Dynamic Thermal Management Of Vertically Stacked Heterogeneous Processors

Ajay Sharma
University of Mississippi

DYNAMIC THERMAL MANAGEMENT OF VERTICALLY STACKED HETEROGENEOUS PROCESSORS

A Thesis
presented in partial fulfillment of requirements
for the degree of Masters of Science
in the Department of Computer and Information Science
The University of Mississippi

by

Ajay Sharma

December 2016

ABSTRACT

To address various physical limitations of traditional 2D circuits, processor architecture is evolving toward a 3D heterogeneous integration (commonly termed as 3DIC) of CPU, GPU and DRAM dies vertically interconnected by a massive number of TSVs (Through-Silicon Vias). Such 3D heterogeneous processors face two major challenges: thermal hotspots and temperature gradients which degrade the performance and thermal reliability of the processor.

In this thesis, we develop a framework for a dynamic thermal management technique where temperature aware workgroup scheduling on CPU/GPU is performed. Our proposed algorithm leverages the temperature history of cores to avoid assigning workgroups to hotspots. Our experiments show that vertical temperature correlation of cores is higher than horizontal correlation on 3D stacked heterogeneous processors. It also demonstrates that, compared to random and FIFO based algorithms (existing workgroup scheduling algorithms), our proposed algorithm achieves fewer hot spots and better temperature gradients for CPU-GPU 3D heterogeneous processors.

ACKNOWLEDGEMENTS

I would first like to thank my research advisor Dr. Byunghyun Jang for his unconditional support and encouragement throughout my masters program.

I would like to thank Dr. Cunningham, my advisor in the starting semesters of the program and the department chair, Dr. Dawn Wilkins, for their continuous guidance and support in both academic and non-academic settings.

I would like to acknowledge Md Mainul Hassan for his hard work on his masters thesis titled "POWER AND HOTSPOT MODELING FOR MODERN GPUS," which has served as a groundwork of this thesis.

I would like to thank all of my fellow HEROES research group members, David Troendle, Kyoshin Choo, Hossein Pourmedani, Tuan Ta, Chelsea Xu and Esraa Gad for helping me develop my technical and social skills.

I thank all faculties and staff members of the Computer and Information Science Department who made learning and research fun. I would like to thank the rest of department colleagues who have been one of the most fun and friendly people to work with.

This research is supported by National Science Foundation (NSF) grant CCF-1337138.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

3D IC (Three-Dimensional Integrated Circuit) technology reduces the length of interconnects in a system/processor as functional blocks are integrated closer in space, thus reducing the latency of data transfer between the blocks. This can improve the performance of the processor significantly. However, 3D integration introduces thermal management challenges due to the high power density resulting from the stacking of computational units on top of each other. High power densities, already a major concern in 2D circuits, are even more severe in 3D circuits [2, 14].

Thermal hot spots increase cooling costs, negatively impact reliability and degrade performance. Hot spots accelerate failure mechanisms such as electromigration, stress migration, and dielectric breakdown, which cause permanent device failures [1]. Leakage is directly related to increased temperature. Further, there is a positive feedback loop between leakage and heat generated by silicon transistors, which is called thermal run-away. Leakage prevention is very important since higher leakage can lead to hot spots. Thus, we need a thermal management technique that controls leakage. Large spatial temperature gradients across the chip cause performance degradation or logic failures. Negative bias temperature instability (NBTI) and hot carrier injection (HCI) cause circuits to miss timing constraints [4].

In this thesis, we propose a dynamic temperature-aware task scheduling algorithm as a thermal management mechanism for 3D heterogeneous processors with GPUs stacked on a

CPU. Heterogeneous processors refer to single chip processors that integrate more than one type of computational architecture. These systems gain performance and energy efficiency not by adding more same types of processors, but by incorporating specialized processing capabilities to handle particular tasks. Recent findings also show that a heterogeneous-ISA (Instruction Set Architecture) chip multiprocessor that exploits different parallelisms offered by multiple ISAs, can outperform the best same-ISA heterogeneous architecture by as much as 21% with approximately 23% energy savings and a reduction of 32% in energy delay product [20]. Thermal management schemes can be broadly classified into:

1. Static thermal management

   This scheme follows a fixed set of rules and does not consider dynamic temperature changes at runtime. Static thermal modeling techniques include integer linear programming (ILP) based static scheduling [5] and some thermal aware floor planning algorithms [8, 15]. However, static thermal management techniques are not enough to alleviate the problem of hot spots in 3D ICs.

2. Dynamic thermal management

   Dynamic thermal management includes thermal aware job allocation [15, 21] and DVFS (Dynamic Voltage and Frequency Scaling) [1, 9]. This scheme has been well researched for homogeneous system architectures but not as much for heterogeneous processor architectures.

In this thesis we develop a framework for power and thermal modeling of 3D heterogeneous processor and analyze their results. Our framework is capable of modeling both homogeneous and heterogeneous thermal models. We discuss the heterogeneous model in detail in this thesis. We have designed the framework into three stages which are architectural simulation, power calculation/modeling and finally thermal modeling. To the best of our knowledge, there has not been any work done in this specific field. Using our robust framework we developed a “Dynamic Thermal Reliability Management (DTRM)” technique to reduce hotspot generation and reduction of temperature gradients in 3D heterogeneous processors. We have developed a thermal aware scheduling algorithm targeting heterogeneous architectures and the OpenCL model. Our algorithm detects and removes hot spots, reduces the thermal stress and improves the overall spatial temperature gradient of the 3D IC. The detailed tasks accomplished in this thesis include:

- We modified an architectural simulator, Multi2sim [19], to generate cycle accurate statistics required by our power modeling tool.
- We used two different versions of McPAT [10] for modeling the power of our heterogeneous processor, one for CPU and the other for the GPU. We made significant changes to both versions to get cycle accurate power consumption data of the CPU and the GPU.
- For thermal modeling we used a well known tool, Hotspot[7]. We specifically used the 3D stacking capability which is only available in the latest version of the product.

CHAPTER 2

BACKGROUND

In this section we provide the technical background required to understand heterogeneous computing, CPU-GPU architectures and the OpenCL programming model. Section 2.1 explains what heterogeneous computing is and Section 2.2 explains how CPU-GPU is one of the best processor combination for achieving heterogeneous computing. Section 2.3 describes OpenCL, the programming language & platform we have used in this work.

2.1 Heterogeneous computing

Heterogeneous systems generally refer to systems that use more than one kind of processor or ISAs (Instruction Set Architecture). These systems gain performance or energy efficiency not just by adding more of the same type of processors, but by adding different types of processors with specialized processing capabilities to handle various tasks better. Heterogeneity in the context of processors refers to the combination of different ISAs for computation.

The most common examples are the combination of two or more of the following architectures:

1. General purpose processors
    - CPU (x86 or ARM)
2. Special purpose processors
    - Digital Signal Processors (DSPs)
    - Graphics Processing Units (GPUs)

- Field Programmable Gate Arrays (FPGAs)

Studies have shown that using a heterogeneous system can significantly reduce the energy requirements of the processors and shows significant improvements in the thermal model of the overall system reducing the cooling and power cost [20]. Conceptually, we can allocate jobs to the most suitable processor, which gives the best performance and efficiency. Recent findings also show that a heterogeneous-ISA chip multiprocessor that exploits diversity offered by multiple ISAs, can outperform the best same-ISA heterogeneous architecture by as much as 21% with 23% energy savings and a reduction of 32% in Energy Delay Product (also known as switching energy, it is the product of power consumption, averaged over a switching event, times the inputoutput delay, or duration of the switching event) [20].

The most popular combination for heterogeneous computing is CPU and GPU. Since both processors already exist in every device, either desktop or mobile devices, the user doesn't have to pay more to achieve heterogeneous computing. A CPU is a collection of cores, each of which is a powerful heavy weight processor suitable for workloads having diverse control-flows and inter-thread-communications. On the other hand, a GPU is a collection of compute units, each of which is a SIMD (Single Instruction Multiple Data) suitable for workloads having data-parallelism with few data-dependencies (i.e., data parallel workloads with less control-flows and inter-thread communications).

Using GPU as a general purpose processor involves programming the GPU. The most popular programming languages to program a GPGPU (General purpose GPU) are OpenCL (Open Computing Language) and CUDA (Compute Unified Device Architecture).

2.2 CPU-GPU architecture

In this section we will discuss the x86 (CPU) and Southern Islands (GPU) architectures, which are the base architectures we used for our simulations and analysis.

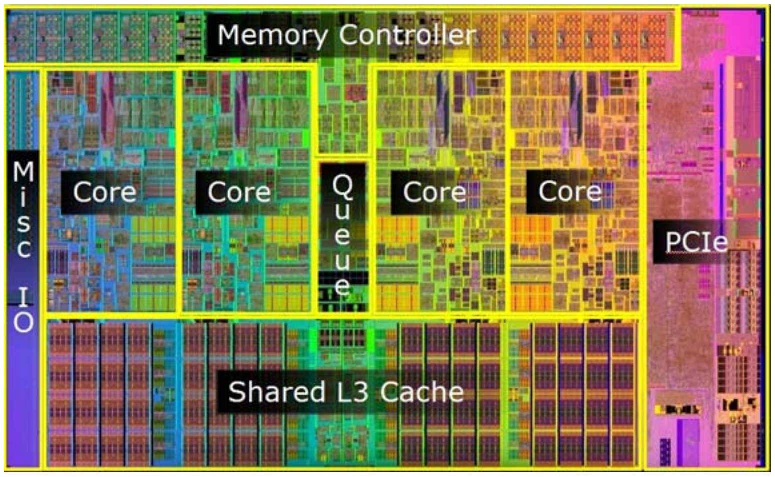

Figure 2.1. Intel i7 chip layout (source: Intel).

2.2.1 x86 CPU architecture

We modeled an x86 architecture for CPU. The x86 architecture is a CISC (Complex Instruction Set Computing) ISA, meaning a single instruction can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions. The hardware implementation of an x86 architecture can have multiple cores and multiple caches. The Intel i7, one of the most popular processors, has 4 cores and 3 levels of cache. L1 and L2 caches are private to each core. L1 is further broken down into instruction and data caches to maximize the locality. The last level shared L3 cache is significantly larger than the upper level caches and also serves as the central point of coherency for private caches. The number of hardware threads inside the core decide the number of compute units in the OpenCL model. In our CPU configuration we chose to have two hardware threads in each core which makes 8 (4 cores x 2 threads per core) compute units in the CPU. Fig 2.1 shows the layout of the Intel i7 processor.

