A Detailed Study of Modern Multicore Architectures

N Sai Krishna
Roll No.44
Section D
Registration No. 150905206
Manipal Institute of Technology

Siddhartha Rao Kamalakara
Roll No.37
Section D
Registration No. 150905160
Manipal Institute of Technology

Pranjal Paliwal
Roll No.34
Section D
Registration No. 150905142
Manipal Institute of Technology

Sumanth C
Roll No.09
Section D
Registration No. 150905036
Manipal Institute of Technology

Madhur Agarwal
Roll No.06
Section D
Registration No. 150905107
Manipal Institute of Technology

***Abstract*— This report presents a detailed study on multicore architectures. The need, evolution and the challenges faced in the multicore architecture design have been described in detail. A case study on one of the multicore chips IBM Power5 is presented. A brief study on GPU and TPU has been presented and the architecture of NVIDIA's TESLA is shown. A performance comparison of CPU and GPU on a discrete heterogeneous architecture is presented. A brief survey of challenges faced in exploiting multicore architectures has been shown.**

TABLE OF CONTENTS

1. INTRODUCTION

With the advent of computationally intensive applications, the need for performance has increased exponentially.This led to the development of multi core architectures. A multi-core processor is a single computing component with two or more independent actual processing units (called "cores"), which are units that read and execute program instructions.

The section 'Evolution and Challenges Faced ' briefs on evolution of multi-core processors followed by introducing the technology and its advantages in todays world and concludes by detailing on the challenges currently faced by multi-core processors. The section ' Factors Affecting Performance of Multi-Core Processors ' presents analysis of various parameters affecting the performance of Multi-core Architectures. Analysis of performance has been done by varying the number of cores, changing L2 cache size, varying directory size from 64 to 2048 entries on a 4 node, 8 node 16 node and 64 node Chip multiprocessor. The section ' Case Study on IBM Power5 ' provides a case study on the IBM Power5 Chip which was produced in 2004. This section details the system structure of Power5 Chip and provides a comparison with the IBM Power4 chip. The section ' Design for Multicore Architectures ' describes how multicore architectures are reigning the microprocessor design because of performance, thread level parallelism, and thermal scaling. The section ' GPU ' describes the motivation behind development of GPUs and the architecture with an emphasis on NVIDIA's Tesla. The section 'TPU' briefly describes the architecture of Google's Tensor Processing Unit and the factors that led to the development of TPUs. The section ' Performance comparison of CPU and GPU ' describes how the GPUs have evolved with time and how it had led to the increase in computing power as compared to the CPUs. The section ' Challenges in exploiting multicore/parallel architectures ' describes the challenges faced by programmers in exploting the multicore architectures.

2. WHY MULTICORE PROCESSORS?

Performance refers to the amount of time it takes to execute a given task. This is not simply clock frequency alone or the number of instructions executed per clock cycle, but rather the combination of both clock frequency and instructions per clock cycle.

$$Performance = IPC \times Frequency \tag{1}$$

Where IPC is Instructions executed Per Clock Unfortunately high clock ratio has some implications in power consumption. Unfortunately leakage power limits frequency scaling and it is the most important constraint of frequency acceleration.

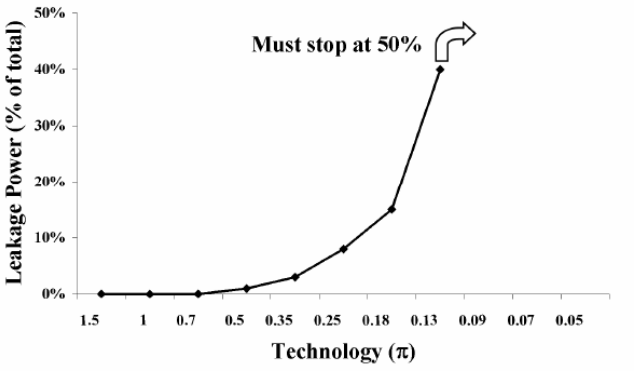

Figure 1. Leakage Power (% of total) vs. process technology

$$IPC = \frac{Uops\ per\ Cycle}{Uops\ per\ Instruction} \quad (2)$$

Uops per Cycle represents the average number of micro operations executed per cycle. Uops per Cycle reflects to micro-architecture issue parallelism. Clearly if the number of execution units is superior then number of Uops issued and executed per one cycle is going to be a higher as well. If we want to have the highest IPC then we need to get Uops per Cycle ratio as high as possible. Uops per Instruction represents common number of micro-operations which are going to be used for an execution of one full instruction. The idea is to minimize number of Uops used to construct an instruction. In an ideal situation Uops per Instruction = 1.

$$Power = dc \times V^2 \times Frequency \quad (3)$$

Where dc is Dynamic Capacitance and V is voltage
Taking into account performance and power equations, CPU designers need to balance IPC efficiency from one side and voltage and frequency from the other to offer a compromise of performance and power efficiency of the processor. New metrics of design success are no longer focused just pure performance, but rather in delivering a new microarchitecture which delivers leadership in both raw performance and in performance per watt.
Multi-core processor systems are going to change the dynamics of the market and enable new innovative designs delivering high performance with an optimized power characteristic. They drive multithreading and parallelism at a higher than instruction level, and provide it to mainstream computing on a massive scale.

