Performance of High-Accuracy PDE Solvers on a Self-Optimizing NUMA Architecture

Sverker Holmgren and Dan Wallin

Uppsala University, Information Technology, Department of Scientific Computing
P. O. Box 120, SE-751 04 Uppsala, Sweden
{Sverker.Holmgren,Dan.Wallin}@tdb.uu.se

Abstract. High-accuracy PDE solvers use multi-dimensional fast Fourier transforms. The FFTs exhibits a static and structured memory access pattern which results in a large amount of communication. Performance analysis of a non-trivial kernel representing a PDE solution algorithm has been carried out on a Sun WildFire computer. Here, different architecture, system and programming models can be studied. The WildFire system uses self-optimization techniques such as data migration and replication to change the placement of data at runtime. If the data placement is not optimal, the initial performance is degraded. However, after a few iterations the page migration daemon is able to modify the placement of data. The performance is improved, and equals what is achieved if the data is optimally placed at the start of the execution using hand tuning. The speedup for the PDE solution kernel is surprisingly good.

1 Introduction

The kernel in many important computational codes consists of multi-dimensional fast Fourier transforms, i.e. 2D, 3D, or higher-dimensional FFTs. One area where such computations arises is the numerical solution of partial differential equations (PDEs) using spectral or pseudospectral discretizations. Such methods are used in a wide spectrum of applications, e.g. computations of turbulent flows for optimization of aircraft performance, numerical weather prediction, and ab initio computations for predicting the outcome of chemical reactions.

When using discretization methods employing structured grids, the data is represented as large multi-dimensional arrays where the size is determined by the number of grid points. Using many grid points generally yields a more accurate solution. For multi-dimensional problems, the resolution is in practice often limited by the amount of main memory available. The major advantage of employing pseudospectral discretizations is that, for many problems, it gives the best possible accuracy for a given number of grid points.

The time-consuming part in a PDE solution algorithm consists of computing approximations of the derivatives. In a pseudospectral scheme, these computations are performed using multi-dimensional FFTs, which are *global* multi-stage grid operations. Each stage has a specific communication pattern involving a

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 602–610, 2001.

large amount of data, and every value in the solution array is updated using information originating from all other grid points. At a first glance, this is a very difficult situation for parallel computations. However, since the communication patterns are static and highly structured, efficient parallel implementations are possible. A number of quite efficient parallel implementations for multi-dimensional FFTs have been developed. For example, the FFTW package [3] includes both a multi-threaded (Pthreads) and a message passing (MPI) implementation.

We study a multi-threaded implementation of a kernel representing a PDE solver employing a pseudospectral discretization [10]. The aim is to examine the parallel performance of an important non-trivial algorithm with significant inherent communication on a cc-NUMA [6] system with SMP nodes. A similar investigation has earlier been performed for a finite difference solver kernel [9], which only involves local grid operations and very little communication. For our more realistic problem, we want to investigate the performance effects of self-optimizations such as page migration and replication. For a programmer, it is of interest to know how successful such techniques are. The result determines the importance of performing careful hand tuning, considering data allocation and thread scheduling policies.

The kernel algorithm is described in Section 2. The WildFire architecture and the different configurations used are presented in Section 3, and in Sections 4–7, a number of performance experiments are presented.

2 A Generic PDE Solver Using a Pseudospectral Method

The high-accuracy derivative approximation in a pseudospectral solver is performed by a convolution, i.e. a transform to frequency space, a local multiplication, and an inverse transform back again. For a uniform grid, the FFT and its inverse yield a very efficient tool for the transformations, resulting in $\mathcal{O}(n^2 \log_2 n)$ arithmetic complexity for computing the derivatives on a grid with n^2 grid points. Normally, the computation is performed within an iterative solver or a time-marching procedure. Hence, a representative kernel for a pseudospectral solver is an iteration where the loop body consists of convolution computations.

The standard implementation of a 2D FFT is to first perform 1D FFTs for all the columns in the data matrix, and then do the same for all the rows. In a convolution computation, this results in a five-stage scheme described in Figure 1. In general, it is sufficient to study 2D problems to get a picture of the performance also for multi-dimensional pseudospectral solvers, since the FFTs for the extra dimensions will be performed locally.

