North Asian International Research Journal of Sciences, Engineering & I.T.

IRJIF. I.F. : 3.821

ISSN: 2454-7514 Vol. 6, Issue-12 December -2020

Index Copernicus Value: 52.88 Citation Index

A Peer Reviewed Refereed Journal

BENEFITS OF HETEROGENOUS COMPUTING

SHEELA GUPTA

ABSTRACT

Today's mobile, IoT, AI applications have increasing performance demands which are outpacing the current advancements in semiconductor and power-storing technologies. Heterogeneous computing is designed to maximize performance while improving thermal and power efficiency. Heterogeneous computing refers to systems that use more than one kind of processor or core, with capabilities to handle a specialized task. Hence, heterogeneous computing combines general-purpose computing with specialized computing to achieve high performance and reduced power consumption. This paper presents some of the primary software and hardware challenges faced in heterogeneous computing systems. This paper also presents some of the key reasons why heterogeneous computing is the future. The goal of this paper is to offer a brief overview of the latest CPU-GPU heterogeneous computing systems (HCS) by taking examples of IBM Cell Broadband Engine and ARM big. LITTLE.

1. INTRODUCTION

Moore's Law states that the number of transistors will double every 18 months. According to Dennard scaling as the transistors get smaller, their power density stays constant. Moore's law combined with Dennard scaling has been the principal driver of the evolution of microprocessor designs for decades. While Moore's law held for more than 50 years, eventually it has become more challenging to utilize its advantages. At the same time, Dennard's Scaling seemed to break down around the 2005-2006 period. This has shifted the focus on multicore processors as an alternative way to improve performance. As the number of cores increases, power restrictions will prohibit all cores from running at their full speed, causing at all times a fraction of the cores to be switched off. There is increased attention to new software paradigms and alternative system architectures to drive future performance improvements. [1]

The transition from the first mobile phones to the smartphone of today has seen an unprecedented increase in demand for computing in mobile devices. There is a steady increase in consumers' demands, user experiences, performance, and accessibility when it comes to our mobile devices. Such interactions not only require very high capacity but also include addressing different requirements for new types of workload. The challenge is, therefore, to provide these new mobile experiences while staying dedicated to sleek, ultra-light phones that do not overheat and give long battery life. Higher performance demands require fast processing speeds which is difficult to fit into a mobile power or thermal budget. Unfortunately, advancements in battery and semiconductor technology have not progressed at the same speed as CPU technology, resulting in a situation where smartphones require higher efficiency, but the same power consumption. [8]

Heterogeneous computing involves using other processors to boost application performance, and power efficiency while more demanding applications are created. Heterogeneous computing refers to systems that use more than one kind of processor or core, with capabilities to handle a specialized task. Figure 1 shows a general idea of an heterogenous system [2].

2. HETEROGENEOUS PARALLELISM

Consider a situation in which you have to select between 2 processors to perform a computation. Processor A has 4 cores, each with a sequential performance P. Processor B has 16 cores, each of which has a sequential performance of P/2 [Figure 2]. Processor B's work rate is potentially more than Processor A. Nonetheless, if we find a computation that does not permit an ideal parallel task distribution and consists of a series of sequential computation, Processor A will be better suited to address that computation. To address a wide range of workload characteristics, we need an asymmetric set of processing cores [Figure 3][2].

Existing high-performance processors on certain tasks produce only a fraction of their peak performance capabilities. This is because different tasks can have very different computational demands resulting in different computer capabilities being required. A single machine architecture may not conform equally well to all the computational requirements of various tasks. Therefore, it is more fitting to use a heterogeneous computing system. Heterogenous computing refers to a series of diverse systems connected by high-speed networks. Modern applications comprise highly specialized computational operations that can be handled by heterogeneous computing systems in an efficient and performant manner. Heterogeneous computing systems use the right processor at the right time for the right task. We rely on specialized processors like GPUs, DSPs along with the CPU to address a wide variety of computational needs.

