International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 5 (2013), pp. 471-478

http://www. irphouse.com /ijict.htm

Analysis of Parallelization Techniques and Tools

S. Prema[1] and R. Jehadeesan[2]

[1,2]Computer Division, Indira Gandhi Centre for Atomic Research, DAE, Kalpakkam, INDIA.

Abstract

Parallel Computing solves computationally large problems by partitioning into multiple tasks and running simultaneously on multicore or multiprocessor environment based on shared or distributed memory architectures. New multicore era demands software programmer to develop parallel programs to completely utilize the hardware parallelism. Writing parallel program manually for complex problem is a tedious and time consuming process. Hence, automatic parallelization tools were evolved to automate the process of converting sequential code to parallel. Parallelization techniques like Dependency Analysis and Loop Parallelization plays a major role in parallelization process. This paper aims at providing brief survey of existing OpenMP based parallelization tools and also the anlaysis on the performance of two such tools on typical problems.

The OpenMP directives based tools Cetus and Par4all are taken for our case studies. The performance of the automated parallelization on sample program were studied and analyzed in terms of execution time and speedup. The study revealed the conditions under which they could effectively parallelize the code and the conditions they could not handle as effectively as manual parallelization. Also many of the available parallelization tools are meant for shared memory hardware architecture and relatively a very few automated tools based on MPI are available for distributed memory architectures. The paper underscores the need for efficient parallelization tools supporting different parallel processing environments, with the ability to identify and exploit the parallelism by inserting parallel directives or APIs in serial programs.

Keywords: Automatic Parallelization; Shared Memory; Distributed Memory; Automatic Parallelization; Dependency Analysis; Loop Parallelization; OpenMP; MPI;

1. Introduction

A drastic shift towards parallel computing from serial computing is mainly due to Von Neumann bottleneck [Laird, 2009], where latency is the most important problem. Parallel computing [Pacheco, 2011] solves large problem concurrently using multiple cores or CPUs, thereby saving execution time. Moore's law has created a major impact in software and hardware industry. Inorder to take the full advantage of the hardware, the software has to be written in a multi-threaded or multi-process manner. Hence parallelism becomes the future of computing. Writing parallel programs manually has become tedious and time consuming process. Hence to achieve high performance and functionally correct parallel programming, automatic parallelization is important [Bliss, 2007]. Automatic parallelization [Midkiff, 2012] is the conversion of sequential code to parallel code inorder to utilize the multiprocessor architecture simultaneously. This converted parallel code can be used to run on multicore machines. The need for automatic parallelization has two main reasons: (i) To achieve effective speedup [Felician, 2004]. (ii) To relieve programmer from complex and time consuming manual parallel programming [Qian, 2012].

2. Parallelization Techniques

Process of parallelization involves: (i) partitioning complex program to tasks and mapping it to different processors (ii) maintaining communication and synchronization between processors (iii) load balancing [Bliss, 2007]. Dependency analysis and Loop parallelization are the most important parallelization techniques.

2.1 Dependency Analysis

Dependency analysis plays a major role in parallelization process [Qian, 2012]. Two statements can be executed in parallel only when there is no dependency between instructions. Hence these dependencies should be removed to make the code parallelizable. Identifying dependency relation between statements involves complex analysis. Long term goal for many researchers centres on dependency analysis.

2.2 Loop Parallelization

Loop parallelization is significant since 90% of the execution time is mostly due to loops in the code [Felician, 2004]. Distribution of loop elements into chunks and assigning to different processors will reduce the latency. In loop parallelization, parallelizing nested loop efficiently is a challenging task.

3. Parallelization Tools

Automatic parallelization tools are designed to convert manually written serial code to parallel code by inserting parallel APIs or directives like OpenMP, OpenCL, MPI, CUDA, etc. [Qian, 2012]. There are many parallelization tools available. Some parallelization tools like ICU-PFC, Cetus, Par4all and Polaris are based on OpenMP which are freely available. Olympix and MPI/Work Pool model tools are based on MPI. In this paper, two parallelization tools were considered, Cetus and Par4all for case studies. Performance analyses were done on Cetus and Par4all in terms of computing time and speedup.

3.1 Cetus

Cetus is an automatic parallelization tool that does source to source transformation [Dave, 2009]. It converts sequential 'C' code to OpenMP inserted parallel 'C' code. OpenMP [Chapman, 2008] is a parallel library that can run on shared memory multiprocessor. Cetus is user-friendly. This tool supports all linux environments. The tool implements privatization, reduction variable recognition and induction variable substitution.

