OPTIMIZING REAL TIME GPU KERNELS USING FUZZY INFERENCE SYSTEM

Deepali Shinde[1], Mithilesh Said[2], Pratik Shetty[3], Swapnil Gharat[4]

[1,2,3]Student, [4]Lecturer, Computer Engineering Dept., Rajiv Gandhi Institute of Technology, University of Mumbai, (India)

ABSTRACT

CPU technology is slowly reaching its threshold, however Moore's Law still holds true for GPUs. With the increasing scope for GPGPU computing more and more applications are being ported to the GPU framework. One of the most suited application areas for GPGPU computing is image processing and computer vision. The high performance given by GPUs makes them ideal for real time applications. However, GPU technology gives optimum results when certain criteria related to degree of parallelism, image size and memory transfers are met. Very small images will consume more time in memory transfers between CPU and GPU than in computation on the GPU, while large images will affect the response time owing to the increased computation. It is necessary to strike a fine balance between the image size and the computation time. We propose to use Fuzzy Inference System to estimate the most suitable values for these parameters and to show the difference between CPU and GPU computing methods. Using these values from the FIS, a programmer can develop deeper insights into the performance of real-time systems using GPUs.

Keywords: CMT, Computer Vision, Fuzzy Inference System, GPGPU Computing, Image Processing.

I INTRODUCTION

GPGPU computing refers to the use of a GPU (graphics processing unit) along with a CPU to accelerate general-purpose scientific and engineering applications.

Pioneered by GPU manufacturers NVIDIA and AMD, GPU computing has quickly become an industry standard, enjoyed by millions of users worldwide and adopted by virtually all computing vendors [1]. GPU computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster. CPU + GPU is a powerful combination because CPUs, which consist of few cores, are optimized for serial processing, while GPUs consist of hundreds or even thousands of smaller, more efficient cores designed for parallel performance. Serial portions of the code run on the CPU while parallel portions run on the GPU. This plays a significant role in operations such as real-time Image Processing and Computer Vision where large matrices are involved comprising of pixel values, which can be operated on simultaneously to a certain extent.

GPUs are devices designed specifically for high performance. Their large number of processor cores enables faster computation. However there is a trade off when it comes to memory access. The transfer of data from CPU to GPU and then back from GPU to CPU takes place via the PCI express bus and is hence slow. There is also the question of where in the system the data resides. Even inside a GPU, the memory accesses from global memory are slower than the accesses from the local memory within each block. As a result image size and the number of blocks involved in a particular calculation play an important role. A GPU relies on the CPU to transfer data to the GPU memory, and to launch the GPU execution. For example, for FFT computation, the initialization of the GPU takes about 0.4 second. To benefit from the potential speedup that a GPU can provide, the task running on a GPU should be compute-intensive to offset the overhead of this initialization time.

Different image capturing devices can give images of varying sizes and of varying complexities. These images represent data that has imprecise boundaries i.e. it is difficult to categorize them into fixed sets. This is where fuzzy logic comes into play. Using a fuzzy inference system we can categorize the images into imprecise sets. We have used a Mamdani Fuzzy Inference system with "image size" and "Compute-time to Mean-time ratio" (CMT) for computation purpose as inputs to the system. This helps in deducing optimal values for the input parameters – image size, number of blocks and tile size.

II RELATED WORK

With the advancements being made in the GPU computing field more and more applications are being developed which take advantage of the GPGPU technology. For real time image processing or computer vision applications, we need to get quick response. Several algorithms and methodologies have been proposed by several authors that focus on the GPU implementation of IP algorithms like Hough Transform and reduction sweep. These algorithms focus on using the parallel architecture of GPUs to improve the speed of processing. The current generation of GPU is technically superior in terms of GFLOPs of processing power to that of current CPUs, but that power is rarely used to its full capabilities. Numerous advances have been made recently in applying the GPU to parallel matrix processing tasks, such as the Fast Fourier Transform.

