Parallel Acceleration on Manycore Systems Using OpenCL and Its Performance Analysis

Rafael Alejandro Vejarano, Phuong Thi Yen, and Jeong-Gun Lee

Dept. of Computer Engineering, Hallym University, South Korea
Email: {rafaelvejarano, phuongyen, jeonggun.lee}@hallym.ac.kr

Abstract. OpenCL (Open Computing Language) is a programming framework for developing applications which efficiently are mapped to heterogeneous to homogeneous, single or multiple device system consisting of CPUs, GPUs and others types of devices. In this paper, we consider the OpenCl's portability for multiplying large matrices. We present three different scenarios: serial, CPU and GPU solutions. Then we compare and analyze the corresponding performances for those implementations. The analysis is carried out to understand manycore CPU and GPU performance characteristics.

Keywords: OpenCL, Matrix Multiplication, Kernel, Work-item.

1.0 Introduction

Nowadays, processors with two or more cores are common. However, parallel software developers must contend with problems not encountered during sequential program development so that it is necessary to understand the characteristics of parallel designs of applications on CPU and GPU [1]. *Parallel Computing* lets us solve computational and data-intensive problems using *manycore* processors. Currently, GPUs (Graphics Processing Units) are used not only for graphics applications, but also for non-graphics applications, so-called GPU computing or GPGPU (General-Purpose computation on GPUs) [2]. Due to their high floating-point operation rates and memory bandwidths, GPUs can accelerate various science and engineering computations.

The main goal of this paper is to investigate OpenCL as a programming standard for parallel processing architectures and provide some analytical background of the parallelized application so that parallel software engineers can decide to map the software functionalities to the devices showing best performance between manycore CPU and GPU. A matrix multiplication program is used as our target benchmark since it is well known and used very widely. Two versions of matrix multiplications are implemented using OpenCL language over CPU and GPU devices.

2 Experiment Results

For sequential runs, the matrix size starts with data elements randomly generated and only one core was fully occupied. The size was varied from 1000x1000 until 6000x6000. While increasing the size of the array, the serial program execution time considerably increases. Execution times in serial and parallel versions over CPU and GPU devices are presented in Table 2. Note the times are only for kernel executions.

Table 2 Execution times and speedup over CPU and GPU devices

Matrix Size	T_{serial} (s) (serial version)	T_{CPU} (s) (parallel version)	Speedup T_{serial}/T_{CPU}	T_{GPU} (s) (parallel version)	Speedup T_{serial}/T_{GPU}
1000	11.283	0.774	14.68	0.156	72.33
2000	92.188	4.718	19.54	0.548	168.23
3000	405.846	18.831	21.55	1.411	287.63
4000	967.419	42.519	22.75	3.222	300.25
5000	2,058.774	98.305	20.94	6.083	338.45
6000	3535.899	166.963	21.18	10.499	336.78

As expected, parallel implementation outperforms the serial one and powerful 336-core GPU device outperforms 8-core CPU device. Note that we observe *super-linear* speedup when running program on CPU device. Intel Core i7 has 8 cores with one host core and 7 device cores. The factor of super-linearity or sub –linearity can be evaluated by "*efficiency*" that is given by the following equation.

$$Efficiency = \frac{Speedup}{\#\_\ of\ cores} \quad (1)$$

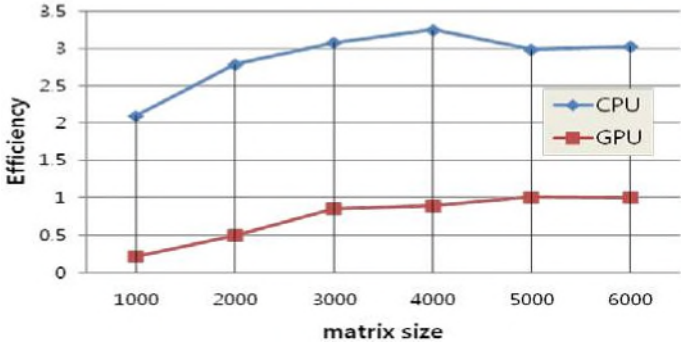

Fig. 1. Speedup efficiency between CPU and GPU.

Figure 1 shows the efficiency we achieved from parallel execution of matrix multiplication on manycore CPU and **GPU**. As shown in the figure, high efficiency of the parallel execution was observed in manycore CPU.

• ***Interpretation of manycore performance***: According to Amdahl's law, the speedup can be achieved not more than the number of cores running in parallel (which is '7' in this case). Super-linear speedup happens when the task is greater than the cache size when executed sequentially, but can fit nicely in each accessible cache when executing in parallel. So-called cache effect results from the different memory hierarchies because in parallel computing, not only the number of processors changes, but also the size of caches from different processors, with the larger accumulated cache size. More or even all of the working sets can fit into caches, when the CPU requests the memory line, data is already in the cache and the memory access time reduces dramatically, which causes the extra speedup in addition to that from the actual computation. It gets a pipeline effect that is only limited by the bandwidth of the memory bus.

