Matrix Multiplication Performance on Commodity Shared-Memory Multiprocessors

G. Tsilikas and M. Fleury
University of Essex, Department of Electronic Systems Engineering,
Wivenhoe Park, Colchester, CO4 3SQ, United Kingdom
e-mail: fleum@essex.ac.uk

Abstract

Cache-oblivious algorithms for matrix multiplication are confirmed as an effective way of exploiting Intel architecture shared-memory multiprocessors. The performance also remains consistent across a wide range of matrix size. The Cilk programming environment remains an effective way of implementing this type of algorithm, but the need for portability and a compiler upgrade route mean that a portability library is a competitive alternative. The paper considers the interaction of matrix multiplication algorithms with the memory hierarchy, as well as multithreading across differing operating system variants and compilers.

1. Introduction

Matrix multiplication, with time complexity O(n^3), is a fundamental algorithm for which the performance is highly-dependent on the organization of the memory hierarchy [5]. Research is on-going into different memory access patterns, especially when using multi-threading on a Symmetric Multiprocessor (SMP) [8; 1]. Matrix multiplication has been used as a demonstrator for the Cilk language [4]. Associated with Cilk are a set of 'cache-oblivious' algorithms [3], including one with a recursive structure for generalized matrix multiplication. The first objective of this research was to judge the performance of this algorithm on two low-cost Intel quad SMPs, typical of commodity multiprocessors normally intended for commercial database servers.

Cilk provides a simple set of extensions to the 'C' programming language, including thread `spawn`, and `sync`, local barrier synchronization. Cilk is pre-processed to output C source code, with optimized thread allocation and memory management. At compile time, a run-time scheduler is linked in, and at run time the scheduler performs work stealing between per-thread task dequeues. Therefore, a second objective of this research was to find what Cilk itself contributed to the performance of the algorithm. It was not expected that the Cilk scheduler's influence would be significant, as matrix multiplication is completely deterministic, but a comparison was made with an in-house, light-weight, multi-threading library aimed at portability (referred to as the portability library).

Underlying Cilk are POSIX threads, which are weakly supported in the Windows NT operating system (o.s.), and, hence, Cilk will not run in this environment. One of the SMP's tested, the Dell PowerEdge 6400 with quad Pentium-III Xeons at 700 MHz with 1 GB memory, 1 MB L2 cache could run Cilk under Linux. However, the other, an Silicon Graphics 540 Visual Workstation with the same processors at 550 MHz with 512 MB memory, 512 KB L2 cache, does not have a suitable version of Linux. Therefore, portability library implementations of the algorithms were used for this machine.

Performance tests were also made on the Dell machine varying the operating system, the compiler, as well as algorithm. This allowed a more direct comparison of the performance of the two o.s. for this algorithm.

2. Matrix multiplication algorithms

A simple-minded or naïve implementation of matrix multiplication applies the mathematical method of calculation, multiplying row by column, and then allocates a set of rows to each thread. This method is used as a base-line in the tests reported herein. Though programmatically convenient, the need to skip over rows (assuming row-major storage) results in poor data locality.

The Cilk algorithm is based in a divide-and-conquer principle. If $C_{[m\times p]} = A_{[m\times n]}B_{[n\times p]}$ then the algorithm halves the largest of the three dimensions (m, n, p). While theoretically the division ends when $m = n = p = 1$, in which case the two elements are multiplied and added to the result matrix, the Cilk algorithm stops dividing the matrices when $m + n + p <= k$, where k relates to the cache line or

block size, though, in practice, k is found experimentally. Effectively, the sub-problem will be small enough to fit in the cache so that a minimal number of cache misses occur. The Cilk memory hierarchy model for practical reasons assumes a single level-one cache.

Cilk performs the divide-and-conquer recursively, which is a convenient way to account for rectangular A and B matrices with differing dimensions, especially when the dimensions are not exact multiples of k. In fact, a number of Cilk examples are coded in a recursive style, possibly to take advantage of Cilk's language features. However, there is an overhead from first setting up the tasks recursively. An alternative is to divide the matrices into small blocks and proceeds to iteratively process these blocks. By decomposing C into per thread strips, it is possible to automatically find the corresponding A and B strips, in such a way that no locking on matrix C is required when partial per-block results are aggregated. The iterative block algorithm was implemented using the portability library.

