Asian Journal of Information Technology 7 (7): 296-299, 2008
ISSN: 1682-3915
© Medwell Journals, 2008

Performance Enhancement in Wide Range of Applications Through Multi Threading Technology

M. Shanthi
Velammal Engineering College, Chennai, Tamilnadu, India

Abstract: This study exploits the implementation of applications using threading in different platforms. Experimentation is done by writing programs using object oriented technology. Various types of data, like image, text and web data is considered for study. This study also discusses about thread operation models like the master/worker model, the peer model and a thread pipeline model. Sample implementations are also provided to highlight some of the recommended paradigms.

Key words: Image, master/worker, multithreaded, peer model, threading

INTRODUCTION

Threads are essentially sub-processes that share the majority of the resources of their parent process and thus require a lesser amount of system resources. Building programs around threads of execution-that is, around specific sequences of instructions-delivers significant performance benefits. Consider, for example, a program that reads large amounts of data from disk and processes that data before writing it to the screen.

On a traditional, single-threaded program where only one task executes at a time, each of these activities happens as a part of a sequence of distinct phases. No data is processed until a chunk of a defined size has been read. So, the program logic that could be processing the data is not executed until disk reads are complete. This leads to inferior performance. On a threaded program, one thread can be assigned to read data, another thread to process it and a third to write it out to the graphics card. These 3 threads can operate in parallel so that data is being processed while disk reads are going on and overall performance improves.

Multithreading can enable us to keep the responsive by ensuring that the program never goes to sleep. Most programs have moments where they pay no attention to the user: they are too busy doing some work. The general principle is that the thread that's responsible for responding to the user and keeping the user interface up to date (usually referred to as the thread) should never be used to perform any lengthy operation. We might thread our application for different reasons: doing the same work faster, doing more work and offsetting the time taken by slow operations.

Performance: Performance is the perceived measure of how fast or efficient the software is and it is critical to the success of all software. The Pentium 4 processor shows immediate performance improvements across most existing software available today, with performance levels varying depending on the application category type.

The basic performance equation is (John and Patterson, 2003):

$$\text{CPU time} = \text{Instruction\_count} * \text{CPI} * \text{clock\_ cycle}$$

This equation separates the 3 key factors that can measure the CPU execution time. The clock rate usually given can measure overall instruction_count by using profilers/simulators without knowing all of the implementation details. CPI varies by instruction type. We measure performance in a parallel program in terms of speedup. This is the ratio of the programs runtime in parallel to the runtime of the best available sequential algorithms. Assuming that an application is multithreaded (programs written to execute in a parallel manner, rather than a serial or purely equential one), there are inherent difficulties in making a program run faster proportional to the number of processors: the program needs to be written in a parallel fashion and the program itself must be resource friendly.

In the following example:

```
for i = 1 to 10
    a (i) = b (i) + c (i)
    x (i) = y (i) + z (i)
end for
```

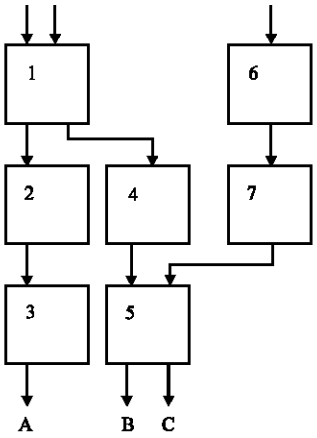

Fig. 1: Block diagram of unique tasks

Table 1: Execution of 7 tasks

Time slot	P1	P2	P3
1	Taks 1	Taks 6	
2	Taks 2	Taks 4	Taks 7
3	Taks 3		Taks 5

this loop can be unrolled and a thread created for each value of i. There are 10 threads which can potentially be executed in parallel, provided sufficient resources are available.

Consider Fig. 1 in which each block represents a task and each task has a unique number. As required by the application taks 2 can be executed only after task 1 is completed and taks 3 can be executed only after taks 1 and taks 2 are both completed. Thus, the line through task 1, 2, 3 forms a thread A of execution. Two other threads B and C are also shown. The thread limits the execution of task to a specific serial manner. Although, the taks 2 has to follow taks 1 and taks 3 has to follow taks 2, taks 4 can be executed in parallel with taks 2 or 3. Similarly, taks 6 can be executed simultaneously with task 1 and so on. Suppose the Multiple Instruction and Multiple Data Stream processor has 3 processors and if task takes the same amount of time to execute, the 7 tasks shown in the Fig. 1 can be executed in the following manner (Table 1).

Here a program is divided into many independent threads. Each thread has an independent stack space but shares a common global address space.

