Task Scheduling Frameworks for Heterogeneous Computing Toward Exascale

Suhelah Sandokji[1], Fathy Eassa[2]
Faculty of Computing and Information Technology, KAU
Jeddah ,Saudi Arabia

***Abstract*—The race for Exascale Computing has naturally led computer architecture to transit from the multicore era and into the heterogeneous era. Many systems are shipped with integrated CPUs and graphics processing units (GPUs). Moreover, various applications need to utilize both CPUs and GPUs executive resources, as many of their unique features prove the significant importance and strengths of using each one of the process units PUs. Several research studies consider partitioning the applications, scheduling their execution and allocating them onto the PUs resources. They investigate the important role of optimization and tackle intelligently scheduled tasks on the combination of CPU/GPU architecture CPUs and GPUs cores in achieving the peace of performance and power consumption of Exascale Computing. In this paper, the evolution of heterogeneous computing architectures, the approaches, and challenges toward achieving Exascale Computing, and the various algorithms and techniques used to partition and scheduling tasks are all reviewed. The existing frameworks and runtime systems utilized to optimize performance and improve energy efficiency in desecrates and fused chips in order to attain the objectives of Exascale Computing will also be reviewed.**

Keywords*—*Exascale computing; heterogenous computing; task scheduler framework

I. INTRODUCTION

High-performance computing is the pillar for modern science. Researchers with great computing powers can make an amazing scientific discovery from climate science to combustion science, business analytics for making a good decision, big data analytics, and many others. Therefore, researchers are looking forward to the next generation of high-performance computing, i.e. "Exascale Computing". Exascale Computing achieves 10^{18} flops on real applications constraints to be within the power of 20 megawatts. Therefore, in designing both the hardware and software architecture systems, the challenge is managing the tradeoff between the performance speed-up and energy consumption. One of the most critical aspects in this management between the software and hardware is related to mapping software application to the best-fit hardware resources. Mapping refers to partitioning the application under execution into tasks, prioritizing these tasks, or scheduling them in lists to be allocated on the processors, reducing the execution results after which the user receives the computation results. Arranging this mapping using an efficient optimum algorithm that decreases the limit range of energy consumed and raises performance is considered an NP-problem. There have been a significant number of research studies that look into achieving the optimum solution to the scheduling problem. In this survey, the scheduling approaches and the research existing in the heterogeneous processors are reviewed.

A. Survey Scope and Limitations

The survey is focused on the scheduling framework that plans the tasks on the combination of CPU and general purpose graphical processing unit GPU in both types of desecrate system and on-chip system. As it is impractical to review all the aspects of the published work that are related to the task scheduling frameworks, we consider here some limitations in order to highlight the paper's scope. We focus on the CPUs/GPUs heterogenous architecture. We don't review the heterogenous architecture that are built based on other types of processors or accelerators, as heterogeneous computing may consist of for instance Field Programming Array FPGA cores and CPUs cores. Also, we didn't discuss the scheduler frameworks that consider only multi GPUs nor single GPU. The paper focuses on the task scheduler framework, language libraries, and framework level techniques. The paper considers the software level techniques, therefore, no circuit/ device/ microarchitectural level techniques are reviewed. Our paper aim is to highlight the key research ideas and the main concepts that provide researchers with the insight required to inspire future improvement in the next generation of the high-performance computing "Exascale Computing".

The remainder of the paper is organized as follows; in the next section, some principles regarding the roadmap for exascale capabilities are highlighted. This section also argues if heterogeneous computing is able to achieve an exascale capability. The following section explains the heterogeneous computing principles, where first heterogeneous computing and its types are defined extensively. Then the parallel computing with different types are defined and how it can achieve optimum heterogeneous computing is discussed. Afterward, the evolution of several components of processors hardware, such as the increase in transistor numbers, core numbers, registers file, new memory types, and new speed interconnections bus is described. The study also highlights the challenges that may limit software improvement of Exascale Computing. Furthermore, the single chip and multiple chips accompanied with accelerator GPUs as well as the algorithms used in task scheduling frameworks and the research of task scheduling framework in two aspects; performance improvement and energy efficiency all are also reviewed.

II. The Roadmap for Exascale

Figure 1 illustrates the roadmap for Exascale Computing. In 2013, the biggest supercomputers such as Titan in the USA or Tsubame KFC Tokyo Tech were 2.5GFlops/W and 4.5GFlops/W, respectively [1,2,3] where heterogeneous computing is used. Both also use K20 GPU, but Tsubame KFC does several improvements as opposed to Titan, one of which is changing the ratio CPU/ GPU, as energy consumption mostly goes more to the GPU and less to the CPU. Thus, one way of thinking to reach exascale is to improve 20PFlops, 10Wand 10^7threads so as by 2023, it will have been duplicated 50times to get 1000GFlops besides only duplicating twice the power consumption. Hence, power efficiency must go up to 25 times of the 2013 range. This efficiency is derived from process technology, better h/w, and s/w architecture and circuits, in addition to utilizing, parallelize and improving the thread from 10^7 to 10^{10}.[1].

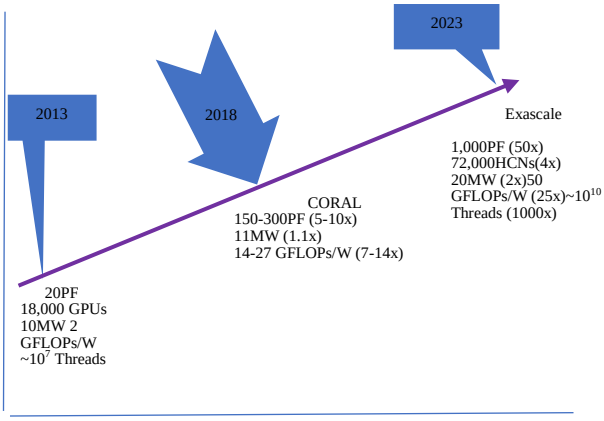

Fig. 1. The Roadmap for Exascale.

The matter that motivates researchers to leverage the heterogeneous Pus (multi CPU cores combined with any many-core accelerator such as GPUs or GFPA) collaboration to achieve high-performance computing. This way, we can benefit from the advantages of each and leverage the intelligent combination of both so as to achieve exascale performance and power consumption.