3. EVOLUTION OF HETEROGENEOUS SYSTEMS

Intel released its first microprocessor, Intel 4004 in 1971 which had 2300 transistors. Since then performance improvements in single-core microprocessors have been possible by following Moore's Law and Dennard scaling. Shrinking transistors meant, more transistors could be accommodated in the same chip area, to increase the performance. However, the performance gains were significantly lower as predicted by Moore’s law. This was due to the fact the performance of some of the components like cache memories did not scale well with the increased area. Even if the transistors were getting smaller and smaller, but this did not result in the doubling of the transistors as predicted. This was due to an increase in wires and increased complexity of the interconnect. Performance improvements were made by increasing the frequency of CPU and instructions per clock cycle (IPC). However, this presented more challenges, as the complexity of the architecture increased. Single-core processors were limited as the energy consumption increased and increase in the operating costs. [4]

Since 2004, there has been a shift towards multicore. Multi-core CPU combines multiple cores into one processor chip. This shift to multicore designs was to compensate for the end of Dennard scaling. A multicore processor can run multiple threads parallelly on different cores with resource contention only for the shared resources. The interconnection network and memory hierarchy are two key components that drive the performance of multicore systems. However, power constraints will act as a barrier to powering all cores at their full speeds. If we continue to scale by adding cores, then only a fraction of cores will be powered on at all times within thermal constraints. In addition to this, the core might not be the most efficient processor.

To continue scaling processing capacity, and at the same time adhering to thermal constraints, requires a different approach. The challenge for future computing systems is not only how to make them run faster, but also how to bring about a similar reduction in energy consumption. Heterogenous computing integrates various processing elements into a coherent environment for power efficiency. Sony, Toshiba, and IBM created a heterogeneous CPU, the Cell Broadband Engine, integrating a central PowerPC with several floating-point coprocessors. The Cell was used in Sony's PlayStation 3 game console. In 2010, Intel launched the first integrated graphics processor. Intel's Sandy Bridge came out in January 2011 a year later. In 2006 AMD started working on APU (Accelerated Processing Unit), which was the first attempt to combine the GPU and CPU on the same die. [5]

4. CHALLENGES IN HETEROGENOUS COMPUTING

4.1 Hardware Perspective

The rate of microprocessor speed enhancement exceeds the rate of DRAM memory speed enhancement. Thus, there is a large performance gap between processor speed and memory speed. This problem has existed since the single-core era. The memory hierarchy is distributed among the different processors in the case of heterogeneous computing. Complicated problems such as which computing components share which level of caches that need to be tackled. Cache coherence is another hurdle. Coherence means ensuring that the same information is interpreted by all processors or bus masters in the network. For example, if I have a processor that generates a data structure in its local cache and then transfers it to a GPU, the same data must be seen by both the processor and the GPU. The GPU can read old, outdated data if the GPU reads from existing DDR. Designing such a hierarchy is a formidable task considering performance issues, and power consumption.

Another task for the hardware developer is to choose the correct interconnection network that must fulfill the various traffic specifications of the different computing units. Several factors like power dissipation, material and topology need to be considered to create an energy-efficient and high performant network in heterogeneous systems. Another difficulty is to spread the workload between the different cores in order to achieve the best performance with the lowest power consumption. [2]

In heterogeneous systems, various computing units support different types of parallelism, such as vector parallelism, data parallelism, etc.

There is a different set of memory architecture associated with each of these parallelisms which weigh into the performance tradeoffs. There is a different set of memory architecture associated with each of these parallelisms which weigh into the performance tradeoffs. The processing elements can have different ISA (Instruction Set Architectures) which can lead to incompatibility and adds to the complexity for the programmer.[6]

4.2 Software Perspective

Programming on a heterogeneous system poses numerous challenges, especially when we scrutinize energy efficiency. It is difficult to design algorithms that work on various heterogeneous platforms and cater to different models of parallelism. A series of algorithms are designed for the targeted environment and the suitable one is selected when the actual underlying hardware is known. It is difficult to provide a source-level programming language for a heterogeneous system that supports various modes of parallelism.

Software portability is difficult to achieve if we are targeting a wide variety of platforms that have different combinations of hardware. It is difficult to design a compiler which will map the various modules of the source code to the respective hardware components. Since we are addressing diverse hardware, performance tuning will also become complicated. Manufacturers that target software for multiple devices, need to take into consideration the source code portability. Since performance tuning and debugging of the source code will require modification of the source code, this will require additional tools like debugger, profiler which intern will need to address a wide variety of hardware.[6]

The reliability of the software is also essential. Since we are developing software to run on various distributed environments, there is an increased chance of failure. It becomes imperative for the programmer to handle these fault scenarios. The strategy to deal with failures can rely on the hardware, or the host system software. The programmer can also design their routines to handle failure.