In Figure 1, each arrow represents a 1D FFT. For a vector of length n, this is a $\log_2 n$-stage computation involving a rather complex but structured communication pattern. There are a number of different FFT algorithms available, for a review see, e.g. [7]. In the experiments presented here, we use an in-place radix-2 Gentleman-Sande version of the FFT, and an radix-2 in-place Cooley-Tukey version for the inverse transforms. This allows for a convolution algorithm where no

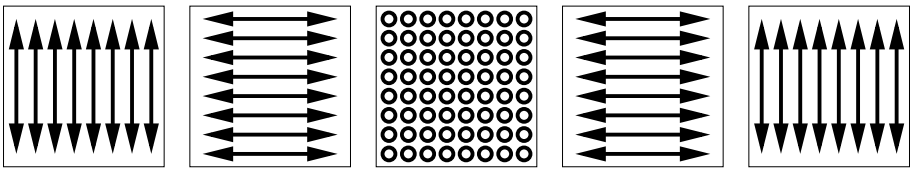


Fig. 1. A single convolution computation for a 2D problem

bit-reversal permutations are required. This is important, since the bit-reversal permutation introduces a lot of communication, and affects the performance significantly. Also, the FFTs should be performed in situ. If workspace is used, the maximal number of grid points is reduced, leading to a less resolved solution.

3 A Self-Optimizing cc-NUMA Architecture

The Sun WildFire system [5] is a prototype architecture developed to evaluate a scalable alternative to symmetric multiprocessors (SMPs). WildFire can be viewed as a cc-NUMA system with self-optimizing features, built from unusually large SMP nodes. Up to four nodes, each with up to 28 CPUs, can be directly connected by a point-to-point network between the WildFire Interfaces (WFI) in each node.

The experiments presented in this paper have been performed on the two-node WildFire system Albireo at the Department of Scientific Computing, Uppsala University. Each node has 16 processors (250 MHz UltraSPARC II with 4 Mbyte L2 cache) and 4 Gbyte memory. Logically, there is no difference between accessing local and remote memory, even though the access time varies: 310ns for local and 1700ns for remote memory. Coherence between all the 32 caches is maintained in hardware, which creates an illusion of a system with 8 Gbyte shared memory.

In order to ease the burden on the programmer, different forms of optimization are supported by the system. A software daemon detects pages which have been placed in the wrong node and migrates them to the other node. The daemon also detects pages used by threads in both nodes and replicates them. WildFire's cache coherence protocol keeps the coherence between replicated memory pages with a cache line granularity. This is called *Coherent Memory Replication* (CMR), but the technique is also sometimes referred to as Simple COMA (S-COMA) [4]. The maximal number of replicated pages as well as other parameters in the page migration and CMR algorithms may be altered by modifying system parameters.

The codes were written in Fortran 90 using double precision complex (16 byte) data. The program was compiled and parallelized using the Sun Forte 6.1 compiler employing OpenMP-directives. In OpenMP, we use the default static scheduling. The experiments were performed on a lightly loaded system.

On WildFire, allocation of data uses a first-touch policy. The allocate statement reserves virtual address space, and the physical memory is allocated on the node where the thread first touching the data resides. The threads normally stay on the processor they are spawned at. The default WildFire scheduling policy is to, if possible, confine the threads to a single node. Only if the number of threads is larger than the number of processors in the first node, threads are spawned also on the other node. The compiler does not support memory placement directives. However, data distribution can be achieved by using a system call to bind the threads to a specific node and utilize the first-touch policy. Data placement should therefore be carried out within a parallel region to achieve a beneficial allocation.

If both page migration and CMR are disabled, the code will run in pure cc-NUMA mode. We have used the configurations listed below. Here, thread matched allocation means that the data is allocated such that the FFTs in phase 1 can be computed without introducing any remote accesses:

1. **Single node SMP** - Data is allocated on one node. The threads are bound to the same node. Migration and replication is turned off.
2. **Single node allocation WildFire** - Data is allocated on one node. The threads are not bound, and the WildFire default scheduling algorithm is used. Migration and replication is turned on.
3. **Thread-matched allocation WildFire** - Data is allocated using thread matching. The threads are not bound, and the WildFire default scheduling algorithm is used. Migration and replication is turned on.
4. **Single node allocation balanced WildFire** - Data is allocated on one node. The threads are evenly distributed between the two nodes and bound. Migration and replication is turned on.
5. **Thread-matched allocation balanced WildFire** - Data is allocated using thread matching. The threads are evenly distributed between the two nodes and bound. Migration and replication is turned on.
6. **Single node allocation balanced cc-NUMA** - Data is allocated on one node. The threads are evenly distributed between the nodes and bound. Migration and replication is turned off.
7. **Thread-matched allocation balanced cc-NUMA** - Data is allocated using thread matching. The threads are evenly distributed between the nodes and bound. Migration and replication is turned off.

