A Study of Recent Contributions on Performance and Simulation Techniques For Accelerator Devices

Hussein Ajam, Michael Opoku Agyeman
Department of Computing and Immersive Technologies
The University of Northampton, United Kingdom
Michael.OpokuAgyeman@northampton.ac.uk

Abstract— **High performance computing platform is moving from homogeneous individual unites to heterogeneous systems. Where each unit is a combination of homogeneous cores and accelerator devices. Accelerator such as GPUs, FPGAs, DSPs, these devices usually designed for the specific and intensive type of computing tasks. The presence of these devices have created fresh and attractive development platforms for developers and designers, brand new performance analysis frameworks and optimization tools. This is the cutting edge in the performance of some accelerator devices like GPUs and Intel's Xeon Phi. We outline some of the existing heterogeneous systems and their development frameworks. The core of this study is a review of performance modeling of these devices. In this paper, we address the emerging issues that affect the performance of these devices and associated techniques employed for simulation and evaluation.**

Keywords—; GPU; FPGA; DSP; GPGPU.

I. Introduction

Accelerators devices are a combination of hardware pieces usually forming mini computers, but they instead designed specifically to perform specific task or subtask effectively. These devices are provided with a Central Processing Unit (CPU), which responsible for processing instructions of software fragments and manipulate them. Since these devices are specifically designed for specific tasks, they tend to have different CPUs architectures, different number of cores, and endless combinations of these cores, diverse instruction sets and various memory hierarchies. Each piece of these devices is designed based on the predefined tasks of a specific board. Since these devices are designed to focus on specific sort of applications, sometimes heterogeneous devices consist different set and types of cores [1-6], and that add more challenges in order to measure performance or find the best suitable simulation technique. High performance computing (HPC) community started the journey of these accelerators when they used Graphics Processing Units (GPUs) as accelerators for general purpose computations [7] and that what derived the term of General Purpose Computing on GPUs (GPGPU) [8]. The main reason behind them was to support image processing and manipulations.

Measuring the performance of these verity of accelerators is a critical task, since they are diverse and been designed for completely different purposes, customizable devices [1][2]. Accurately scale of performance requires extensive programs writing just to achieve primary implementation. Unfortunately, there is no outstanding tool or model that can be considered as the reference instrument for performance prediction and tuning [9], building specific hardware and inject them with software to cover a single family of accelerators is not an efficient way, especially with the verity of factors that can influence devices performance. The biggest challenge is comparing and measure the performance of processors and accelerators attached to it simultaneously. The complexity of that task gets rapidly higher when we want to reach the optimization level of a device or trying to reach the aggregated performance of their processors, not to mention the parallel systems with thousands of hybrid nodes which adds increasingly difficulties to this task. In this paper, our main objective is reviewing different literatures about precisely to compile, organize and analyze the performance of accelerator-based computing. The paper is organized as follows, providing some background knowledge on some accelerator-based computing hardware's, some of their development tools and outline their modeling methodologies. Then we move on to review and compare the performance, power consumption and finally their simulation techniques. The last section will cover the conclusion and discussion about our own views and findings.

II. Background

Since we are discussing different factors and issues regarding HPC devices, we are providing background information about some different terms and tools to use throughout the rest of this paper. We used the word device many thimes in this papper but we do not deticate to any phisical aspects of these device in our research, our focus is about the processors and their different cores in adition to GPUs and how they integrate to each other and eventually find ways to measure their performance based on CPUs and accelerator attached to it.

A. Accelerators and Heterogeneous Architectures

We are focusing on Heterogenous Architecture in this paper, Heterogenous regarding architectures with multi-cores, each core has different characteristics, but they are designed to work together consistently. Accelerators can be classified into: GPUs and many-cores such as Intel's Xeon Phi coprocessor [10]. Other approaches like Field Programmable Gate Arrays (FPGAs) and Digital Signal Processors (DSPs) [11]. GPUs are programmable computational accelerators that aim to accelerate a wide range of parallel applications [12], composed of a set of processing cores and a memory hierarchy. A global memory space is accessible to all the cores in the GPU [13]. GPUs were first designed as high compute density hardware, they had fixed processors' instructions sets that can chive fixed number of instructions. Over the time, they have been improved rapidly and gained increasingly general-purpose capabilities such as support for flexible control flow and random memory accesses, the raw computing capacity of a GPU has increased greatly and exceeds the capacity of general-purpose CPU [12]. The Intel Xeon Phi accelerator is the second-generation product of these family, designed based on the concept of Many Integrated Core (MIC) architecture which basically means a combination of many CPU cores grouped over a single chip [13]. And the memory space will be shared between the GPUs and CPUs which reduced the expenses of the older accelerators where they used to use independence memory space each [11].

