UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 224

Multithreaded PDE Solvers on Non-Uniform Memory Architectures

MARKUS NORDÉN

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2006

ISSN 1651-6214
ISBN 91-554-6656-7
urn:nbn:se:uu:diva-7149

Dissertation presented at Uppsala University to be publicly examined in 2446, 2, Uppsala, Friday, October 20, 2006 at 10:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract
Nordén, M. 2006. Multithreaded PDE Solvers on Non-Uniform Memory Architectures. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 224. i-xii+33 pp. Uppsala. ISBN 91-554-6656-7.

A trend in parallel computer architecture is that systems with a large shared memory are becoming more and more popular. A shared memory system can be either a uniform memory architecture (UMA) or a cache coherent non-uniform memory architecture (cc-NUMA).

In the present thesis, the performance of parallel PDE solvers on cc-NUMA computers is studied. In particular, we consider the shared namespace programming model, represented by OpenMP. Since the main memory is physically, or *geographically* distributed over several multi-processor nodes, the latency for local memory accesses is smaller than for remote accesses. Therefore, the *geographical locality* of the data becomes important.

The focus of the present thesis is to study *multithreaded* PDE solvers on cc-NUMA systems, in particular their memory access pattern with respect to geographical locality. The questions posed are: (1) How large is the influence on performance of the non-uniformity of the memory system? (2) How should a program be written in order to reduce this influence? (3) Is it possible to introduce optimizations in the computer system for this purpose?

The main conclusion is that geographical locality is important for performance on cc-NUMA systems. This is shown experimentally for a broad range of PDE solvers as well as theoretically using a model involving characteristics of computer systems and applications.

Geographical locality can be achieved through migration directives that are inserted by the programmer or — possibly in the future — automatically by the compiler. On some systems, it can also be accomplished by means of transparent, hardware initiated migration and replication. However, a necessary condition that must be fulfilled if migration is to be effective is that the memory access pattern must not be "speckled", i.e. as few threads as possible shall make accesses to each memory page.

We also conclude that OpenMP is competitive with MPI on cc-NUMA systems if care is taken to get a favourable data distribution.

Keywords: PDE solver, high-performance, NUMA, UMA, OpenMP, MPI, data migration, data replication, thread scheduling, data affinity

Markus Nordén, Department of Information Technology, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

© Markus Nordén 2006

ISSN 1651-6214
ISBN 91-554-6656-7
urn:nbn:se:uu:diva-7149 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-7149)

To Sara

List of Papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Markus Nordén, Sverker Holmgren, Michael Thuné: *OpenMP versus MPI for PDE Solvers Based on Regular Sparse Numerical Operators*. Future Generation Computer Systems 22(1-2): 194-203 (2006).

II Sverker Holmgren, Markus Nordén, Jarmo Rantakokko, Dan Wallin: *Performance of PDE Solvers on a Self-Optimizing NUMA Architecture*. Parallel Algorithms and Applications 17(4): 285-299 (2002).

III Henrik Löf, Markus Nordén, Sverker Holmgren: *Improving Geographical Locality of Data for Shared Memory Implementations of PDE Solvers*. Proceedings of International Conference on Computational Science 2004 II: 9-16 (2004).

IV Markus Nordén, Henrik Löf, Jarmo Rantakokko, Sverker Holmgren: *Geographical Locality and Dynamic Data Migration for OpenMP Implementations of Adaptive PDE Solvers*. Technical Report 2006-038, Department of Information Technology, Uppsala University (2006). To appear in proceedings of International Workshop on OpenMP 2006.

V Markus Nordén: *Performance Modelling for Parallel PDE Solvers on cc-NUMA Systems*. Technical Report 2006-041, Department of Information Technology, Uppsala University (2006).

Comments on my participation

The work with this thesis have given me insight into a range of representative numerical methods, implementation aspects, computer architecture and experimental studies.

I	I was responsible for the implementation of the CFD solver and the experiments.
II	I was responsible for the finite difference solver. This is the OpenMP version of the CFD solver in paper I and some results appear in both papers. However, in paper I we compared two different programming models (OpenMP and MPI) for the same PDE solver (a finite difference solver) whereas in paper II we use only one programming model (OpenMP) but compare the performance of three PDE solvers based on different numerical methods (finite difference methods, unstructured finite volume methods and pseudospectral methods).
III	I was responsible for the implementation of the industrial type multi-grid solver and the experiments regarding it.
IV	I was responsible for most of the implementation of the application studied. The experiments were performed jointly by myself and Henrik Löf.
V	I am the sole author of this paper.

In all papers experiments were designed, results analyzed and conclusions drawn jointly by all authors.

Acknowledgements

I wish to take the opportunity to thank some people for making my time writing this thesis so enjoyable.

First of all I would like to thank my supervisors Michael Thuné and Sverker Holmgren for offering me a PhD student position, for lots of inspiring discussions, for valuable feedback on everything I have written and for guiding me towards this thesis.

I would like to thank my co-authors of the papers in this thesis; Henrik Löf, Jarmo Rantakokko, Dan Wallin, Malik Silva, and Richard Wait.

I also want to thank Erik Berg, Erik Hagersten and Henrik Johansson for valuable discussions and for analyzing the cache behaviour of my programs with different simulators.

Thanks to all my colleagues, in particular Tom Smedsaas and Malin Ljungberg, at the Department of Scientific Computing for making it a wonderful workplace, with a spirit of warmth and friendliness that has helped me get through all days when nothing seemed to go right. Thank you also for bringing me on climbing expeditions, dance courses, Vasaloppet and other social activities.

Thanks to my parents for all your support and for showing interest in my work.

Tank you, Sara, for your love and patience!

This work has partially been funded by Sun Microsystems, Inc. and the Parallel and Scientific Computing Institute (PSCI), Sweden.

Contents

1 Introduction 1
2 cc-NUMA Systems 3
  2.1 The Sun WildFire System 3
  2.2 The Sun Fire 15000 System 4
3 Optimization of Geographical Locality 7
4 Model Equations 9
  4.1 The Euler equations 9
  4.2 The advection equation 9
5 PDE solvers studied 11
  5.1 The Finite Difference CFD solver 11
  5.2 The Industrial CFD Solver 11
  5.3 The Adaptive PDE Solver 12
6 Results 15
  6.1 Paper I 15
  6.2 Paper II 17
  6.3 Paper III 18
  6.4 Paper IV 21
  6.5 Additional results not incuded in paper IV 21
  6.6 Paper V 23
7 Conclusions 27
A Sammanfattning på svenska 29
Bibliography 31

1. Introduction

Computational methods have become a central tool in science and engineering. Many important phenomena in nature are modeled by partial differential equations (PDEs). For example, such equations describe flow of fluids and propagation of electromagnetic and sound waves. Also, PDEs arise in many other application areas ranging from chemistry to economics. The equations are often impossible to solve analytically and numerical solution methods must be employed.