The scalability of the program is much more complicated in a heterogeneous system. In case of programs targeting homogenous systems, only have to consider scaling based on the compute units of the same type. However, in the case of heterogeneous systems, the type of the compute units needs to be taken into account.[2]

5. CPU-GPU HETEROGENEOUS COMPUTING

GPU can be used to solve processing-intensive applications when used as a CPU accelerator. To achieve high-performance computing, the collaboration between CPU and GPU is unavoidable. Accelerated GPU computing requires redirecting the compute-intensive operations to the GPU. Theoretically, the CPU and GPU are capable of performing the same computations, however, there is a large difference in execution model and architecture. The tremendous computational power of the GPUs comes at the same cost as the CPU.

5.1 Workload Distribution Techniques

We want to get the best out of both the processing units and this means the computation needs to be divided rationally. The whole task is divided into small sub-tasks, which are as fine-granular as possible. When a subtask cannot be subdivided any further then the maximum granularity is reached. Each of these subtasks will then be assigned to a processing unit. Figure 5 illustrates the basic idea behind this. [7]

5.1.1 Native Partitioning

In the native partitioning approach, the granular tasks are associated with the processing units based on the count or relevance in the system. To take an example of this, consider we have a CPU with 2 cores and a GPU. The CPU is the main processor and the GPU is used only as accelerator, then 2/3 of all the tasks will be mapped to the CPU and the rest will be mapped to the GPU. Figure 6(a) shows the ideal periods for this approach. CPU core 0 acts as master, CPU core 1 and GPU act as slaves. We can easily imagine that this partitioning approach will result in large idle periods for the GPU.[7]

5.1.2 Partitioning by Processing Unit Performance

In this partitioning scheme the granular tasks is distributed based on the relative performance of the participating processing units. For a particular task if the GPU is twice as fast as the CPU, then the tasks will be mapped according to this (see Figure 6(a)). However, it is imperative to know the performance capabilities of the processing units for a particular problem beforehand. [7]

5.1.3 Partitioning by the nature of the sub-tasks

In this scheme of partitioning, we first identify which processing unit is suitable for which processing units and allocate the tasks accordingly. CPU is suitable for task-based parallelism and latency-oriented tasks, whereas the GPU is suitable for data-based parallelism and suitable where throughput is required. In Figure 7, the red tasks are CPU affine, whereas the blue task is GPU affine.[7]

6. ARM big. LITTLE TECHNOLOGY

Arm big.LITTLE technology is a heterogeneous chip architecture that integrates two types of processors, "big" processor, and "LITTLE" processor. The "LITTLE" processor is designed to maximize power efficiency by drawing as little power as possible. The "big" performant processor is designed to cater to high compute requirements within thermal thresholds. The design of big.LITTLE technology is designed to cater to the increasing performance demands of the current smartphones while maintaining power efficiency.

The Cortex-A7 CPU is able to perform most of the tasks of smartphone users. When the performance requirements demand more processing power like in the case of gaming, the Cortex-A15 cores are turned on to meet this requirement. The tasks are migrated to the Cortex-A15 cluster. When the task's performance requirement is reduced, the task will again be migrated to the cluster of Cortex-A7 cores. This migration is possible because both the Cortex-A7 and Cortex-A15 processors support the same ARMv7 Instruction Set Architecture.

The ARM big.LITTLE technology involves constant data transfer between big and LITTLE processors. If this migration took place via DRAM, it would be a slow process and a sophisticated software implementation would be required to maintain cache coherency. This seamless migration of data is possible because of "Cache Coherent Interconnect". The Cortex-A15 and Cortex-A7 processors make use of AXI Cache Coherency Extensions (AXE) and AMBA AXI protocol.[8]

6.1 Software Execution Models

There are 2 software execution models for ARM big.LITTLE technology

6.1.1 CPU Migration

In this execution model, pairs consisting of a big and LITTLE processor are formed. Only one of the two processors is active at any point in time. The inactive pair is powered down. The active processor amongst the two processors is chosen by the scheduler based on the instantaneous performance demands. This model requires the same number of big and LITTLE processors. This is illustrated in Figure 9.

6.1.2 Global Task Scheduling

Here the scheduler tracks the performance requirement of each thread. The scheduler also maintains the current state of the processors. With the help of past performance metrics and other statistical data, it figures out the best option between the big and LITTLE processors. for that thread. All the unused processors can then be powered off.