In this section we will bring to light some previous work that has been done to clearly show the differences between CPU and GPU computation using several techniques. It is a known fact that GPUs offer varying measures of speedup for a range of applications as compared to CPUs. These measures have been analyzed and quantified by various researchers for different algorithms and applications using certain parameters. Miroslav Mintal [2] highlights the speed-up of GPUs over CPUs by demonstrating acceleration of distance matrix calculations to model pedestrian movement. He used a distance vector matrix to specify distance of each cell from the target(hence target cell distance was zero) and a gradient map which represents a vector field which shows a direction the pedestrian should take in order to reach a target. The pedestrian can arrive to the destination and avoid all obstacles using this map. Even though the algorithm could not be completely parallelized, it was optimized to a great extent.

Renato Farias et al. [3] presented a graph-based image segmentation algorithm called Reduction Sweep, designed with easy and efficient parallelization in mind. Because it only uses local image characteristics, the Reduction Sweep algorithm can compare and join many different regions of the present a graph-based image segmentation algorithm designed with easy and efficient parallelization in mind. Because it only uses local

image characteristics, the Reduction Sweep algorithm can compare and join many different regions of the image independently. This leads to faster execution without compromising on visual quality. No prior knowledge of the image is required, although tweaking the parameters may lead to better results for individual images. They then applied various forms of Reduction Sweep (example RS-Sequential, RS-Threads, RS-GPU and RS-Hybrid) on different images and then compared the difference in computation time.

But keeping computation speedups aside, there is one more important question to answer. Chris Gregg and Kim Hazelwood [4] raise a valid query about where in the system does the data to be processed reside and how the time taken to transfer this data between CPU and GPU via the PCI Express bus affects the overall time. They run various benchmarks (FFT, Convolution, Search, Sort, etc.) on small and large data sets where the kernels are divided into types varying from none to large data transfer from CPU to GPU and vice-versa. Using the transfer and computation times measured, they inferred how memory-transfer overhead affects large data sets which in turn made them propose a new ordered classification to label applications to indicate such dependencies.

III INTRODUCTION TO FUZZY INFERENCE SYSTEM

Fuzzy Logic is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both. Fuzzy logic offers soft computing paradigm the important concept of computing with words. FL's approach to control problems mimics how a person would make decisions, only much faster. A fuzzy inference system (FIS) is a system that uses fuzzy set theory to map inputs (*features* in the case of fuzzy classification) to outputs (*classes* in the case of fuzzy classification). The primary work of this system is decision making.

An FIS is constructed of five functional blocks. They are:

1. A *rule base* that contains numerous fuzzy IF-THEN rules.
2. A *database* that defines the membership functions of fuzzy sets used in fuzzy rules.
3. *Decision-making unit* that performs operation on the rules.
4. *Fuzzification interface unit* that converts the crisp quantities into fuzzy quantities.
5. *Defuzzification interface unit* that converts the fuzzy quantities into crisp quantities.

Initially, in the fuzzification unit, the crisp input is converted to a fuzzy input. After this process, rule base is formed. Database and rule base are collectively called the *knowledge base*. Finally, defuzzification process is carried out to produce crisp output. Mainly, the fuzzy rules are formed in the rule base and suitable decisions are made in the decision-making unit.

Mamdani Fuzzy Inference method is the most commonly used in applications, due to its simple structure of 'min-max' operations. It contains the following four steps:

Step 1: Evaluate the antecedent for each rule.

Step 2: Obtain each rule's conclusion.

Step 3: Aggregate conclusions.

Step 4: Defuzzification [5].