B. Development Tools for Accelerators

Accelerators applications are usually written using software that built and provided by the Accelerators manufacture companies, such as Quartus by Intel Altera for FPGAs accelerators which are also designed by Intel Altera [14]. These types of software usually consist of manufacturer-specific tool chains or can be implementations of standard APIs [21] which may provide a space of portability for code between different devices. However, standards APIs applications will not provide accurate performance measurements. As mentioned earlier there is no such ultimate stander tools but instead fine-tuning and specific device application is the way to fully explode capabilities of an accelerator. Another way to program accelerators is using frameworks:

1. OpenCL [15] which is a standardized, vendor-neutral framework, used to program almost all different accelerators, defining a hardware model and an API.
2. CUDA [16] This one was built specifically for NVIDIA GPUs, it still implements similar concepts from OpenCL, but uses slightly different terminologies, this allows developers to use GPU in general purpose processing.
3. Others such as OpenMP [13] and MPI [15].

C. Tools and Techniques for Performance Modeling

We can simply use debuggers and the devices manuals sheets to port and tune applications into an accelerator device. However, this ease does not let us to estimate or predict the performance capability of that device, especially with parallel programing and multi code units. We can Imagine that the same results application can be written in endless ways, and each way will be performed under certain circumstances on the different cores of a device which make prediction even more complex. We focus in this research on models of parallel applications executed on heterogeneous devices, that can be used to help to predict their combined performance over these devices. Moreover, efficiently testing application code process itself has gained enormous complexity, before applying any performance modeling application first and most important to make sure that the used code is concurrency bugs free. That requires a sequence of steps from a scheduler to manifest [17]. As schedulers are nondeterministic, both detecting and reproducing these faults are hard. Traditionally, concurrent programs are heavily stress-tested the application is run many times hoping for the right set of decisions from the scheduler that unearth latent bugs [17].In this section, we describe methods to characterize devices and applications in order to estimates performance models of the outlined devices.

1. Characterizing Devices

Two types of characteristics can be collected for any accelerator device:

- **a. Fixed characteristics:** Can be gathered from the devices manufactures manuals, data-sheets, and manufactures APIs which request information from its devices automatically and fill out a list of data obtained to get analysis in the API and passed to different processing point and get feedback.
- **b. Dynamic characteristics:** Obtained from hardware resources, performance counters [11].

2. Characterizing Applications

To get applications characteristics, we should describe the program into a matrix of characteristics, and eventually feed the performance module with this matrix, the accuracy of that module will be down to the quality of generated matrix. Different methods can generate application matrixes [11] like Analysis of the source code, analysis of an intermediate representation (IR) and disassembling the final binary files.

3. Modeling Methodology

Performance models are designed to list the behavior of an application on a device. Performance models should generate reports of performance from input applications ported into a specific device. Some researchers [11] identify the modeling performance of parallel systems into three main approaches: **Analytical modeling** [18] form a set of equations represent most of system characteristics in mathematical way and compare results of different sets' equations. **Machine Learning** [19] imply artificial intelligence in the performance modeling task by machinery studying the relations and behavioral of systems and eventify assemble some new valuable information. **Simulation** [18] tools to represent models behavioral and evaluate performance of hardware as well as applications.

III. Review of Performance Models

As we discussed before, performance models are representation for systems performance (Application and hardware), the output of these modules will depend on the system inputs and it should help to classify the target system. Performance of a particular system can be exploited from many parameters, things like power consumption, to execution time and others. But in order to define a useful module that can help to understand and develop the system, we have to go through a sequence of actions step-by-step:

1. Calculate the execution time of a specific application on a specific device.
2. Identify performance bottlenecks, modify the code for the next time.
3. Calculate the estimate power consumption.
4. List details of the resources usage for the next step (Simulation).

