Experiences with the OpenMP Parallelization of DROPS, a Navier-Stokes Solver written in C++

Christian Terboven[1], Alexander Spiegel[1], Dieter an Mey[1], Sven Gross[2], and Volker Reichelt[2]

[1] Center for Computing and Communication, RWTH Aachen University, Germany
`{Terboven|Spiegel|anMey}@rz.rwth-aachen.de`,
WWW home page: `http://www.rz.rwth-aachen.de`

[2] Institut für Geometrie und Praktische Mathematik, RWTH Aachen University, Germany
`{Gross|Reichelt}@igpm.rwth-aachen.de`,
WWW home page: `http://www.igpm.rwth-aachen.de`

Abstract. In order to speed-up the Navier-Stokes solver DROPS, which is developed at the IGPM (Institut für Geometrie und Praktische Mathematik) at the RWTH Aachen University, the most compute intense parts have been tuned and parallelized using OpenMP. The combination of the employed template programming techniques of the C++ programming language and the OpenMP parallelization approach caused problems with many C++ compilers, and the performance of the parallel version did not meet the expectations.

1 Introduction

The Navier-Stokes solver DROPS [2] is developed at the IGPM (Institut für Geometrie und Praktische Mathematik) at the RWTH Aachen University, as part of an interdisciplinary project (SFB 540: Model-based Experimental Analysis of Kinetic Phenomena in Fluid Multi-phase Reactive Systems [1]) where complicated flow phenomena are investigated.

The object-oriented programming paradigm offers a high flexibility and elegance of the program code facilitating development and investigation of numerical algorithms. Template programming techniques and the C++ Standard Template Library (STL) are heavily used.

In cooperation with the Center for Computing and Communication of the RWTH Aachen University detailed runtime analysis of the code has been carried out and the computationally dominant program parts have been tuned and parallelized with OpenMP.

The UltraSPARC IV- and Opteron-based Sun Fire SMP-Clusters have been the prime target platforms, but other architectures have been investigated, too.

It turned out that the sophisticated usage of template programming in combination with OpenMP is quite demanding for many C++ compilers. We observed a high variation in performance and many compiler failures.

In chapter 2 the DROPS package is described briefly. In chapter 3 we take a look at the performance of the original and the tuned serial code versions. In chapter 4 we describe the OpenMP parallelization. The performance of the OpenMP version is discussed in chapter 5. Chapter 6 contains a summary of our findings.

2 The DROPS multi-phase Navier-Stokes solver

The aim of the ongoing development of the DROPS software package is to build an efficient software tool for the numerical simulation of three-dimensional incompressible multi-phase flows. More specifically, we want to support the modeling of complex physical phenomena like the behavior of the phase interface of liquid drops, mass transfer between drops and a surrounding fluid, or the coupling of fluid dynamics with heat transport in a laminar falling film by numerical simulation. Although quite a few packages in the field of CFD already exist, a black-box solver for such complicated flow problems is not yet available.

From the scientific computing point of view it is of interest to develop a code that combines the efficiency and robustness of modern numerical techniques, such as adaptive grids and iterative solvers, with the flexibility required for the modeling of complex physical phenomena.

For the simulation of two-phase flows we implemented a levelset technique for capturing the phase interface. The advantage of this method is that it mainly adds a scalar PDE to the Navier-Stokes system and therefore fits nicely into the CFD framework. But still, the coupling of the phase interface with the Navier-Stokes equations adds one layer of complexity.

The main building blocks of the solution method are the following:

Grid generation and grid refinement. Only tetrahedral grids without hanging nodes are used. The grids form a hierarchy of stable triangulations to enable the use of multi-grid solvers. The hierarchical approach also facilitates the coarsening of the grids.

Time discretization. For the stable time discretization of the instationary problems an implicit Fractional Step scheme is used.

Spatial discretization. The LBB-stable Taylor-Hood Finite Element pair (P_2-P_1) is used for the spatial discretization of the Navier-Stokes equations. For the level set equation the quadratic P_2 element is used.

Iterative solution methods. We decouple the Navier-Stokes-Level-Set system via a fixed point iteration which is also used to linearize the Navier-Stokes equations. The linearized equations which are of Stokes-type are treated by a Schur complement (inexact Uzawa) technique. The resulting convection-diffusion problems are solved by Krylov-subspace or multi-grid methods.

