Architecturally Truly Diverse Systems: A Review

Roger D. Chamberlain

Washington University in St. Louis, St. Louis, Missouri, USA

Abstract

The pairing of traditional multicore processors with accelerators of various forms (e.g., graphics engines, reconfigurable logic) can be referred to generally as *architecturally diverse* systems. Our interest in this work is *truly* diverse systems, in which more than one accelerator is used in the execution of an application. These systems have the potential for substantial performance gains relative to multicores alone; however, they pose significant difficulties when it comes to application development.

In spite of these difficulties, the use of accelerators in high performance computation, generally, has grown substantially over the past decade. This is primarily due to a pair of forces. First, with the demise of Dennard scaling, power has become a substantial limiting factor in systems development, pushing computations to be more power efficient (a strength of many accelerators). Second, the application development environments for accelerators have improved substantially in recent years.

We review the use of multiple, distinct accelerators deployed in a individual system or, more to the point, used concurrently within an individual application. We give a history of architecturally diverse systems that use multiple accelerators, discuss the motivations for diversity in accelerators, and describe the approaches that both system designers and application developers have used to put accelerators to beneficial use.

Keywords: graphics engine, GPU, reconfigurable logic, FPGA

1. Introduction

The exploitation of Moore's Law [1, 2] and Dennard Scaling [3, 4] has enabled an unprecedented increase in computational capability over ap-

Email address: roger@wustl.edu (Roger D. Chamberlain)

proximately five decades. Smaller feature sizes in semiconductor technology, combined with greater power efficiency associated with each transistor, have resulted in a long stretch of time in which application performance greatly improved simply by waiting for the next generation of processors to be shipped.

No longer. Since the early 2000s, the clock rates of individual processors have stayed relatively constant, and improved performance has primarily been due to a combination of speculative computation and parallelism. Speculative computation has been used to increase the performance of individual processor cores, and the number of cores on a processor chip has been expanding for over a decade.

In addition to the transition of traditional processors into multicores, the past two decades have also seen the advent of accelerators, or *architecturally diverse* systems. Here, a multicore processor is augmented with one or more specialized computational resources. While most commonly graphics engines or reconfigurable logic, other accelerators that have seen use include digital signal processors and vector processors (e.g., Intel's Xeon Phi).

Accelerators typically gain performance by exchanging generality for specificity. While general purpose processor cores try to be very good at everything, accelerators forego good performance on some computational tasks so that they can excel on a more limited set of target tasks. They accomplish this goal by specializing the computational data path, each in their own specific way.

Graphics engines (or GPUs, graphics processing units) are now ubiquitous in the Top500 supercomputer lists.[1] Graphics engines trade off single thread performance (i.e., minimum latency to execute a sequential set of instructions) for greater computational throughput on multiple threads. A single fetch-decode unit controls a collection of execution units, each of which performs the same instruction on different data elements. This enables the graphics engine to excel at regular dense matrix computations (e.g., vector-matrix multiplication) and other compute problems for which the same operation is to be performed on a large set of data.

In 2015, Mittal and Vetter [5] surveyed techniques for effective computation using a combination of multicores and graphics engines. They consider systems in which the graphics engines are on a peripheral bus (typically PCIe in modern systems) as well as fused systems (both compute resources on a common chip).

[1] https://www.top500.org

Reconfigurable logic (or FPGAs, field-programmable logic arrays) has been around for a longer time than graphics engines, but it has only recently come to be seen as a viable option for high performance computation. The unique advantage of reconfigurable logic is that the actual structure of the data path can be altered to match the needs of the specific computation at hand.

Compton and Hauck [6] described the state of reconfigurable logic targeting high performance computations back when the largest chips were capable of about one million equivalent logic gates. Since then, capacities have increased 100-fold. In addition, it is now common for reconfigurable logic chips to also have some chip area dedicated to specific functions (e.g., memory, multipliers, etc.).

More recently, Trimberger [7] provides a retrospective on the first 30 years of reconfigurable logic technology, breaking it down into three "ages," which he calls *invention*, *expansion*, and *accumulation*. Escobar et al. [8] provide a comprehensive analysis of the use of reconfigurable logic in high performance computing. They extract features from families of high performance computing applications and assess their suitability for hardware implementation. Microsoft is now installing reconfigurable logic in its Azure cloud systems [9].

In this paper, we are interested in exploring *truly architecturally diverse* systems, by which we mean systems that include not just one, but two (or potentially more) distinct accelerators. Most of these systems contain both graphics engines and reconfigurable logic, in addition to the traditional multicore that forms the base of the system.

Here, we review the use of multiple accelerators used to execute an individual application. We describe early efforts in this area and then concentrate on how system designers and application developers have utilized multiple accelerators. Moving forward, what properties (of both the application and the execution platform) are important in matching applications to platforms, and how can we perform the appropriate deployments with little or no explicit developer guidance.

The paper is organized as follows. Section 2 describes the early history of multiple accelerator use (roughly defined as 2010 or before). Section 3 follows with a description of a number of machines that support multiple accelerators, both early machines and more recent machines. Section 4 lists a number of applications that have used multiple accelerators in their execution. Section 5 then describes development environments that target multiple accelerator use, and Section 6 concludes and describes future work.

2. Early History

Hardware acceleration of applications has a long history. In 1984, Blank [10] published a survey of hardware accelerators used in computer-aided design that described a set of special-purpose engines going back nearly 20 years (the first being described in 1969 [11]). Blank describes accelerators for logic simulation, design rule checking, placement, and routing. These are all tasks that remain computationally expensive today, especially as designs of digital systems have grown in size and complexity.

These accelerators were all dedicated hardware engines, capable only of executing the specific application for which they were originally designed and built. As a result, none of them were economically viable over any length of time. During this time period, the march of technology (specifically the combination of Moore's Law and Dennard Scaling) quickly overcame any functionality-limited design. It simply wasn't long before general-purpose machines became fast enough to eliminate the performance advantage that was available via hardware specialization.

While the example given above is computer-aided design, there were similar efforts in other domains (e.g., N-body simulations in astrophysics [12, 13], LISP machines [14, 15], Java machines [16, 17]), all of which suffered a similar fate. A recent counterexample to this trend is the advent of customized architectures applied to convolutional neural network computations, e.g., the Tensor Processing Unit [18].

2.1. Comparing Different Accelerators

While dedicated purpose hardware acceleration has had limited success, accelerators that can be repurposed to multiple applications have seen substantially more utilization. There has been a significant amount of work dedicated to effectively using individual accelerators on a wide range of problems. Excellent reviews include those by Mittal and Vetter [5] for graphics engines, Escobar et al. [8] for reconfigurable logic, and Brodtkorb et al. [19] for three accelerators: graphics engines, reconfigurable logic, and Cell processors. Chung et al. [20] provided a retrospective of the field in 2010.

Approximately a decade ago, there was a fair amount of research that went into comparing one accelerator with another. The bulk of that work compared graphics engines to reconfigurable logic on a wide range of applications and/or application kernels. Examples include dense matrix multiplication [20, 21], sparse matrix-vector multiplication [22], Gaussian elimination [23], FFT [20], reduction [21], encryption [23], image processing [24, 25], lithography simulation [26], pseudo-random number generation [21, 27, 28],

N-body simulation [21], computational biology [23], and computational finance [20, 29].

Kapre and DeHon [30] compared the performance of graphics engines, reconfigurable logic, and Cell processors on a circuit simulation application, and Baker et al. [31] did the same exploring matched filter computations. Application performance compared across graphics engines and Cell processors includes CT reconstruction [32] and partial differential equation simulations [33].

Table 1 summarizes the majority of the reported results from these studies. The clear takeaway from this table is that there is no clear winner for all of these applications which accelerator gives the greater performance. It clearly depends on the properties of the application and the implementation.

Table 1: Performance comparisons for different accelerators on a variety of applications, all reported between 2007 and 2010.

Application	Ref.	Graphics Engine	Reconfig. Logic	Notes
matrix multiply	[20]	541	1,491	GFLOP/s
	[21]	120×	60×	speedup vs. 1 core
reduction	[21]	120×	80×	speedup vs. 1 core
image processing	[24]	100	600	frames/s
	[25]	847	258	frames/s
lithography sim.	[26]	8×	15×	speedup vs. 1 core
pseudo-random number generation	[21]	90×	89×	speedup vs. 1 core
	[27]	3×10^9	26×10^9	random numbers/s
	[28]	14×10^9	44×10^9	random numbers/s
N-body sim.	[21]	70×	5×	speedup vs. 1 core
comp. finance	[20]	10,756	7,800	Mopts/s
	[29]	50×	162×	speedup vs. 1 core
circuit sim.	[30]	131×	182×	speedup vs. 1 core
matched filter	[31]	6×	2×	speedup vs. 1 core

While which accelerator will win a head-to-head performance competition varies with application and implementation, a number of clear trends are evident from the work described in this section. In particular, the three types of compute engine (multicores, graphics engines, and reconfigurable logic) each have their own strengths and weaknesses relative to the other two. These can be briefly stated as follows:

- **Multicores** – Strengths: generality, application portability, ease of

programmability. Weaknesses: lowest parallelism of the group, rigid memory subsystem, low power efficiency.

- **Graphics engines** – Strengths: high parallelism (esp. for data parallel problems), high density of floating-point units, high throughput memory (including coalescing). Weaknesses: rigid computational data path, limited suitability for a variety of applications.

- **Reconfigurable logic** – Strengths: extremely high flexibility, suitability for fine-grain parallelism, power efficiency. Weaknesses: difficulty of programmability, lower clock rates (necessitating greater parallelism to achieve similar performance).

The end result, not surprisingly, is that a particular accelerator sees performance benefits particularly when the needs of the application are a match with the capabilities of the accelerator. E.g., high-volume floating-point multiplications on regular data structures are well suited to a graphics engine, while specialized bit-level manipulations are well suited to reconfigurable logic, in which the data path is customized the to specific needs of the application.

While the above conclusions were based upon investigations completed by 2010, they are still as true today as they were then.

