OpenMP versus MPI for PDE Solvers Based on Regular Sparse Numerical Operators [⋆]

Markus Nordén, Sverker Holmgren, and Michael Thuné

Uppsala University, Information Technology, Dept. of Scientific Computing, Box 120
SE-751 04 Uppsala, Sweden
{markusn, sverker, michael}@tdb.uu.se

Abstract. Two parallel programming models represented by OpenMP and MPI are compared for PDE solvers based on regular sparse numerical operators. As a typical representative of such an application, the Euler equations for fluid flow are considered.
The comparison of programming models is made with regard to UMA, NUMA, and self optimizing NUMA (NUMA-opt) computer architectures. By NUMA-opt, we mean NUMA systems extended with self optimizations algorithms, in order to reduce the non-uniformity of the memory access time.
The main conclusions of the study are: (1) that OpenMP is a viable alternative to MPI on UMA and NUMA-opt architectures, (2) that OpenMP is not competitive on NUMA platforms, unless special care is taken to get an initial data placement that matches the algorithm, and (3) that for OpenMP to be competitive in the NUMA-opt case, it is *not* necessary to extend the OpenMP model with additional data distribution directives, *nor* to include user-level access to the page migration library.

Keywords: OpenMP; MPI; UMA; NUMA; Optimization; PDE; Euler; Stencil

1 Introduction

Large scale simulations requiring high performance computers are of importance in many application areas. Often, as for example in fluid dynamics, electromagnetics, and acoustics, the simulations are based on PDE solvers, i.e., computer programs for the numerical solution of partial differential equations (PDE). In the present study, we consider parallel PDE solvers involving regular sparse operators. Such operators typically occur in the case of finite difference or finite volume methods on structured grids, either with explicit time-marching, or with *implicit* time-marching where the resulting algebraic systems are solved using an iterative method.

In the present article, we compare two programming models for PDE solver applications: the shared name space model and the message passing model. The

[⋆] The work presented here was carried out within the framework of the Parallel and Scientific Computing Institute. Funding was provided by Sun Microsystems and the Swedish Agency for Innovation Systems.

question we pose is: *will recent advances in computer architecture make the shared name space model competitive for simulations involving regular sparse numerical operators*? The tentative answer we arrive at is *"yes"*.

We also consider an additional issue, with regard to the shared name space model, viz., whether it requires explicit data distribution directives. Here, our experiments indicate that the answer is *"no"*.

The state-of-the-art for parallel programming of large scale parallel PDE solvers is to use the message passing model, which assumes a local name space in each processor. The existence of a default standard for this model, the Message Passing Interface (MPI) [5], has contributed to its strong position. However, even more important has been its ability to scale to large numbers of processors [6]. Moreover, many major massively parallel computer systems available on the market present, at least partly, a local name space view of the physically distributed memory, which corresponds to the assumptions of the message passing model.

However, with recent advances in SMP server technology has come a renewed and intensified interest in the shared name space programming model. There is now a de facto standard also for this model: OpenMP [1]. However, it is still an open question how well OpenMP will scale beyond the single SMP server case. Will OpenMP be a viable model also for clusters of SMPs, the kind of computer architecture that is currently dominating at the high end?

Clusters of SMPs typically provide non-uniform memory access (NUMA) to the processors. One approach to OpenMP programming in a NUMA setting is to extend the model with directives for data distribution, in the same spirit as in High-Performance Fortran. By directing the initial data placement explicitly, the same way as an MPI programmer would need to do, the user would be able to ensure that the different OpenMP threads get reasonably close to their data. This argument was put forward in, e.g., [4].

Another, more orthodox approach, was taken by Nikolopoulos et al. [2, 3], who claim that data distribution directives should *not* be added to OpenMP, since that would contradict fundamental design goals for the OpenMP standard, such as platform-independence and ease of programming. Moreover, they claim that directives are *not necessary* for performance, provided that the OpenMP implementation is supported by a dynamic page migration mechanism. They have developed a user-level page migration library, and demonstrate that the introduction of explicit calls to the page migration library into the OpenMP code enables OpenMP programs without distribution directives to execute with reasonable performance on both structured and non-structured scientific computing applications [2, 3].