The several layers of nesting in the solvers (from the outer fixed point iteration down to the convection-diffusion-type solvers) induced by the structure of the mathematical models require fast inner-most solvers as well as fast discretization methods since many linear systems have to be regenerated in each time step. Apart from the numerical building blocks, software engineering aspects such as the choice of suitable data structures in order to decouple the grid generation and finite element discretization (using a grid based data handling) as much as possible from the iterative solution methods (which use a sparse matrix format) are of main importance for performance reasons.

The code is programmed in C++ and uses several attractive facilities offered by this programming language.

3 Portability and Performance of the Serial Program Version

3.1 Platforms

The main development platform of the IGPM is a standard PC running Linux using the popular GNU C++ compiler [3]. Because this compiler does not support OpenMP, we had to look for adequate C++ compilers supporting OpenMP on our target platforms.

Table 1 lists compilers and platforms which we considered for our tuning and parallelization work. It also introduces abbreviations for each combination of hardware, operating system and compiler, which will be referred to in the remainder of the paper.

The programming techniques employed in the DROPS package (Templates, STL) caused quite some portability problems due to lacking standard conformance of the compilers (see table 3). The code had to be patched for most compilers.

From the early experiences gathered by benchmarking the original serial program and because of the good availability of the corresponding hardware we concentrated on the OPT+icc and USIV+guide platforms for the development of the OpenMP version. We used XEON+icc (running Windows) for verification of the OpenMP codes using the Intel ThreadChecker.

3.2 Runtime profile

The runtime analysis (USIV+guide platform) shows that assembling the stiffness matrices (SETUP) costs about 52% of the total runtime, whereas the PCG-method including the sparse-matrix-vector-multiplication costs about 21% and the GMRES-method about 23%. Together with the utility routine LINCOMB these parts of the code account for 99% of the total runtime. All these parts have been considered for tuning and for parallelization with OpenMP.

It must be pointed out that the runtime profile heavily depends on the number of mesh refinements and on the current timesteps. In the beginning of a program run the PCG-algorithm and the matrix-vector-multiplication take about 65% of the runtime, but because the number of iterations for the solution of the linear equation systems shrinks over time, the assembly of the stiffness matrices is getting more and more dominant. Therefore we restarted the program after

code	**machine**	**processor**	**operating system**	**compiler**
XEON+gcc333 XEON+gcc343	standard PC	2x Intel Xeon 2.66 GHz	Fedora-Linux	GNU C++ V3.3.3 GNU C++ V3.4.3
XEON+icc81	standard PC	2x Intel Xeon 2.66 GHz	Fedora-Linux and Windows 2003	Intel C++ V8.1
XEON+pgi60	standard PC	2x Intel Xeon 2.66 GHz	Fedora-Linux	PGI C++ V6.0-1
XEON+vs2005	standard PC	2x Intel Xeon 2.66 GHz	Windows 2003	MS Visual Studio 2005 beta 2
OPT+gcc333 OPT+gcc333X	Sun Fire V40z	4x AMD Opteron 2.2 GHz	Fedora-Linux	GNU C++ V3.3.3 GNU C++ V3.3.3, 64bit
OPT+icc81 OPT+icc81X	Sun Fire V40z	4x AMD Opteron 2.2 GHz	Fedora-Linux	Intel C++ V8.1 Intel C++ V8.1, 64bit
OPT+pgi60 OPT+pgi60X	Sun Fire V40z	4x AMD Opteron 2.2 GHz	Fedora-Linux	PGI C++ V6.0-1 PGI C++ V6.0-1, 64bit
OPT+path20 OPT+path20X	Sun Fire V40z	4x AMD Opteron 2.2 GHz	Fedora-Linux	PathScale EKOpath 2.0 PathScale EKOpath 64bit
OPT+ss10	Sun Fire V40z	4x AMD Opteron 2.2 GHz	Solaris 10	SunStudio C++ V10
USIV+gcc331	Sun Fire E2900	12x UltraSPARC IV 1.2 GHz, dual core	Solaris 9	GNU C++ V3.3.1
USIV+ss10	Sun Fire E2900	12x UltraSPARC IV 1.2 GHz, dual core	Solaris 9	Sun Studio C++ V10
USIV+guide	Sun Fire E2900	12x UltraSPARC IV 1.2 GHz, dual core	Solaris 9	Intel-KSL Guidec++ V4.0 + Sun Studio 9
POW4+guide	IBM p690	16x Power4 1.7 GHz, dual core	AIX 5L V5.2	Intel-KSL Guidec++ V4.0
POW4+xlC60	IBM p690	16x Power4 1.7 GHz, dual core	AIX 5L V5.2	IBM Visual Age C++ V6.0
POW4+gcc343	IBM p690	16x Power4 1.7 GHz, dual core	AIX 5L V5.2	GNU C++ V3.3.3
IT2+icc81	SGI Altix 3700	128x Itanium 2 1.3 GHz	SGI ProPack Linux	Intel C++ V8.1