4 Parallelization of the Pseudospectral Solver Kernel

As seen in Figure 1, the 1D FFTs in the convolution algorithm are first carried out for the columns of the data matrix, and then for the rows. For large number of grid points, experiments show that applying the FFTs directly to the matrix rows is not efficient. Using this type of implementation leads to extremely poor cache utilization, and the performance and speedup for large problems is not acceptable. If the threads reside in both nodes, optimizations like page migration and CMR are not able to detect and adapt to the changing access pattern fast

enough, and in practice almost no migration/replication occurs. Hence, a large amount of remote accesses further degrades the performance.

To improve cache utilization and to allow for more efficient communication between the nodes, our experiments show that a better scheme is to explicitly transpose the data matrix, and again apply the FFTs to matrix columns. After applying the 1D FFTs in one direction, FFTs in the other direction should be computed. If the threads reside on more than one SMP node, some data will always be located on a remote node when the transpose is performed. On a two-node system with evenly distributed data, the lower left and the upper right matrix blocks will have to be exchanged between the nodes in the transpose operation, see Figure 2. Half of the data matrix will bounce back and forth

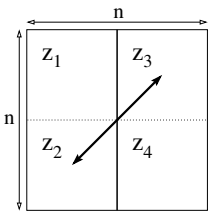


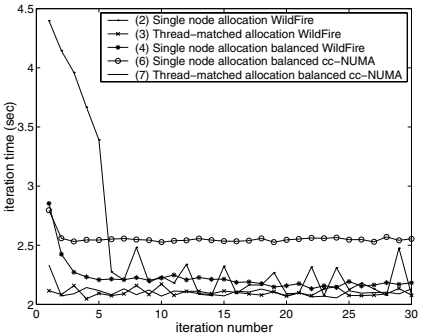
Fig. 2. The $n \times n$ matrix z consists of the blocks z_1, z_2, z_3 and z_4. If the data is evenly distributed between the two SMP nodes, the z_2 and z_3 block will travel across the WFI when the matrix transpose is applied

between the two nodes, still causing a large amount of communication over the WFI. In our implementation, the parallel transpose operation is performed using the ZTRANS routine in the Sun Performance Library.

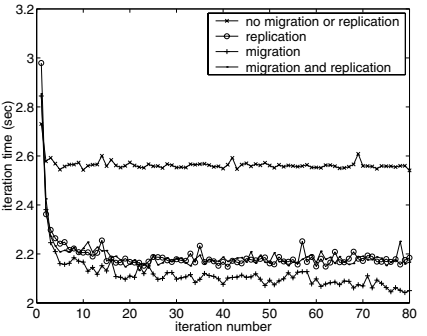
5 Impact of Migration and Replication

The time per iteration for the PDE solver kernel for several configurations using 24 threads is shown in Figure 3(a).

The performance results for the finite difference algorithm presented in [9] are derived using the single node allocation WildFire configuration (2). Using the same setting, the results for the pseudospectral solver kernel are similar. There is a significant decrease of the time per iteration during the first 6-7 iterations, and then the curve levels out. The WildFire optimizations move/replicate data from the remote to the local node, and remote accesses become more and more rare. Hence, the time per iteration decreases to a steady-state. Further investigation shows that the amount of replicated pages is small. The number of pages migrated is large at the beginning but decreases over time. The same phenomena is also present for the single node allocation WildFire with balanced thread scheduling (4), but here the initial time per iteration is shorter. The reason could be that, when all processors on a single node are computing as in configuration



(a) Iteration times for the first iterations on different computer configurations.



(b) Iteration times for the single node allocation balanced configurations.

Fig. 3. Results for a 2048×2048 grid using 24 threads

(2), the bus is heavily loaded in this node, and the bandwidth available for page migration will be small. The page migration daemon will suffer from this, and the migrating pages will be unaccessable for a longer time.

Using all the threads in a single node for computations leads to large variation in iteration times, probably because activities of other users stall the computations. This is most apparent in the WildFire configurations with the default scheduling policy (2,3).

