experimental HCS uses Opteron 246 CPU with Radeon HD 5870 GPU.

Kim et al. [2012] present SnuCL which enables OpenCL applications to run in a distributed manner on a cluster. SnuCL makes all the OpenCL compatible PUs (e.g. a CPU or a GPU) on a cluster logically appear on a single local node. SnuCL internally uses MPI but does not require use of MPI in the application source. Using SnuCL, OpenCL applications written for a single node can run on the entire cluster without any modification. Thus, the application can utilize the PUs of a compute node similar to the PUs in the host node. This helps the programmer in utilizing all the PUs in a cluster for heterogeneous computing. Evaluations are performed on a CPU/GPU heterogeneous cluster whose compute node has Xeon X5660 CPU with GeForce GTX 480 GPU.

5. TECHNIQUES FOR IMPROVING ENERGY EFFICIENCY

Table V summarizes the techniques for saving energy in HCSs (DVFS = dynamic voltage/frequency scaling). Note that while many other techniques which aim to improve performance may also save energy, in this table we only show those techniques which have been evaluated on the basis of energy efficiency. We now discuss a few of these techniques.

Table V. A classification of approaches used for energy saving in HCSs

Classification	References
By intelligent workload partitioning and performance improvement	[Choi et al. 2013; Hamano et al. 2009; Liu et al. 2012; Liu and Luk 2012; Luk et al. 2009; Ma et al. 2012; Rofouei et al. 2008; Silberstein and Maruyama 2011; Takizawa et al. 2008]
By DVFS on both CPU and GPU	[Liu et al. 2012; Ma et al. 2012; Wang and Song 2011]
By DVFS on CPU only	[Anzt et al. 2011; Liu et al. 2011]
By resource scaling or low-power modes	[Liu et al. 2011; Silberstein and Maruyama 2011]

5.1. Workload partitioning based techniques

Liu and Luk [2012] propose an approach for utilizing CPU, GPU and FPGA for improving energy efficiency of scientific computing and demonstrate its use for high performance linpack (HPL) benchmark. Their technique uses profiling to record the runtime power consumption and execution time (which includes data transfer time and computation time) of all PUs, along with the power consumption of communication channel. Using these parameters, the problem of finding the right workload division among PUs while optimizing energy efficiency is modeled as a linear programming problem. Their experimental HCS has Xeon W3505 CPU and Tesla C2070 GPU.

Takizawa et al. [2008] propose SPRAT (stream programming with runtime auto-tuning), a runtime environment for improving energy efficiency by workload division. SPRAT uses a performance model which accounts for both computation time of PUs and communication time; and dynamically selects a PU for executing a kernel such that the system energy is minimized. Their experimental system uses Core 2 Quad CPU and one of the three GPUs, which, in decreasing order of performance are: GeForce 8800 GTX, GeForce 8800 GT and GeForce 8600 GTS.

Hamano et al. [2009] propose an HCT which schedules tasks to PUs with the aim of optimizing energy efficiency. Their technique takes into account the performance of tasks on different PUs and energy consumption of PUs when they are busy and idle. Using this, their technique estimates the energy efficiency for different mapping

of tasks to different PUs and chooses the mapping which leads to minimum energy consumption. Their evaluation platform has Phenom 9850 quad-core CPU along with GeForce 8800 GTS GPU.

5.2. DVFS based techniques

Wang and Song [2011] present a technique for saving energy in HCSs. Their technique models the problem of workload division and voltage scaling as an integer linear programming problem, with the objective of minimizing energy consumption (of both PUs and system buses) for a given performance constraint. Voltage scaling and workload division affect each other and also have effect on performance and energy consumption of PUs. Their model also accounts for the communication cost. With the optimal workload division, minimum execution time is obtained with highest voltage levels. Compared to this time, for different percentage of increase in execution time (i.e. performance loss), the energy consumption can be calculated, using which the trade-off between performance and energy consumption can be explored. They perform experiments on a machine equipped with a Core I7-920 Quad-core CPU and a Radeon HD 4870 4870 GPU.

Anzt et al. [2011] propose a technique for saving energy in HCSs while executing iterative linear solvers. After starting a GPU kernel, CPU stays in busy-wait loop and performs no useful work. During this time, CPU energy can be saved by using DVFS (dynamic voltage/frequency scaling) with little performance loss. Further, when the execution time of kernel is large enough; by noting that time once, the CPU can be transitioned to sleep state for that duration in future iterations, i.e. future calls to the kernel since the solver takes nearly same time in different iterations. Their experimental HCS uses an 8-core Opteron 6128 CPU and a Tesla C1060 GPU.

Ma et al. [2012] propose a framework for power management of HCSs. Their technique divides the workload between CPU and GPU based on workload characteristics to achieve load-balancing and reduce idling. Afterwards, the frequency and voltage of CPU and frequency of GPU are adjusted to achieve energy savings with minimal performance degradation. Their experimental system uses a dual core AMD Phenom II X2 CPU along with a GeForce 8800 GTX GPU.

Liu et al. [2012] discuss a workload-division technique for saving energy in an HCS while also meeting task deadlines. Their technique first maps tasks on a CPU or a GPU, such that their deadlines are met and load-balancing can be achieved; and then applies DVFS to both PUs to save energy. When the average-case execution times are shorter than worst-case execution times, extra slack is generated which can be further exploited using DVFS to save extra energy. Their experiments are performed using Xeon 5160 CPU and Radeon HD 5770 GPU.

5.3. Resource scaling based techniques

Liu et al. [2011] propose a technique based on waterfall model for improving energy efficiency of large scale heterogeneous clusters, in which each node may have several CPU-GPU pairs. Their technique transitions the node into one of the three possible power states, namely busy (all CPUs and GPUs of a node are working), spare (at least a single CPU-GPU pair is free) and sleep (all CPU-GPU pairs are free). At the time of reduced workload, a node in sleep state is powered off and at the time of additional workload, a node is woken up. Further, their technique schedules the tasks on available CPU-GPU pair with the view to minimize their execution time; and scales the voltage of CPU to save energy while meeting task deadlines. Finally, to achieve load-balancing, their technique migrates a small fraction of GPU's share of task to the CPU in the same CPU-GPU pair when required. Their performance models are based on Tianhe-1A supercomputer, which uses 6-core Xeon X5670 CPU and Tesla M2050 GPU.

Silberstein and Maruyama [2011] present an algorithm for optimizing energy efficiency of an HCS while running applications which comprise of multiple interdependent tasks. Their algorithm takes into account the energy consumption of each task on CPU and GPU, along with the data transfer cost and constructs a schedule with provably minimal total consumed energy. They assume that both a CPU and a GPU can be powered-off when idle and incur no overhead when they are powered-on. Their experimental platform uses Core 2 Quad CPU and GeForce GTX 285 GPU.

6. FUSED CPU-GPU CHIPS AND COMPARISON WITH DISCRETE SYSTEMS

Discrete GPUs have separate memory spaces from the CPU and hence, data transfer happens over a connection bus (e.g. PCIe bus), which incurs large overhead. Also, these systems require large programming effort, since the programmer has to manage the data that is manipulated by both CPU and GPU. To address these limitations, researchers have designed fused HCSs which feature shared memory space between CPU and GPU. This reduces their data transfer time which especially benefits the applications which require large communication between CPU and GPU. Table VI summarizes the works which compare fused CPU-GPU chips with discrete CPU-GPU systems. We now discuss a few of these works.

Table VI. Classification of works related to fused HCSs

Classification	References
Comparison of fused HCSs with discrete HCSs	[Boyer et al. 2013; Conti et al. 2012; Daga et al. 2011; Delorme 2013; Hetherington et al. 2012; Lee et al. 2012a; Spafford et al. 2012; Sun et al. 2012; Ukidave et al. 2013]
Data transfer overhead evaluation	[Boyer et al. 2013; Conti et al. 2012; Daga et al. 2011; Delorme 2013; Hetherington et al. 2012; Lee et al. 2012a; Spafford et al. 2012]
Energy efficiency evaluation	[Spafford et al. 2012; Ukidave et al. 2013]

Delorme [2013] present two strategies for implementing radix sort on the fused HCS. The coarse-grained implementation divides the data to be sorted between CPU and GPU at the beginning of algorithm. Afterwards, they individually sort the data and finally, the their outputs are merged to produce single sorted output. The fine-grained implementation with dynamic partitioning shares data after each pass of the sorting algorithm. In this implementation, algorithm repartitions data after each step and thus, a piece of data does not belong to a specific PU, rather the entire data must be visible to both PUs during kernel execution. This implementation requires more communication and synchronization than the coarse-grained implementation, however it also provides better performance due to better load-balancing. He also shows that both these implementations outperform the state of the art GPU radix sort implementation, which shows the merit of utilizing CPU for performing computations in an HCS. They perform experiments using two APUs, viz. an AMD A6-3650 APU (with quad-core AMD CPU and Radeon HD 6530D GPU) and an AMD A8-3850 APU (with quad-core AMD CPU and Radeon HD 6550D GPU), of which A8-3850 APU shows higher performance.