3. Commodity multiprocessors

Commodity quad or 4-way PC multiprocessors of the type considered in this paper are becoming common. [9] surveyed sixteen such systems all with similar characteristics, with the survey taking place around the time of purchase of the machines used in this paper. The server specification appears more stable than that for personal PC's, perhaps because of the greater need to balance memory hierarchy with processor speed. Most surveyed had 550 MHz Pentium-III Xeon processors, though faster clock speeds are possible. Level-2 cache size is an important determinant of performance, as it acts to reduce contention to the common memory, with a size of 512 KB per processor being typical, though a size of 1 MB or more is possible. Multiprocessor cache coherency is governed by an Intel variant of the MESI protocol [2]. Current systems also include a level 3 cache, with on-chip level 2 cache.

The Intel multiprocessor architecture [6] is symmetric, in the sense that no processor has a privileged position. Therefore, all processors can execute the operating system. Specifically: a) all processors share the same memory address space b) all processors share access to the same I/O system. The intention of the SMP architecture is to support scalability, though at the time of writing PC 8-way systems appear to be rare.

In the Intel multiprocessor architecture, Figure 1, a distributed APIC (Advanced Programmable Interrupt Controller), *i.e.* with an APIC module at each node, is used to coordinate the processors. A bootstrap processor (BP) is negotiated at power-up. Subsequently, once the presence of other application processors (APs) is determined, a task is loaded into memory of a given processor under the control of the multiprocessor operating system, and then the processor is started by an interrupt over the APIC or Interrupt Controller Communications (ICC) bus. Once started and until shutdown, the role of the BP is in abeyance.

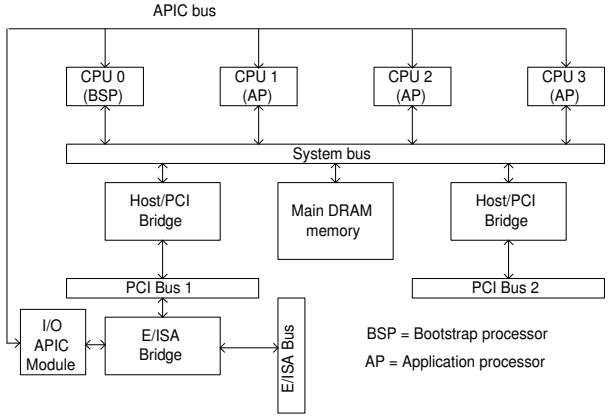

Figure 1. Standard Intel SMP architecture

PC servers connect to a memory controller over a shared front side or system bus, typically running at between 100 and 133 MHz, and otherwise a potential bottleneck. Unlike, earlier uniprocessor Pentium-based systems, the front side bus can overlap or split up to eight transactions at a time. The SGI machine has a customised 256-bit memory bus and the manufacturer claims a bandwidth of 3.2 GB/s between memory controller and main memory. The front side or system bus between processors and memory controller operates at 100 MHz, but the PowerEdge 6400 bus is similarly clocked.

4. Software issues

The API is built upon the following model of computation. Threads are assigned from a thread pool to various processors within a multiprocessor. In Fig. 2, tasks are assigned to threads, and thence to processors, which form a virtual parallel machine. When any thread finishes what it was doing it can then request a new task from the task pool, with limited start-up overhead.[1]

The performance tests were performed separately with the 2.4 and 2.6 version of the Linux kernel, being the memory resident part of the o.s. common to all software releases. The main changes to the 2.6 kernel[2] that have a

[1]Further details of the API of the portability library and of other features of the systems under test are to be found in "Multimedia Applications on Commercial Multiprocessors", G. Tsilikas and M. Fleury, submitted to the Parallel Computing Journal, 2003.

[2]Refer to "Towards Linux 2.6", A. K. Santhanam,

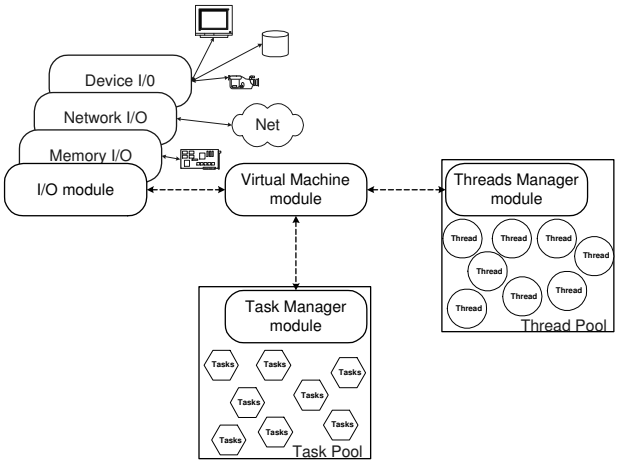

Figure 2. Portability library computation model

bearing on SMP multithreading performance for the matrix multiplication application are: improved threading support, with a re-write of the code to allow the Native POSIX Thread Library to run, and a 1:1 user to kernel thread ratio; a new thread/process scheduling algorithm, removing a global timeslice leading to idling processors or processes "bouncing" between processors; and the use of memory pools to speed up memory allocation.

