Mathematical Modelling and Analysis
Volume 12 Number 4, 2007, pages 459–468
© 2007 Technika ISSN 1392-6292 print, ISSN 1648-3510 online

EVALUATION OF THE HYPER-THREADING TECHNOLOGY FOR HEAT CONDUCTION-TYPE PROBLEMS

S. IVANIKOVAS and G. DZEMYDA

Institute of Mathematics and Informatics
Akademijos St. 4, LT 08663, Vilnius, Lithuania
E-mail: `Ivanikovas@gmail.com; Dzemyda@ktl.mii.lt`

Received February 21, 2007; revised July 20, 2007; published online December 15, 2007

Abstract. Symmetric Multi Processor (SMP) systems become popular and available for most users nowadays. The main factor that affects the performance while working with a large amount of data with the SMP system is operating memory speed. Hyper-Threading technology lets to increase the amount of threads working in parallel and to speed-up the execution of program using the same hardware. A standard heat conduction problem was chosen as a benchmark to test the SMP system performance while working with memory-intensive tasks. The results of practical tests and the quantitative estimations of efficiency are presented in the article. The experiments indicate how Hyper-Threading technology helps to use the SMP systems and dual-core processor systems more effectively while working with a large amount of data.

Key words: parallel algorithms, hyper-threading, symmetric multi processors, heat conduction

1. Introduction

The usage of parallel processes lets us to increase the computing efficiency greatly even when an ordinary technology and hardware is used [1, 9, 11]. Parallel programming is widely used while working with super computers and clusters. Nowadays SMP (Symmetric Multi Processor) systems and computers with HT (Hyper-Threading) technology have become popular and available for most users [6]. Since such systems had appeared, the usage of parallel processes became actual for all computers. HT technology lets an operating system to treat one physical processor as two logical processors [2, 6]. In this paper, the advantages of HT technology while working with memory-intensive tasks are presented. It is shown, that parallel computing can be effective even with a single processor system using HT technology. While working with SMP systems, the HT technology lets us to execute more threads in parallel. Using more threads we can optimize the work of operating memory and increase the efficiency of system work.

The paper is organized as follows: the next section presents the review of HT technology and SMP systems and indicates the problem of slow operating memory and system bus work. The benefits of using HT technology for solving this problem are also presented in this section. Sections 3 and 4 illustrate the problem of working with large amount of data with SMP system and describe the realization of the discrete heat conduction algorithm, which was chosen for the examination of the problem. In section 5, the results of computational tests are presented and analysed. Finally, the last section summarizes the results of the tests and presents the conclusion of the work.

2. Hyper-Threading Technology and SMP Systems

Parallel computing is the processing of one task in different places at the same time. The complexity of nowadays applications requires to use parallel computing and create parallel algorithms to solve real-world tasks [9, 11].

A wide variety of tools helps to create a parallel program by modifying the serial one. While creating parallel algorithm, the whole task is divided into smaller parts and the parallel execution of these parts is organized. We seek for the uniform distribution of the work among processors, minimizing data transfer and fitting program for the computer type and architecture, by creating parallel algorithms [1].

Hyper-Threading technology was firstly used in 2002 in Intel Xeon processors. The essence of HT technology is that the physical processor execution resources are shared and the architecture state is duplicated for two logical processors [7]. Nowadays, this technology is widely used in Intel Pentium 4 processors and is available for most users. HT technology enables so called thread-level parallelism (TLP) and allows multi-threaded software applications to execute threads in parallel within one single processor. Benchmark tests proved that some applications can experience a 30 percent gain in performance using this technology. Thus HT technology lets to use processor more effectively [6].

Symmetric Multi Processing, or SMP, is a multiprocessor computer architecture where two or more identical processors are connected to a single shared main memory. Most common multiprocessor systems today use an SMP architecture. Dual processor systems and systems with dual-core processors are available for most users nowadays. The Intel Pentium processor Extreme Edition combines the HT technology with dual-core processing to get PCs capable of handling four software threads. Newly appeared quad-core processors are another step for making multi-core processors common for the ordinary users.

SMP systems allow any processor to work on any task no matter where the data for that task is located in memory. With proper operating system support, SMP systems can easily move tasks between processors to balance the workload efficiently.

In the SMP system, both processors have their own caches and registers. HT system has only one cache and processors registers are divided between two logical processors.