2.2.2 CPU job scheduling

Job scheduling on CPUs is handled by the operating system. There are many different scheduling techniques implemented by different operating systems. All processes have a state where it is running such as ready, waiting etc.. We are interested in the ready state only. All processes in the ready state are enqueued in the ready queue to run on the next available core. The storage structure for the ready queue and the algorithm used to select the next process are not necessarily a FIFO queue. There are several alternatives to choose from, as well as numerous adjustable parameters for each algorithm.

The most common types of scheduling policies include SJF (shortest job first) and priority scheduling. Since the job size and priority for all the workgroups are more or less same, these algorithms do not address the specific need of OpenCL execution Model. For the OpenCL execution model, simple scheduling algorithms like round robin or FIFO are as good as SJF and priority scheduling to get good performance. Many thermal aware schedulers [1, 9, 22, 10] have been developed in the past but none has been developed from the perspective of equal job size and priority which is the case in the OpenCL model.

2.2.3 Southern Islands GPU Architecture

GPU (Graphics Processor Unit) is a specialized processor designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to screen. In recent years GPUs have been used for more general purpose processing though the use of programming models such as CUDA and OpenCL. All mordern GPUs support OpenCL. A host program is used to transfer data to the GPU, launch a program on the GPU, and retrieve data once the GPU has finished. We used a Southern Islands GPU, which is the latest GPU architecture from AMD. The reason behind selecting this architecture was to get support from the latest OpenCL versions. GPUs follow the SIMD (single instruction multiple data) execution model. The most basic computing unit in the architecture is the

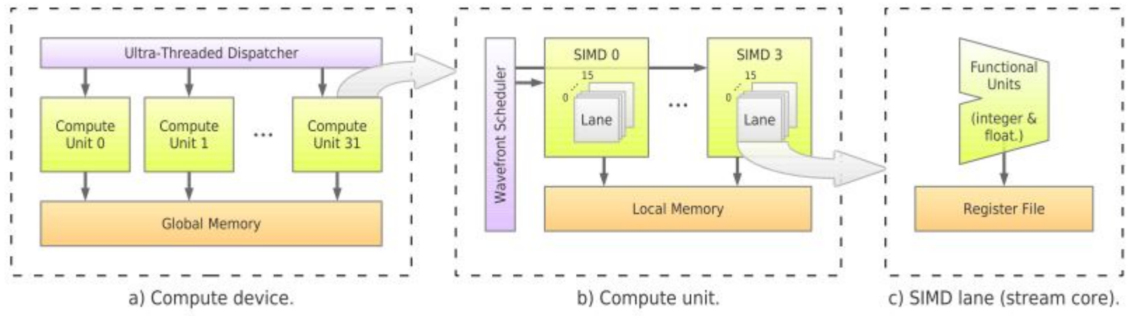

Figure 2.2. A simplified GPU architecture.

PE (Processing Element). A group of 16 PEs make a compute unit or a SIMD engine. A compute unit is equivalent to a core. Each PE executes the same instruction on different operands in the same address space. So PEs are synchronized in a compute unit and execute the same instruction on separate data in a lock-step parallel fashion. The GPU that we modeled has 32 compute units which gives it hundreds of computational nodes (32x16). In the SIMD model every PE executes the same instruction but on different set data at the same clock cycle, thus making it highly parallel.

Fig 2.2 illustrates a simplified GPU model of the Southern Islands architecture. The SIMD lane is a Processing Element (PE). The work-items (threads) in the OpenCL kernel program are grouped into workgroups. The ultra thread dispatcher dispatches the workgroups to the compute unit depending upon its availability. Once a workgroup is assigned to a compute unit, it can't be migrated to another compute unit in the middle of execution so it completes the execution on the same assigned compute unit. The maximum number of threads that execute together in a workgroup is 64 (for AMD southern islands architecture). This group of threads is called a wavefront. Workgroup can have multiple wavefronts depending upon its size. Every compute unit has a wavefront scheduler that schedules the wavefronts for better hardware utilizations. All work-items (threads) in a workgroup share a special common memory space called local memory. Local memory is private to a compute unit and frequently accessed data are moved here to accelerate data access.

2.2.4 GPU scheduler

Unlike CPUs, GPUs do not have threads with different execution times and priority. Every job in a GPU is divided into workgroups. Every workgroup (of the same kernel) executes the same code and at any instance only one kernel can run on a conventional GPU. For example, if the GPU is running a Matrix Multiplication kernel, all workgroups of Matrix Multiplication will have the same size and will execute the same code (machine instructions) on the compute unit to complete execution. Due to the uniformity of the workgroup size, GPUs only need a simple scheduler to schedule workgroups evenly across all the compute units.

GPUs don't have a software scheduler like CPUs. A GPU's scheduler is embedded in the hardware and is called the ultra-thread dispatcher (see Figure 2.3) and is responsible for workgroup scheduling. A workgroup is the quantum for scheduling used by the ultra thread dispatcher. The ultra thread dispatcher dispatches the workgroups to available compute units and the scheduling algorithm followed here is FIFO (First In First Out) since all workgroups have equal priority and require the same execution time on a compute unit. After the kernel is launched on the GPU each workgroup gets a unique id and is added to the ready workgroup queue. The ultra thread dispatcher assigns the first workgroup in the queue to the first available compute unit (FIFO) in the GPU. Once the workgroup is assigned to a compute unit it cannot be retracted from it. The workgroup will finish the execution and then the compute unit will be assigned a new workgroup. Figure 2.3 shows the inside of a compute unit.

Inside the compute unit, the work-items in a workgroup are further grouped in to wavefronts. A wavefront is a group composed of 64 work-items (in case of AMD Southern Islands). These 64 work-items execute at the same time on the compute unit (as a batch), each on one processing element. The wavefronts are collected in multiple queues called wavefront pools. Wavefronts are assigned to the Processing Elements by the wavefront scheduler.

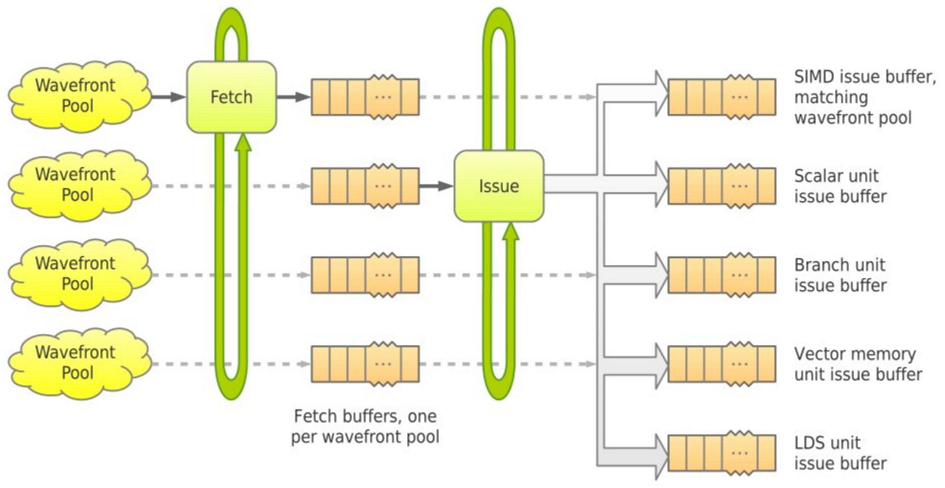

Figure 2.3. Pipeline inside a compute unit.

In our work we modify the ultra thread dispatcher to assign workgroups to the cores with lower thermal stress to achieve better temperature gradient and lower the risk of thermal hotspots.

2.3 The OpenCL programming model

OpenCL (Open Computing Language) is an open source, royalty-free standard for cross-platform, parallel programming on diverse processors. OpenCL significantly accelerates execution and improves the responsiveness of a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software.

The normal practice is to offload the computation-intensive and data-parallel workload of the computation to the OpenCL device (CPU or GPU). This does have an overhead of copying data from the host device (CPU) to the guest device (GPU) when a discrete GPU is used. At the end it depends whether or not data transfer time is less than the time saved by executing the program on the device using the OpenCL model of execution, if it is, then the application is a good fit for execution on GPU.

A typical OpenCL application has a host program (usually written in C) and a guest

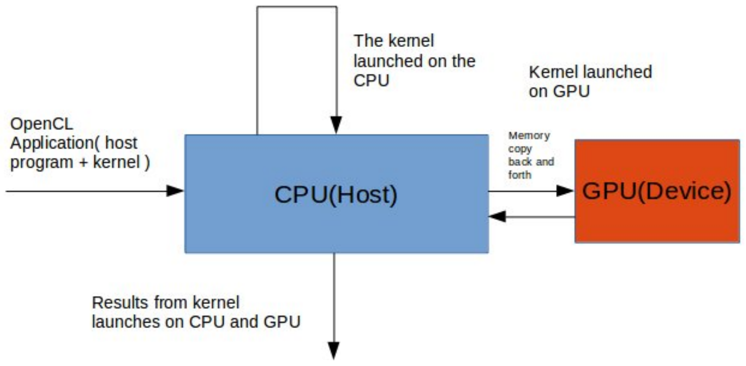
The kernel launched on the CPU
Kernel launched on GPU
OpenCL Application(host program + kernel)
Memory copy back and forth
CPU(Host)
GPU(Device)
Results from kernel launches on CPU and GPU

Figure 2.4. Block diagram of heterogeneous computing by using CPU and GPU as OpenCL devices.

program which is written in OpenCL (this program is called the *kernel*). To launch a kernel on an OpenCL device (CPU and/or GPU) the user has to setup the device first and then launch the kernel onto it with thread configuration information. Figure 2.4 shows the block diagram for the setup.

