Scheduling Challenges and Opportunities in Integrated CPU+GPU Processors

(Invited Special Session Paper)

Kapil Dev
School of Engineering
Brown University
Providence, RI 02912
kapil_dev@brown.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI 02912
sherief_reda@brown.edu

ABSTRACT

Heterogeneous processors with architecturally different devices (CPU and GPU) integrated on the same die provide good performance and energy efficiency for wide range of workloads. However, they also create challenges and opportunities in terms of scheduling workloads on the appropriate device. Current scheduling practices mainly use the characteristics of kernel workloads to decide the CPU/GPU mapping. In this paper we first provide detailed infrared imaging results that show the impact of mapping decisions on the thermal and power profiles of CPU+GPU processors. Furthermore, we observe that runtime conditions such as power and CPU load from traditional workloads also affect the mapping decision. To exploit our observations, we propose techniques to characterize the OpenCL kernel workloads during run-time and map them on appropriate device under time-varying physical (i.e., chip power limit) and CPU load conditions, in particular the number of available CPU cores for the OpenCL kernel. We implement our dynamic scheduler on a real CPU+GPU processor and evaluate it using various OpenCL benchmarks. Compared to the state-of-the-art kernel-level scheduling method, the proposed scheduler provides up to 31% and 10% improvements in runtime and energy, respectively.

Keywords

Heterogeneous CPU+GPU processors; infrared imaging; scheduling; OpenCL kernels

1. INTRODUCTION

Heterogeneous CPU+GPU processors are becoming mainstream due to their versatility in terms of performance and power tradeoffs [13]. Further, emerging programming paradigms (e.g., OpenCL) for these processors allow mapping the kernel workloads fluidly between CPU and GPU devices. However, scheduling a kernel on the correct device

is a nontrivial task. To this end, recent years have witnessed multiple research efforts devoted to efficient scheduling schemes for heterogeneous systems [13, 15, 7, 1, 19, 14]. The survey paper by Mittal *et al.* provides an excellent overview of the state-of-art techniques for such systems [13]. We notice that most of the recent work has focused on discrete GPUs. In this paper, we focus on the integrated CPU+GPU processors, where the CPU and GPU devices share a total power budget and have strong thermal interactions since the two devices are on the same die.

Diamos *et al.* [7], Augonnet *et al.* [1], and Lee *et al.* [11] propose performance-aware dynamic scheduling solutions for single application cases running on discrete GPU systems. Pienaar *et al.* propose a model-driven runtime solution similar to OpenCL, but their approach requires writing programs using nonstandard constructs for implementing directed acyclic graphs [15]. In Maestro, data orchestration and tuning is proposed for OpenCL devices using an additional layer over the existing OpenCL Runtime [16]. In Qilin, adaptive mapping of computations on CPU and GPU devices is implemented to minimize both runtime and energy of system [12]; however, they require the complete application to be rewritten using their custom APIs. In this work, we present a scheduler that does not require re-writing the application; instead, the proposed scheduler exchanges relevant information with the application for scheduling its kernels on appropriate devices.

Application-level device-contention aware scheduling schemes based on average historical runtime have also been proposed [9, 10, 4]. However, such schemes may not lead to an appropriate device selection for all kernels because different kernels may have different preferred devices based on the data size and kernel characteristics [15]. Other works proposed kernel-level scheduling [15, 7, 1, 19, 2]. However, all prior work assumed fixed system conditions (e.g., chip power cap and CPU workload conditions), leading to potentially inefficient scheduling under dynamic system conditions. As shown in this paper, the runtime and energy of the kernel is a function of both kernel characteristics and the run-time conditions. The chip power cap could change dynamically for multiple reasons such as saving battery life, adapting to the user behavior in mobile system, or to meet different power budgets in server processors. Similarly, the number of CPU cores available for the kernel may vary depending on the existence of other workloads or clock/power gating of certain cores by server power management unit [5].