2.2. First Use of Multiple Accelerators on a Common Problem

The first known (to this author) combination of a graphics engine and reconfigurable logic used on a common problem was presented in two papers by Kelmelis et al. [34, 35] in 2006. They used a pair of PCI cards (an EM Photonics Celerity card with a Xilinx Virtex-II FPGA and an NVIDIA GeForce 7800 GTX graphics card) to accelerate the execution of the finite-difference time-domain (FDTD) simulation of electromagnetic waves [34] and nanoscale devices [35]. In the electromagnetic wave simulation, the 2D problem was solved using the graphics engine, exploiting its greater memory bandwidth, while the 3D problem was solved using the reconfigurable logic, building a custom cache. As such, this problem didn't yet meet our interest in using both accelerators on a common problem.

In [35], the authors describe what is believed to be the first ever use of multicores, graphics engines, and reconfigurable logic to execute a single application. In the problem partitioning, the graphics engine is responsible for the mesh construction over the 3D space, which is a regular computation that maps well onto the graphics engine's data parallel pipelines. The reconfigurable logic is responsible for performing the field-update computations,

which benefits from the custom cache designed to provide input data to the field updates.

This combined machine was then used to execute a full-wave electromagnetic circuit simulation in which the graphics engine was used for coefficient calculations and matrix setup while the reconfigurable logic was responsible for the iterative matrix solver.

2.3. Other Early Uses of Multiple Accelerators

A second example of simultaneous use of multiple accelerators was reported by Yeung et al. [36] in 2008. The authors propose the use of the map-reduce framework for heterogeneous systems, show the performance of a number of benchmarks (e.g., Monte Carlo simulation, European options pricing, cipher attack, N-body simulation) using each accelerator independently, and then show performance with combined use of both the graphics engine and the reconfigurable logic.

The third known example of using multiple accelerators on a common problem is the deployment of a financial Monte Carlo simulation (a value-at-risk computation) on graphics engines and reconfigurable logic by Singla et al. [37], also described in 2008. The authors report a speedup of 74× relative to an 8-core implementation, using an NVIDIA GeForce GTX 260 as the graphics processor and a Xilinx Virtex-4 LX80 FPGA for the reconfigurable logic. The processors are 2.2 GHz AMD Opterons.

In 2010, there were three publications that describe such deployments. First, Tsoi and Luk [38] deployed an N-body simulation application on the Axel cluster, a heterogeneous system in which each cluster node (out of 16 total) includes both reconfigurable logic and a graphics engine. The map-reduce framework is used to deploy applications on the Axel system.

Second, Bauer et al. [39] deployed a video-based pedestrian detection problem on a PC system that includes a graphics card and in which the frame grabber board has an embedded reconfigurable logic chip. The reconfigurable logic is responsible for feature extraction and the graphics engine is responsible for classification (using a Gaussian kernel support vector machine).

Third, Tse et al. [40] deployed a pair of Monte Carlo simulation applications on a cluster with 8 Virtex-5 FPGAs and 8 Tesla C1060 GPUs. Their asset simulation application achieved a speedup of 44× relative to a 16-node cluster of AMD quad-core 2.4 GHz multicores. Energy efficiency improved almost 20-fold. A first for this paper was the investigation of various scheduling policies for assignment of work to the various accelerators.

2.4. Early Machines and Development Environments

In addition to the Axel cluster [38] mentioned above, another early machine that included both graphics engines and reconfigurable logic is the Quadro Plex (QP) [41] cluster at NCSA. The QP, deployed in 2007, was a 16 node cluster in which each compute node contained two dual-core 2.4 GHz AMD Opterons, four NVIDIA Quadro FX 5600 graphics engines and one Nallatech H101-PCIX reconfigurable logic accelerator card. While [41] describes several applications deployed on the cluster, they all appear to use one accelerator or the other, but not both.

Kastl and Loimayr [42] described a machine designed to accelerate cryptanalytic algorithms. However, in the initial publication the reconfigurable logic was not operational and any application testing was limited to graphics engine acceleration.

Early development environments include Auto-Pipe [43, 44] and Harmony [45]. Auto-Pipe was originally introduced in 2006 as an environment for streaming applications supporting both traditional multicores and reconfigurable logic [43], and was later extended to support graphics engines [44]. This was the environment used by Singla et al. [37].

The Harmony environment supports applications that have been decomposed into *computer kernels* whose execution can be subject to explicit *control decisions* [45]. As such, it has much in common with Auto-Pipe's streaming model, with a slightly richer set of topologies supported. While the design of Harmony includes reconfigurable logic support, the experimental evaluation provided in [45] is limited to multicores and graphics engines.

The work described in this section was all published by 2010. In the sections that follow, we will concentrate on research that has been published since then.

3. Machines

The ability to physically construct a machine with multiple accelerators has never really been in question. Frankly, they are straightforward to construct using readily available commercial components. Figure 1 illustrates the classic approach to such a design. Clusters of nodes of this style can be readily constructed using either Ethernet or Infiniband as the communication technology.

In the figure, a multicore host is attached to a reconfigurable logic board and to a graphics engine via the host's I/O bus. Current systems use PCIe as the I/O bus, however, most of the early systems described in the previous section used PCI-X. Each of the elements of the machine (the multicore, the

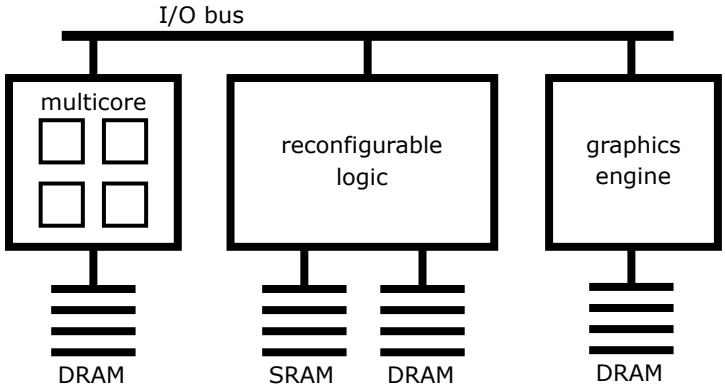

Figure 1: Individual node architecture for an multiple-accelerator system.

reconfigurable logic, and the graphics engine) has its own physical memory. It is common for reconfigurable logic boards to have both SRAM and DRAM on board. Since there is not, by default, a common address space that encompasses all of these physical memories, data communications between execution engines can be a significant issue both in application development and performance. We will come back to this point later.

After the initial set of machines described in Section 2.4 above, a number of additional machines have been designed, built, and deployed that support multiple accelerators.

In 2012, the Chimera system was described [46]. It combined Intel multicores, NVIDIA Tesla C2070 graphics engines, and an Altera (now Intel) Stratix-IV reconfigurable logic card using the motherboard's PCIe bus. The authors describe approaches to deploying a variety of applications on the system. However, performance results are not provided.

A multiple accelerator cluster built at the Center for Development of Advanced Computing (C-DAC) in Bangalore, India, was described in 2014 [47]. Here, the authors adapted the StarPU development environment [48] to support reconfigurable logic as well as simultaneous support of CUDA and OpenCL in the same application. A Monte Carlo simulation application is described, achieving a 22× speedup relative to a Xeon processor.

Also in 2014, Wu et al. [49] describe a heterogeneous compute platform with multicores, graphics engines, and reconfigurable logic that has a focus on power efficient computing. The authors explore four application examples, but similar to the QP machine, chose to exploit one accelerator type or another, but not both simultaneously.

Proaño et al. [50] describe an open-source framework for integrating ac-

celerators into a cloud infrastructure. Their goal is to enable both graphics engines and reconfigurable logic as compute resources in the Infrastructure as a Service (IaaS) cloud service model. With AWS having already announced the availability of graphics engines in the cloud, the authors focused on exploring the issues associated with enabling the reconfigurable logic. More recently, AWS has announced the availability of FPGA-enabled nodes in their cloud infrastructure.

In 2015, Rethinagiri et al. [51] described a pair of computational platforms that included both graphics engines and reconfigurable logic. One was targeting high-performance computing (HPC) applications and the other was targeting high-performance embedded (HPE) applications. In both cases, their interest was a combination of performance and energy efficiency. Table 2 gives the particulars of the two platforms (extracted from Table I of [51]).

Table 2: Specifications for HPC and HPE platforms [51].

Domain	Engine	Specification	Communication
HPC	cores	Intel Xeon E5-2634	Gen3 ×16
	GPU	NVIDIA Telsa K40	Gen3 ×16
	FPGA	Xilinx Virtex-7 VC709	Gen3 ×8
HPE	cores and GPU	NVIDIA Jetson TK1	Gen2 ×1
	cores and FPGA	Xilinx Zynq-7015	Gen2 ×4

The authors give performance results and energy efficiency for five applications across the pair of machines: computed tomography, face recognition, video coding, character recognition, and motion tracking.

The SuperDragon system [52] contains 64 nodes, each of which is comprised of an Intel Westmere processor, an NVIDIA Tesla C2050 graphics engine, and a pair of Xilinx FPGAs (an XC5VLX330-1 for computation and an XC5VLX70T for control and communications management), connected by PCIe. The 64 nodes communicate via an Infiniband network. The authors use a Cryo-electron microscopy application to explore various mapping options for assigning work to the different accelerators.

Segal and Margala [53] describe a system constructed using commodity cloud nodes (from AWS) and a locally deployed node containing a reconfigurable logic card. They used the SparkCL framework [54], which adds accelerator support to Apache Spark, to deploy an N-body simulation. They conclude that which accelerator gives the best performance depends in part on the dynamics of the particle interactions in the N-body simulation.

This latter point is worth reinforcing. Not only does the application algorithm impact what type of accelerator gives the best performance, for many problems the nature of the input data can also influence this result. In many cases, it might very well be impractical to know, for certain, which accelerator is to be preferred prior to the actual execution. This leaves the door open for adaptive approaches to be effective.

