Parallel Matrix Multiplication on Multi-Core Processors using SPC\(^3\) PM

Muhammad Ali Ismail, Talat Altaf, S.H. Mirza

Abstract—Matrix Multiplication is used as building block in many of important applications covering nearly all subject areas. With the advent of multi-cores every processor has built-in parallel computational power and that can be utilized to speed-up the program execution and reducing the execution time. In this paper we have discussed a simple, highly efficient and scalable implementation of a common matrix multiplication using a newly developed parallel programming model SPC\(^3\) PM for general purpose multi-core processors. We have tested our approach for up to 24 cores with different matrices size varying from 100 x 100 to 10000 x 10000 elements. And for all these tests our proposed approach has shown much improved performance and scalability.

Keywords—Multi-core Processing, High Performance Computing, Parallel Matrix Multiplication.

I. INTRODUCTION

The SPC\(^3\) PM, (Serial, Parallel and Concurrent Core to Core Programming Model), is a new parallel programming model developed as a part of PhD research [1], [2]. The development of SPC3 PM is motivated with an understanding that existing parallel developments tools do not provide adequate support for general purpose multi-core programming and unable to utilize all available cores efficiently as they are designed for either specific parallel architecture or certain program structure. The SPC\(^3\) PM is developed to equip a common programmer with multi-core programming tool for scientific and general purpose computing. The SPC\(^3\) PM provides a set of rules for algorithm decomposition and a library of primitives that exploit parallelism and concurrency on multi-core processors. The programming model is serial-like task-oriented and provides thread level parallelism without the programmer requiring a detailed knowledge of platform details and threading mechanisms. It has also many other unique features that distinguish it with all other existing parallel programming models. It supports both data and functional parallel programming. Additionally, it supports nested parallelism, so one can easily build larger parallel components from smaller parallel components. A program written with SPC\(^3\) PM may be executed in serial, parallel and concurrent fashion on available cores. besides, it also provides processor core interaction feature that enables the programmer to assign any task or a number of tasks to any of the cores or set of cores. Besides, the ability to use SPC\(^3\)PM on virtually any processor or any operating system with any C++ compiler makes it very flexible

II. SPC\(^3\) PM, (SERIAL, PARALLEL, CONCURRENT CORE TO CORE PROGRAMMING MODEL)

SPC\(^3\) PM, (Serial, Parallel, Concurrent Core to Core Programming Model), is a serial-like task-oriented multi-threaded parallel programming model for multi-core processors, that enables developers to easily write a new parallel code or convert an existing code written for a single processor. The programmer can scale it for use with specified number of cores. And ensure efficient task load balancing among the cores [1], [2].

SPC\(^3\) PM is motivated with an understanding that existing general-purpose languages do not provide adequate support for parallel programming [3], [4]. Existing parallel languages are largely targeted to scientific applications. They do not provide adequate support for general purpose multi-core programming whereas SPC\(^3\) PM is developed to equip a common programmer with multi-core programming tool for scientific and general purpose computing. It provides a set of rules for algorithm decomposition and a library of primitives that exploit parallelism and concurrency on multi-core processors. SPC\(^3\) PM helps to create applications that reap the benefits of processors having multiple cores as they become available.

SPC\(^3\) PM provides thread parallelism without the programmers requiring having a detailed knowledge of platform details and threading mechanisms for performance and scalability. It helps programmer to control multi-core processor performance without being a threading expert. To use the library a programmer specifies tasks instead of threads and lets the library map those tasks onto threads and threads onto cores in an efficient manner. As a result, the programmer is able to specify parallelism and concurrency far more conveniently and with better results than using raw threads. The ability to use SPC\(^3\) PM on virtually any processor or any
operating system with any C++ compiler also makes it very flexible.

SPC³ PM has many unique features that distinguish it with all other existing parallel programming models. It supports both data and functional parallel programming. Additionally, it supports nested parallelism, so one can easily build larger parallel components from smaller parallel components. A program written with SPC³ PM may be executed in serial, parallel and concurrent fashion. Besides, it also provides processor core interaction to the programmer. Using this feature a programmer may assign any task or a number of tasks to any of the cores or set of cores.

A. Key Features

The key features of SPC³ are summarized below.