Bailey *et al.* studied scheduling for power constrained

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ESTIMedia'16, October 01-07, 2016, Pittsburgh, PA, USA

© 2016 ACM. ISBN 978-1-4503-4543-9/16/10...\$15.00

DOI: <http://dx.doi.org/10.1145/2993452.2994307>

systems [2]; however, their approach is only applied in an off-line mode as it lacked the capability to switch from CPU to GPU or vice-versa during run-time. In this work, we propose a dynamic scheduler that takes the run-time conditions in to account for efficient scheduling. In particular, the main contributions of this work are as follows.

- We use infrared imaging techniques to elucidate the impact of scheduling decisions on the thermal and power profiles of CPU+GPU processors.
- We identify some of the challenges and opportunities in scheduling on CPU+GPU processors. In particular, we show that the best scheduling device on these processors not only depends on the kernel characteristics but also depends on the system run-time conditions such as power cap and number of available CPU cores.
- We propose a machine learning based scheduling framework for CPU+GPU processors that makes the device selection decision to minimize runtime or energy by taking both kernel characteristics and run-time system conditions (e.g., processor power cap and number of available CPU cores) into account. In contrast to previous works [2, 19], which focus only on run-time minimization, our scheduler could provide better scheduling for both runtime or energy, depending on the user preference.
- We implemented and tested our scheduler on a real state-of-the-art CPU+GPU processor-based system using OpenCL benchmarks. For the studied benchmarks, we show that the proposed power-aware dynamic scheduler provides up to 31% better performance than the state-of-the-art scheduling schemes. Further, our scheduler provides up to 10% higher energy savings than the developer-based energy minimization scheduling choice.

The rest of the paper is organized as follows. We discuss the challenges and opportunities for scheduling on CPU+GPU processors in section 2. In section 3, we describe the proposed scheduling framework for efficient mapping of kernels on CPU+GPU processors. The experimental setup and the scheduling results are presented in section 4 and Section 5 concludes the paper.

2. CHALLENGES & OPPORTUNITIES IN SCHEDULING

In this section, first we highlight the thermal and power effects of scheduling for both CPU-only (e.g., SPEC workloads [17]) and CPU/GPU applications (e.g. OpenCL kernels). Then, we discuss two key challenges and opportunities for efficient scheduling of kernels on CPU+GPU processors.

2.1 Thermal & Power Effects of Scheduling

1. Scheduling of OpenCL kernels on CPU or GPU devices has big impact on thermal and power characteristics. Figure 1 shows the detailed thermal and power maps of processor when a vector-multiplication (VectMult) kernel is launched on its CPU and GPU devices. We obtained the thermal and power maps using infrared imaging techniques proposed in our earlier work [6]. As we can see, the two devices have different runtime, total power and

Figure 1: Detailed thermal and power maps of an AMD A10-5700 processor for VectMult kernel scheduled on CPU and GPU devices. CPU x86 modules are located on the left side and GPU cores on the right side of the die.

peak temperature. For this kernel, the best scheduling device is GPU because it not only performs better in runtime (5.5 $\times$), but also dissipates lesser power (0.8 $\times$) with lower peak die temperature (26 $^{\circ}$ C lower). Hence, it is important to schedule OpenCL kernels on appropriate device because scheduling provides both thermal and power management. The correct device not only depends on the kernel characteristics, but it also depends on the run-time system conditions, as described in the next sub-section.

2. Scheduling of traditional CPU-based application impact power and thermal budget available for GPU. Figure 2 shows the thermal and power maps of a CPU+GPU processor when a CPU-only benchmark (hmmmer) is launched on two different CPU cores: C0 and C3. We observe that the performance, peak temperature and total power changes only slightly (0.4%, 2.2 $^{\circ}$ C and 2.9 W, respectively), because all four CPU cores of the processor have the same micro-architecture. Given a total power budget for the chip, the power consumed by the CPU reduces

Figure 2: Detailed thermal and power maps of an AMD A10-5700 processor for hmmmer benchmark scheduled on CPU cores 0 and 3.

the power budget available for the GPU. Furthermore, the scheduling decision on the CPU cores can impact the temperature of the GPU; for instance, when `hammer` is scheduled on core C3, the temperature of GPU increases by about 10 °C compared to scheduling it on core C0.