A. Estimate Execution Time

Estimate the execution time is very important factor in performance modeling. But in the same time, it is a difficult task because of the verity of issues that would cause delays and changing in execution time and others factors. This topic can be classified depending on different modules nature. We take GPU as a platform to review estimation execution time. For a use case example, we will highlight a generic mechanism to estimate execution time in GPUs, this is not the ultimate way but its briefly highlight general steps. Before Estimating time in GPUs, we need to classify the process of execution an application on a GPU [20], in these steps:

a. Dispatch the data of an application into GPU memory.
b. Execute the application
c. Collect the results from the GPU memory.

There are different models to calculate execution time; we will highlight one of them in this paper [20]. The module calculates each step consumption time individually and finally add them together to find the total time. In the Equation 1, (W dispatch) is the amount of data to copy from CPU into GPU memory, divided by the bandwidth of the dispatch. βdispatch is the error term of the linear regression.

$$T_{\text{dispatch}}(W_{\text{dispatch}}) = \frac{W_{\text{dispatch}}}{dispatchBandwidth} + \beta_{\text{dispatch}} \qquad (1)$$

Equation 2, the execution time can be calculated as:

$$T_{\text{exec}}(W_{\text{exec}}) = \frac{W_{\text{exec}}}{GPUspeed} + \beta_{\text{exec}} \qquad (2)$$

Equation 3, (Wcollect) is the amount of memory to copy from GPU to CPU when the application has finished, divided by the bandwidth of the collect.

$$T_{\text{collect}}(W_{\text{collect}}) = \frac{W_{\text{collect}}}{collectBandwidth} + \beta_{\text{collect}} \qquad (3)$$

Equation 4, Finally adding these values together to find the total consumption time.

$$T_{\text{GPU}} = T_{\text{dispatch}} + T_{\text{exec}} + T_{\text{collect}} \qquad (4)$$

As mentioned before this is not the only way to measure. The the modern architectures require more complex theory and it variegate from one to another based on devices family and hierarchy, but this is a straight foreword way to follow and understand the idea behind it.

B. Bottleneck and Code Optimization

While the application code being transformed into an accelerator, it is very important to analysis its performance during the runtime, and looking for any potential bottleneck, and fix them by making some recommendation to modify the code. And just like any programing compiler, for example, it can give you recommendation to fix the situation and remove the code from the bottle neck. However, the new modification recommendation can kill the performance or open a new bigger problem. So, these modifications should be under study and carefully check before applying them. Accelerators programing software usually have profilers in their toolkits (such as NVIDIA Visual Profiler for the CUDA platform), these tools should be used to optimize the code. Their job is analyzing the code execution and spotting any bottleneck.

What are the bottlenecks and Why they should be eliminated? Bottleneck is any code segment that contends threads Bottleneck may consist of a single thread or multiple threads that need to reach a synchronization point before other threads can make progress [21]. There are two general types of bottlenecks, threads that stall due to a bottleneck with called writer [21] and the ones that execute a bottleneck called *executers.* A single instance of a bottleneck can be responsible for one or many waiters [21]. Bottlenecks have a great impact on the performance because the processor spends many executing cycles to execute these bottlenecks which waste a tremendous amount of time sometimes. Bottlenecks cause thread serialization. Therefore, a parallel application spends most of its executing time in bottlenecks, which make the situation worse is these types of application. Some examples of bottlenecks.

Amdahl's Serial Portions [4 Bottlenecks]: One thread exists on a critical path and should be scheduled on the fastest core to minimize execution time.

Critical Sections: Only one thread can execute a particular section at a given time, and any other thread wanting to execute the critical section must wait.

Barriers: When a thread reaches a barrier, it must wait until all other threads reach the same barrier.

Pipeline Stages: In a pipelined parallel program, loop iterations are split into stages that execute on different threads. Threads executing other pipeline stages wait for the slowest pipeline stages

C. Selecting Code Optimization

As mentioned before, manufacturers manuals recommending different optimization techniques that can be applied to accelerate the execution times of program. And Some of these optimization tools can even be implemented automatically by compilers. For example, in this book [22], the author describes a methodology and some heuristics to find and optimize parallel code in the Xeon Phi. He provides a taxonomy of potential optimizations, relates the metrics that indicate the presence of bottlenecks and describes some good practices to remove them a combination of optimizations is not trivial, because of possible negative interactions among them.

D. Power Consumption Estimation

Over the last 15 years, HPC manufacture companies always looking to reduce the power consumption of their devises, and to achieve that, they have to know exactly the performance of their existing products regarding power consumption. Application power modeling aims to estimate the power required to execute a selected segment of code on a specific device. To estimate accurately, we must not only look at the code and device properties, but also application inputs and some other characteristics.