3.2 Par4all Tool

Par4all is an open source automatic parallelizer which does parallelization and optimization for 'C' program. When manually opted, the parallelized codes create new OpenMP, OpenCL or CUDA source codes. The converted parallel codes are then ported to multicore systems for execution [Amini, 2012]. Par4all can be installed only on debian or ubuntu platform. Par4all does array privatization, reduction variable recognition and induction variable substitution.

4. Performance Analysis on Cetus and Par4all

4.1. Case Study

A survey was made on the usage of OpenMP clauses like private, shared, atomicity, reduction, first private, last private and collapse clauses in parallel programs. Analysis of Cetus and Par4all tools were done using sample programs. Cetus and Par4all does parallelization only for 'for' loops whereas it does not parallelize 'while' loops and 'do-while' loops. Cetus and Par4all failed to insert parallel directives for some pre-defined functions like drand48(), srand(), etc. Cetus and Par4all parallelize user-defined functions. Cetus does inlining of function thereby reducing the call and return time.

4.2. Experiment

The Cetus tool was installed in Red-Hat Linux 2.6 and Par4all tool was installed in Ubuntu-12.04. Sample programs were defined to analyse the tools: (i) square root of complex numbers (ii) matrix multiplication. In supercomputing parallel cluster, the program was made to run on one node that has two quad-core processors. The total

memory freely available in the node is 15.67 GB. The coding was made to run in 'icc' compiler. Timing and speedup analysis were done for sequential coding and for converted parallel coding (mainly loops). The converted parallel coding was executed by varying number of threads as 2, 4 and 8 and its results were noted.

4.2.1 Square Root of Complex Numbers in a one dimensional array. Calculation of real and imaginary part of complex number (a + bi) is done for a large array size. The loops used are single loop in the program. Cetus and Par4all tools showed effective parallelization results which are depicted in Fig.1. Hence, manual parallelization was not required. The speedup for Cetus converted parallel code showed maximum speed up as 1.35, 2.29 and 3.15 for 2 threads, 4 threads and 8 threads respectively and Par4all converted parallel code showed maximum speedup as 1.37, 2.34 and 3.17 for 2 threads, 4 threads and 8 threads respectively which are depicted in Fig.2. It is observed that, parallelization made on single loop program by Cetus and Par4all does not yield effective speedup i.e. ideal speedup, which may be due to stall cycles in the pipeline execution that are caused because of data dependency between two applications running on available threads.

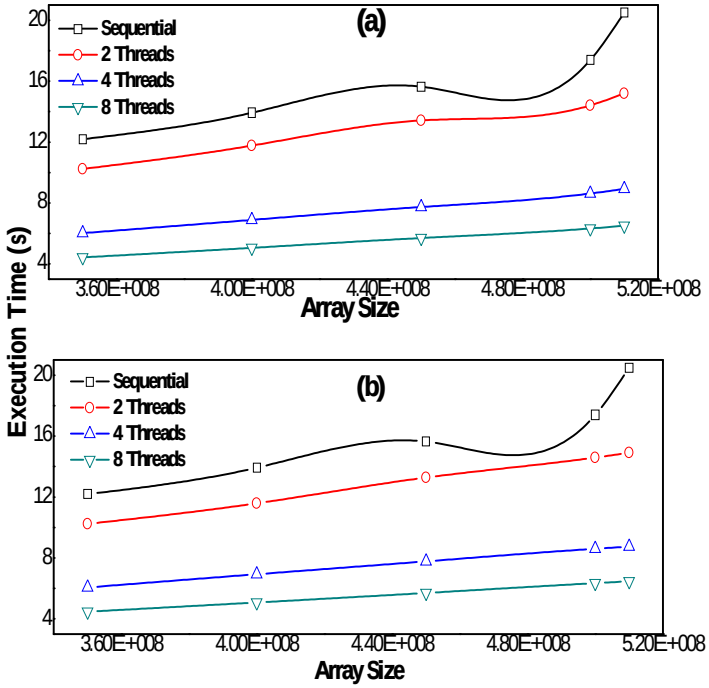

Figure 1: Execution time of square root of complex numbers obtained by (a) Cetus (b) Par4all.