In realistic settings, the numerical solution of PDEs requires large scale computations. The computations are usually carried out on parallel computers. During the last decade, parallel computers with cache coherent non-uniform memory architectures, cc-NUMA [8], have gained in popularity. Such systems have a logically shared memory system. This allows for a shared namespace programming model such as e.g. OpenMP [9, 30]. However, for the full realization of the potential of cc-NUMA systems, it is essential that numerical algorithm design and implementation take full advantage of the underlying system architecture.

Improving the locality of data is one of the most important optimizations to achieve good performance. In modern computer systems, *temporal* and *spatial locality* of data accesses is exploited by introducing a memory hierarchy with several levels of cache memories. For cc-NUMA systems, an additional form of locality also has to be taken into account. In such systems the main memory is physically, or *geographically* distributed over several multi-processor nodes. The access time for local memory is smaller than the time required to access remote memory, and the *geographical locality* of the data influences the performance of applications.

Recently, chip multi-processors (CMPs) [1, 23] have entered the scene. When used in large systems, CMPs will introduce phenomena on the cache level that are similar to what we see in conventional cc-NUMA systems. Knowledge about parallel implementation aspects will therefore play an increasingly important role in the future.

The focus of the present thesis is to study *multithreaded* PDE solvers on cc-NUMA systems, in particular their memory access pattern with respect to geographical locality. The main questions we pose are:

- How large is the influence on performance of the non-uniformity of the memory system?
- How should a program be written in order to reduce this influence?

- Is it possible to introduce optimizations in the computer system for this purpose?

We address these questions in the following way: In paper I we study a multithreaded PDE solver on a cc-NUMA system with built-in self optimization. We also make a comparison with a message passing version of the same program. In paper II we compare three different parallel PDE solvers implemented with OpenMP on the same system. In paper III we perform similar experiments with more advanced PDE solvers on another cc-NUMA system of a later generation. In paper IV we continue in the same spirit and study an adaptive PDE solver with a dynamic memory access pattern. For such access patterns it is a challenging task to achieve geographical locality. In paper V, finally, we develop a theoretical model of the memory performance of a PDE solver running on a NUMA-system.

2. cc-NUMA Systems

Examples of cc-NUMA systems are e.g. Sun WildFire [14], Sun Fire 15000 [6], SGI Origin [25], SGI Altix [35], Compaq Alpha Server GS-series [7] and Hewlett Packard Superdome [17].

The non-unitormity of a cc-NUMA system can be quantified by the NUMA-ratio, which is defined as the quotient of the latencies for remote and local memory. Currently, the NUMA-ratio for commonly used large cc-NUMA servers ranges from 2 to 6 [6, 13, 24, 25, 31]. If the NUMA-ratio is large, improving the geographical locality of data may lead to large performance improvements. Therefore, on a cc-NUMA system, an application could be optimized by explicitly placing data close to the CPU that is most likely to access it. This has been recognized by many researchers, and the study of geographical placement of data in cc-NUMA systems has been an active research area for some years [2–5, 27, 29, 33]. Also, some techniques aimed at improving geographical locality are commonly employed today, both in the design of computer systems and in the design of algorithms and codes.

In order to ease the burden on the programmer, different forms of optimization may be supported by the system. For example, on Sun WildFire and SGI Origin systems, memory pages accessed by a processor in a remote node are registered by hardware counters and may be migrated to the node where they are used by a page migration daemon.

On Sun Fire 15000 there is no automatic migration, but in later releases of Solaris 9 the user can make a system call that advises the operating system to migrate a memory page to the node from which it is accessed the next time [32]. A similar feature is also available on the Compaq Alpha Server GS-series [2].

The experiments presented in this thesis have been performed on two different cc-NUMA systems: a Sun WildFire prototype system (sometimes also referred to as Sun Orange) [14, 16], and a Sun Fire 15000 (SF15k) [6] system, which is a commercial cc-NUMA computer.

2.1 The Sun WildFire System

The Sun WildFire architecture [14] was developed with the intention to find an alternative to symmetric multiprocessors (SMPs) that made it possible to build larger shared memory systems. The system can be viewed as a cc-

NUMA computer 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 in each node.

The experiments on a WildFire system presented in this thesis were performed on the two-node WildFire system at the Department of Information Technology, Uppsala University. Each node has 16 processors (250 MHz UltraSPARC-II with 4 MB L2 cache) and 4 GB memory. Logically, there is no difference between accessing local and remote memory, even though the access time varies: 310 ns for local and 1700 ns for remote memory. Coherence between all the 32 caches is maintained in hardware, which creates an illusion of a system with 8 GB shared memory. The processors are by now quite old and slow, but the self-optimization features built into the system are still highly interesting. We regard the WildFire system as a prototype for a kind of parallel computer architecture of the future, implying that the results we present are relevant.

On the WildFire system, a software daemon detects pages which have been placed in the wrong node and migrates them to the correct node. It also detects which pages are used by threads in both nodes and replicates them, hereby avoiding ping-pong effects. The cache coherence protocol maintains 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) [15]. The maximum number of replicated pages as well as other parameters in the page migration and CMR algorithms may be altered by modifying system parameters.

2.2 The Sun Fire 15000 System

The Sun Fire 15000 (SF15k) system [6] is a commercially available cc-NUMA system. A SF15k system consists of up to 18 nodes (CPU/Memory boards), each with 4 processors and up to 32 GB memory. In addition to that, the system can be equipped with up to 17 “MaxCPU boards”, each with 2 processors. In all, a SF15k system can thus have up to 106 CPUs and 576 GB memory. Within a node, the access time to local main memory is uniform. The nodes are connected via a crossbar interconnect, forming a cc-NUMA system. The NUMA-ratio is only approximately 2, which is small compared to other commercial cc-NUMA systems available today.

The experiments on a SF15k system presented in this thesis were performed on the 48 CPU (12 nodes with 4 CPUs each) SF15k system at the Department of Information Technology, Uppsala University. The CPUs are 900 MHz UltraSPARC-IIICu and each node contains 4 GB of local memory. In paper III and IV, a dedicated partition consisting of four nodes was used. The scheduling of threads between the nodes was in paper III controlled by creating So-

laris processor sets and binding the threads to them and in paper IV by means of an environment variable.

The experiments in paper III were performed using the 12/03-beta release of Solaris 9, where a static first-touch page placement strategy is introduced. Support for dynamic, application-initiated migration of data is available in the form of a migrate-on-next-touch feature [32]. Migration is initiated by a system call, where the operating system is advised to reset the mapping of virtual to physical addresses for a given range of memory pages, and to redo the first-touch data placement. The effect is that a page will be migrated if a thread in another node performs the next access to it. The same features are also used in paper IV, where a later release of Solaris was available.

3. Optimization of Geographical Locality

As mentioned above, the performance of a program on a cc-NUMA system depends on the geographical locality of data. It is therefore important that the data distribution matches the memory access pattern of the program.

