International Journal of Research in Advent Technology (E-ISSN: 2321-9637) Special Issue
National Conference "IAEISDISE 2014", 12-13 September 2014

Exploiting the Systematic Optimization Techniques for the present Heterogeneous Multi-core and Many-core Computing Systems

Chetna Dabas[1]
Department of Computer Science and Engineering [1],
Jaypee Institute of Information Technology[1]
Email: cherry.dabas@gmail.com[1]

Abstract- Stagnancy has been noticed in the Moore's Law era for CPU computing across the timeline. CPU's are not getting any noticeable faster due to a saturation point in the underlying hardware. It is due to some crucial constraints like power consumption in computation or design constraints. The focus is shifting towards the hardware to the software optimizations to handle the future prototyping or simulations which may be of hundreds of times faster in computing than the present multi-core and many-core systems. To an extent, even the capabilities of the present multi-core or many-core systems are not fully exploited to extract the maximum performance. This research paper aims to focus on the exploiting and presenting some of such parallel and systematic optimization techniques to the existing multi-core and many core systems in the heterogeneous computing environment and proposing their extensions to the future heterogeneous multi-core and many-core systems. This research paper also presents the implementation of the proposed techniques in form of a small run able application.

Index Terms- Multi-core; Optimization; Hetrogenous Computing; Many-core.

1. INTRODUCTION

The term heterogeneous computing refers to specific systems that are incorporated with more than one type of processor. These types of systems may be multi-core or many core systems in which the performance boost is achieved by inculcating multiple numbers of cores. These systems work in accordance with the design of co-processor model and allow handling of specific complex tasks by providing expertise capabilities in order to handle various tasks. As far as research is concerned, the heterogeneous computing has not only proven its capability in enabling us imagine the N-body simulation which is an NP hard problem but it caters its strength in affecting human lives in an impressive positive way. It takes lots of the most interesting applications on such kind of systems, such as, simulation of grid, fluid flow problems or ray tracers etc.

A commendable example application of heterogeneous computing is in Computerized Tomography Scanners (CT) for the reduction of X-rays in CT Scans which literally has affected lives. The Computerized Tomography can be considered as one of the computerized images techniques which hold its take off long back and still being used as one the prime useful techniques. The CT scanner performs the job of rotating around the body of the patient and taking a series of X-ray pictures. These images are then integrated and reduced to develop an image of the cross-section of the body. These slices then further may be used by the machine to produce a 3D model of affected section of the body. The X-Ray radiations generated as a part of this process are very harmful to the body. By the use of supercomputing, the amount of harmful X-ray radiations have been lowered by an impressive factor of 20x with much clearer picture generated. More importantly, the Cone Beam CT plays an important part in the IGRT (Image-Guided radiation Therapy) which is used in cancer treatment. In IGRT, during radiation therapy, iterative scans are conducted in order to target the tumors precisely, and minimize radiation damage in the surrounding tissue is highly crucial there. In such applications, the supercomputing technology has played a thankful role where it not only has raised the procedure as safer, faster and clearer but also has affected lives in a big affirmative way [1, 3].

Apart from the lucrative constructs and designs that the supercomputing technology provides in connection with the present heterogeneous Multi-core and Many-core computing systems, the fact that still persists is that still there exist a lot of scope in making

the computing systems still more faster. This may be achieved with the help of a proper sequence of systematic optimization techniques.

2. EVOLUTION OF COMPUTING

First, the world witnessed the evolution of the Central Processing Unit (CPU) which has crossed various milestones across the yester years. The multi-cycle operation in the simplest Central processing Unit was composed up of the well known sequence of the fetch, decode and execute phases. Here, the first phase was loading the instruction to be executed, the second phase was decoding the instruction or register read operation. The third phase was the execution operation and formally the last one was writing the results back or in other terms finishing off the execution process. The next evolution of CPU's consisted up of pipelining where multiple instructions were allowed in-flight at once. Modern CPU's used longer pipelines. The major problems related with pipelining process were mainly the branch delay and memory latency.