Our contribution is in the same spirit, and goes a step further, in that we execute our experiments on a *self optimizing* NUMA (NUMA-opt) architecture, and rely exclusively on its built-in page migration and replication mechanisms. That is, no modifications are made to the original OpenMP code. The platform we use is the experimental Orange (previously known as Wildfire) architecture from Sun Microsystems [7]. It can be configured, in pure NUMA mode (no page migration and replication), and alternatively in various self optimization modes (only migration, only replication, or both). Moreover, each node of the system is an SMP, i.e., exhibits UMA behavior. Thus, using one and the same platform, we have been able to experiment with a variety of computer architecture types under *ceteris paribus* conditions.

Our Orange system consists of two 16-processor nodes, with UltraSparc II processors (i.e., not of the most recent generation), *but* with a sophisticated self optimization mechanism. Due to the latter, we claim that the Orange system can be regarded as a prototype for the *kind* of parallel computer platforms that we will see in the future. For that reason, we find it interesting to study the issue of OpenMP versus MPI for this particular platform.

The results of our study are in the same direction as those of Nikolopoulos et al. Actually, our results give even stronger support for OpenMP, since they do not presume user-level control of the page migration mechanisms. Moreover, our results are in agreement with those of Noordergraaf and van der Pas [10], who considered data distribution issues for the standard five-point stencil for the Laplace equation on a Sun Orange system. Our study can be regarded as a generalization of theirs to operators for non-scalar and non-linear PDEs, and also including a comparison to using a message passing programming model.

2 The Stencil Operator

The experiments reported below are based on a stencil which comes from a finite difference discretization of the nonlinear Euler equations in 3D, describing compressible flow. The application of this stencil operator at a certain grid point requires the value of the operand grid function at 13 grid points. This corresponds to 52 floating point numbers, since the grid function has four components.

Moreover, we assume that the physical structured grid is *curvilinear*, whereas the computations are carried out on a rectangular computational grid. This introduces the need for a mapping from the computational to the physical grid. Information about this mapping has to be available as well, and is stored in a 3×3-matrix that is unique for each grid point, which means nine more floating point numbers. In all, 61 floating point numbers have to be read from memory and approximately 250 arithmetic operations have to be performed at each grid point in every iteration.

The serial performance of our stencil implementation is close to 100 Mflop/s. This is in good agreement with the expectations according to the STREAM benchmark [9]. (See [11] for further discussion of the serial implementation.)

3 Computer System Configurations

On the Sun Orange computer system used here, there are a number of configurations to choose between. First of all, there are two self optimization mechanisms, page migration and replication, that can be turned on and off independently in the operating system.

Table 1. The computer system configurations used in the parallel experiments

Configuration	Thread scheduling	Memory allocation	Page migration	Page replication	Architecture type
Configuration 1	One node	One node	Off	Off	UMA
Configuration 2	Default	One node	On	On	NUMA-opt
Configuration 3	Default	Matching	On	On	NUMA-opt
Configuration 4	Balanced	One node	On	On	NUMA-opt
Configuration 5	Balanced	Matching	On	On	NUMA-opt
Configuration 6	Balanced	One node	Off	Off	NUMA
Configuration 7	Balanced	Matching	Off	Off	NUMA
Configuration 8	Balanced	One node	Off	On	NUMA-opt
Configuration 9	Balanced	One node	On	Off	NUMA-opt

Using these mechanisms it is possible to configure the Orange systems so as to represent a variety of architecture types. First, using only one server of the system gives a UMA architecture. Secondly, using both servers, but turning off the self optimization mechanisms, gives a NUMA. Finally, self optimizing NUMA systems with various degrees of self optimization can be studied by turning on the page migration and/or replication.

For the investigation of how OpenMP performs in different environments, we are interested in the variation not only in architecture type, but also in thread placement and data placement. This variation can also be achieved in the Orange system.

Table 1 summarizes the different Orange system configurations that were used in the parallel experiments reported below. With Configuration 1 we only use the resources in one node, i.e. we are running our program on an SMP server and the number of threads is limited to 16.

Configuration 2 and 3 both represent the default Orange system settings, with all self optimization turned on. The difference is that for the former all the data are initially located in one node, whereas for the latter they are distributed in a way that matches the threads already from the beginning.

For Configuration 4–9 the load of the SMP nodes is balanced, in that the threads are scheduled evenly between the nodes. The configurations differ, however, in the way that data are initialized and which self optimization mechanisms are used. Configuration 6 and 7, with no self optimization, represent pure NUMA systems.