2.3.1 The OpenCL execution model

The OpenCL execution model follows the SIMD model of execution. Each OpenCL thread is called a work-item and is an instance of the kernel. The number of kernel instances is controlled by the user in the host program at the time of kernel launch. Figure 2.5 shows the processing elements inside a compute unit and how compute units can be visualized inside an OpenCL device. Multiple instances of the kernel are launched via an NDRange, which specifies the total number of work-items. NDRange is the representation of all the work-items in a 1D, 2D or 3D cube/cuboid or equivalent 1D or 2D shape. Figure 2.6 gives a visual of a work-item, wavefront and workgroup. The NDRange shows how workgroups are arranged inside it. Note that each workgroup is of equal size and has a unique workgroup id determined by x and y position of the workgroup in the NDRange. In Figure 2.5, the subfigure for workgroup shows the arrangement of the wavefronts in the workgroups. In the example shown by the Figure, every workgroup has 9 wavefronts comprising 64 work-items.

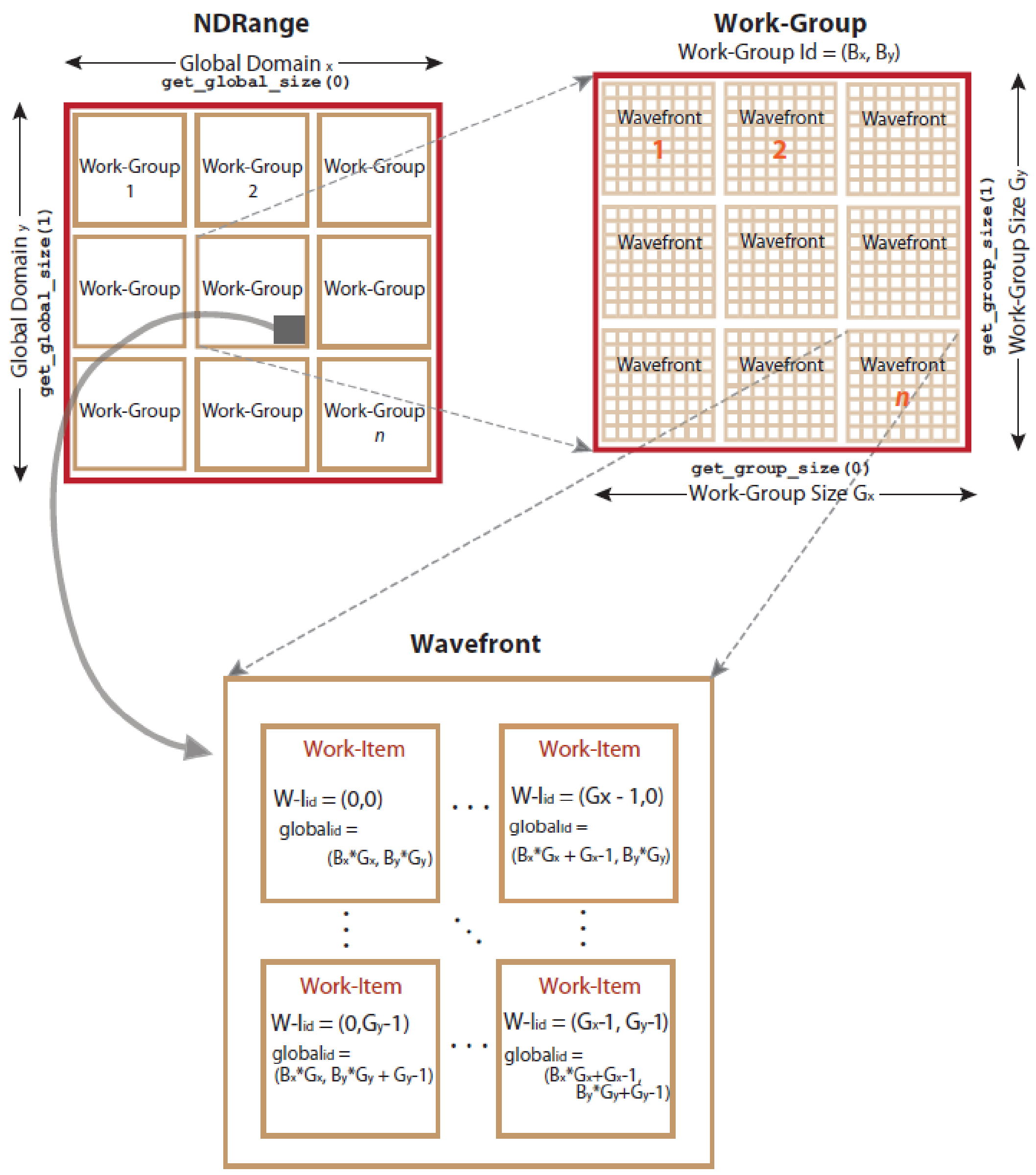

Figure 2.5. Visualization of a workgroup, wavefront and a workitem inside the NDRange [16].

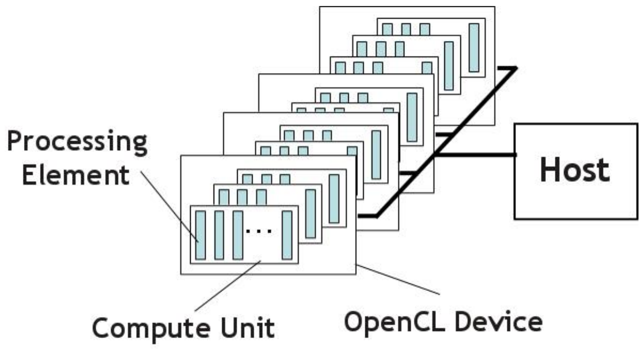

Figure 2.6. Host and device breakdown in OpenCL model [6].

Each work-item executes on a PE in the compute unit. Each workgroup is permanently assigned to a compute unit. There is no workgroup migration from one compute unit to other. Once assigned to a compute unit the workgroup evacuates only after completion. Wavefronts (64 work-items per wavefront for AMD devices) execute in parallel on the SIMD engine. Each work-item has a unique global thread and a local id that is unique inside a workgroup. In this thesis we control thermal issues by the scheduling of the workgroups to compute units having lower thermal stress and higher thermal reliability.

A wavefront scheduler schedules the wavefronts on the compute unit. The program counter for each work-item in a wavefront is always the same since it is a SIMD machine.

Data transfer has been an inherent problem for OpenCL model of execution. Introduction of 3D chips reduce the data transfer time (both inter device and intra device). OpenCL 2.x versions introduced the concept of shared virtual memory (SVM) which enables host and device to share the same virtual memory address space.

2.4 Workgroup scheduling in OpenCL devices

The OpenCL model schedules the workload at the granularity of workgroups irrespective of the device. This design helps local work-items (work-items in the same workgroup) to exploit local memory for high speed data sharing.

After the kernel is launched on the device each workgroup gets a unique id and is

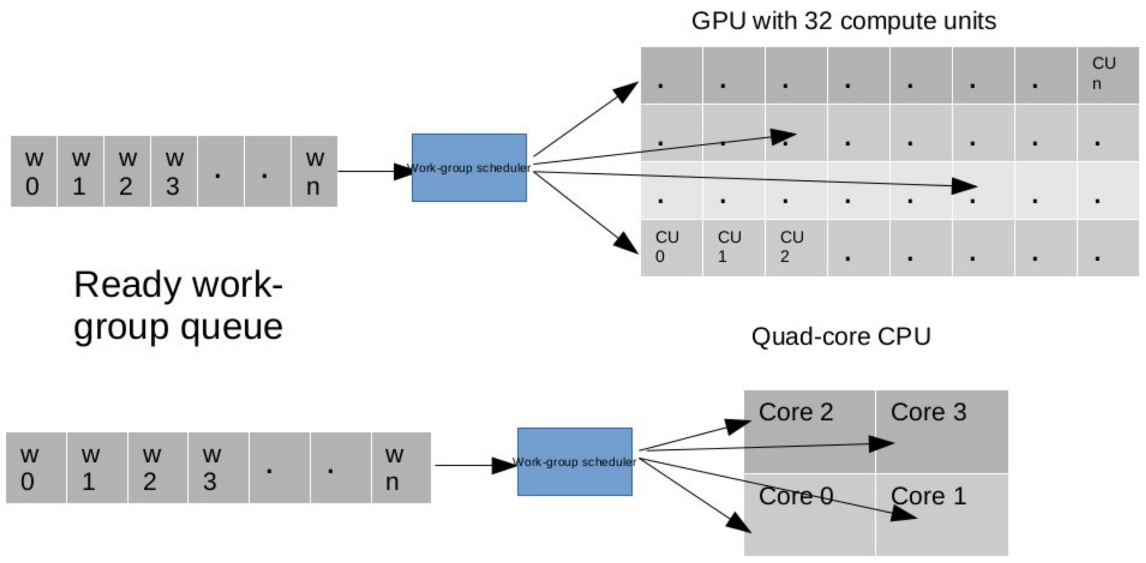

Figure 2.7. Workgroup scheduler block digagram for CPU and GPU.

added to the ready workgroup queue or gets a ready state. The workgroup waits in the ready queue until the scheduler, the operating system or the ultra thread dispatcher, schedules the workgroup to the next available compute unit or core. Figure 2.7 shows the ready queue and the processing cores/compute units.

The workgroups get equal priority and are scheduled to the compute units and cores using the FIFO (First In First Out) algorithm. We implement our proposed DTRM (Dynamic Thermal and Reliability Management) scheduling algorithm in the ultra thread dispatcher for the GPU and the operating system scheduler for the CPU.

CHAPTER 3

3D PROCESSORS AND THERMAL MODEL

One of the major challenges in designing heterogeneous system is to reduce the data transfer time, inter and intra device. Data transfer time has a direct correlation with the length of the interconnects used in the system. To reduce this data transfer time a 3D IC is an ideal solution, s three-dimensional integrated circuit (3D IC) is an integrated circuit has stacked silicon wafers or dies interconnected vertically by through-silicon vias (TSVs) so that they behave as a single device to achieve performance improvements at reduced power and smaller footprint than conventional two dimensional processes.

The most important functional part of a 3D IC is the interconnect network and its routing. In this section we discuss the most popular interconnect networks and see why TSV based 3D integration is the best for our applications.