3. BACKGROUND AND WORK

Systematic optimization techniques refer to systematically optimizing the parallel problems on multi-core and many-core computing platforms. It specifically refers to the repeated and feedback sequence of analyzing, parallelizing, optimizing and then deploying the application in a circular fashion. The precise goal while optimizing the multi-core or many core parallel programs is to blend the problem faster or solve a bigger problem or solve a complex problem.

There are certain characterizations of optimizations in the supercomputing scenario. The characterizations include choosing good algorithms, in other words it means choosing an algorithm which is fundamentally parallel. As an example if a heap sort algorithm is taken under consideration; it works better on the Central Processing Unit rather than the Graphics Processing Unit since it keeps updating sharable data structure called heap. The next characterization may be the architecture specific detailed optimizations. There may be yet another characterization called the micro optimization at the instruction level. Apart from keeping these optimizations in the mind, the prime idea is to perform the optimizations in a systematic manner. In the systematic optimization process for the multi-core and many-core systems, the analysis part points to profiling the entire application in order to extract the important information like by what factor it can benefit or how it can benefit?. It specifically means the understanding of the hotspots in the application under consideration and understanding weak and strong scaling of the problem under consideration. The parallelize part crucially depicts the choice of an approach like libraries, directives or programming languages and the choice of an algorithm. The optimize part in sequence specifically refers to the profile driven optimization. The deploy phase incorporated the multi-core accelerated code. All these phases happen in a cyclic manner and need to be implemented in real for performance enhancement on multi-cores and many core systems [2, 5, 6, 7].

The prime idea of the work presented here is to apply such systematic optimization techniques to an application. The application taken under consideration is the process of converting the Data Matrix code stored in row major order on a computer system to column major order. A Data Matrix code is specifically a matrix barcode consisting up of black and white cells arranged in square or rectangular pattern. The application under consideration uses a square pattern and dark cell and white cell corresponds to 0 and 1 values respectively. Data matrix codes are normally verified by hardware equipment and software. For this purpose, some computer processing is required. After the completion of this process on computer, which in between, requires a translation of the data matrix data to be stored in column major order on the system for processing (which has to be fast enough for good performance), the data matrix code is send to a reader camera which decodes the data for various purposes like stock verification etc.

The next section presents the detailed specifications of the work environment, system specifications, working of the application with its implementation snapshots, performance analysis and systematic optimization with analysis and results.

3.1. *Work Environment and System Specifications*

The work presented as a part of this research paper is carried out using CUDA version 5.0 [5], Visual Studio 2010 and on the system with the following specifications:

- Operating system: Microsoft Windows XP, 32-bit (service pack)
- DirectX runtime Version: 9.0
- Graphics Card Information:
- Driver Version - 306.94
- DirectX Support – 10
- CUDA Cores – 16

- Core Clock – 500 MHz
- Shader Clock – 1026 MHz
- Memory Data Rate – 666 MHz

- Memory Interface – 128 bit
- Memory – 512 MB
- Memory Type – DDR2
- Video Bios Version - 60.86.34.10.96
- IRQ – 16
- Bus – PCI Express X16

In the presented work, GeForce 8500 GT graphics card was used for the experimentation.

It has 2 MP(s) x 8 (Cores 1MP) =16 cores. The compute performance scaling factor associated with the same was 12.0.

3.2. *Working and Implementation Snapshots*

This small application namely Data Matrix Application is primarily responsible for collecting data from the Data Matrix Data which is assumed to be stored in the row major order then converting it into column major order and storing it back. This application has two major versions. One version is the basic serial and sequential version. The other one is the parallel version which has further optimized versions depending upon the levels of systematic optimizations being incorporated. During all the parallel versions of the Data Matrix application, there is one common pattern of data communication that was followed. This incorporates a heterogeneous computing environment which involves the computation by both the CPU and the GPU. The GPU under consideration further has 16 cores. The data here in the presented scenario, is initially stored on the in the memory having direct interaction with the CPU. The CPU then initiates the process of sending the entire data to the GPU, the GPU then performs the computation and return backs the results. All this data communication amongst the CPU and the GPU having 16 CUDA cores happens via the PCI Express X16 bus.