Finally, in 2016 Contassot-Vivier and Vialle [55] described a pair of machines that contain both an NVIDIA graphics engine and also a Xeon Phi. This is the first known combination of these two accelerators in a single machine. The authors give performance data for a Jacobi relaxation application on each of these two machines. They report that effective management of communications (between host and accelerators) is essential to achieving good performance on this type of machine.

4. Applications

The vast majority of the applications that have been deployed on more than one accelerator have combined graphics engines and reconfigurable logic as the two types of accelerator. Tables 3 and 4 summarize the literature for these applications, indicating the application itself, the year of publication (including the citation), and various notes about the specific implementation. Particularly, for each application it is indicated whether (or not) the application assigns distinct functions to the two different accelerator types, and also whether (or not) the map-reduce programming model is used as part of the implementation.

A large fraction of the applications listed partition the workload across distinct functions. Out of 41 total applications, 29 (or about 70%) divide the application in this way. These decisions on the part of the application developers reflect the reality that the different accelerators have distinct strengths and weaknesses, and different portions of the application can differently take advantage of those strengths and/or ameliorate the weaknesses.

An example of partitioning the workload based on the distinct functions to be performed is described by Danczul et al. [56]. The authors are attempting a brute force attack on the password for a set of PDF encrypted files. This requires both MD5 hash computations as well as RC4 stream cypher computations. They assign the MD5 hashes to the graphics engines, in part due to its limited memory requirements, and assign the RC4 computations to the reconfigurable logic (in which the memory requirements of the RC4 algorithm can be accommodated efficiently).

Table 3: Applications deployed on multiple accelerators.

Application	Ref.	Year	Notes
EM sim	[35]	2006	First known example*
Monte Carlo simulation	[36]	2008	European options pricing†
	[37]	2008	Financial value-at-risk computation*
	[40]	2010	Asian options and generalized asset pricing
	[47]	2014	Computation of π*
N-body simulation	[36]	2008	Dynamic workload distribution†
	[38]	2010	Introduced Axel cluster†
	[53]	2016	AWS GPUs plus local FPGA
crypto	[36]	2008	Brute force key search†
	[56]	2013	Password search for PDF documents*
machine learning	[39]	2010	Pedestrian detection*
	[57]	2018	Character recognition*
vision	[58]	2011	Stereo 3D video production*
	[59]	2014	Hand tracking*
	[51]	2015	Video coding and motion tracking*
medical imaging	[60]	2012	Includes both static and dynamic assignment*
	[61]	2012	Cardiac mapping
	[62]	2013	Transmural electrophysiological imaging*
	[51]	2015	Cone beam computed tomography*
	[52]	2015	Cryo-electron microscopy
	[63]	2018	Ultrasonic imaging*
signal and image processing	[64]	2013	Pedestrian recognition*
	[51]	2015	Face recognition and character recognition*
	[65]	2015	Real-time RF localization
	[66]	2016	Data acquisition for experimental physics*
	[67]	2018	Add dynamic function distribution to above*
	[68]	2018	Increase frame rate to 9000 frames/s*
astronomy	[69]	2014	Radio antenna cross-correlation*
comp bio	[70]	2014	Protein-protein sequence alignment*
	[71]	2016	Genome-wide association studies*
	[72]	2017	Genome-wide interaction studies*
	[73]	2019	Computation of gene-gene interactions*

*Assigns distinct functions to the two accelerators. †Utilizes map-reduce.

Table 4: Applications deployed on multiple accelerators (cont.).

Application	Ref.	Year	Notes
data acquisition	[74]	2016	Linear accelerator*
	[75]	2017	Trigger systems for Large Hadron Collider*
	[76]	2018	3D waveform oscilloscope
	[77]	2019	Linear accelerator*
driving	[78]	2017	Driver assistance (lane detection)
	[79]	2018	Autonomous vehicle computations*

*Assigns distinct functions to the two accelerators. †Utilizes map-reduce.

In a large fraction of the implementations where the workload is divided by function onto each accelerator, the computation is organized as a pipeline. Data flows into one accelerator, the portion of the work assigned to that accelerator is performed, the output of that pipeline stage then flows into the other accelerator, where the portion of the work assigned to that accelerator is performed.

Sbîrlea et al. [60] extend this to general dataflow graphs, supporting a richer set of topologies than just a simple pipeline. The authors also incorporate the ability to support a work stealing scheduler across execution platforms.

A clear benefit of the map-reduce paradigm is that it is relatively straightforward to express the computational parallelism available in an algorithm using this approach. Several applications took advantage of this (4 of 41, or about 10% of those listed). In addition, it can also support the dynamic assignment of workload across the available computational resources, illustrated as early as 2008 by Yeung et al. [36]. This approach, however, has not been used for recent deployments, likely due to its inability to effectively exploit the particular advantages of each accelerator. If the workload is uniform, it stands to reason that a uniform architecture is the preferred deployment option.

For the set of Monte Carlo simulations listed in the table, four of the five problems attempted ([40] includes two) are some form of financial computation. This hints at the strong interest in accelerated computation on the part of the financial industry.

Any time multiple compute resources are used in an application, there is the potential for data movement between those resources to be a performance bottleneck. Bittner et al. [59] demonstrate direct graphics engine to reconfigurable logic DMA data transfers over PCIe, avoiding a copy to/from

the host memory. Ammendola et al. [80] also exploit a combination of FPGAs and graphics engines; however, the FPGAs are not used for application functionality but instead are used to implement GPU to GPU communications.

In the past decade, there has been a continuation of the research that seeks to compare one accelerator type with another. Examples of this that compare reconfigurable logic with graphics engines include: scientific computations [81], compressed sensing and Cholesky decomposition [82], sliding-window computations [83], de novo assembly (in computational biology) [84], erasure coding [85], database join [86], and data integration [87]. The quantitative performance comparisons for these experiments are shown in Table 5. Carabaño et al. [88] compare energy efficiency, performance, and productivity across accelerators, and Véstias and Neto [89] explore accelerator trends, particularly as they impact both peak performance and sustained performance. O'Neal and Brisk [90] provide predictive models for each of the constituent computational engines: multicores, graphics engines, and reconfigurable logic, quantifying both performance and power consumption.

Table 5: Performance comparisons for graphics engines and reconfigurable logic, reported since 2010.

Application	Ref.	Graphics Engine	Reconfig. Logic	Notes
Quantum MC simulation	[81]	50×10^6	90×10^6	interactions/s
Cholesky decomposition	[82]	38×	15×	speedup vs. 1 core
image convolution	[83]	20	40	frames/s
genome alignment	[84]	13×	115×	speedup vs. 1 core
erasure coding	[85]	3.9	1.2	GB/s throughput
database join	[86]	127	70	μs for 8192 elements
bio data conversion	[87]	5.3	15.3	GB/s throughput
ML data conversion	[87]	6.2	1.8	GB/s throughput

A number of investigations have concluded that which accelerator is preferred often depends upon the properties of the input data. Examples of applications for which this is true include: linear algebra [91, 92], sparse matrix multiplication [93], and vision [94]. In general, the conclusions drawn in the previous decade (see Section 2.1) still hold for this work as well.

Graphics engines and reconfigurable logic are not the only accelerators

available, however. An investigation that included the Cell in the mix implemented a biological sequence alignment problem [95]. Work comparing the Cell with graphics engines (but not including reconfigurable logic) includes problems such as wavelet transforms [96], Bayesian analysis in bioinformatics [97], molecular dynamics [97], and the TPC decision support benchmark [97].

Research that compares graphics engines, reconfigurable logic and the Xeon Phi include an investigation of sparse matrix multiplication [93] as well as the development of predictive models for power and energy for all three execution platforms [98]. Dropping reconfigurable logic from the comparison yields the following application investigations: Ising model simulation [99], microscopy [100], quantum chemistry [101], quantum many-body simulations [102], and Jacobi relaxation [55]. Performance data from some of the above systems is presented in Table 6. As is the case for the previous comparisons, which accelerator has better performance is again a function of the properties of the application itself. Productivity, performance, and energy quantification is pursued by Memeti et al. [103].

Table 6: Performance comparisons for graphics engines and Xeon Phi.

Application	Ref.	Graphics Engine	Xeon Phi	Notes
sparse matrix multiply	[93]	60	59	Gflops/s
Ising model simulation	[99]	3.72	7.03	ns/spin for L^{128} lattice
Jacobi relaxation	[55]	4.96	3.02	speedup vs. 1 core

We next turn our focus away from individual applications, and focus instead on the development environments that enable these applications to be designed, implemented, deployed, tested, and executed.

5. Development Tools

The general responsibilities associated with application development tools can be quite broad. First, they might assist in the expression of the computation to be executed (either by supporting one or more languages or introducing a potentially new language). Second, they might assist the execution of the application, often through resource management and/or scheduling. We will consider each of these in turn.

5.1. Expression of the Application's Computation

Here we are interested in tools that help the application developer express the computation that is to be performed. Essentially, what must the programmer say to enable the execution platform to do what is desired. Traditionally, application expression using even a single accelerator has been somewhat difficult (e.g., requiring the use of low-level languages like Verilog and/or VHDL for reconfigurable logic design), and the addition of a second accelerator with potentially vastly different properties does nothing to make the task any easier.

Probably the easiest place to make progress on this front is the development of libraries. By encapsulating either a portion of the computation or some support function in a library, the application developer is relieved of the responsibility to implement that functionality.

As a first example, Thoma et al. [104] describe a framework for supporting accelerator to accelerator communications (specifically supporting data transfers between GPUs and FPGAs). In particular, PCIe transfers are made directly, device-to-device, without data being copied into the main memory of the multicore host.

For a second example, Moore et al. [105] describe an implementation of the VSIPL++ signal processing library that supports both graphics engines and reconfigurable logic. The same source code can be used on a multicore as well as exploit available accelerators.

As a third example, Zhu et al. [106] present CNNLab, a framework for neural network implementations that supports both graphics engines and reconfigurable logic. Each layer of the model is prescribed via an API.

For reconfigurable logic on the Zynq SoC, Xilinx supports the use of Python on the embedded processor core for coordination and library invocation (with the instantiation of the library functionality on the reconfigurable logic fabric).