3.1 3D interconnection networks and ICs

Enabling design in the vertical dimension permits a large degree of freedom in choosing an on-chip network topology. Due to interconnect length constraints and layout complications, the more conventional 2D ICs have placed limitations on the design of network interconnects. With the advent of 3D ICs, a wide range of on-chip network structures that were not explored earlier can now be explored. Feero et al 2009 [5] analyzed well-known 3D and 2D interconnect networks and found superior performance and efficiency of the 3D networks.

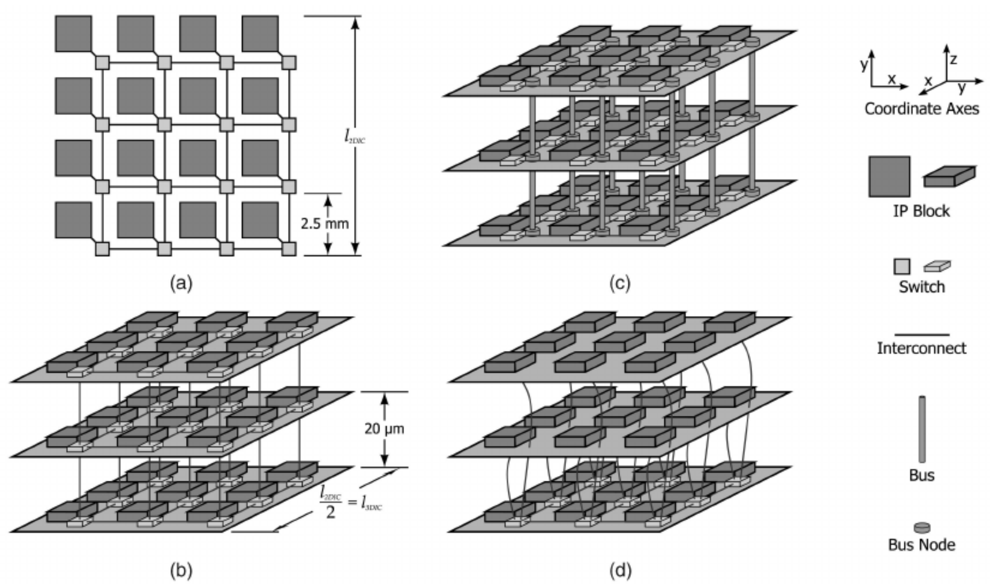

Figure 3.1. 3D interconnects [5].

Figure 3.1 compares the 3D interconnect mesh and 2D interconnect mesh, the difference in length of the network interconnects is significant. 2D Mesh, 3D Mesh, stacked Mesh and Ciliated Mesh are the most popular NoCs (Network on Chips) for system on chip architecture.

The most popular 2D NoC architecture is the 2D mesh architecture, the structure of the mesh is shown in Figure 3.1(a). The architecture consists of a matrix of m x n switches interconnecting the functional blocks. This is a very popular architecture due to its regular structure and inter-switch wires. Many 3 dimensional topologies have been derived from this network. A small extension of this type of interconnect is the 3D mesh NoC which is shown in the Figure 3.1(b). It uses a 7 port- switch one in each direction and one for the functional block. Another variation is a 3D Stacked Mesh as found in Figure-3.1(c). This leverages the short inter-layer distances that is the primary advantage in 3D chips, which are around 20μm. The 3D Stacked Mesh architecture is a combination of a packet-switched network and a bus network. Furthermore, each bus has only a small number of nodes, keeping the overall capacitance on the bus small and greatly simplifying bus arbitration. A third method of construction of a 3D NoC is by adding layers of functional blocks and adding the switches to only one layer or a small number of layers, like in the 3D Ciliated Mesh structure. The 3D Ciliated Mesh is a 4 x 4 x 2 3D mesh-based network with 2 functional blocks per switch, where the two functional blocks occupy almost the same footprint but are fabricated at different layers (Figure-3.1(d)). In a ciliated 3D Mesh network, each switch contains seven ports. This architecture will have a lower overall bandwidth than a complete 3D Mesh due to multiple functional blocks in switches and reduced connectivity[5]. we have chosen a TSV-based 3D integration, the primary reason of chosing this network is that our current thermal modeling tool (hotspot) accurately models this network.

3.2 3D heterogeneous processors and their thermal model

3D heterogeneous processors comprise a CPU (x86, ARM, MIPS etc.) which distributes the workload and one or more accelerator processors stacked on or under it. Stacking GPU over CPU reduces the data transfer time between the CPU and GPU making heterogeneous computing on 3D processors faster compared to 2D processors where interconnect lengths are much higher and by simply stacking more accelerators on the CPU the parallelism can be massively scaled [13]. The inherent problem with heterogeneous computing is the delay in the data transfer between devices significantly reduces the data transfer latency because of the reduced interconnect lengths as described in the previous section.

3D chips, though having great benefits, are also challenging to design and fabricate. Based on fabrication processes and mediums of vertical interconnects, 3D integration technologies can be basically classified as through silicon vias (TSV) based, monolithic and contactless. The **TSV-based 3D integration** connects multiple layers of active silicon using a through silicon vias, i.e. vertical metal interconnects. The **Monolithic 3D integration** technique serially grows the device layers on a conventional CMOS or silicon-on-insulator

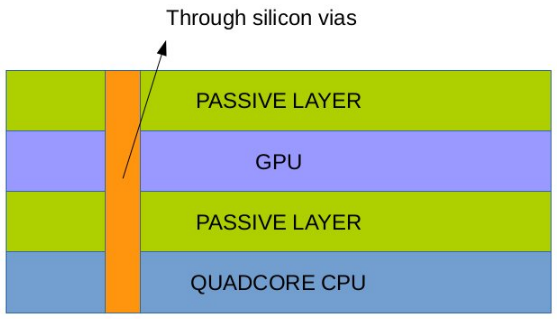

Figure 3.2. The cross-section of a heterogeneous processor.

plane. By employing a sequential fabrication process, monolithic 3D ICs allow the pitch of vertical interconnects smaller than TSVs. Therefore, monolithic 3D integration technology enables finer-grained stacking at the gate level or even transistor level. This technology is not so popular for heterogeneous architectures due to fabrication complexity. The **Contactless 3D integration** leverages the coupling of electric or magnetic fields for interlayer communication and the data transfer rate is comparable with TSVs. Monolithic integration can lead to up to 8% smaller area due to smaller size of inter-tier vias than the TSVs, 12% longest path delay and 7% lower power [11]. But due to fabrication complexity of monolithic and contactless 3D integration a TSV 3D integration is the best choice for our 3D processor. Our 3D processor includes a quad-core CPU with x86 architecture at the bottom level then a layer of passive material and then a southern islands GPU above that and then again a passive layer above that. Figure 3.2 shows the cross-section of our modelled 3D processor.

We exploit the heterogeneity of this processor through the OpenCL programming model. We treat both processors as separate devices and launch the same kernel on both devices. For work that we have done till now, we divided the total workload without any profiling and pre-performance evaluation to CPU and GPU. The project focuses only on thermal management of 3D processors and performance evaluation remains as a future work.

The inherent problem of high power density in the 3D chips demands a very good thermal management system. Figures 3.3 and 3.4 show the heat maps of the CPU and GPU after the execution of Matrix Multiplication on a 2D processor. Core 1 of the CPU

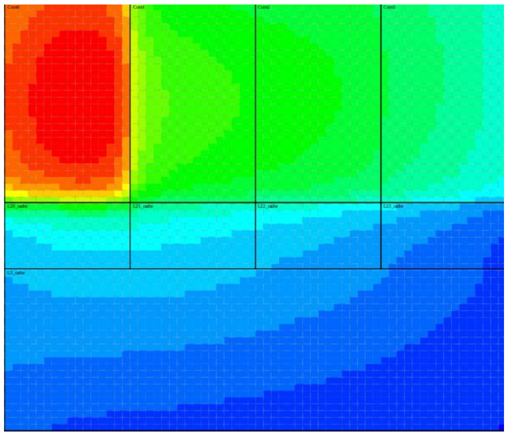

Figure 3.3. Quadcore CPU heat map (Matrix Multiplication) generated by our framework.

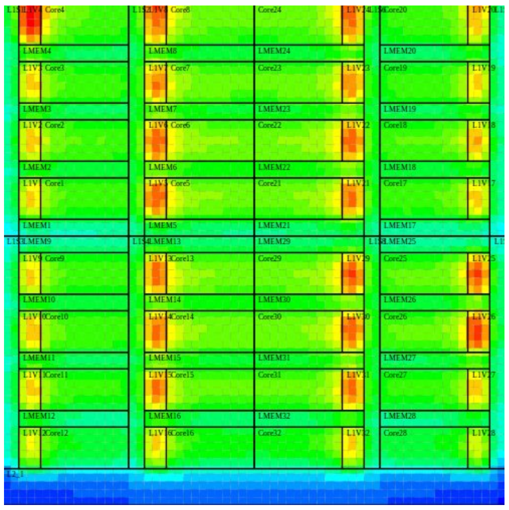

Figure 3.4. GPU heat map (Matrix Multiplication) generated by our framework.

is hotter than others because the host program runs on the first core in addition to the OpenCL work-items. After the superimposition of the processor there is significant vertical heat interaction.

3.3 Related work

Several works have been done in dynamic and static thermal reliability management of 3D multi-core processors at both software and hardware level but not a lot has been done for 3D heterogeneous processors. *Coskun et al 2009* [3] proposed several dynamic job scheduling techniques. They introduce a probabilistic policy that computes the probabilities of sending workload to cores at each interval based on an analysis of the temperature history on the chip. The probability values are decremented or incremented depending on whether the at 75°C in order to avoid allocating workload to cores that have temperatures slightly below 80°C. They sample the temperature every 10 ms and after every 10 samples the average temperature of the cores in the last 10 samples are calculated. This is the probability function used by

Coskun et al. to calculate the core probabilities.

$$Pn = P0 + -W$$

$$Pn = NewProbability$$

$$P0 = PreviousProbability$$

W is the weight or the amount by which the probability is changed depending on the temperature from last sampling environment.

$$W = \frac{\beta}{Avth}$$

Liu et al 2010 [12] proposed a static task scheduling algorithm where the algorithm assigns hot jobs to the cores close to the heat sink and cool jobs to the cores far from the heat sink. Due to this fact that there is a strong thermal correlation between two adjacent layers and the cores in one stack have only a small difference in temperatures. Therefore, the hottest/coolest core stack will have the largest temperature drop/rise, which may lead to temperature oscillations and task thrashing between those two stacks, potentially leading to more thermal emergencies. *Wang et al 2010* [21] proposed the technique where they consider the delay in transfering the data and the thread from one core to other core. The scheduling algorithm uses both temperature and distance from hottest core to shcedule the hot-thread. if the core is far away from the hot core then the copying latency increases so we choose a less cooler core but closer to the hot core to reduce delay/copying latency. None of the above works are for heterogenous processors having CPU-GPU stacked and hence we address this problem with our temperature aware dynamic scheduling policy.

