July 2012

Software Performance Analysis with Parallel Programming Approaches

Suneeta H. Angadi

CSE RNSIT, Bengaluru, India, suneeta.angadi@gmail.com

Follow this and additional works at: <https://www.interscience.in/ijcsi>

Part of the [Computer Engineering Commons](#), [Information Security Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

Angadi, Suneeta H. (2012) "Software Performance Analysis with Parallel Programming Approaches," *International Journal of Computer Science and Informatics*: Vol. 2 : Iss. 1 , Article 7.

DOI: 10.47893/IJCSI.2012.1063

Available at: <https://www.interscience.in/ijcsi/vol2/iss1/7>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

Software Performance Analysis with Parallel Programming Approaches

Suneeta H. Angadi, G. T. Raju & Abhishek B

CSE RNSIT, Bengaluru, India
E-mail : suneeta.angadi@gmail.com

Abstract – The term software performance engineering (SPE) is a systematic and quantitative approach for constructing software systems to meet the performance objectives such as response time, throughput, scalability and resource utilization. Optimization is major concern in achieving performance parameters. Optimization is performed during run-time, or in the design phase. This paper proposes the coding practices in Open Multi Processing (OpenMP) and Open Computing Language (OpenCL) that outperforms the conventional algorithms for searching, matrix multiplication and routing tasks in terms of response time.

Keywords - Software Performance, Optimization, OpenMP, and OpenCL.

I. INTRODUCTION

Software performance adds value to software in terms of performance attributes for betterment. Performance is the activity of collecting the information about the execution characteristics of a program [6]. One of the parameter to measure performance is the execution time. Hence change in the design from sequential to parallel approach may result in lesser execution time and is demonstrated through coding practices in OpenMP and OpenCL on the case studies: Searching, Matrix Multiplication, and Routing tasks. Rest of the paper is organized as follows. Section 2 and 3 giving detailed description of SPE oriented towards parallel programming, section 4 explains about sequential approach, section 5 focuses on OpenMP, OpenCL practice is described in section 6, section 7 discusses comparative study of different styles of coding and case studies, paper is concluded by mentioning future research in the field.

II. PARALLEL PROGRAMMING

Many sequential programs spend considerable time blocked, e.g. waiting for memory or I/O, this time can be used by another thread in the program (rather than being given by the OS to someone else's program). Concurrency can be solution to this as concurrent execution will help in achieving better performance. Concurrency refers to managing resources that have shared usage. Concurrency ensures most efficient use of

system resources. Efficient resource utilization is the key to maximizing performance of computing systems. Multi-core programming is a way to implement concurrency through parallelism. Multi-core processors use chip multiprocessing [8][9].

This research work compares execution time with sequential, OpenMP (Multicore programming implementation) which is proven to be important method for programming shared-memory parallel computers, and OpenCL programming (focuses on host and device programming) paradigms.

III. RELATED WORK

Dami'an A. et al compare Performance of MPI, UPC and OpenMP on Multicore Architectures in [8]. M R Pimple [9] propose a programming approach for the algorithms running on shared memory multi-core systems by using blocking, coupled with parallel programming paradigm, OpenMP.

In this paper we present analysis of existing implementations through sequential, OpenMP and OpenCL for *task parallelism*.

IV. SEQUENTIAL APPROACH

In sequential approach, the execution is based on the problem considered. Most of the programs are sequential, as they execute a sequence of instructions in a pre-defined order. There is a *single thread* of

execution or control. The composition of functions is replaced by a sequence of steps and the execution time will be almost $O(n)$, $O(n^2)$, $O(n^3)$,...

V. PROGRAMMING IN openMP

OpenMP is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. Comprises of three primary API components: Compiler Directives, Runtime Library Routines, and Environment Variables. It is a widely used parallel programming model for shared-memory multiprocessor (SMP) architectures[5].

OpenMP specifies a set of compiler directives, library routines, and environment variables for determining shared memory parallelism in C/C++ programs. The parallelism in OpenMP is based on the fork-and-join execution model, where a program is initialized as a single thread named master thread. This thread is executed sequentially until the first parallel construct is encountered. This construct usually defines a parallel section, i.e. a block which can be executed by a number of threads in parallel. Having met a parallel section, the master thread creates a team of threads that executes the statements being contained in the parallel section concurrently. There is an implicit synchronization at the end of the parallel region, after which only the master thread continues its execution.