Fig. 1 corresponds to a typical snapshot of the visual studio 2010 project containing proposed Data Matrix application namely Chetna7 containing trace application running in Visual Studio 2010 with CUDA and NVIDIA Visual Profiler running in the Task Manager. Fig. 2 is a snapshot of the CPU usage of the system, it also depicts the status of physical memory in terms of total memory, available memory and the system cache information associated with the system used for the experimentation of the presented work. Fig. 3 shows the CUDA 5.0 correctly installed and running on the system of experimentation. CUDA 5.0 is a parallel language in which the proposed Data matrix application is designed.

The details of various optimized versions of the Data Matrix parallel application are explained in the up next paragraph.

3.3. *Analysis and Results*

Fig. 4 represents a snapshot of the zoomed view of the timeline of the Data Matrix application with its detailed graph description. It depicts global load efficiency, global store efficiency, duration and throughput etc. The global load efficiency which bears a value of 1 here represents that how efficient the global loads actually were or in other words, how many bytes were actually useful of all the bytes that were retrieved with each memory transaction from the Data Matrix. Fig. 5, projects one of the snapshots of the parallel versions of the running Data Matrix application with the compute kernel details.

As a prime and base part of the Data Matrix application, a serial version of the process of picking up of the Data Matrix data and conversion of its data elements from a row major order to column major order was performed. It took a total time duration of 157.7ms which was quite high.

Then after analyzing the execution time of the serial version, the application was refined to its first parallel version which included the process of launching 1 thread corresponding to each row in the Data Matrix. After this version was executed, the execution time retrieved came out to be 5.8ms. Here the problem was low DRAM utilization. Then second parallel version was designed where one thread was associated with each element in the data matrix and the goal was to keep coalesced reads and writes. But again it was observed that the DRAM utilization parameter was bearing a low value, a little raised than the previous version. It gave an execution time of 0.75ms which was quite better if compared with its serial counter. Hence the performance in multi-core systems may be enhanced by incorporating systematic optimization techniques on heterogeneous computing systems.

The results obtained are listed in the Table 1 below.

Table 1. Results

S.No.	**Systematic Optimization of Data Matrix Application on 16 core System**		
	Version	Time taken	DRAM Utilization
1	Serial	157.7ms	0.05%
2	Parallel	5.8ms	3.8%
3	Parallel per Element	0.75ms	24.1%