Standalone Models: Wang and Ranganathan [23], to start modeling first need to execute a set of micro-benchmarks with the help of external power consumption meters to characterize the target device. The module finally will estimate the power conception based on the execution time and other factors. Then after measuring that, the aim is identifying the enough number of multiprocessor that would provide the best power consumption and sufficient performance any ways they must not affect other performance factors by reducing power consumption.

Single Simulators model: The second type of power consumption modules is the single simulator ones, these are tied to one type of devices. Most of these are simulators and used for multiple performance simulating not only power consumption, will cover some in the simulation section the next.

E. Simulation

Simulators can be defended as representatives of models behavioral; they are used widely to evaluate performance of hardware as well as applications. Gathered simulation data can be used for multiple proposes for instance help to develop the simulated systems (hardware and applications). A Simulator should can be classified as a general performance tool but general under the specific type of device, not all devices, this tool will provide different performance measurement from bottleneck detection to power consumption and code optimization. The level of accuracy is also down to the devices itself, so kind of both simulator and the processor help each other, the processor should provide the simulator accurate modeled details and the quality of the workloads in order to receive accurate and valid simulation from the simulator. There are different types of simulators, we are interested in GPUs related simulators, but still highlighting others. GPUs related can be classified furthermore to a simulator for GPUs-based accelerators and simulators of Hybrid architecture (GPUs + CPUs).

Sequential GPU Simulators: Attila [24], One of the earliest graphics programs simulator software, a highly configurable approach that was classified as a generic tool, although it has been used mostly for GPUs but it was completely GPU manufacture independent. It simulates by gathering dynamic traces from OpenGL applications. One main downside about it does not support CUDA or GPGPU.

GPGPU simulators [25], One of the most widely used simulators, specifically designed to fulfill GPGPU. Provide functional and cycle-level timing simulation for NVIDIA GPUs.

Parallel GPU Simulators: Barra-Sim [16] is a functional GPGPU Simulator for the NVIDIA GPUs. Support CUDA codes. GPGPU - simulator [26] parallelized GPGPU-Sim. They divide the functional units into shared and parallel components.

Simulators of Hybrid Architectures: This type of simulators is capable of modeling hybrid architectures (CPU and GPU) that run heterogeneous applications. Gem5-GPU [27] is a full-system CPUGPU simulator, written specifically for Gem5 [28]. It can simulate programs for GPU and CPU simultaneously. Multi2Sim [29] is a simulation framework for heterogeneous Systems, was generally designed to include different architectures and modules like superscalar, Multithreaded, and Multicore. A summary of the simulation tool discussed in this paper is provided in Table 1.

Simulator	Heterogeneous	CUDA codes	Require Configur-ation	Accuracy
GPGPU-Sim [22]	No, only for NVIDIA GPUs	yes	yes	98.3 %
Barra-Sim [12]	No, only for NVIDIA GPUs	yes	yes	Low
Multi2Si-m [25]	GPU+CPU	Yes	No	7 – 30 %

Table 1 Simulators

IV. CONCLUSION

Huge diversity of the HPC devices and fast growing development in such short period of time (about 15 years), Researchers and companies are working to achieve better performance regardless the tremendous level of high performance they achieved so far. The developers of HPC devices may be capable of developing and moving forward even faster, but they facing the challenge of power consumption, for example increasing the number of cores in multicore architectures can boost the performance, but in the same time, it will consume much power which should be in the consideration when designing these architectures. As stated many times before, Parallel Application always harder and more complex than the ordinary application in all different situations and performance factors. The programmers have to be familiar and in-depth with the devices they are using before start writing code in order to achieve the best possible performance. Performance exploding is a very complicated task, because of the many parameters that affect performance of a system. There is no super general simulation tool that can fit all different architectures and applications. Homogenous architectures are out to date term, but in the same time Heterogenous used the same Homogenous cores, the challenge is how to synchronize them!

V. References

[1] S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–35, 2015.

[2] Y. Gao and P. Zhang, "A Survey of Homogeneous and Heterogeneous System Architectures in High Performance Computing," *2016 IEEE Int. Conf. Smart Cloud*, pp. 170–175, 2016.

[3] M. O. Agyeman, A. Ahmadinia and N. Bagherzadeh, "Performance and Energy Aware Inhomogeneous 3D Networks-on-Chip Architecture Generation," in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 6, pp. 1756-1769, June 1 2016.