Generalized Parallelized code of case studies

```
omp_set_number_of_threads
#pragma omp parallel for
for i 1 to n
    search element
    multiply element of matrix *three for loops
parallelized
    find shortest path
end for
```

VI. PROGRAMMING IN OpenCL

OpenCL (Open Computing Language) [7] is a standard for programming heterogeneous multiprocessor platforms. OpenCL is a programming language standard which enables the programmer to express the application by structuring its computation as *kernels*. The OpenCL compiler is given the explicit freedom to parallelize the execution of *kernel instances* at all the levels of parallelism.

OpenCL standard is based on structuring computation into kernels, and specifying that there are no dependencies between kernel instances by default. The implementation is free to execute code from the different “kernel instances” sequentially, in parallel, or in an interleaved fashion, as long as the synchronization primitives present in the kernel descriptions are respected.

OpenCL programs are divided in two parts: one that executes on the **device** (on the GPU) and other that executes on the **host** (the CPU). In order to execute code on the device, programmers can write special functions (called **kernels**), which are coded with the OpenCL Programming Language. On the other hand, the host program offers an API so that device execution can be handled. The host can be programmed in C or C++ and it controls the OpenCL environment (context, command-queue.).

The amount of performance benefit an application will realize by using OpenCL depends entirely on the extent to which it can be parallelized. Code that cannot be sufficiently parallelized should run on the host, unless doing so would result in excessive transfers between host and device.

Amdahl's law specifies the maximum speed-up that can be expected by parallelizing portions of a serial program. Essentially, it states that the maximum speed-up (S) of a program is

$$S = 1 / (1 - P) + P/N$$

where P is the fraction of the total serial execution time taken by the portion of code that can be parallelized and N is the number of processors over which the parallel portion of the code runs.

For most purposes, the key point is that the greater P is, the greater the speed-up. An additional caution is implicit in this equation, which is that if P is a small number (so not substantially parallel), increasing N does little to improve performance. To get the largest lift, best practices suggest spending most effort on increasing P ; that is, by maximizing the amount of code that can be parallelized. CPU and GPU timers are the performance metrics considered.

Generalized Program code with OpenCL approach

Kernel(for searching/matrix multiplication)

Global id, local id and workgroup size

Context, Command queue

Release Kernel

VII. EXPERIMENTAL RESULTS AND DISCUSSIONS

Experiments have been conducted on *Intel compiler for OpenCL* and for OpenMP core i5 is considered over large input set for the case studies: linear search, binary search, matrix multiplication and distance vector routing. 25 runs of each of the case studies are done mean value is mentioned. More number of trials are considered for consistency.

Performance in terms of execution time is analyzed.

7.1 Matrix multiplication

Fig. 1 shows the execution time variation of Sequential, OpenMP, OpenCL for Matrix Multiplication problem. For openMP code was experimented with two and four number of threads. It is observed that OpenCL approach outperforms the Open MP and Sequential approaches. OpenMP and Sequential approach for matrix multiplication almost align with each other as evidenced in the following graph mean values are- for sequential 1.0382256, openMP 1.034909(two threads); 1.034475(four threads) and for openCL 0.05056.

Fig. 1 : Performance analysis on Matrix

Fig. 2 : Performance analysis on Binary Multiplication Search

7.2 Binary Search

Fig. 2 shows the execution time variation of Sequential, OpenMP, OpenCL for Binary Searching problem. Again number of threads are two and four. It is observed that with OpenCL approach execution time is zero i.e. kernel execution time is observed to be zero, with Open MP and Sequential approaches closer values are obtained. Mean values are- for sequential 0.0002081, openMP 0.001384(two threads); 0.000221(four threads) and for openCL zero kernel execution time.

