International Journal of Computer Applications (0975 – 8887)
Volume 111 – No 7, February 2015

Enhancing Computational Performance using CPU-GPU Integration

Sukanya.R
B.E Third year, Department of Computer Science and Engineering
SSN College of Engineering, Chennai

Swaathikka.K
B.E Third year, Department of Computer Science and Engineering
SSN College of Engineering, Chennai

Soorya.R
B.E Second year, Department of Computer Science and Engineering
SASTRA University, Tanjore

ABSTRACT
The scope of computers has been expanding into increasing number of fields. With the growing need for computationally intense applications in every field, it is necessary to constantly meet the demands of performance requirements. The performance of the system relies heavily upon the processor and therefore the development of this technology is crucial. The processors of this age are handling growing amounts of data that depend on the speed, efficiency and data handling capacity of the processors. The scope of pushing the performance of a single processor has reached its threshold owing to factors such as cost, heat and power consumed. This lead to the advent of the age of multicore processor technology. But this model is also approaching a plateau in its scope by 2017 as predicted by Moore's Law. It is imperative that an alternate and viable technology that produces high performance is found.

GPUs are the answer to the search for such a highly powerful yet feasible technology. These units are capable of handling computationally intense tasks by performing the operations on huge data sets in parallel. This type of parallel processing breaks the problem into discrete parts that can be solved concurrently. So while the conventional CPUs use the power of a single core to solve a problem, the GPU solves the same problem using about a hundred processors. So, while increasing the number of cores in a processor is not possible, integrating CPUs with GPUs and passing the intense workloads to the GPU, which will process it faster, to achieve an overall high performance is a viable model. Coherence between the two units is important to distribute the workload such that the parts with large data, that suit parallel processing is handled by GPU and the serial tasks are controlled by the CPU.

General Terms
Multicore processors, Graphic Processing Units.

Keywords
CPU, GPU, GPGPU, Parallel Programming, CUDA, OpenCL, Moore's Law.

1. INTRODUCTION
The processor is in essence the brain of a computer system. It is the part of the device that performs the arithmetic and floating point operations that is necessary for the functioning. Microprocessor or CPU is a complete computation engine that is fabricated on a single chip that consists of millions of transistors.

The performance of the system hence obviously depends directly on the processor's performance. To increase this facet of hardware, engineers scaled up the number of transistors in the processor. This improved the computing power by magnitudes. But the plateau of cramming large number of processors was reached with Intel's Tejas CPU. This CPU was projected to run at 7 GHz, but it never did so. It dissipated 150 watts at 2.8 GHz Core frequency. This lead to the advent of multicore technology for harnessing more computing power.

Looking at the current scenario, the fastest processors of our time is the Intel i7-3770k, with 4 cores and the CPU clocking maximum frequencies of 3.9 GHz utilizing 166-244 Watts power.[1]

2. CURRENT PERFORMANCE IMPROVEMENT TRENDS
When extreme amounts of computational capabilities are required, several methods are used by today's developers. High-performance computing can be achieved by utilizing many CPUs working in parallel so that computation-intensive programs can complete running in a relatively short amount of time. These clusters are aimed at achieving improved and high level of performance. Parallel programming paradigms involve two issues, one is the efficient use of CPUs on one process and the other is the communication between nodes to support interdependent parallel processes that run on different nodes and handling mutually dependent data. This is essentially parallel computing. A parallel program consists of a set of processes sharing data with each other by using a shared memory over an interconnected network. Another way is to accelerate existing hardware beyond normal operation. Some processor manufacturers like Intel's Turbo Boost allow the processor to overclock and achieve higher performance when there is enough power available, without any risk of overheating.

3. NEED FOR ALTERNATIVE
This trend of doubling the number of cores does not warrant a corresponding increase in the performance. A research by the US government's Sandia Labs found this increase actually exhibited less than linear improvement.

Further increases hit a wall due to the negative effects on power, cost and heat generation. It then decreased exponentially due to contention for memory bus and lack of memory bandwidth. Performance scaling for recent CPUs has failed to increase to the levels expected by the industry. This was predicted by Gordon Moore. This image shows the increase in transistor numbers over the years[2]