However, when trying to optimize for geographical locality, one should be aware of that a program can have different execution phases with different memory access patterns. For example, an industrial PDE solver parallelized with OpenMP often has a serial initialization phase that is followed by a parallel region where the computations takes place and often also a serial post processing phase. Furthermore, the parallel phase can sometimes consist of several subphases with different memory access pattern, e.g. if an adaptive method is used.

For a well implemented parallel PDE solver, we can expect that the extent of the serial execution phases is small compared to the parallel phase. Since the computationally most intensive execution phase of a parallel PDE solver is the parallel region, it is most important that the data distribution of the program matches the memory access pattern of this phase.

Several solutions have been proposed that address the issue of optimization of geographical locality. One commonly used approach employs the first touch principle, where a memory page normally is placed in the same node as the processor that accesses it first. This is done automatically by the operating system without any intervention from the user. However, it is a static strategy and it is often insufficient in order to achieve good performance for parallel PDE solvers if the data structures are initialized serially. Furthermore, the first touch principle assume that the memory access pattern does not change dynamically.

One way to remedy some of the deficiencies of the first touch principle is to let the user specify the data distribution instead. This was done in High Performance Fortran [18] and it is also proposed as an extension to OpenMP by Bircsak et. al. in [2]. However, this would contradict the design goals of OpenMP. Moreover, it may be impossible to predict the memory access pattern if, for example, it depends on the data itself. This is for example usually the case for PDE solvers that use unstructured grids.

Another approach is to introduce *user initiated* redistribution of data, as opposed to *user controlled* distribution or redistribution. This is available in the form of a migrate-on-next-touch system call on SF15k and the Compaq

Alpha Server GS-series. An interval of virtual addresses is given by the user and the corresponding memory pages are marked as candidates for migration. Their virtual to physical address mapping is reset and upon the next access to a page it is migrated to the node from which it is accessed.

A similar approach is proposed by Nikolopoulos et. al. [27, 28]. However, in those papers a software library is described that, by means of hardware counters, records the memory access pattern for a set of virtual addresses during e.g. the first iteration of a parallel PDE solver. A redistribution of data then takes places that is based on the recorded memory access pattern.

Another solution is to build an adaptive system, where data automatically is redistributed dynamically when the memory access pattern of a program changes. This kind of self optimization is available on the Sun WildFire system [14] and SGI Origin systems [25]. On those systems hardware counters are used to keep track of which memory pages are frequently accessed by processors in a remote node and a software daemon migrates those pages.

4. Model Equations

4.1 The Euler equations

As mentioned earlier, PDEs arise in many different application areas. One important application area is fluid dynamics, where flow of air, water and other substances is studied. Such flow can be described by the Euler equations

$$\frac{\partial W}{\partial t} = \frac{\partial}{\partial x} f(W) + \frac{\partial}{\partial y} g(W) + \frac{\partial}{\partial z} h(W)$$

$$W = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}$$

$$f = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(\rho E + p) \end{pmatrix}, g = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ v(\rho E + p) \end{pmatrix}, h = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ w(\rho E + p) \end{pmatrix}$$

where, ρ is the density; u, v, and w are the velocity components, corresponding to the three space directions x, y, and z, respectively; p is the pressure, and E is the total energy per unit of mass.

The equations above are the *conservative formulation* of the Euler equations, but there is also a corresponding *non-conservative* formulation. Furthermore, the equations can sometimes be simplified. If, for example, the flow is assumed to be isentropic, one of the equations can be eliminated.

The study of fluid flow is very important in e.g. the aircraft industry and the use of computers to solve fluid dynamics problems is usually referred to as computational fluid dynamics (CFD).

4.2 The advection equation

Another, much simpler, equation is

$$u_t = \alpha u_x + \beta u_y$$

which describes advection or convection, i.e. how something (heat, pollution, etc.) is transported by the flow of some fluid (water, air, etc.) if diffusion can be neglected.

Thanks to its simplicity, the advection equation is often used as a model problem when studying advanced numerical methods.

5. PDE solvers studied

5.1 The Finite Difference CFD solver

The experiments reported in paper I and II are based on a stencil which comes from a finite difference discretization of the Euler equations in 3D where isentropic flow is assumed.

In the PDE solver, the stencil is repeatedly applied to a grid function. This is computationally very intensive and the solver will spend most of its time in the routine responsible for applying the stencil to the grid function. Boundary conditions, etc., could affect the performance slightly, e.g. by introducing some load imbalance. However, we are convinced that the over all memory characteristics will not be disturbed significantly by other computations. Therefore we have chosen to focus on the stencil application routine in the present study.

Our stencil is based on second order centered finite difference approximations of the first order spatial derivatives in the Euler equations. In addition, it includes artificial viscosity, via the addition of second order centered finite difference approximations of fourth order spatial derivatives. Thus, with respect to its center point, the stencil extends two grid points to each side, in each direction, and consequently includes 13 grid points, including the center point. For each of these points, the stencil has a corresponding “coefficient”. These coefficients are 4×4-matrices, where each matrix element is a function of the grid point coordinates, the time variable, and the grid function to which the stencil is to be applied. For that reason, it is not possible to compute and store the stencil once and for all. Instead, the stencil coefficients for a certain grid point at a certain time, and for a certain grid function, are computed when they are needed. In all, approximately 250 arithmetic operations have to be performed at each grid point every time the stencil is applied to a grid function.

5.2 The Industrial CFD Solver

We have also implemented a more advanced CFD solver kernel, that uses the same kind of methods that are used in industrial codes. It solves the conservative form of the compressible Euler equations and is based on the methodology presented by Jameson [21].

The equations are discretized by means of a finite volume method. The flux vectors are split, so that the discretization takes into account the directions of propagation in the solution (so called flux-splitting, see e.g. [19]).

The resulting discrete scheme is a set of nonlinear algebraic equations, which has to be solved in order to advance the solution from time level n to time level $n+1$. Applying Newton's method to these nonlinear equations yields a linear system for each Newton iteration.

These linear systems are solved iteratively, via a Symmetric Gauss-Seidel (SGS) method. We modify the scheme presented in [21] slightly, and introduce a red-black-ordered version of the SGS method. This will result in a much more efficient parallelization than the original SGS algorithm. At this point, we also let the time step go to infinity, since we are solving for steady state.

Finally, the resulting SGS-Newton iteration is embedded into a multigrid scheme for convergence acceleration. We use the full multigrid approach described by Jameson [20].

For our experiments, we use a code that only implements the computational kernel described above. The boundary conditions for the numerical scheme are ignored. In an application code, local boundary conditions are used, which have a similar structure to the upwind difference scheme used in the interior of the domain.