3. Evolution and Challenges Faced

The growing market and the demand for faster performance drove the industry to manufacture faster and smarter chips. Semiconductor industry once driven by performance being the major design objective, is today being driven by other important considerations such chip fabrication costs, fault tolerance, power efficiency and heat dissipation. This led to the development of multi-core processors which have been effective in addressing these challenges.
A Multi-core processor is typically a single processor which contains several cores on a chip. The cores are functional units made up of computation units and caches. These multiple cores on a single chip combine to replicate the performance of a single faster processor. The individual cores on a multi-core processor dont necessarily run as fast as the highest performing single-core processors, but they improve overall performance by handling more tasks in parallel. Single core processors running multiple programs would assign time slice to work on one program and then assign different time slices for the remaining programs. If one of the processes is taking longer time to complete then all the rest of the processes start lagging behind. However, In the case of multi-core processors if you have multiple tasks that can be run in parallel at the same time, each of them will be executed by a separate core in parallel thus boosting the performance as shown in Figure 2.
Multi-core processors could be implemented in many ways based on the application requirement. It could be implemented as

1) Group of homogeneous cores
2) Group of heterogeneous cores
3) A combination of both

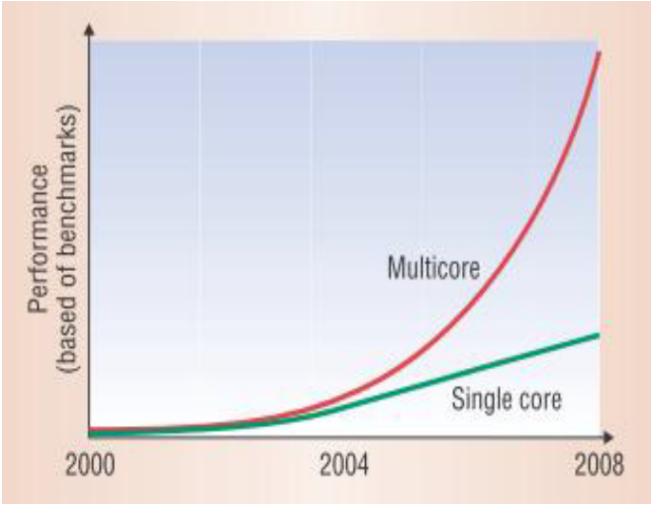

Figure 2. Multicore chips perform betterbased on Intel tests using the SPECint2000 and SPECfp2000 benchmarksthan single-core processors

Challenges faced:-

1) Porting legacy software programs developed years ago to multi core software programs.
2) On-chip interconnects are becoming a critical bottle-neck in meeting performance of multi-core chips.
3) Increased design complexity due to possible race conditions as the number of cores increase in a multi-core environment.
4) Interaction between on chip components (cores, memory controllers and shared components viz. cache and memories)

4. Factors Affecting Performance of Multi-Core Processors

To understand the performance trade-offs between wide-issue processors and single chip multiprocessors in a more quantitative way, researchers had compared the performance of a six-issue dynamically scheduled superscalar processor with a 4 × two-issue multiprocessor. The results show that on applications that cannot be parallelized, the superscalar microarchitecture performs better than one processor of the multiprocessor architecture. For applications with fine grained thread-level parallelism the multiprocessor microarchitecture can exploit this parallelism so that the super-scalar microarchitecture is at most 10% better. For applications with large grained thread-level parallelism and multiprogramming workloads the multiprocessor micro-architecture performs 50100% better than the wide superscalar micro-architecture.
Three types of architectures are defined. They are:-

1) Super-Scalar Processors
2) Single Chip Multiprocessor (CMP) [Multiple Processors in a single chip]
3) Multiple-Chip Multiprocessor (M-CMP) [Combines multiple CMPs to further increase performance]

Results

For these results the directory size is varied from 256 to 2048 entries on a 4 node, 16 node and 64 node Chip multiprocessor. All simulations in this section are configured with same cache size: 32KB L1I + 32KB L1D and 512KB L2 cache per core.

It can be observed that L1 miss is greatly affected by the directory size.

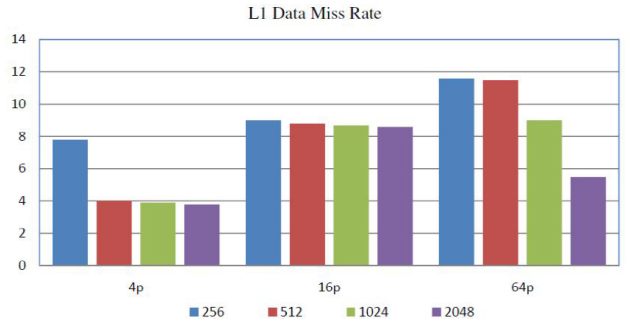

Figure 3. L1 Data Miss Rate

The number of L2 misses increases as no of cores within the system increases, nevertheless the miss rate decrease to 40%-80% because of the larger total L2 cache on chip.

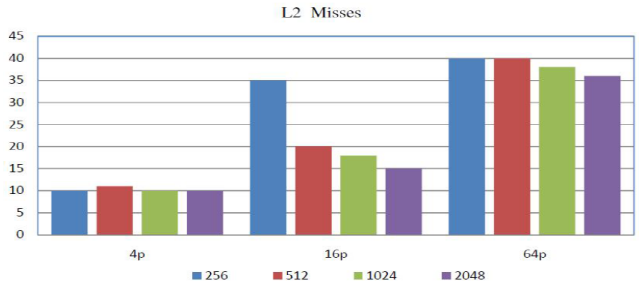

Figure 4. L2 Misses

L2 Replacement occurs when the L2 cache is full and another allocation is required. According to LRU policy, a block will be chosen and replaced by a new one.