We used the Sun Forte 6.2 (early access, update 2) compiler. It conforms to the OpenMP standard, with no additional data distribution directives. The configurations with matching data distribution were obtained by adding code for *initializing* data (according to the first-touch principle) in such a way that it was placed were it was most frequently needed. In this way, the same effect was obtained *as if* OpenMP had been extended with data distribution directives.

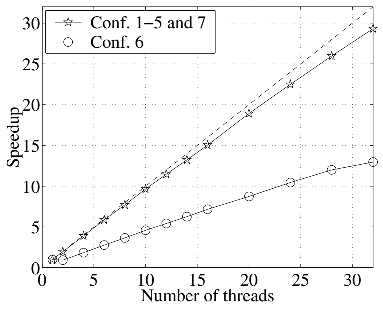


Fig. 1. Speedup per iteration for an OpenMP solver for the nonlinear Euler equations in 3D. The speedup was measured with respect to the time needed to carry out one iteration, once the system is fully adapted. Different curves correspond to different configurations of the parallel computer platform, see Table 1

4 OpenMP and the Effect of Self Optimization

The first series of experiments studies the performance of OpenMP for all the configurations discussed above. In particular we are interested in evaluating the effect of self optimization (in the computer system) on the performance of parallel programs based on OpenMP.

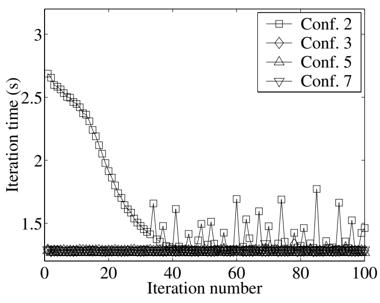
The reason for introducing self optimization is to allow the system to scale well even when more than one SMP node is used. Therefore, speedup is a good measure of how successful the optimization is. However, it is important to let the system adapt to the algorithm before any speedup measurements are done. Consequently, we have measured speedup with respect to the time needed to carry out one iteration once the system is fully adapted, i.e., after a number of iterations have been performed, see below.

The time-per-iteration speedup results for our application are shown in Figure 1. As can be seen, all configurations scale well, except for Configuration 6. The latter corresponds to a NUMA scenario, with no self optimization, and where data are distributed unfavorably. The configurations that rely on the self optimization of the system show identical speedup as the configurations that rely on hand-tuning. That is, after the initial adaption phase, the self optimization mechanisms introduce no further performance penalty.

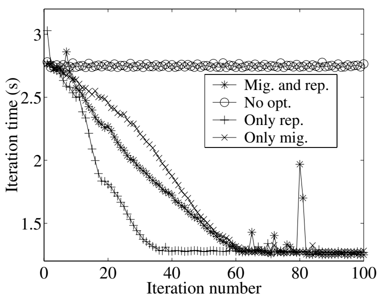
Next, we study how long it takes for the system to adapt. The adaption phase should only take a fraction of the total execution time of the program, otherwise the self optimization is not very useful.

We have measured the time per iteration for the different configurations. As can be seen in Figure 2, the adaption phase takes approximately 40–60 iterations for our program. This is fast enough, since the execution of a PDE solver usually involves several hundreds or even thousands of such iterations.

Consequently, the conclusion of the speedup and adaption-time experiments, taken together, is that the self optimization in the Orange system serves its purpose well for the kind of application we are considering.



(a) Different thread scheduling, data distribution and optimization



(b) Different self optimization mechanisms

Fig. 2. Time per iteration of our test program for different configurations of the computer system. The graphs refer to the 24 processor case, and similar results were obtained for other numbers of processors. After 40–60 iterations the system has adapted to the memory access pattern of the algorithm. This overhead is negligible in comparison to the typical total number of iterations

With regard to OpenMP, the conclusion is that additional data distribution directives are *not* needed for PDE solvers based on regular sparse numerical operators. This holds provided that the computer system is equipped with efficient self optimization algorithms, as is the case with the Orange system prototype used in our experiments.

In Table 2 we also show how many memory pages are migrated and replicated when using the different self optimization techniques. All the data were initialized to reside on one node, and the thread allocation is balanced over the nodes.

When using both migration and replication, approximately half of the data are migrated to the node where they are used. There are also some memory pages that are replicated, probably those that are used to store data on the border between the two halves of the grid, and therefore are accessed by threads in both nodes.