5.3 The Adaptive PDE Solver

The adaptive PDE solver in paper IV solves the advection equation with periodic boundary conditions on a square. The initial solution is a Gaussian pulse, as illustrated in Figure 5.1. As time evolves the pulse moves diagonally out through one of the corners of the domain and comes back in from the opposite corner without changing shape. The PDE is discretized by a second-order accurate finite difference method in space and the classical fourth order Runge-Kutta method in time. We use a block-wise adaptive grid [12], i.e. a structured cartesian grid that is divided into a fixed user-defined number of blocks where the resolution in each block is determined by an estimate of the error in the block, as shown in Figure 5.1. As a simple error estimate in the adaptation criterion we use the maximum value of the solution in a block. In a real-life application, a more sophisticated error estimate e.g. based on applying the spatial difference operator on a coarse and a fine block discretization would be used [11]. However, this would not affect the parallel performance much, and the conclusions drawn from the experiments presented in paper IV would remain valid. If the error estimate exceeds a threshold somewhere in a block, the resolution of the grid is refined with a factor two in the entire block. On the contrary, if the error is small enough, the grid in the block is coarsened with a factor two.

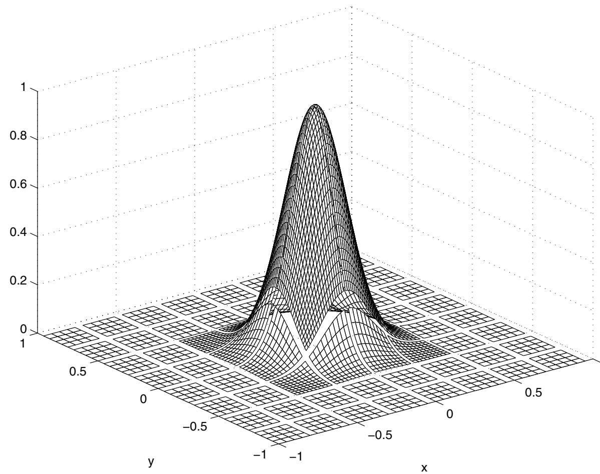

Figure 5.1: Initial solution for the adaptive PDE solver. The computational grid is divided into blocks and the resolution of each block is adapted in order to resolve the solution with sufficient accuracy. For visibility reasons, the figure shows a grid with fewer blocks and lower resolution than what is used in the computations.

The code is written in Fortran 90 and parallelized using OpenMP. The parallelization is coarse grained over entire blocks, i.e. each thread is responsible for a set of blocks. The blocks have two layers of ghost cells which are updated by reading data from the neighboring blocks. When the grid resolution changes in any of the blocks, the entire grid block structure is repartitioned using the Jostle diffusion algorithm [34] and the work partitioning between the threads is changed accordingly.

6. Results

6.1 Paper I

In paper I, we compare two programming models for PDE solver applications: the shared name space model and the message passing model. The question we pose is: *Will recent advances in computer architecture make the shared name space model competitive for simulations involving regular sparse numerical operators*?

We also consider an additional issue, whether the shared name space model requires explicit data distribution directives.

The computer system we use is the WildFire system described in section 2.1. 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., it exhibits uniform memory access (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. The different computer system configurations are presented in more detail in Table 6.1.

As a model application, we use the finite difference CFD solver described in section 5.1. We have three different versions of it; an OpenMP version, an MPI version [26] and a hybrid version that is parallelized with OpenMP within the SMP nodes and uses MPI for the communication between the nodes.

In the comparison between OpenMP and MPI, we carry out 100 iterations in each experiment. Then, we measure time-per-iteration speedup for the final 40 iterations. That is, the adaptation phase for the NUMA-opt configurations is not included in the comparison. The reason is that if the adaptation phase was included, the speedup results would depend on the number of iterations. The adaptation phase (approximately 60 iterations) will typically be amortized over thousands of iterations.

Figure 6.1 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(2) (NUMA with matching data placement), and NUMA-opt(2) (NUMA with page migration as well as replication switched on) configuration is competitive, and even somewhat better than the other alternatives.

For the OpenMP version of our PDE solver, we also study the effects of the self optimization mechanisms available on the WildFire system. As reference cases we use two NUMA configurations, NUMA(1) with all data in

Table 6.1: *Some of the computer system configurations used in the parallel experiments. The threads are evenly distributed over the nodes, except for the UMA configuration, where all threads are scheduled on one node. In the text, the "configuration labels" are used for referring to the various cases.*

Configuration label	Memory allocation	Page mig.	Page repl.	Architecture type
UMA	One node	Off	Off	UMA
NUMA(1)	One node	Off	Off	NUMA
NUMA(2)	Matching	Off	Off	NUMA
NUMA-opt(2)	One node	On	On	NUMA-opt
NUMA-opt(3)	One node	Off	On	NUMA-opt
NUMA-opt(4)	One node	On	Off	NUMA-opt

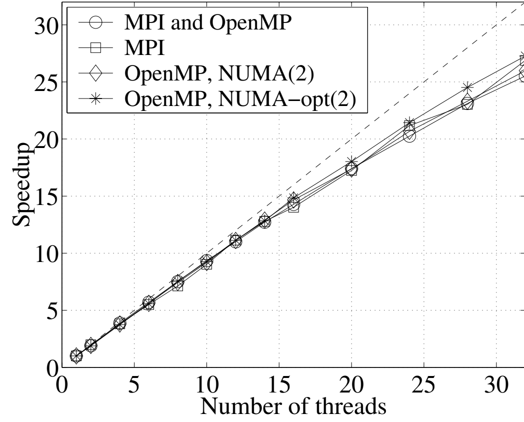

Figure 6.1: Speedup per iteration for different versions of the finite difference CFD solver.

one node and NUMA(2) with data distributed in order to match the threads. The times per iteration for the reference cases are ca. 2.75 s/iteration and ca. 1.25 s/iteration, respectively. In Figure 6.2 we see that the self optimizing NUMA-opt configurations begin at the NUMA(1) level and that all of them converge to the NUMA(2) level. The latter is reached after ca. 40 iterations if only replication is used, otherwise after ca. 60 iterations.

The main conclusions in paper I 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.

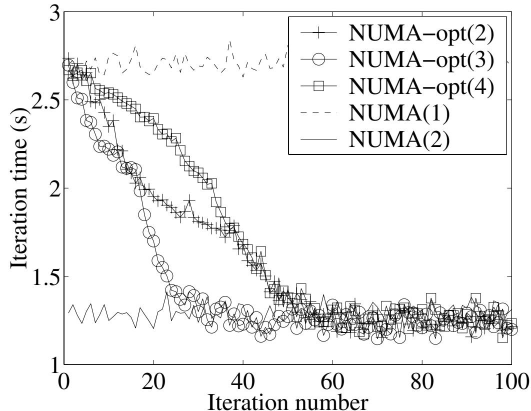

Figure 6.2: Time per iteration for the finite difference CFD solver when both migration and replication is switched on (NUMA-opt(2)), with only replication (NUMA-opt(3)) and with only migration (NUMA-opt(4)). The curves refer to the 24 processor case, and similar results were obtained for other numbers 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.

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.

These conclusions are valid for regular sparse numerical operators with a very regular memory access pattern and only local communication.