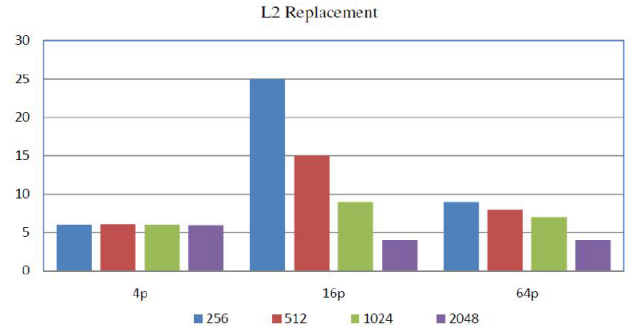

Figure 5. L2 Replacement

On 16 core node we can observe that larger directory size does not improve the directory replacements. The reason is that so many data are mapped to the same location resulting in many conflicts.

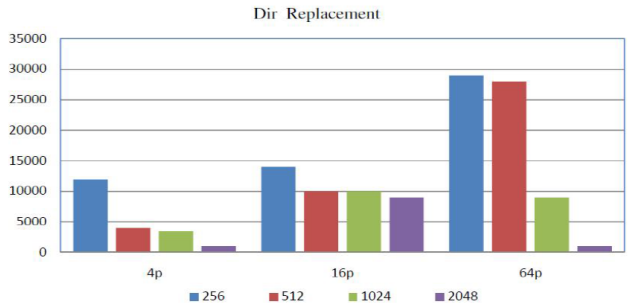

Figure 6. Directory Replacements

5. Case Study on IBM Power5

This section provides a case study on the IBM Power5 Chip a dual core multithreaded processor which was produced in 2004. There are three different methods for handling multiple threads. They are:-

1) Coarse-grained multithreading
2) Fine-grained multithreading
3) Simultaneous multithreading

In coarse-grained multithreading, only one thread executes at any instance. When a thread encounters a long-latency event, such as a cache miss, the hardware swaps in a second thread to use the machines resources, rather than letting the machine remain idle. By allowing other work to use what otherwise would be idle cycles, this scheme increases overall system throughput. A variant of coarse-grained multithreading is fine-grained multithreading. Machines of this class execute threads in successive cycles, in round-robin fashion. Accommodating this design requires duplicate hardware facilities. When a thread encounters a long-latency event, its cycles remain unused. Finally, in simultaneous multithreading (SMT), as in other multithreaded implementations, the processor fetches instructions from more than one thread. What differentiates this implementation is its ability to schedule instructions for execution from all threads concurrently. With SMT, the system dynamically adjusts to the environment, allowing instructions to execute from each thread if possible, and allowing instructions from one thread to utilize all the execution units if the other thread encounters a long latency event. The Power5 design implements two-way SMT on each of the chips two processor cores.

Power5 system structure

In this Figure 7, the system structure of Power 4 Chip (a) and Power 5 Chip (b) are shown. Moving the level-three (L3) cache from the memory side to the processor side of the fabric lets the Power5 more frequently satisfy level-two (L2) cache misses with hits in the 36-Mbyte off-chip L3 cache, avoiding traffic on the inter-chip fabric. References to data not resident in the on-chip L2 cache cause the system to check the L3 cache before sending requests onto the interconnection fabric, thus reducing traffic on the fabric and allowing Power5-based systems to scale to higher levels of symmetric multiprocessing. Initial Power5 systems supported 64 physical processors. Power5 processor cores are designed to support both enhanced SMT and single threaded (ST) operation modes.