Zhou et al. 2010 [23] proposed a technique where they treat vertically adjacent cores as a super core for scheduling because vertically adjacent cores have strong thermal correlation. The biggest drawback is this technique will only work for a homogeneous floor plan which limits its application.

Yin et al. 2011 [21] proposed an algorithm to reduce core temperature as quickly as possible through making the core which has a temperature violation be idle in next clock tick. It is achieved by migrating the tasks according to the relative positions of each core. In a quad-core Processor, the scheduler migrates the task in the clockwise or counter-clockwise direction or opposite idle core .

Zhao et al. 2013 [22] propose a thread migration algorithm that the hottest and the second coldest threads switch places and the second hottest and the coldest also switch in the 4 core processor only when the variance is large enough. The major problem with this technique is that it degrades the performance of the processor because there is an over-head of transfering a thread from one core to other which also involves the transfering of data from L1 cache of previous core to the later.

CHAPTER 4

EXPERIMENTAL SETUP AND OUR PROPOSED DTRM SCHEDULER

4.1 Experimental setup

Our thermal modeling framework is divided into three sections. The first section simulates different benchmarks from AMDAPPSDK [17] on a well known cycle accurate simulator, Multi2sim [19]. We modified Multi2sim to generate the required statistics to generate power consumption data from McPAT [10]. Two different versions of McPAT was modified, one for CPU and other for GPU, to generate cycle accurate power consumption data. This cycle accurate power consumption data was then parsed into a csv file for Hotspot [18], a popular tool for thermal modelling, to take in as input.

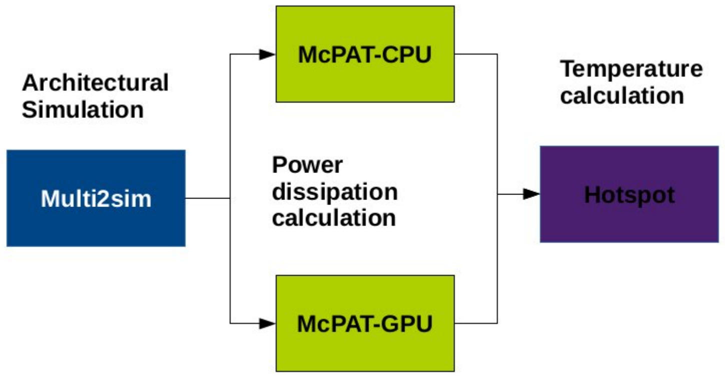

Figure 4.1. Thermal modeling framework for CPU-GPU 3D processors.

4.1.1 Architectural simulation

Multi2Sim is a simulation framework for CPU-GPU heterogeneous computing written in C. It includes models for superscalar, multi-threaded, and multi-core CPUs, as well as GPU architectures. In this section we present an introduction to Multi2Sim, and show how to perform basic simulations and extract performance results [19].

We used Multi2sim 4.2 non-developer version for simulating benchmarks and getting the runtime statistics at every 925 cycles (we will refer this as the sampling interval). The reason to choose 925 cyle was because the frequency of our GPU was 925 MHz.

Processors in Multi2sim can be configured by changing configuration files which are in the .ini format. This file has a key-value structure, key being the functional block (e.g., core) and value usually being a number or boolean (e.g., 4).

We configured the host CPU as a quad-core x86 CPU with three levels of cache and the GPU as a Southern Island architecture of 32 compute units with a cluster size of 8 compute units.

The cycle accurate hardware statistics are generated using the timing model of CPU and GPU in the simulator. The particular files that we modified were cpu.c and gpu.c. We added the function **X86CpuXmlInit (X86Cpu self, long long uop_stats,char prefix, int peak_ipc)** that records the statistics every 925 cycles and then dumps it into a XML file with the cycle name on the file. The XML file needed by the McPAT requires to have usage statistics of all functional blocks in it so we imported the memory modules and list modules in the cpu.c file to get the usage statistics of **mod_t** type (structure used by Multi2sim for memory modules) data structures.

Table 5.1 shows the TLB configuration details of our modeled CPU. We fine-tuned the trade-off between size and latency values by performing multiple dummy simulations with different cache configurations (e.g., varying set, bank, line size). Table 5.2 shows the configuration details of different cache level and their latency numbers which directly affect the execution time. These are the numbers we got after fine tuning the cache configurations to resemble with our modeled processor in Multi2Sim.

Table 4.1. CPU TLB configuration.

Charateristic	Instruction TLB	Data TLB	Second level TLB
Size	128	64	512
Associativity	4-way	4-way	4-way
Replacement	Pseudo-LRU	Pseudo-LRU	Psuedo-LRU
Access Latency	1 cycle	1cycle	6cycles
Miss Latency	7cycles	7cycles	Hundreds of cycles

McPAT requires the XML file to have all the functional blocks in it, some of these are not modeled by Multi2sim so we hard-coded these values to reasonable numbers. A major challenge was file management. Since simulation generated more than a million XML files in every simulation which often threw the simulating system into a deadlock.

4.1.2 Power calculation

We used McPAT, an integrated power, area, and timing modeling framework that supports comprehensive design space exploration for multi-core and many-core processor configurations. At the micro-architectural level, McPAT includes models for the fundamental components of a chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, integrated memory controllers, and multiple-domain clocking. At the circuit and technology levels, McPAT supports critical-path timing modeling, area modeling, and dynamic, short-circuit, and leakage power modeling for each of the device types forecast in the ITRS roadmap including bulk CMOS, SOI, and doublegate transistors. McPAT has a flexible XML interface to facilitate its use with many performance simulators

Table 4.2. CPU cache configuration.

Charateristic	L1	L2	L3
Size	32 KBI/32 KB D	256KB	2 MB per core
Associativity	4-wayI/8-way D	8-way	16-way
Access Latency	4 cycles,pipelined	10 cycles	35 cycles
Replacement	Pseudo-LRU	Pseudo-LRU	Psuedo-LRU

[10].

We modified McPAT to read the cycle interval files one by one and generate power consumption results for the functional blocks in it. First we used the latest non-developer version which had new bugs so we used an older version.

Using the XML files containing the usage statistics, McPAT calculates the power consumption for each block in the input XML file. The power consumed by each operation is fixed, for example, a L1 cache read consumes the same amount of power every time. This power consumed by a single operation is then multiplied by the amount of times the operation was performed. Therefore:

$$power_{L1-cache-reads} = N_r * P_{L1-read}$$

$$N_r = number\ of\ cache\ reads$$

$$P_{L1-read} = power\ consumed\ per\ cache\ read$$

The power consumed per cache read is determined by the number of transistors and internal structure of the cache. The physical parameters used by us in the processor are listed in the table below.

4.1.2.1 Power calculation in silicon devices

In silicon transistor devices, the power consumption is composed of two types of components: static power and dynamic power. So:

$$P = P_{static} + P_{dynamic}$$

Static power is generated by the leakage current while the dynamic power is generated by the switching of transistors. Total dynamic power of the device can be calculated by adding the individual powers of the components:

Table 4.3. Technology parameters for modeled CPU.

Parameter	Available Options	GPU
Clock frequency	in unit of Hz	3.33GHz
Feature size	16nm to 180nm	65 nm
Core type	Out-of-order, in-order	In-order
Embedded	True, false	False
Interconnect projection	Aggressive, conservative	Aggressive
Component type	Core, uncore	Core
Device type	High-performance type, low standby power type, low operating power type	High-performance type
Homogeneous cores	1 means all cores are the same, 0 means heterogeneous	1
Number of cache levels	Number of cache levels	3
Number of cores	Number of cores	4

$$P_{total} = \sum P_i$$

Where i represents the i^{th} component of the device. The dynamic power of each component is calculated in a bottom up fashion. This means that McPAT calculates the power at a basic circuit level and sums it up to get the components power. Dynamic power dissipation of CMOS circuits is computed by:

$$Power = \alpha C V_{dd} \delta V f_{clk}$$

Here C is the total load capacitance, V_{dd} is the supply voltage, δV is the voltage swing during switching, and f_{clk} is the clock frequency of the processor. C depends on the physical material used, doping and circuit level parameters. Here α is the activity factor and is usually determined by the frequency of the switching of the circuits.

4.1.3 Temperature analysis

For temperature analysis we used the well known temperature analysis tool called Hotspot [18]. We used the latest version of the tool with 3D stacking capability added by Coskun et al 2010 [4]. Hotspot requires four input files from the user for temperature trace generation. First is the power trace file which contains the power consumed by each component in each sampling interval. The format of this file must be csv since Hotspot has discontinued support for MS Excel sheets. The second file needed is the Layer Configuration File which contains the information of every layer in the 3D model and the file names of the floorplans of CPU, GPU and through silicon vias. The third file is the Hotspot.config files which contains the circuit level specification of the silicon devices (capacitance, specific resistivity, etc). This is the file where we specify the interval length of the power trace file and the initial temperatures for functional blocks if a separate file for it is not specified. The fourth file contains the initial temperatures of the components of the chips. Hotspot uses these as the initial temperature of the components in the chip layout and computes the new temperatures of the components in the chip.

We parse the file output by the McPAT and restructure it into a Hotspot input file format. We then plug in this file to Hotspot 6.0 [18] which generates the temperature of different functional units in the processors (e.g., core, L1, L2, L3 etc.). We didn't make any changes to Hotspot and used its 3D stacking capability to generate the transient temperatures.