6.2 Paper II

In paper II we go one step further than we did in paper I and compare OpenMP implementations of three different PDE solvers on the WildFire system. In addition to the solver from paper I, we also study an unstructured finite volume solver with a highly irregular memory access pattern and a pseudospectral solver with a structured memory access pattern, but global dependencies. An important question is: *what differences, from a cc-NUMA performance point of view, are there between different numerical methods*?

Our results show that optimal data placement is important for the performance for all solvers. When relying on the self-optimization mechanisms for data distribution, the performance after the migration and replication process has reached steady-state is the same as what we achieve if we place data optimally at the beginning of the execution using hand tuning. For reasonably large PDE problems, we find that the time needed by the system to redistribute data is negligible compared to the total solve time. This shows that, for the applications studied, the self-optimizing features are successful, and

shared memory codes without explicit data distribution directives yield performance on par with codes with hand tuned data placement.

We do, however, see that the performance of the pseudospectral solver and the unstructured finite volume solver suffer from running on a NUMA system even with an optimal data distribution. The unstructured solver also scales poorly when running on one SMP node. This is not a shortcoming of the self-optimization mechanisms, but due to properties of the algorithms themselves.

For the pseudospectral solver the reason is that it involves global communication. Despite that the solution matrix is transposed in order to avoid unnecessary remote accesses we still see the effects of the slower accesses.

The unstructured finite volume solver, on the other hand, has a "speckled" memory access pattern, that makes it difficult to associate one home node of a memory page even though the communication pattern is of the nearest neighbour kind. To remedy this would require algorithmic modifications like graph partitioning or bandwidth minimization of the coefficient matrix used in the solver.

6.3 Paper III

In paper III we examine how different data placement schemes affect the performance of two important classes of parallel codes from large-scale scientific computing. The main issues considered are:

1. What impact does geographical locality have on the performance for the type of algorithms studied?
2. How does the performance of an application-initiated data migration strategy based on a migrate-on-next-touch feature compare to that of standard data placement schemes?
3. How should such an application-initiated migration strategy be invoked?

Most experiments presented in paper III are performed using the SF15k system described in section 2.2, but some experiments are also performed using the Sun WildFire system described in section 2.1.

To evaluate different methods for improving geographical locality, we study the performance of four large-scale PDE solvers, employing both structured and unstructured grids. We have noted that benchmark codes often solve simplified PDE problems using standardized algorithms, which may result in different performance result than for kernels from advanced application codes. We therefore perform experiments using kernels from industrial applications as well as standard benchmark codes from the NAS NPB3.0-OMP suite [22].

The following PDE solvers are studied:

The NAS-MG benchmark Solves the Poisson equation on a structured grid using a multi-grid method.

An industrial-type CFD solver The solver described in section 5.2.

The NAS-CG benchmark A sparse system of equations with an unstructured coefficient matrix is solved using the conjugate gradient method.

An industrial CEM solver A computational electromagnetics (CEM) code, solving the Maxwell equations around an aircraft using an unstructured grid [10].

We focus on isolating the effects of the placement of data, and do not attempt to assess the much more complex issue of the scalability of the codes. First, we measure the execution time for our codes using four threads on a single node. In this case, the first touch policy results in that all application data is allocated locally, and the memory access time is uniform, i.e. UMA mode. We then compare the UMA timings to the corresponding execution times when executing the codes in cc-NUMA mode, running a single thread on each of the four nodes. Here, three different data placement schemes are used:

Serial initialization (SI) The main data arrays are initialized in a serial section of the code, resulting in that the pages containing the arrays are allocated on a single node. This is a common situation when application codes are naively parallelized using OpenMP.

Parallel initialization (PI) The main data arrays are initialized in pre-iteration loops within the main parallel region. The first-touch allocation results in that the pages containing the arrays are distributed over the four nodes.

Serial initialization + Migration (SI+MIG) The main arrays are initialized using serial initialization. A migrate-on-next-touch directive is inserted at the first iteration in the algorithm. This results in that the pages containing the arrays will be migrated according to the scheduling of threads used for the main iteration loop.

From our results, it is clear that the geographical locality of data does affect the performance for all four codes. For the industrial CFD solver, both the PI and the SI+MIG strategy are very successful and the performance is effectively the same as for the UMA case. This code has a very good cache hit rate, and the remote accesses produced for the SI strategy do not reduce the performance very much either. For the NAS-MG code the smaller cache hit ratio results in that this code is more sensitive to geographical misplacement of data. Also, NAS-MG contains more synchronization primitives than the industrial CFD solver, which possibly affects the performance when executing in cc-NUMA mode.

For the NAS-CG code, the relatively dense matrix results in reasonable cache hit ratio and the effect of geographical misplacement is not very large. For the industrial CEM solver, the matrix is much sparser, and the caches are

not so well utilized as for NAS-CG. In the industrial CEM solver code, the sparse matrix data are read from a file, and it is not possible to include pre-iteration loops in this code. There is a significant difference in performance between the unmodified code (SI) and the version where a migrate-on-next-touch directive is added (SI+MIG).

We have also studied the overhead for the dynamic migration in the SI+MIG scheme. For NAS-MG and the industrial CEM solver, the migration overhead is significant compared to the time required for one iteration. If the SI+MIG scheme is used for these codes, approximately five iterations must be performed before there is any gain from migrating the data. For the NAS-CG code the relative overhead is smaller, and migration is beneficial if two iterations are performed. For the industrial CFD solver, the relative overhead from migration is small, and using the SI+MIG scheme even the first iteration is faster than if the data is kept on a single node.

The main conclusions in paper III are:

1. Geographical locality is important for the performance of applications on cc-NUMA systems.
2. Application-initiated migration leads to better performance than parallel initialization in almost all cases examined, and in some cases the performance is close to that obtained if all threads and their data reside on the same node.
3. Ideally, optimization of geographical locality should be fully automatic, without any need for intervention from the user. In that case, the compiler has to identify execution phases with different memory access pattern at compile time or they must be detected by the system during run time. As mentioned earlier, the most apparent, and probably also most important, alteration of memory access pattern occurs at the transition from serial to parallel execution at the beginning of a parallel region. Therefore, we believe that it should be worth trying to let OpenMP compilers on cc-NUMA systems automatically insert a migrate-on-next-touch for all shared data at the beginning of a parallel region.

Clearly, there are limitations to the validity of these conclusions. The domain of the SF15k system used in this study has only four nodes with four CPUs each. A system with a large number of nodes would be more challenging for the optimization techniques. We expect the overhead for migration to constitute a larger amount of the total execution time on such a system. However, this remains to be investigated.

Finally, we also performed a qualitative comparison of the results from the SF15k system to results obtained on the WildFire system. As mentioned in section 2.1, this system supports fully transparent adaptive memory placement optimization in the hardware. Our results show that this is also a viable alternative. The conclusions from paper I are thus valid also for more advanced numerical methods.