Finally, Abalenkovs et al. [107] explore the performance of libraries for dense linear algebra deployed on multicores, graphics engines, and the Intel Xeon Phi.

As a next step in the direction of simplifying application development, several groups have used what are, in effect, coordination languages. Here, the individual portions of the overall computation that is to be performed on an accelerator is programmed using native tools for that accelerator. A coordination language is then used to stitch the whole application together.

Examples of this approach include the Auto-Pipe system, which originally supported just multicores and reconfigurable logic [43], but was later

expanded to include graphics engines [44]; the Axel cluster [38], which initially used the map-reduce framework for coordination, extended later to richer coordination capabilities [108]; and a data-flow coordination model called Concurrent Collections [60].

All of the approaches discussed so far still require some amount of the application to be developed in the low-level languages supported by the accelerators (a particularly problematic issue for reconfigurable logic). With the availability of OpenCL as a potential common language, Ahmed [109] built the necessary infrastructure to enable a single OpenCL application to exploit both graphics engines and reconfigurable logic for a common application. The Liquid Metal project [110] at IBM introduced a new language, Lime, a Java-compatible object-oriented language, that can be compiled for and executed on both graphics engines and reconfigurable logic. In these systems, the reconfigurable logic is expressed in a high-level language (either OpenCL or Lime, respectively) and a high-level synthesis process is used to convert the source code into the gate-level designs necessary for deployment on reconfigurable logic.

Most research into application development languages and systems for accelerators has focused on one accelerator. When the target is a graphics engine, by far the most common two languages are CUDA and OpenCL. CUDA, being proprietary, is limited to NVIDIA graphics engines, while OpenCL, as an open standard, can be used across a much wider set of manufacturers' platforms.

Both CUDA and OpenCL are primarily used to express data parallelism when targeting graphics engines, particularly regular computations on densely packed data structures. The MERCATOR system [111] attempts to widen the scope of applicability to irregular computations, such as manipulation of sparse data. While CUDA has been almost exclusively used for graphics engines, FCUDA [112] is a system that does a source-to-source translation of CUDA code to C (suitable for high-level synthesis), enabling CUDA applications to be deployed on reconfigurable logic.

OpenACC is a directive-based approach to application expression, used initially for graphics engines, that has been adapted to support FPGAs [113] via source-to-source translation into OpenCL.

While historically the only way to realize efficient and performant applications on reconfigurable logic was to use a hardware description language (such as Verilog or VHDL), there are now a number of options for high-level synthesis tools that enable application expression at a level of abstraction much closer to that used in graphics engines or traditional processor cores. OpenCL as a source language has already been mentioned above. Many oth-

ers are based on C/C++ or a subset thereof. Table 7 summarizes many of these tools, and Nane et al. [114] provide a broader survey of FPGA-specific tools. These high-level synthesis tools are now quite effective in practice, delivering reasonable performance with design effort comparable to graphics engine application development [115, 116, 117].

Table 7: Tools for application expression on accelerators.

Tool	Ref.	License	Input Language(s)	Deployment Target(s)
Auto-Pipe	[44]	academic	VHDL,CUDA	FPGA,GPU
Liquid Metal	[110]	commercial	Lime	FPGA,GPU
OpenACC	[113]	academic	C/C++	FPGA,GPU
NVIDIA CUDA	[118]	commercial	CUDA	GPU
MERCATOR	[111]	academic	CUDA	GPU
DWARV	[119]	academic	C	FPGA
FCUDA	[112]	academic	CUDA	FPGA
Intel HLS		commercial	C/C++/OpenCL	FPGA
LegUp	[120]	academic	C	FPGA
Maxeler	[121]	commercial	MaxJ	FPGA
ROCCC	[122]	academic	C	FPGA
Vivado HLS		commercial	C/C++/OpenCL	FPGA

While high-level synthesis can be extremely beneficial in terms of enabling application developers to author their codes at a higher level of abstraction than was previously available, they are not without continuing issues. Cabrera and Chamberlain [116] recently investigated the implications of adjusting design parameters in OpenCL implementations of the Needleman-Wunsch biosequence alignment algorithm targeting reconfigurable logic. They reported performance variability on an Intel FPGA of over 100× across different parameter configurations. Knowing how to tune the implementation is clearly crucial to achieving good performance.

Productivity, performance, and energy quantification is pursued by Memeti et al. [103] for languages that can be deployed to graphics engines and the Xeon Phi. Their work compared OpenCL, OpenACC, OpenMP, and CUDA. They developed a developer productivity measure that quantifies the fraction of code lines required for parallelization. Two things they conclude are: (1) human factors are quite important (a fact well known in software development circles), and (2) OpenMP generally requires less programmer effort that the other approaches.

The European EXTRA project has investigated the issue of design space exploration for reconfigurable systems [123, 124]. The focus of this project has been reconfigurability at a courser granularity than traditional FPGAs (e.g., CGRAs, or course grained reconfigurable arrays), although the experimental deployments include FPGAs. A unique feature of this project is the explicit consideration of the need for effective reconfigurability, i.e., what is deployed on the accelerator will vary either during the execution of the application or between applications. A wide number of applications have been implemented [125, 126, 127], as well as domain specific language approaches to application expression [128, 129].

The take-home message from all of this is that if one's goal is to develop applications that can be deployed across both graphics engines and reconfigurable logic, one of the OpenCL variants (from Intel or Xilinx) is the most appropriate choice. They both employ knowledge gained from the various academic efforts, and have matured into well-supported commercial tools.

5.2. Managing the Application's Execution

Given the ability to deploy various components of a problem onto distinct accelerators, Liu and Luk [130, 131] provided an early study of how one should allocate problem components onto execution resources that include graphics engines and reconfigurable logic. They explored three metrics: performance, energy efficiency, and temperature. This was expanded by Spacey et al. [132] to describe a formal model for partitioning tasks across distributed hardware components. Their experimental work was performed on the Axel cluster [38], but was limited to multicores and reconfigurable logic, not including the graphics engines. Lösch and Platzner [133] present reMinMin, a scheduling technique that is focused on optimizing the total energy for a set of tasks executed across multicores, graphics engines, and reconfigurable logic. They report experimental results for four applications, with the heuristic scheduling algorithm coming within 2% of optimum in all cases that were investigated.

Kicherer et al. [134] come at the problem from a completely different perspective. They are interested in the circumstance where an application is to be executed on any number of different hardware platforms, some of which include accelerators and some not. They describe an approach to enable these applications to be seamlessly portable, taking advantage of accelerators when present, but still executing correctly when not present.

The above partitioning and mapping approaches result in static assignments. There has also been some work in the area of dynamic approaches.

Bogdański et al. [135] use on-line machine learning techniques to dynamically adjust the computation as it proceeds. Karia and Lopez [136] provide a comprehensive overview of prior work in scheduling for heterogeneous system, and then propose their own, Alternative Processor within Threshold, to address some of the issues present in the prior work. Both of these groups used simulation of a multiple accelerator system for their quantitative evaluation.

Belviranli et al. [137] schedule loop iterations dynamically, resizing blocks as needed to prevent underutilization and load imbalance. Bolchini et al. [138] describe an approach that dedicates some of the computational resources to dynamic measurement of the current performance, feeding that information back into the scheduling decisions, effectively making the system self-adaptive. Both of the above groups evaluate their approaches on a pair of systems, one containing graphics engines and the other containing reconfigurable logic, but don't include a machine that contains both accelerators.

An approach that addresses both the issues of application expression and application execution is EngineCL [139]. EngineCL is a high-level framework that provides load balancing across devices that are authored in OpenCL. An experimental comparison of static versus dynamic load balancing approaches yields strong evidence for the improvements that are realizable with dynamic scheduling.

6. Conclusions and Future Directions

6.1. Conclusions

Over the past 15 years, there has been a growing interest in the use of multiple accelerators on individual applications, the utilization of *truly* architecturally diverse systems. The reasons for this are many, but the dominant consideration is that the benefits achievable through acceleration are strongly dependent upon the properties of the application as well as the properties of the implementation.

Since applications are far from monolithic, by exploiting a different accelerator for different parts of an application, one can achieve better results overall than if the accelerator choice is limited to just a single option.

The vast majority of recently developed applications that exploit multiple accelerators deploy different portions of the application to distinct accelerators. Portions of the application that work well on reconfigurable logic

are deployed on reconfigurable logic, and portions that work well on graphics engines are deployed on graphics engines. This is not surprising, as each computational platform has its own unique strengths and weaknesses.

While performance is still the dominant figure of merit, there is increasing interest in energy consumption and/or energy efficiency as an important consideration in how one wishes to deploy an application in production use. Accelerators often provide strong advantages in this circumstance, furthering the interest in their use.

6.2. Future Directions

As an application developer with over a decade of experience using architecturally diverse systems, there are a pair of things this author would like to see achieved in the future. Both items directly relate to the development tools available to authors of applications.

First, the compiler technology, especially for reconfigurable logic, remains woefully behind the capabilities of compilers for traditional software systems. Even things such as the degree of loop unrolling, which happens regularly and invisibly in software compilers, must be explicitly specified by the developer for high-level synthesis.

The reason that parameters such as loop unrolling degree are exposed to application developers is that we don't yet understand the implications of the various choices prior to deployment and empirical measurement. In the software world, the compiler has a reasonably well understood model of the implications of code generation choices that it makes. This is simply not yet the case in the domain of high-level synthesis for reconfigurable logic.

Second, debugging tools are simply in their infancy. Today, an application developer must use separate tools that are designed for the individual accelerator(s) in use. At the very least, they have different user interfaces and modes of operation. More seriously, the visibility that they provide into the operations of the application (especially those internal to an accelerator) are quite deficient.

While there is clearly need for additional work, the fundamental forces that make truly architecturally diverse systems attractive are only getting stronger. Dennard Scaling has essentially halted, and Moore's Law is slowing down. Architectural diversity is one of the more promising approaches to addressing the challenges inherent in this modern reality.

Acknowledgements

This work was supported by the National Science Foundation under grants CNS-1527510 and CNS-1763503.