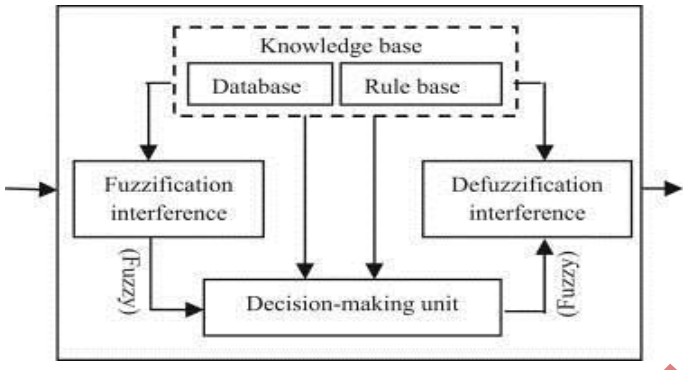


Fig. 1: Fuzzy Inference System block diagram

IV MEASURING GPU COMPUTATION PERFORMANCE

GPUs have advanced to a great extent over the past decade. Faster and robust GPU devices are being developed that support different APIs. In certain applications GPU computation provides a speedup of several hundred times. Many publications have claimed that GPU computation provides substantial speedup over multicore CPUs, while some others have emphasized on the fact that with fine tuning CPU codes can also be speeded up thereby reducing the gap between CPU and GPU computation. But in shear terms of computing capability (measured as number of floating point operations per second) GPUs tend to overmatch CPUs.

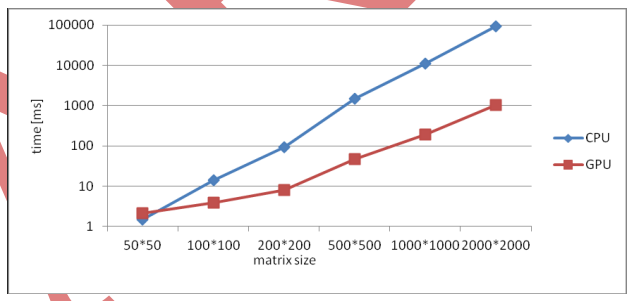


Fig. 2: Time comparison for Distance Matrix Calculation

The above figure shows the CPU and GPU execution by Miroslav Mintal of the distance matrix algorithm, where the CPU is an Intel® Core™ i7 Q740 processor and the GPU being an NVIDIA GeForce GT 420M. As seen in the figure, the GPU computation takes longer for smaller matrices. This is due to the necessary transfer between CPU and GPU memories. The GPU calculation of the large scale matrices is quicker, due to the larger scope for parallel computation. The algorithm can handle various sizes and shapes of the matrices, not only the square ones, the restriction being they have to fit into the global memory.

The high compute capability of GPUs comes at a cost. This cost is often the trade off induced in terms of memory transfers between the CPU (host) and the GPU (device). This transfer takes place on the PCI express bus which has limited bandwidth. General PCIe x16 G2 bus gives around 8GB/s data transfer rate at peak performance. Another major aspect to consider while measuring GPU computation performance is the time required to access data from the global memory of the GPU. As a result it is important to consider these factors while measuring the performance of the system.

With the advent of DDR5 memory for NVIDIA GPUs, 1.85 GHz memory clock cycles can deliver up to 177.5GB/s of global memory access bandwidth. Also, the PCI 3.0 technology promises to leverage the data transfer between the devices. However these technologies are quite costly in monetary terms. For low cost solutions we need to fine tune the existing systems to boost application performance.

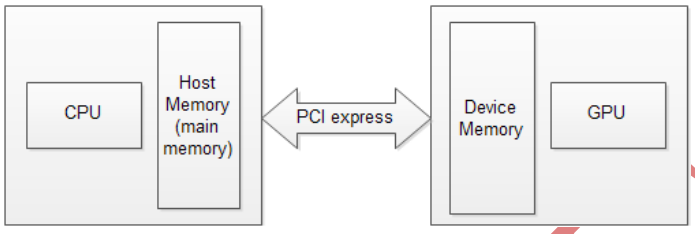


Fig. 3: Memory transfer bottleneck between CPU and GPU

Global memory access is dependent on the GPU device and may vary from device to device. However data transfer between CPU and GPU still remains an issue in most of the systems as it takes place on the common PCI express bus and can act as a bottleneck. Several possible solutions have been suggested to overcome these problems and to improve performance. For example,

1. Unified Virtual Addressing(UVA)
   With UVA, the host memory and the device memories of all installed supported devices share a single virtual address space [4]. This reduces the overhead of maintaining separate pointers for different device memories.
2. Concurrent Copy and Execute strategies
   Concurrent copy and execute refers to the process of overlapping kernel execution with data transfer. This method can be used when the chunks of data are independent of each other and they can be operated on as and when they are transferred to the GPU.
3. Zero Copy for integrated CPUs and GPUs
   Zero copy allows the device to get access to the host memory directly.
4. Kernel fusion and GPU resident processing pipelines
   Kernel fusion is a technique of merging two or more kernels to form a single kernel. This way data transfers can be reduced if the data to be operated on is the same for those kernels. This concept is similar to the idea of loop optimization in compilers. Furthermore it leads to the idea of running GPU resident pipelines where several operations to be performed on the data are merged together in a single kernel. Using such pipelines is a common practice in computer vision and other real-time applications.