MATERIALS AND METHODS

The analysis stage involves profiling a serial application to determine regions of the application that can most benefit from threading. During the design phase, the critical regions are examined, that are identified during the analysis phase to determine the design changes

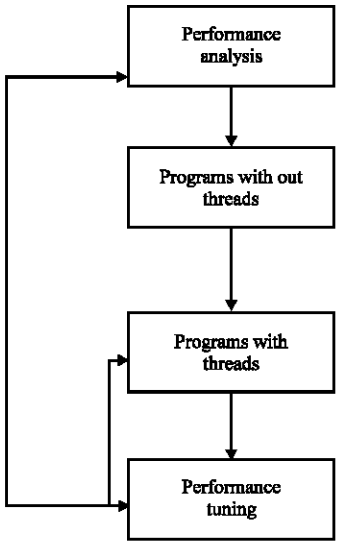

Fig. 2: Steps in threading methodology

required to accommodate a threading paradigm. During implementation and testing, the execution performance and the correctness of the threaded application are verified in Fig. 2 (Intel Corporation, 2003).

Literature survey: Balakrishnan and Nandy (2000) showed that multithreaded architectures coupled with SIMD parallelism provide performance improvement in excess of 2% over conventional super scalar architectures. In Pop and Kumar (2006), the results indicate that processors can substantially benefit from multithreading, even in systems with small caches, provided sufficient network bandwidth exists. Increased network contention due to multithreading has a major effect on performance. Their experimental results show that by taking advantage of SMT technology we achieve a 30-70% improvement in throughput over single threaded implementations on in-memory database operations. In Guitart (2005), multithreading system provides speedup factors between 1.27 and 1.65. Simulations were performed in 3 steps. The first step is a sequential execution of tasks, like in the uniprocessor system. The second step is a parallel multithreaded simulation with one thread for each task. Threads are then scheduled by operating system and synchronized by locks and the conditions variables. Third step is also a multithreaded parallel simulation but with the same number of threads and processors. These threads execute tasks according to their structure with a simple scheduling. Watheq *et al.* (1998) explained the concept by maintaining multiple process contexts in hardware and switching among them in few cycles, multithreaded processors can overlap computation with

memory accesses and reduce processor idle time. Panit and Moore (2002) presents an architectural study of JMA, a highperformance multithreaded architecture which supports Java-multithreading and real time scheduling whilst remaining low-power.

Multimedia applications are fast becoming one of the dominating workloads for modern computer systems. Since, these applications normally have large data sets and little data-reuse, many researchers believe that they have poor memory behavior compared to traditional programs and that current cache architectures cannot handle them well. It is therefore, important to quantitatively characterize the memory behavior of these applications in order to provide insights for future design and research of memory systems. Since, typical media processing applications, especially 3D animation graphics, have very large data sets and little data-reuse, many researchers believe that existing cache schemes cannot handle these applications efficiently when compared to traditional.

RESULTS AND DISCUSSION

Threading in banking applications: Program is written in Java with and without threads for banking application. Program with thread takes less time than the program with out thread. In this application, we consider only 2 operations deposit and withdrawal with 20 records. Study shows that searching time varies based on the position of the record and the value of the amount deposited or withdrawn (size of the figure 1000, 100000). Since, the process involves user interaction with the system, timing difference plays a major role in the analysis. Thus, the analysis is repeated 10 times with the same input value and the average is considered as the final value. The following Fig. 3 shows the analysis for the deposited value of Rs. 1000. The program is run in Pentium processor with HT supported machine and database used MS ACCESS AND JAVA SWING.

The average is found to be 296 ms without threads and 140 ms with threads. The Fig. 3 shows that threads are applicable for all applications to efficient functioning.

Threading in client-server applications: The evolution of the multi-threaded processor design is the trend for the next generation desktop processors. To add a record from the client side, a program is written and executed in Java with and without thread. The results are tabulated.

If we compare the average value of 2, program with threading takes the advantage (117 ms) over without threading (157 ms). So, it is evident that threading in client-server application is faster always irrespective of number of records (Fig. 4).

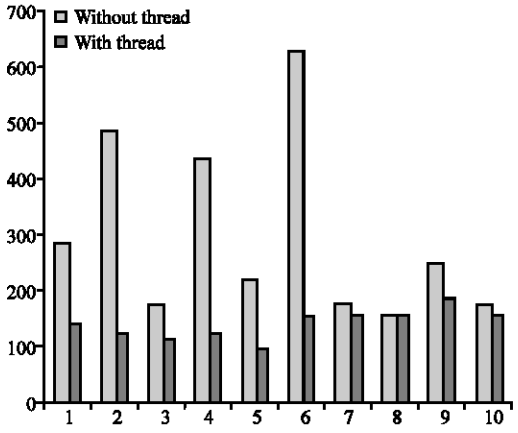

Fig. 3: Threading in banking application

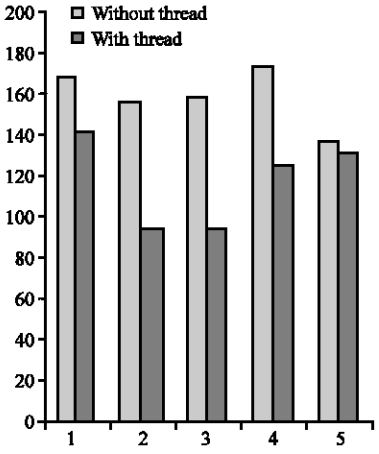

Fig. 4: Threading in client-server applications

Threading in scientific applications

Random number generation: The following are sample data which shows the results for random number generation upto 3000 numbers. The program is written in Java with and without threads. The program is executed in Pentium HT machine and the time taken by the CPU in milliseconds is listed Fig. 5.