Table 1. Compilers and platforms

100 time steps and let it run for 10 time steps with 2 grid refinements for our comparisons.

3.3 Data Structures

In the DROPS package the Finite Element Method is implemented. This includes repeatedly setting up the stiffness matrices and then solving linear equation systems with PCG- and GMRES-methods.

Since the matrices arising from the discretization are sparse, an appropriate matrix storage format, the CRS (compressed row storage) format is used, in which only nonzero entries are stored. It contains an array *val* - which will be referred to later - for the values of the nonzero entries and two auxiliary integer arrays that define the position of the entries within the matrix.

The data structure is mainly a wrapper class around a `valarray<double>` object, a container of the C++ Standard Template Library (STL).

Unfortunately, the nice computational and storage properties of the CRS format are not for free. A disadvantage of this format is that insertion of a non-zero element into the matrix is rather expensive. Since this is unacceptable when building the matrix during the discretization step, a sparse matrix builder class has been designed with an intermediate storage format based on STL's `map` container that offers write access in logarithmic time for each element. After the assembly, the matrix is converted into the CRS format in the original version.

3.4 Serial Tuning Measures

On the Opteron systems the PCG-algorithm including a sparse-matrix-vector-multiplication and the preconditioner profits from manual prefetching. The performance gain of the matrix-vector-multiplication is 44% in average, and the speed-up of the preconditioner is 19% in average, depending on the addressing mode (64bit mode profits slightly more than 32bit mode).

As the setup of the stiffness matrix turned out to be quite expensive we reduced the usage of the `map` datatype. As long as the structure of the matrix does not change, we reuse the index vectors and only fill the matrix with new data values. This leads to a performance plus of 50% on the USIV+guide platform and about 57% on the OPT+icc platform. All other platforms benefit from this tuning measure as well.

Table 2 lists the results of performance measurements of the original serial version and the tuned serial version. Note that on the Opteron the 64bit addressing mode typically outperforms the 32bit mode, because in 64bit mode the Opteron offers more hardware registers and provides an ABI which allows for passing function parameters using these hardware registers. This outweights the fact that 64bit addresses take more cache space.

4 The OpenMP Approach

4.1 Assembly of the Stiffness Matrices

The matrix assembly could be completely parallelized, but it only scales well up to about 8 threads, because the overhead increases with the number of threads used (see table 4).

The routines for the assembly of the stiffness matrices typically contain loops like the following:

```
for (MultiGridCL::const_TriangTetraIteratorCL
   sit=_MG.GetTriangTetraBegin(lvl),
   send=_MG.GetTriangTetraEnd(lvl);
   sit != send; ++sit)
```