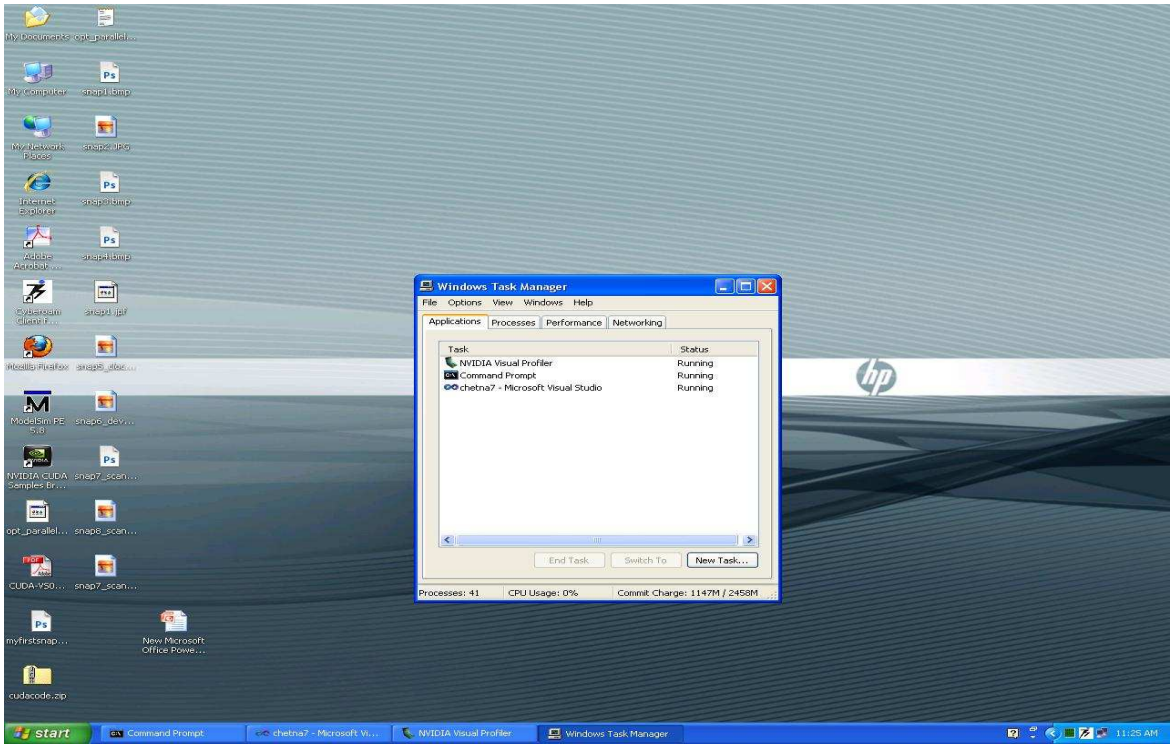

Fig. 1. A snapshot of the Project containing the proposed Data Matrix Application namely Chetna7 running in Visual Studio 2010 with CUDA and NVIDIA Visual Profiler running in the Task Manager