III. The Heterogeneous Computing, Principals, and Types

Nowadays, instead of CPU versus GPU debates, researchers, programmers, and computer architects are exploring PUs paradigm to find different approaches for computing and programming on efficient algorithms. This paradigm, which is known as Heterogeneous Computing (HC), refers to the utilization of the strength of diverse processing cores to maximize performance. The combinations vary from the CPUs with graphics processing units (GPUs), (see Fig4) to field programmable gate arrays (FPGAs) or both or Cell Broadband Engine Architecture (CBEA), Heterogeneous Computing (HC). Strengthening the combination architectures and accomplishing load balancing are the main targets to tone with the needs of each application, by refraining from idle time for both Processing Units (PUs). Some processors achieved more heterogeneous integration by fabricating them on the same chip as a system on chip (SoC), Such as AMD Llano [3], Intel Sandy Bridge, and Ivy Bridge [4]. One example of heterogeneous Multi-Processor System-on-Chip (MPSOC) is the Samsung Exynos[5]. The Samsung Exynos architecture consists of 4Arm Cortex-A7 (little), 4Arm Cortex-A15 (big) and ArmMali-T628GPU cores. As modern embedded systems become gradually based on MPSoC, developers are motivated to adapt algorithms and techniques that convey this hardware evolution.

A. From Parallel to Heterogeneous Computing Principles and Challenges

One of the key techniques in the HC is tuning the work scheduler to leverage the parallelism efficiently. The modern hardware of single node architecture has several parallelism layers by which the performance of our program can be optimized. Here, the different types of parallelism in a single node equipped with an accelerator such as GPU are mentioned. The type of the parallelism varies based on the type of connection between the unit processing and its types.(Figure 2 and 3 illustrates the types of parallelism. At the highest level is the Multi-chip parallelism, when there is more than one physical processor chip connected by a bus in the same computer. In this type, the resources and components, specifically the system memory, are shared. The communication between the cores is by the Peripheral Component Interconnect Express (PCIe) bus. The second level of parallelism is Multi-core on-chip parallelism, which is similar to the multi-chip parallelism, except that there is a single chip that combines the processor cores. In this type, the processor units share the resources that are a single chip, thus the communication is much better as when using the on-chip cache. This makes communications even less costly. When an accelerator such as GPU is connected To CPU cores on-chip, we refer to it as an Accelerated Processing Unit (APU) or heterogeneous Multi-Processor Systems-on-Chips (MPSoCs). This type is also called an integrated/fused system in contrast to the first type that is a conventional discrete system. The third type is Multi-context (thread) parallelism, where a single core is able to initialize multiple execution contexts and switch between them with reasonable or no overhead. In a multi-context system, a task can be executed on each context and in this case, there would be a separate hardware program counter for each execution context. When the processor unit is able to perform the order for one or more instructions per cycle, it then achieves the Instruction Parallelism level (ILP), which requires using multiple instruction units. HCS leverages several types of parallelisms and combines between these techniques in order to decrease the cycle per instruction and increase the efficiency of the utilization of resources.[6]

Following are several techniques used for HPC and examples for these approaches: we will start by mention the ILP techniques.

Hardware pipelining, this technique is applied when the processor unit gives one instruction order or more per cycle simultaneously in the pipeline. Next, the Vector parallelism, when there is an array of arithmetic units over which an order is duplicated. Also, Very long instruction words (VLIWs) technique used in particular architectures. [6]

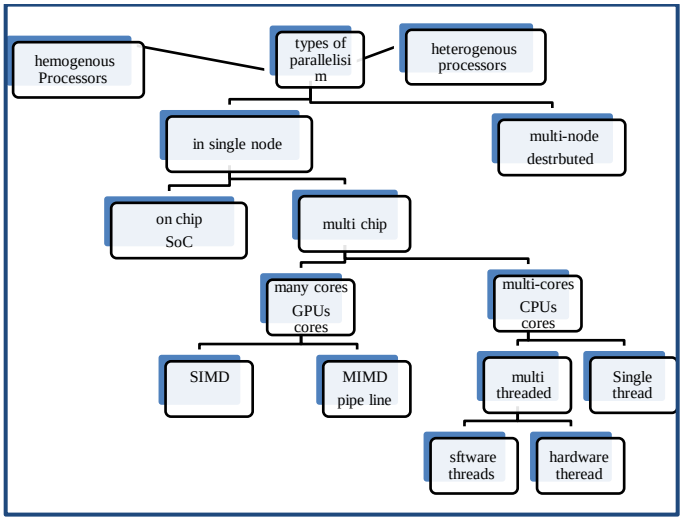

Fig. 2. Levels of Parallelism.

There are also some other techniques that can solve memory latency such as:

Hardware multi-threading :when the execution units are shared by set of execution contexts. If memory demands stall occur the CPU instantaneously switches between these contexts , declining the effect of latencies. This ought not to be confused with software threads, as the different execution contexts normally are stored in main memory.[6]

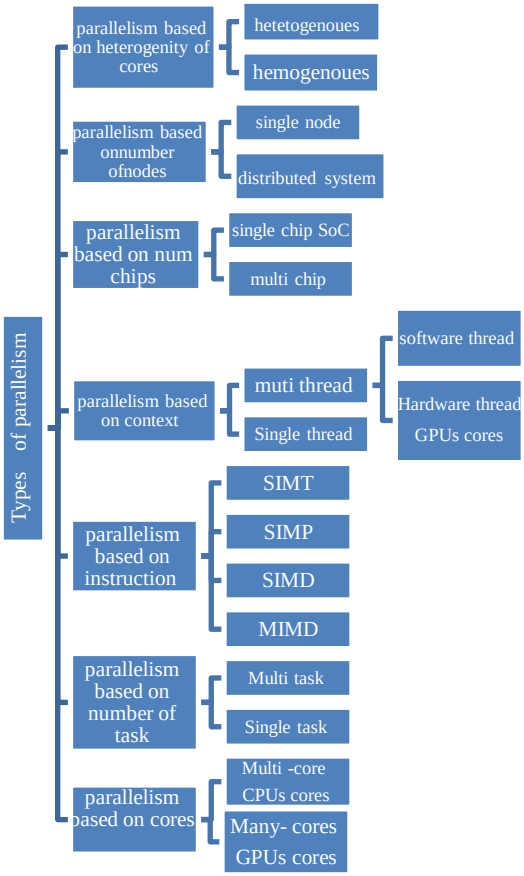

Fig. 3. Types of Parallelism.

Out-of-order execution: in some cases, latencies caused by data dependencies are minimized by reordering the instructions stream execution statically by the compiler. Otherwise, the penalty of this latency in that the conventional systems cause processor stalls.[6]

Such techniques are frequently combined, making program execution complex and tough to predict.