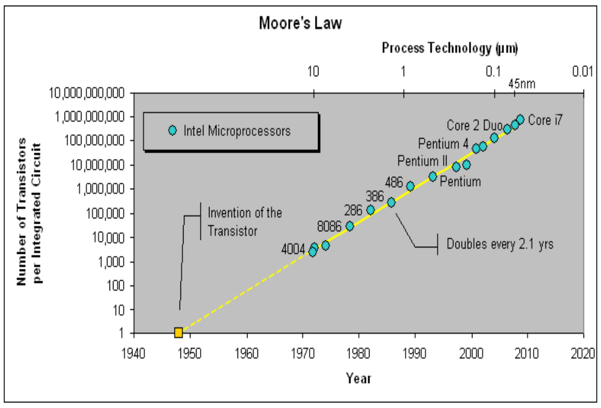


Fig 1: Graphical representation of Moore's Law

Moore's Law states that the number of transistors per square inch on integrated circuits had doubled every year since their invention. Although the pace has slowed, the number of transistors per square inch has since doubled approximately every 18 months. This is used as the current definition of Moore's law.[3]

The prediction projected by Moore's Law is that the limit to growth in the CPU market, where speeds are concerned, will be reached once transistors can be shrunk as small as atomic particles. After this point, there is no more way of accommodating more and more transistors in a single chip to derive high performance, as is the trend of this age.

Thus, throughput from sequential codes has reached the plateau of their performance between subsequent generations of the processors. Today's processor fabrication uses the process of creating masks using deep sub-micron photolithography, which uses light to make the conductive paths and electronic components that are found in our core's chips. This process also limits the development of core technology due to cost factor.

The move towards new processor technologies is about increased performance or efficiency under optimal economic cost. While the latest microprocessor architectures from Intel and AMD have shown performance increases, they haven't resulted in the doubling of performance that computer scientists relied upon the 70's through the 2000's to get performance increases in their code without changing the top level execution model.

4. GPGPU AS AN ALTERNATIVE

It has therefor become a necessity to look for alternative strategies to develop processor technologies that deliver high performance. The ideal characteristics are optimal power consumption, high processing speeds but at a lesser cost. Another important factor to consider is the heat generation, as high thermal levels can damage the hardware and also be harmful to the users.[4] This paper is going to analyze the use of GPUs alongside CPUs to improve performance at a feasible condition. That is, using the computational prowess of the GPU in cooperation with the task efficient CPUs to achieve an overall performance increase.

4.1 GPU and its Performance

GPU(Graphic Processor Unit), also called visual processing unit, is a specialized electronic circuit designed to rapidly manipulate and alter memory to speed up the creation of images in a frame buffer that is intended for output to a display.

In other words, GPUs are optimized for taking huge batches of data and performing the same operation repeatedly at very fast rate.

The GPUs goes far beyond basic graphics controller functions, and is a powerful computational device that can also be programmed.

Kepler-based Quadro K6000, which is considered the fastest GPU of our time, has 12GB of super-fast DDR5 graphics memory, 2,880 streaming multiprocessor cores, ultra-low latency video I/O, and the ability to drive four simultaneous displays at up to 4K resolution. [5]

This sort of high power that GPUs give can be put to use in varied fields. GPUs can operate at high rates and more cost efficiently than CPUs in an array of important sectors such as medicine, natural resources, national security and emergency services.[6] Its computational prowess is being harnessed to accelerate financial modeling, scientific research, oil and gas exploration.

The reasons GPUs increase performance because they employ hundreds of cores to work on a single program, processing different data sets simultaneously, thereby increasing the throughput tremendously. The simplest explanation is that a lot of extra hands make less work and enable finishing the work faster. But, for maximum output, cooperation with the main processor is vital. [7]This is covered in the later section.

4.2 Difference between CPUs and GPUs

On a basic hardware aspect, CPUs clock higher rates in their cores, whereas GPUs run on a much lesser clock frequency. But, GPU achieves its high computational capability through the use of its large number of cores. The power consumption in a CPU is high, while in GPU it is much lesser. Architecturally CPUs are composed of fewer cores with lots of cache memory and can only handle few threads at a time. However, GPUs are comprised of hundreds of cores that can process thousands of threads at the same time. This image probably explains the simplest difference between CPUs and GPUs. [8]