Though kernel v. 2.6 provides a `futex`, which avoids blocking in some circumstances, locks were only used at initialization time in the blocking algorithm. Locking was needed but in the recursive algorithm substituted for the Cilk algorithm. Cilk reduces locking overhead by a 'fast' and 'slow' clone of a Cilk procedure, the latter operating when work stealing is underway.

5. Timing Results

Figure 3 shows tests for a range of square matrix size to double precision, with Red Hat v. 7.2. The plots are annotated with the number of threads, and though the underlying Linux o.s., kernel version 2.4.9-31smp, is responsible for thread placement, it is safe to assume that threads are placed on a per-processor basis. The non-Cilk algorithms were implemented without calls to the portability library, and a recursive single processor version of the Cilk algorithm was directly derived from Cilk by removing Cilk directives. A value of $k = 128$ was found to give good performance on the Dell machine for the Cilk algorithm, which incidentally implies that tuning to cache line sizes will improve performance, and, hence, the algorithm is not completely oblivious of cache size. A value of 32 was by found by experiment to be a good setting for the block size in the block algorithm. The `gcc v. 2.96` compiler optimization settings did have a significant effect for algorithms run under the portability library, either level two or three being selected depending on algorithm. However, the Cilk implementation did not benefit from more aggressive optimization settings. Under Linux, good speed-ups are achieved

`http://www-106.ibm.com/developerworks/linux/library/l-inside.html`.

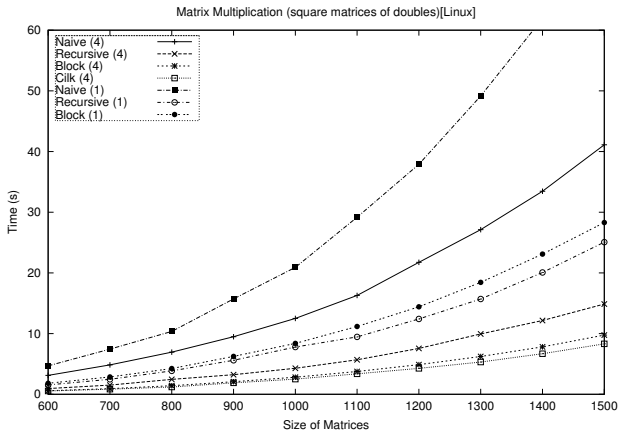

Figure 3. Matrix multiplication on the Dell PowerEdge 6400 running Linux

by Cilk followed closely by the block algorithm. It is clear that the naïve algorithm is not at all competitive, as even the multithreaded version is outperformed by the sequential versions of both the Cilk and block algorithms.

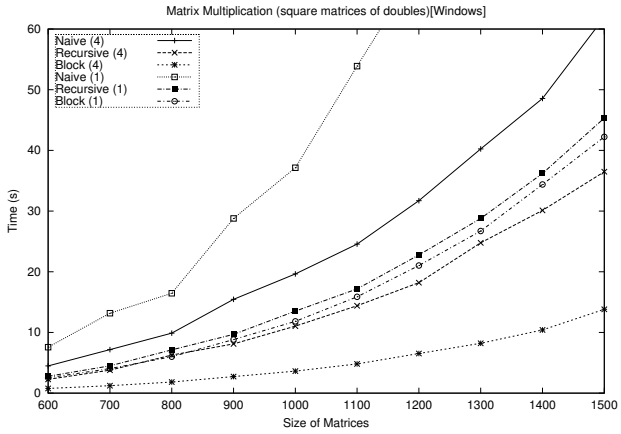

Figure 4. Matrix multiplication on the SGI 540 Workstation running Windows NT

Under Windows NT (Service Pack 3), Fig. 4, running on the SGI machine and compiling with `cl` version 12.00.8804 for 80x86, again the recursive and block algorithms outperform the naïve one. However what should be noted is that the block algorithm seems to perform better than the recursive one, which is intended as a Cilk equivalent. That is true for the sequential version but even more so for the parallel version.