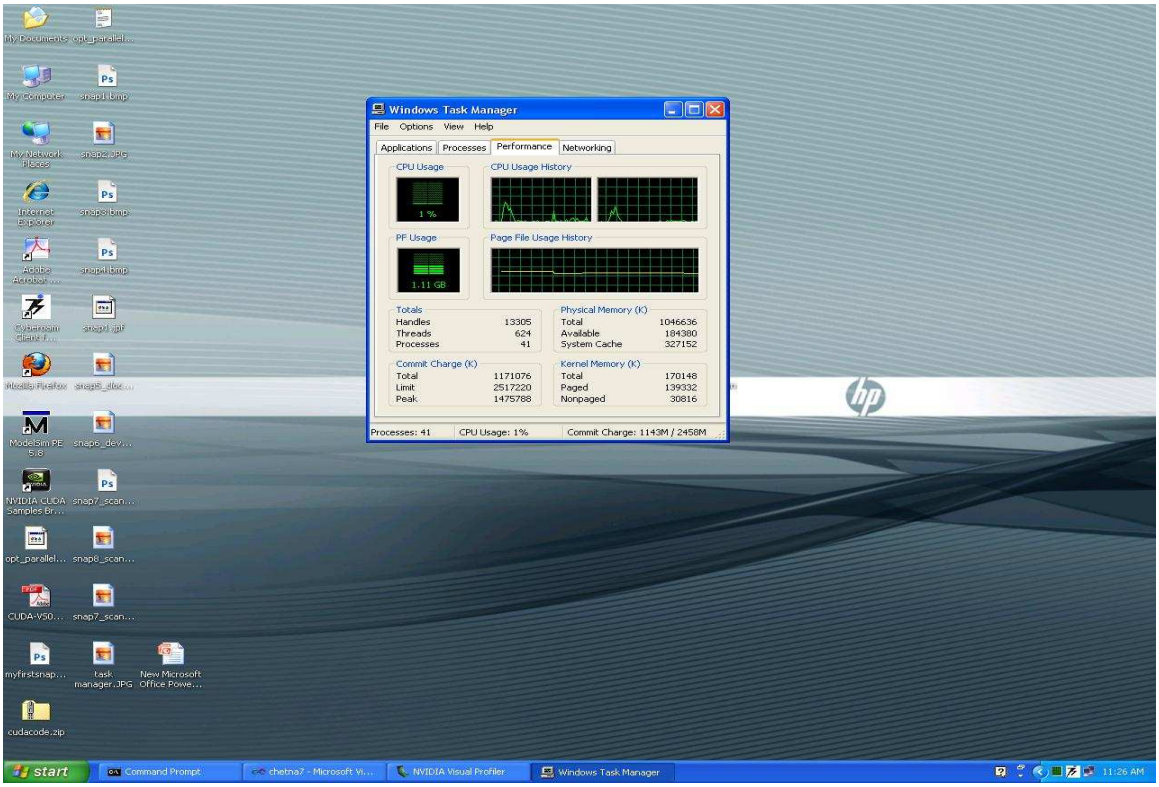

Fig. 2. A Snapshot of the CPU Usage of the System along with the status of Physical Memory