Operating memory is much slower than processors can access it and even single-processor machines tend to spend a considerable amount of time waiting for the data to arrive from memory. SMP makes this worse, as only one processor can access memory at a time. It is possible for several processors to be starving while one of them is working with memory. There is no such problem working with NUMA (Non-Uniform Memory Architecture). But this architecture is more complex and not so widely spread as UMA (Uniform Memory Architecture).

So, user can work with four logical processors while using the SMP system with double Intel Xeon processor with HT technology or the PC based on Intel Pentium Extreme Edition processor. Using thread-level parallelism it is possible to get an improvement by a factor of (nearly) the number of additional processors in some applications. But as it was said the performance of the system is limited by system bus and RAM speed while working with SMP system with large amount of data.

3. Work with Large Amount of Data Using SMP System

The main aim of the work was to investigate how the processing speed increases using more threads while working with large data arrays. As it was presented in the work [2], the usage of several threads with ordinary computations (floating point and integer calculations) gives a very small gain in performance on HT technology. But combining several different works (computing and working with RAM) gives much better results.

In this work the performance of multi-threaded programs was tested using memory intensive tasks. A standard heat conduction problem was chosen to test the SMP system performance. This problem and the algorithm which was used during the tests are presented in Chapter 4. The OpenMP standard was chosen to create the multi-threaded program. This is one of the most popular ways to program applications for SMP systems. It lets to create the multi-threaded programs in a simple and effective way [8, 10].

Programs were tested using Pentium 4 HT based system (Pentium 4 HT 3.2GHz, 512 KB L2 cache, 512 MB RAM) and Dual Xeon Server (Dual Xeon 3.2 GHz, 1 MB L2 cache, 2GB RAM). As Intel Xeon processors also have the HT technology, there was an ability to work with four threads on this system.

Several criteria were used to measure program speed-up and efficiency of computer usage. The speed-up was estimated by $S_p = \dfrac{T_1 - T_p}{T_1}$ and the efficiency of algorithm is estimated by $E_p = \dfrac{T_1}{pT_p}$. Here p is the number of threads used, T_1 is the sequential program working time and T_p is the multi-threaded program working time using p threads [1, 9].

4. Heat Conduction-Type Problems

Heat conduction type problems are widely used. The particularity of this type of tasks is a large amount of data to be processed. Tasks like weather forecast, heat spread or heat conduction, can be qualified as the tasks of this type. In this work, numerically solved a heat conduction problem, while edges of flat rectangular plate are heated and the temperature of the plate is calculated. A heat flows through thermally conductive materials by a process generally known as "gradient transport" [4].

To solve the chosen heat conduction problem the method of finite differences was used. First we define a discreet grid on the plate

$$w_h = \{(x_i, y_j): \ \ x_i = ih, \ y_j = jh, \quad 0 \le i, j \le N\},$$

where N is the number of rows and columns in the matrix w_h, $h = 1/N$ is a space step of the grid. The solution $U_{ij} = U(x_i, y_j)$ is calculated only in nodes of the grid. The dimension of the grid w_h is $(N+1)^2$.

In each node of the grid, continuous fluxes of the heat are approximated by finite differences. To calculate the value of the solution at the given node, four neighbouring nodes are used. So we have a system of linear equations

$$U_{ij} = \frac{U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}}{4}, \quad 1 \le i, j \le N-1.$$

The values of the solution on side nodes of the grid are found from the Dirichlet boundary conditions. The remaining system of $(N-1)^2$ linear equations is solved using Jacobi iteration method [1]. First, initial approximation U^0_{ij} is chosen. Then the iteration process

$$U^{s+1}_{ij} = \frac{U^s_{i-1,j} + U^s_{i+1,j} + U^s_{i,j-1} + U^s_{i,j+1}}{4}, \quad 1 \le i, j \le N-1.$$

is repeated until two neighbouring approximations are sufficiently close. Here s denotes the iteration number. It is well known that the total number of iterations is proportional to the amount of nodes $\mathcal{O}(N^2)$ [1, 3]. The computations performed in this task are rather simple, but we have to process a large amount of data, so this task is memory-intensive.

The parallel algorithm is based on data parallelism. The same computations are performed with all the data. So, we can get a parallel algorithm by dividing the nodes of the grid among processors (see Fig.1). Each of the processors gets a part of the matrix and performs the iterative calculations with its local part of the data. After each iteration, neighbouring processors synchronize the values of side nodes to continue the iterative process. The description of the parallel algorithm is presented in Table 1.