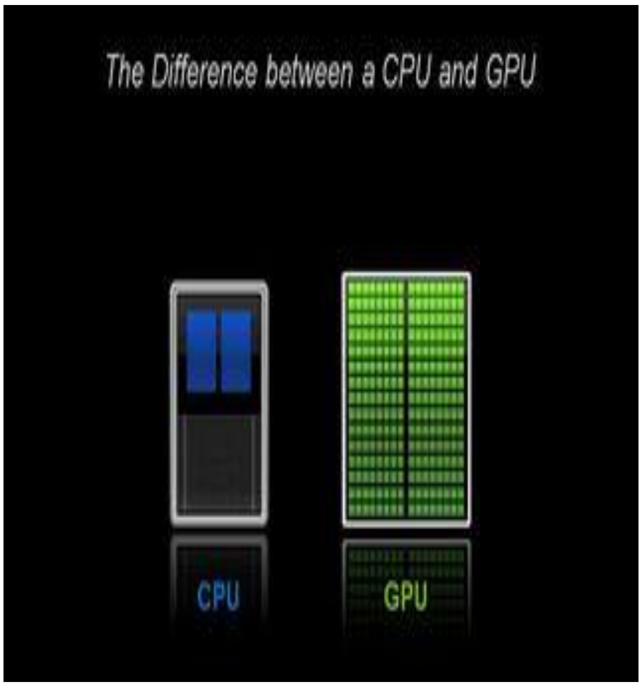


Fig 2: Difference between a CPU and GPU

required. CUDA is an excellent choice for computationally datasets since GPUs use enormous parallel interfaces to connect with its memory, which is 10 times faster than a CPU memory interface.Though CUDA was explicitly designed to work on Nvidia’s graphic cards, it can also run on CPUs, it CPU memory interface. Though CUDA was explicitly designed to work on Nvidia’s graphic cards, can also run on CPUs, though not as fast. CUDA platform is capable of supporting other computational interfaces, including the

While CPUs have fast caches which are great for data reuse, GPUs have lots of math units which make computation faster. CPU provides fine branching granularity whereas GPUs have faster access to the onboard memory. CPUs have high throughput on parallel tasks while on the other hand, GPUs give high throughput on parallel data. Also, while GPUs can run lots of processes or threads, GPUs run the same program on each fragment. Thus, CPUs are great for task parallelism and GPUs for data parallelism.

4.3 Introduction to GPGPU

GPGPU(General Purpose Graphic Processor Unit) work on the concept of SIMD(Single Instruction Multiple Data) architecture to process data. To get how GPUs works, an understanding of the concepts of stream and kernel are required. A large array of data is called stream and the operations to be performed on that data is called kernel. The point to note is that the same operation(kernel) is performed on all the data(stream). The data in the set can vary, but the operation to be performed on them remains the same. This is called stream processing and is the result of the SIMD architecture of the GPU. As it is apparently seen, when parallel processing of large amounts of data are to be operated on similarity this model is highly efficient.[9]

4.4 Available Programming Platforms

4.4.1 CUDA

CUDA (Computer Unified Device Architecture), is a C/C++ programming language based parallel computing platform offered by Nvidia for general purpose computation. CUDA is suited only for highly parallel applications since in order to run a program efficiently on a GPU, hundreds of threads are

4.4.2 OpenCL

OpenCL is a framework for programming that works across multiple platforms. It is an open standard platform that can work on all types of hardware that aims at parallel execution. This can be a CPU, GPU, DSP or FPGA.[11] It integrates well with C, C++, Python, Java and more.

OpenCL includes a language for writing routines called kernels and APIs that are used to define and control the GPGPU platforms. This can access GPGPUs from all supported GPU vendors.[12]

OpenCL kernels can run on different types of devices as well as dispatch kernels to multiple devices at once. That is, the kernels running on multiple devices can be synchronized and data can be shared between them.[13] Its functions and data structures are unique. [12]

Khrono’s Group’s OpenCL, Microsoft's Direct compute. Python, Perl, Fortan, Java, Ruby, Lua, Haskell, MATLAB and IDL are also supported using third party wrappers.[10]

CUDA is extremely well suited for handling huge amounts of data load, intense independent computing, CUDA utilises thousands of threads in parallel, that work on a set of varying data but applying the same operation (kernel) on them. It is capable of handling flow control statements like if..else statements even in this highly parallel architecture.

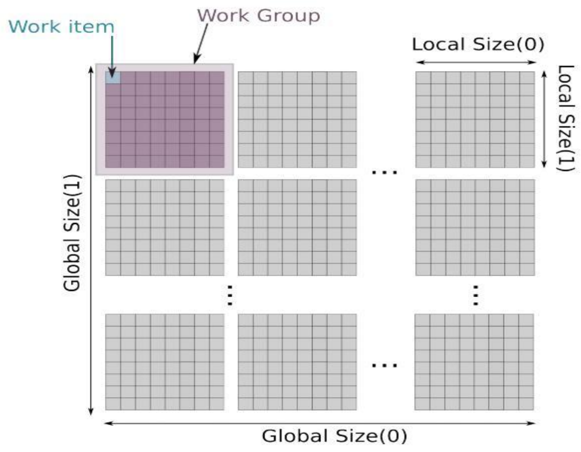


Fig 3: An illustration of the working of Parallel Processing[14]

5. CPU GPU INTEGRATION

CPU-GPU integration can be done using two methods:

- **Physical integration** of CPU and GPU on the same chip. Example, AMD Trinity, Intel's Ivy Bridge processors
- **Logical integration** to avoid explicit data copying through memory.