Thus it is clear that several strategies are used to overcome the long latency induced due to memory transfers from CPU to GPU and back.

To measure the effect of memory transfer on GPU computation we have used a term known as Compute-time to Mean-time ratio or the CMT ratio:

$$CMT = CT/(t1 + CT + t2) \qquad (1)$$

Where,

CMT = Compute-time to Mean-time ratio

CT = time required for running the kernel (computation)

t1 = time required to transfer data from CPU to GPU

t2 = time required to transfer data from GPU to CPU

Compute-time to Mean-time ratio depends on factors such as the number of memory allocation and memory copy operations in the program, kernel operations and nature of the kernel i.e. serial or parallel. A programmer can fuse multiple kernels into a single kernel which then operates on a single set of data. This can increase the CMT ratio to a considerable extent. On the other hand if too many memory transfer operations are required, while the amount of computation is small, it can result in a very low CMT ratio. Let us take the simple example of real-time image processing where there may be several operations which need to be performed on a single image in one go. In such cases the effect of data transfer can be offset by kernel fusion.

However when talking about real-time Computer Vision systems, "how much is too much?" is a question to be asked. Even a high CMT ratio may not ensure good performance if the size of the image is too large and the processing time required is also too large. At the same time, a very small image may not provide enough information. Thus a right balance needs to be struck for the image size. Also the CMT ratio of the algorithm plays an important role. However, deciding these parameters is a human task. It therefore needs a solution that fits the human decision making system. A system that gets closest to human decision making is Fuzzy Logic and Fuzzy Inference System (FIS). We have used Mamdani FIS to model these parameters and help a programmer in deciding their optimal values to leverage application performance.

The FIS has two inputs namely,

- CMT ratio
- Image size

Computation time is the output. Ranges of values for the inputs and the output are decided by using values from results of experiments performed on different sizes of images and for different operations which give different CMT ratios. The fuzzy rule base is decided using these values as reference. Let us take a look at the system and its results for different image sizes and different CMT ratios.

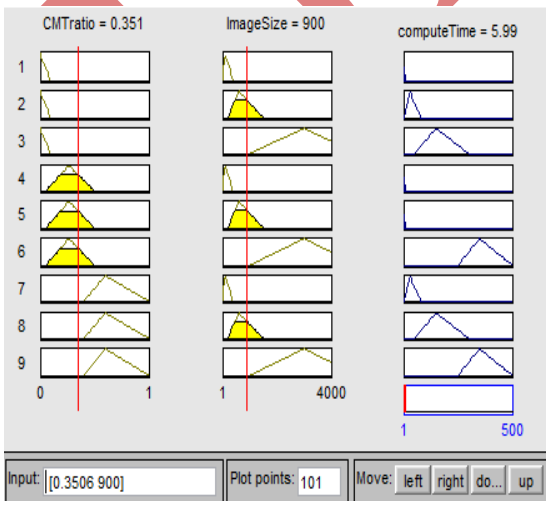


Fig. 4

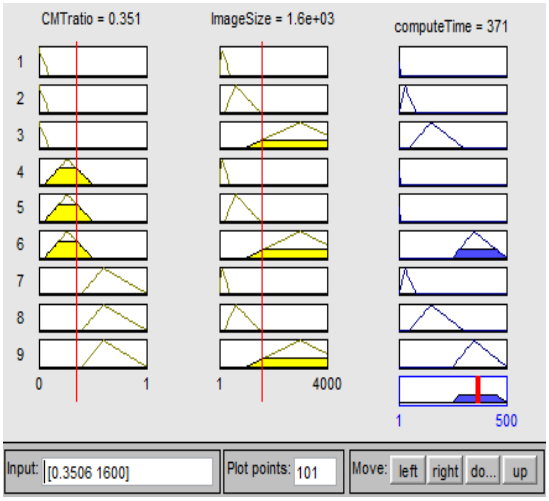


Fig. 5

Fig. 4: Estimation of Computation time for image size of 900x900 pixels on a GPU an

