Influence of Vector Processors on GPGPU architecture and the future of Computing

Department Of Information Technology, Monash University Malaysia

Nabil Ahmed
School of Information Technology
Monash University Malaysia
Major: Computer Science
e-mail: nahm39@student.monash.edu

Abstract— **The idea of vector processing has been around for decades. Although, the idea is a very simple one, but it can do wonders when it comes to the world of computer architecture. In this paper we will discuss what are vector processors including the pipelined variant of the vector CPU. We will see how the idea of vector processing inspired the work of GPGPU and revolutionized the parallel computing architecture. And finally, the Kernel for operating the CUDA architecture by NVIDEA that utilizes the GPU architecture to maximize performance is shown.**

Keywords-component: GPGPU, GPU, Vector Processing, SIMD, CUDA

What are Vector Processors?

The idea of Vector processing was first introduced in the early 1960's. Although it was a revolutionary idea at the time it is a very simple one. Use one Central Processing Unit(CPU) to control all the Arithmetic Logic Units (ALU's) or co-processors and pass one common instruction. The instruction is applied to different data for each of the co-processors to work on. All these data's can be put into a one-dimensional array which are referred to as vectors. In high level language terms this basically means applying one algorithm to large dataset. The dataset is passed in the form of an array. [1]

Difference between Array processor and Vector Processor

Array processors apply Instruction on multiple elements but at same time.
Vector processor apply Instruction on multiple elements but in repeated time steps.
The very first vector processor architecture first came into existence in the early 1970's. They were mainly focused on solving enormous calculations involving matrix and vector. These Processors were called the **Pipelined Vector Processor.** [1]

Non-pipelined processor vs Pipelined Processor

A processor in theory works by fetching the instruction from memory, decoding it, executing it and writing it to some form of memory.
A Non-pipelined processor does all of this in one step. The CPU perform one instruction per cycle, but each cycle is very long. This is because the next instruction cannot the decoded until and unless the preceding instruction has finished its execution. This effects the throughput (the amount of instructions that can be executed per unit time). Much of the other processor hardware remains idle during this period. A simple diagram is shown in figure 1(a) demonstrating the Non-pipelined variant of the processor. So, to utilize and to make the best use of all the components of a processor for enhancing performance the Idea of Pipelining was introduced. [8]
Pipelining is one form of embedded parallelism in a Computer System. What it basically does is the instruction that we sent in the Non-pipelined variant, it breaks that instruction into sever subprocess utilizing the power of the CPU components. Let's say it breaks that instruction into four stages: Instruction Fetch (IF) > Instruction Decode (ID) > Operand Fetch (OF) > Execution (EXEC) (shown in Fig 1.b). What the Pipelined variant of the processor is essentially doing is it is concurrently executing the instruction from process 1, fetching the operand from process 2, Decoding the instruction of process 3 and fetching the operation of instruction 4 (shown in the 4th column or 4th unit of time- x-axis figure 1.c). So, pipelining in general is a process that decomposes a process into subprocess each of which can be executed efficiently on a special dedicated autonomous unit that functions in parallel with the others. [8]

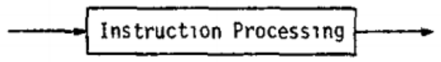


FIGURE 1a. Non-pipelined processor.

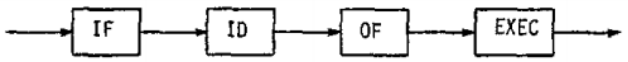


FIGURE 1b. Pipelined processor.