Hetherington et al. [2012] compare the performance of Memcached (a key-value store application) on discrete HCS with that on fused HCS. They observe that on discrete systems, the time of data transfer between CPU and GPU becomes a significant fraction of total execution time which negatively affects the performance of GPU. On excluding the data transfer overhead, however, discrete GPU provides large performance gain. Their experiments use two fused HCSs viz. Llano A8-3850 APU (with Radeon HD 6550D GPU) and Zacate E-350 (with Radeon HD 6310 GPU) and one Radeon HD 5870 discrete GPU. They note that the fused chips such as Llano and

Zacate have lower computation power compared to the discrete GPU, however their ability to avoid the transfer of data enables them to provide higher performance than the discrete GPU chip.

Similarly, Spafford et al. [2012] show that low CPU-GPU data transfer overhead is a key advantage of fused HCS which makes its energy efficiency better than that of a discrete GPU. At the same time, the penalty of contention between CPU and GPU is higher for a fused HCS (due to fused memory hierarchy) than for a discrete GPU. Further, compared to a discrete CPU with nearly similar power budget, the fused HCS uses a simpler CPU microarchitecture design and hence, it provides lower performance on computationally intensive tasks than the discrete CPU. Their experiments use A8-3850 Llano APU as the fused HCS and Radeon HD 5670 and FirePro v8800 as two discrete GPUs.

Fused HCSs (or APUs) replace the PCIe data transfer cost by fast memory block transfers between the CPU and GPU memory partitions. Daga et al. [2011] show that due to this, for a kernel benchmark, AMD E-series Zacate APU with only 80 GPU cores provides more than 3× improvement over discrete AMD Radeon HD 5870 GPU which has a total of 1600 more powerful GPU cores. They also show that compared to a discrete GPU, APU reduces the parallelization overhead.

Lee et al. [2012a] compare the performance of memory accesses on a fused HCS and a discrete GPU for different memory access patterns. They note that on the fused HCS, a CPU memory access from the GPU is nearly as fast as a GPU memory access. This provides the opportunity of GPU directly accessing data on the CPU without copying and it especially benefits the applications with little data reuse. For their experiments, they use A8-3850 APU as the fused HCS and Radeon HD5870 as the discrete GPU.

Ukidave et al. [2013] study the performance and energy consumption of discrete and fused HCSs for different FFT (fast Fourier transform) implementations with different input sizes. They show that with growing input data size, the energy efficiency of these HCSs increases due to better utilization of the data-parallel resources on the GPUs. They also show that the power consumption increases with the number of OpenCL kernel calls and increased use of the GPU fetch unit. They perform experiments using two discrete GPUs viz. GeForce GTX 480 and Radeon HD 7770 and two fused systems viz. AMD A8-3850 APU and Core i7-3770 Ivy Bridge processor (which uses Intel Series-4000 embedded GPU).

7. APPLICATION AREAS OF AND BENCHMARKS FOR HETEROGENEOUS COMPUTING

7.1. Application domains of heterogeneous computing

Table VII classifies the works discussed in this paper based on their application domains. These domains are, by no means, exhaustive as heterogeneous computing can be applied in several other domains also which are computationally intensive and/or provide opportunity for parallelization.

7.2. Benchmarks for Heterogeneous Computing

Due to their unique characteristics, HCSs require special benchmarks to gain insights into their functioning. To fulfill this need, several benchmark suites have been proposed which help in meaningfully comparing their architectures and programming environments against similar systems. Table VIII summarizes these benchmarks. Notice that some of these benchmarks provide implementations in different languages which allow comparison using those programming models. We now briefly discuss these benchmark suites.

SHOC (Scalable HeterOgeneous Computing) [Danalis et al. 2010] provides both low-level microbenchmarks (to evaluate architectural features of the system) and applica-

Table VII. Classification of research works based on their application (or workload) domains

Classification	References
Maths, numerical methods and/or algebraic routines	[Agullo et al. 2011; Álvarez-Melcón et al. 2013; Anzt et al. 2011; Becchi et al. 2010; Benner et al. 2011, 2010; Bernabé et al. 2013; Binotto et al. 2010; Boyer et al. 2013; Clarke et al. 2012; Conti et al. 2012; Daga et al. 2011; Danalis et al. 2010; Diamos and Yalamanchili 2008; Dziekonski et al. 2011; Endo et al. 2010; Gregg et al. 2011, 2010; Gummaraju et al. 2010; Hong et al. 2010; Horton et al. 2011; Jiménez et al. 2009; Kim et al. 2012; Kofler et al. 2013; Kothapalli et al. 2013; Lang and Rünger 2013; Lee et al. 2012b; Li et al. 2012; Liu and Luk 2012; Ltaief et al. 2011; Luo et al. 2011; Ma et al. 2013; Mariano et al. 2012; Matam et al. 2012; Meredith et al. 2011; Muramatsu et al. 2011; Ogata et al. 2008; Ohshima et al. 2007; Pai et al. 2010; Pandit and Govindarajan 2014; Papadrakakis et al. 2011; Pienaar et al. 2011; Prasad et al. 2011; Scogland et al. 2012; Siegel et al. 2010; Spafford et al. 2012; Stefanski 2013; Stpiczynski 2011; Stpiczynski and Potiopa 2010; Su et al. 2013; Takizawa et al. 2008; Tomov et al. 2010; Ukidave et al. 2013; Veldema et al. 2011; Venkatasubramanian and Vuduc 2009; Vömel et al. 2012; Wang et al. 2013c; Wen et al. 2012; Yang et al. 2010; Zhong et al. 2012]
Video processing, Imaging and/or computer vision	[Agulleiro et al. 2012; Choudhary et al. 2012; Deshpande et al. 2011; Hartley et al. 2008, 2010; Lecron et al. 2011; Li et al. 2011; Liu et al. 2009; Mistry et al. 2013a; Nigam et al. 2012; Pajot et al. 2011; Park et al. 2011; Pienaar et al. 2012; So et al. 2011; Teodoro et al. 2012, 2013, 2009; Toharia et al. 2012; Tsuda and Nakamura 2011; Wang et al. 2013b]
Data mining, processing and/or database systems	[Banerjee et al. 2012; Banerjee and Kothapalli 2011; Becchi et al. 2010; Breß et al. 2013; Chen et al. 2012; Delorme 2013; Gelado et al. 2010; Gharaibeh et al. 2012; He and Hong 2010; Hetherington et al. 2012; Hong et al. 2011; Jablin et al. 2012; Kothapalli et al. 2013; Lee et al. 2012a; Liu et al. 2011; Munguia et al. 2012; Pandit and Govindarajan 2014; Pienaar et al. 2012; Pirk et al. 2012; Ravi et al. 2010; Shirahata et al. 2010]
Physics	[Hardy et al. 2009; Hawick and Playne 2013; Hermann et al. 2010; Humphrey et al. 2012; Jetley et al. 2010; Joselli et al. 2008; Korwar et al. 2013; Lu et al. 2012a; Nakasato et al. 2012; Panetta et al. 2009; Rahimian et al. 2010; Teodoro et al. 2013; Wu et al. 2012; Yang et al. 2013]
Chemistry	[Bhaskaran-Nair et al. 2013; Hampton et al. 2010; Ma et al. 2013]
Bioinformatics and/or medical science	[Agulleiro et al. 2012; Chai et al. 2013; Chen et al. 2010; Hartley et al. 2008, 2010; Lecron et al. 2011; Li et al. 2011; Liu et al. 2009; Rahimian et al. 2010; Shen et al. 2010; Singh and Aruni 2011; So et al. 2011; Wang et al. 2013b; Xiao et al. 2011; Yao et al. 2010]
Smith-Waterman algorithm	[Chen et al. 2010; Luk et al. 2009; Singh and Aruni 2011]
N-body simulations	[Hu et al. 2011; Jetley et al. 2010; Joselli et al. 2008; Nakasato et al. 2012; Rahimian et al. 2010; Tsoi and Luk 2010]
Electromagnetics	[Álvarez-Melcón et al. 2013; Dziekonski et al. 2011; Gao et al. 2012; Stefanski 2013]