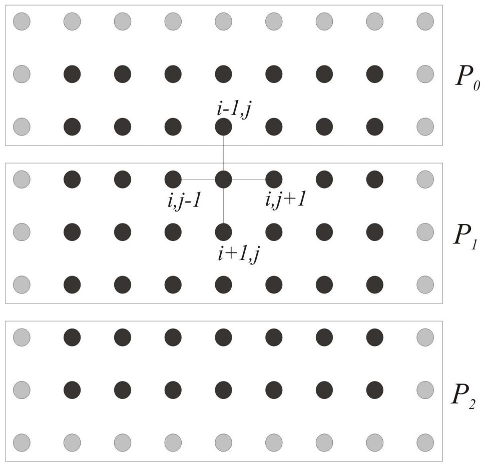


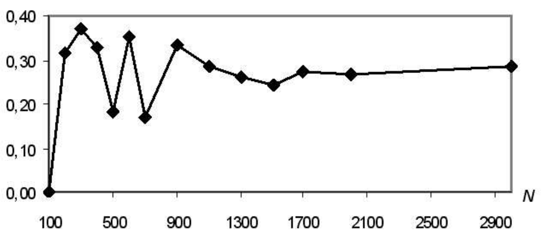
Figure 1. The distribution of the grid nodes among processors.

Table 1. The structure of the parallel algorithm.

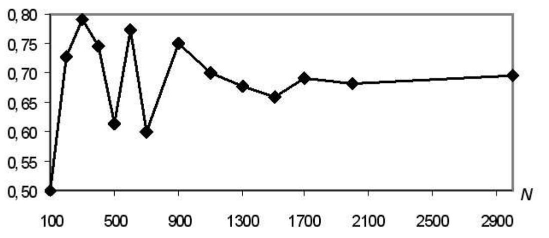
```
1.  Set the number of threads, matrix dimension ($N$),
    precision and side conditions.
2.  Initialize OpenMP:
       #pragma omp parallel
3.  Each thread initializes its own part $U_{ij}$ of the whole matrix $w_h$ .
4.  error = 2*precision; e = $N$ / threads + 2.
5.  Each thread processes its part of the matrix:
       while(maxerror > precision)
           for(i=1;i<(e-1);i++)
             for(j=1;j<(N+1);j++)
                $U_{ij} = (U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}) /4;$
6.  Each thread evaluates the error by finding the biggest
    difference between the values of $U_{ij}$ in previous and present iteration.
7.  Maximum error value for all threads is found.
    This code is executed by only one thread at a time:
       maxerror=0.0;
       for(i=0;i<threads;i++)
          if (my_ID = = i) if (maxerror<error)
             { #pragma omp critical
             maxerror=error; }
8.  Threads synchronize the values of side nodes of $U_{ij}$
    to continue the iterative process.
9.  If precision is not reached, iterative steps from 4 to 7 are repeated.
10. Parts of matrix $w_h$ processed in different threads are collected to one array.
```

5. Practical Results

Tests with serial and two threads parallel program were performed using Pentium 4 HT based system. These tests showed that the usage of two threads lets us to increase the system performance up to 37% while working with a smaller amount of data and about 30% with a large amount of data using single processor system with Hyper-Threading technology (see Fig.2).



a)



b)

Figure 2. The speed-up (*a*) and the efficiency (*b*) of the parallel algorithm. N is the number of rows and columns in the matrix of the system of linear equations.

The obtained results are close to the announced speed-up of HT technology. So, we can compute faster and use system possibilities more effectively by creating multi-threaded applications to process a large amount of data while working with single processor system with HT technology. We can observe local maximums in Fig.2. The explanation of such variation of the curves is a particularity of cache memory usage and the limitation of the size of this memory. This effect also occurs while working with SMP system. The detailed analysis of this effect is given after the presentation of the results obtained on SMP system.

The tests with sequential and two, three and four thread parallel programs were performed using Dual Xeon Server. The program working time was measured during the experiments. The results are presented in Table 2. The results obtained working with Pentium 4 HT based system, are also presented in the table for the comparison.