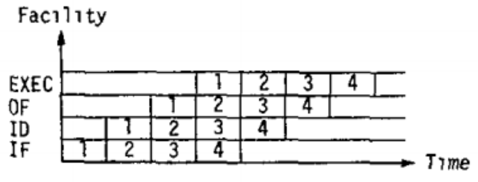
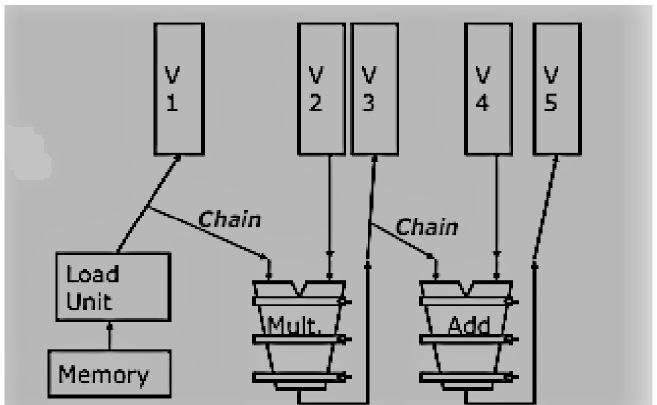


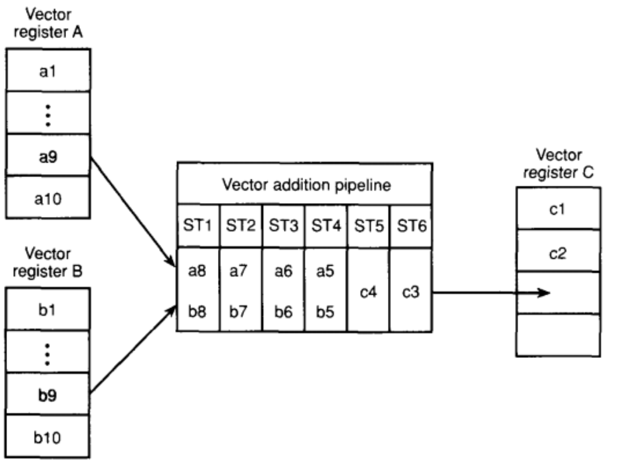
FIGURE 1c. Space-time diagram.

I.

The **Pipelined Vector Processor** used multiple pipelines with different functions assigned to each, to solve both vector and scalar Boolean and arithmetic operations. Both of which can run in parallel. These machines used the process of chaining.
An example of chaining is shown below:
Load Vector v_1
Multiply Vector v_1 and v_2 and put results into v_3
Add Vector v_3 and v_4 and put results into v_5



This means it would serially pass vector elements through a pipeline and use the results of one pipeline as the input of another. An example (see figure 2) shows a vector addition unit with six pipeline stages. Let's say each stage takes 10 nanoseconds to execute, then it would take 60 nanoseconds for the operands a1 and b1 for their results to be available. If the pipeline is full it would give result every 10 nanoseconds. Thus, for achieving maximum performance gain an eventful or filled pipeline is desirable.

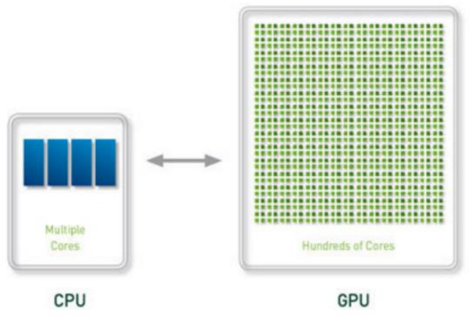


SIMD
They perform single instruction on multiple data streams. Their process archetypally involves executing the same instruction on different data simultaneously by using multiple processors.
Why is vector processing Important?
All modern CPU architecture contains vector processing. Vector Processors are typically referred to as SIMD. SIMD processing can be of two types: Array processor and Vector Processor.
Advantages of Vector Processor:
- Dependencies within vector is not essential this allows them to have deep pipelines.
- They access memory frequently and thus they interleave several memory banks and thus it makes the bandwidth of the memory prefetching bigger.
-Loops does not need to be coded explicitly and thus instruction sequence requires less branches.
Disadvantages of Vector Processor:
-Requires data parallelism to work. Very inefficient if parallelism is irregular.
-The Bandwidth of Memory can become a blockage: if balance within memory operation is not preserved or if mapping of the data to the memory banks is not done appropriately.