When using only replication, approximately half of the data are replicated to the other node and when only migration is allowed, half of the data are migrated.

With regard to the optimization modes, the conclusion of our experiments is that migration is sufficient for the kind of numerical operators we consider here. Combined replication and migration does not lead to faster adaption. The third alternative, replication only, gives slightly faster adaption, but at the expense of significant memory overhead.

Table 2. Iteration time and the number of pages that are migrated and replicated using the different optimization mechanisms. The threads are scheduled evenly between the nodes but data initially resides in just one of the nodes. In these experiments 24 threads were used and in all 164820 memory pages were used

Optimization	Iter. time	# Migrs	# Repls
Mig. and rep. (4)	1.249	79179	149
Only rep. (8)	1.267	N/A	79325
Only mig. (9)	1.264	79327	N/A

4.1 OpenMP versus MPI

We now proceed to comparing OpenMP and MPI. We have chosen to use balanced process/thread scheduling for both the MPI and OpenMP versions of the program. Every process of the MPI program has its own address space and therefore matching allocation is the only possibility. It should also be mentioned that since the processes have their own memory, there will be no normal memory pages that are shared by different processes. Consequently, a program that uses MPI will probably not benefit from migration or replication. This is also confirmed by experiments, where we do not see any effects of self optimization on the times for individual iterations as we did in the previous section.[1]

The experiments below also include a hybrid version of the program, which uses both MPI and OpenMP. There, we have chosen OpenMP for the parallelization within the SMP nodes and MPI for the communication between the nodes.

Now to the results. We have already seen that the OpenMP version scales well for both the UMA and self optimizing NUMA architectures. The results for Configuration 1 and the different NUMA-opt configurations were virtually identical. On the other hand, the speedup figures for the NUMA type of architecture (Configuration 6) were less satisfactory.

Turning to MPI, that programming model is not aware of the differences between the three architecture types, as discussed above. The same holds for the hybrid version, since it uses one MPI process for each node, and OpenMP threads within each such process. Consequently, the execution time was virtually the same for all MPI cases, regardless of architecture type, and similarly for the hybrid OpenMP/MPI cases.

[1] Accesses are made to the same address by different processes when we use MPI communication routines. This communication is normally performed so that one process writes the data to an address that is shared, and another process subsequently reads from the same address. Since the memory access pattern for that memory page is that one process always writes, after which another process reads, neither migration nor replication would improve performance. The reason is that in the case of migration the page would always be remotely located, as seen from one of the processes, and in the case of replication every new cache line that is to be read would result in a remote access since it has been updated on the other node since it was fetched last time.

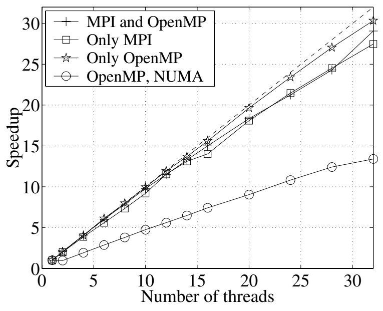


Fig. 3. Speedup for different versions of the non-linear Euler solver

Figure 3 shows the results in terms of time-per-iteration speedup. The MPI and hybrid versions give the same performance for all three architecture types. For OpenMP, the NUMA architecture gives significantly lower performance. However, for the UMA and NUMA-opt configurations OpenMP is competitive, and even somewhat better than the other alternatives.

Most likely, the MPI and hybrid versions scale less well because of time needed for buffering data during the communication. The reason why the hybrid version does not scale better than the pure MPI version is that even though there are fewer processes in the hybrid version than in the MPI version when the same number of processors are used, the amount of data to be exchanged is still the same for each process. The communication has to be done serially within the processes. Therefore, while one of the threads of an MPI process is busy sending or receiving data, the other threads of that process will be idle, waiting for the communication to take place.

5 Conclusions and Future Work

The main conclusions of our study are:

1. OpenMP is competitive with MPI on UMA and self optimizing NUMA architectures.
2. OpenMP is not competitive on pure (i.e., non-optimizing) NUMA platforms, unless special care is taken to get an initial data placement that matches the algorithm.
3. For OpenMP to be competitive in the self optimizing NUMA case, it is *not* necessary to extend the OpenMP model with additional data distribution directives, *nor* to include user-level access to the page migration library.