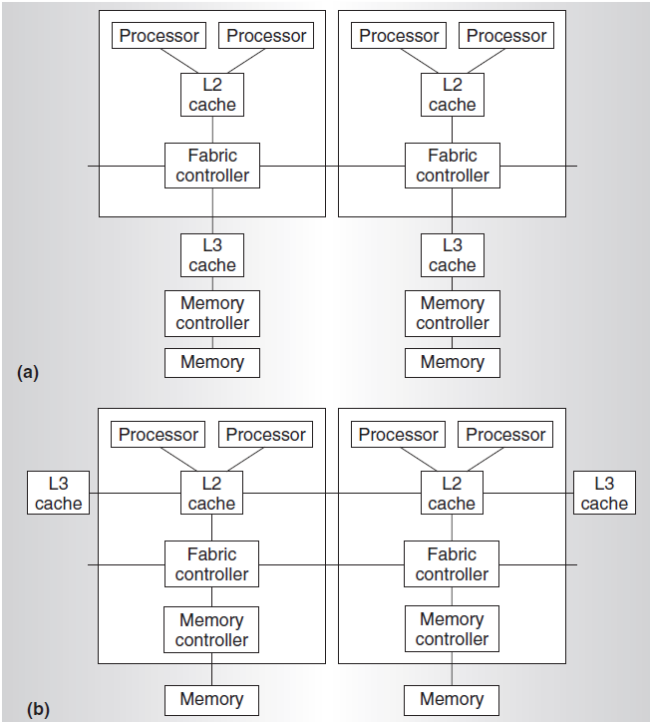

Figure 7. Power4 (a) and Power5 (b) system structures

To improve SMT performance for various workload mixes and provide robust quality of service, two features were added to the Power5 chip: dynamic resource balancing and adjustable thread priority. The objective of dynamic resource balancing is to ensure that the two threads executing on the same processor flow smoothly through the system. Dynamic resource-balancing logic monitors resources such as the GCT (Group Completion Table) and the load miss queue to determine if one thread is hogging resources. Adjustable thread priority lets software determine when one thread should have a greater (or lesser) share of execution resources. The Power5 microprocessor supports eight software-controlled priority levels for each thread. Level 0 is in effect when a thread is not running. Levels 1 (the lowest) through 7 apply to running threads. The out-of-order execution Power5 design coupled with dual 2-way simultaneous multithreaded processors provides instruction and thread level parallelism.

6. Design for Multicore Architectures

This section will tell us how multicore architectures are reigning the microprocessor design because of performance, thread level parallelism, and thermal scaling.

The Metrics used by us are

1) The performance of the system is measured by execution time of running a given parallel applications.
2) Power effectiveness is calculated by the system energy, viz. the amount of energy needed by an application.

The outline of the CMP(Chip Multi Processor) is composed of different architectures viz, present with (2, 4, 8) cores in the system, and an Issue width (2, 4, 6, 8) along with L2 cache size (256, 512, 1024, 2048 KB). This is the size of the private L2 cache of each processor.

Performance

The bar chart shows the average execution time, for every configurations. We can see that when scaling the system from 2 to 4, and from 4 to 8 processors, the latency is reduced by a 47% in each step which is a trend across all parameter. It means that communication hike is not appreciated and high parallel effectiveness is achieved.

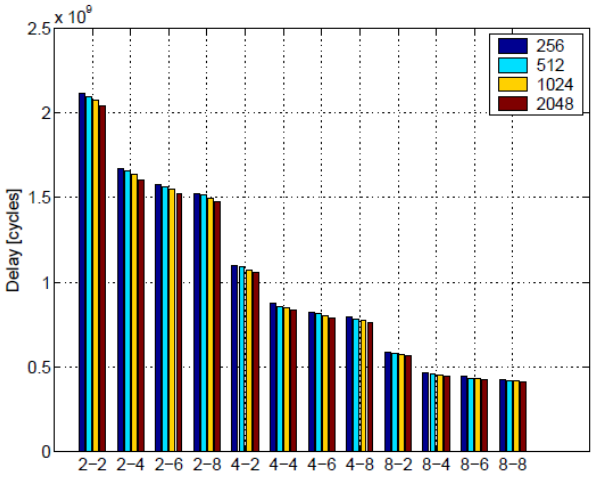

Figure 8. Execution time (delay). Each group of bars (L2 cache varying) is labeled with No. of procs-issue width

Number of cores

It is observed that when the number of processors are added, the system energy moderately increases even though the area doubles as core count doubles.

Issue width

By altering the issue width and, core complexity, energy has a minimum at 4-issue which occurs for 2, 4 and 8 processors and these offer the best balance between complexity, power consumption and performance.

L2 size

L2 cache gives great contribution to chip area, therefore it affects leakage energy considerably. The static energy part for our simulations ranges above 37%
Different parts cause hotspots in the processors. Power density for each microarchitecture unit provides a substitute to temperature, but doesnt suffice in explaining effects due to the thermal coupling and spreading area. It is important to care about the geometric characteristics of the layout. We can point all those impacting on chip heating as follows:

1) Proximity of hot units: If two or more hotspots come close due to thermal coupling there is an increase in temperature.
2) Relative positions of hot and cold units: A layout interleaving hot and cold units will result in less power density (thus less temperature).
3) Accessible spreading silicon: Pieces placed in such a position, that limits its spreading perimeter, will result in rise

in temperature.

These principles, are related to the idea of dropping the global power density, which applies to core microarchitecture, as well as CMP system architecture. In the first case, they recommend not to make too close hot units. For what concerns CMPs they can be shown as three important things to consider

1) Closeness of the cores
2) Position of cores and L2 caches
3) Placement of cores in the die.

We can briefly say that

1) The choice of L2 cache size is not only matter of performance/area, but also temperature.
2) Layout of a geometric characteristics are important for chip temperature. The position of processors and caches must be carefully determined to avoid hotspot thermal coupling. Several other factors may affect design choices, such as area, yield and reliability.

7. GPU

Introduction

Graphics Processing units took multicore architectures a step further. GPUs are being used widely for rendering graphics, tensor manipulation, deep learning etc. We present a detailed analysis of GPU architectures focusing on NVIDIAs Tesla. GPUs are a combination of the Multithreading programming model(Single Program Multiple Threads model) and SIMD execution model. In a GPU, the program is divided into multiple threads. The threads that perform the same operations on different data are grouped into warps. This grouping of threads is done dynamically based on the type of operations. These warps are sent to an SIMD pipeline for a round robin fashioned execution. This is the fine grain multithreading model. This modified SIMD is called the Single Instruction Multiple Thread model (SIMT). The advantages of SIMT include the ability to treat threads separately and flexible grouping of threads into warps. This SIMD pipeline is called a Shader core in GPU architecture. All the threads in the warp share a common Program Counter. There is an instruction cache in the pipeline to allow faster instruction access followed by a decoder and multiple scalar pipelines. The shader cores are connected to a Memory controller and a dynamic memory system via an Interconnection network.