Data and task parallelism are different sorts of parallelism. Task-parallel methodology approximately views the problem as a set of tasks with clearly characterized communication patterns and dependencies. Pipelining could be a representative model. On the other hand, data-parallel methodology roughly views the problem as a set of operations carried out on clusters of data in a generally uniform fashion. The focus of this paper is on the combination of CPU/GPU architecture, CPUs and GPUs. The extremely diverse architectures and programming models of every type of the heterogeneous architecture present quite a few challenges in accomplishing such collaborative computing. Due to the interaction amid CPUs and GPUs in a heterogeneous system, performance optimization and energy efficiency depend on considering the characteristics of both the Pus. For this cause, usual techniques of CPU-only or GPU-only optimization might not work efficiently in a heterogeneous system. Hence, novel techniques are obligatory so as to realize the potential and opportunities of heterogeneous computing and shift towards the objectives of exascale performance.

IV. Prospects and Difficulties of Heterogeneous Computing Towards Exascale Computing

A. Evolution of Hardware Architecture of PUs

Table 1 reviews the hardware architecture development during the last few years. Some of the main parameters affecting the performance and the energy consumption are considered. We also take into account the transistor counts, number of cores, hardware or software manage caches, types of memory and bandwidth.

TABLE I. Hardware Evolution Recent 10 Years

parameter	Before	Now
Transistor Count	CPUs:1B transistors	Oracle SPARC M8 CPU >10Bon a chip[7].
		Stratix(FPGA)30B transistors.[7,8]
	GT200 GPU 1.4B transistors	GTX TITAN X GPU contains8B transistors [9].
Number Of Cores	GTX 280has 240 core	GTX TITAN=3072 cores Oracle Cranks up the Cores to 32 with Sparc M8 Chip.[10]
Managed Caches,	GPUs only software- managed caches, GT200 no L2	Large hardware -managed caches. Fermi GPU only had768KB LLC, the Kepler GPU had1536KB LLC, and the Maxwell GPU had2048KB LLC.[11]
3D Stacking	No	NVIDIA's Pascal GPU [11] Intel's Knights Landing [12].
Interconnect Bandwidth	The bottle neck in performance of CPUs and GPUs [13 and 14].	NVLink, offer 5to 12× bandwidth[11]

1) Examples of heterogeneous architecture computers: Here are examples of well-known heterogeneous computing architectures:

Xeon Phi:.[14] Knights Landing

The second-generation Phi.[14]

CBEA [15,16]

Nvidia GPU[11]

FPGA[8]

Finally, it can be said that there had been several approaches for achieving exascale capabilities through heterogeneous computing such as [17] it is evident —via these trends— that the never-ending evolution process of hardware architecture of both CPUs and GPUs is still ongoing.

2) Motivations for heterogeneous computing: Although utilizing GPU and FPGA as stand-alone devices appears promising, there is a number of compelling reasons for shifting towards a heterogeneous computing approach:

Each one of the AUs has a unique strength along its weakness aspects, (See fig.4). By combining AUs with different architecture, we aim to leverage the pros of each AU and overcome the cons. A modern multi-core CPUs usually own several tens of cores. These cores are caricaturized by multi-instruction and out of order issue cores. They also operate at high frequency. In addition, caches' size has increased in a way that eliminates most of the single thread execution latency caused by memory and cache miss late penalty.

Therefore, we consider CPUs suitable for latency-critical applications and memory intensive instructions. Contrast to GPU architecture, which is characterized by using a huge number of in-order cores, these cores use shared control, shared memory, and smaller cache size for each Stream multi-processor with lower frequency. Consequently, GPUs are appropriate for throughput-critical applications [18]. Therefore, it is reasonable to use a heterogeneous architectural system that consists of two or more types of cores, and schedule the application tasks between those cores; each task to the best fit or suitable type of execution unit. This way, we optimize the performance more than if we only use traditional CPU or GPU alone [19].

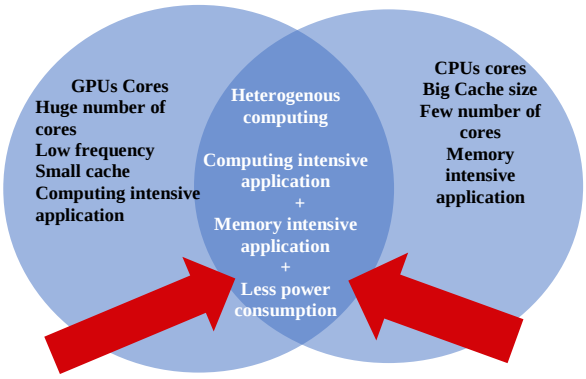

Fig. 4. Heterogenous Computing Advantages.

Mapping algorithm to best fit Pu's characteristic. When a user needs to schedule data transfers intensive applications, in some cases the branch difference or the time, uninterrupted execution are not allowed for GPU cores. On the other hand, performance is improved if CPUs are used. This does not only happen for diverse applications, but CPUs are also superior for diverse phases in single application. The matter that motivates to look at both sides in equality, both CPUs and GPUs. [20,21,22]. For example, it is eligible for developers to consider CPUs efficiency for processing short list queries contrast to GPUs that are efficient to process queries of long list. [23]

Exascale Computing is characterized by 10^{18} flops/s on real applications within the power of 20 megawatts. It assess requires using best fit scheduling and resource utilization improvement. Using GPUs alone or CPUs alone is considered a waste of resource as the average of utilization is low. For example, when allocating the task to GPU, the CPU starts initializing the kernel and keeps waiting idle for GPU computing results. [19]. Furthermore, when GPU is used for memory intensive tasks, memory bandwidth is considered the bottleneck which led to energy consumption and the utilization is even low [24,25]. Scheduling tasks intelligently on heterogeneous architecture enables the elimination of this problem [26,27]. Therefore, the June 2017 Green500 states the most energy efficient supercomputers, and CPU-GPU heterogeneous systems achieve the top 13 systems in the whole list [2].

CPUs have usually been used as a host for GPU in systems equipped with GPU to administer I/O and scheduling; nevertheless, due to the continuing innovations improvements of CPU performance even further, using its computation capabilities also has happened to be more appealing. Additionally, while numerous initial works report that GPUs supply up to 100× to 1000× speedup, other researchers claim that on applying careful optimizations on both CPUs and GPUs, CPUs may even outperform the performance of GPUs [28,29]. Due to this, different sums of load divisions to CPUs and GPUs can lead to very much dissimilar performance. These findings highlight the significance of utilizing the computational capabilities of CPUs as well.

V. Challenges Toward Achieving Exascale Performance and Energy through Heterogeneous Computing

There is a wide gap between the CPUs and the accelerators GPUs in different aspects such as performance, energy consumption, and efficiency. That gap is related to difference in architecture and programming model. Therefore, there are many considerations and challenge faces when scheduling tasks on such combination. Here, some issues are mentioned briefly:

HCS architecture that may achieve exascale features.