Figure 2 shows speedup performance characteristics of matrix multiplications on a manycore CPU.

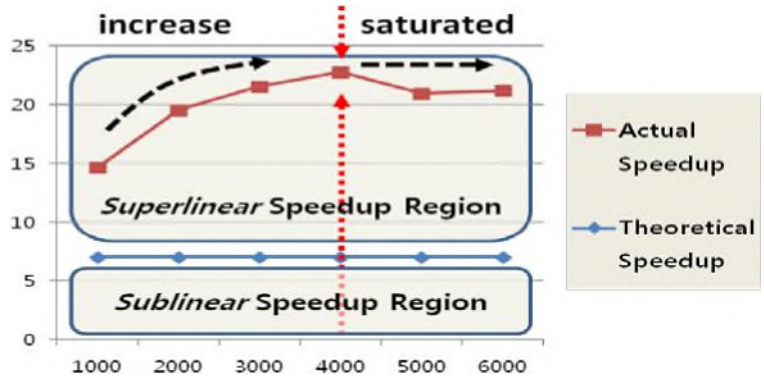

Fig. 2. Speedup performance of matrix multiplications on a manycore CPU.

• ***Decision of computing resource type***: If we can choose a best computing resource between CPU and GPU for a given workload application, then much efficient utilization can be possible by mapping an application to its best working computing platform. This sort of work are very popular in a "***hardware-software co-design***" which is a procedure for finding an optimal mapping of partitioned functionalities to CPU or FPGA/ASIC. Now, the hardware-software co-design principle can be evolved to "***CPU-GPU co-design***" in a modern heterogeneous computing era based on GPGPU. The decision can be possible if we have pre-estimated working characteristics of a given application. Such a decision will depend on parameters such as; inherent parallelism in applications, overhead for the parallel computation such as memory copy between host and GPU device, and the number of parallel working cores, etc.For example, if a matrix multiplication is supposed to be executed then we have to decide whether host or GPU device runs it. If we have pre-estimated data such as TCPU and TGPU for a given matrix size together with overheads T_{CPU_Over} and T_{GPU_Over} for CPU and GPU executions respectively, then we can make proper decision for best performance. The decision equation will looks like the following equation.

Decision Eq. : If($T_{CPU} + T_{CPU\_Over} > T_{GPU} + T_{GPU\_Over}$) then Run the code on GPU **(2)**

else Run the code on CPU

In general, T_{CPU_Over} is relatively smaller than delay parameters, so CPU-side execution time can be approximated by T_{CPU}. T_{GPU_Over} is a data-size depending parameters and non-linear to amount of data to be copied between host and device. Some of GPU initialization time will be added to $T_{GPU\_Over.}$ For a GTX 460 GPU device, memory copy bandwidth is around 1.8GB/s at best performance, so the approximated value of T_{GPU_Over} can be calculated as the following equation.

$$T_{GPU\_Over}(\text{matrix-size}) = \{[\, 3 \times 4B \times (\text{matrix-size})^2 \,] / 1.8GB\} + \text{CPU\_Initialize\_Time} \quad (3)$$

For the matrix multiplication, due to the high parallelism, always GPU outperform CPU for the case that matrix size is larger than 1000. When the matrix size reduces less than 100, then CPU is expected to outperform GPU due to overhead of GPU execution. In general, T_{CPU_Over} is relatively smaller than delay parameters, so CPU-side execution time can be approximated by T_{CPU}.

3 Conclusion

This paper has demonstrated the effectiveness of manycore CPU and GPUs in dealing with the parallelization of matrix-vector multiplication on a very basic level as well the portability of OpenCL through different platforms. Furthermore, some analyses have been carried out to understand manycore CPU and GPU performance characteristics. Such analysis approach can be further extended to include more system parameters and refined to fit the actual execution time of parallelized applications. Finally the analytical models can be used as basic partitioning criteria in a CPU-GPU co-design environment.

References

1. Ami Marowka, A Study of the Usability of Multicore Threading Tools, International Journal of Software Engineering and Its Applications Vol. 4, No. 3, July 2010.
2. Pritam Prakash Shete1, Venkat P. P. K.2, Dinesh M. Sarode3, Mohini Laghate4 and S. K. Bose5, Applying Object Oriented Design Patterns to CUDA based Pyramidal Image Blending - An Experience, International Journal of Software Engineering and Its Applications Vol. 6, No. 2, April, 2012.