TESLA

The realization that a GPU can also be used for heavy weight matrix/tensor operations isnt surprising. Since GPUs operate on millions of pixels at a time and pixels are nothing but two dimensional coordinates on the screen, GPUs were optimized and extended to perform tensor computations as well. NVIDIA developed a new framework called Compute Unified Device Architecture (CUDA) in C language to develop high performance parallel computing applications. NVIDIA Tesla was designed keeping CUDA and graphics in mind. Tesla clocks at 600MHz.

Architecture

Tesla has 128 streaming processor cores organized as 16 multiprocessors in 8 independent processing units called Texture/Processor Clusters. These TPCs are connected to Render Output (ROP)units, Level 2 cache and a dynamic memory via an interconnection network. An ROP unit takes pixel,texture pixel(texel) information, processes it via matrix and vector operations into final pixel and depth values. This process is called Rasterization.

SPA(Streaming processor array)

The array processors carry out GPUs programmable calculations. Shading is the process of depicting depth perception in 3D models by using shader programs. The SPAs execute shader thread programs, GPU computing programs and provide thread control and management.

Texture Processor Cluster (TPC)

Each TPC contains a eometry controller, an SM controller, two streaming multiprocessors and a texture unit.

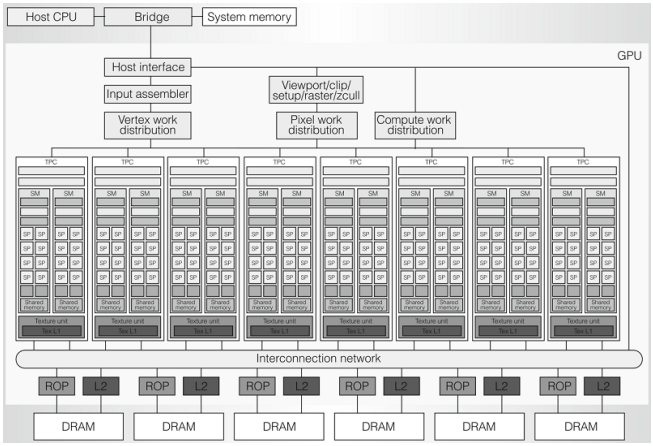

Figure 9. Architecture of TESLA

Workflow

The vertex work distribution module distributes the vertex shader programs across the TPCs. The TPCs process the input and store the result on the on-chip buffers. These buffers pass their results to the raster block to be rasterized into pixel fragments which are then directed to pixel work distribution module. The PWD module distributes work across TPCs for pixel fragment processing. The output shaded pixel fragments are sent to ROPs via an interconnection network for processing depth and color ROP units.The compute work distribution block dispatches compute thread arrays to the TPCs. The SPA acceptsand processes work for multiple logical streams simultaneously.

Each streaming multiprocessor core contains a scalar multiply add unit, giving the SM 8 MAD units. Each Special Function Unit(SFU) contains four floating point multipliers. The SM uses the texture unit as an additional execution unit and uses the SMC and ROP units to implement external memory load, store and access. The SM also a multithreaded instruction fetch and issue unit(MT issue), a read-only cache and a 16KB R/W shared memory. The SMs SIMT instruction unit creates, manages and schedules threads for execution in a warp of 32 threads. Each SM manages 24 warps, with a total of 768 threads. A warp, as mentioned before, consists of 32 threads of the same type i.e vertex, geometry, pixel or compute. The SM controller groups 8 pixel quads into 32 threads which is one warp. The SM schedules and executes multiple warp types concurrently. The SM warp scheduler operates at half the 1.5-GHz processor clock rate. At each

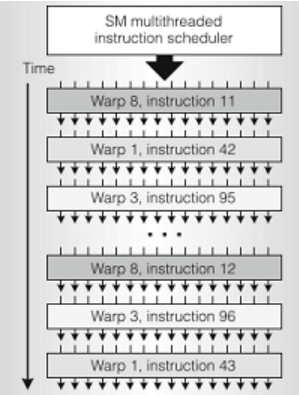

Figure 10. WARP Scheduling

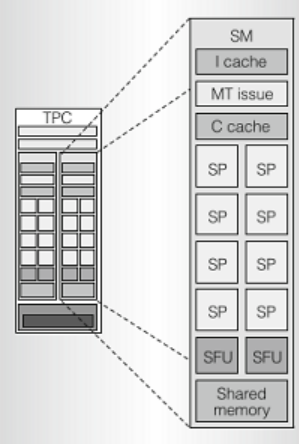

Figure 11. Internal Structure of a Streaming Multiprocessor

cycle, it selects one of the 24 warps to execute a SIMT warp instruction. An issued warp instruction executes as two sets of 16 threads over four processor cycles. The SP cores and SFU units execute instructions independently, and by issuing instructions between them on alternate cycles, the scheduler can keep both fully occupied.

Advancements in GPU architectures

GPUs mainly take advantage of the number of the cores. The core idea is the presence of a lot of dumber cores instead of a few intelligent ones like in a CPU. NVIDIAs 1080 has 2560 cores. In 2014, NVIDIA launched Tesla K80 with 4992 cores. In 2017, NVIDIA launched Volta 100 which packs a monstrous 21 billion transistors and 5120 cores and clocks at 1455 MHz. Volta is meant for compute purposes in datacenters instead of graphics.