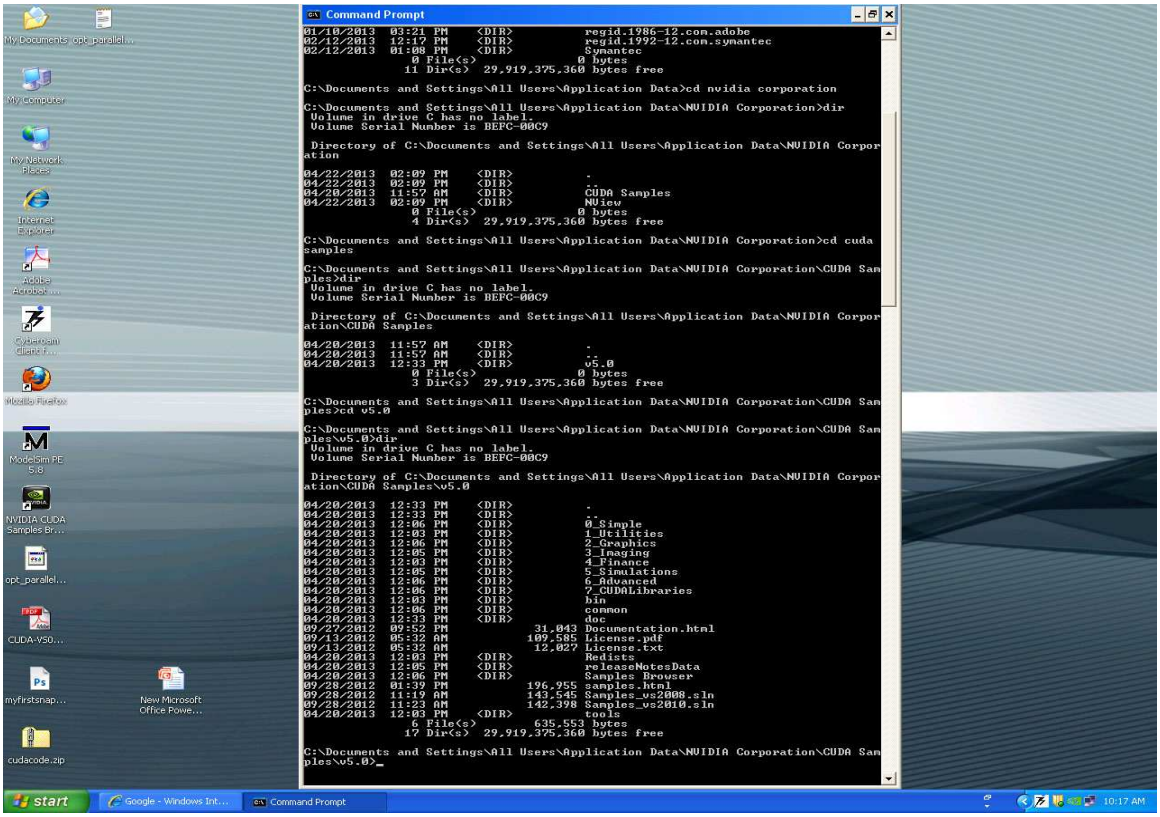

Fig. 3. A Snapshot of CUDA v5.0 correctly installed and running on the system

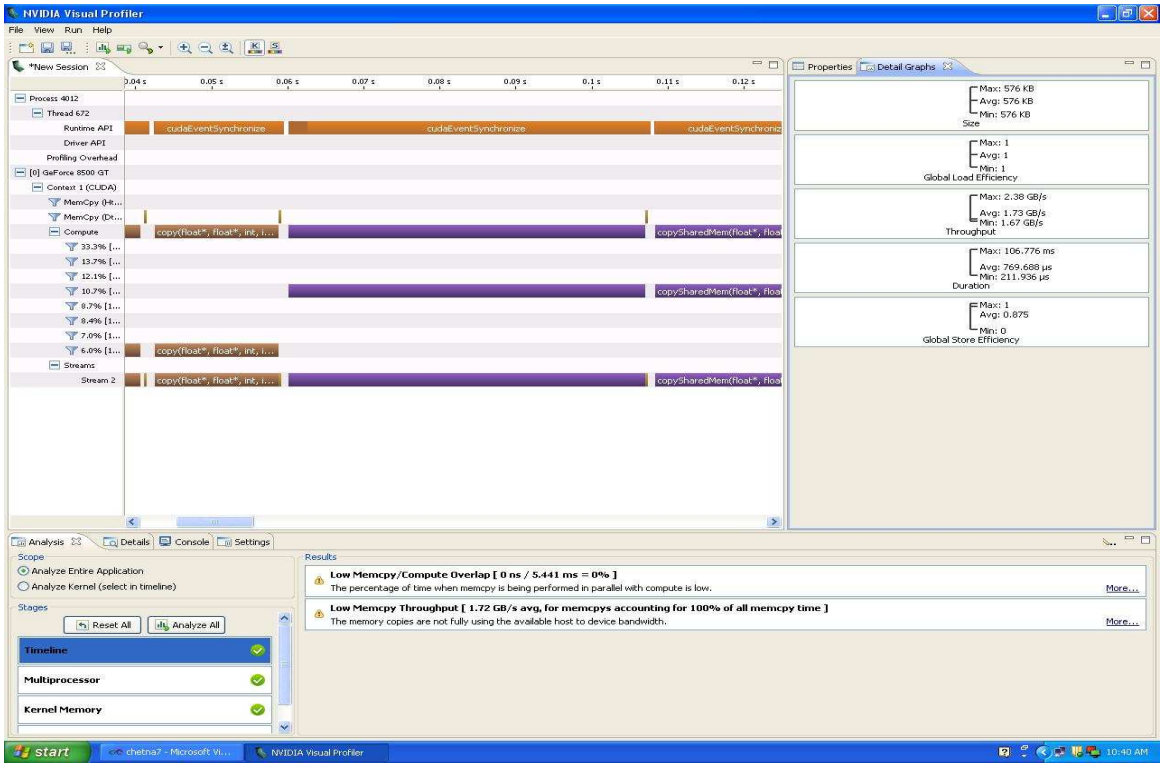

Fig. 4. A Snapshot of the zoomed view of the timeline of the Data Matrix application with detailed graph