The average values are 787 ms (with out thread) and 164 ms (with thread). It is evident that threading is advantage.

Threading in producer consumer applications: Two application program for producer consumer is written in Java. In the first program, Main itself a thread and main forks another thread. This thread deviates in to producer and consumer. The other program, Main is a thread which interleaves into 2 threads into producer and consumer. The first program takes 4125 ms and the second takes 4890 ms (Fig. 6).

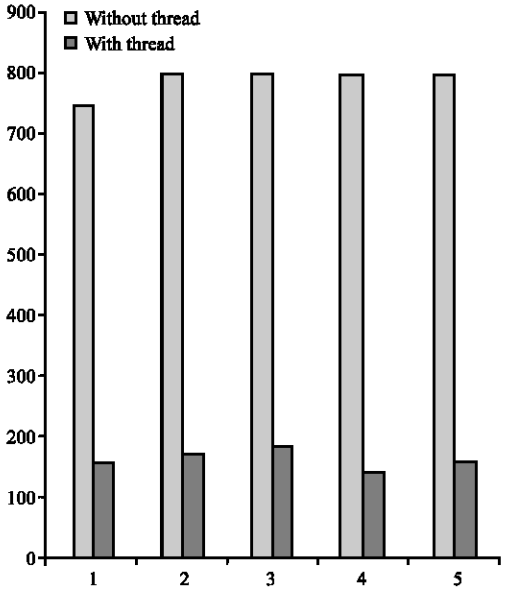

Fig. 5: Threading in scientific applications

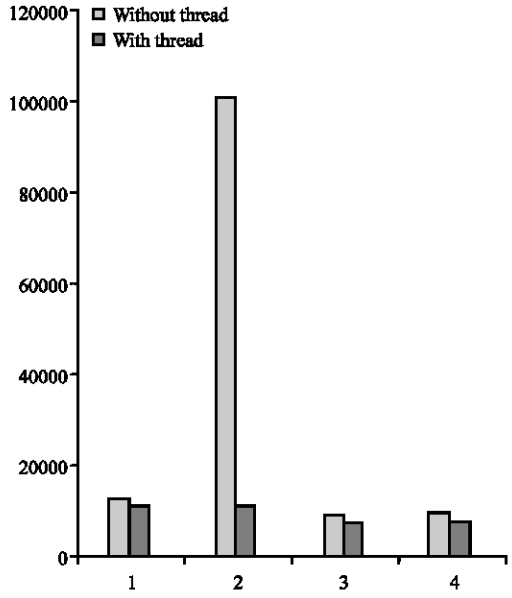

Fig. 6: Threading in producer consumer applications

Table 2: Threading in multimedia applications

S. No	No. of threads	Time (ms)	CPI
1	Without thread	203	4.7
2	Two	188	3.4

Threading in multimedia applications: Threading is really advantages because it increases the processor time irrespective of the image format. This is observed by writing a multimedia program in Java for loading the images. The program is written with and with out threading concept and executed in Pentium IV Processor with HT enabled systems. Based on the tabulated values, it was evident that threading gives better performance (Table 2).

CONCLUSION

Multithreading has emerged as one of the most promising and exciting techniques used to increase the performance of applications. Many software applications deliver appreciable performance gains on the Pentium 4 processor by directly benefiting from higher clock rates and micro architectural enhancements. This is only an introduction on how to get an easy performance gain using thread to take advantage of Multithreading Technology on an Intel Pentium processors. In future, this may be extended to thread operation models.

REFERENCES

Balakrishnan, S. and S.K. Nandy, 2000. Performance evaluation of multithreaded architectures for media processing applications. In: Proc. IEEE. Int. Symp. Circ. Syst. ISCAS, 1: 531-534.

Guitart, F.J., 2005. Performance Improvement of Multithreaded Java Applications Execution on Multiprocessor Systems Performance.

Intel Corporation Threading Methodology, 2003. Principles and Practices version 2.0.

John, L.H. and D.A. Patterson, 2003. Computer Architecture: A Quantitative Approach. 3rd Edn. ISBN: 1-55860-596-7A.

Panit, W. and S. Moore, 2002. The Java-Multithreading Architecture (JMA) for embedded processors. Computer Laboratory, University of Cambridge, UK. IEEE. Int. Conf. Comput. Design, pp: 527.

Pop, R. and S. Kumar, 2006. On performance improvement of concurrent applications using simultaneous multithreaded processors as NoC resources. Engineering Jonkoping University. In: Norchip Conference, pp: 191-196.

Watheq, M. El-Kharashi, F. ElGuibaly and K.F. Li, 1998. Multithreaded processors: The upcoming generation for multimediachips. Advances in Digital Filtering and Signal Processing. IEEE. Symp., 5 (6): 111-115.