- SPC³ is a new shared programming model developed for multi-core processors.
- SPC³ PM works in two steps: defines the tasks in an application algorithm and then arranges these tasks on cores for execution in a specified fashion.
- It provides Task based Thread-level parallel processing.
- It helps to exploit all the three programming execution approaches, namely, Serial, Parallel and Concurrent.
- It provides a direct access to a core or cores for maximum utilization of processor.
- It supports major decomposition techniques like Data, Functional and Recursive.
- It is easy to program as it follows C/C++ structure.
- It can be used with other shared memory programming model like OpenMP, TBB etc.
- It is scalable and portable.
- Object oriented approach

B. Programming with SPC³ PM

SPC³ PM provides a higher-level, shared memory, task-based thread parallelism without knowing the platform details and threading mechanisms. This library can be used in simple C / C++ program having tasks defined as per SPC³ PM Task Decomposition rules. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner. The result is that SPC³ PM enables you to specify parallelism and concurrency far more conveniently, and with better results, than using raw threads.

Programming with SPC³ is based on two steps. First describing the tasks as it specified rules and then programming it using SPC³ PM Library.

1) Steps involved in the development of an application using SPC³ PM

- The user determines that his application can be programmed to take advantage of multi-core processors.
- The problem is decomposed by the user following the SPC³ PM 'Task Decomposition Rules'.
- Each Task is coded in C /C++ as an independent unit to be executed independently and simultaneously by each core.
- Coding of Main Program using SPC³ PM Library to allow the user to run the program in serial, parallel or concurrent mode.
- Compilation of code using any standard C/C++ compiler.
- Execution of Program on a multi-core processor

2) Rules for Task Decomposition

The user can decompose the application / problem on the basis of following rules.

- The user should be able to breakdown the problem in various parts to determine if they can exploit Functional, Data or Recursive decomposition.
- Identify the loops for the loop parallelism and may be defined as Tasks.
- Identify independent operations that can be executed in parallel and may be coded as independent Tasks.
- Identify the large data sets on which single set of computations have to be performed. Target these large data sets as Tasks.
- Tasks should be named as Task1, Task2,… TaskN. If a Task returns a value it should be named with suffix ‘R’ like TaskR1, TaskR2…. TaskRN.
- There is no limit on the number of Tasks.
- Each Task should be coded using either C/C++/VC++/C# as an independent function.
- A Task may or may not return the value. A Task should only intake and return structure pointer as a parameter. Initialize all the shared or private parameters in the structure specific to a Task. This structure may be shared or private.
- Arrange the tasks using SPC³ PM Library in the main program according to the program.

3) Program Structure

Programming structure for SPC³ PM is shown in figure 1.

C. SPC³ PM Library

SPC³ PM provides a set of specified rules to decompose the program into tasks and a library to introduce parallelism in the program written using C/ C++. The library provides three basic functions.

- Serial
- Parallel
- Concurrent

Serial function is used to specify a Task that should be executed serially. When a Task is executed with in this function, a thread is created to execute the associated task in sequence. Parallel function is used to specify a Task that should be executed in parallel. When a Task is executed with in this function, a team of threads is created to execute the associated task in parallel and has an option to distribute the work of the Task among the threads in a team. These threads are scheduled on the available cores either by operating
system or as specified by the programmer. Concurrent function is used to specify the number of independent tasks that should be executed in concurrent fashion on available cores. These may be same tasks with different data set or different tasks. When the Tasks are executed defined in this function, a set of threads equal or greater to the number of tasks defined in concurrent function is created such that each task is associated with a thread or threads.

**III. MATRIX ALGORITHM**

We have selected a standard and basic matrix multiplication algorithm [5]-[7] in which the product of a \((m \times p)\) matrix \(A\) with a \((p \times n)\) matrix \(B\) is a \((m \times n)\) matrix denoted \(C\) such that

\[
C_{i,j} = \sum_{k=1}^{p} A_{i,k} B_{k,j}
\]

Where \(1 \leq i \leq m\) is the row index and \(1 \leq j \leq n\) is the column index. In order to compare the performance of SPC3 PM, this algorithm is implemented using two different approaches. The first is the standard serial C/ C++ code. The other is in C++ using the concurrent function of SPC3 PM. Pseudo code for the program using SPC3PM is shown in table I.

In serial C/C++ implementation the basic computations of addition and multiplication are placed within the three nested ‘for’ loops. For SPC3 PM using concurrent function, a Task is defined having the basic algorithm implementation. The idea is to execute this task concurrently on different cores with different data set. Every Task has its own private data variables defined in a structure ‘My_Data’. All the private structures are associated with their tasks and initialized accordingly. Using the Concurrent function of SPC3 PM, the required number of concurrent tasks are initialized and executed.