References

[1] G. E. Moore, Cramming more components onto integrated circuits, Electronics 38 (8) (1965) 114–117.

[2] C. A. Mack, Fifty years of Moore's Law, IEEE Transactions on Semiconductor Manufacturing 24 (2) (2011) 202–207. doi:10.1109/TSM.2010.2096437.

[3] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, A. R. LeBlanc, Design of ion-implanted MOSFET's with very small physical dimensions, IEEE Journal of Solid-State Circuits 9 (5) (1974) 256–268. doi:10.1109/JSSC.1974.1050511.

[4] M. Bohr, A 30 year retrospective on Dennard's MOSFET scaling paper, IEEE Solid-State Circuits Society Newsletter 12 (1) (2007) 11–13. doi:10.1109/N-SSC.2007.4785534.

[5] S. Mittal, J. S. Vetter, A survey of CPU-GPU heterogeneous computing techniques, ACM Comput. Surv. 47 (4) (2015) 69:1–69:35. doi:10.1145/2788396.

[6] K. Compton, S. Hauck, Reconfigurable computing: A survey of systems and software, ACM Comput. Surv. 34 (2) (2002) 171–210. doi:10.1145/508352.508353.

[7] S. M. Trimberger, Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology, Proceedings of the IEEE 103 (3) (2015) 318–331. doi:10.1109/JPROC.2015.2392104.

[8] F. A. Escobar, X. Chang, C. Valderrama, Suitability analysis of FPGAs for heterogeneous platforms in HPC, IEEE Transactions on Parallel and Distributed Systems 27 (2) (2016) 600–612. doi:10.1109/TPDS.2015.2407896.

[9] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka,

D. Chiou, D. Burger, A cloud-scale acceleration architecture, in: Proc. of 49th IEEE/ACM International Symposium on Microarchitecture, MICRO, 2016, pp. 7:1–7:13. doi:0.1109/MICRO.2016.7783710.

[10] T. Blank, A survey of hardware accelerators used in computer-aided design, IEEE Design Test of Computers 1 (3) (1984) 21–39. doi:10.1109/MDT.1984.5005647.

[11] A. R. McKay, Comment on "Computer-Aided Design: Simulation of Digital Design Logic", IEEE Transactions on Computers C-18 (9) (1969) 862. doi:10.1109/T-C.1969.222783.

[12] D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, M. Umemura, A special-purpose computer for gravitational many-body problems, Nature 345 (6270) (1990) 33–35. doi:10.1038/345033a0.

[13] P. Hut, J. Makino, Astrophysics on the GRAPE family of special-purpose computers, Science 283 (5401) (1999) 501–505.

[14] R. D. Greenblatt, T. F. Knight, J. T. Holloway, D. A. Moon, A LISP machine, SIGIR Forum 15 (2) (1980) 137–138. doi:10.1145/1013881.802703.

[15] H. Hayashi, A. Hattori, H. Akimoto, ALPHA – a high-performance LISP machine equipped with a new stack structure and garbage collection system, in: Proc. of 10th International Symposium on Computer Architecture, ISCA, 1983, pp. 342–348. doi:10.1145/800046.801672.

[16] J. M. O'Connor, M. Tremblay, picoJava-I: the Java virtual machine in hardware, IEEE Micro 17 (2) (1997) 45–53. doi:10.1109/40.592314.

[17] M. Schoeberl, A Java processor architecture for embedded real-time systems, Journal of Systems Architecture 54 (1-2) (2008) 265–286. doi:10.1016/j.sysarc.2007.06.001.

[18] N. Jouppi, C. Young, N. Patil, D. Patterson, Motivation for and evaluation of the first Tensor Processing Unit, IEEE Micro 38 (3) (2018) 10–19. doi:10.1109/MM.2018.032271057.

[19] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, O. O. Storaasli, State-of-the-art in heterogeneous computing, Scientific Programming 18 (1) (2010) 1–33. doi:10.3233/SPR-2009-0296.

[20] E. S. Chung, P. A. Milder, J. C. Hoe, K. Mai, Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?, in: Proc. of 43rd IEEE/ACM Int'l Symposium on Microarchitecture, 2010, pp. 225–236. doi:10.1109/MICRO.2010.36.

[21] D. H. Jones, A. Powell, C.-S. Bouganis, P. Y. K. Cheung, GPU versus FPGA for high productivity computing, in: Proc. of Int'l Conference on Field Programmable Logic and Applications, FPL, 2010, pp. 119–124. doi:10.1109/FPL.2010.32.

[22] Y. Shan, T. Wu, Y. Wang, B. Wang, Z. Wang, N. Xu, H. Yang, FPGA and GPU implementation of large scale SpMV, in: Proc. of IEEE 8th Symposium on Application Specific Processors, SASP, 2010, pp. 64–70. doi:10.1109/SASP.2010.5521144.

[23] S. Che, J. Li, J. W. Sheaffer, K. Skadron, J. Lach, Accelerating compute-intensive applications with GPUs and FPGAs, in: Proc. of Symposium on Application Specific Processors, SASP, 2008, pp. 101–107. doi:10.1109/SASP.2008.4570793.

[24] S. Asano, T. Maruyama, Y. Yamaguchi, Performance comparison of FPGA, GPU and CPU in image processing, in: Proc. of Int'l Conference on Field Programmable Logic and Applications, FPL, 2009, pp. 126–131. doi:10.1109/FPL.2009.5272532.

[25] J. Bodily, B. Nelson, Z. Wei, D.-J. Lee, J. Chase, A comparison study on implementing optical flow and digital communications on FPGAs and GPUs, ACM Trans. Reconfigurable Technol. Syst. 3 (2) (2010) 6:1–6:22. doi:10.1145/1754386.1754387.

[26] J. Cong, Y. Zou, FPGA-based hardware acceleration of lithographic aerial image simulation, ACM Trans. Reconfigurable Technol. Syst. 2 (3) (2009) 17:1–17:29. doi:10.1145/1575774.1575776.

[27] X. Tian, K. Benkrid, Mersenne twister random number generation on FPGA, CPU and GPU, in: Proc. of NASA/ESA Conference on Adaptive Hardware and Systems, AHS, 2009, pp. 460–464. doi:10.1109/AHS.2009.11.

[28] D. B. Thomas, L. Howes, W. Luk, A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation, in: Proc. of ACM/SIGDA Int'l Sympo-

sium on Field Programmable Gate Arrays, FPGA, 2009, pp. 63–72. doi:10.1145/1508128.1508139.

[29] X. Tian, K. Benkrid, High-performance quasi-Monte Carlo financial simulation: FPGA vs. GPP vs. GPU, ACM Trans. Reconfigurable Technol. Syst. 3 (4) (2010) 26:1–26:22. doi:10.1145/1862648.1862656.

[30] N. Kapre, A. DeHon, Performance comparison of single-precision SPICE model-evaluation on FPGA, GPU, Cell, and multi-core processors, in: Proc. of Int'l Conference on Field Programmable Logic and Applications, FPL, 2009, pp. 65–72. doi:10.1109/FPL.2009.5272548.

[31] Z. K. Baker, M. B. Gokhale, J. L. Tripp, Matched filter computation on FPGA, Cell and GPU, in: Proc. of 15th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM, 2007, pp. 207–218. doi:10.1109/FCCM.2007.52.

[32] H. Scherl, B. Keck, M. Kowarschik, J. Hornegger, Fast GPU-based CT reconstruction using the Common Unified Device Architecture (CUDA), in: IEEE Nuclear Science Symposium Conference Record, Vol. 6, 2007, pp. 4464–4466. doi:10.1109/NSSMIC.2007.4437102.

[33] S. Rostrup, H. De Sterck, Parallel hyperbolic PDE simulation on clusters: Cell versus GPU, Computer Physics Communications 181 (12) (2010) 2165–2179. doi:10.1016/j.cpc.2010.07.049.

[34] E. J. Kelmelis, J. R. Humphrey, J. P. Durbano, F. E. Ortiz, Accelerated modeling and simulation with a desktop supercomputer, in: Proc. of SPIE, Enabling Technologies for Simulation Science X, Vol. 6227, 2006, pp. 62270N–1 – 62270N–9. doi:10.1117/12.668281.

[35] E. J. Kelmelis, J. P. Durbano, J. R. Humphrey, F. E. Ortiz, P. F. Curt, Modeling and simulation of nanoscale devices with a desktop supercomputer, in: Proc. of SPIE, Nanomodeling II, Vol. 6328, 2006, pp. 632804–1 – 632804–12. doi:10.1117/12.681085.

[36] J. H. C. Yeung, C. C. Tsang, K. H. Tsoi, B. S. H. Kwan, C. C. C. Cheung, A. P. C. Chan, P. H. W. Leong, Map-reduce as a programming model for custom computing machines, in: Proc. of 16th Int'l Symposium on Field-Programmable Custom Computing Machines, FCCM, 2008, pp. 149–159. doi:10.1109/FCCM.2008.19.

[37] N. Singla, M. Hall, B. Shands, R. D. Chamberlain, Financial Monte Carlo simulation on architecturally diverse systems, in: Proc. of Workshop on High Performance Computational Finance, WHPCF, 2008. doi:10.1109/WHPCF.2008.4745401.

[38] K. H. Tsoi, W. Luk, Axel: A heterogeneous cluster with FPGAs and GPUs, in: Proc. of 18th ACM/SIGDA Int'l Symposium on Field Programmable Gate Arrays, FPGA, 2010, pp. 115–124. doi:10.1145/1723112.1723134.

[39] S. Bauer, S. Köhler, K. Doll, U. Brunsmann, FPGA-GPU architecture for kernel SVM pedestrian detection, in: Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition – Workshops, 2010, pp. 61–68. doi:10.1109/CVPRW.2010.5543772.

[40] A. H. T. Tse, D. B. Thomas, K. H. Tsoi, W. Luk, Dynamic scheduling Monte-Carlo framework for multi-accelerator heterogeneous clusters, in: Proc. of Int'l Conference on Field-Programmable Technology, FPT, 2010, pp. 233–240. doi:10.1109/FPT.2010.5681495.