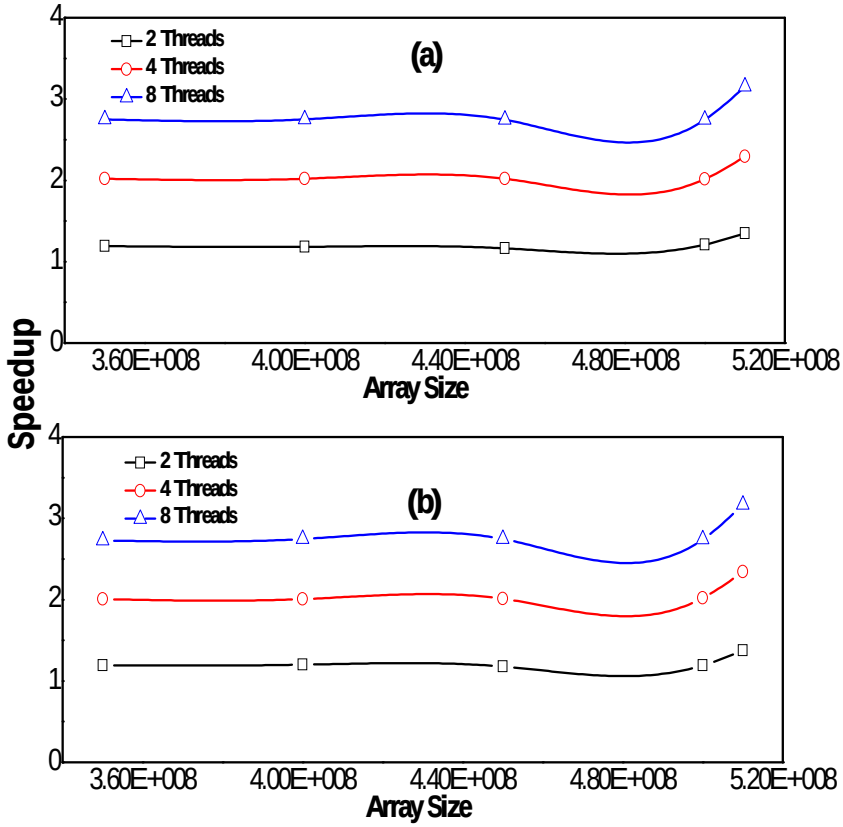

Figure 2: Speedup of square root of complex numbers obtained by (a) Cetus (b) Par4all.

4.2.2 Matrix Multiplication. Calculation of matrix multiplication, c[n][n] += (a[n][n] * b[n][n]) was done for a large matrix size. Nested loops were used in the program. It was observed that Cetus and Par4all parallelizes outer loop but failed to parallelize inner loop. Hence manual parallelization was done. Manually converted parallel code showed better timing results and it showed good parallelization results, which are shown in Fig.3. Cetus converted parallel code with 'nested loop disabled' showed maximum speedup as 1.96, 3.51 and 6.54 for 2, 4 and 8 threads respectively. When 'nested loop enabled' it showed maximum speedup as 3.53, 5.26 and 5.01 for 2, 4 and 8 threads respectively; this is due to overhead in number of threads. Par4all converted parallel code showed maximum speedup as 5.61, 11.33 and 10.09 for 2, 4 and 8 threads respectively; this is due to partial parallelization of nested loops. Manually converted parallel code showed better speedup as 6.39, 7.73 and 13.01 for 2, 4 and 8 threads respectively, which are depicted in Fig.4.

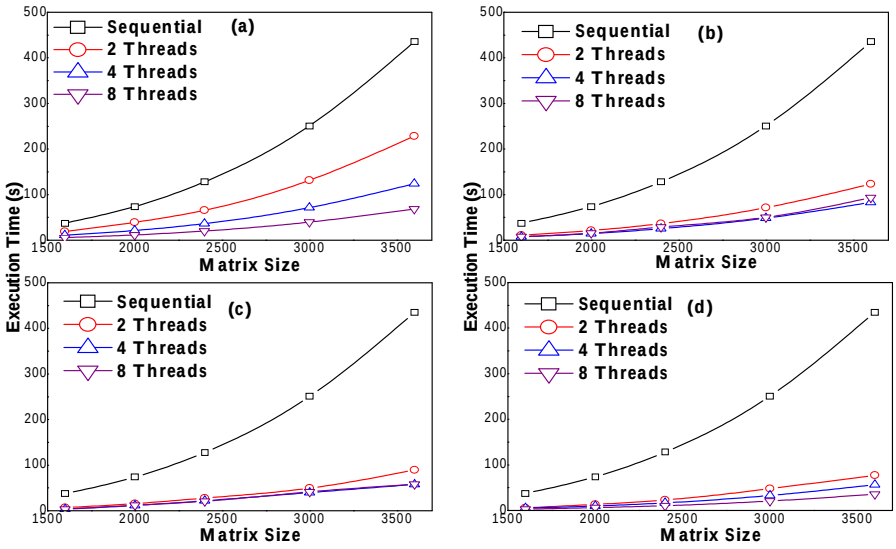

Figure 3: Execution time of matrix multiplication obtained by (a) Cetus with 'nested loop disabled' (b) Cetus with 'nested loop enabled' (c) Par4all (d) Manual.