Achieving load balance on both CPU and GPU.

Memory bottleneck, memory bandwidth and size and memory contentions.

There is another concern related to scheduling algorithms such as:

Data partitioning and job partitioning in addition to accounting the data dependencies.

Static verses dynamic algorithms for scheduling and allocating task on heterogeneous cores

Overlapping data transfer and computation algorithms.

Underutilization of GPU multiprocessor and related resources.

Partitioning kernels contrast to partitioning devices.

Calculation systems power affectionately.

In the next section, some of these challenges are discussed and some of the research studies addressing them are mentioned.

VI. Techniques of Workload Scheduling

Tables 2 categorizes the techniques projected for dividing the workload among a CPUs and a GPUs at the stage of an algorithm or through program execution. Scheduling algorithms may classify based on when the scheduling is done, at the compile time or runtime, see Table 2. In the dynamic algorithm, the verdict about operating the subtasks or program phases or code portions on a particular PU is made at runtime. The subtasks that are implemented on a particular PU are already decided before program execution in the static algorithm, the mapping of subtasks to PUs is fixed. Here we review some of the researches that are based on each of the static and the dynamic techniques beside other schedulers that combine the both strategies.

TABLE II. Scheduler Type based on when Occur the Scheduling

Type of scheduling	**Examples of The researches**
Static scheduling	30,31,32,50,67,69,70,..
Dynamic scheduling	34,35,36,77,76,51,49,52,53,68,…
Combine static and dynamic	37,38,39,40,41,42,43,44,45,46,47,48,54,55,56,57,58 ,59,60,61,62,63,64,65,66,67,69,71,72,73,74,75

A. Static Strategy

Many Static algorithms have been used for partitioning the tasks to available cores [30]. Qilin scheduler framework uses static algorithm [31]. An adaptive mapping method on heterogeneous parallel processing platforms (HPPPs) is proposed; comprising of one CPU and one GPU. The reason behind Qilin is to make load adjusting on HPPPs, so it maps the divided workload to all processors concurring to their capabilities. First in the offline phase, Qilin tests execution times of a task with different problem (data) sizes, and then it formulates linear regression as the prediction model. Then, in the online phase with the new coming problem dimension, the ideal splitting proportion of workload of task on CPU and GPU is gotten by the model. Then a database recording all of the tasks sampled is created. Qilin investigates the database for the division proportion while dispersing workload on HPPPs. Luke et al. [31] suggest an effective method for workload partition. Although, the approach deals with one CPU and one GPU only, what possesses a limitation of computational workload distribution (CWD) especially on HPPPs consisting of multiple CPUs and GPUs is the number of heterogeneous processors. One more work related to CWD proposes a "waterfall energy consumption model" [32] for power concern. The authors implement a task mapping method, β-migration, on GPU. Tasks could be divided into CPU sub-task and GPU sub-task. In this method, CPU sub-task does not move parts to GPU since CPU sub-task is not proper for GPU. Whereas CPU sub-task and GPU sub-task do not work in a balanced style, the β proportion of GPU sub-task is relocated to CPU.

B. The Dynamic Strategy

The dynamic strategy in [33,34,35] shows a mechanism that disperses the workload to processors evenly in the initial state (e.g., the first iteration of a for-loop in a program), and after that collects the execution periods of all processors and re-distribute the workload based on the achieved respective performance measured by minimum code intrusion run-time. Such a strategy can be regarded as a light runtime profiling, so the most important matter of this strategy is how to measure each processor's performance with minimum overhead.

The dynamic strategy, if compared to the static one, cannot reach a precise extent of workload distribution at first for efficient load balancing. Furthermore, the dynamic approach requires data migrations among processors, thus causing communication overhead than a static approach. Therefore, there is a tradeoff between the dynamic and static strategies. However, each one of them is suitable for different circumstances, as will be explained in other cases.

There are many heterogeneous scheduling frameworks that are based on a combination of dynamic and static scheduling algorithms.

Some examples are as follows:

STAR PU

StarPU, is a runtime system capable of performing tasks scheduling over multicore machines qualified with GPU accelerators. StarPU employs a software virtual shared memory (VSM), which achieves a high-level programming interface, and data transfers between processing units is then automated. To enable tasks to be dynamically scheduled, it uses static and dynamic scheduling strategies such as heterogeneous early finish task HEFT in addition to work stealing algorithm in order to get considerable speedups and elevated efficiency over multicore machines with multiple accelerators. In addition, it evaluates the performance of these applications over clusters featuring multiple GPUs per node and MPI could be combined with it. StarPU is a C library that gives an API to explain the application data, capable of submitting tasks that are dispatched and executed transparently over the whole machine in an efficient way and asynchronously.[36,37,38]The main drawbacks regarding StarPU that it does not support independent loops and also

recursive parallel algorithms as it does not allow create tasks recursively.

Several researchers built scheduler frameworks on top of StarPU as to extend or improve it such as [39]. They investigate the applicability of throughput enhancement by the co-scheduling of poorly-scaling tasks on sub-partitions of the GPU. This is being done to elevate utilization efficiency. Scalability of GPUs was studied, and the incorporation of this insight to use the CUDA API to partition the GPU.

Sequential task-based programming model was studied by Emmanuel, et al. [40] to achieve efficiencies in multiple nodes scale similar to its successfully efficient on the environment scale of a single node that combined with accelerators supported with OpenMP standard. They extend StaPU with an inner-node data management layer to post communication automatically. They achieve a performance competing with both pure Message Passing Interface (MPI)-based ScaLAPACK Cholesky reference execution and the DPLASMA Cholesky code, which executes a different (non-sequential) task-based programming paradigm.

Common runtime of IBM OpenCL

It enhances the OpenCL programming encounter as it covers up the low details of data movement from the programmer synchronization and automatically runs multiple OpenCL platforms and duplicated components, such as contexts and memory objects. Moreover, it also controls event dependencies cross-queue scheduling.[41]

The GPU And Multi-core Aware (GAMA)

It is a framework to aid computational scientists in the development or porting of data-parallel applications to heterogeneous computing platforms CPU/GPU cores. GAMA is specially designed to efficiently execute irregular applications where it is hard to make the needed estimation to compute the input data sets.[42]. In addition to many others such as QUARK [43], PaRSEC[44], SuperMatrix[45], StarSs[46] and Kaapi[47].