[41] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, W. Hwu, QP: A heterogeneous multi-accelerator cluster, in: Proc. of 10th LCI Int'l Conf. on High-Performance Clustered Computing, 2009.

[42] W. Kastl, T. Loimayr, A parallel computing system with specialized coprocessors for cryptanalytic algorithms, in: Proc. of Sicherheit, Gesellschaft für Informatik (GI), Berlin, Germany, 2010, pp. 73–83. URL `http://dl.gi.de/handle/20.500.12116/19801`

[43] M. A. Franklin, E. J. Tyson, J. Buckley, P. Crowley, J. Maschmeyer, Auto-Pipe and the X language: A pipeline design tool and description language, in: Proc. of Int'l Parallel and Distributed Processing Symposium, IPDPS, 2006. doi:10.1109/IPDPS.2006.1639353.

[44] R. D. Chamberlain, M. A. Franklin, E. J. Tyson, J. H. Buckley, J. Buhler, G. Galloway, S. Gayen, M. Hall, E. B. Shands, N. Singla, Auto-Pipe: Streaming applications on architecturally diverse systems, Computer 43 (3) (2010) 42–49. doi:10.1109/MC.2010.62.

[45] G. F. Diamos, S. Yalamanchili, Harmony: An execution model and runtime for heterogeneous many core systems, in: Proc. of 17th Int'l

Symposium on High Performance Distributed Computing, HPDC, 2008, pp. 197–200. doi:10.1145/1383422.1383447.

[46] R. Inta, D. J. Bowman, S. M. Scott, The "Chimera": An off-the-shelf CPU/GPGPU/FPGA hybrid computing platform, International Journal of Reconfigurable Computing 2012 (2012) Article ID 241439. doi:10.1155/2012/241439.

[47] Alankrutha P., Deepika H. V., Mangala N., N. Sarat Chandra Babu, Multi-accelerator cluster runtime adaptation for enabling discrete concurrent-task applications, in: Proc. of IEEE Int'l Advance Computing Conference, IACC, 2014, pp. 754–760. doi:10.1109/IAdCC.2014.6779418.

[48] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: A unified platform for task scheduling on heterogeneous multicore architectures, Concurrency and Computation: Practice and Experience 23 (2) (2011) 187–198. doi:10.1002/cpe.1631.

[49] Q. Wu, Y. Ha, A. Kumar, S. Luo, A. Li, S. Mohamed, A heterogeneous platform with GPU and FPGA for power efficient high performance computing, in: Proc. of Int'l Symposium on Integrated Circuits, ISIC, 2014, pp. 220–223. doi:10.1109/ISICIR.2014.7029447.

[50] J. Proaño, C. Carrión, B. Caminero, An open-source framework for integrating heterogeneous resources in private clouds, in: Proc. of 4th Int'l Conference on Cloud Computing and Services Science, CLOSER 2014, SCITEPRESS - Science and Technology Publications, Lda, Portugal, 2014, pp. 129–134. doi:10.5220/0004936601290134.

[51] S. K. Rethinagiri, O. Palomar, J. A. Moreno, O. Unsal, A. Cristal, Trigeneous platforms for energy efficient computing of HPC applications, in: Proc. of IEEE 22nd Int'l Conference on High Performance Computing, HiPC, 2015, pp. 264–274. doi:10.1109/HiPC.2015.19.

[52] G. Tan, C. Zhang, W. Wang, P. Zhang, SuperDragon: A heterogeneous parallel system for accelerating 3D reconstruction of cryo-electron microscopy images, ACM Trans. Reconfigurable Technol. Syst. 8 (4) (2015) 25:1–25:22. doi:10.1145/2740966.

[53] O. Segal, M. Margala, Exploring the performance benefits of heterogeneity and reconfigurable architectures in a commodity cloud, in:

Proc. of Int'l Conference on High Performance Computing Simulation, HPCS, 2016, pp. 132–139. doi:10.1109/HPCSim.2016.7568327.

[54] O. Segal, P. Colangelo, N. Nasiri, Z. Qian, M. Margala, SparkCL: A unified programming framework for accelerators on heterogeneous clusters, arXiv preprint arXiv:1505.01120 (2015).

[55] S. Contassot-Vivier, S. Vialle, Algorithmic scheme for hybrid computing with CPU, Xeon-Phi/MIC and GPU devices on a single machine, in: G. R. Joubert, H. Leather, M. Parsons, F. Peters, M. Sawyer (Eds.), Parallel Computing: On the Road to Exascale, Advances in Parallel Computing, IOS Press, Amsterdam, 2016, pp. 25–34. doi:10.3233/978-1-61499-621-7-25.

[56] B. Danczul, J. Fuß, S. Gradinger, B. Greslehner, W. Kastl, F. Wex, Cuteforce Analyzer: A distributed bruteforce attack on PDF encryption with GPUs and FPGAs, in: Proc. of Int'l Conference on Availability, Reliability and Security, ARES, 2013, pp. 720–725. doi:10.1109/ARES.2013.94.

[57] X. Liu, H. A. Ounifi, A. Gherbi, Y. Lemieux, W. Li, A hybrid GPU-FPGA-based computing platform for machine learning, Procedia Computer Science 141 (2018) 104–111. doi:10.1016/j.procs.2018.10.155.

[58] P. Greisen, S. Heinzle, M. Gross, A. P. Burg, An FPGA-based processing pipeline for high-definition stereo video, EURASIP Journal on Image and Video Processing 2011 (1) (2011) 18. doi:10.1186/1687-5281-2011-18.

[59] R. Bittner, E. Ruf, A. Forin, Direct GPU/FPGA communication via PCI express, Cluster Computing 17 (2) (2014) 339–348. doi:10.1007/s10586-013-0280-9.

[60] A. Sbîrlea, Y. Zou, Z. Budimlíc, J. Cong, V. Sarkar, Mapping a dataflow programming model onto heterogeneous platforms, in: Proc. of 13th ACM SIGPLAN/SIGBED Int'l Conference on Languages, Compilers, Tools and Theory for Embedded Systems, LCTES, 2012, pp. 61–70. doi:10.1145/2248418.2248428.

[61] P. Meng, M. Jacobsen, R. Kastner, FPGA-GPU-CPU heterogenous architecture for real-time cardiac physiological optical mapping, in:

Proc. of Int'l Conference on Field-Programmable Technology, FPT, 2012, pp. 37–42. doi:10.1109/FPT.2012.6412108.

[62] S. Skalicky, S. Lopez, M. Lukowiak, Distributed execution of transmural electrophysiological imaging with CPU, GPU, and FPGA, in: Proc. of Int'l Conference on Reconfigurable Computing and FPGAs, ReConFig, 2013. doi:10.1109/ReConFig.2013.6732278.

[63] D. Cacko, M. Walczak, M. Lewandowski, Low-power ultrasound imaging on mixed FPGA/GPU systems, in: Proc. of Joint Conference - Acoustics, 2018, pp. 1–6. doi:10.1109/ACOUSTICS.2018.8502371.

[64] B. da Silva, A. Braeken, E. H. D'Hollander, A. Touhafi, J. G. Cornelis, J. Lemeire, Comparing and combining GPU and FPGA accelerators in an image processing context, in: Proc. of 23rd Int'l Conference on Field Programmable Logic and Applications, FPL, 2013. doi:10.1109/FPL.2013.6645552.

[65] M. Alawieh, M. Kasparek, N. Franke, J. Hupfer, A high performance FPGA-GPU-CPU platform for a real-time locating system, in: Proc. of 23rd European Signal Processing Conference, EUSIPCO, 2015, pp. 1576–1580. doi:10.1109/EUSIPCO.2015.7362649.

[66] J. Nieto, D. Sanz, P. Guillén, S. Esquembri, G. de Arcas, M. Ruiz, J. Vega, R. Castro, High performance image acquisition and processing architecture for fast plant system controllers based on FPGA and GPU, Fusion Engineering and Design 112 (2016) 957–960. doi:10.1016/j.fusengdes.2016.04.004.

[67] S. Esquembri, J. Nieto, M. Ruiz, A. de Gracia, G. de Arcas, Methodology for the implementation of real-time image processing systems using FPGAs and GPUs and their integration in EPICS using Nominal Device Support, Fusion Engineering and Design 130 (2018) 26–31. doi:10.1016/j.fusengdes.2018.02.051.

[68] H. Zhang, B. Xiao, Z. Luo, Q. Hang, J. Yang, High-speed visible image acquisition and processing system for plasma shape and position control of EAST Tokamak, IEEE Transactions on Plasma Science 46 (5) (2018) 1312–1317. doi:10.1109/TPS.2018.2805911.

[69] J. Kocz, L. J. Greenhill, B. R. Barsdell, G. Bernardi, A. Jameson, M. A. Clark, J. Craig, D. Price, G. B. Taylor, F. Schinzel, D. Werthimer, A scalable hybrid FPGA/GPU FX correlator,

Journal of Astronomical Instrumentation 3 (1) (2014) 10pp. doi:10.1142/S2251171714500020.

[70] A. Papadopoulos, Accelerating bioinformatics and biomedical applications via massively parallel and reconfigurable systems, Ph.D. thesis, Dept. of Electrical and Computer Engineering, Univ. of Cyprus (2014). URL http://hdl.handle.net/10797/14216

[71] J. C. Kässens, L. Wienbrandt, M. Schimmler, J. Gonzalez-Dominguez, B. Schmidt, Combining GPU and FPGA technology for efficient exhaustive interaction analysis in GWAS, in: Proc. of IEEE 27th Int'l Conference on Application-specific Systems, Architectures and Processors, ASAP, 2016, pp. 170–175. doi:10.1109/ASAP.2016.7760788.

[72] L. Wienbrandt, J. C. Kässens, M. Hübenthal, D. Ellinghaus, Fast genome-wide third-order SNP interaction tests with information gain on a low-cost heterogeneous parallel FPGA-GPU computing architecture, Procedia Computer Science 108 (2017) 596–605. doi:10.1016/j.procs.2017.05.210.