Fig. 5: Estimation of computation time for an image of 1600x1600 pixels on a GPU

As seen from the above two images, the computation time changes drastically for change in image size even at a constant CMT ratio indicating that the memory transfer time increases for a large image. For a programmer to judge the optimal values for image size and CMT ratio, the plot depicted in Fig. 6 is useful as it provides a range of values of computing time. However, decision of these parameters lies totally with the programmer. FIS can only help him in deciding the best possible values for good performance.

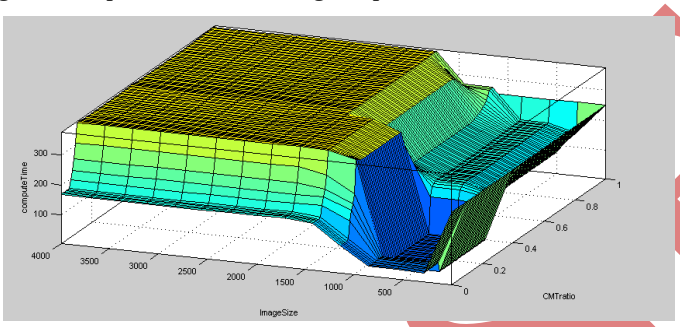
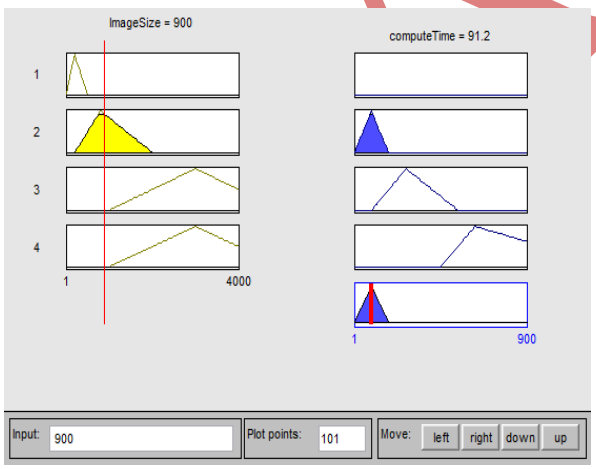


Fig. 6: 3D surface view plot for FIS

X-axis represents CMT ratio, Y-axis represents Image size and Z-axis represents computation time

Similarly, a FIS for the CPU also has been designed which helps us to show the relative performances of both, CPUs and GPUs. FIS for CPU has only one input – image size - and one output – computation time. There is no memory transfer overhead involved as the image is available to the CPU on-chip. Hence we have not included the CMT ratio as an input to the CPU system.



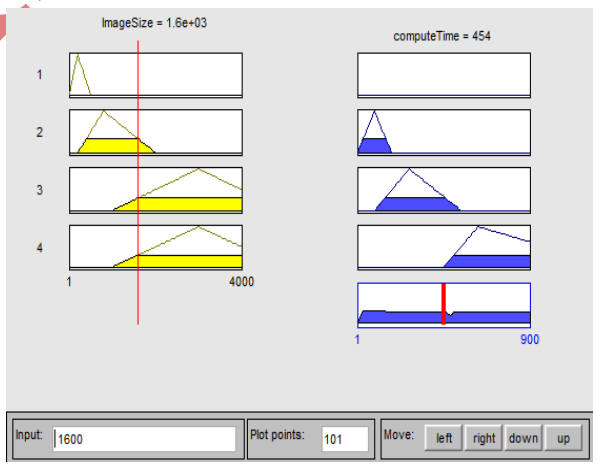


Fig. 7 **Fig. 8**

Fig. 7: Estimation of computation time for image size of 900x900 pixels on a CPU

Fig. 8: Estimation of computation time for image size of 1600x1600 pixels on a GPU

For the medium sized (900x900 pixel) images, the computation time on a CPU is comparatively low owing to the size of the image. However, for large image sizes (1600x1600 pixels), the computation time for a CPU is

substantially larger than that of a GPU. This large computation time arises from the large number of operations performed on the image. GPUs are very well suited for such parallel tasks. However, for small image sizes, CPUs can perform well. GPUs are affected by the memory transfers.