Table VIII. A classification of benchmark suites for heterogeneous computing

Name	Languages/versions	Scaled using MPI?	Example Application Fields
SHOC	serial, OpenCL, CUDA	Yes	scientific computing , linear algebra, molecular dynamics etc.
Parboil	serial, OpenMP, OpenCL and CUDA	No	image processing, bio-molecular simulation, astronomy and fluid dynamics
Rodinia	OpenCL, OpenMP and CUDA	No	medical imaging, bioinformatics, data mining, molecular dynamics
Valar	OpenCL	No	computer vision, digital signal processing, data mining and physics

tion kernels (to evaluate the features of the system such as intranode and internode communication between PUs). In addition to serial version, SHOC provides an embarrassingly parallel version (which executes on different PUs or nodes of a cluster, but have no communication between PUs or nodes), and a true parallel version (which measures multiple nodes, with single or multiple PUs per node, and also involves communication). Thus, SHOC benchmarks scale effectively across a single PU to a large cluster. For the same fundamental algorithm, Parboil [Stratton et al. 2012] provides versions of varying levels of optimizations. This feature enables the compiler writers to evaluate source and compiler optimizations on different architectures.

The choice of programs in Rodinia [Che et al. 2009] is based on Berkeley's dwarf taxonomy and is aimed at covering different types of parallel communication patterns and synchronization techniques. Valar [Mistry et al. 2013b] benchmark suite provides OpenCL applications for studying interaction of processing units in HCSs. Programs in this suite are categorized based on whether the majority of algorithm execution is done on a single PU, or on multiple PUs with or without substantial communication. Also, the programs are characterized by whether their behavior is latency sensitive, streaming-type or has QoS (quality-of-service) requirement.

8. CONCLUSION AND FUTURE DIRECTIONS

In recent years, CPUs and GPUs are increasingly being seen as indispensable coprocessors, instead of substitutes for each other. As a result, heterogeneous computing has been actively researched and utilized in variety of applications ranging from natural sciences to engineering etc. Several challenges still remain, and we believe that the research in near future will provide solutions to them. We now briefly mention some directions for future research.

Many algorithm-level techniques require the programmer to manually partition the workload between PUs, identify the subtasks suitable for each PU and/or profile the performance of each PU. This is a tedious process which does not scale well beyond small problems. Fully-automatic toolchains are required to extend the benefits of heterogeneous computing to a wide range of application domains.

Several existing large-scale codes are written in either CPU or GPU languages. Porting them to HCSs will incur large overhead and is also error-prone. Significant amount of work needs to be done before recently introduced HCS programming frameworks can reach the level of maturity, robustness and popularity as enjoyed by the conventional CPU programming languages.

Mobile computing systems, which now number nearly equal to the population of the earth [International Telecommunication Union 2012], generally provide multimedia services under real-time performance constraints and hence they are attractive targets for acceleration through the use of heterogeneous computing. However, mobile systems also impose very stringent cost and power budgets [Mittal 2014c]. While the large cost and power consumption of HCSs may be acceptable in high-end computing systems, using them in mobile systems calls for an order-of-magnitude improvement in their energy efficiency.

While fused HCSs address several limitations of discrete HCSs, they also introduce new tradeoffs and require the design of suitable data-access strategies to take full advantage of their fast interconnection. Moreover, existing fused HCSs are not strictly superior to discrete GPUs on all parameters and hence, despite their potential, extensive evaluation and development on them has been lacking. Novel solutions are required at device, architecture, programming and system level before the fused HCSs can become commonplace in mainstream computing systems.

In this paper, we have synthesized the research work on heterogeneous computing by showing the broad spectrum of heterogeneous computing techniques, their key re-

search ideas and applications. We surveyed research works at different levels of abstraction, viz. algorithm, runtime, compiler, programming model etc. We discussed both both fused and discrete CPU-GPU systems along with the benchmark suits proposed to evaluate them. It is hoped that this paper will be highly beneficial to computer architects, researchers and application-developers and will inspire novel ideas and open promising research avenues.

REFERENCES

Alejandro Acosta, Robert Corujo, Vicente Blanco, and Francisco Almeida. 2010. Dynamic load balancing on heterogeneous multicore/multi-GPU systems. In *International Conference on High Performance Computing and Simulation (HPCS)*. 467–476.

JI Agulleiro, F Vazquez, EM Garzon, and JJ Fernandez. 2012. Hybrid computing: CPU+ GPU co-processing and its application to tomographic reconstruction. *Ultramicroscopy* 115 (2012), 109–114.

Emmanuel Agullo, Cédric Augonnet, Jack Dongarra, Mathieu Faverge, Hatem Ltaief, Samuel Thibault, and Stanimire Tomov. 2011. QR factorization on a multicore node enhanced with multiple GPU accelerators. *Int. Parallel & Distributed Processing Symp.* (2011), 932–943.

Omer Erdil Albayrak, Ismail Akturk, and Ozcan Ozturk. 2012. Effective Kernel Mapping for OpenCL Applications in Heterogeneous Platforms. In *41st International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 81–88.

Alejandro Álvarez-Melcón, Domingo Giménez, Fernando D Quesada, and Tomás Ramírez. 2013. Hybrid-Parallel Algorithms for 2D Green's Functions. *Procedia Computer Science* 18 (2013), 541–550.

Hartwig Anzt, Vincent Heuveline, José I Aliaga, Maribel Castillo, Juan C Fernandez, Rafael Mayo, and Enrique S Quintana-Orti. 2011. Analysis and optimization of power consumption in the iterative solution of sparse linear systems on multi-core and many-core platforms. In *International Green Computing Conference and Workshops (IGCC)*. IEEE, 1–6.

Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. 2011. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience* 23, 2 (2011), 187–198.

Eduard Ayguade, RosaM. Badia, Daniel Cabrera, Alejandro Duran, Marc Gonzalez, Francisco Igual, Daniel Jimenez, Jesus Labarta, Xavier Martorell, Rafael Mayo, JosepM. Perez, and EnriqueS Quintana-Ort. 2009. A proposal to extend the OpenMP tasking model for heterogeneous architectures. In *Evolving OpenMP in an Age of Extreme Parallelism*. 154–167.

Ana Balevic and Bart Kienhuis. 2011. An Efficient Stream Buffer Mechanism for Dataflow Execution on Heterogeneous Platforms with GPUs. In *First Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM)*. IEEE, 53–57.

Dip Sankar Banerjee, Aman Kumar Bahl, and Kishore Kothapalli. 2012. An On-Demand Fast Parallel Pseudo Random Number Generator with Applications. In *InternationalParallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. 1703–1711.

Dip Sankar Banerjee and Kishore Kothapalli. 2011. Hybrid algorithms for list ranking and graph connected components. In *International Conference on High Performance Computing*. 1–10.

Michela Becchi, Surendra Byna, Srihari Cadambi, and Srimat Chakradhar. 2010. Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory. In *22nd ACM symposium on Parallelism in algorithms and architectures*. 82–91.

Mehmet E Belviranli, Laxmi N Bhuyan, and Rajiv Gupta. 2013. A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures. *ACM Transactions on Architecture and Code Optimization (TACO)* 9, 4 (2013), 57.

Peter Benner, Pablo Ezzatti, Daniel Kressner, Enrique S Quintana-Ortí, and Alfredo Remón. 2011. A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU–GPU platforms. *Parallel Comput.* 37, 8 (2011), 439–450.

Peter Benner, Pablo Ezzatti, Enrique S Quintana-Ortí, and Alfredo Remón. 2010. Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function. In *Euro-Par 2009–Parallel Processing Workshops*. 132–139.

Gregorio Bernabé, Javier Cuenca, and Domingo Giménez. 2013. Optimization techniques for 3D-FWT on systems with manycore GPUs and multicore CPUs. *Procedia Computer Science* 18 (2013), 319–328.