To implement shared memory, coherence of software and hardware is needed. But, these mechanisms affect the performance of the system. Also, coherence between CPUs and GPUs is difficult since GPGPU applications require a much higher bandwidth due to their SIMT execution model. Scaling to a higher bandwidth is challenging since it is difficult to support more than one instruction per cycle at the directory.

With throughput, memory bandwidth should also increase, which is facilitated by die stacking of DRAMs which is the process of vertically stacking them, hence reducing their footprint on the board and providing high bandwidths(in this case 1TB/s). Since GPUs are throughput oriented, it runs risk of overwhelming the CPUs if the requests are not well filtered by the cache.

To mitigate this, Heterogeneous System Coherence (HSC)[15] was developed for integrated CPU-GPU systems to reduce coherence bandwidth effects of GPU memory requests. This system brings about a massive bandwidth reduction. It reduces the problems due to the mismatched speeds of CPU and GPUs by easing the load on the coherence network and shifting to the high bandwidth direct access bus. In HSC, standard directory is replaced by regional directories and regional buffer is added in the L2 cache. HSC improves performance by an average of 2 times and maximum of 4.5 times of the directory protocol.

5.1 Analysis of Performance

To analyze the proposed model, we performed a test on the performances of CPU and GPU based on the Matrix Multiplication program. The CPU was separately tested by running the Matrix Multiplication using C++. The time taken for the CPU to perform this operation on the matrices of different sizes were recorded. The graph below shows how poorly the CPU handles such huge volumes of data. The analysis was performed on an Intel i7 processor running an 64-bit windows 8 system.

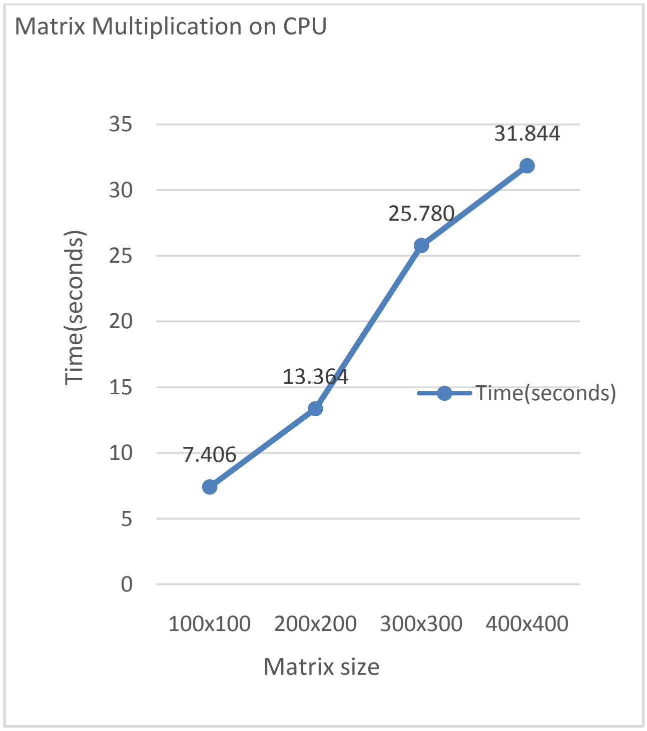
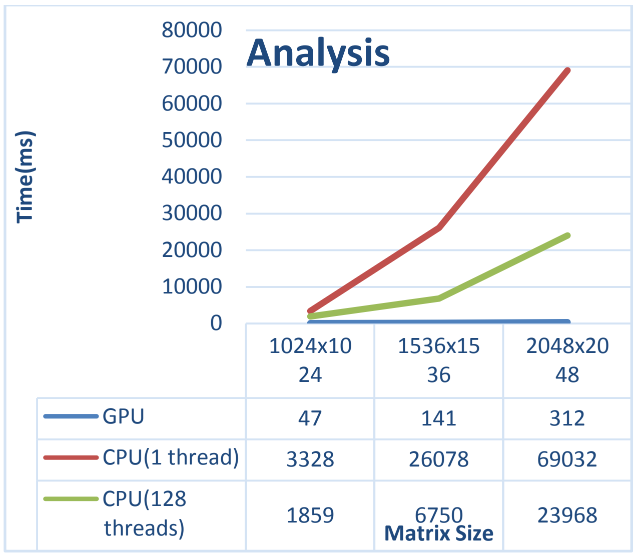