**IV. PERFORMANCE ANALYSIS**

The following tables II shows the execution time in seconds for each of using serial C/C++ and SPC3 PM with different sizes of matrices for 4, 8, 12 and 24 parallel / concurrent threads.
The speedup of the parallel function of SPC³ PM is compared with standard C++ algorithm for different number of parallel threads and matrix sizes and presented in table III. The speedup obtained is linear and scalable and depends on the number of parallel threads not on the problem size. For any given number of parallel threads, the speedup remains linear as the problem size increases. This comparison is also presented in figure 2.

### TABLE II
EXECUTION TIME IN SECONDS FOR EACH OF USING SERIAL C/C++ AND SPC³ PM WITH DIFFERENT SIZES OF MATRICES FOR 4, 8, 12 AND 24 PARALLEL / CONCURRENT THREADS.

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Number of Parallel Threads</th>
<th>Execution Time (Sec)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Serial Time</td>
<td>SPC³ PM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 X 100</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 X 1000</td>
<td></td>
<td>13</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000 X 2000</td>
<td></td>
<td>154</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000 X 3000</td>
<td></td>
<td>551</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000 X 4000</td>
<td></td>
<td>1374</td>
<td>202</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000 X 5000</td>
<td></td>
<td>2707</td>
<td>396</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6000 X 6000</td>
<td></td>
<td>4642</td>
<td>682</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7000 X 7000</td>
<td></td>
<td>7285</td>
<td>1086</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000 X 8000</td>
<td></td>
<td>10925</td>
<td>1619</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9000 X 9000</td>
<td></td>
<td>15606</td>
<td>2303</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000 X 10000</td>
<td></td>
<td>21497</td>
<td>3161</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TABLE III
SPEEDUP FOR MATRIX MULTIPLICATION USING SPC³ PARALLEL FUNCTION WITH DIFFERENT NUMBER OF PARALLEL THREADS AND DIFFERENT MATRIX SIZES.

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>N=4</th>
<th>N=8</th>
<th>N=12</th>
<th>N=24</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 X 100</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>1000 X 1000</td>
<td>4.30</td>
<td>6.50</td>
<td>6.50</td>
<td>13.00</td>
</tr>
<tr>
<td>2000 X 2000</td>
<td>4.28</td>
<td>6.16</td>
<td>8.56</td>
<td>15.40</td>
</tr>
<tr>
<td>3000 X 3000</td>
<td>3.40</td>
<td>6.64</td>
<td>8.48</td>
<td>15.31</td>
</tr>
<tr>
<td>4000 X 4000</td>
<td>3.40</td>
<td>6.48</td>
<td>8.38</td>
<td>15.98</td>
</tr>
<tr>
<td>5000 X 5000</td>
<td>3.67</td>
<td>6.25</td>
<td>8.20</td>
<td>15.83</td>
</tr>
<tr>
<td>6000 X 6000</td>
<td>3.73</td>
<td>6.60</td>
<td>8.10</td>
<td>15.32</td>
</tr>
<tr>
<td>7000 X 7000</td>
<td>3.51</td>
<td>6.63</td>
<td>8.65</td>
<td>15.30</td>
</tr>
<tr>
<td>8000 X 8000</td>
<td>3.53</td>
<td>6.27</td>
<td>8.46</td>
<td>15.39</td>
</tr>
<tr>
<td>9000 X 9000</td>
<td>3.42</td>
<td>6.23</td>
<td>8.67</td>
<td>15.44</td>
</tr>
<tr>
<td>10000 X 10000</td>
<td>3.96</td>
<td>6.56</td>
<td>8.07</td>
<td>15.02</td>
</tr>
</tbody>
</table>
V. CONCLUSION
The results from this study show that the SPC³ PM provides a simpler, scalable and effective way to perform matrix multiplication on multi-core processors. With SPC³ PM, the programmer can execute a simple and standard matrix multiplication algorithm in parallel on multi-core processors in much less execution time than that of serial C/C++ code. By using the proposed approach, the programmer can develop efficient parallel matrix multiplication applications easily that can make most of multi-core processors.

REFERENCES

Muhammad Ali Ismail did his M.Engg in Computer Engineering with specialization in Computer Systems and Design in 2007 and in 2011 he got his PhD in Computer Engineering in the field of Multi-Core Computing. His areas of interest include Serial and Parallel Processing computer architectures, distributed computing, Cluster and Grid Computing, Memory Management and related Algorithms, Parallel Programming languages, models and algorithms. He is a member of IEEE (USA) and IEE (UK).

Fig. 2 Speedup comparison of matrix multiplication using SPC3 Parallel with different number of parallel threads and different matrix sizes.