Kiran Bhaskaran-Nair, Wenjing Ma, Sriram Krishnamoorthy, Oreste Villa, Hubertus JJ van Dam, Edoardo Aprà, and Karol Kowalski. 2013. Non-iterative Multireference Coupled Cluster Methods on Heterogeneous CPU-GPU Systems. *J. of Chemical Theory and Computation* (2013).

Alecio PD Binotto, Christian Daniel, Daniel Weber, Arjan Kuijper, Andre Stork, Carlos Pereira, and Dieter Fellner. 2010. Iterative SLE solvers over a CPU-GPU platform. In *Int. Conf. on High Performance Computing and Communications*. 305–313.

Alecio PD Binotto, Carlos E Pereira, Arjan Kuijper, Andre Stork, and Dieter W Fellner. 2011. An effective dynamic scheduling runtime and tuning system for heterogeneous multi and many-core desktop platforms. In *IEEE 13th International Conference on High Performance Computing and Communications (HPCC)*. 78–85.

Murilo Boratto, Pedro Alonso, Carla Ramiro, and Marcos Barreto. 2012. Heterogeneous computational model for landform attributes representation on multicore and multi-GPU systems. *Procedia Computer Science* 9 (2012), 47–56.

Michael Boyer, Kevin Skadron, Shuai Che, and Nuwan Jayasena. 2013. Load balancing in a changing world: dealing with heterogeneity and performance variability. In *ACM International Conference on Computing Frontiers*.

Alexander Branover, Denis Foley, and Maurice Steinman. 2012. AMD Fusion APU: Llano. *Micro, IEEE* 32, 2 (2012), 28–37.

Sebastian Breß, Felix Beier, Hannes Rauhe, Kai-Uwe Sattler, Eike Schallehn, and Gunter Saake. 2013. Efficient Co-Processor Utilization in Database Query Processing. In *Information Systems*.

Jun Chai, Huayou Su, Mei Wen, Xing Cai, Nan Wu, and Chunyuan Zhang. 2013. Resource-efficient utilization of CPU/GPU-based heterogeneous supercomputers for Bayesian phylogenetic inference. *The Journal of Supercomputing* (2013), 1–17.

Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization*. 44–54.

Bo Chen, Yun Xu, Jiaoyun Yang, and Haitao Jiang. 2010. A new parallel method of smith-waterman algorithm on a heterogeneous platform. In *Algorithms and Architectures for Parallel Processing*. Springer, 79–90.

Linchuan Chen, Xin Huo, and Gagan Agrawal. 2012. Accelerating MapReduce on a coupled CPU-GPU architecture. In *SC*. 25:1–25:11.

Hong Jun Choi, Dong Oh Son, Seung Gu Kang, Jong Myon Kim, Hsien-Hsin Lee, and Cheol Hong Kim. 2013. An efficient scheduling scheme using estimated execution time for heterogeneous computing systems. *The Journal of Supercomputing* (2013), 1–17.

Siddharth Choudhary, Shubham Gupta, and PJ Narayanan. 2012. Practical time bundle adjustment for 3D reconstruction on the GPU. In *Trends and Topics in Computer Vision*. 423–435.

David Clarke, Aleksandar Ilic, Alexey Lastovetsky, and Leonel Sousa. 2012. Hierarchical partitioning algorithm for scientific computing on highly heterogeneous CPU+ GPU clusters. In *Euro-Par Parallel Processing*. Springer, 489–501.

Christian Conti, Diego Rossinelli, and Petros Koumoutsakos. 2012. GPU and APU computations of Finite Time Lyapunov Exponent fields. *J. Comput. Phys.* 231, 5 (2012), 2229–2244.

JR da S Junior, Esteban W Clua, Anselmo Montenegro, and Paulo A Pagliosa. 2010. Fluid simulation with two-way interaction rigid body using a heterogeneous GPU and CPU environment. In *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*. IEEE, 156–164.

Mayank Daga, Ashwin M Aji, and Wu-chun Feng. 2011. On the efficacy of a fused CPU+ GPU processor (or APU) for parallel computing. In *Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*. IEEE, 141–149.

Satish Damaraju, Varghese George, Sanjeev Jahagirdar, Tanveer Khondker, Robert Milstrey, Sanjib Sarkar, Scott Siers, Israel Stolero, and Arun Subbiah. 2012. A 22nm IA multi-CPU and GPU system-on-chip. In *International Solid-State Circuits Conference*. 56–57.

Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *GPGPU*. 63–74.

Michael Christopher Delorme. 2013. Parallel Sorting on the Heterogeneous AMD Fusion Accelerated Processing Unit. (2013). Master of Applied Science Thesis, University of Toronto.

Aditya Deshpande, Ishan Misra, and PJ Narayanan. 2011. Hybrid implementation of error diffusion dithering. In *International Conference on High Performance Computing*. 1–10.

Gregory F Diamos and Sudhakar Yalamanchili. 2008. Harmony: an execution model and runtime for heterogeneous many core systems. In *High Performance Distributed Computing*. 197–200.

Shuai Ding, Jinru He, Hao Yan, and Torsten Suel. 2009. Using graphics processors for high performance IR query processing. In *Int. Conf. on World wide web*. 421–430.

Adam Dziekonski, Adam Lamecki, and Michal Mrozowski. 2011. Tuning a hybrid GPU-CPU V-Cycle multilevel preconditioner for solving large real and complex systems of FEM equations. *IEEE Antennas and Wireless Propagation Letters* 10 (2011), 619–622.

Toshio Endo, Akira Nukada, Satoshi Matsuoka, and Naoya Maruyama. 2010. Linpack evaluation on a supercomputer with heterogeneous accelerators. In *International Symposium on Parallel & Distributed Processing (IPDPS)*. 1–8.

Eric J Fluhr, Joshua Friedrich, Daniel Dreps, Victor Zyuban, Gregory Still, Christopher Gonzalez, Allen Hall, David Hogenmiller, Frank Malgioglio, Ryan Nett, and others. 2014. 5.1 POWER8^{TM}: A 12-core server-class processor in 22nm SOI with 7.6 Tb/s off-chip bandwidth. In *International Solid-State Circuits Conference (ISSCC)*. 96–97.

Peng Cheng Gao, Yu Bo Tao, Zhi Hui Bai, and Hai Lin. 2012. Mapping the SBR and TW-ILDCs to heterogeneous CPU-GPU architecture for fast computation of electromagnetic scattering. *Progress In Electromagnetics Research* 122 (2012), 137–154.

Michael T Garba and horacio González-vélez. 2012. Asymptotic peak utilisation in heterogeneous parallel CPU/GPU pipelines: a decentralised queue monitoring strategy. *Parallel Processing Letters* 22, 02 (2012).

Eric Gardner. 2014. https://software.intel.com/en-us/articles/what-disclosures-has-intel-made-about-knights-landing. (2014).

Isaac Gelado, John E Stone, Javier Cabezas, Sanjay Patel, Nacho Navarro, and Wen-mei W Hwu. 2010. An asymmetric distributed shared memory model for heterogeneous parallel systems. In *ACM SIGARCH Computer Architecture News*, Vol. 38. 347–358.

Abdullah Gharaibeh, Lauro Beltrão Costa, Elizeu Santos-Neto, and Matei Ripeanu. 2012. A yoke of oxen and a thousand chickens for heavy lifting graph processing. In *International Conference on Parallel Architectures and Compilation Techniques*. ACM, 345–354.

Green500. 2014. Green500 Supercomputers. Retrieved from www.green500.org. (2014).

Chris Gregg, Michael Boyer, Kim Hazelwood, and Kevin Skadron. 2011. Dynamic heterogeneous scheduling decisions using historical runtime data. In *2nd workshop on applications for multi-and many-core processors*.

Chris Gregg, Jeff Brantley, and Kim Hazelwood. 2010. Contention-aware scheduling of parallel code for heterogeneous systems. In *USENIX Workshop on Hot Topics in Parallelism, HotPar*.

Chris Gregg and Kim Hazelwood. 2011. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *ISPASS*. 134–144.

Dominik Grewe and Michael FP O'Boyle. 2011. A static task partitioning approach for heterogeneous systems using OpenCL. In *Compiler Construction*. Springer, 286–305.