4.2 Thermal-aware task scheduling algorithm and probability calculation

In this work we only target the thermal management of the hardware components irrespective of performance slowdowns. Performance tuning is an aspect of the future work for the thesis. We have employed a temperature aware scheduler which leverages the temperature history of

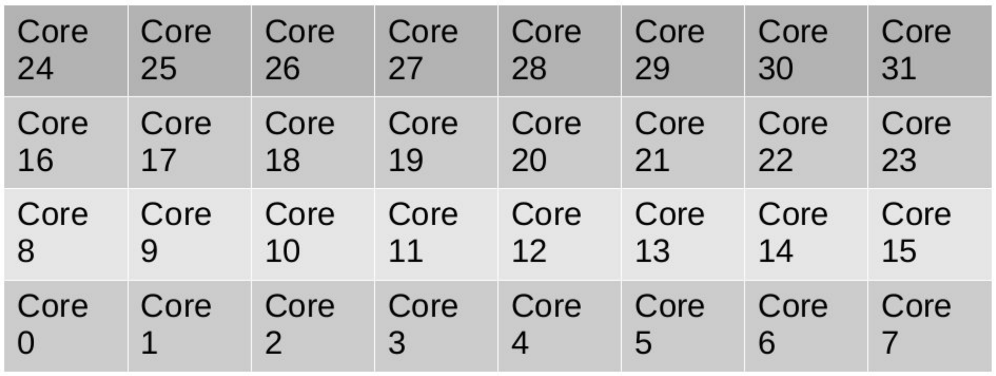

Figure 4.2. Floorplan of GPU in the upper level.

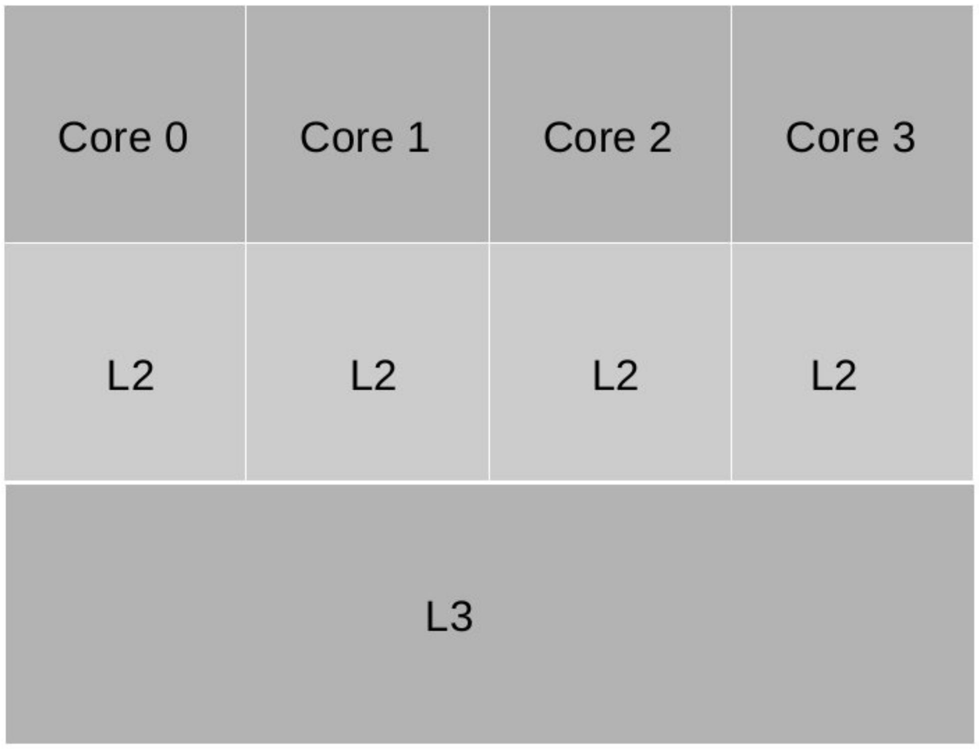

Figure 4.3. Floorplan of CPU at the bottom level.

the cores and other components to calculate the better cores for scheduling the workgroups. The reason behind using the temperature history is to reduce the thermal stress of the components. Higher temperatures in the history of the cores suggest higher thermal stress and surges the likelihood of device failure [2]. Dynamic temperature aware scheduling in a heterogeneous environment is challenging since scheduling has to be controlled seperately for CPU and GPU. The CPU scheduling is controlled by the operating system while the GPU scheduling is controlled by the hardware. The idea of making a super scheduler which could assign work to both CPU and GPU with the same workload became out of the question. To overcome this problem we made an offline scheduler which works between McPAT and Hotspot in our framework. We designed the same scheduling algorithm for both CPU and GPU.

We use a probabilistic approach for assigning work to the scheduler. Our scheduler schedules a workgroup to a compute unit and then reads the temperature in each sampling interval (10μs), it then updates the probabilities of the cores/compute units to get a new work group.

We calculate the probability of these cores by using their temperature history, thermal stress and geometric location of the core on the die. We choose temperature history and thermal stress to model our probability because cores with high temperature history are more likely to get hot than cores that have low temperature history. Cores with high temperature history have higher thermal stress and are more likely to have hotspots. (for example a core at the top layer near the heat sink will have low temperature history while a core at the bottom layer and away from heat sink will have low temperature history).

4.2.1 Probability calculation of cores

We calculate the probabilty of a core by the following function:

$$P_t = P_{t-1} + \Delta P_{inc/dec} \tag{4.1}$$

Δ P is calculated in the following way:

$$\delta T = (Tpref - Tavg)$$

$$\Delta P_{inc} = \eta_i * \delta T * k_i \tag{4.2}$$

$$\Delta P_{dec} = \eta_d * \delta T * k_d \tag{4.3}$$

Equations 4.2 and 4.3 are the most important equations as they calculate the change in probability of a core by using the change in temperature.

η is the position constant of the core/compute unit. For example, if a core is located at the center of the of the chip then it is more likely to become a hotspot due to heat interaction with the neighboring cores than a core located at the corner of the processor and near the heat sink.

Putting it together the change in probability is correlated to the change in temperature and position of the core in the 3D chip.

η values varies from 0.5 to 1 depending on the location of the core. Lesser values for cores at the corners and higher for cores at the center.

After we calculate the probability, we assign the workgroups to the "cool cores." We monitor the temperature of the cores every 10μs and update the probabilities.

In the OpenCL model we cannot exempt a workgroup from a core/compute unit in the middle of the workgroup execution. Hence there is no possibility of thread migration in the OpenCL model for both CPU and GPU. In the meantime, if the temperature of a core rises to an alarming temperature (85°C) then we stop the execution on the core untill the temperature comes below the operating temperature range ($<$ 75°C), once the core gets into the operating temperature the workgoup execution is resumed from the point where it was stopped.

4.2.2 Our proposed DTRM algorithm

Below is our proposed DTRM scheduler algorithm, this is an accurate abstract structure of the code as well.

```
#OUR PROPOSED DTRM SCHEDULING ALGORITHM
for each sampling interval:
    evacuateCores()

    sensor.getTemperatures(CPU,GPU)

    GPU.calculateProb()
    CPU.calculateProb()

    gpu_cool_cores = GPU.getCoolCores()
    cpu_cool_cores = CPU.getCoolCores()

    for cores in gpu_cool_cores:
        GPU.assign(gpu_cool_cores.pop(i).core_id)

    for cores in cpu_cool_cores:
        CPU.assign(cpu_cool_cores.pop(i).core_id)
```

The above snippet is from our proposed DTRM scheduler, we used a for loop to keep tabs on the temperature at every sampling interval. The algorithm first checks if any core in CPU or GPU has completed execution and then adds them to available core queue. In a physical processor the temperature will be sensed by a sensor while here we use Hotspot to get the temperature at every interval. In the above code the sensor.getTemperature(CPU,GPU) function calls the hotspot and gets the temperature for the particular interval. Using these temperatures we calculate the probability for our cores with the function CPU.calculateProb() and GPU.calculateProb(). These functions use equation 4.1 to calculate the probability for the cores. The CPU.get_cool_cores() and GPU.get_cool_cores() function returns a list of available cores sorted in the order of their probabilities.

CHAPTER 5

RESULTS

In this chapter we analyze the results in two aspects:

1. Framework for power and thermal model of 3D heterogeneous processors

2. Our proposed DTRM (Dynamic Thermal Reliablity Management) scheduler

We first visualize and analyze the results from our framework and then show the results from our proposed DTRM scheduler.

5.1 3D thermal model

We simulated several benchmarks from AMDAPPSDK [17] on our framework and visualized the power and temperature trends of different benchmarks. We found strong temperature correlation across layers. Figures 5.1 and 5.2 show the temperature trend of the cores of the CPU in 2D and in 3D processors for a Matrix Multiplication benchmark.

Comparing the temperatures between 2D and 3D processors for the same benchmark (Matrix Multiplication) we see that the temperatures of 2D processor never goes to an alarming point whereas in the 3D processor temperature goes to an alarming temperature after 30% of benchmark execution is complete. In both experiments we used DVFS (Dynamic Voltage and Frequency Scaling) as our thermal management technique.

A similar temperature trend is observed for the GPU where power consumption of the cores are lower. Figure 5.3 and 5.4 show the temperature trends for a 2D GPU and then

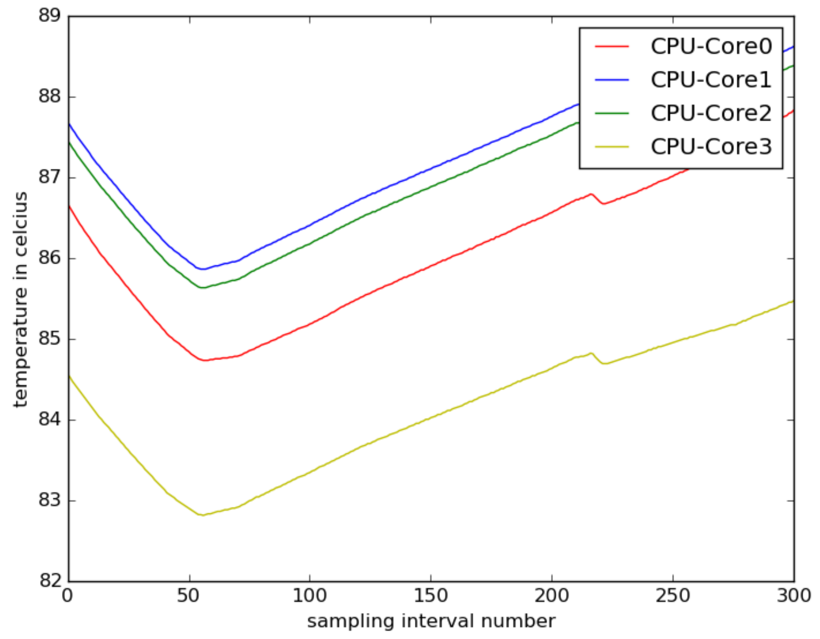

Figure 5.1. Temperature trend of CPU cores for MatrixMultiplication in a 2D processor.