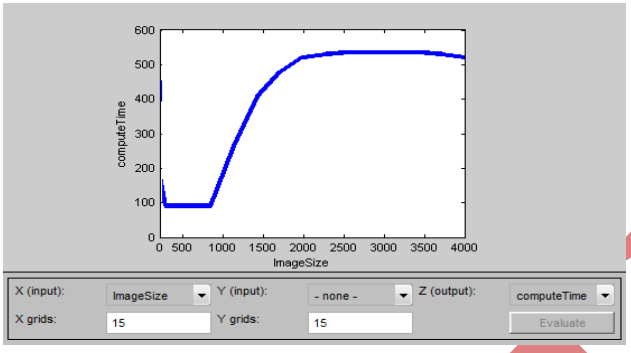


Fig. 9: Plot of FIS for CPU computation.

The availability of data on-chip is a beneficial factor for the CPU. But, computationally GPUs are much faster than CPUs and hence deliver a better overall performance.

V CONCLUSION

In this article, we point out the pros and cons of GPGPU computing. Overall speedup depends on various factors such as ratio of computation time to memory transfer time, size of data set and also optimization of code for CPU and GPU. This speedup can be summarized into the following table.

Table 1: Computation time for different values of Image size for CPU and GPU.

	Image Size (in px)	CMT ratio	Compute-time (in ms)
GPU	900x900	0.3506	5.99
	1600x1600	0.3506	371
CPU	900x900	-	91.2
	1600x1600	-	454

Note: the value of the CMT ratio may vary from algorithm to algorithm and depends on the strategy used.

Therefore, to strike a balance between these factors and achieve most optimal results i.e. quicker response for applications ported to GPUs we propose the use of Mamdani Fuzzy Inference System. Like desktop and laptop GPUs, mobile GPUs also suffer from long latency of the off-chip global memory [7]. Using the Mamdani model, developers can verify what combination of CMT ratio and image size yields the most desired outcome. Real-time applications require high performance as they need faster response. This fuzzy inference system provides an insight into the design strategy which a developer can follow in order to achieve this high performance. This methodology is not restricted only to Image processing and Computer vision applications. It can be extended to other applications of GPU computing as well.

REFERENCES

[1] http://www.nvidia.com/object/what-is-gpu-computing.html

[2] Miroslav Mintál, "*Accelerating Distance Matrix Calculations Utilizing GPU*", Journal of Information, Control and Management Systems, Vol 10, No 1 (2012), Available:http://kifri.fri.uniza.sk/ojs/index.php/JICMS/article/view/1363

[3] Renato Farias, Ricardo Farias, Ricardo Marroquim, "*Parallel Image Segmentation Using Reduction-Sweeps On Multicore Processors and GPUs*",SIBGRAPI 2013, Available: http://www.ucsp.edu.pe/sibgrapi2013/eproceedings/technical/114627_2.pdf

[4] Chris Gregg, Kim Hazelwood, "*Where is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer*", International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, TX. April 2011.

[5] S.N. Sivanandam, S.N. Deepa, "*Principles of Soft Computing, Second Edition",* Wiley India Pvt. Ltd., New Delhi.

[6] "*CUDA C BEST PRACTICES GUIDE*", DG-05603-001_v4.1 | January 2012, Design Guide.

[7] Guohui Wang, Yingen Xiong, Jay Yun, and Joseph R. Cavallaro "*Accelerating computer vision algorithms using opencl framework on the mobile GPU- a case study"*, ICASSP 2013, Vancouver Convention & Exhibition centre, May 26-31, 2013.