This approach is better than CPU migration in many ways. Firstly, we no longer need to have the same number of big and LITTLE processors. We can target the interrupts to a particularly big or LITTLE processor, whereas in case of CPU migration, the pair had to be considered for the interruption. In the case of CPU migration, only half of the core is powered on at any given time. This limits the performance capability and prevents the system from deploying all the cores to meet the load demands. In the case of Global Task Scheduling, these cores can be individually powered off or on, which means when the time comes, the system can use the full processing power available by turning on all the cores.[8]

7. CELL BROADBAND ENGINE

IBM cell broadband engine is based on Cell Broadband Architecture. The Cell Broadband Engine was developed by IBM, SONY and Toshiba. The Cell broadband engine is a heterogeneous mixture of regular microprocessors. It includes one POWER Processing Element (PPE) and eight Synergistic Processing Elements (SPEs). The PPE act as a desktop processor and the SPE act as an accelerator like a GPU. The performance improvements in conventional desktop processors were made by increasing the clock frequency and increasing the complexity of the architecture by introducing new pipeline designs. This would result in inadequate utilization of the chip area and aggravate the power dissipation. The architecture of the Cell Broadband Engine is presented in the Figure 10.

The main design point behind the architecture of the Cell is to improve the performance/power ratio. The performance improvement is achieved by using simple microprocessors that take up less space and has less power dissipation. All the cores are connected via a high-speed interconnect with a high data bandwidth. This allows the cores to access memory simultaneously.

7.1 Power Processing Element (PPE)

PPE is the chief processor of the Cell. It is responsible for running the host operating systems and coordinating the SPEs. PPE is responsible for scheduling the processes on the PPE. It supports out of order execution and dynamic branch prediction.

7.2 Synergistic Processing Element (SPE)

The SPE consist of eight Synergistic Processing Units (SPU) and Memory Flow Controller (MFC). SPU have access to instructions and data directly from its local store (scratchpad memory).

7.3 Element Interconnect Bus (EIB)

The PPE and SPEs communicate with the help of the Element Interconnect Bus. It is a circular data bus comprising of one address bus and 16B-wide data rings. Two of the rings are clockwise and the other two counterclockwise. To access a data ring, the requester must send out a request to the EIB bus arbiter, which then decides which requester should be granted access to the data ring. The data arbiter gives the memory controller the highest priority, while all the other requests are attended in a round-robin manner. The arbiter will not grant access to a requester if it is going to interfere with an existing data transfer. Each ring can allow up to three concurrent data transfers if there is no overlap between them.[9]

8. SUMMARY

In this paper, we looked at how heterogenous computing combines general-purpose computing with specialized computing as a response to the growing performance requirement and energy-efficient computing. We also looked at some of the challenges faced in designing a heterogeneous system. There is a paradigm shift in the mainstream hardware which is becoming increasingly heterogeneous and distributed. Heterogenous computing integrates various computing elements into one cohesive environment for energy-efficient computing. We also looked at CPU-GPU heterogeneous computing and how we can harness their computational powers by effectively distributing the workloads amongst them. Heterogeneous computing is still in the infancy stage and is here to stay. Hardware manufacturers see great promise in heterogeneous computing are investing more time and energy to develop energy-efficient solutions for modern-day processing needs.

9. REFERENCES

[1] Esmaeilzadeh, Hadi, et al. "Power challenges may end the multicore era." Communications of the ACM 56.2 (2013): 93-102.ACM Special Interest Group Proceedings Word template.

[2] Zahran, Mohamed. "Heterogeneous computing: Here to stay." Communications of the ACM 60.3 (2017): 42-45.

[3] “Modern Heterogeneous System Examples” ttp://15418.courses.cs.cmu.edu/spring2013/article/37 askumar, jkonstan, tnebel, and vvallabh

[4] Antonio González, “Trends in Processor Architecture”. Universitat Politècnica de Catalunya, Barcelona, Spain

[5] How Heterogeneous Systems Evolved and the Challenges, Going Forward https://opensourceforu.com/2016/12/how-heterogeneous-systems-evolved-and-the-challenges-going-forward/

[6] Adve, Vikram, et al. "Virtual Instruction Set Computing for Heterogeneous Systems." Proceedings of the 2012 4th USENIX Workshop on Hot Topics in Parallelism (HotPar'12), Berkeley, California. Vol. 136. 2012.

[7] Lammel, Steffen. "CPU-GPU Heterogeneous Computing." Memory (capacity) 1 (2015): 12GB.

[8] White Paper, "big.LITTLE Technology: The Future of Mobile" https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[9] Thomas Chen , Ram Raghavan, Jason Dale, and Eiji Iwata. "Cell Broadband Engine Architecture and its first implementation" https://www.ibm.com/developerworks/library/pa-cellperf