2.2 Impact of Run-time Conditions

1. Chip power cap could affect scheduling decision. As the chip power cap could change dynamically for saving battery life or meeting different power budgets in processors, we show that the power cap can potentially impact the scheduling decisions. Figure 3 shows runtime and energy versus power cap trends for two kernels (LUD.K2 and LBM.K1) that we observed on our CPU+GPU processor. In particular, we observe two types of trends in runtime and energy. In one trend the runtime/energy of the LBM.K1/LUD.K2 kernels are always lower on GPU or CPU devices irrespective of the power cap, while in the second trend (runtime for the LUD.K2 kernel and energy for the LBM.K1 kernel) the optimal device is a function of its power cap. These trends arise because of the kernel characteristics and the differences in maximum operating frequency and architectures of CPU and GPU devices in a CPU+GPU processor. On the studied processor, the maximum GPU frequency is 1.25 GHz, while the maximum CPU frequency is 4 GHz. Hence, a power-aware scheduler is needed to maximize the performance or energy savings on CPU+GPU processors.

2. Number of available CPU cores could impact scheduling decision. Figure 4 shows the scheduling maps that minimize runtime for two kernels (LUD.K2 and LBM.K1) when number of CPU cores available to the kernel are varied from 1 to 4. The availability of CPU cores to OpenCL kernels could change because of utilization from traditional CPU-bound applications. First, we notice that, at a fixed power cap (say 80 W), as the number of CPU cores available for the kernel reduces, the best device for runtime could change from CPU to GPU. This is because the compute throughput of CPU reduces with decrease in number of CPU cores, which in turn increases the kernel runtime on CPU. This is an expected behavior, but it has implications on the scheduling decisions as shown here. Second, from the scheduling maps in Figure 4, we notice that the impact of power cap (also shown in Figure 3 earlier) and the num-

Figure 3: Runtime and energy versus power-cap for two kernels: (a) LUD.K1 and (b) LBM.K2 on GPU/CPU devices, of an Intel Haswell processor.

Figure 4: Runtime-optimal devices for LUD.K2 and LBM.K1 kernels at 3 different power caps (20, 40, 80W) and 4 different number of CPU-cores (1C to 4C) for an Intel Haswell processor.

ber of available CPU cores on performance is higher for the LUD.K2 kernel than for LBM.K1. Therefore, as the number of the available CPU cores reduce from 4 to 2, the runtime optimal device for LBM.K1 is still CPU, while it changes to GPU for the LUD.K2 kernel. Since existing GPUs do not support running multiple kernels simultaneously, we focus on CPU load-aware scheduling in this work.

3. PROPOSED METHODOLOGY

In this section, we describe the proposed framework and the machine-learning models used to achieve kernel-level scheduling under varying runtime conditions such as power cap and CPU utilization from traditional workloads.

Framework Architecture. The high-level organization of our framework is given in Figure 5. The figure shows interactions among the OS, OpenCL Runtime and the proposed *scheduler*. Since device mapping decisions from our scheduler needs to be communicated to the OpenCL application for implementation, we developed custom APIs implemented in C language to exchange information between the application and the scheduler using efficient UNIX sockets. These APIs have negligible overhead on the actual application as demonstrated by our results later in the paper. To make correct decision for all kernels of an application, the scheduler keeps track of the current phase for each kernel by exchanging kernel-id (kID), phase (e.g., kernel-enqueue), and process-id (pid) information with the application. The scheduler monitors the hardware performance counters (PMCs) and run-time conditions (i.e., chip power cap and number of free CPU cores) from the OS, and uses them as inputs to an *a priori* trained support-vector machine (SVM) classifier to predict the device that minimizes energy or runtime for the kernel.