GPGPU (General Purpose Computing on Graphic Processing Units)
General-purpose computing by using GPUs became popular around 2001. Current graphics processing units (GPU's) contains array of pipelines. They can be called vector processors since they use a similar approach to hide memory latencies.
Traditionally Graphical Processing Units (GPUs) were only used for solving graphics intensive tasks. Over time their performance increased exponentially. Recently GPU's have

become a fascinating field in the world of Computer Science. It can be used to solve problems that requires high computation power. Previously it was done by the Central Processing Unit (CPU) alone. Scientists and researchers identified that they can harness the power of the GPU to not only solve problems related to graphics or images but also general-purpose works. [3]

GPU is often considered the future of computation. GPU has proved that it can perform calculations faster than a CPU in terms of computation speed. The main task of the GPU is to process 3D functions. They perform efficiently when computing these kinds of problems. The CPU and the GPU both consists of a single processor Chip. The main difference between them is the number of processors inside them. The figure below shows the core comparison in CPU's (multiple cores) and GPU's (hundreds of cores).



Computations of 3D model is difficult for the CPU.
The idea of the GPGPU is simple. The sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU. Programming instructions frequently involves in deriving data from memory. This is done with the assistance of the CPU. Modern CPU's are equipped for crunching greater amount of numbers than was previously possible. Handling tremendous amount of information or crunching huge numbers puts a ton of pressure on CPU. It is now possible to do these efficiently with the help of a Graphics Processing Unit(GPU). Difficult and intense mathematical calculations can be passed to the GPU freeing up the CPU resources for other higher priority tasks. Ancient CPU's used to do all the tasks alone as seen above in vector processing. Nowadays, CPU's can transfer their workload to other microprocessors. The figure below shows the difference between CPU and GPU. [4]

GPU Architecture

GPU architecture is immensely parallel. There are many real-world problems and applications that can take great advantage of this parallel architecture. On top that GPUs can solve arithmetic problems effectively and this enhances them to program the pipeline in various ways. The figure at the end shows the architecture of the GPU.

The figure shows an array of processors that can perform high degree of threading (a thread of execution is the smallest sequence of programmed instructions).

Difference between CPU and GPU RAMs

The CPU RAM is a memory which is used for storing data. Any part (byte) of the memory can be randomly accessed by the instruction running on the processor.
The GPU RAM however holds videos images, texture information for three-dimensional rendering. The GPU RAMs are different than that of the CPU. The latency of the GPU memory is higher than that of the CPU memory (transferring graphics instead of data). But, since GPU makes use of their parallel architecture, they have higher bandwidth while transferring data. This makes up for the higher latency.

GPGPU

Previously the functionality of the GPU was very limited. Indeed, for a long period of time the GPU's only accessed and utilized a limited portions of graphics pipeline. The GPU capacity for processing is limited. It depends on the fragments and independent vertices. The upside to this are the multiple cores in the GPU which can be used for doing the processing in parallel. The effectiveness of this parallelism is seen into effect when the algorithm needs to process several fragments or vertices in a similar way. So, GPU processors can run one single kernel that can function in parallel on a stream of data at once. What is a stream? A stream is basically similar set of encryptions or codes or instructions that can be computed in a similar fashion. Since, streams provide similar computations they provide parallelism. There can be many inputs as well as outputs and GPU's require both writeable memories and readable memories thus achieving data-parallelism. Writeable memories are required because the Arithmetic Logic Unit (ALU) of GPU performs heavy computations on various fragment pipelines and vertices. For doing all these complex operations the operand has to write frequently to the GPU memory as it must remember a lot of variables which is not possible by only using a Readable memory. Due to all these factors the computational power of the GPU increases exponentially. All these powers were not utilized until recently and the future of Computing seems irradiating. Processing Non-graphical elements by using GPU is known as GPGPU. GPGPU's are usually used for computing tasks that are mathematically heavy. The task is run in parallel for completing it quickly i.e. lower time to compute. The GPGPU is a very specialized architecture which can be used to great advantage by running it in parallel if utilized properly by the algorithm. However, there is a downside to this architecture. Due to its nature of parallelism it cannot be used for solving problems using integer operands. It also cannot perform bitwise shifting or precision arithmetic (double). All these operations can be performed by the CPU very easily, but it is impossible for the GPU which is why the GPU is very unusual processor model. Due to all these tempting potentials of the GPU architecture numerous programmers are shifting their coding nature in such a way that their applications would be less CPU oriented and more towards the GPU. Some of the programmers even rewrote their programs so that their application could utilize the full power of the GPU. However, there is a challenge for the programmer which is to obtain parallelism in their coding. This means rearranging or changing their data to vector quantities rather than a scalar one. Parallelism in GPU architecture is present at every level.