Jayanth Gummaraju, Laurent Morichetti, Michael Houston, Ben Sander, Benedict R Gaster, and Bixia Zheng. 2010. Twin peaks: a software platform for heterogeneous computing on general-purpose and graphics processors. In *Parallel Architectures and Compilation Techniques (PACT)*. 205–216.

Sumit Gupta. 2014. http://blogs.nvidia.com/blog/2014/03/25/gpu-roadmap-pascal/. (2014).

Tomoaki Hamano, Toshio Endo, and Satoshi Matsuoka. 2009. Power-aware dynamic task scheduling for heterogeneous accelerated clusters. In *Int. Symp. on Parallel & Distributed Processing*. 1–8.

Scott S Hampton, Sadaf R Alam, Paul S Crozier, and Pratul K Agarwal. 2010. Optimal utilization of heterogeneous resources for biomolecular simulations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–11.

David J Hardy, John E Stone, and Klaus Schulten. 2009. Multilevel summation of electrostatic potentials using graphics processing units. *Parallel Comput.* 35, 3 (2009), 164–177.

Timothy DR Hartley, Umit Catalyurek, Antonio Ruiz, Francisco Igual, Rafael Mayo, and Manuel Ujaldon. 2008. Biomedical image analysis on a cooperative cluster of GPUs and multicores. In *22nd annual international conference on Supercomputing*. 15–25.

Timothy DR Hartley, Erik Saule, and Umit V Catalyurek. 2010. Automatic dataflow application tuning for heterogeneous systems. In *International Conference on High Performance Computing (HiPC)*. 1–10.

K. A. Hawick and D. P. Playne. 2013. Parallel Algorithms for Hybrid Multi-core CPU-GPU Implementations of Component Labelling in Critical Phase Models. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. 45–51.

Zhengyu He and Bo Hong. 2010. Dynamically tuned push-relabel algorithm for the maximum flow problem on CPU-GPU-hybrid platforms. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. 1–10.

Everton Hermann, Bruno Raffin, François Faure, Thierry Gautier, and Jérémie Allard. 2010. Multi-GPU and multi-CPU parallelization for interactive physics simulations. In *Euro-Par 2010-Parallel Processing*. Springer, 235–246.

Tayler H Hetherington, Timothy G Rogers, Lisa Hsu, Mike O'Connor, and Tor M Aamodt. 2012. Characterizing and evaluating a key-value store application on heterogeneous CPU-GPU systems. In *ISPASS*. 88–98.

Chuntao Hong, Dehao Chen, Wenguang Chen, Weimin Zheng, and Haibo Lin. 2010. MapCG: writing parallel program portable between CPU and GPU. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 217–226.

Sungpack Hong, Tayo Oguntebi, and Kunle Olukotun. 2011. Efficient parallel graph exploration on multi-core CPU and GPU. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 78–88.

Mitch Horton, Stanimire Tomov, and Jack Dongarra. 2011. A class of hybrid lapack algorithms for multicore and GPU architectures. In *Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*. IEEE, 150–158.

Qi Hu, Nail A Gumerov, and Ramani Duraiswami. 2011. Scalable fast multipole methods on distributed heterogeneous architectures. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 36.

Alan Humphrey, Qingyu Meng, Martin Berzins, and Todd Harman. 2012. Radiation modeling using the Uintah heterogeneous CPU/GPU runtime system. In *Conf. of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*.

Xin Huo, Vignesh T Ravi, and Gagan Agrawal. 2011. Porting irregular reductions on heterogeneous CPU-GPU configurations. In *Int. Conf. on High Performance Computing*. 1–10.

Thomas B Jablin, James A Jablin, Prakash Prabhu, Feng Liu, and David I August. 2012. Dynamically managed data for CPU-GPU architectures. In *International Symposium on Code Generation and Optimization (CGO)*. 165–174.

Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V Kalé, and Thomas R Quinn. 2010. Scaling hierarchical N-body simulations on GPU clusters. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

Wei Jiang and Gagan Agrawal. 2012. Mate-CG: A map reduce-like framework for accelerating data-intensive computations on heterogeneous clusters. In *IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*. 644–655.

Víctor J Jiménez, Lluís Vilanova, Isaac Gelado, Marisa Gil, Grigori Fursin, and Nacho Navarro. 2009. Predictive runtime code scheduling for heterogeneous architectures. In *High Performance Embedded Architectures and Compilers*. Springer, 19–33.

Mark Joselli, Marcelo Zamith, Esteban Clua, Anselmo Montenegro, Aura Conci, Regina Leal-Toledo, Luis Valente, Bruno Feijó, Marcos d'Ornellas, and Cesar Pozzer. 2008. Automatic dynamic task distribution between CPU and GPU for real-time systems. In *International Conference on Computational Science and Engineering*. 48–55.

Jungwon Kim, Sangmin Seo, Jun Lee, Jeongho Nah, Gangwon Jo, and Jaejin Lee. 2012. SnuCL: an OpenCL framework for heterogeneous CPU/GPU clusters. In *ICS*. 341–352.

Klaus Kofler, Ivan Grasso, Biagio Cosenza, and Thomas Fahringer. 2013. An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning. In *ICS*.

Sai Kiran Korwar, Sathish Vadhiyar, and Ravi S Nanjundiah. 2013. GPU-enabled efficient executions of radiation calculations in climate modeling. In *International Conference on High Performance Computing (HiPC)*. IEEE, 353–361.

Kishore Kothapalli, Dip Sankar Banerjee, PJ Narayanan, Surinder Sood, Aman Kumar Bahl, Shashank Sharma, Shrenik Lad, Krishna Kumar Singh, Kiran Matam, Sivaramakrishna Bharadwaj, Rohit Nigam, Parikshit Sakurikar, Aditya Deshpande, Ishan Misra, Siddharth Choudhary, and Shubham Gupta. 2013. CPU and/or GPU: Revisiting the GPU Vs. CPU Myth. *arXiv preprint arXiv:1303.2171* (2013).

Jens Lang and Gudula Rünger. 2013. Dynamic Distribution of Workload Between CPU and GPU for a Parallel Conjugate Gradient Method in an Adaptive FEM. *Procedia Computer Science* 18 (2013), 299–308.

Fabian Lecron, Sidi Ahmed Mahmoudi, Mohammed Benjelloun, Saïd Mahmoudi, and Pierre Manneback. 2011. Heterogeneous computing for vertebra detection and segmentation in X-ray images. *Journal of Biomedical Imaging* 2011 (2011), 5.

Changmin Lee, Won W Ro, and Jean-Luc Gaudiot. 2012b. Cooperative heterogeneous computing for parallel processing on CPU/GPU hybrids. In *16th Workshop on Interaction between Compilers and Computer Architectures (INTERACT)*. IEEE, 33–40.

Janghaeng Lee, Mehrzad Samadi, Yongjun Park, and Scott Mahlke. 2013. Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 245–256.

Kenneth Lee, Heshan Lin, and Wu-chun Feng. 2012a. Performance characterization of data-intensive kernels on AMD Fusion architectures. *Computer Science-Research and Development* (2012), 1–10.

Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. 2010. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In *International Symposium on Computer Architecture (ISCA)*. 451–460.

Hung-Fu Li, Tyng-Yeu Liang, and Jun-Yao Chiu. 2013. A compound OpenMP/MPI program development toolkit for hybrid CPU/GPU clusters. In *Journal of Supercomputing*. 1–25.

Jiajia Li, Xingjian Li, Guangming Tan, Mingyu Chen, and Ninghui Sun. 2012. An optimized large-scale hybrid DGEMM design for CPUs and ATI GPUs. In *ICS*. 377–386.

Linchuan Li, Xingjian Li, Guangming Tan, Mingyu Chen, and Peiheng Zhang. 2011. Experience of parallelizing cryo-EM 3D reconstruction on a CPU-GPU heterogeneous system. In *20th International Symposium on High performance distributed computing*. 195–204.

Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. 2012. Power-efficient time-sensitive mapping in heterogeneous systems. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 23–32.

Ding Liu, Ruixuan Li, Xiwu Gu, Kunmei Wen, Heng He, and Guoqiang Gao. 2011. Fast Snippet Generation Based On CPU-GPU Hybrid System. In *International Conference on Parallel and Distributed Systems (ICPADS)*. 252–259.

Qiang Liu and Wayne Luk. 2012. Heterogeneous systems for energy efficient scientific computing. In *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 64–75.