Kernel Classification. Similar to the previous work [19], we also use SVM-based classifiers to predict the optimal device in our scheduler because for scheduling purposes, it is sufficient to predict the relative ratio of energy or runtime of each device (CPU/GPU) for a kernel; i.e., the exact values of runtime and energy on two devices are not needed. Unlike our work, the previous work used only static code profiling and work group sizes to make device decisions without considering the run-time conditions (e.g., power cap and CPU-load or number of available cores) in the scheduling process. So, there could be performance loss in a dynamic system environment, as shown in the results section.

Our goal is to build a classifier based on measurements on only one device (CPU or GPU) so that during testing correct device could be predicted by running a kernel only on one device. On our system, the publicly available driver does not support collecting GPU counters in real time, so we use

Figure 5: Block diagram of the proposed scheduler for CPU+GPU processors.

CPU measurements to predict the appropriate device. During training phase, we run multiple OpenCL applications on both CPU and GPU devices at different power cap and workload conditions while collecting the PMCs only during CPU runs. These PMCs, along with system conditions are used as features to build reliable SVM models (off-line process). Table 1 shows the list of PMCs used as the feature space for the SVM-based classifier. We selected these PMCs to represent the characteristics of different kernels and to minimize classification accuracy for finding suitable device across all kernels and system conditions. For example, kernels with higher branch instructions benefit more on CPU, while those with higher LLC misses and resource stalls perform better on GPU because the latencies due to such misses and stalls are hidden by GPU architecture. The trained models are stored and used by the scheduler to make suitable decisions for different kernels. In particular, we use the following standard SVM equation to predict the appropriate device ($c < 0$ for GPU, and CPU otherwise) at run-time:

$$c = \sum_{i=1}^n \alpha_i \cdot l_i \cdot \kappa(s_i, x) + b, \quad (1)$$

where, n denotes the number of support vectors, b denotes the bias of SVM classifier, x is the test feature, and s_i and l_i denote the i^{th} support vector and its class-label, respectively. We studied different kernel-functions (linear, gaussian and polynomial) to partition the feature space using SVM and found the polynomial kernel function performing the best. This is mainly because the performance and energy are typically nonlinear function of system statistics. In particular, we used 3^{rd} order polynomial basis function, denoted as $\kappa(s_i, x) = (1 + s_i \cdot x)^3$, in our SVM classifier. Although training an SVM model is computationally expensive, the testing is far less expensive than the training. Further, it is worth mentioning that these models need to be built only once, so if the underlying hardware changes, the SVM models could be retrained for the new hardware. Furthermore, we build different SVM models for runtime and energy goals as the device decision for minimizing one is typically different than the other.

Table 1: List of performance counters

INSTRUCTIONS RETIRED	LLC MISSES
UNHALTED REFERENCE CYCLES	L1 DCACHE LOAD MISSES
CPU CLK UNHALTED	L1 DCACHE STORE MISSES
BRANCH INSTRUCTIONS RETIRED	L2 RQSTS:ALL DEMAND MISS
BRANCH MISSES	RESOURCE STALLS:ANY

Table 2: OpenCL benchmarks and their kernels

Benchmarks	#Kernels
Streamcluster (SC)	1 (K1)
Particlefilter (PF)	4 (K2-K5)
LU Decomposition (LUD)	3 (K6-K8)
Distance-cutoff Coulombic potential (CUTCP)	1 (K9)
Lattice-Boltzmann method (LBM)	1 (K10)
Hotspot (HS)	1 (K11)
Heartwall (HW)	1 (K12)
Computational fluid-dynamics (CFD)	3 (K13-K15)
Sparse matrix-vector multiplication (SPMV)	1 (K16)
General Matrix Multiply (SGEMM)	1 (K17)
Stencil (STL)	1 (K18)
Finite-difference time-domain method (FDTD)	3 (K19-K21)
Gram-Schmidt Process (GSCHM)	3 (K22-K24)