8. TPU

In 2017, Google announced a new computing architecture called Tensor Processing Unit (TPU). It showed huge performance improvements and lesser power consumption. On an average, a TPU is 10x to 30x times faster than its contemporary CPUs and GPUs(Tesla K80). As mentioned before, GPUs are being widely used in the field of Artificial Intelligence; Deep learning to be more specific. Unlike GPUs which process computer graphics as well, a TPU is designed to handle computationally heavy tensor operations for training deep neural networks.
A very brief overview of Neural Networks:

1) Multi-layer Perceptron: These networks have a series of neurons stacked together forming layers. The input to the neuron is a weighted sum of outputs from the previous layer. The neuron is activated by introducing a non-linearity to the weighted sum. This is done using an activation function (eg: ReLU, Sigmoid, SeLU).
2) Convolutional Neural Networks: CNNs are neural networks that specialize in understanding grid-like data. This includes images, time-series data etc. These networks have a receptive field that strides through the input and applies convolution function to produce feature maps. These feature maps are again convolved to get lower dimensional representation of the input data.
3) Recurrent Neural Networks: Each subsequent layer is a collection of nonlinear functions of weighted sums of outputs and the previous state. The most popular RNN is the Long Short Term Memory. The art of the LSTM is in deciding what to forget and what to pass on as state to the next layer. The weights are reused across time steps.

Its evident from the above definitions that neural networks use a lot of memory. For instance, an MLP has 20 million parameters, an LSTM has 52 million parameters and a CNN has 100 million parameters to be optimized. All the operations are memory intensive and require an efficient pipeline.
In order to speed up the performance at the inference time of a neural network, it is important to reduce interactions with the host CPU. So, the TPU is designed to be a coprocessor on the I/O bus that can be plugged in just like a GPU. Instead of TPU fetching instructions to decode and execute, the host server directly sends TPU instructions to be executed.

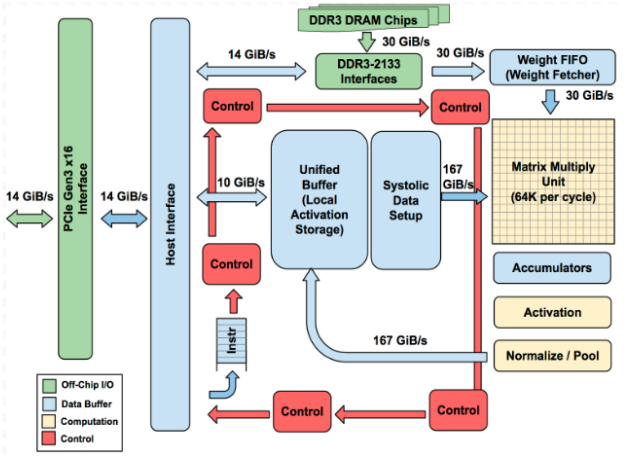

Figure 12. Block diagram of a TPU

The TPU instructions from the host server are fetched through the PCIe(Peripheral Component Interconnect Express). These instructions are stored in an instruction buffer. The Matrix Multiply unit handles all the heavy weight of a TPU. It contains 256x256 MACs that can perform 8-bit multiply and adds on signed or unsigned integers. The 16 bit products are collected in the 4MB of 32 bit accumulators below the matrix unit. The 4MB holds 4096, 256 element, 32 bit accumulators.
The matrix product produces one 256 element partial sum per clock cycle. When using a mix of 8-bit weights and 16-

bit activations, the Matrix Unit computes at half the speed and it computes at quarter the speed when both are 16-bits. It reads and writes 256 values per clock cycle and can perform either a matrix multiply or a convolution. The matrix unit holds one 64 KB tile of weights plus one for double buffering. Deviating a little bit, double buffering is also used in computer graphics to improve rendering. Double buffering involves usage of one set of data while another is being collected. This unit(MU) is designed for dense matrices. The weights for matrix unit are staged through an on chip Weight FIFO that reads from an off-chip 8GB DRAM called Weight Memory. This 8GB supports many active models. The weight FIFO is 4 tiles deep. The intermediate results are held in the 24MB on-chip Unified Buffer, which can serve as inputs to the matrix unit.

TPU Instructions

TPU instructions follow the CISC instruction set architecture since transfer through PCIe bus is relatively slow. The average clock cycles per instruction of these CISC instruction ranges from 10-20.

1) Read_Host_memory: reads data from host CPU memory to Unified buffer.
2) Read_Weights reads weights from Weight memory into the weight FIFO as I/P to the Matrix Unit.
3) MatrixMultiply/Convolve: causes matrix unit to perform a matrix multiply unit or a convolution from the unified buffer to the accumulators. Matrix operation takes Y*256 input, multiplies it by a 256x256 constant weight input and produces a Z*256 output, taking Z pipelined cycles to complete.
4) Activate: performs the non linear function of the neuron as mentioned before. Input: Accumulators; Output: Unified Buffer. Can also perform pooling operations for the convolutions.
5) Write_Host_Memory writes data from UB into the CPU host memory.