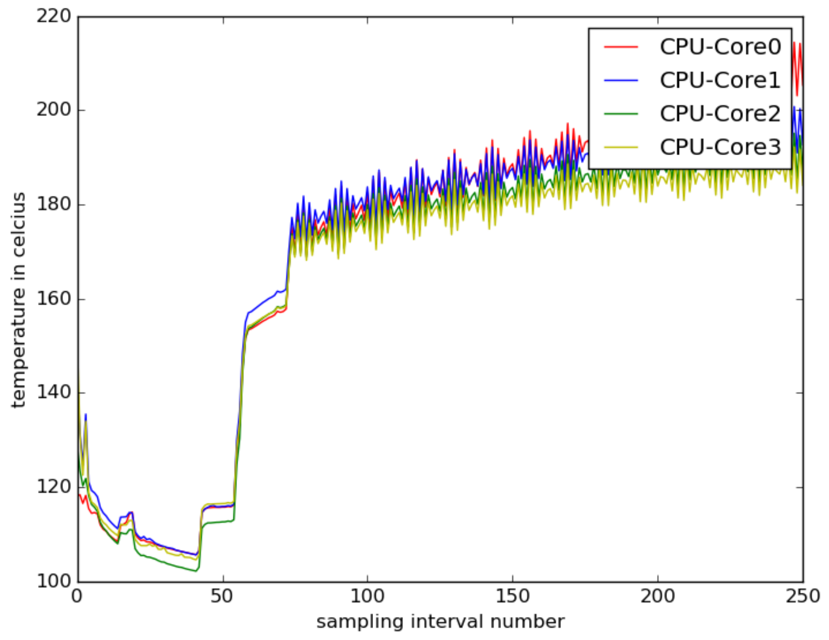

Figure 5.2. Temperature trend of CPU cores for MatrixMultiplication in a 3D processor.

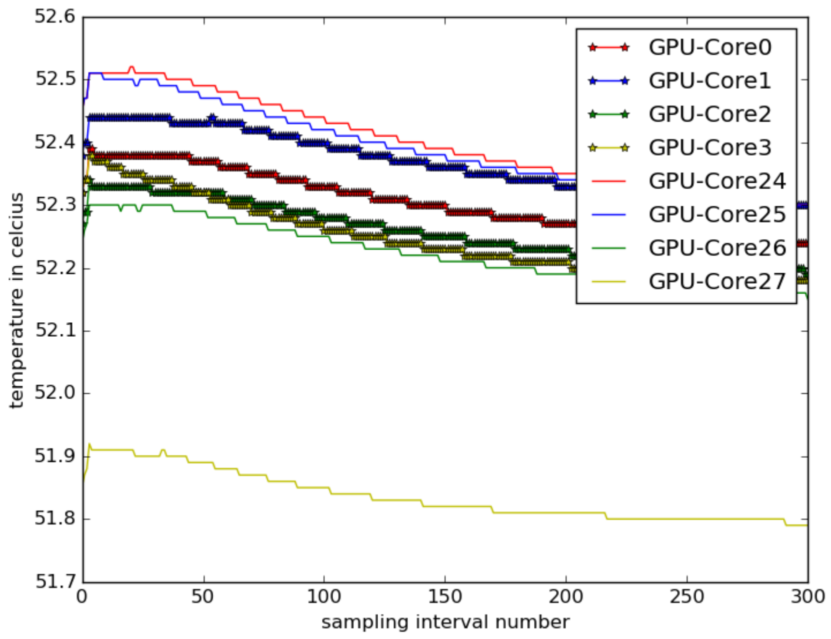

Figure 5.3. Temperature trend of GPU cores for MatrixMultiplication in a 2D processor.

a GPU embedded in the second layer of the 3D processor. We can see the temperatures of cores right above the cores of CPU follow the same temperature trend (Figures 5.2 and 5.4). Cores 24-25 geometrically lie directly above core 0 of the CPU and due to vertical heat transfer the temperature trend is similar. The temperatures of the GPU chip in our 3D processor is lower than CPU because it is closer to the heat sink and also it consumes significantly less energy than the CPU. Hence addressing the vertical temperature correlation is more important than addressing horizontal core correlation.

We can clearly observe the need for a good dynamic thermal management technique which addresses the temperature correlation effects caused by the position of the cores in 3D processors.

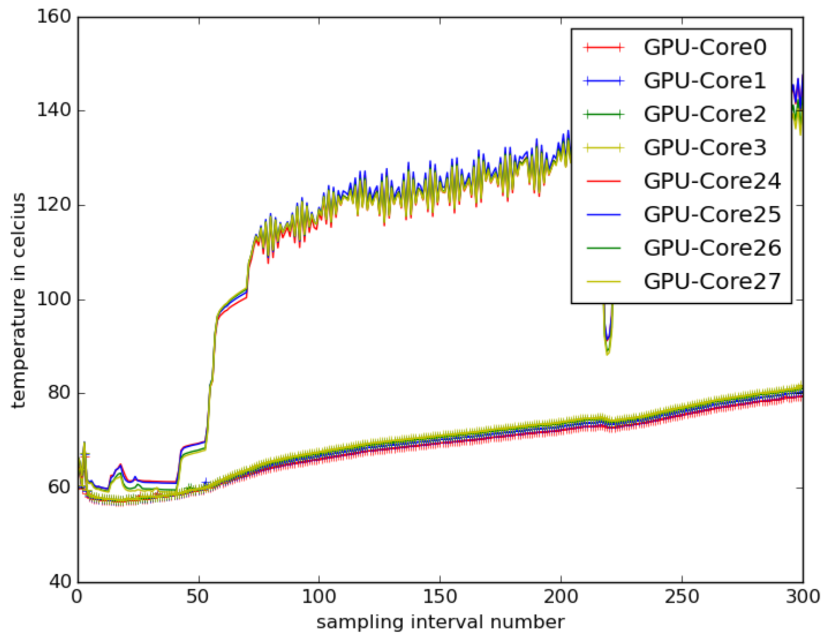

Figure 5.4. Temperature trend of GPU cores for MatrixMultiplication in a 3D processor.

5.2 The impact of temperature aware DTRM

In this section we compare our proposed DTRM scheduler against the existing schedulers in 2D and 3D heterogeneous processors. Both our proposed DTRM scheduler and the first available core scheduler use FIFO as their prioritizing technique for workgroups but our scheduler picks the core under least thermal stress as the core to schedule the work group where as current CPU-GPU workgroup scheduler schedules the job to the first available core. In section 4.3 we explained the working of proposed DTRM scheduler and in this section we present and analyze the results of our scheduler.

Figure 5.5 shows the temperature variations in core 0 of CPU. We set the initial temperatures of the chip very high so we have existing hotspots. We reduced the max temperature during execution to less than 90°C which is a safe operating temperature compared to 200 degrees Celsius without our DTRM scheduler (Figure-5.2).

In Figure 5.6 we show the temperature trends of all CPU cores and we can see they

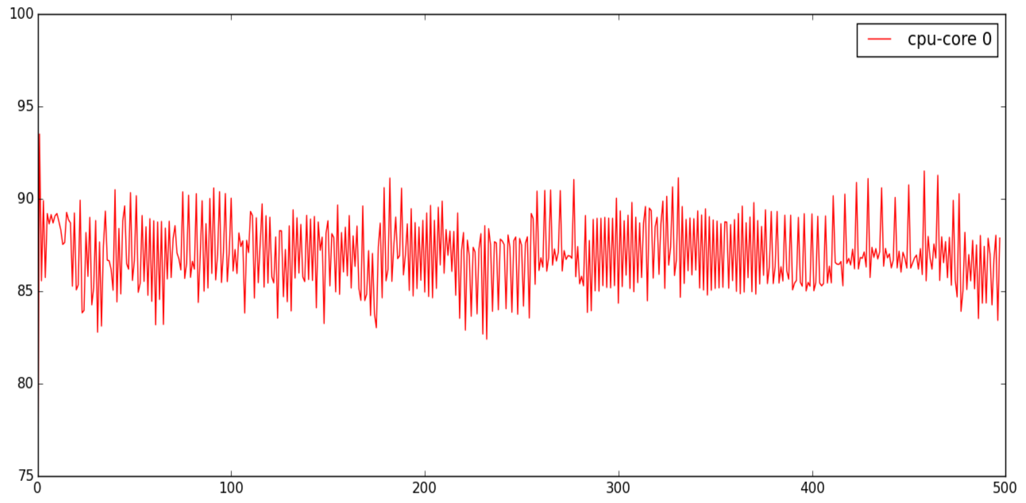

Figure 5.5. Temperature trend of CPU core 0 for MatrixMultiplication in a 3D processor with our DTRM scheduler.

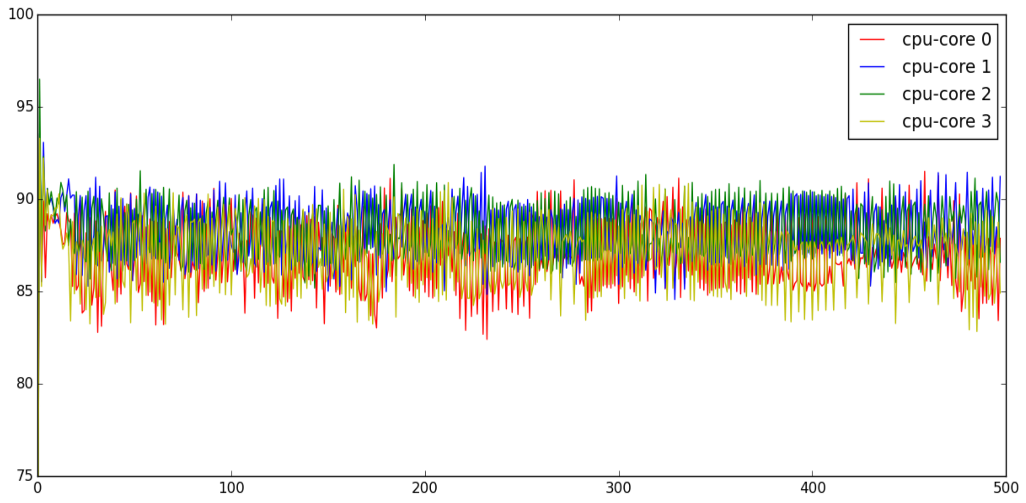

Figure 5.6. Temperature trend of CPU cores for MatrixMultiplication in a 3D processor with our DTRM scheduler.