The tests have shown that the usage of a sequential program while working with the SMP system is inefficient. The same program with the same data worked faster using a single processor system. The usage of two threads

Table 2. The results of tests using single processor system. N is the number of rows and columns in the matrix of the system of linear equations.

N	Pentium 4 HT		Dual Xeon Server			
	Serial	2 threads	Serial	2 threads	3 threads	4 threads
100	0,593	0,594	0,688	0,5	0,313	0,204
200	1,891	1,297	2,266	1,062	0,891	0,735
300	4,578	2,89	5,203	2,469	2,344	1,687
400	7,563	5,078	13,813	4,391	3,891	2,234
500	11,437	9,328	25,844	8,563	8,171	5,484
600	17,484	11,297	36,031	16,015	13,203	9,906
700	22,579	18,782	53,281	23,312	19,203	14,531
900	39,156	26,157	80,204	36,907	31,172	25,5
1100	62,047	44,313	138,25	58,469	46,437	33,204
1300	91,391	67,641	198,515	84,25	65,625	54,579
1500	129,422	98,234	258,047	115,063	90,609	74,063
1700	176,188	127,907	331,203	143,891	120,391	100,063
2000	264,078	193,281	424,843	192,719	167,922	158,562
3000	756,266	542,281	1028,484	514,813	419,312	358,125

working with the SMP system let us to increase the performance up to 68% (see Fig.3). So, the efficiency of the system work increases. But even using two threads working with the SMP system and with single processor system, programs working time is comparable.

Using HT technology we can work with up to 4 threads in parallel. So, using three threads the speed-up increases up to 72% and working with 4 threads program speed-up reaches 84% (see Fig.3).

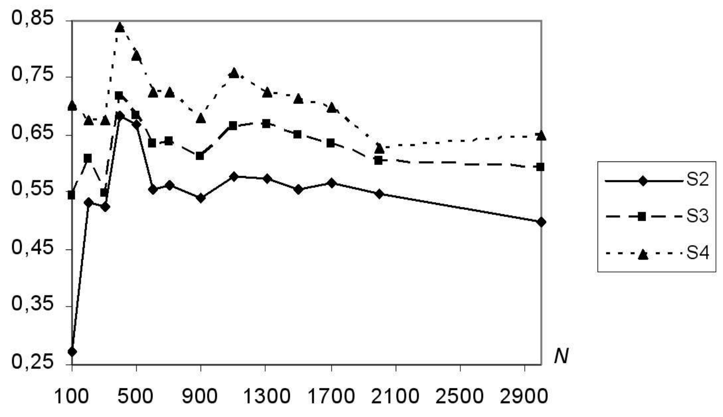


Figure 3. The speed-up of the parallel algorithm.

Working with two threads the efficiency of algorithm is larger than 1 (see Fig.4) it means that using 2 threads not only computation resources affect the speed-up, but also the optimization of memory work. Similar result was presented in [5] for CG benchmark. Working with smaller amount of data the affect of memory optimization is also rather big.

Using more than two threads working in parallel we have no more computation resources (there are only two physical processors in the system) but we still have a speed-up of the program. The tests showed that the usage of three threads is less effective than the usage of four threads. The amount of threads used should be multiple of two to achieve the best results. Such amount of threads is due to the HT technology essence. This technology divides each physical processor into two logical processors. So, to optimize the work of all processors and to achieve better results it is better to use an even amount of threads. Thus the speed-up is largest by using 4 threads (all virtual processors are used).

The efficiency of the parallel algorithm is sometimes bigger using 4 threads than using 3 threads working with smaller amount of data (especially with $N = 500$). The efficiency of the algorithm using 3 or 4 threads is less than 1 (see Fig.4), so it is possible to make conclusion, that in this case operative memory is used more effectively.

We can notice that the speed-up coefficient decreases for larger values of N. This effect occurs because of the influence of operative memory. For bigger N the influence of operative memory is less, so the speed-up coefficient decreases. Fore large N ($N>5000$) the speed-up coefficient stabilizes, the decrement of its value stops. Using 2 threads the efficiency of algorithm is less than using 1 for large matrixes.

Speed-up (also the efficiency of algorithm) grows till $N=500$ and then it declines. This effect occurs because of processors cache. Each processor in the system has 1MB cache memory. Total amount of cache for two processors is 2MB. Working with real numbers (double format in C language uses 8 bytes of memory) the matrix of dimension 500x500 takes about 2MB of memory. So it is the biggest matrix which fits in to the 2 MB cache memory. Working with such amount of data the usage of RAM should be minimal. The particularities of cache memory are the reason of another local maximum of curves in Figures 3 and 4. This maximum occurs with N from 1000 till 1200. 1000 is the multiple of 500 so the usage of cache memory is the possible reason of such variations of the curves.