For the other configurations shown in Figure 3(a), the behavior is different. The first iteration takes longer time, but after this, a steady-state is immediately reached. Here, the relatively slow first iteration can be explained by cache effects. For the cc-NUMA configurations (6) and (7) the result is natural, since the adaptive optimizations are shut off. For the single node allocation balanced cc-NUMA configuration (6), one of the nodes perform exclusively remote accesses, leading to unbalanced execution times and a significantly larger time per iteration in steady-state.

The performance is almost the same for the thread-matched allocation WildFire (3) and cc-NUMA configurations (7). The memory is initially optimally placed for the first FFT, and in the matrix transpose a minimal amount of communication takes place. The WildFire optimizations are not activated, but it is also clear that they do not introduce any performance degradation.

Figure 3(b) shows an interesting, but not yet fully understood, result. Here, the single node allocation WildFire (4) configuration has been tested with different optimization strategies. With no migration and replication, the configuration is equivalent to the cc-NUMA case (6). The default setting is to enable both optimizations. Interestingly, the best results are achieved when only migration is enabled, and similar results have been observed also for a number of different problem sizes.

6 Speedup

Speedup results for a 2048×2048 grid are shown for a number of different configurations in Figure 4. The graphs show the average time per iteration when steady-state has been reached, c.f. Section 5. The results are normalized by the execution time of a single thread.

In general, the results are remarkably good. As mentioned before, the algorithm uses global operations, and involves heavy communication. The single node SMP (1) and the WildFire configurations using the standard scheduling policy (2,3) show very similar behavior up to about 14 threads. This is natural, since for these cases only one SMP node is involved in the computations. For 16 threads, the first SMP node is filled, and for the WildFire configurations (2,3), it is possible that one of the threads have been moved by the OS scheduler to the other (almost idle) SMP node. This is not possible for the SMP configuration (1), where the threads are bound to a single node. Again, the problem of computing on a filled SMP node results in a degradation of performance.

There is a short plateau in the speedup curve around 16 threads for the WildFire configurations (2,3). Here, the threads begin to be spawned on the other SMP node. For the balanced configurations (5,7), there is a more even growth in speedup as the number of threads is increased. The amount of communication causing remote accesses is constant, which should result in a smooth speedup curve. The communication results in that it is favorable to use the default thread scheduling, compared to spawning the threads in a balanced way on the two nodes.

The speedup is considerably smaller for the single node allocation cc-NUMA configuration (6) than for the other configurations. The reason is again that a large amount of remote accesses are being performed by threads in one of the nodes.

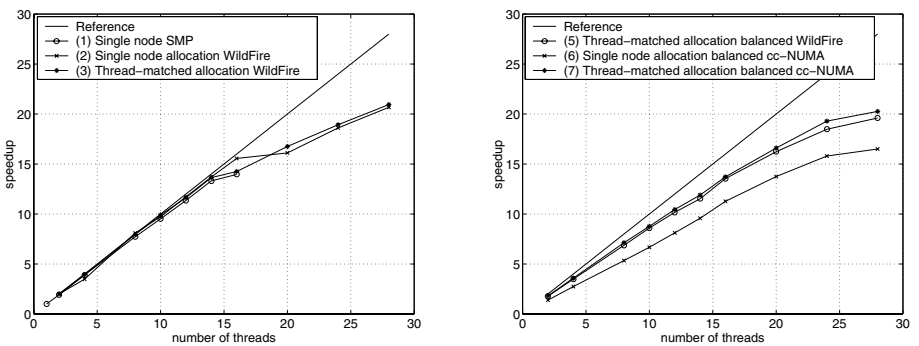


Fig. 4. The speedup for different configurations, compared to the execution time of a single thread. The grid size is 2048×2048

7 Impact of Problem Size

For the single node allocation WildFire configurations, we find that the number of iterations performed before steady-state is reached grows as the number of grid points is increased. This result is consistent with the results in [9].

As seen in Figure 5, there is no performance gain in using more than one SMP node for a small problem. However, as the problem size grows, the slope of the speedup curve once again approaches the ideal speedup ratio when the "WildFire-plateau" mentioned in Section 6 has been passed. For a very large grid, possibly more than two dimensions, the performance gain from using more than one SMP node will be large. Note that, for such problems, the memory of the additional SMP nodes will probably also be needed.