Instruction Level Parallelism

Instruction Level Parallelism (ILP) is essentially the amount of instructions that can be executed concurrently in parallel. A simple example can be:

1. z = x + y
2. a = b + c
3. e = a*z

Here, instruction 1 and 2 can be computed concurrently. Because the operand of Instruction 1 and 2 does not overlap each other and they are independent. However, for computing instruction 3 it needs the result from instruction 1 and 2. So, a program containing a lot of instructions like the one shown in 1 and 2 will have a high level of Instruction Level Parallelism (ILP) in it. A high level of ILP is desired for effectively computing general algorithms in the GPU processor. The higher the ILP the faster the processing in the GPU architecture.

Recent Developments

CUDA

Compute Unified Devise Architecture (CUDA) is NVIDEA's GPU architecture designed to compute general purpose task using GPU. The benefit of using CUDA is that it gives enormous power for computing task to the user. There are 128 cores co-operating in the CUDA architecture. Using CUDA is only beneficial if the algorithm that is being run is highly parallel in nature. For increasing performance of the algorithm many thread are required. The more the number of threads, the better the performance. CUDA is not useful for serial algorithms. For getting the full benefits of the CUDA architecture the problem must be broken down i.e. the serial algorithm in at least a 1000 thread. If the serial algorithm can be broken down, then only it can be converted into a parallel one although breaking down a serial algorithm is not always possible. For getting the best performance out of the CUDA architecture a minimum of thousand thread is desirable after breaking down the algorithm. [5]

Advantage of CUDA

One huge advantage is that when the programmer wants to transfer application to the CUDA technology the whole program does not need to be re-written. The programmer can simply call CUDA method by writing kernel call for computing large and complex mathematical operations. This saves a lot of time for the programmer and increases their efficiency. They just need to code the parts of the program where complex mathematical problems are involved.

L1 and L2 caches in CPU's

The CPU contains very fast and small units of memory called caches. These helps to store the data and operands that are frequently used by the CPU registers. The L1 and L2 caches are essentially two levels of caches that most of the CPU's are equipped with. L1 cache are very fast. If the data that the CPU is looking for is not in the L1 cache, then the processor will look in the L2 cache. L2 cache is bigger in size than the L1 cache but slower. If the data is not in the L2 cache, then the processor will fetch the data from the main memory.

Register files in GPU's

GPU's have very large register files which is needed for supporting their huge multithreaded architecture. The capacity of the register files in the GPU is larger than even the L1 and L2 cache. For improving the latency traditional CPU's use way larger cache compared to their tiny register files. [4]

Accelerator

Accelerators are hardware devices designed to solve subroutines or specific tasks efficiently. They are attached to the CPU. The accelerator manipulates the data by performing the task assigned to it. The CPU manages the retrieval and copying of data that is altered by the accelerator. Using GPU as accelerator for solving general purpose task is known as GPGPU. The computer scientists cloaked scientific designs as graphics manipulations and were awed by the results. After realizing the potential of the GPU's, they started making accelerators but without the video connectors in them. This became quite popular among the High-Performance Computing Community (HPC). [2]

Conclusion

The article introduces the vector processor architecture. It gives an overview of its history, when the idea was first introduced and how it evolved into the makings of the GPGPU. The article discusses about the difference between the array processors and vector processors. It also discusses about the non-pipeline against the pipeline variant of the processor which is quite essential to grasp a proper understanding of the underlying theory. It then goes onto explain the internals of the working of the GPU and how GPUs can be used for general purpose computing. Lastly, the article introduces the accelerators very briefly. From the article it is evident that a lot of exciting things are happening in the world of Computer Science. As the GPGPU computations will evolve more into the future it is palpable that with further research and experimentation we can do a lot more with GPUs and Accelerators.