code	compiler options	runtime [s] original version	runtime [s] tuned version
XEON+gcc333	-O2 -march=pentium4	3694.9	1844.3
XEON+gcc343	-O2 -march=pentium4	2283.3	1780.7
XEON+icc81	-O3 -tpp7 -xN -ip	2643.3	1722.9
XEON+pgi60	-fast -tp piv	8680.1	5080.2
XEON+vs2005	compilation fails	n.a.	n.a.
OPT+gcc333	-O2 -march=opteron -m32	2923.3	1580.3
OPT+gcc333X	-O2 -march=opteron -m64	3090.9	1519.5
OPT+icc81	-O3 -ip -g	2516.9	1760.7
OPT+icc81X	-O3 -ip -g	2951.3	1521.2
OPT+pgi60	-fast -tp k8-32 -fastsse	6741.7	5372.9
OPT+pgi60X	-fast -tp k8-64 -fastsse	4755.1	3688.4
OPT+path20	-O3 -march=opteron -m32	2819.3	1673.1
OPT+path20X	-O3 -march=opteron -m64	2634.5	1512.3
OPT+ss10	-fast -features=no%except -xtarget=opteron	3657.8	2158.9
USIV+gcc331	-O2	9782.4	7845.4
USIV+ss10	-fast -xtarget=ultra3cu -xcache=64/32/4:8192/512/2	7749.9	5198.0
USIV+guide	-fast +K3 -xipo=2 -xtarget=ultra4 -xcache=64/32/4:8192/128/2 -lmtmalloc	7551.0	5335.0
POW4+guide	+K3 -backend -qhot -backend -O3 -backend -g [-bmaxdata:0x80000000]	5251.9	2819.4
POW4+xlC60	compilation fails	n.a.	n.a.
POW4+gcc343	-O2 -maix64 -mpowerpc64	3193.7	2326.0
IT2+icc81	-O3 -ip -g	9479.0	5182.8

Table 2. Platforms, compiler options and serial runtime of the original and the tuned versions. Note that we didn't have exclusive access to the Power4 and Itanium2 based systems for timing measurements.

Such a loop construct cannot be parallelized in OpenMP, because the loop iteration variable is not of type integer. Therefore the pointers of the iterators are stored in an array in an additional loop, so that afterwards a simpler loop running over the elements of this array can be parallelized.

Reducing the usage of the `map` STL datatype during the stiffness matrix setup as described in chapter 3 turned out to cause additional complexity and memory requirements in the parallel version. In the parallel version each thread fills a private temporary container consisting of one map per matrix row. The structure of the complete stiffness matrix has to be determined, which can be parallelized over the matrix rows. The master thread then allocates the `valarray` STL objects. Finally, the matrix rows are summed up in parallel.

If the structure of the stiffness matrix does not change, each thread fills a private temporary container consisting of one `valarray` of the same size as the array *val* of the final matrix.

This causes massive scalability problems for the guidec++-compiler. Its STL library obviously uses critical regions to be threadsafe. Furthermore the guidec++ employs an additional allocator for small objects which adds more overhead. Therefore we implemented a special allocator and linked to the Sun-specific memory allocation library *mtmalloc* which is tuned for multithreaded applications to overcome this problem.

4.2 The Linear Equation Solvers

In order to parallelize the PCG- and GMRES-method, matrix and vector operations, which beforehand had been implemented using operator overloading, had to be rewritten with C-style `for` loops with direct access to the structure elements. Thereby some synchronizations could be avoided and some parallelized `for`-loops could be merged.

The parallelized linear equation solvers including the sparse-matrix-vector-multiplication scale quite well, except for the intrinsic sequential structure of the Gauss-Seidel preconditioner which can only be partially parallelized. Rearranging the operations in a blocking scheme improves the scalability (`omp_block`) but still introduces additional organization and synchronization overhead.

A modified parallelizable preconditioner (`jac0`) was implemented which affects the numerical behavior. It leads to an increase in iterations to fulfill the convergence criterium. Nevertheless it leads to an overall improvement with four or more threads.

The straight-forward parallelization of the sparse matrix vector multiplication turned out to have a load imbalance. Obviously the nonzero elements are not equally distributed over the rows. The load balancing could be easily improved by setting the loop scheduling to `SCHEDULE(STATIC,128)`.

4.3 Compilers

Unfortunately not all of the available OpenMP-aware compilers were able to successfully compile the final OpenMP code version. Table 3 gives a survey of how successful the compilers have been.

Only the GNU C++ and the Pathscale C++ compilers were able to compile the DROPS code without any source modifications. Unfortunately the GNU C++ compiler does not support OpenMP, and the Pathscale C++ compiler currently does not support OpenMP in conjunction with some C++ constructs.

The Intel C++ compiler does not respect that a `valarray` is guaranteed to be filled with zero after construction. This is necessary for DROPS working correctly, so we changed the declaration by explicitly forcing a zero-filled construction.