Fig 4 : Graph representing the performance analysis of CPUs for Matrix Multiplication

Further, we went on to analyse the GPU's performance in comparison to the CPU on handling parallel data, with the same matrix multiplication operation. We used CUDA programming platform that is able to utiliseNvidia's GPU unit for general computing. The test was performed using Nvidia reference program on matrix multiplication performance analysis using CUDA. The analysis was performed on Nvidia's GEFORCE GT 755M graphics card. This unit contains 384 cores and clocks 980 MHz core frequency. The CPU used was the same Intel i7 processor. Windows 8 operating system running a 64-bit architecture was used.The CPU's performance while running a single thread as well as 128 threads were considered. The second case(128 threads) is using a level of parallelism using the CPU itself.

As it can be infered from the graph, the GPU fared extremely well even under the high taxing 2048x2048 matrix multiplication while the CPU perfromed poorly both using single and multi threads in comparison. There was a vast difference in the performance seen. The analyis strinly supports the cause of using GPGPU's along with CPU to increse perfromances of such tasks.



	1024x1024	1536x1536	2048x2048
GPU	47	141	312
CPU(1 thread)	3328	26078	69032
CPU(128 threads)	1859	6750	23968

Fig 5: Graph depicting the performance of GPUs for Matrix Multiplication

5.2 Feasibility

As already mentioned GPUs only handle highly parallel operations on large datasets. But, during the working of our system, not all functions are concurrent. A level of serialism is required. For this reason combining the use of GPUs and CPUs and enabling them to perform in sync with each other will be very beneficial. Hence, our entire program need not be based on GPGPUs. Rather a big application that involves many functions and a user interface, then invoking the kernel functions created using the GPGPU programming platforms like CUDA or OpenCL only for those parts of your program that needs highly intense computation of large data and leaving the rest to the serial functioning of CPU would be an ideal model that would yield high performance. This type of combining both CPU and GPU will prove a feasible model of increasing performance through GPGPUs.

6. 6. CONCLUSION

GPU is an incredibly powerful unit that finds its use in varying applications and fields. Although, the processor technology is predicted to reach the end of its development scope by the addition of more cores, using alternate and creative ways like integrating GPUs ensure that the thirst for high performance is quenched. This technology is still in its nascent stage and the scope for its development is tremendous. This model is highly cost and power efficient and also delivers high performance. This might be the key to the future of processor technology. Possible areas where this integration will prove beneficial are highly computation intensive artificial intelligence machines, problems related to differential equations, numerical integration, linear algebra, number theory, computational quantum physics, molecular dynamics, coding involving images and videos. It may also be used for dealing with biomimicry. The area opens up a wide array of uses with the possibility of converting usual serial programs into programs with levels of parallelism in order to harness higher performance using GPGPUs. This conversion requires thinking in the parallel execution paradigm and hence needs to be explored.

7. REFERENCES

[1] Experiment on Intel's Tejus CPU: http://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck

[2] Graph depicting Moore's Law: http://forums.anandtech.com/showthread.php?t=2281246

[3] Moore's Law: http://computer.howstuffworks.com/moores-law3.htm

[4] GPGPU Technology: https://www.tacc.utexas.edu/news/feature-stories/2010/8-things-you-should-know-about-gpgpu-technology

[5] Parallel Computing Concepts: https://www2.cisl.ucar.edu/docs/parallel_concepts.

[6] Application of GPGPU in Computational Chemistry: http://blogs.nvidia.com/blog/2010/04/01/the-world-is-parallelgpus-in-chemistry-research/

[7] Working mechanism behind GPGPU: http://gizmodo.com/5252545/giz-explains-gpgpu-computing-and-why-itll-melt-your-face-off

[8] Difference between CPU and GPU: http://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

[9] Basics of GPGPU Programming : http://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units#GPGPU_programming_concepts

[10] Basics of CUDA: http://www.nvidia.com/object/cuda_home_new.html

[11] Working with OpenCL: http://streamcomputing.eu/knowledge/what-is/opencl/

[12] Introduction to OpenCL: http://www.drdobbs.com/parallel/a-gentle-introduction-to-opencl/231002854.

[13] An Introduction to the OpenCL Programming Model by Jonathan Tompson and Kristofer Schlachter

[14] 2D Data-Parallel execution in OpenCL (from [Boydstun2011])

[15] Heterogeneous System Coherence for Integrated CPU-GPU Systems by University of Wisconsin