4. EXPERIMENTAL RESULTS

Setup: We perform our experiments on a real system equipped with an Intel Haswell Core i7-4790K CPU-GPU processor and 16 GB memory. The CPU has 4 cores with 4×256 KB L2 caches and an 8 MB L3 cache. It has an HD graphics 4600 GPU with 20 execution units, each with 16 cores, integrated on the same die. The nominal frequency of CPU cores is 4 GHz, while the GPU cores run up to 1.25 GHz. To show the effectiveness of the proposed scheduler, we have selected 13 OpenCL benchmarks from 3 popular benchmark suites to cover wide range of workload characteristics – Rodinia [3], Parboil [18], and Polybench [8]. The list of benchmark, along with the number of kernels in each benchmark are shown in Table 2. In total, there are 24 OpenCL kernels. Further, to vary the CPU-load, we run one or more instances of workloads from SPEC CPU2006 suite [17] on CPU cores.

Furthermore, to build the SVM models for minimizing energy, we measure the energy of each kernel using Intel’s running average power limit (RAPL) APIs. We implement the time-varying power cap by setting hardware model-specific registers (MSR) corresponding to package power limit. Further, in this paper, we focus mainly on applications with multiple kernel iterations, which is typically the case for the scientific applications. The runtime/energy of a kernel in its very first iteration could be significantly different than its runtime/energy in the latter iterations due to unknown hardware state and cache warm up during the first iteration. Therefore, we use first two iterations of kernels to run on CPU to collect the performance counters and use them in the SVM model to predict the best device. Further, the scheduler keeps the history of optimal device for each kernel under different power caps to avoid invoking the SVM predictor for the previously seen power cap value.

First Experiment: In first experiment, we provide results showing the impact of power cap and CPU-load conditions on the scheduling decisions for minimizing runtime and energy. We compare our proposed scheduler against the following two state-of-the-art scheduling methods.

1. **Application-level-user-preference-based (App-level):** It chooses the device (CPU or GPU) that minimizes the total runtime or energy at application-level, instead of kernel-level [10, 4]. This case could also represent the user/developer’s static method of selecting the optimal device. Also, this method is both power- and CPU load-oblivious as the user typically profiles the program at only default/maximum power cap when no other workload is running on the system.

2. **Kernel-level-static-conditions-based (K-level)**: It chooses the device that minimizes the total runtime at kernel-level assuming fixed power cap and CPU-load conditions [19]. This method performs better than **App-level** method, but suffers performance/energy loss under dynamic conditions.

Figure 6 shows the comparison of performance for different scheduling methods under two different power cap and CPU-core conditions for selected kernels. The figure also shows the average performance gains/loss for all 24 kernels in the last set of bars. While both **App-level** and **K-level** methods are assumed to be profiled at fixed 80 W power cap under no-workload conditions, our proposed method, also called **Ours** in Figure 6 considers both run-time physical and CPU load-conditions, leading to better overall performance.

All results in Figure 6 are normalized to the **App-level** case, which typically performs worse than both **K-level** and **Ours** cases. Under no-workload conditions (i.e., 4-core case), both **K-level** and **Ours** methods provide similar, but better performance than the **App-level** method. This is because both **K-level** and **Ours** methods make decisions at kernel-level and for some of the selected kernels (e.g., K3, K13), the scheduling device has huge impact on their performance. For example, kernel K3 is too short in duration that sending it to GPU causes un-necessary runtime overhead. The **App-level** scheduling method wrongly chooses GPU device for this kernel because K3, along with three other kernels K2, K4 and K5, belong to the same application (**particle-filter**); among them, K2 and K5 are dominant in runtime

Figure 6: Comparison of runtime against state-of-the-art schedulers (**App-level** and **K-level**) at two power cap values and CPU-load conditions: a) OpenCL on 4 cores at 80 W, b) OpenCL on 1 core at 20 W. **App-level** scheduler chooses the best device at application-level [4] and **K-level** method picks the best device at kernel-level [19]; unlike **Ours**, the other two methods assume fixed system conditions.