6.4 Paper IV

In paper IV we investigate the impact of geographical locality for the adaptive PDE solver presented in section 5.3. This application has a dynamic access pattern and no single data distribution can give optimal geographical locality throughout the entire execution of the program.

The dynamic access pattern implies that a system needs to support some kind of runtime data redistribution to maintain geographical locality. On the other hand, the overhead of such redistribution could possibly outweigh the benefits of the improved geographical locality.

In paper IV we use the SF15k system described in section 2.2 and our results show that the impact of geographical locality is large even though the NUMA-ratio of the system is only two.

Our results also show that we can improve geographical locality and overall performance significantly by means of data migration. This implies that the overhead of migration is low.

However, our experiments were performed using only four threads. The overhead from page migration will probably increase with the number of nodes and CPUs. On the other hand, the NUMA-ratio is probably also higher for larger systems, exaggerating the impact of geographical locality. Scalability of migration for adaptive PDE solvers on larger systems remains to be studied.

Furthermore, the refinement patterns of adaptive PDE solvers often vary a lot depending on the physics of the problem studied. If data migration is to be used in a more general setting, the frequency of invoking migration must be tuned to match the refinement patterns of the problem being studied. If large amounts of data need to be migrated we may have to reduce the number of migration calls to amortize the overhead over several time steps. However, for the model problem studied in paper IV, using a diffusion type partitioner resulted in fairly low need for data migration.

6.5 Additional results not incuded in paper IV

A question that was not addressed in paper IV, due to page limitations, is how migration shall be invoked. In paper III we noticed that migration could be invoked in three different ways:

- User *controlled* migration, i.e. that the user specifies exactly where each memory page should reside.
- User *initiated* migration, i.e. that the user specifies that a certain memory page should be migrated but relies on a next-touch mechanism to determine where
- Transparent migration, i.e. the need for migration is detected and performed automatically, without any intervention from the user.

Table 6.2: *Execution time for different migration strategies.*

Data selectivity	Affinity strategy	Execution time (h)
Some	Explicit	3.96
All	Explicit	4.37
Some	On touch	4.49
All	On touch	4.51
UMA		4.09
NUMA		6.64

For adaptive PDE solvers, with a dynamically changing memory access pattern, the situation is more complex than for a PDE solver with a static access pattern.

First of all, for an adaptive PDE solver, with a grid consisting of several smaller blocks, the next-touch strategy may have a problem with “false sharing” of memory pages. If two different threads are responsible for two adjacent blocks, the “wrong” thread may be the first to touch a memory page and hence “stealing” it despite it will only perform a minority of all accesses to that page.

Furthermore, there is no single data distribution that is optimal throughout the entire execution, but there may be a need for migration every time the computational mesh is adapted. However, at such situations, a majority of the data may already reside in a favourable part of the physical memory. Thus, the overhead for migration may be reduced by only migrating data that is unfavourably located.

In addition to the experiments presented in paper IV, we have also made a complementary study, where we compare four different migration strategies for the adaptive PDE solver. They differ in two ways:

Data selectivity, i.e. if *all* shared data is migrated at every adaptation occation or only *some* memory pages (those belonging to blocks that have been moved to another thread) are migrated.

Affinity strategy, i.e. if it is stated *explicitly* where a memory page should migrate or if that is decided by the next-touch strategy.

Results from the study, in terms of execution times, are found in Table 6.2. Furthermore, execution times for the UMA and NUMA cases in paper IV are also included.

The two migration strategies with explicit affinity are examples of user controlled migration, as defined in the beginning of this section, and the two with affinity on touch are user initiated. Since there is no support for transparent

migration on the SF15k system, we were not able to study any such migration strategy.

The results show that the execution time is reduced significantly for all four migration strategies in comparison with the NUMA case. The main conclusion is thus that the decision to migrate at all is more important than which migration strategy to use.

There is, however, a difference in execution time also between the different migration strategies. When migrating all data, the overhead for migration is higher than when migrating only misplaced data and when using the on touch affinity, the data distribution will in general not become optimal due to stealing of falsely shared pages.

From a programmer perspective, user initiated migration — i.e. using on touch affinity — of all data is most appealing since it is the least demanding to implement, but it also gives the least improvement of execution time.

Improving only the data selectivity, i.e. migrating misplaced data with on touch affinity, is a somewhat contradictory strategy. On one hand it does not require that the programmer specifies where to migrate a memory page but on the other hand he is assumed to know which memory pages are misplaced. This strategy is only marginally better than migrating all data with on touch affinity.

Migrating all data explicitly can be seen as a compromise, where the programmer is relieved of the burden of knowing anything about the previous data distribution but where he has to specify the new data distribution. In this case, the data distribution will be optimal but there will be unnecessary overhead from trying to migrate memory pages that are already favourably placed.

Finally, fully user controlled migration, i.e. migrating only misplaced data with explicit affinity specifications, gives an optimal data distribution with minimal migration overhead. As expected, this migration strategy gives the best improvment of execution time, but it is not indisputable that the improvment is worth all the extra implementation work compared to user initiated migration of all data.

6.6 Paper V

When running a parallel PDE solver on a NUMA-system, performance is degraded by remote memory accesses. In paper V we set up a detailed model of the memory performance of a PDE solver running on a NUMA-system. Due to the complexity of modern computers, such a detailed model inevitably is mainly of theoretical interest. Therefore we also develop a simplified model, which is an approximation of the detailed model that mimics the important features of the memory system using only a few parameters. This allows us to describe different kinds of NUMA-systems and PDE solvers conveniently and we can see what influence different parameters have.

Assuming that all threads make approximately the same number of memory accesses and have the same cache miss ratio, the simplified model tells us that the aggregate memory latency of a program is

$$T_M(\rho, N, \theta) \approx \rho \frac{N}{|\Theta|}(L\tau_L + (1-L)\tau_R)$$

where ρ is the average cache miss ratio, N the total number of memory accesses, $|\theta|$ the number of threads used, τ_L the latency for local memory and τ_R the latency for remote memory. The quotient of the latencies to remote and local memory is referred to as the NUMA-ratio, $\nu = \frac{\tau_R}{\tau_L}$.

The remaining parameter, L, is a measure of the aggregate locality of the program. It is the the maximum of the localities of all threads, L_θ, i.e.

$$L = \min_{\theta \in \Theta} L_\theta$$

where L_θ is the quotient of the number of local memory accesses and the total number of memory accesses made by a thread.

We notice that some remote accesses are inevitable due to communication and introduce a measure of the optimal locality, L^*, that corresponds to a data distribution that minimizes remote accesses for a certain program.

Different actors are involved in optimization for efficient utilization of computer systems. Computer architects try to minimize the NUMA-ratio when building new systems, application developers try to write programs with as high optimal locality as possible for the chosen numerical method and developers of memory management systems try to find algorithms for data distribution that yield optimal locality.