[4] M. O. Agyeman and A. Ahmadinia, "A systematic generation of optimized heterogeneous 3D Networks-on-Chip architecture," NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013), Torino, 2013, pp. 79-83.

[5] M. O. Agyeman and A. Ahmadinia, "Optimising Heterogeneous 3D Networks-on-Chip," International Symposium on Parallel Computing in Electrical Engineering, Luton, 2011, pp. 25-30.

[6] M. O. Agyeman, A. Ahmadinia and A. Shahrabi, "Low power heterogeneous 3D Networks-on-Chip architectures," International Conference on High Performance Computing & Simulation, Istanbul, 2011, pp. 533-538.

[7] J. D. Owens *et al.*, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1. pp. 80–113, 2007.

[8] GPGPU.org, "About." [Online]. Available: http://gpgpu.org/about. [Accessed: 25-Feb-2017].

[9] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[10] R. Bott, *Intel Xeon Phi Coprocessor High Performance Programming*, no. 1. 2014.

[11] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 272–281, 2015.

[12] M. Silberstein, "GPUs: High-performance Accelerators for Parallel Applications," *Ubiquity*, vol. 2014, no. August, pp. 1–13, Aug. 2014.

[13] N. P. Tran, D. H. Choi, and M. Lee, "Optimizing cache locality for irregular data accesses on many-core intel Xeon Phi accelerator chip," *Proc. - 16th IEEE Int. Conf. High Perform. Comput. Commun. HPCC 2014, 11th IEEE Int. Conf. Embed. Softw. Syst. ICESS 2014 6th Int. Symp. Cybersp. Saf. Secur.*, pp. 153–156, 2014.

[14] Altera, "Intel FPGA and SoC." [Online]. Available: https://www.altera.com/. [Accessed: 23-Feb-2017].

[15] K. Opencl, "OpenCL Specification," *ReVision*, pp. 1–385, 2009.

[16] "CUDA Zone | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/cuda-zone. [Accessed: 24-Feb-2017].

[17] P. Banga, A. Pai, S. Roy, and M. Chaudhuri, "Accelerating Schedule Space Exploration of Multi-threaded Programs with GPUs," no. 978, pp. 115–124, 2016.

[18] R. Jain, *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.*, vol. 491. 1991.

[19] T. M. Mitchell, *Machine Learning*. 1997.

[20] P. Velho, D. A. G. de Oliveira, E. L. Padoin, and P. O. A. Navaux, "Accurate Analytic Models to Estimate Execution Time on GPU Applications," *XI Parallel Distrib. Process. Work.*, pp. 1–4, 2013.

[21] J. a. Joao, M. A. Suleman, O. Mutlu, and Y. N. Patt, "Bottleneck identification and scheduling in multithreaded applications," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 1, p. 223, 2012.

[22] Rezaur Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application.* .

[23] Y. Wang and N. Ranganathan, "An instruction-level energy estimation and optimization methodology for GPU," *Proc. - 11th IEEE Int. Conf. Comput. Inf. Technol. CIT 2011*, pp. 621–628, 2011.

[24] V. M. del Barrio, C. Gonzalez, J. Roca, and A. Fernandez, "ATTILA: a cycle-level execution-driven simulator for modern GPU architectures," *2006 IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 231–241, 2006.

[25] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing {CUDA} Workloads Using a Detailed {GPU} Simulator," *Ispass*, pp. 163–174, 2009.

[26] S. Lee and W. W. Ro, "Parallel GPU architecture simulation framework exploiting work allocation unit parallelism," *ISPASS 2013 - IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 107–117, 2013.

[27] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A Heterogeneous CPU-GPU Simulator," *IEEE Comput. Archit. Lett.*, vol. 14, no. 1, pp. 34–36, 2015.

[28] N. Binkert *et al.*, "The gem5 Simulator," *Comput. Archit. News*, vol. 39, no. 2, p. 1, 2011.

[29] R. Ubal, P. Mistry, D. Schaa, H. Ave, and D. Kaeli, "Multi2Sim : A Simulation Framework for CPU-GPU Computing," *Proc. 21th Int. Conf. Parallel Archit. Compil. Tech. (PACT'12). Minneapolis, Minnesota, USA. Sept. 19-23. USA ACM Press*, pp. 335–344, 2012.