8 Conclusions

The pseudospectral solver kernel is an example of a non-trivial algorithm with heavy communication. The parallelization on the WildFire system is surprisingly successful; Using 28 OpenMP threads distributed over two SMP nodes, the speedup is approximately 21. For problems of interest in applications, the number of grid points will be even larger than used in the experiments, and the performance will probably be further improved.

The WildFire system will perform page migration if the initial distribution of data over the SMP nodes is not optimal. After some iterations, a steady-state is reached where no further migration occurs. For all configurations where the data is optimally distributed in steady-state, the difference in performance is very small. The WildFire migration optimization makes up for programming errors and/or deficiencies in the programming model, without introducing a performance loss when the data is optimally placed from the beginning. Note that, if the data is allocated on only one of the nodes and the optimizations are disabled, i.e. the code is executed in pure cc-NUMA mode (configuration 6), the performance is significantly reduced.

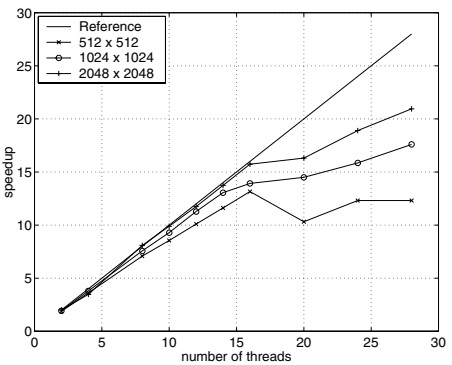


Fig. 5. Thread-matched WildFire configuration (3) speedups

The WildFire system using the default configuration exhibits a typical speedup behavior for a problem involving communication, e.g. the pseudospectral solver kernel: Until the number of threads is almost equal to the number of processors in an SMP node, the performance is identical to that of the SMP. When the number of threads is further increased, there is a short plateau in the speedup curve before it starts to grow again. If the problem is large enough, the slope of the speedup curve will again be close to optimal.

The speedup curve becomes smoother using a balanced thread scheduling policy. For the pseudospectral solver this implies a small performance loss when the number of threads is small, because of the large amount of communication over the WFI. However, distributing the threads in a balanced way over the SMP nodes might yield improved performance for a memory bound algorithm with a small amount of communication.

Note that the initial distribution of data has a large effect on the execution time if the convolution in the pseudospectral solver is only performed a small number of times. The goal of algorithm improvements, e.g. preconditioning, is to reduce the number of iterations in the computational scheme. It is important to make sure that the data is optimally distributed from the beginning if only a few iterations are required. There is currently discussion whether directives for data distribution should be included in OpenMP [8,1]. Data placement can be achieved without such directives on the WildFire system using the first-touch scheduling policy. In certain cases, when data can not be accessed within a parallel region, e.g. file I/O, data distribution directives would be useful.

References

1. Bircsak J. et al., *Extending OpenMP for NUMA Machines*, Proceedings of Supercomputing 2000. 610
2. Falsafi M., Wood D. A., *Reactive NUMA: A Design for Unifying S-COMA with CC-NUMA*, Proceedings of ACM/IEEE International Symposium on Computer Architecture 1997.
3. Frigo M., Johnson S. G., *FFTW: An Adaptive Software Architecture for the FFT*, 1998 ICASSP proceedings (vol. 3, p. 1381). 603
4. Hagersten E., Saulsbury A., Landin A, *Simple COMA Node Implementations*, Proceedings of Hawaii International Conference on System Science, 1994. 604
5. Hagersten E., Koster M., *WildFire: A Scalable Path for SMPs*, Proceedings of 5th International Symposium on High-Performance Architecture, 1999. 604
6. Lenoski D. E., Weber W. D., *Scalable shared-memory multiprocessing*, Morgan Kaufmann publishers, 1995. 603
7. van Loan C., *Computational Frameworks for the Fast Fourier Transform*, Society for Industrial and Applied Mathematics, Philadelphia, 1992. 603
8. Nikolopoulo D. S. et al., *Is Data Distribution Necessary in OpenMP?*, Proceedings of Supercomputing 2000. 610
9. Noordergraaf L., van der Pas R., *Performance Experiences on Sun's WildFire Prototype*, Proceedings of Supercomputing 99, 1999. 603, 606, 609
10. Fornberg F., *A Practical Guide to Pseudospectral Methods*, Cambridge University Press, 1998. 603