[73] L. Wienbrandt, J. C. Kässens, M. Hübenthal, D. Ellinghaus, 1,000x faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis, Journal of Computational Science 30 (2019) 183–193. doi:10.1016/j.jocs.2018.12.013.

[74] M. Vogelgesang, L. Ardila Perez, M. Caselle, S. Chilingaryan, A. Kopmann, L. Rota, M. Weber, A heterogeneous FPGA/GPU architecture for real-time data analysis and fast feedback systems, in: Proc. of 5th International Beam Instrumentation Conference, IBIC, 2016, pp. 626–629. doi:10.18429/JACoW-IBIC2016-WEPG07.

[75] M. Caselle, L. Ardila Perez, M. Balzer, T. Dritschler, A. Kopmann, H. Mohr, L. Rota, M. Vogelgesang, M. Weber, A high-speed DAQ framework for future high-level trigger and event building clusters, Journal of Instrumentation 12 (3) (2017) C03015. doi:10.1088/1748-0221/12/03/C03015.

[76] N. Hu, X. Zhou, X. Li, C. Wang, 3D waveform oscilloscope implemented on coupled FPGA-GPU embedded system, in: Proc. of 5th International Conference on Information Science and Control Engineering, ICISCE, 2018, pp. 1–5. doi:10.1109/ICISCE.2018.00010.

[77] M. Caselle, L. Rota, A. Kopmann, S. Chilingaryan, M. M. Patil, W. Wang, E. Bründermann, S. Funkner, M. Nasse, G. Niehues, et al., Ultrafast linear array detector for real-time imaging, in: Proc. SPIE 10937, Optical Data Science II, 2019, p. 1093704. doi:10.1117/12.2508451.

[78] X. Wang, L. Liu, K. Huang, A. Knoll, Exploring FPGA-GPU heterogeneous architecture for ADAS: Towards performance and energy, in: S. Ibrahim, K. Choo, Z. Yan, W. Pedrycz (Eds.), Proc. of Int'l Conf. on Algorithms and Architectures for Parallel Processing, Lecture Notes in Computer Science, Vol. 10393, Springer, Cham, Switzerland, 2017, pp. 33–48. doi:10.1007/978-3-319-65482-9_3.

[79] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, J. Mars, The architectural implications of autonomous driving: Constraints and acceleration, in: Proc. of 23rd International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, ACM, 2018, pp. 751–766. doi:10.1145/3173162.3173191.

[80] R. Ammendola, M. Bernaschi, A. Biagioni, M. Bisson, M. Fatica, O. Frezza, F. Lo Cicero, A. Lonardo, E. Mastrostefano, P. S. Paolucci, D. Rossetti, F. Simula, L. Tosoratto, P. Vicini, GPU peer-to-peer techniques applied to a cluster interconnect, in: Proc. of IEEE Int'l Symposium on Parallel Distributed Processing, Workshops and PhD Forum, 2013, pp. 806–815. doi:10.1109/IPDPSW.2013.128.

[81] R. Weber, A. Gothandaraman, R. J. Hinde, G. D. Peterson, Comparing hardware accelerators in scientific applications: A case study, IEEE Transactions on Parallel and Distributed Systems 22 (1) (2011) 58–68. doi:10.1109/TPDS.2010.125.

[82] D. Yang, G. D. Peterson, H. Li, Compressed sensing and Cholesky decomposition on FPGAs and GPUs, Parallel Computing 38 (8) (2012) 421–437. doi:10.1016/j.parco.2012.03.001.

[83] P. Cooke, J. Fowers, G. Brown, G. Stitt, A tradeoff analysis of FPGAs, GPUs, and multicores for sliding-window applications, ACM Trans. Reconfigurable Technol. Syst. 8 (1) (2015) 2:1–2:24. doi:10.1145/2659000.

[84] P. Meng, M. Jacobsen, M. Kimura, V. Dergachev, T. Anantharaman, M. Requa, R. Kastner, Hardware accelerated alignment algorithm for

optical labeled genomes, ACM Trans. Reconfigurable Technol. Syst. 9 (3) (2016) 18:1–18:21. doi:10.1145/2840811.

[85] G. Chen, H. Zhou, X. Shen, J. Gahm, N. Venkat, S. Booth, J. Marshall, OpenCL-based erasure coding on heterogeneous architectures, in: Proc. of IEEE 27th Int'l Conference on Application-specific Systems, Architectures and Processors, ASAP, 2016, pp. 33–40. doi:10.1109/ASAP.2016.7760770.

[86] M. Roozmeh, L. Lavagno, Implementation of a performance optimized database join operation on FPGA-GPU platforms using OpenCL, in: Proc. of IEEE Nordic Circuits and Systems Conference, NORCAS, 2017. doi:10.1109/NORCHIP.2017.8124981.

[87] C. J. Faber, A. M. Cabrera, O. Booker, G. Maayan, R. D. Chamberlain, Data integration tasks on heterogeneous systems using OpenCL, in: Proc. of 7th International Workshop on OpenCL, IWOCL, 2019. doi:10.1145/3318170.3318187.

[88] J. Carabaño, F. Dios, M. Daneshtalab, M. Ebrahimi, An exploration of heterogeneous systems, in: Proc. of 8th Int'l Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, ReCoSoC, 2013. doi:10.1109/ReCoSoC.2013.6581542.

[89] M. Véstias, H. Neto, Trends of CPU, GPU and FPGA for high-performance computing, in: Proc. of 24th Int'l Conference on Field Programmable Logic and Applications, FPL, 2014. doi:10.1109/FPL.2014.6927483.

[90] K. O'Neal, P. Brisk, Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey, in: Proc. of IEEE Symposium on VLSI, ISVLSI, 2018, pp. 763–768. doi:10.1109/ISVLSI.2018.00143.

[91] L.-P. García, J. Cuenca, F.-J. Herrera, D. Giménez, On guided installation of basic linear algebra routines in nodes with manycore components, in: Proc. of 7th Int'l Workshop on Programming Models and Applications for Multicores and Manycores, PMAM, 2016, pp. 114–122. doi:10.1145/2883404.2883422.

[92] S. Skalicky, S. Lopez, M. Lukowiak, J. Letendre, D. Gasser, Linear algebra computations in heterogeneous systems, in: Proc.

of IEEE 24th Int'l Conference on Application-specific Systems, Architectures and Processors, ASAP, 2013, pp. 273–276. doi:10.1109/ASAP.2013.6567589.

[93] H. Giefers, P. Staar, C. Bekas, C. Hagleitner, Analyzing the energy-efficiency of sparse matrix multiplication on heterogeneous systems: A comparative study of GPU, Xeon Phi and FPGA, in: Proc. of IEEE Int'l Symposium on Performance Analysis of Systems and Software, ISPASS, 2016, pp. 46–56. doi:10.1109/ISPASS.2016.7482073.

[94] M. Malik, F. Farahmand, P. Otto, N. Akhlaghi, T. Mohsenin, S. Sikdar, H. Homayoun, Architecture exploration for energy-efficient embedded vision applications: From general purpose processor to domain specific accelerator, in: Proc. of IEEE Computer Society Annual Symposium on VLSI, ISVLSI, 2016, pp. 559–564. doi:10.1109/ISVLSI.2016.112.

[95] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, X. Tian, High performance biological pairwise sequence alignment: FPGA versus GPU versus Cell BE versus GPP, International Journal of Reconfigurable Computing 2012 (2012) Article ID 752910. doi:10.1155/2012/752910.

[96] M. Błażewicz, M. Ciżnicki, P. Kopta, K. Kurowski, P. Lichocki, Two-dimensional discrete wavelet transform on large images for hybrid computing architectures: GPU and CELL, in: M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Traff, G. Vallée, J. Weidendorfer (Eds.), Proc. of Euro-Par 2011: Parallel Processing Workshops, Vol. 7155 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 481–490. doi:10.1007/978-3-642-29737-3_53.

[97] F. Pratas, P. Transcoso, L. Sousa, A. Stamatakis, G. Shi, V. Kindratenko, Fine-grain parallelism using multi-core, Cell/BE, and GPU systems, Parallel Computing 38 (8) (2012) 365–390. doi:10.1016/j.parco.2011.08.002.

[98] K. O'Brien, I. Pietri, R. Reddy, A. Lastovetsky, R. Sakellariou, A survey of power and energy predictive models in HPC systems and applications, ACM Comput. Surv. 50 (3) (2017) 37:1–37:38. doi:10.1145/3078811.

[99] F. Wende, T. Steinke, Swendsen-Wang multi-cluster algorithm for the 2D/3D Ising model on Xeon Phi and GPU, in: Proc. of Int'l Conference for High Performance Computing, Networking, Storage and Analysis, SC13, 2013. doi:10.1145/2503210.2503254.

[100] G. Teodoro, T. Kurc, J. Kong, L. Cooper, J. Saltz, Comparative performance analysis of Intel Xeon Phi, GPU, and CPU: A case study from microscopy image analysis, in: Proc. of IEEE 28th Int'l Parallel and Distributed Processing Symposium, IPDPS, 2014, pp. 1063–1072. doi:10.1109/IPDPS.2014.111.

[101] S. S. Leang, A. P. Rendell, M. S. Gordon, Quantum chemical calculations using accelerators: Migrating matrix operations to the NVIDIA Kepler GPU and the Intel Xeon Phi, Journal of Chemical Theory and Computation 10 (3) (2014) 908–912. doi:10.1021/ct4010596.

[102] D. I. Lyakh, An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVidia Tesla GPU, Computer Physics Communications 189 (2015) 84–91. doi:10.1016/j.cpc.2014.12.013.

[103] S. Memeti, L. Li, S. Pllana, J. Kołodziej, C. Kessler, Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming productivity, performance, and energy consumption, in: Proc. of Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, ARMS-CC, 2017. doi:10.1145/3110355.3110356.

[104] Y. Thoma, A. Dassatti, D. Molla, E. Petraglio, FPGA-GPU communicating through PCIe, Microprocessors and Microsystems 39 (7) (2015) 565–575. doi:10.1016/j.micpro.2015.02.005.