Wenjie Liu, Zhihui Du, Yu Xiao, David A Bader, and Chen Xu. 2011. A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters. In *International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*. 82–92.

Yixun Liu, Andriy Fedorov, Ron Kikinis, and Nikos Chrisochoides. 2009. Real-time non-rigid registration of medical images on a cooperative parallel architecture. In *IEEE International Conference on Bioinformatics and Biomedicine*. 401–404.

Hatem Ltaief, Stanimire Tomov, Rajib Nath, Peng Du, and Jack Dongarra. 2011. A scalable high performant Cholesky factorization for multicore with GPU accelerators. In *High Performance Computing for Computational Science–VECPAR 2010*. Springer, 93–101.

Fengshun Lu, Junqiang Song, Xiaoqun Cao, and Xiaoqian Zhu. 2012a. CPU/GPU computing for long-wave radiation physics on large GPU clusters. *Computers & Geosciences* 41 (2012), 47–55.

Fengshun Lu, Junqiang Song, Fukang Yin, and Xiaoqian Zhu. 2012b. Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters. *Computer physics communications* 183, 6 (2012), 1172–1181.

Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. 2009. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *International Symposium on Microarchitecture (MICRO)*. 45–55.

Li Luo, Chao Yang, Yubo Zhao, and Xiao-Chuan Cai. 2011. A scalable hybrid algorithm based on domain decomposition and algebraic multigrid for solving partial differential equations on a cluster of CPU/GPUs. In *Int. Workshop on GPUs and Scientific Applications*. 45–50.

Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *International Conference on Parallel Processing (ICPP)*. 48 –57.

Wenjing Ma, Sriram Krishnamoorthy, Oreste Villa, Karol Kowalski, and Gagan Agrawal. 2013. Optimizing tensor contraction expressions for hybrid CPU-GPU execution. *Cluster Computing* (2013), 1–25.

Artur Mariano, Ricardo Alves, Joao Barbosa, Luis Paulo Santos, and Alberto Proenca. 2012. A (ir) regularity-aware task scheduler for heterogeneous platforms. In *International Conference on High Performance Computing*.

K Matam, SK Bharadwaj, and Kishore Kothapalli. 2012. Sparse Matrix Matrix Multiplication on Hybrid CPU+ GPU Platforms. In *High Performance Computing Conference (HiPC)*.

Insieme Compiler. 2014. http://www.dps.uibk.ac.at/insieme/index.html. (2014).

International Telecommunication Union. 2012. http://www.itu.int/dms_pub/itu-d/opb/ind/D-IND-ICTOI-2012-SUM-PDF-E.pdf. (2012).

OpenACC Standard. 2014. Retrieved from http://www.openacc-standard.org/. (2014).

OpenMP 4.0. 2014. Retrieved from http://openmp.org/wp/2013/07/openmp-40/. (2014).

Jeremy S Meredith, Philip C Roth, Kyle L Spafford, and Jeffrey S Vetter. 2011. Performance implications of nonuniform device topologies in scalable heterogeneous architectures. *IEEE Micro* 31, 5 (2011), 66–75.

Perhaad Mistry, Yash Ukidave, Dana Schaa, and David Kaeli. 2013a. A Framework for Profiling and Performance Monitoring of Heterogeneous Applications. *Programmability Issues for Heterogeneous Multicores (MULTIPROG-2013)* (2013).

Perhaad Mistry, Yash Ukidave, Dana Schaa, and David Kaeli. 2013b. Valar: a benchmark suite to study the dynamic behavior of heterogeneous systems. In *GPGPU*. 54–65.

Sparsh Mittal. 2012. A Survey of Architectural Techniques For DRAM Power Management. *International Journal of High Performance Systems Architecture* 4, 2 (2012), 110–119.

Sparsh Mittal. 2014a. A Survey Of Techniques for Managing and Leveraging Caches in GPUs. *Journal of Circuits, Systems, and Computers (JCSC)* 23, 8 (2014).

Sparsh Mittal. 2014b. A Survey of Architectural Techniques For Improving Cache Power Efficiency. *Elsevier Sustainable Computing: Informatics and Systems* 4, 1 (2014), 33–43.

Sparsh Mittal. 2014c. A Survey of Techniques For Improving Energy Efficiency in Embedded Computing Systems. *International Journal of Computer Aided Engineering and Technology (IJCAET)* (2014).

Sparsh Mittal and Jeffrey S. Vetter. 2015. A Survey of Methods for Analyzing and Improving GPU Energy Efficiency. *Comput. Surveys* 47, 2 (2015), 19:1–19:23.

Timothy Prickett Morgan. 2014. http://www.enterprisetech.com/2014/08/13/oracle-cranks-cores-32-sparc-m7-chip/. (2014).

Lluis-Miquel Munguia, David A Bader, and Eduard Ayguade. 2012. Task-based parallel breadth-first search in heterogeneous environments. In *19th International Conference on High Performance Computing (HiPC)*. 1–10.

Jun-ichi Muramatsu, Takeshi Fukaya, Shao-Liang Zhang, Kinji Kimura, and Yusaku Yamamoto. 2011. Acceleration of Hessenberg Reduction for Nonsymmetric Eigenvalue Problems in a Hybrid CPU-GPU Computing Environment. *International Journal of Networking and Computing* 1, 2 (2011).

Alin Murarașu, Josef Weidendorfer, and Arndt Bode. 2012. Workload balancing on heterogeneous systems: a case study of sparse grid interpolation. In *Euro-Par 2011: Parallel Processing Workshops*. Springer, 345–354.

Naohito Nakasato, Go Ogiya, Yohei Miki, Masao Mori, and Ken'ichi Nomoto. 2012. Astrophysical Particle Simulations on Heterogeneous CPU-GPU Systems. In *arXiv preprint arXiv:1206.1199*.

Andrew Nere, Sean Franey, Atif Hashmi, and Mikko Lipasti. 2012. Simulating cortical networks on heterogeneous multi-GPU systems. *J. Parallel and Distrib. Comput.* (2012).

Rohit Nigam, , and P.J. Narayanan. 2012. Hybrid Ray Tracing and Path Tracing of Bezier Surfaces Using a Mixed Hierarchy. In *Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)*. 35:1–35:8.

NVIDIA. 2015. http://www.geforce.com/hardware/desktop-gpus. (2015).

Tetsuya Odajima, Taisuke Boku, Toshihiro Hanawa, Jinpil Lee, and Mitsuhisa Sato. 2012. GPU/CPU Work Sharing with Parallel Language XcalableMP-dev for Parallelized Accelerated Computing. In *Int. Conf. on Parallel Processing Workshops (ICPPW)*. 97–106.

Yasuhito Ogata, Toshio Endo, Naoya Maruyama, and Satoshi Matsuoka. 2008. An efficient, model-based CPU-GPU heterogeneous FFT library. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. 1–10.

Satoshi Ohshima, Kenji Kise, Takahiro Katagiri, and Toshitsugu Yuba. 2007. Parallel processing of matrix multiplication in a CPU and GPU heterogeneous environment. In *High Performance Computing for Computational Science-VECPAR 2006*. Springer, 305–318.

Edson Luiz Padoin, Laércio Lima Pilla, Francieli Zanon Boito, Rodrigo Virote Kassick, Pedro Velho, and Philippe OA Navaux. 2012. Evaluating application performance and energy consumption on hybrid CPU+ GPU architecture. *Cluster Computing* (2012), 1–15.

Sreepathi Pai, R Govindarajan, and MJ Thazhuthaveetil. 2010. PLASMA: portable programming for SIMD heterogeneous accelerators. *Workshop on Language, Compiler, and Architecture Support for GPGPU* (2010).

Anthony Pajot, Loïc Barthe, Mathias Paulin, and Pierre Poulin. 2011. Combinatorial Bidirectional Path-Tracing for Efficient Hybrid CPU/GPU Rendering. In *Computer Graphics Forum*, Vol. 30. 315–324.

Prasanna Pandit and R Govindarajan. 2014. Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices. In *International Symposium on Code Generation and Optimization (CGO)*. Article 273, 273:273–273:283 pages.