Comparing the aggregate memory latency when running a program with locality $L > L^*$ on a NUMA system with running the same program on a UMA system where $L = 1$, the overhead can be attributed to two sources: (1) the non-uniformity of the hardware and (2) the suboptimal data distribution. We quantify those contributions with the NUMA-factor, $F_\nu = L^* + \nu - L^*\nu$, and the locality factor $F_L = \frac{L+\nu-L\nu}{L^*+\nu-L^*\nu}$, and have

$$T_M(\rho, N, L) = F_L F_\nu T_M(\rho, N, 1)$$

Analyzing typical implementations of four classes of numerical methods, we have made estimates of their optimal locality that are presented in Table 6.3. The optimal locality is highly implementation dependent and we have assumed that measures have been taken in order to avoid "speckled" memory access patterns. Unordered local methods are, however, exceptions, since their optimal locality can be improved by reordering the grid points, transforming them into ordered local methods.

We have shown that it is possible to make a data distribution yielding almost ideal geographical locality for PDE solvers using ordered local methods (e.g. finite difference methods and finite volume methods on structured grids). This

Table 6.3: *Optimal locality for the different categories of numerical methods. The parameter b is the number of floating point numbers in one cache line, δ the number of spatial dimensions of the computational domain and $|\tilde{\Lambda}|$ is the number of locality groups used for the computations.*

Method category	Optimal locality
Ordered local	$L^* \approx 1$
Unordered local	$L^* \approx \frac{2+\frac{b}{\vert\tilde{\Lambda}\vert}}{2+b}$
Semiglobal	$L^* \approx \frac{\delta-1}{\delta} + \frac{1}{\delta\vert\tilde{\Lambda}\vert}$
Global	$L^* \approx \frac{1}{\vert\tilde{\Lambda}\vert}$

makes them very unsensitive to high NUMA-ratios and allows them to scale well on virtually any NUMA-system.

For PDE solvers using unordered local methods (e.g. finite element methods and finite volume methods on unstructured grids), semiglobal methods (e.g. pseudospectral methods) or global methods (e.g. in N-body dynamics), the optimal data distribution gives more moderate geographical locality. This make them more sensitive to high NUMA-ratios and therefore they show limited scalability beyond a single locality group.

In spite of this, our results show that for realistic NUMA-ratios, the potential performance gain of replacing a naive data distribution (i.e. the first touch principle for a serial initialization phase) by an optimal data distribution can be considerable.

7. Conclusions

The main conclusion of the present thesis is that geographical locality is important for the performance of multithreaded PDE solvers on cc-NUMA systems. We have shown this theoretically as well as through experiments for several kinds of numerical methods.

Geographical locality can be achieved through data distribution directives where the programmer explicitly controls the memory placement. This usually gives optimal geographical locality, but it can be difficult to implement — in particular for programs with dynamic memory access patterns or where the access pattern is not known a priori. Furthermore, it contradicts the design goals of OpenMP.

A more appealing alternative is to optimize geographical locality through migrate-on-next-touch directives. Such directives can be introduced either by the programmer or possibly automatically by future compilers. Our results show that such directives inserted by the programmer are effective.

Yet another approach is to provide self optimization in the system in terms of hardware initiated dynamic migration and replication of memory pages. This was done in the WildFire system and our experiments have shown that it can be very useful, at least for applications with static memory access patterns. Whether it is effective also for applications with dynamic memory access patterns remains to be studied.

We also conclude that OpenMP is competitive with MPI on UMA systems and that OpenMP can be competitive with MPI also on cc-NUMA systems if care is taken to get a data distribution that matches the memory access pattern.

In the future, most PDE solvers — also for problems on a more modest scale — will probably be parallel as CMPs make their way into every desktop computer. Knowledge about parallel implementation aspects, in particular with respect to memory utilization, will then play an increasingly important role.

A. Sammanfattning på svenska

Datorberäkningar har blivit ett allt viktigare verktyg inom teknik och naturvetenskap. Många viktiga fenomen i naturen kan beskrivas med hjälp av partiella differentialekvationer, PDE:er. Till exempel kan de beskriva flöden av vätskor och gaser eller utbredning av ljudvågor och elektromagnetiska vågor. PDE:er förekommer även inom många andra sammanhang, bland annat inom så vitt skilda områden som kemi och ekonomi. Eftersom ekvationerna oftast är omöjliga att lösa exakt får man förlita sig på numeriska metoder och datorberäkingar.

I realistiska sammanhang kräver numerisk lösning av PDE:er stora beräkningar. Dessa genomförs vanligen på parallelldatorer. På senare år har parallelldatorer med så kallat icke-uniformt delat minne, NUMA (Non-Uniform Memory Architecture), blivit alltmer populära eftersom de är enklare och billigare att bygga än parallelldatorer med uniformt minne, UMA (Uniform Memory Architecture). I en sådan dator är minnet logiskt sett gemensamt för alla processorer, vilket innebär att man kan skriva sina program med någon delat-minne-modell, t.ex. OpenMP. Men för att utnyttja datorn till fullo måste hänsyn tas till hur den är konstruerad.

God lokalitet hos data är det viktigaste för att uppnå god prestanda. På moderna datorer drar man nytta av temporal och spatial lokalitet genom att använda s.k. cache-minnen. På NUMA-datorer måste man även ta hänsyn till en tredje slags lokalitet. I sådana datorer är minnet fysiskt, eller geografiskt, distribuerat över flera processornoder. Att använda lokalt minne är effektivare än att använda ickelokalt minne och den geografiska lokaliteten hos data kommer att påverka programmets prestanda.

I den här avhandlingen studeras parallella PDE-lösare på NUMA-datorer, framför allt hur de använder minnet med avseende på geografisk lokalitet. Frågorna jag ställer är:

- Hur mycket påverkas prestanda av att minnessystemet är ivke-uniformt?
- Hur ska ett program skrivas för att minimera denna påverkan?
- Är det möjligt att införa automatisk självoptimering i datorerna för att minska effekterna av att minnessystemet är icke-uniformt?

För att svara på frågorna har jag genomfört experiment med tre olika PDE-lösare:

- Ett program som använder en finit differensmetod för att lösa Eulers ekvationer. Eftersom det är en relativt enkel numerisk metod är programmet minnesintensivt.

- Ett program som löser Eulers ekvationer med hjälp av en finit volymmetod och ett antal modifieringar som är vanliga i den typ av program som används inom flygindustrin. Eftersom det är en avancerad numerisk metod är programmet relativt beräkningsintensivt.
- Ett program som löser transportekvationen med hjälp av en finit differensmetod på ett adaptivt beräkningsnät. Eftersom nätet anpassar sig efter lösningen förändras arbetsbördan i olika delar av nätet med tiden och arbetsfördelningen måste anpassas därefter. Därför är det svårare att uppnå god geografisk lokalitet.

De NUMA-datorer jag använt är ”Sun WildFire” och ”Sun Fire 15000”. Den förra är en prototyp med stor skillnad i åtkomsttid mellan lokalt och ickelokalt minnne, men med automatisk självoptimering, medan den senare är en serietillverkad dator där skillnaden mellan de olika delarna av minnet är mindre.