9. Performance comparison of CPU and GPU

For comparing the performance, the key parameters which are analyzed are latency and throughput. The execution time for performing tasks with increasing workloads is observed for both GPU and CPU. The tasks are written using Compute Unified Device Architecture (CUDA) in C language, the two parameters latency and throughput are measured with the increasing workload. As the graphics involved in the computer application demanded for high quality real time graphic rendering the need for better and faster graphic hardware increased, earlier around 1990 there were no GPUs instead Video Graphic Array (VGA) controllers were used in PCs for the graphic needs.
In 1999 NVIDIA introduced the first GPU it consist of a single chip and had the capability of real time 3D rendering. It used floating point arithmetic to calculate 3D geometry and vertices first, to be applied it to pixel lighting and color values then to handle high dynamic range scenes. Later in 2001 NVIDIA GeForce introduces General shader programmability which allowed the application developer to work with the instruction of the floating point vertex engine.
GeForce introduced 8800 GPU in 2006, featuring an array of unified processors. The unified processor supports dynamic partitioning of the array of processors to vertex shading stage, geometry processing (first introduced in GeForce 8800 GPU) and pixel processing stage. Earlier GPUs can only be utilized to process graphic data, after the introduction of GPGPUs General Purpose GPUs by NVIDIA GPUs became fully programmable and can now handle the application traditionally handled by CPUs. NVIDIA developed CUDA C/C++ libraries which helps the developers to use the fully programmable GPU capabilities.
In 2010, NVIDIA introduced Kepler architecture adding dynamic parallelism functionalities and Hyper Q. Dynamic parallelism allowed GPU to schedule the task through the best hardware path without the involvement of CPU. Hyper Q on the other hand allows multiple CPU cores of the PC to call the GPU increasing the GPU utilization.

Experimentation

The comparison in this experiment is between a dual core Intel 3rd generation i5-3210 processor with 4GB RAM and the same processor when combined with a NVIDIA GT630M with a dedicated 2GB RAM. The CPU supports multithreading with 4 threads and each core operates at 2.5 GHz. GPU operates at 0.95 GHz.
Time taken to add elements to the GPU and the CPU were recorded and plotted as shown in the Figure add figure number time periods of all the block sizes are plotted on Y axis in logarithmic scale. These plots are compared by the CPUs time to add block. The plot concludes that for the block size 1 to 128 the CPUs performance is better than GPU but as the block size is increased from 256 to 512 GPU outperforms CPU.

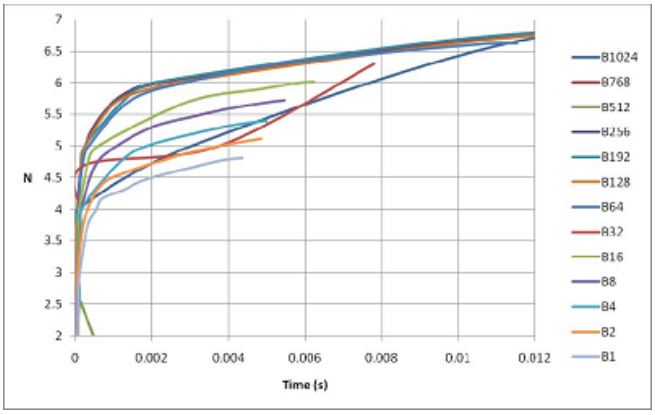

Figure 13. GPU Process time for all blocks

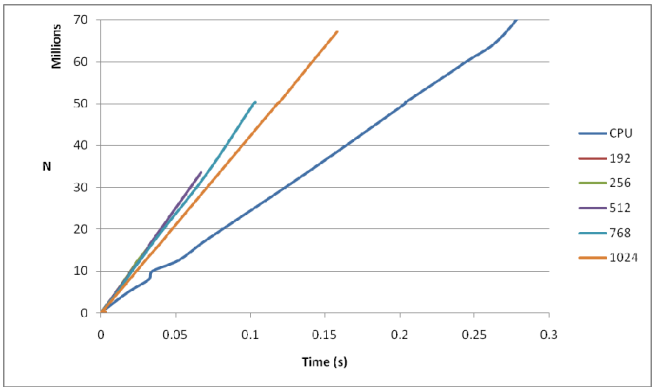

Figure 14. Execution time of CPU and GPU for block size 192, 256, 512, 768 and 1024

Comparison of speed up is performed, as shown in figure add- figure -number, CPU speed up is compared with four block sizes of 192, 256,512, 768 and as 1024 is the largest input for CPU it is also considered. From the observed readings it is deduced that when the workload on CPU is small it gives a much better performance than GPU. However as the workload starts increasing the time taken by the GPU decreases and it outperforms CPU. The speed up of GPU with CPU as reference is shown in Figure add figure number.