Jairo Panetta, Thiago Teixeira, Paulo RP de Souza Filho, Carlos A da Cunha Finho, David Sotelo, F da Motta, Silvio Sinedino Pinheiro, I Pedrosa, Andre L Romanelli Rosa, Luiz R Monnerat, , L.T. Carneiro, and C.H.B. de Albrecht. 2009. Accelerating Kirchhoff migration by CPU and GPU cooperation. In *SBAC-PAD*. 26–32.

M Papadrakakis, G Stavroulakis, and A Karatarakis. 2011. A new era in scientific computing: domain decomposition methods in hybrid CPU–GPU architectures. *Computer Methods in Applied Mechanics and Engineering* 200, 13 (2011), 1490–1508.

Song Jun Park, James A Ross, Dale R Shires, David A Richie, Brian J Henz, and Lam H Nguyen. 2011. Hybrid core acceleration of UWB SIRE radar signal processing. *IEEE Transactions on Parallel and Distributed Systems* 22, 1 (2011), 46–57.

Phitchaya Mangpo Phothilimthana, Jason Ansel, Jonathan Ragan-Kelley, and Saman Amarasinghe. 2013. Portable Performance on Heterogeneous Architectures. In *International conference on Architectural support for programming languages and operating systems*. 431–444.

Jacques A Pienaar, Srimat Chakradhar, and Anand Raghunathan. 2012. Automatic generation of software pipelines for heterogeneous parallel systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–12.

Jacques A Pienaar, Anand Raghunathan, and Srimat Chakradhar. 2011. MDR: performance model driven runtime for heterogeneous parallel platforms. In *ACM International conference on Supercomputing*. 225–234.

Holger Pirk, Thibault Sellam, Stefan Manegold, and Martin Kersten. 2012. X-device query processing by bitwise distribution. In *Eighth International Workshop on Data Management on New Hardware*. ACM, 48–54.

Usman Pirzada. 2015. wccftech.com/nvidia-gtx-titan-x-revealed-gdc-2015/. (2015).

Matthew Poremba, Sparsh Mittal, Dong Li, Jeffrey Vetter, and Yuan Xie. 2015. DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches. In *DATE*. 1543 – 1546.

Ashwin Prasad, Jayvant Anantpur, and R Govindarajan. 2011. Automatic compilation of MATLAB programs for synergistic execution on heterogeneous processors. In *ACM Sigplan Notices*, Vol. 46. 152–163.

Abtin Rahimian, Ilya Lashuk, Shravan Veerapaneni, Aparna Chandramowlishwaran, Dhairya Malhotra, Logan Moon, Rahul Sampath, Aashay Shringarpure, Jeffrey Vetter, Richard Vuduc, Denis Zorin, and George Biros. 2010. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *SC*. 1–11.

Vignesh T Ravi and Gagan Agrawal. 2011. A dynamic scheduling framework for emerging heterogeneous systems. In *Int. Conf. on High Performance Computing (HiPC)*. 1–10.

Vignesh T Ravi, Wenjing Ma, David Chiu, and Gagan Agrawal. 2010. Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. In *24th ACM International Conference on Supercomputing*. 137–146.

Vignesh T Ravi, Wenjing Ma, David Chiu, and Gagan Agrawal. 2012. Compiler and runtime support for enabling reduction computations on heterogeneous systems. *Concurrency and Computation: Practice and Experience* 24, 5 (2012), 463–480.

Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William Kaiser, and Majid Sarrafzadeh. 2008. Energy-aware high performance computing with graphic processing units. In *Workshop on power aware computing and system*.

Bratin Saha, Xiaocheng Zhou, Hu Chen, Ying Gao, Shoumeng Yan, Mohan Rajagopalan, Jesse Fang, Peinan Zhang, Ronny Ronen, and Avi Mendelson. 2009. Programming model for a heterogeneous x86 platform. In *ACM Sigplan Notices*, Vol. 44. 431–440.

Thomas RW Scogland, Wu-chun Feng, Barry Rountree, and Bronis R de Supinski. 2014. CoreTSAR: Adaptive Worksharing for Heterogeneous Systems. In *Supercomputing*. 172–186.

Thomas RW Scogland, Barry Rountree, Wu-chun Feng, and Bronis R de Supinski. 2012. Heterogeneous task scheduling for accelerated OpenMP. In *IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*. 144–155.

Jie Shen, Ana Lucia Varbanescu, Henk Sips, Michael Arntzen, and Dick G Simons. 2013. Glinda: a framework for accelerating imbalanced applications on heterogeneous platforms. In *ACM International Conference on Computing Frontiers*. 14.

Wenfeng Shen, Daming Wei, Weimin Xu, Xin Zhu, and Shizhong Yuan. 2010. Parallelized computation for computer simulation of electrocardiograms using personal computers with multi-core CPU and general-purpose GPU. *Computer methods and programs in biomedicine* 100, 1 (2010), 87–96.

Takashi Shimokawabe, Takayuki Aoki, Tomohiro Takaki, Akinori Yamanaka, Akira Nukada, Toshio Endo, Naoya Maruyama, and Satoshi Matsuoka. 2011. Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer. In *SC*. 1–11.

Koichi Shirahata, Hitoshi Sato, and Satoshi Matsuoka. 2010. Hybrid map task scheduling for GPU-based heterogeneous clusters. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*. 733–740.

Sambit K Shukla and Laxmi N Bhuyan. 2013. A hybrid shared memory heterogeneous execution platform for PCIe-based GPGPUs. In *Int. Conf. on High Performance Computing*. 343–352.

Jakob Siegel, Oreste Villa, Sriram Krishnamoorthy, Antonino Tumeo, and Xiaoming Li. 2010. Efficient sparse matrix-matrix multiplication on heterogeneous high performance systems. In *Int. Conf. on Cluster Computing Workshops*. 1–8.

Mark Silberstein and Naoya Maruyama. 2011. An exact algorithm for energy-efficient acceleration of task trees on CPU/GPU architectures. In *International Conference on Systems and Storage*.

Jaideeep Singh and Ipseeta Aruni. 2011. Accelerating Smith-Waterman on Heterogeneous CPU-GPU Systems. In *Int. Conf. on Bioinformatics and Biomedical Engineering (iCBBE)*. 1–4.

HK-H So, Junying Chen, Billy YS Yiu, and Alfred CH Yu. 2011. Medical ultrasound imaging: To GPU or not to GPU? *IEEE Micro* 31, 5 (2011), 54–65.

F Sottilea, C Roedla, V Slavnicb, P Jovanovicb, D Stankovicb, P Kestenerc, and F Houssenc. 2013. GPU Implementation of the DP code. *Partnership for Advanced Computing in Europe* (2013).

Kyle Spafford, Jeremy Meredith, and Jeffrey Vetter. 2010. Maestro: data orchestration and tuning for OpenCL devices. In *Euro-Par 2010-Parallel Processing*. Springer, 275–286.

Kyle L Spafford, Jeremy S Meredith, Seyong Lee, Dong Li, Philip C Roth, and Jeffrey S Vetter. 2012. The tradeoffs of fused memory hierarchies in heterogeneous computing architectures. In *9th conference on Computing Frontiers*. 103–112.

Tomasz P Stefanski. 2013. Implementation of FDTD-Compatible Green's Function on Heterogeneous CPU-GPU Parallel Processing System. *Progress In Electromagnetics Research* 135 (2013), 297–316.

John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 3 (2010).

Przemysław Stpiczynski. 2011. Solving linear recurrences on hybrid GPU accelerated manycore systems. In *Federated Conference on Computer Science and Information Systems*. 465–470.

Przemysław Stpiczynski and Joanna Potiopa. 2010. Solving a Kind of BVP for ODEs on heterogeneous CPU+ CUDA-enabled GPU Systems. In *International Multiconference on Computer Science and Information Technology (IMCSIT)*. IEEE, 349–353.

John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and WMW Hwu. 2012. *Parboil: A revised benchmark suite for scientific and commercial throughput computing*. Technical Report. Center for Reliable and High-Performance Computing.

Yu Su, Ding Ye, and Jingling Xue. 2013. Accelerating inclusion-based pointer analysis on heterogeneous CPU-GPU systems. In *Int. Conf. on High Performance Computing*. 149–158.

Enqiang Sun, Dana Schaa, Richard Bagley, Norman Rubin, and David Kaeli. 2012. Enabling task-level scheduling on heterogeneous platforms. In *5th Annual Workshop on General Purpose Processing with Graphics Processing Units*. ACM, 84–93.