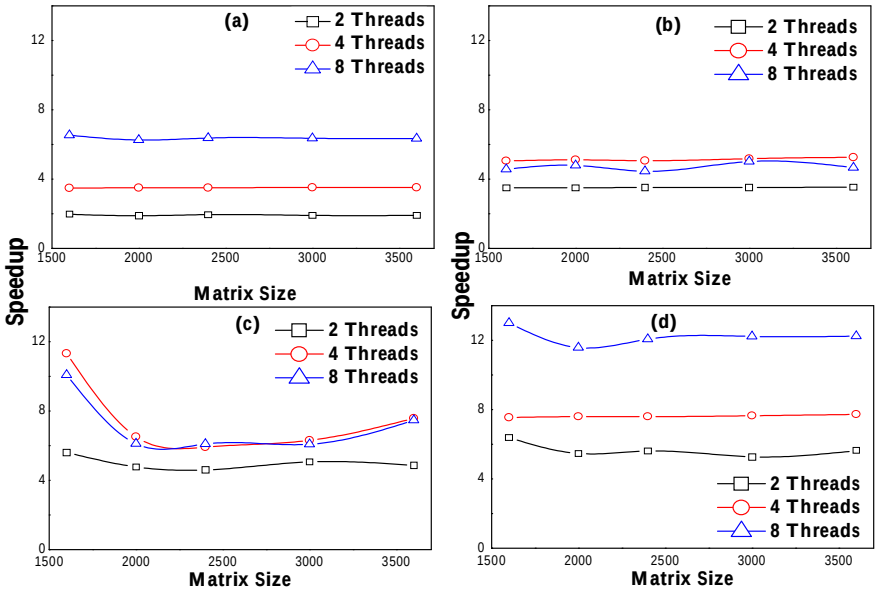

Figure 4: Speedup of matrix multiplication obtained by (a) Cetus with nested loop disabled (b) Cetus with nested loop enabled (c) Par4all (d) Manual.

5. Conclusions

Performance analysis on Cetus and Par4all was done using sample programs and it was concluded that the tools does parallelization in an effective way for single loops. Cetus and Par4all does not show effective results for nested loops. Manual parallelization was done for matrix multiplication problem and the performance degradation was rectified and achieved maximum speedup than tool converted code. Parallelization tools that efficiently insert parallel directives or APIs should be developed. Cetus and Par4all tools are based on OpenMP directives for shared memory processors. There are very few MPI based tools. Hence there is a need to develop OpenMP and MPI based tools for shared memory and distributed memory processors.

References

[1] A Felician (2004), How to Parallelize a Sequential Program, Preceedings of the 5th Biennial International Symposium. Brasov, Romania, pp. 424 – 428

[2] A Laird (2009), The Von Neumann Architecture Topic Paper #3, Survey of Programming Languages.

[3] B Chapman et al. (2008), Using OpenMP, The MIT Press, London, England.

[4] C Dave et al. (2009), Cetus: A Source-To-Source Compiler Infrastructure for Multicores, IEEE Computer, **42**, *12*, pp. 36-42

[5] M Amini et al. (2012), Par4all: From Convex Array Regions to Heterogeneous Computing, 2nd International Workshop on Polyhedral Compilation Techniques (IMPACT), Paris, France

[6] N Bliss (2007), Addressing the Multicore Trend with Automatic Parallelization, Lincoln Laboratory Journal, **17**, *1*, pp. 187-198

[7] P S Pacheco (2011), An Introduction to Parallel Programming, University of San Francisco, USA.

[8] S P Midkiff (2012), Automatic Parallelization: An Overview of Fundamental Compiler Techniques, Purdue University, West Lafayette

[9] Y Qian (2012), Automatic Parallelization Tools, Proceedings of the World Congress on Engineering and Computer Science, San Francisco, USA, **1**, pp. 97-101.