CPU	GPU
–Really fast caches (great for data reuse)	– Lots of math units
– Fine branching granularity	– Fast access to onboard memory
– Lots of different processes/threads	– Run a program on each fragment/vertex
– High performance on a single thread of execution	– High throughput on parallel tasks
CPUs are great for task parallelism	GPUs are great for data parallelism
CPU optimised for high performance on sequential codes (caches and branch prediction)	GPU optimised for higher arithmetic intensity for parallel nature (Floating point operations)

Table 1: Comparison between CPU and GPU

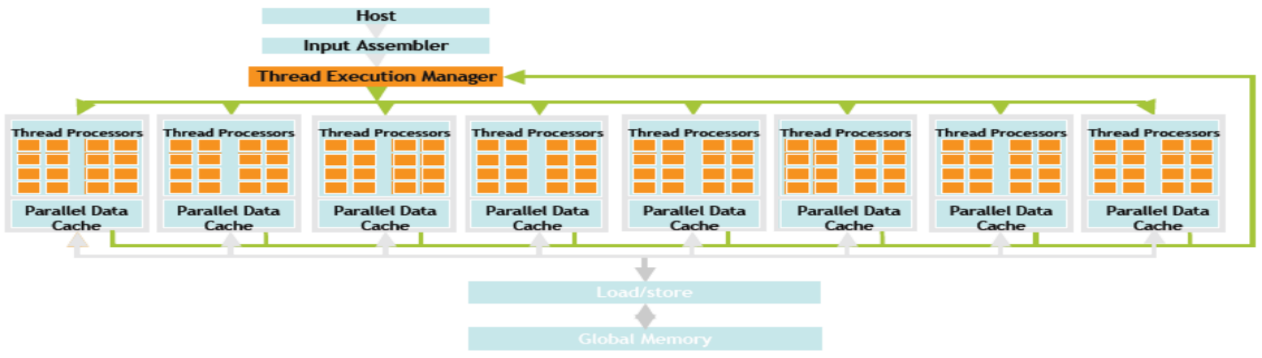


Figure (GPU architecture)

REFERENCES

[1] R. Duncan, "A Survey of Parallel Computer Architectures," in *Computer*, vol. 23, no. , pp. 5-16, 1990.

[2] U. Lopez-Novoa, A. Mendiburu and J. Miguel-Alonso, "A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing," in *IEEE Transactions on Parallel & Distributed Systems*, vol. 26, no. 1, pp. 272-281, 2015.

[3] Quan Deng, Youtao Zhang, Minxuan Zhang, Jun Yang, "Towards warp-scheduler friendly STT-RAM/SRAM hybrid GPGPU register file design", *Computer-Aided Design (ICCAD) 2017 IEEE/ACM International Conference on*, pp. 736-742, 2017

[4] Ghorpade, Jayshree, et al. "GPGPU processing in CUDA architecture." *arXiv preprint arXiv:1202.4347* (2012).

[5] Han, Tianyi David, and Tarek S. Abdelrahman. "hiCUDA: High-level GPGPU programming." *IEEE Transactions on Parallel and Distributed systems* 22.1 (2011): 78-90.

[6] Owens, John D., et al. "A survey of general-purpose computation on graphics hardware." *Computer graphics forum*. Vol. 26. No. 1. Oxford, UK: Blackwell Publishing Ltd, 2007.

[7] BUCK I., PURCELL T.: A toolkit for computation on GPUs. In *GPU Gems*, Fernando R., editor. Addison Wesley, pp. 621–636, March 2004.

[8] Ramamoorthy, Chittoor V., and Hon Fung Li. "Pipeline architecture." *ACM Computing Surveys (CSUR)* 9.1 (1977): 61-102.

[9] J.J. Dongarra, ed., *Experimental Parallel Computing Architectures*, North-Holland, Amsterdam, 1987

[10] BOUBEKEUR T., SCHLICK C.: Generic mesh refinement on GPU. In *Graphics Hardware 2005*, pp. 99–104,July 2005.