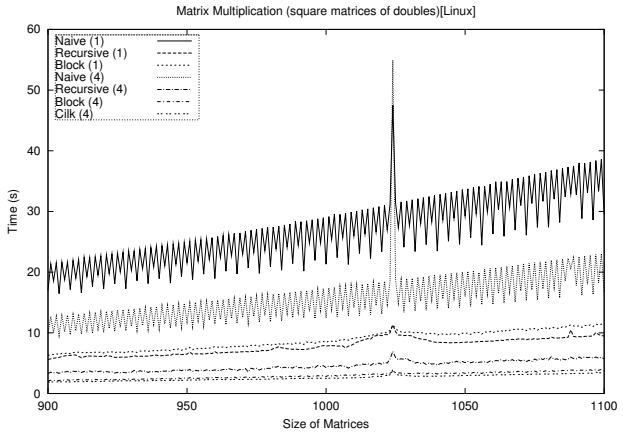

Figure 5. Fine-grained timings of matrix multiplication on the Dell PowerEdge 6400 running Linux

A further comparison was made by running the same tests but taking timings at more frequent intervals, Figs. 5 & 6. There are performance drop spikes for both machines, though these are more frequent for the SGI machine. The spikes are very severe in their effect on the naïve algorithm. The cache-oblivious algorithms, block, recursive or Cilk, is to smooth out the oscillations seen in the naïve algorithm plots, also making performance more predictable. The same can be said about increasing the number of threads (processors) from one to four.

Winograd's algorithm for matrix multiplication [1], $O(n^{\log 7})$ time complexity for multiplications, was found to be much slower on the SGI machine. For example, with no threads for a 500×500 pixel image using the naïve algorithm, the time was 5.598 s, but for the Winograd algorithm the time was 9.916 s. For the Dell machine, no clear difference in timing was found between the two algorithms. These results show that the theoretical advantage of reducing multiplications is easily balanced by an increase in the number of additions.

To allow a more direct comparison between the performance of the operating systems, Windows 2000 Pro (Service Pack 5) was also installed on the Dell PowerEdge 6400. (As is well-known, Windows 2000 is a variant of Win-

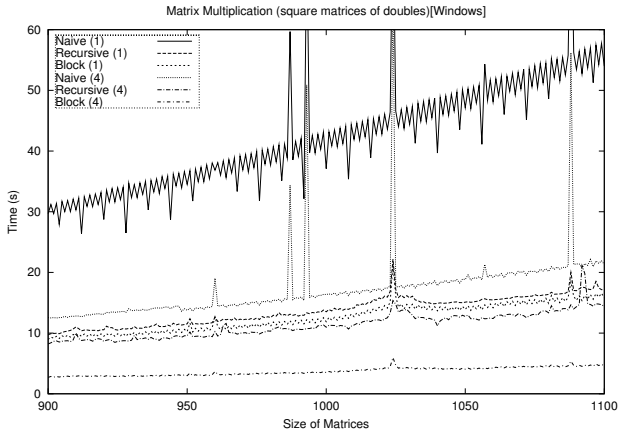

Figure 6. Fine-grained timings of matrix multiplication on the SGI 540 Workstation running Windows NT

dows NT [7].) The results of taking timings at coarse- and fine-grained intervals is seen respectively in Figs. 7 & 8. The coarse-grained measurements show that especially for larger-sized matrices the performance of the Dell machine is much improved, especially for the block algorithm. Both the coarse- and fine-grained measurements show more consistent performance over a range of matrix sizes than when running the same algorithms on the SGI machine.

Comparative timings when the same algorithms were run on the Dell machine, but with different versions of the Linux kernel. Red Hat Linux version 7.2 supports version 2.4 of the kernel, whereas the Debian software release is able to support version 2.6.4 of the kernel. There are some differences in compiler technology. version 2.96 of the `gcc` compiler is a specialized version from Red Hat, and, therefore, the nearest equivalent under Debian is version 2.95. Cilk would not compile under version 3.3 of the compiler. Figure 9's comparison between the block algorithm with change of kernel demonstrates a consistent improvement in performance, all measurements being taken with four threads (processors). However, there is a greater improvement from changing the compiler version with the same kernel, version 2.6. The Cilk implementation remains good in all circumstances. However, timings for Cilk under different kernel versions were virtually identical, Fig. 10 which may be attributable to the Cilk run-time load balancing component. In this instance, the Cilk load balancer may over-ride kernel thread-scheduling. When finer-grained measurements were made, it became apparent that the coarse-grained timings in Fig. 9 are somewhat misleading, as there are matrix sizes for which the block algorithm outperforms the Cilk implementation, Fig 11, particularly