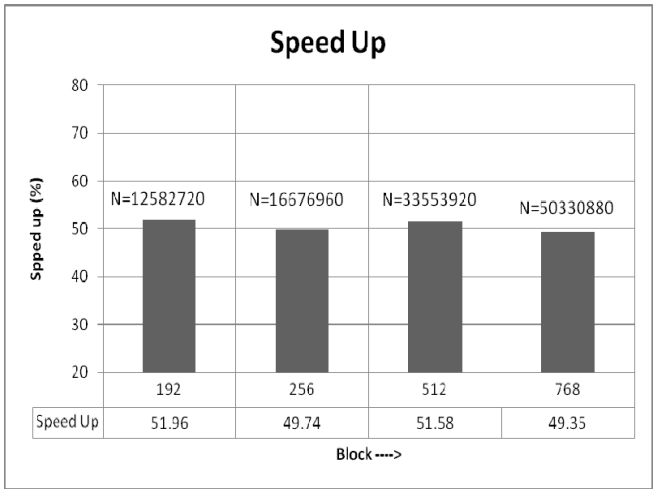

Figure 15. Speed of GPU with CPU as reference

Our second parameter for comparison is throughput, throughput is the number of tasks completed by a processor in unit time.
From the observed readings the maximum input elements that can be taken by the CPU and still output a valid value is found out to be 16777216, while if a GPU is configured with a block size of 1024 the number of elements that the GPU can process in a single dimension is 67107840, which is about four times larger than the CPU. These value conclude that in case of throughput GPU is far superior to CPU.
From the experiments performed and the results obtained it is safe to say that when the workload or the number of input is small the performance of CPU was better than GPU but as the workload increased the GPU performed about 50% faster than the CPU.
As in case of number of tasks performed (throughput) GPU always has an upper hand on the CPU as it can perform 4 times the tasks performed by CPU in same time.

10. Challenges in exploiting multicore/parallel architectures

The process of converting code written for a single core platform to code that can exploit multicore/parallel architectures is difficult. Especially, with the advent of GPUs, parallelizable code has been given emphasis since it has significant performance boosts. Fortunately, libraries like CUDA have made the task simpler. But, modifying legacy software to adapt to new hardware is still an intense task.
In 1965, Gordon Moore predicted that the number of transistors on a chip would roughly double every 18 months. Intel chips have reached 2GHz in 2001 and consequently, software has experienced exponential hikes in performance. The clock speeds have continued to increase and surpassed 15GHz in 2010. Then, the increase came to a halt as it became infeasible to cool the heat generated from the amount of transistors that could fit on a single chip.

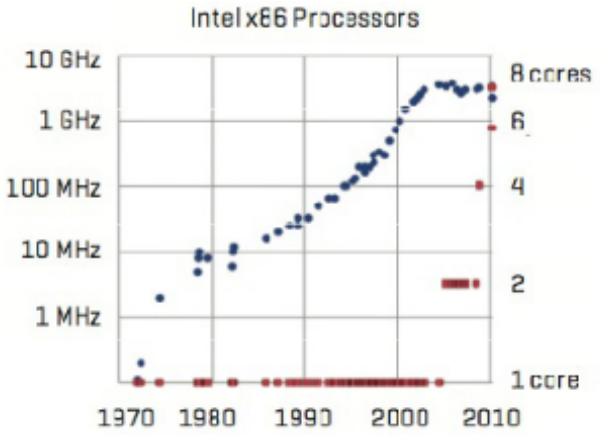

Figure 16. Clock Speeds of Intel Processors over time

Challenges

1) Testing parallel programs is difficult
2) Insufficient tools to aid parallel programmers
3) Managing data dependencies when writing parallel code is difficult
4) Interdependence of abstraction levels
5) The connection configuration of cores on a processor may vary widely
6) Locks are used to enforce data dependence in parallel systems; Locks can lead to race conditions, priority inversion and in the worst case, deadlock.

Instead of locks, transactional memory can be used. It is based on the concept of database transactions whereby a list of operations must be completed before the transaction can be deemed as finished. If the operation is interrupted, all the operations involved are cancelled. Transactional memory is still being developed.

11. Summary

Multi core architectures are paving the way to creation of new multi-core programming languages and softwares to port legacy software to multi-core aware software programs. Multi-core architectures can enhance user experiences in multitasking environments. GPUs have been developed to render high quality graphics with less latency. They've evolved to handle huge computations for other applications as well due to the presence of a large number of cores. TPU is an example of application specific multicore architecture. It's being used for deep learning and is under active development. We can expect many such application specific multi core architectures in the near future.

References

[1] Pawel Gepner, Michal F. Kowalik, "Multi-Core Processors: New Way to Achieve High System Performance", Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)

[2] Balaji Venu, "Multi-core processors An overview", arXiv:1110.3535 [cs.AR]

[3] Nitin Chaturvedi, S Gurunarayanan, "Study of Various Factors Affecting Performance of Multi-Core Processors", International Journal of Distributed and Parallel Systems (IJDPS) Vol.4, No.4, July 2013

[4] R. Kalla, Balaram Sinharoy, J.M. Tendler, "IBM Power5 Chip: A Dual-Core Multithreaded Processor", IEEE Micro (Volume: 24, Issue: 2, Mar-Apr 2004)

[5] Winnie Thomas, Rohin D. Daruwala, "Performance comparison of CPU and GPU on a discrete heterogeneous architecture"

[6] Kayvon Fatahalian, Mike Houston, "A closer look at GPUs"

[7] Erik Lindholm, John Nickolls, Stuart Oberman, John Montrym, "NVIDIA Tesla: A unified graphics and computing architecture"

[8] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, et al., " In-Datacenter Performance Analysis of a Tensor Processing Unit"

[9] Anne Meade, Jim Buckley, J. Collins, "Challenges of Evolving Sequential to Parallel Code: An Exploratory Review"

[10] Matteo Monchiero, Ramon Canal, Antonio Gonzalez, Design Space Exploration for Multicore Architectures: A Power/Performance/Thermal View"