Figure 7: Demonstration of dynamic scheduling to minimize energy for 3 kernels, k1-k3 of the LUD application; (a) time-varying power cap (b) the execution of three kernels on different devices over time; (c) normalized energy for three different scheduling schemes (GPU, CPU, Ours).

and their performance is better on GPU, so the (**App-level**) method chooses GPU for all four kernels of this application.

The performance gains from **Ours** method increase as the power cap changes from 80 W to 20 W and the number of available cores change from 4 to 1, as shown in Figure 6(a)-(b). This is because **Ours** method finds out the current CPU-load on the system before making a scheduling decision, while the other two methods do not take such system information into account. In particular, for the 1 core case, wherein 3 out of 4 cores are busy with other workloads and only 1 CPU-cores is available for the OpenL kernel, **Ours** method provides 40% and 31% better average performance than **App-level** and **K-level** methods, respectively.

Further, as seen in Figure 6(a-b), it's possible that the **K-level** method could lead to worse scheduling results (e.g., K13 and K15) than the **App-level** method. This is because the kernel-level decisions made under fixed power cap (80 W) and under no-load conditions could be different than the decision at **App-level** under the same conditions. It is worth mentioning that the correct device decisions by **App-level** method for these two kernels is purely incidental. However, the correct decisions from **Ours** method are not incidental because **Ours** method takes varying system conditions into account while making device decisions.

Second Experiment: To show the behavior of our scheduler over time, we applied arbitrary time-varying power capping traces to the hardware by writing to the package power limit MSR. Figure 7 shows the response of our scheduler for minimizing the energy for the LUD application with 3 kernels (k1-k3). Similar scheduling behavior is observed for runtime minimization. As we can see from the Figure, the optimal device selection at different power caps varies across different kernels. For kernel k1 the scheduler selects CPU as the optimal device under all power caps. This is because each iteration of k1 is a relatively short (< 0.1 ms) and therefore, the overhead of launching it on GPU leads to higher energy than if it is run on CPU itself. On the other hand, kernel k2 is always scheduled on GPU to minimize the energy. The optimal device for k3 kernel is power-dependent.

For this kernel, CPU is the optimal energy device at lower power cap, while GPU minimizes the energy at higher power cap. Our scheduler accurately predicts the energy-optimal devices for each kernel under a time-varying power cap.

We compare the total energy dissipation of the LUD applications in 3 scheduling cases: GPU, CPU, and Ours method. Here, GPU case is the same as the App-level method decision [4] as GPU provides lower energy at high power cap value. Obviously, the energy savings from Ours method will depend on the time-varying power capping-pattern. Nevertheless, for the selected power cap trace, Ours method provides up to 10% and 24% lower energy than the App-level (GPU) and CPU device cases, respectively.

We evaluated the runtime overhead of adding custom APIs and SVM to all applications. For all kernels executed under different run-time conditions, the maximum runtime overhead is about 1.9%. Given high potential of performance improvement and energy savings, we believe that this overhead is reasonably acceptable.

5. CONCLUSIONS & FUTURE WORK

In this paper, we highlighted the challenges and opportunities in scheduling on CPU+GPU processors. To address the shortcomings of previous approaches, we presented a scheduling method that takes in to account the system dynamic conditions, along with the kernel characteristics to minimize runtime or energy on these processors. We fully implemented the scheduler and tested it on a real integrated CPU+GPU system. The proposed scheduler outperforms the application-based and state-of-the-art scheduling methods by 40% and 31%, respectively. Similarly, our scheduling framework provides up to 10% more energy saving for selected time-varying power capping pattern than the user-based application-level scheduling scheme.