VII. Workload Scheduling Purposes

Answering why a particular scheduling of tasks to PUs is carried out, the following works are classified based on two main criteria -see table3- :

Improving performance.

Improving energy consumption.

First, we survey the research that addresses improving performance.

C. Improving Performance

Scheduling tasks on processors could be induced by the PU characteristic/capability itself and/or the subtasks. In case subtasks are similar, the scheduling then is based on achieving best performance or efficient power consumption or avoiding memory contention or achieving load balance. On the other hand, if the subtasks are different, deciding if a subtask ought to be mapped is based on the computation of an intensive task mapped to GPUs and memory intensive task to CPU.

TABLE III. Classify the Researchs based on the Scheduling Purposes

Type of scheduler based on the purpose of scheduling	Examples of the researches
Improve Performance by leverage memory utilization	48,49
Improve performance by improve load balance and resource underutilization	30,31,35,37,40,45,52,53.
Improve performance or in heterogenous SoC embedded	54,5,56,57,58,59,60,61
Improve performance and energy consumption in heterogenous SoC on desktop	62,64,65,63
Improve energy efficiency	70,71.72.73,74,75,76,77

1) Scheduling to concern memory limitations: Some cases exhibit the following: a subtask cannot be mapped on a particular PU; when the memory footprint of a subtask exceeds the memory size of the GPU. For example, the GPGPUs huge data demands costly memory modules, such as GDDR, to sustain high data bandwidth. The high cost poses limitations on the total memory capacity on hand to GPGPUs, and the data needs to be transferred among the host CPUs and GPGPUs. However, the data transfer long latency has resulted in considerable performance overhead. To ease this matter, modern GPGPUs have actualized the non-blocking data transfer, which enables a GPGPU to carry out computing while the data is under transmission. In [48], it is proposed that a capacity aware scheduling algorithm exploits the non-blocking data transfer in modern GPGPUs. By effectively benefitting from the non-blocking transfers, experiments show an average of 24.01% performance improvement when compared to other approaches existing, which only consider memory capacity.

One of the most critical problems in HC is scheduling in the presence of memory contention. This occurs when processors have used concurrency multi CPU cores and many cores as GPU are competing on the available resources such as register files memory and interconnected network that cause the contention. In the presence of memory contention, entirely static scheduling is hard coding and likely fails to solve it [49]. Given that the environment of multiprogramming is changing dramatically, the resources availability and the workload programs behaviors are indeterministic. This has an effect on the right mapping and scheduling of the required tasks. An approach that can adjust the mapping decision concurring to the dynamic computing environment is what is needed, by taking into account the target program behavior. To resolve the memory contention, Luk et al in [31] incorporate offline profiling to decide the best partitioning between the CPU and GPU as Grewe and O'Boyle [50] apply machine learning techniques to forecast the ideal partitioning. Both approaches arrived at promising results but only under the assumption that no other programs are operating on the system. Ravi et al. [51] use a dynamic "task farm" method for task mapping where the task is partitioned into chunks of a fixed number, and one chunk is sent to each device. When a device is done

processing, it calls for a new chunk. However, this dynamic approach achieves poor performance in the presence of GPU contention. Another approach was used in [49] where they use runtime information and the code features of the program under execution to make the task partitioning prediction to achieve the best dynamic task scheduling. The good point in this dynamic scheduler is the low overhead and it avoids the penalty of long online searching.

2) Scheduling to prevent resource underutilization: In reality, the majority of GPU applications do not utilize all of the available components in a system efficiently, either entirely failing in using a component or using it to a limited fragment of its full potential. This underutilization can harm both energy efficiency and performance. Some research related the underutilization of GPU to the penalty of CPU/GPU communication and coordinate. Consequently, they reduce the CPU/GPU interaction and synchronization aiming to improve the performance improves the energy consumption. Boyer [52] studied the iterative algorithms that are implemented and found that it causes the underutilization of the GPU resources, since they produce a high overhead of CPU/GPU arrangements. Thus, he presented several strategies to be applied when implementing such algorithms. Applying these strategies would improve the performance and reduce the communication overhead. He also applied dynamic algorithms for allocation parallel kernels and efficiently utilized the available resources and achieved the load balance between the cores. Moreover in allocation kernels on GPU stream multi processors the kernels are usually executed sequentially, one kernel a time even in systems with multiple powerful devices.

Some other framework schedulers harnessing kernel partitioning to achieve the load balance on GPUs and improve the performance. One of the heterogeneous systems that used kernel partitioning is done by Ghose et al[53]. They present an in-depth analysis of control flow divergence of OpenCL kernels. Since branches have a significant impact on OperCL kernel performance, the author uses divergence as a guide to partition a kernel across CPU and GPU. A machine learning model is trained by using the amount of divergence in a program; then this model is used to predict unseen program's optimal partition. Splitting a kernel into parts and spreading these parts into distinct devices can be done statically or dynamically. But most of these research studies are not heterogeneous computing systems, and thus they are out of our scope.

3) Scheduling for achieving load balance: The load balancing across the resources in the system is the most important goal of the scheduler's framework which considers performance improvement. As load balancing enhances the overall system performance, there are various approaches for achieving this. Some approaches focus on the static algorithms and arrange the balancing plan during the compile time such as heterogeneous early finish time HEFT [30,31]. Other research studies use dynamic strategies such as work stealing or work share algorithms[35,37,40,45]. On the other hand, some research studies relate the solution for imbalance load to resource underutilization. Therefore, they consider improving resource utilization as mentioned in a previous point.

In the next section, the research studies that consider scheduling tasks on heterogeneous multiprocessor system-on-chip (MPSoC), aiming to improve the performance or energy consumption or both, are discussed.

D. Scheduling Tasks on Heterogeneous Multiprocessor System-on-chip (MPSoC)

Nowadays, MPSOC is considered a hot topic, where there have been many pieces of research introducing different approaches. [54,55, 56, 57, 58, 59, 60, 61]. The work mostly considered Samsung Exynos 5422 SoC which utilizes 4 big and/or 4 little cores that possess the same instruction set architecture ISAs [55, 57, 60]. Nonetheless, the majority of the effort was applied to the identical type of cores [58, 59, 60, 61]. In spite of that, there were a number of endeavors to parallelize both big and LITTLE cores [55], exploit cores are inapplicable with this approach having diverse ISAs such as CPU and GPU for the reason that they handle instructions in many ways. A number of studies were conducted to develop the MPSOC on desktop platforms [62, 63, 64,65]. Task scheduling and harmonization between CPU/GPU cores using such platforms cores require additional consideration. In [63], an algorithm to improve the throughput was proposed. The algorithm divides both the power budget and the workload between the CPU/GPU cores of an AMD Trinity single chip heterogeneous platform, using the same AMD platform by [64], but memory dispute occurs due to access of the same bank in different patterns by the CPU and GPU. In addition, in [62], partition considered the work and mapped them to threads between the CPU/GPU cores but that did not revolve around the energy FreeOCL [66]; a similar open source framework is used for the Arm CPU that acts as both the host processor and an OpenCL device. This provides concurrent use of CPU and GPU to carry out the application threads, but in [67,69], a static dividing is performed by using all the CPU and GPU cores. Researchers of [68] established that the influence of temperature-induced variability on circuit lifetime can be elevated due to stress and exceed over the value estimated bearing in mind the circuit average temperature. The researcher presents a simulation framework for the BTI degradation analysis of DVFS designs that considers thermal profiles under the Dynamic Thermal Management (DTM) system influence.