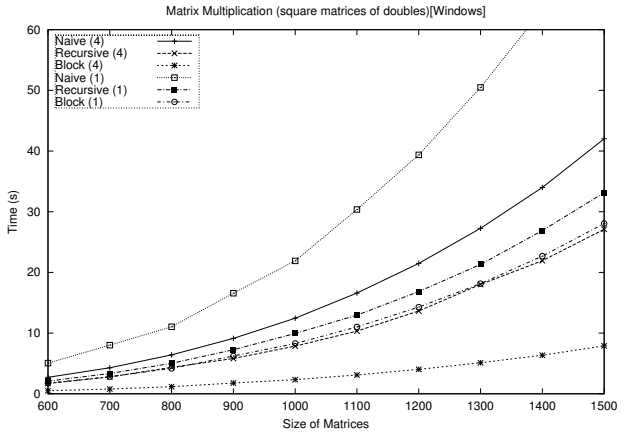

Figure 7. Matrix multiplication on the Dell PowerEdge 6400 running Windows Pro 2000

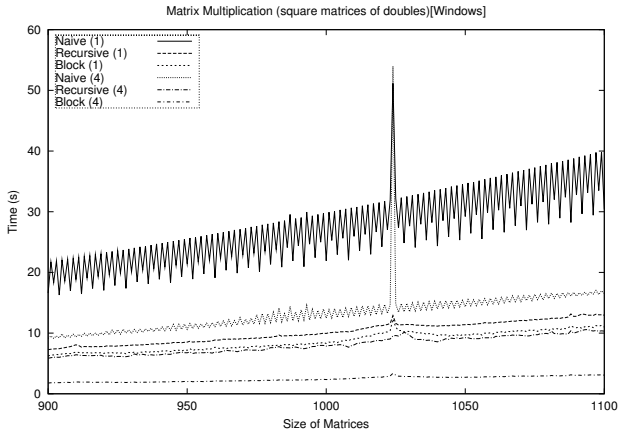

Figure 8. Fine-grained timings of matrix multiplication on the Dell PowerEdge 6400 running Windows Pro 2000

at the performance-drop spike.

Comparison between the times for block/Cilk algorithms on the Dell machine with the two operating systems, shows equivalent performance. Speed-up on the Dell machine, Fig. 12 with timings taken for 1, 2, 3 and 4 processors, shows the Cilk and block algorithms gaining little beyond three processors, whereas the other algorithms gain from more processors, but equally the performance is weaker. Marginally best performance arises from the Windows 2000 Pro (W2K) block algorithm version.

6. Conclusions

The tests showed that simple observation of cache line sizes, can achieve effective performance of matrix multiplication on commodity SMPs. Compared to a cache-dependent algorithm, the timings of cache-oblivious algorithms are considerably less prone to fluctuations or even performance loss spikes. This performance improvement arises either by implementation using the Cilk language or directly in C. The Cilk run-time component does improve performance over and above the use of a cache-oblivious algorithm. This may be ascribable to built-in thread scheduling or some other effect. However, Cilk is dependent on the presence of POSIX threads and, currently, on the compiler version. This means that the algorithm will not run on one on the common operating systems for commodity SMPs, namely Windows NT or 2000 Pro. By means of a portability library it was possible to overcome this problem, and at the same time, the library re-compiled under an improved version of the compiler. The main scheduling method in the portability library was through a task pool. However, with so few threads and no need to dynamically load balance, it is not thought that the method of scheduling is significant. In respect to matrix multiplication, changes to the Linux kernel and to the supplied compiler appear to be matched by developments in the Windows environment.

References

[1] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, 1999.

[2] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, San Francisco, 1999.

[3] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297, 1999.

[4] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded library. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 212–223, 1998.

[5] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University, Baltimore, 1996.

[6] Intel Corporation, Mt. Prospect, IL. *Multiprocessor Specification*, 1997.

[7] D. A. Solomon. *Inside Windows NT*. Microsoft Press, Redmond, WA, 1998.