[105] N. Moore, M. Leeser, L. S. King, VForce: An environment for portable applications on high performance systems with accelerators, Journal of Parallel and Distributed Computing 72 (9) (2012) 1144–1156. doi:10.1016/j.jpdc.2011.07.014.

[106] M. Zhu, L. Liu, C. Wang, Y. Xie, CNNLab: a novel parallel framework for neural networks using GPU and FPGA, arXiv preprint arXiv:1606.06234 (2016).

[107] M. Abalenkovs, A. Abdelfattah, J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, A. YarKhan, Parallel

programming models for dense linear algebra on heterogeneous systems, Supercomputing Frontiers and Innovations 2 (4) (2015) 67–86. doi:10.14529/jsfi150405.

[108] K. H. Tsoi, A. H. Tse, P. Pietzuch, W. Luk, Programming framework for clusters with heterogeneous accelerators, SIGARCH Comput. Archit. News 38 (4) (2010) 53–59. doi:10.1145/1926367.1926377.

[109] T. Ahmed, OpenCL framework for a CPU, GPU, and FPGA platform, Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto (Dec. 2011).
URL `http://hdl.handle.net/1807/30149`

[110] J. Auerbach, D. F. Bacon, I. Burcea, P. Cheng, S. J. Fink, R. Rabbah, S. Shukla, A compiler and runtime for heterogeneous computing, in: Proc. of 49th Design Automation Conference, DAC, 2012, pp. 271–276. doi:10.1145/2228360.2228411.

[111] S. V. Cole, J. Buhler, MERCATOR: A GPGPU framework for irregular streaming applications, in: Proc. of Int'l Conference on High Performance Computing Simulation, 2017, pp. 727–736. doi:10.1109/HPCS.2017.111.

[112] A. Papakonstantinou, K. Gururaj, J. A. Stratton, D. Chen, J. Cong, W.-M. W. Hwu, Efficient compilation of CUDA kernels for high-performance computing on FPGAs, ACM Trans. Embed. Comput. Syst. 13 (2) (2013) 25:1–25:26. doi:10.1145/2514641.2514652.

[113] S. Lee, J. Kim, J. S. Vetter, OpenACC to FPGA: A framework for directive-based high-performance reconfigurable computing, in: Proc. of IEEE Int'l Parallel and Distributed Processing Symposium, IPDPS, 2016, pp. 544–554. doi:10.1109/IPDPS.2016.28.

[114] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, K. Bertels, A survey and evaluation of FPGA high-level synthesis tools, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (10) (2016) 1591–1604. doi:10.1109/TCAD.2015.2513673.

[115] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, S. Matsuoka, Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs, in: Proc. of Int'l Conference for High Perfor-

mance Computing, Networking, Storage and Analysis, SC'16, 2016, pp. 409–420. doi:10.1109/SC.2016.34.

[116] A. M. Cabrera, R. D. Chamberlain, Exploring portability and performance of OpenCL FPGA kernels on Intel HARPv2, in: Proc. of 7th International Workshop on OpenCL, IWOCL, 2019. doi:10.1145/3318170.3318180.

[117] T. Kenter, Invited Tutorial: OpenCL design flows for Intel and Xilinx FPGAs: Using common design patterns and dealing with vendor-specific differences, in: FSP Workshop 2019; Sixth International Workshop on FPGAs for Software Programmers, 2019, pp. 1–8.

[118] H. Nguyen, GPU Gems 3, Addison-Wesley Professional, 2007.

[119] R. Nane, V. Sima, B. Olivier, R. Meeuws, Y. Yankova, K. Bertels, DWARV 2.0: A CoSy-based C-to-VHDL hardware compiler, in: Proc. of 22nd Int'l Conference on Field Programmable Logic and Applications, 2012, pp. 619–622. doi:10.1109/FPL.2012.6339221.

[120] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, J. H. Anderson, LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems, ACM Trans. Embed. Comput. Syst. 13 (2) (2013) 24:1–24:27. doi:10.1145/2514740.

[121] N. Trifunovic, B. Perovic, P. Trifunovic, Z. Babovic, A. R. Hurson, A novel infrastructure for synergistic dataflow research, development, education, and deployment: the Maxeler AppGallery project, in: Advances in Computers, Vol. 106, Elsevier, 2017, pp. 167–213.

[122] J. Villarreal, A. Park, W. Najjar, R. Halstead, Designing modular hardware accelerators in C with ROCCC 2.0, in: Proc. of 18th IEEE Int'l Symposium on Field-Programmable Custom Computing Machines, 2010, pp. 127–134. doi:10.1109/FCCM.2010.28.

[123] C. B. Ciobanu, A. L. Varbanescu, D. Pnevmatikatos, G. Charitopoulos, X. Niu, W. Luk, M. D. Santambrogio, D. Sciuto, M. Al Kadi, M. Huebner, et al., EXTRA: Towards an efficient open platform for reconfigurable high performance computing, in: Proc. of IEEE 18th Int'l Conference on Computational Science and Engineering, 2015, pp. 339–342.

[124] C. B. Ciobanu, G. Stramondo, A. L. Varbanescu, A. Brokalakis, A. Nikitakis, L. D. Tucci, M. Rabozzi, L. Stornaiuolo, M. Santambrogio, G. Chrysos, C. Vatsolakis, C. Georgios, D. Pnevmatikatos, EXTRA: An open platform for reconfigurable architectures, in: Proc. of 18th Int'l Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS, 2018, pp. 220–229. doi:10.1145/3229631.3236092.

[125] R. Cattaneo, G. Natale, C. Sicignano, D. Sciuto, M. D. Santambrogio, On how to accelerate iterative stencil loops: A scalable streaming-based approach, ACM Trans. Archit. Code Optim. 12 (4) (2015) 53:1–53:26. doi:10.1145/2842615.

[126] Wenlai Zhao, Haohuan Fu, W. Luk, Teng Yu, Shaojun Wang, Bo Feng, Yuchun Ma, Guangwen Yang, F-CNN: An FPGA-based framework for training convolutional neural networks, in: Proc. of IEEE 27th Int'l Conference on Application-specific Systems, Architectures and Processors, 2016, pp. 107–114. doi:10.1109/ASAP.2016.7760779.

[127] J. Arram, T. Kaplan, W. Luk, P. Jiang, Leveraging FPGAs for accelerating short read alignment, IEEE/ACM Trans. Comput. Biol. Bioinformatics 14 (3) (2017) 668–677. doi:10.1109/TCBB.2016.2535385.

[128] B. Lindsey, M. Leslie, W. Luk, A domain specific language for accelerated multilevel Monte Carlo simulations, in: Proc. of IEEE 27th Int'l Conference on Application-specific Systems, Architectures and Processors, 2016, pp. 99–106. doi:10.1109/ASAP.2016.7760778.

[129] G. Inggs, D. B. Thomas, W. Luk, A domain specific approach to high performance heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 28 (1) (2017) 2–15. doi:10.1109/TPDS.2016.2563427.

[130] Q. Liu, W. Luk, Objective-driven workload allocation in heterogeneous computing systems, in: Proc. of International Conference on Field-Programmable Technology, FPT, 2011. doi:10.1109/FPT.2011.6132695.

[131] Q. Liu, W. Luk, Heterogeneous systems for energy efficient scientific computing, in: O. Choy, R. Cheung, P. Athanas, K. Sano (Eds.), Proc. of 8th Int'l Symp. on Applied Reconfigurable Computing, Reconfigurable Computing: Architectures, Tools and Applications, Vol. 7199

of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 64–75. doi:10.1007/978-3-642-28365-9_6.

[132] S. Spacey, W. Luk, D. Kuhn, P. H. Kelly, Parallel partitioning for distributed systems using sequential assignment, Journal of Parallel and Distributed Computing 73 (2) (2013) 207–219. doi:10.1016/j.jpdc.2012.09.019.

[133] A. Losch, M. Platzner, reMinMin: A novel static energy-centric list scheduling approach based on real measurements, in: Proc. of IEEE 28th Int'l Conference on Application-specific Systems, Architectures and Processors, ASAP, 2017, pp. 149–154. doi:10.1109/ASAP.2017.7995272.

[134] M. Kicherer, F. Nowak, R. Buchty, W. Karl, Seamlessly portable applications: Managing the diversity of modern heterogeneous systems, ACM Trans. Archit. Code Optim. 8 (4) (2012) 42:1–42:20. doi:10.1145/2086696.2086721.

[135] M. Bogdanski, P. R. Lewis, T. Becker, X. Yao, Improving scheduling techniques in heterogeneous systems with dynamic, on-line optimisations, in: Proc. of Int'l Conference on Complex, Intelligent, and Software Intensive Systems, 2011, pp. 496–501. doi:10.1109/CISIS.2011.81.

[136] S. S. Karia, S. Lopez, Alternative processor within threshold: Flexible scheduling on heterogeneous systems, in: Proc. of IEEE Int'l Parallel and Distributed Processing Symposium Workshops, 2017, pp. 42–53. doi:10.1109/IPDPSW.2017.175.

[137] M. E. Belviranli, L. N. Bhuyan, R. Gupta, A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures, ACM Trans. Archit. Code Optim. 9 (4) (2013) 57:1–57:20. doi:10.1145/2400682.2400716.

[138] C. Bolchini, G. C. Durelli, A. Miele, G. Pallotta, M. D. Santambrogio, An orchestrated approach to efficiently manage resources in heterogeneous system architectures, in: Proc. of 33rd IEEE Int'l Conference on Computer Design, ICCD, 2015, pp. 200–207. doi:10.1109/ICCD.2015.7357104.

[139] M. A. Dávila Guzmán, R. Nozal, R. Gran Tejero, M. Villarroya-Gaudó, D. Suárez Gracia, J. L. Bosque, Cooperative CPU, GPU, and

FPGA heterogeneous execution with EngineCL, The Journal of Supercomputing 75 (3) (2019) 1732–1746. doi:10.1007/s11227-019-02768-y.