In [54] they divide the workload on CPU/GPU savings of the average temperature of the chip while keeping performance needs. A lower thermal behavior exhibits an enhanced long-term reliability (lifetime) of the SoC. Grasso et.al.[69] focus on analyzing the embedded GPU ARM Mali-T604 GPU. They used an OpenCL Full Profile support and investigated the utilization and optimization techniques that efficiently leverage the hardware resources. They implemented and evaluated the frame work and concluded that ARM Mali GPU Compute Architecture was able to achieve the performance and energy consumption that allows them to be a good candidate for future HPC systems.

E. Energy-Aware Scheduling Frameworks

There have been several types of research that optimize scheduling algorithms aiming to reduce the energy consumption [70,71,72,73,74,75,76,77,78]. Some use a static algorithm such as a generic algorithm. In [76] they introduce new chromosome structure to implement a generic algorithm that adapts to schedule scalable and various tasks in efficient energy consumption. Instead of individual tasks and machines, it is based upon groups of alike tasks and machines. This fresh arrangement is highly scalable and divided into three phases. The first two phases of the algorithm exhibit near-constant execution time notwithstanding of the number of tasks to be listed or the figures of machines in the system. Only the final phase of the algorithm is dependent upon the number of tasks and machines; this could be relieved by executing the phase for only a subset of the solutions from the last population.

Some others use algorithms that schedule the tasks in a way that controls the system energy consumption at a certain point of time. Other research studies aim to schedule tasks while keeping the system energy at a limit given range and prevent the system energy overloaded. This point of keeping a system in a limit range of energy is critical to achieving the exascale systems constraints. One of those uprising techniques is "Power capping", which is used to control power consumption in a data center in a certain period of time. In [77] the researchers study the impact of power variation of scheduling multi programming concurrently. They present an efficient algorithm for power capping.

VIII. Conclusion

Recently, the use of different processors simultaneously, such as CPU cores combined with different accelerators GPUs cores, for achieving efficiency in performance and energy control of Exascale Computing, has been intensively researched. Heterogeneous computing emerges as a promise paradigm. Different approaches were proposed to leverage heterogeneous computing toward Exascale Computing. Several of these approaches have been reviewed in this paper. Also, the evolution of hardware and the various approaches proposed for partitioning, scheduling and allocating the workload on the heterogeneous architecture, desecrate systems and fused systems or heterogeneous SoC were discussed as well. These research studies are done with two main goals: performance improvement and energy efficiency. In conclusion, we confirm the need for accommodating software development along with the quick evolution of the hardware. In our review, a huge gap has been found between the algorithms improvements to utilize the available hardware resources and the evolution in hardware. Tuning the scheduling techniques and the combining of algorithms by exploring other software techniques to reduce the underutilization of the accelerators resources -specifically GPUs resources is bound to decrease this gap.

References

[1] https://youtu.be/IIzjMr4f-8U

[2] https://www.top500.org/green500/lists/2017/11/

[3] A. Branover, D. Foley, M Steinman. AMD fusion APU: Llano. IEEE Micro, 32, 2(2012), 28–37.

[4] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T.Kurts.. "A fully integrated multi- CPU, GPU and memory controller 32nm processor". In Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'11). 264–266. Damaraju et al. 2012

[5] Exynos 5 Octa (5422). (2016). www.samsung.com/exynos/

[6] A. R. Brodtkorb, C. Dyken, T.R. Hagen, J. M. Hjelmervik, and O.O. Storaasli, "State-of-the-art in Heterogeneous Computing," Scientific Programming, vol. 18, no. 1, pp. 1-33, 2010. doi:10.3233/SPR-2009-0296

[7] https://www.nextplatform.com/2017/09/18/m8-last-hurrah-oracle-sparc/

[8] "Altera's 30 billion transistor FPGA". Gazettabyte. 28 June 2015. Retrieved 24Jan 2017. http://www.gazettabyte.com/home/2015/6/28/alteras-30-billion-transistor-fpga.html

[9] http://www.enterprisetech.com/2018/03/13/oracle-cranks-cores-32-sparc-m8-chip/.

[10] https://www.techpowerup.com/gpudb/2632/geforce-gtx-titan-x

[11] https://www.nvidia.com/en-us/data-center/pascal-gpu-architecture/

[12] https://ark.intel.com/products/codename/48999/Knights-Landing

[13] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang,S. Pakin and J. Sancho, Entering the petaflop era: The architecture and performance of Roadrunner, in: Supercomputing, November 2008, IEEE Press, Piscataway, NJ, USA, 2008, pp. 1–11

[14] https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html

[15] http://searchdatacenter.techtarget.com/definition/IBM-Roadrunner

[16] https://gizmodo.com/5090737/ibm-roadrunner-tops-cray-as-the-official-worlds-fastest-supercomputer

[17] Schulte, M.J., Ignatowski, M., Loh, G.H., Beckmann, B.M., Brantley, W.C., Gurumurthi, S., Jayasena, N., Paul, I., Reinhardt, S.K. and Rodgers, G., 2015. Achieving exascale capabilities through heterogeneous computing. IEEE Micro, 35(4), pp.26-36.S.

[18] Mittal. A survey of techniques for managing and leveraging caches in GPUs. Journal of Circuits, Systems, and Computers (JCSC) 23, 8 (2014).

[19] Mittal S, Vetter JS. A survey of CPU-GPU heterogeneous computing techniques. ACM Computing Surveys (CSUR). 2015 Jul 21;47(4):69

[20] H. Hong; Z. H.B. Hong. Dynamically tuned push-relabel algorithm for the maximum flow problem on CPU-GPU-hybrid platforms. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10). 1–10.