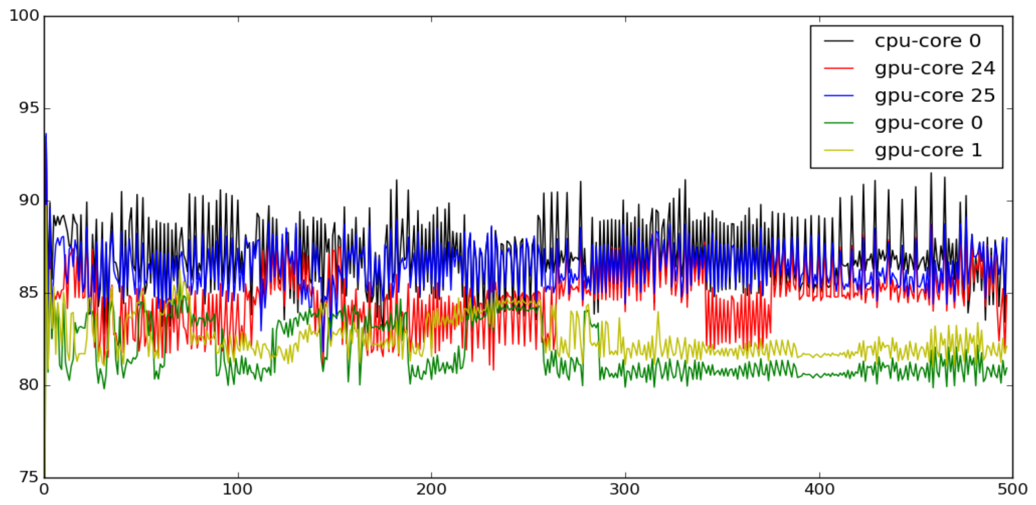

Figure 5.7. Temperature trend of GPU cores vs core 0 of CPU for MatrixMultiplication in a 3D processor with our DTRM scheduler.

are almost random but all are operating in the same range of temperature (80-90 degree Celsius). The most important factor that we observe here is that the spatial temperature gradient has a very low value.

In Figure 5.7 we can see the vertical correlation of the cores again (CPU-core 0 and GPU cores 24, 25), the portions where the temperature correlation is broken. We can safely say it is where the GPU core was not scheduled a workgroup. GPU cores 0 and 1 lie above the L3 cache of CPU and hence there temperature trend is much different from core 24 and 25.

CHAPTER 6

CONCLUSION

Although CPU-GPU 3D heterogeneous processors have numerous benefits and potential, there has been few research on their thermal modeling and dynamic management technique. Once the research on 3D heterogeneous processors gains momentum the heterogeneous architecture community will need a robust power and thermal modeling framework to address the thermal and reliability concerns. Our framework fulfills this need and opens the door to more interesting ideas of dynamic temperature aware workgroup scheduling. We verified that vertical temperature correlation between layers is much higher than horizontal temperature correlation. With our proposed DTRM (Dynamic Thermal and Reliability Management) scheduler we recorded fewer hotspots and better temperature gradients in our heterogeneous CPU-GPU 3D processor.

We developed an accurate thermal modeling framework for 3D heterogeneous processors which can be used by the architectural community. We also developed a DTRM scheduler which can be used as benchmark for schedulers designed in the future under this work. Performance concern caused by selective scheduling remains as future work.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] JEDEC Solid State Technology Association et al. Failure mechanisms and models for semiconductor devices. *JEDEC Publication JEP122-B*, 2003.

[2] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 469–479, Dec 2006.

[3] A. K. Coskun, J. L. Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici. Dynamic thermal management in 3d multicore architectures. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 1410–1415, April 2009.

[4] Ayse Kivilcim Coskun, T# 138 Rosing, Keith A Whisnant, and Kenny C Gross. Static and dynamic temperature-aware scheduling for multiprocessor socs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(9):1127, 2008.

[5] B. S. Feero and P. P. Pande. Networks-on-chip in a three-dimensional environment: A performance evaluation. *IEEE Transactions on Computers*, 58(1):32–45, Jan 2009.

[6] Khronos OpenCL Working Group et al. The opencl specification. *Version*, 1(29):8, 2008.

[7] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[8] W-L Hung, Greg M Link, Yuan Xie, Narayanan Vijaykrishnan, and Mary Jane Irwin. Interconnect and thermal-aware floorplanning for 3d microprocessors. In *7th International Symposium on Quality Electronic Design (ISQED'06)*, pages 6–pp. IEEE, 2006.

[9] H. Kufluoglu and M. A. Alam. A unified modeling of nbti and hot carrier injection for mosfet reliability. In *Computational Electronics, 2004. IWCE-10 2004. Abstracts. 10th International Workshop on*, pages 28–29, Oct 2004.

[10] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM, 2009.

[11] Chang Liu and Sung Kyu Lim. A design tradeoff study with monolithic 3d integration. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pages 529–536. IEEE, 2012.

[12] S. Liu, Jingyi Zhang, Qing Wu, and Qinru Qiu. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 390–398, March 2010.

[13] N. Miura, Y. Koizumi, E. Sasaki, Y. Take, H. Matsutani, T. Kuroda, H. Amano, R. Sakamoto, M. Namiki, K. Usami, M. Kondo, and H. Nakamura. A scalable 3d heterogeneous multi-core processor with inductive-coupling thruchip interface. In *Cool Chips XVI (COOL Chips), 2013 IEEE*, pages 1–3, April 2013.

[14] K. Puttaswamy and G. H. Loh. Thermal herding: Microarchitecture techniques for controlling hotspots in high-performance 3d-integrated processors. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 193–204, Feb 2007.

[15] Karthik Sankaranarayanan, Sivakumar Velusamy, Mircea Stan, Kevin Skadron, et al. A case for thermal-aware floorplanning at the microarchitectural level. *Journal of Instruction-Level Parallelism*, 7(1):8–16, 2005.

[16] AMD Staff. http://docplayer.net/3122009-accelerating-sequential-computer-vision-algorithms-using-openmp-and-opencl-on-commodity-parallel-hardware.html, 2014.

[17] AMD Staff. Opencl and the amd app sdk v2. 4, 2014.

[18] Mircea R Stan, Kevin Skadron, Marco Barcella, Wei Huang, Karthik Sankaranarayanan, and Sivakumar Velusamy. Hotspot: A dynamic compact thermal model at the processor-architecture level. *Microelectronics Journal*, 34(12):1153–1165, 2003.

[19] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2sim: a simulation framework for cpu-gpu computing. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 335–344. ACM, 2012.

[20] A. Venkat and D. M. Tullsen. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 121–132, June 2014.

[21] X. Yin, Y. Zhu, L. Xia, J. Ye, T. Huang, Y. Fu, and M. Qiu. Efficient implementation of thermal-aware scheduler on a quad-core processor. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1076–1082, Nov 2011.

[22] Dali Zhao, H. Homayoun, and A. V. Veidenbaum. Temperature aware thread migration in 3d architecture with stacked dram. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on*, pages 80–87, March 2013.

[23] X. Zhou, J. Yang, Y. Xu, Y. Zhang, and J. Zhao. Thermal-aware task scheduling for 3d multicore processors. *IEEE Transactions on Parallel and Distributed Systems*, 21(1):60–71, Jan 2010.

VITA

- Ajay Sharma Address - 13203-C, 900 whirlpool drive, Oxford MS, 38655
  email - asharma5@go.olemiss.edu — phone - 6623803521

- Professional summary I am a graduate student at the University of Mississippi seeking full time opportunities starting from December 2016. Along with my graduate studies I worked as a Graduate Teaching Assistant and successfully completed two internships which helped me develop my technical and non-technical skills. . Skills Highlights

  – Programming Languages: Core Java, J2EE(STRUTS, JSP, SERVLETS), C++, C, Python,JavaScript, HTML

  – Web development: JSP, Servlets, Struts, Web-Logic server

  – Back-end: SQL, Oracle-SQL-Developer

  – Tools: Github, GDB, gcc/g++, Makefiles

- Experience

  – University of Mississippi Teaching Assistant Oxford, MS [January 2016 Present] Gave one-on-one tutoring to students in the classroom program. Evaluation of code quality and correctness in Java Projects.

  – FedEx Services Summer Intern - Collierville, TN [May 2016 - August 2016] Successfully developed and optimized web-app to improve the safety of an aircraft. Met the deadline for the project with detailed documentation and reports I used state of the art technologies (JSPs, Servlets, Struts..) to achieve faster and better user experience.

  - University of Mississippi Research Assistant - Oxford, MS [July 2015 December 2015] Developed a framework for power and thermal modeling of heterogeneous processors.
  - Jabil Circuits Summer Intern - Memphis, TN [June 2015 July 2015] Developed java module for automatic optimal volume packaging of delivery boxes in a container. I delivered the project in time with detailed reports on implementation and design.

- Education
  - Graduate student at the University of Mississippi [August 2014 - Present] (Expected graduation - December 2016) Major: Computer Science [GPA 3.93] Important Courses taken: Java advanced Advanced Algorithms Design and Analysis Machine Learning Advanced Operating Systems
  - Bachelor of Technology from NIT Surat [July 2009 May 2014] Achievements A percentile of 99.1 in AIEEE 2009 (All India Engineering Entrance Exam)-2009 where 1.1 million candidates participated. Scholar in National Cyber Olympiad, National Science Olympiad and National Math Olympiad.