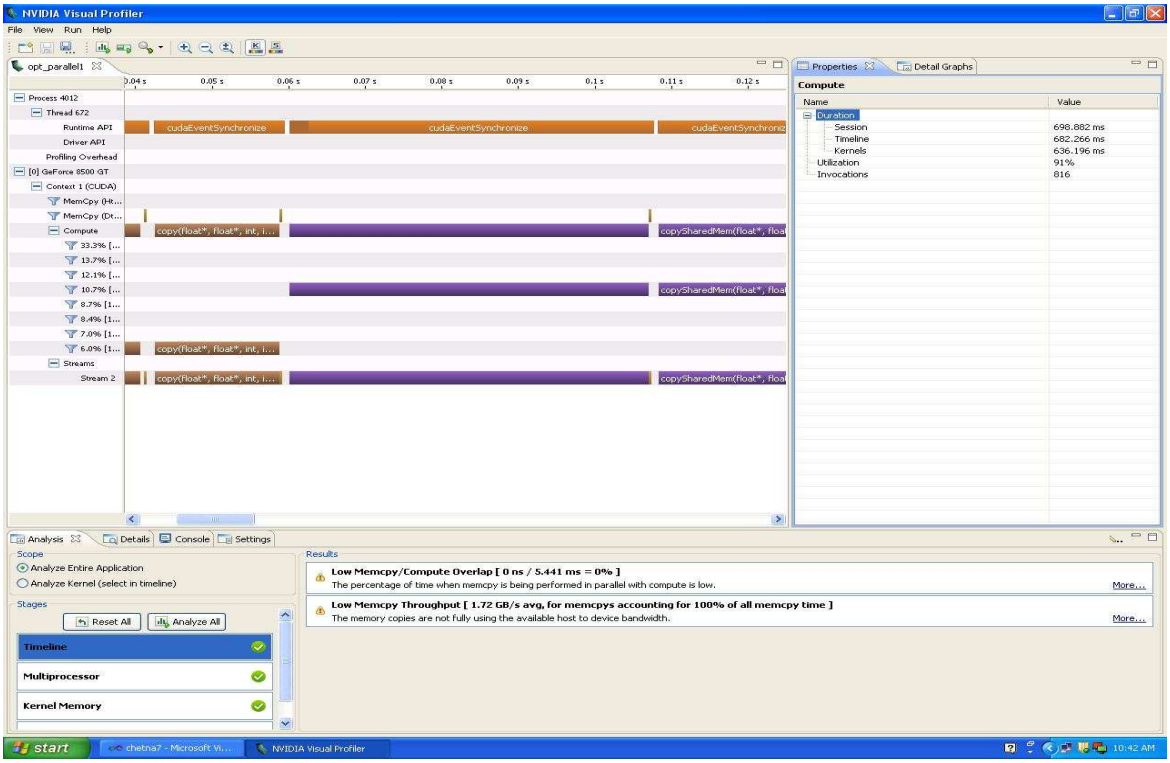

Fig. 5. A Snapshot of the Data Matrix application with its associated properties View

4. CONCLUSIONS AND LIMITATIONS

If the optimizing strategies are applied in a systematic manner and analysis and monitoring of bottlenecks is carried out at each stage, the performance of applications on the present multi-core systems may be enhanced by a significant level. It was a small enhancement across various stages for the heterogeneous system under consideration with 16 cores; this enhancement may be quite higher on systems with 32 cores or even higher on many core tiled processors. The limitation of the proposed work is that the results may vary from one computing platform to another and one application to another.

REFERENCES

[1] Guillem, Pratx and Lei Xing, "GPU computing in medical physics: A Review," Medicine Physics, 38 (5), 2685-2697 (2011).

[2] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips, "GPU Computing," In Proceedings of the IEEE, Vol.26, no. 5, pp. 879-899,May 2008.

[3] Michael D. Mc Cool, "Scalable Programming Models for Massively Multi-core processor," In Proceedings of the IEEE, Vol.96, no. 5, pp. 816-831, May 2008.

[5] Nvidia,"CUDA".http://www.nvidia. com/objectlcuda_home_new.html, Mar. 24, 2012.

[6] Rui Yang, Johnson Thomas,"Processing Dependent Tasks on a Heterogeneous GPU Resource Architecture," In 2nd IEEE International Conference on Parallel, Distributed and Grid Computing," pp. 627-632, 2012.

[7] Chuntao Hong, Dehao Chen, Wenguang Chen, Weimin Zheng, Haibo Lin, "MapCG: writing parallel program portable between CPU and GPU", Proceedings of the 19th international conference on Parallel architectures and compilation techniques, September 11-15, 20 I 0, Vienna, Austria.

[8] K. Asanovic, R. Bodik, B. Catanzaro et al., "The landscape of parallel computing research: A view from Berkeley,"EECS, University of California, Berkeley, Tech. Rep.UCB/EECS-2006-183, 2006