[21] A. Nere, S. Franey, A. Hashmi, and M. Lipasti. 2012. Simulating cortical networks on heterogeneous multi-GPU systems. J. Parallel and Distrib. Comput. 43, 7 (July 2012), 953–971.

[22] J. Shen, A.Varbanescu, H. Sips, M. Arntzen, and D. G. Simons. Glinda: A frameworkfor accelerating imbalanced applications on heterogeneous platforms. In Proceedings of the ACM International Conference on Computing Frontiers. ACM, New York, NY, Article 14. 2013

[23] S. Ding, J. He, H. Yan, and T. Suel.. Using graphics processors for high performance IR query processing. In Proceedings of the 18th International Conference on World Wide Web (WWW'09)P.421–430 ,2009

[24] M. Daga, A. M. Aji, and W. Feng. On the efficacy of a fused CPU+ GPU processor (orAPU) for parallel computing. In Symposium on Application Accelerators inHigh-Performance Computing(SAAHPC). IEEE, 141–149. 2011.

[25] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, Philip C. Roth, and J. S. Vetter. The tradeoffs of fused memory hierarchies in heterogeneous computing architectures. In Proceedings of the 9th Conference on Computing Frontiers. 103–112. 2012.

[26] I. Gelado, J. E. Stone, J. Cabezas, S. Patel, N. Navarro, and W. W. Hwu. An asymmetric distributed shared memory model for heterogeneous parallel systems. In ACM SIGARCH Computer Architecture News, 38 1 (March 2010), 347–358. 2010.

[27] Q. Hu, N. A. Gumerov, and R. Duraiswami. Scalable fast multipole methods on distributed heterogeneous architectures. In Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, New York, NY, Article 36. . 2011

[28] J. Gummaraju, L. Morichetti, M. Houston, B. Sander, B. R. Gaster, and B. Zheng. 2010. Twin peaks: A software platform for heterogeneous computing on general-purpose and graphics processors. In Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10). ACM, New York, NY, 205–216.

[29] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey.. Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. In Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10). ACM, New York, NY,P 451–460. 2010

[30] D. Grewe and Michael F. P. O'Boyle.. A static task partitioning approach for heterogeneous systems using OpenCL. In Proceedings of the 20th International Conference on Compiler Construction:Part of the Joint European Conferences on Theory and Practice of Software. Springer, Berlin, P 286–305. 2011

[31] C.-K. Luk et al., "Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping," in the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-42, pp. 45–55, 2009.

[32] W. Liu et al., "A Waterfall Model to Achieve Energy Efficient Tasks Mapping for Large Scale GPU Clusters," in the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, IPDPSW, pp. 82–92, 2011.

[33] A.P.D. Binotto et al., "Towards dynamic reconfigurable load-balancing for hybrid desktop platforms," in the IEEE International Symposium on Parallel & Distributed Processing Workshops and Phd Forum, IPDPSW, pp. 1–4, 2010.

[34] I. Galindo et al., "Dynamic load balancing on dedicated heterogeneous systems," In Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, pp. 64–74, 2008.

[35] C. Augonnet, J. Clet-Ortega, S. Thibault and R. Namyst, "Data-Aware Task Scheduling on Multi-accelerator Based Platforms," 2010 IEEE 16th International Conference on Parallel and Distributed Systems, Shanghai, 2010, pp. 291-298.doi: 10.1109/ICPADS.2010.129

[36] Augonnet Cdric et al., "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures", Concurrency and Computation: Practice and Experience, vol. 23.2, pp. 187-198, 2011.

[37] C. Augonnet et al. "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures". In: Concurrency and Computation: Practice and Experience 23.2 (2011), pp. 187–198.

[38] Augonnet Cdric et al., "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures", Concurrency and Computation: Practice and Experience, vol. 23.2, pp. 187-198, 2011.

[39] Janzén, J., Black-Schaffer, D., & Hugo, A. (2016, October). Partitioning GPUs for Improved Scalability. In Computer Architecture and High Performance Computing (SBAC-PAD), 2016 28th International Symposium on (pp. 42-49). IEEE.

[40] Agullo, Emmanuel, et al. "Achieving high performance on supercomputers with a sequential task-based programming model." IEEE Transactions on Parallel and Distributed Systems (2017).

[41] Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering, 12(3), 66-73.

[42] João Barbosa. GAMA framework: Hardware Aware Scheduling in Heterogeneous Environments. Tech. rep. Computer Science Dept., University of Texas at Austin, Sept.2012.7

[43] YarKhan, J. Kurzak, J. Dongarra, "QUARK UsersGuide: QUeueing And Runtime for Kernels", UTK ICL, 2011.

[44] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Hrault, J. Dongarra, "PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability", Computing in Science and Engineering, vol. 15, no. 6, pp. 3645, Nov. 2013.

[45] E. Chan, F. G. Van Zee, P. Bientinesi, E. S. Quintana-Orti, G. Quintana-Orti, R. Van de Geijn, "SuperMatrix: a multithreaded runtime scheduling system for algorithms-byblocks", 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, pp. 123132, 2008.

[46] J. Planas, R. M. Badia, E. Ayguad, J. Labarta, "Hierarchical task-based programming with StarSs", International Journal of High Performance Computing Applications, vol. 23, no. 3, pp. 284299, 2009.

[47] T. Gautier, X. Besseron, L. Pigeon, "Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi-processors", 2007 International Workshop on Parallel Symbolic Computation ser. PASCO 07, pp. 1523, 2007.

[48] H. W. Liu, H. K. Kuo, K. T. Chen and B. C. C. Lai, "Memory capacity aware non-blocking data transfer on GPGPU," SiPS 2013 Proceedings, Taipei City, 2013, pp. 395-400.doi: 10.1109/SiPS.2013.6674539

[49] Grewe D., Wang Z., O'Boyle M.F.P. (2014) OpenCL Task Partitioning in the Presence of GPU Contention. In: Cașcaval C., Montesinos P. (eds) Languages and Compilers for Parallel Computing. LCPC 2013. Lecture Notes in Computer Science, vol 8664. Springer, Cham

[50] Dominik Grewe and Michael F.P. O'Boyle. A static task partitioning approach for heterogeneous systems using opencl. In CC, 2011.

[51] 40]Vignesh T. Ravi, Wenjing Ma, David Chiu, and Gagan Agrawal. Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. In ICS, 2010.

[52] Boyer, M. (2013). Improving Resource Utilization in Heterogeneous CPU-GPU Systems (Doctoral dissertation, Ph. D. thesis, University of Virginia, Virginia).

[53] Ghose, A., Dey, S., Mitra, P., & Chaudhuri, M. (2016, February). Divergence aware automated partitioning of OpenCL workloads. In Proceedings of the 9th India Software Engineering Conference (pp. 131-135). ACM.