[8] R. A. van de Geijn and J. Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997.

[9] M. Weitz. 4-way servers. *Windows NT Magazine*, pages 141–143, 2000.

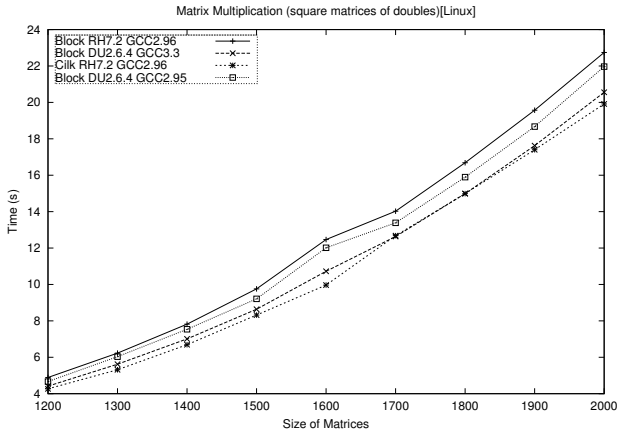

Figure 9. Matrix multiplication on the Dell PowerEdge 6400 running different versions of the Linux kernel and the `gcc` compiler

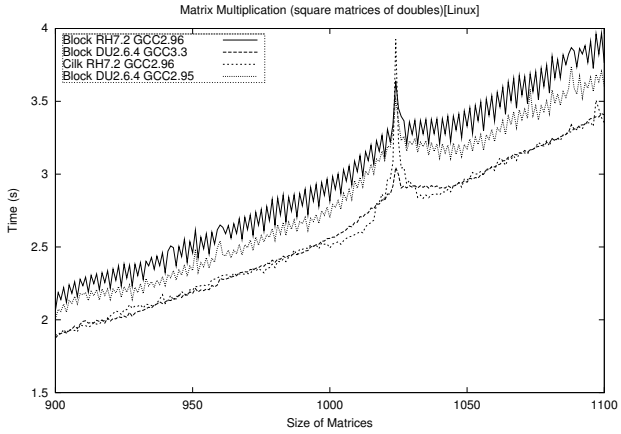

Figure 11. Fine-grained timings of matrix multiplication on the Dell PowerEdge 6400 running different versions of the Linux kernel and the `gcc` compiler

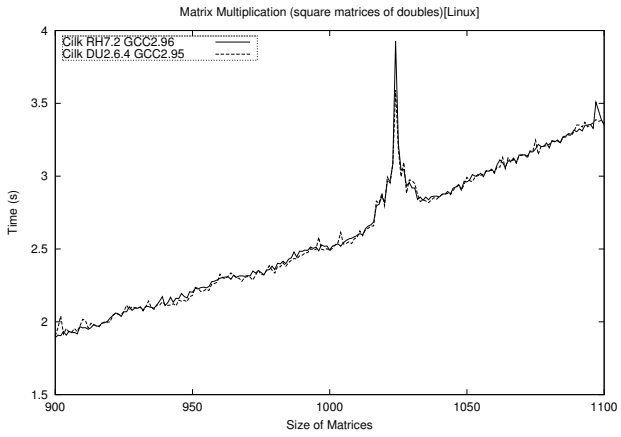

Figure 10. Fine-grained timings of Cilk matrix multiplication on the Dell PowerEdge 6400 running different versions of the Linux kernel and the `gcc` compiler

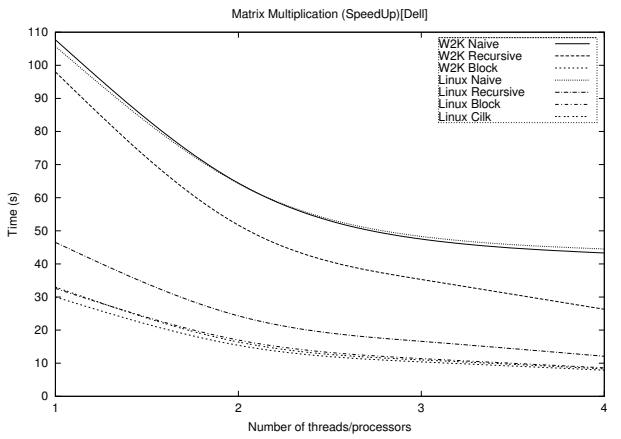

Figure 12. Speed-up under Linux and Windows 2000 Pro on the Dell PowerEdge 6400