Den viktigaste slutsatsen i avhandlingen är att geografisk lokalitet är viktigt för att uppnå god effektivitet när man använder NUMA-datorer för att lösa PDE:er. Det har jag visat dels genom teoretiska beräkningar med en modell av minnesutnyttjandet, dels genom de ovannämna experimenten med flera olika slags PDE-lösare på två NUMA-system.

Geografisk lokalitet kan uppnås genom direktiv i programmet, där man uttryckligen talar om vad som ska lagras var i minnet. Sådana direktiv ger i allmänhet mycket bra resultat, men det kan vara svårt att avgöra vilket som är bästa sättet att använda minnet på – i synnerhet för program där sättet att använda minnet förändras under tiden de körs eller där det inte är känt i förväg hur minnet kommer att användas. Dessutom motsäger införandet av direktiv av detta slag delvis idéerna bakom OpenMP.

Ett mer tilltalande alternativ är direktiv där minnessystemet uppmanas att fördela om data i minnet baserat på hur data används i programmet. Sådana direktiv kan införas antingen av programmeraren eller möjligen automatiskt av framtida kompilatorer. Våra resultat visar att sådana direktiv som införts av programmeraren är verkningsfulla.

Ett tredje alternativ är att låta datorerna tillhandahålla automatisk självoptimering, där data fördelas om i minnet för att förbättra den geografiska lokaliteten utan man behöver införa några direktiv om det. Sådan självoptimering fanns det i WildFire-systemet. Våra experiment med det har visat att det fungerar mycket bra, åtminstone för program där sättet att använda minnet inte förändras under tiden de körs.

Jag har också visat att OpenMP står sig väl i konkurensen med MPI på NUMA-datorer om man anstränger sig för att utnyttja minnet på ett lämpligt sätt.

Bibliography

[1] L.A. Barroso et al. Piranha: A scalable architecture based on single-chip multiprocessing. In *Proceedings of the International Symposium on Computer Architecture*. ACM Press, 2000.

[2] J. Bircsak et al. Extending OpenMP for NUMA machines. *Scientific Programming*, 8(8):163–181, 2000.

[3] T. Brecht. On the importance of parallel application placement in NUMA multiprocessors. In *Proceedings of the Fourth Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*. USENIX, Sept 1993.

[4] J.M. Bull and C. Johnson. Data distribution, migration and replication on a cc-NUMA architecture. In *Proceedings of the Fourth European Workshop on OpenMP*. OpenMP Architecture Review Board, 2002.

[5] R. Chandra et al. Data distribution support on distributed shared memory multiprocessors. In *Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation*, pages 334–345. ACM Press, 1997.

[6] A. Charlesworth. The Sun Fireplane system interconnect. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, page 7. ACM Press, 2001.

[7] Compaq. *Compaq AlphaServer GS Series Architecture White Paper*, 2000.

[8] D. Culler, J.P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1998.

[9] L. Dagum and R. Menon. OpenMP: An industry-standard API for shared-memory programming. *Computational Science and Engineering, IEEE*, 5(1):46–55, Jan.-March 1998.

[10] F. Edelvik. *Hybrid Solvers for the Maxwell Equations in Time-Domain*. Doctoral thesis, Mathematics and Computer Science, Department of Information Technology, University of Uppsala, 2002.

[11] L. Ferm and P. Lötsetdt. Space-time adaptive solutions of first order PDEs. *Journal of Scientific Computing*, 26(1):83–110, Jan. 2006.

[12] L. Ferm and P. Lotstedt. Adaptive error control for steady state solutions of inviscid flow. *SIAM Journal on Scientific Computing*, 23(5):1777–1798, 2002.

[13] K. Gharachorloo et al. Architecture and design of AlphaServer GS320. *ACM SIGPLAN Notices*, 35(11):13–24, 2000.

[14] E. Hagersten and M. Koster. WildFire: A scalable path for SMPs. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*. IEEE Computer Society, 1999.

[15] E. Hagersten, A. Saulsbury, and A. Landin. Simple COMA node implementations. In *Proceedings of Hawaii International Conference on System Science*. IEEE Computer Society, 1994.

[16] J.L. Hennessy and D.A. Patterson. *Computer Architecture; A Quantative Approach.* Morgan Kaufmann, 3rd edition, 2003.

[17] Hewlett Packard. *HP Superdome White Paper*, September 2000.

[18] High Performance Fortran Forum. *High Performance Fortran Language Specification*, 1993.

[19] C. Hirsch. *Numerical computation of internal and external flows. Vol. 2,* Computational methods for inviscid and viscous flows. John Wiley & Sons, 1990.

[20] A. Jameson. Solution of the Euler equations for two-dimensional, transonic flow by a multigrid method. *Appl. Math. Comp.*, 13:327–356, 1983.

[21] A. Jameson and D. A. Caughey. How many steps are required to solve the Euler equations of steady, compressible flow: In search of a fast solution algorithm. In *15th Computational Fluid Dynamics Conference*. AIAA, 2001.

[22] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. NAS Technical Report NAS-99-011, NASA Ames Research Center, 1999.

[23] P. Kongetira et al. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro*, 25(2):21–29, 2005.

[24] K. Krewell. EV7 stresses memory bandwidth. *Microprocessor Report*, March 2003.

[25] J. Laudon and D. Lenoski. The SGI Origin: a cc-NUMA highly scalable server. In *Proceedings of the 24th annual international symposium on Computer Architecture*, pages 241–251. ACM Press, 1997.

[26] Message Passing Interface Forum, `http://www.mpi-forum.org/`. *MPI Documents*.

[27] D.S. Nikolopoulos et al. A transparent runtime data distribution engine for OpenMP. *Scientific Programming*, 8(8):143–162, 2000.

[28] D.S. Nikolopoulos, C.D. Polychronopoulos, and E. Ayguade. Scaling irregular parallel codes with minimal programming effort. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, page 16. ACM Press, 2001.

[29] L. Noordergraaf and R. van der Pas. Performance experiences on Sun's WildFire prototype. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 38. ACM Press, 1999.

[30] OpenMP Architechture Review Board, `http://www.openmp.org/`. *OpenMP Specifications*.

[31] Silicon Graphics. *SGI Origin 3000 series*, 2000.

[32] Sun Microsystems. *Solaris Memory Placement Optimization and Sun Fire Servers*, January 2003.

[33] B. Verghese et al. Operating system support for improving data locality on cc-NUMA compute servers. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 279–289. ACM Press, 1996.

[34] C. Walshaw, M. Cross, and M.G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Parallel Distributed Computing*, 47(2):102–108, 1997.

[35] M. Woodacre et al. *The SGI Altix 3000 Global Shared-Memory Architecture*. SGI, 2003.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 224

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2006

Distribution: publications.uu.se
urn:nbn:se:uu:diva-7149