[54] Weber Wachter, Eduardo, Merrett, Geoff V., Singh, Amit and Al-Hashimi, Bashir (2017) Reliable mapping and partitioning of performance-constrained OpenCL Applications on CPU-GPU MPSoCs At 15th IEEE/ACM Symposium on Embedded Systems for Real-Time Multimedia, Seoul, Korea, Republic of. 15 - 20 Oct 2017.

[55] Ali Aalsaud, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang, and Alex Yakovlev.2016. Power–Aware Performance Adaptation of Concurrent Applications inHeterogeneous Many-Core Systems. In Proceedings of the 2016 International Symposium on Low Power Electronics and Design (ISLPED '16). ACM, New York NY, USA, 368–373. https://doi.org/10.1145/2934583.2934612

[56] Karunakar Reddy Basireddy, Amit Singh, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2017. ITMD: run-time management of concurrent multi-threadedapplications on heterogeneous multi-cores. In Conference on Design, Automationand Test in Europe 2017 (DATE'17). https://eprints.soton.ac.uk/406291/

[57] Kiran Chandramohan and Michael F.P. O'Boyle. 2014. Partitioning Data-paralle Programs for Heterogeneous MPSoCs: Time and Energy Design Space Exploration.In Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages,Compilers and Tools for Embedded Systems (LCTES '14). ACM, New York, NY, USA,73–82. https://doi.org/10.1145/2597809.2597822

[58] B. Donyanavard, T. MAijck, S. Sarma, and N. Dutt. 2016. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).1–10.

[59] Gangwon Jo, Won Jong Jeon, Wookeun Jung, Gordon Taft, and Jaejin Lee. 2014.OpenCL Framework for ARM Processors with NEON Support. In Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing (WPMVP '14). ACM, New York, NY, USA, 33–40. https://doi.org/10.1145/2568058.2568064

[60] J. Ma, G. Yan, Y. Han, and X. Li. 2016. An Analytical Framework for Estimating Scale-Out and Scale-Up Power Efficiency of Heterogeneous Manycores. IEEE Trans. Comput. 65, 2 (Feb 2016), 367–381. https://doi.org/10.1109/TC.2015.2419655

[61] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini. 2016. Workload-aware power optimization strategy for asymmetric multiprocessors. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE). 531–534.

[62] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. 2012. Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE). In Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12).

IEEE Computer Society, Washington, DC, USA,213–224. http://dl.acm.org/citation.cfm?id=2337159.2337184

[63] Indrani Paul, Vignesh Ravi, Srilatha Manne, Manish Arora, and Sudhakar Yalamanchili. 2013. Coordinated Energy Management in Heterogeneous Processors.In Proceedings of the International Conference on High Performance Computing,Networking, Storage and Analysis (SC '13). ACM, New York, NY, USA, Article 59,12 pages. https://doi.org/10.1145/2503210.2503227

[64] Hao Wang, Vijay Sathish, Ripudaman Singh, Michael J. Schulte, and Nam Sung Kim. 2012. Workload and Power Budget Partitioning for Single-chip Heterogeneous Processors. In Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12). ACM, New York, NY, USA,401–410. https://doi.org/10.1145/2370816.2370873

[65] Hao Wang, Ripudaman Singh, Michael J. Schulte, and Nam Sung Kim. 2014.Memory Scheduling Towards High-throughput Cooperative Heterogeneous Computing. In Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT '14). ACM, New York, NY, USA, 331–342. https://doi.org/10.1145/2628071.2628096

[66] FreeOCL: Multi-platform implementation of OpenCL 1.2 targeting CPUs.(2017). Retrieved 2017 from https://github.com/zuzuf/freeocl

[67] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra. 2015. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In 2015 33rd IEEE International Conference on Computer Design (ICCD). 208–215. https://doi.org/10.1109/ICCD.2015.7357105

[68] H. Chahal, V. Tenentes, D. Rossi, and B. M. Al-Hashimi. 2016. BTI aware therma management for reliable DVFS designs. In 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). 1–6. https://doi.org/10.1109/DFT.2016.7684059

[69] I. Grasso, P. Radojkovic, N. Rajovic, I. Gelado, and A. Ramirez. 2014. Energy Efficient HPC on Embedded SoCs: Optimization Techniques for Mali GPU. In 2014IEEE 28th International Parallel and Distributed Processing Symposium. 123–132. https://doi.org/10.1109/IPDPS.2014.24

[70] M. Oxley, S. Pasricha, H. J. Siegel, A. A. Maciejewski, J. Apodaca,D. Young, L. Briceno, J. Smite, S. Bahirat, B. Khemka, A. Ramirez,and Y. Zou, "Makespan and energy robust stochastic static resource allocation of a bag-of-tasks to a heterogeneous computing system," IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 10, pp.2791–2805, Oct 2015.

[71] M. Halappanavar, M. Schram, L. de la Torre, K. Barker, N. Tallent, and D. Kerbyson, "Towards efficient scheduling of data intensive high energy physics workflows (works '15)," in 10th workshop on Workflows in Support of Large-Scale Science, Nov 2015, pp. 1–9.

[72] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole,"An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environments," in 22nd Heterogeneity in Computing Workshop (HCW 2013), in the proceedings of the IPDPS 2013 Workshops & PhD Forum (IPDPSW), May 2013.

[73] K. M. Tarplee, R. Friese, A. A. Maciejewski, H. J. Siegel, and E. Chong,"Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques," IEEE Transactions on Parallel and Distributed Systems, vol. 26, 2015.

[74] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Scalable linear programming based resource allocation for makespan minimization in heterogeneous computing systems," Journal of Parallel and Distributed Computing, vol. 84, pp. 76–86, 2015.

[75] Friese, Ryan D. "Efficient genetic algorithm encoding for large-scale multi-objective resource allocation." In Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International, pp. 1360-1369. IEEE, 2016.

[76] Feng, Y., Li, G., & Sethi, S. P. (2018). A three-layer chromosome genetic algorithm for multi-cell scheduling with flexible routes and machine sharing. International Journal of Production Economics, 196, 269-283.

[77] Shoukourian H, Wilde T, Auweter A, Bode A. Power variation aware configuration adviser for scalable hpc schedulers. InHigh Performance Computing & Simulation (HPCS), 2015 International Conference on 2015 Jul 20 (pp. 71-79). IEEE.

[78] S. Mittal and J. S. Vetter. A survey of methods for analyzing and improving GPU energy efficiency. ACM Computing Surveys 47, 2, Article 19 (2015).