Future Directions: First, in the current CPU+GPU processors, the total power is distributed between CPU and GPU devices by the default drivers (e.g., Intel RAPL and *pstate* drivers), which vary the DVFS of CPU and GPU cores independently. Further, the current DVFS schemes are oblivious of the scheduling policies. As a future work, we would vary the DVFS of both CPU and GPU devices synergistically and combine that with the run-time aware scheduling to further improve the performance and energy efficiency of these processors. Second, we would apply the proposed scheduling techniques to systems with more than two devices (e.g., multiple CPUs and GPUs of different power and performance limits). Finally, as the future GPU architectures evolve and allow running multiple workloads simultaneously, we could add GPU-load awareness also in our scheduler.

Acknowledgments: We thank Xin Zhan for his help with some of the experiments. This work is partially supported by NSF grants 1305148 and 1438958.

6. REFERENCES

- [1] C. Augonnet et al. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurr. Comput. : Pract. Exper.*, 23(2):187–198, Feb. 2011.
- [2] P. E. Bailey et al. Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems. In *International Conference on Parallel Processing*, pages 371–380, Sept 2014.
- [3] S. Che et al. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Intl. Symp. on Workload Characterization*, pages 44–54, Oct 2009.
- [4] H. J. Choi et al. An Efficient Scheduling Scheme Using Estimated Execution Time for Heterogeneous Computing Systems. *The Journal of Supercomputing*, pages 886–902, 2013.
- [5] K. Dev et al. Workload-aware Power Gating Design and Run-time Management for Massively Parallel GPGPUs. In *IEEE Computer Society Annual Symposium on VLSI*, pages 242–247, 2016.
- [6] K. Dev, A. Nowroz, and S. Reda. Power mapping and modeling of multi-core processors. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 39–44, Sept 2013.
- [7] G. F. Diamos and S. Yalamanchili. Harmony: An Execution Model and Runtime for Heterogeneous Many Core Systems. In *Intl. Symp. on High Perf. Distributed Computing*, pages 197–200, 2008.
- [8] S. Grauer-Gray et al. Auto-Tuning a High-Level Language Targeted to GPU Codes. In *Innovative Parallel Computing*, pages 1–10, May 2012.
- [9] C. Gregg, J. S. Brantley, and K. Hazelwood. Contention-Aware Scheduling of Parallel Code for Heterogeneous Systems. In *HotPar’10*, 2010.
- [10] C. Gregg et al. Dynamic Heterogeneous Scheduling Decisions Using Historical Runtime Data. In *Proc. of the 2nd workshop on applications for multi- and many-core processors*, 2011.
- [11] J. Lee et al. SKMD: Single Kernel on Multiple Devices for Transparent CPU-GPU Collaboration. *ACM Tran. Comp. Sys.*, 33:1–27, Aug. 2015.
- [12] C.-K. Luk, S. Hong, and H. Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In *Proc. of the IEEE/ACM Intl. Symp. on Microarchitecture*, pages 45–55, 2009.
- [13] S. Mittal and J. S. Vetter. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comp. Surv.*, 47(4):1–35, July 2015.
- [14] P. Pandit and R. Govindarajan. Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices. In *Proc. of Intl. Symp. on Code Generation and Optimization*, pages 273–283, 2014.
- [15] J. A. Pienaar, A. Raghunathan, and S. Chakradhar. MDR: Performance Model Driven Runtime for Heterogeneous Parallel Platforms. In *Proc. of the Intl. Conference on Supercomputing*, pages 225–234, 2011.
- [16] K. Spafford, J. Meredith, and J. Vetter. Maestro: Data Orchestration and Tuning for OpenCL Devices. In *Proc. of the International Euro-Par Conference on Parallel Processing: Part II*, pages 275–286, 2010.
- [17] C. D. Spradling. SPEC CPU2006 Benchmark Tools. *SIGARCH Comp. Arch. News*, 35(1):130–134, 2007.
- [18] J. A. Stratton et al. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. *UIUC, Tech. Rep. IMPACT-12-01*, 2012.
- [19] Y. Wen, Z. Wang, and M. O’Boyle. Smart Multi-Task Scheduling for OpenCL Programs on CPU/GPU Heterogeneous Platforms. In *Intl. Conference on High Performance Computing*, pages 1–10, 2014.