Case study	Sequential	OpenMP 2 Threads	OpenMP 4 Threads	OpenCL	
				Kernel Setup Time	Execution Time
Matrix multiplication	1.0382256	1.034909	1.034475	1.77096	0.05056
Binary Search	0.000208	0.001384	0.000221	1.7256	0
Linear Search	1.057635	8.031968	---	1.76812	0.00824
Distance Vector	0.0616	0.046	---	---	---

Fig. 3 Performance analysis on Linear Search

Fig. 4 Performance analysis on Distance Vector Routing

7.3 Linear Search

Fig. 3 shows the execution time variation of Sequential, OpenMP, OpenCL for Linear Searching problem. Experimentation has been done for two threads. It is observed that OpenCL approach outperforms the Open MP and Sequential approaches. Sequential can be considered as second choice of implementation where as OpenMP proves to be inefficient. mean values are- for sequential 10.57635, openMP 8.031968 (two threads), and for openCL 0.00824

7.4 Distance Vector Routing

Fig. 4 shows Performance analysis of Distance Vector routing problem with Sequential and OpenMP approaches . It is observed that Open MP is better choice over Sequential style, mean values are- for sequential 0.0616, openMP 0.046

Overall Choice of approach depends on type of the problem to be solved. For matrix multiplication parallel 4 thred OpenMP and OpenCL is better. There is no task dependency hence thread and kernel instances parallel running reduces execution time. For binary search sequential and OpenCL hold good, since binary search is based on divide conquer design technique there will be dependency in specific sized problem instance. OpenCL is most efficient for linear search. Distance vector can adopt OpenMP compared to sequential.

Some sample trials of each example are run and average value is considered for comparison. Table 1 gives performance analysis information of the each of the case study with approach adopted.

Table 1. Average Execution time analysis of case studies

VIII. CONCLUSION

In this paper an attempt is made for parallelizing existing implementations. Comparative study illustrates OpenMP and OpenCL approaches prove to be better for reducing execution time. Even though there is an overhead of kernel set up time in case of OpenCL but from an end user point of view execution time must be as much less as possible. Paper analyses parallelization of respective case study's existing algorithm , optimized implementation and then parallelized approach would be better analysis. In this direction Code optimization and parallel implementation of matrix multiplication is being performed.

As a first step towards performance analysis, evaluation of the parallel languages like OpenMP, and OpenCL is considered further contributions can be made by focusing on one of the these themes in depth.

REFERENCES

- [1] Murray Woodside, Greg Franks, Dorina C. Petriu, "The Future of Software Performance Engineering", 0-7695-2829-5/07 2007 IEEE.
- [2] Mahmut Kandemir and Ismail Kadayif , "Compiler-Directed Selection of Dynamic Memory Layouts", ACM 2001 1-581 13-364-2/01/04
- [3] Jayshankar Sankarasetty, Kevin Mobley, Libby Foster, Tad Hammer, Terri Calderone "Software Performance in the Real World:Personal Lessons from the Performance Trauma Team", WOSP'07, February 5-8, 2007, Buenos Aires, Argentina.
Copyright 2007 ACM 1-59593-297-6/07/0002
- [4] P. Dziurzanski, W. Bielecki, K. Trifunovic, M. Kleszczonok, "A system for transforming an ANSI C code with OpenMP directives into a SystemC description"
- [5] James Cownie, John DelSignore, Jr., , Bronis R. de Supinski Karen Warren, "A White Paper Prepared for the OpenMP

- Architectural Review Board on DMPL: An OpenMP DLL Debugging Interface”
- [6] Mubrak S. Mohsen, Rosni Abdullah, and Yong M. Teo, “A Survey on Performance Tools for OpenMP”, World Academy of Science, Engineering and Technology 49 2009
 - [7] Pekka O. Järaskeläinen, Carlos S. de La Lamay, Pablo Huertay and Jarmo H. Takala, “OpenCL-based Design Methodology for Application-Specific Processors”
 - [8] Damián A. Mallón¹, Guillermo L. Taboada², Carlos Teijeiro², Juan Touriño², Basilio B. Fraguera², Andrés Gómez¹, Ramón Doallo², and J. Carlos Mourino¹ “Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures” M. Ropo et al. (Eds.): EuroPVM/MPI 2009, LNCS 5759, pp. 174–184, 2009. Springer-Verlag Berlin Heidelberg 2009
 - [9] M.R. Pimple, S.R. Sathe “Architecture Aware Programming on Multi-Core Systems”, International Journal of Advanced Computer Science and Applications, Vol. 2, No. 6, 2011