Hiroyuki Takizawa, Katsuto Sato, and Hiroaki Kobayashi. 2008. SPRAT: Runtime processor selection for energy-aware computing. In *Int. Conf. on Cluster Computing*. 386–393.

Yu Shyang Tan, Bu-Sung Lee, Bingsheng He, and Roy H Campbell. 2012. A map-reduce based framework for heterogeneous processing element cluster environments. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 57–64.

George Teodoro, Tahsin M Kurc, Tony Pan, Lee AD Cooper, Jun Kong, Patrick Widener, and Joel H Saltz. 2012. Accelerating Large Scale Image Analyses on Parallel, CPU-GPU Equipped Systems. In *Int. Parallel & Distributed Processing Symposium*. 1093–1104.

George Teodoro, Tony Pan, Tahsin M Kurc, Jun Kong, Lee AD Cooper, and Joel H Saltz. 2013. Efficient Irregular Wavefront Propagation Algorithms on Hybrid CPU-GPU Machines. *Parallel Comput.* (2013).

George Teodoro, Rafael Sachetto, Olcay Sertel, Metin N Gurcan, W Meira, Umit Catalyurek, and Renato Ferreira. 2009. Coordinating the use of GPU and CPU for improving performance of compute intensive applications. In *Int. Conf. on Cluster Computing and Workshops*. 1–10.

Pablo Toharia, Oscar D Robles, Ricardo SuáRez, Jose Luis Bosque, and Luis Pastor. 2012. Shot boundary detection using Zernike moments in multi-GPU multi-CPU architectures. *J. Parallel and Distrib. Comput.* 72, 9 (2012), 1127–1133.

Stanimire Tomov, Rajib Nath, and Jack Dongarra. 2010. Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing. *Parallel Comput.* 36, 12 (2010), 645–654.

Top500. 2014. Top500 Supercomputers. www.top500.org. (2014).

Kuen Hung Tsoi and Wayne Luk. 2010. Axel: a heterogeneous cluster with FPGAs and GPUs. In *International Symposium on Field programmable gate arrays*. 115–124.

Fernando Tsuda and Ricardo Nakamura. 2011. A technique for collision detection and 3D interaction based on parallel GPU and CPU processing. In *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*. 36–42.

Abhishek Udupa, R Govindarajan, and Matthew J Thazhuthaveetil. 2009. Synergistic execution of stream programs on multicores with accelerators. *ACM Sigplan Notices* 44, 7 (2009), 99–108.

Yash Ukidave, Amir Kavyan Ziabari, Perhaad Mistry, Gunar Schirner, and David Kaeli. 2013. Quantifying the Energy Efficiency of FFT on Heterogeneous Platforms. *ISPASS* (2013).

Ronald Veldema, Thorsten Blass, and Michael Philippsen. 2011. Enabling multiple accelerator acceleration for Java/OpenMP. In *Workshop on Hot Topics in Parallelism (HotPar)*, Vol. 11.

Sundaresan Venkatasubramanian and Richard W Vuduc. 2009. Tuned and wildly asynchronous stencil kernels for hybrid CPU/GPU systems. In *ICS*. 244–255.

Uri Verner, Assaf Schuster, and Mark Silberstein. 2011. Processing data streams with hard real-time constraints on heterogeneous systems. In *ICS*. 120–129.

Jeffrey S Vetter and Sparsh Mittal. 2015. Opportunities for Nonvolatile Memory Systems in Extreme-Scale High Performance Computing. *Computing in Science and Engineering* (2015).

Christof Vömel, Stanimire Tomov, and Jack Dongarra. 2012. Divide and Conquer on Hybrid GPU-Accelerated Multicore Systems. *SIAM Journal on Scientific Computing* 34, 2 (2012), C70–C82.

Richard Vuduc, Aparna Chandramowlishwaran, Jee Choi, Murat Guney, and Aashay Shringarpure. 2010. On the limits of GPU acceleration. In *Workshop on Hot Topics in Parallelism*.

Guibin Wang and Wei Song. 2011. Communication-Aware Task Partition and Voltage Scaling for Energy Minimization on Heterogeneous Parallel Systems. In *12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. 327–333.

Yueqing Wang, Yong Dou, Song Guo, Yuanwu Lei, and Dan Zou. 2013a. CPU–GPU hybrid parallel strategy for cosmological simulations. *Concurrency and Computation: Practice and Experience* (2013).

Yu Wang, Haixiao Du, Mingrui Xia, Ling Ren, Mo Xu, Teng Xie, Gaolang Gong, Ningyi Xu, Huazhong Yang, and Yong He. 2013b. A Hybrid CPU-GPU Accelerated Framework for Fast Mapping of High-Resolution Human Brain Connectome. *PloS one* 8, 5 (2013), e62789.

Zhenning Wang, Long Zheng, Quan Chen, and Minyi Guo. 2013c. CAP: co-scheduling based on asymptotic profiling in CPU+ GPU hybrid systems. In *International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM, 107–114.

Mei Wen, Huayou Su, Wenjie Wei, Nan Wu, Xing Cai, and Chunyuan Zhang. 2012. Using 1000+ GPUs and 10000+ CPUs for Sedimentary Basin Simulations. In *IEEE International Conference on Cluster Computing (CLUSTER)*. 27–35.

Dieter Wendel, Ronald Kalla, Joshua Friedrich, James Kahle, Jens Leenstra, Cedric Lichtenau, Balaram Sinharoy, William Starke, and Victor Zyuban. 2010. The POWER7 processor SoC. In *International Conference on IC Design and Technology (ICICDT)*. 71–73.

Qiang Wu, Canqun Yang, Feng Wang, and Jingling Xue. 2012. A fast parallel implementation of molecular dynamics with the Morse potential on a heterogeneous petascale supercomputer. In *Int. Parallel and Distributed Processing Symposium Workshops & PhD Forum*. 140–149.

Tin-Yu Wu, Wei-Tsong Lee, Chien-Yu Duan, and Tain-Wen Suen. 2013. Enhancing Cloud-Based Servers by GPU/CPU Virtualization Management. In *Advances in Intelligent Systems and Applications-Volume 2*. Springer, 185–194.

Shucai Xiao, Heshan Lin, and Wu-chun Feng. 2011. Accelerating Protein Sequence Search in a Heterogeneous Computing System. In *Int. Parallel & Distributed Processing Symposium*. 1212–1222.

Ming Xu, Feiguo Chen, Xinhua Liu, Wei Ge, and Jinghai Li. 2012. Discrete particle simulation of gas-solid two-phase flows with multi-scale CPU-GPU hybrid computation. *Chemical Engineering Journal* (2012).

Canqun Yang, Feng Wang, Yunfei Du, Juan Chen, Jie Liu, Huizhan Yi, and Kai Lu. 2010. Adaptive optimization for petascale heterogeneous CPU/GPU computing. In *IEEE International Conference on Cluster Computing (CLUSTER)*. 19–28.

Chao Yang, Wei Xue, Haohuan Fu, Lin Gan, Linfeng Li, Yangtong Xu, Yutong Lu, Jiachang Sun, Guangwen Yang, and Weimin Zheng. 2013. A peta-scalable CPU-GPU algorithm for global atmospheric simulations. In *Symp. on Principles and Practice of Parallel Programming*. 1–12.

Ping Yao, Hong An, Mu Xu, Gu Liu, Xiaoqiang Li, Yaobin Wang, and Wenting Han. 2010. CuHMMer: A load-balanced CPU-GPU cooperative bioinformatics application. In *International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, 24–30.

Marcelo Yuffe, Ernest Knoll, Moty Mehalel, Joseph Shor, and Tsvika Kurts. 2011. A fully integrated multi-CPU, GPU and memory controller 32nm processor. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 264–266.

Ziming Zhong, Vladimir Rychkov, and Alexey Lastovetsky. 2012. Data Partitioning on Heterogeneous Multicore and Multi-GPU Systems Using Functional Performance Models of Data-Parallel Applications. In *Int. Conf. on Cluster Computing*. 191–199.

V Zyuban, SA Taylor, B Christensen, AR Hall, CJ Gonzalez, J Friedrich, F Clougherty, J Tetzloff, and R Rao. 2013. IBM POWER7+ design for higher frequency at fixed power. *IBM Journal of Research and Development* 57, 6 (2013), 1–1.