Similarly working with single processor HT system we have maximums with N 300, 600 and 900. Working with bigger data arrays the influence of RAM increases and the particularities of the processor cache memory work become not so significant. That is why we have no maximums with bigger N.

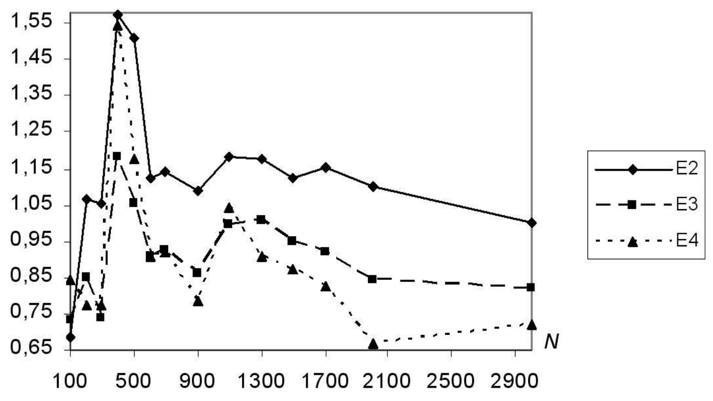


Figure 4. The efficiency of algorithm.

6. Conclusions

In this paper, the evaluation of HT technology working with memory-intensive tasks is presented. The experiments were carried out on different systems: Pentium 4 HT based single processor system and Dual Xeon Server. The tests showed that HT technology lets to increase the speed of single processor system for about 30% by optimizing the work with operating memory. Working with the SMP system, the usage of serial program is inefficient. Even working with two threads, program working time is comparable to single processor system results. HT technology lets to increase the system speed-up to 85%, enhance processor workload and optimize the work of RAM.

The division of physical processor into two logical ones lets to increase efficiency of system work and to solve the problem of shared memory usage partly. While one of the logical processors is calculating, the other one can work with RAM. So, the overall efficiency of physical processor usage increases while working with the large amount of data and using HT technology.

The investigation showed that the usage of HT technology is the efficient way to increase the system speed. To achieve better results we need to control the computation processes properly.

References

[1] R. Čiegis. *Parallel algorithms and net technologies.* Technika, Vilnius, 2005. (in Lithuanian)

[2] G. Dzemyda and S. Ivanikovas. Particularities of parallel programming for personal computers. *Information sciences*, **34**, 257–262, 2005. (in Lithuanian)

[3] G.H. Gloub and C.F. van Loan. *Matrix computations.* The Johns Hopkins University Press, New York, 1996.

[4] A. Grama, G. Karypis, V. Kumar and A. Gupta. *Introduction to parallel computing: design and analysis of algorithms.* Addison Wesley, 2nd edition, New York, Amsterdam, Sydney, Tokyo, 2003.

[5] R.E. Grant and A. Afsahi. A comprehensive analysis of multithreaded OpenMP Applications on Dual-Core Intel Xeon SMPs, workshop on multithreaded architectures and applications (MTAAP'07). In: *In Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007).* Long Beach, California, USA, March 26-30, 2007.

[6] David Koufaty and Deborah T. Marr. Hyper-threading technology in the Nerburst microarchitecture. *IEEE Micro*, **23**, 2003. Issue IEEE Computer Society Press

[7] D. Marr, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J. A.Miller and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, **6**(01), 2002.

[8] M.J. Quinn. *Parallel Programming in C with MPI and OpenMP.* McGraw-Hill Inc., New York, 2004.

[9] L. Ridgway Scott, T. Clark and B. Bagheri. Scientific Parallel Computing. Princeton University Press, 2005.

[10] G. Mattson Timothy. An introduction to OpenMP 2.0. In: *Lecture Notes in Computer Science, Vol.1940. Third International Symposium, ISHPC 2000*, Springer berlin/Heidelberg, 2000, Tokyo, Japan, October 16–18, 2000.

[11] V.V. Voevodin and Vl.V. Voevodin. *Parallel computing.* BHV-Peterburg.