Clearly, there are limitations to the validity of these conclusions:

– They refer to applications involving regular sparse numerical operators. Such operators exhibit a very regular memory access pattern with only local communication, therefore it should be quite easy for the system to adapt to the algorithm. Further investigations are needed before the conclusions can be extended to applications with a highly irregular memory access pattern. However, the results reported by Nikolopoulos et al. [3], for OpenMP extended with user-level calls to a page migration library, give hope that our conclusions *will* in fact generalize to such applications.

– The Orange system used in this study has only two nodes. A massively parallel platform with a large number of nodes would be more challenging for the self optimization algorithms. We expect such systems to appear in the future, and we conjecture that it will be possible to generalize the migration and replication algorithms of the Orange system in such a way that the OpenMP model will be competitive on them as well. However, this remains to be proven.

For the near future, the really large scale computations will be carried out on massively parallel clusters of SMP (or heterogeneous clusters in a "grid" setting), with a local name space for each node. Then MPI, or the hybrid OpenMP/MPI model are the only alternatives. In fact, the results reported in [6] indicate that for some applications, the hybrid model is to prefer for large numbers of processors.

Our results for the NUMA case show that even for an SMP cluster equipped with an operating system that presents a shared name space view of the entire cluster, the MPI and hybrid models are still the best alternatives, in comparison with standard OpenMP. The data placement required for OpenMP to be competitive indicates the need for additional data distribution directives. On the other hand, since many platforms use the first-touch principle, an alternative way to achieve such data placement is via a straightforward initialization loop. Consequently, in our opinion, adding data distribution directives to OpenMP, in order to address the NUMA type of architecture, would not be worth its prize in terms of contradicting the design goals of OpenMP.

In the long term perspective, our results speak in favor of efficiently self optimizing NUMA systems, in combination with standard OpenMP, i.e., with no additional data distribution directives. As mentioned, we conjecture that self optimization algorithms of the type found in the Orange system can be generalized to work efficiently also for massively parallel NUMA systems. If this turns out to be true, programming those systems with standard OpenMP will allow for rapid implementation of portable parallel codes.

The work reported here is part of a larger project, "High-Performance Applications on Various Architectures" (HAVA). Other subprojects of HAVA consider other kinds of applications, for example pseudospectral solvers [8, 12], and solvers based on unstructured grids. The next phases of the present subproject will be to consider first a finite difference based *multi-grid solver* for the Euler equations, and then structured adaptive mesh refinement for the same application. The latter, in particular, provides additional challenges, for self optimization algorithms *as well as* for user-provided load balancing algorithms.

References

1. OpenMP Architechture Review Board. OpenMP Specifications.
2. D. S. Nikolopoulos et al. Is Data Distribution Necessary in OpenMP? In *SC2000 Proceedings*. IEEE, 2000.
3. D. S. Nikolopoulos et al. Scaling Irregular Parallel Codes with Minimal Programming Effort. In *SC2001 Proceedings*. IEEE, 2001.
4. J. Bircsak et al. Extending OpenMP for NUMA Machines. In *SC2000 Proceedings*. IEEE, 2000.
5. Message Passing Interface Forum. MPI Documents.
6. W. D. Gropp et al. High-performance parallel implicit CFD. *Parallel Computing*, 27:337–362, 2001.
7. E. Hagersten and M. Koster. WildFire: A Scalable Path for SMPs. In *Proceedings of the 5th IEEE Symposium on High-Performance Computer Architecture (HPCA)*, 1999.
8. S. Holmgren and D. Wallin. Performance of a pseudo-spectral PDE solver on a self-optimizing NUMA architecture. In *Proc. of Euro-Par 2001*. Springer-Verlag, 2001.
9. J. D. McCalpin. Sustainable Memory Bandwidth in Current High Performance Computers. Technical report, Advanced Systems Division, Silicon Graphics, Inc., 1995.
10. L. Noordergraaf and R. van der Pas. Performance Experiences on Sun's WildFire Prototype. In *SC99 Proceedings*. IEEE, 1999.
11. M. Nordén, S. Holmgren, and M. Thuné. OpenMP versus MPI for PDE solvers based on regular sparse numerical operators. Report in preparation.
12. D. Wallin. Performance of a high-accuracy PDE solver on a self-optimizing NUMA architecture. Master's thesis, Uppsala University School of Engineering, 2001. Report No. UPTEC F 01 017.