In all cases marked with an (ok) modifications were necessary to get the serial DROPS code to compile and run.

code	DROPS serial	OpenMP support	DROPS parallel
XEON+gcc333	ok	no	n.a.
XEON+gcc343	ok	no	n.a.
XEON+icc81	(ok)	yes	ok
XEON+pgi60	(ok)	yes	compilation fails
XEON+vs2005	compilation fails	yes	compilation fails
OPT+gcc333	ok	no	n.a.
OPT+gcc333X	ok	no	n.a.
OPT+icc81	(ok)	yes	ok
OPT+icc81X	(ok)	yes	compilation fails
OPT+pgi60	(ok)	yes	compilation fails
OPT+pgi60X	(ok)	yes	compilation fails
OPT+path20	ok	no	n.a.
OPT+path20X	ok	no	n.a.
OPT+ss10	(ok)	yes	compilation fails
USIV+gcc331	ok	no	n.a.
USIV+ss10	(ok)	yes	ok
USIV+guide	(ok)	yes	ok
POW4+guide	(ok)	yes	ok
POW4+xlC60	compilation fails	yes	compilation fails
POW4+gcc343	ok	no	n.a.
IT2+icc81	(ok)	yes	1 thread only

Table 3. Compiler's successes

5 Performance of the OpenMP Version

OpenMP programs running on big server machines operating in multi-user mode suffer from a high variation in runtime. Thus it is hard to see clear trends concerning speed-up. This was particularly true for the SGI Altix. Exclusive access to the 24 core Sun Fire E2900 system helped a lot.

On the 4-way Opteron systems the *taskset* Linux command was helpful to get rid of negative process scheduling effects.

5.1 Assembly of the Stiffness Matrices

Setting up the stiffness matrices could be completely parallelized as described in the previous chapter. Nevertheless the scalability of the chosen approach is limited. The parallel algorithm executed with only one thread clearly performs worse than the tuned serial version, because the parallel algorithm contains the additional summation step as described above (see 4.1). It scales well up to about 8 threads, but then the overhead which is caused by a growing number of dynamic memory allocations and memory copy operations increases. On the USIV+ss10 platform there is still some speedup with more threads, but on the USIV+guide platform we had to limit the number of threads used for the SETUP routines to a maximum of eight in order to prevent a performance decrease for a higher thread count (table 7 and 8). Table 4 shows the runtime of the matrix setup routines on the USIV+guide platform.

code	serial original	serial tuned	parallel (jac0) 1	2	4	8	16
XEON+icc81	1592	816	1106	733	577	n.a.	n.a.
OPT+icc81	1368	778	1007	633	406	n.a.	n.a.
USIV+guide	4512	2246	2389	1308	745	450	460
USIV+ss10	4604	2081	2658	1445	820	523	383
POW4+guide	4580	2119	2215	2285	3659	4726	5995

Table 4. C++ + OpenMP: matrix setup

5.2 The Linear Equation Solvers

The linear equation solvers put quite some pressure on the memory system. This clearly reveals the memory bandwidth bottleneck of the dual processor Intel-based machines (XEON+icc).

The ccNUMA-architecture of the Opteron-based machines (OPT+icc) exhibits a high memory bandwidth if the data is properly allocated. But it turns out that the OpenMP version of DROPS suffers from the fact that most of the data is allocated by the master thread because of the usage of the STL datatypes.

As an experiment we implemented a modification of the stream benchmark using the STL datatype `valarray` on one hand and simple C-style arrays on the other hand. These arrays are allocated with *malloc* and initialized in a parallel region.

Table 5 lists the memory bandwidth in GB/s for the four stream kernel loops and a varying number of threads. It is obvious that the memory bandwidth does not scale when `valarrays` are used. The master thread allocates and initializes (after construction a `valarray` has to be filled with zeros by default) a contiguous memory range for the `valarray` and because of the first touch memory allocation policy, all memory pages are put close to the master thread's processor. Later on, all other threads have to access the master thread's memory in parallel regions thus causing a severe bottleneck.

The Linux operating system currently does not allow an explicit or automatic data migration. The Solaris operating system offers the Memory Placement Optimization feature (MPO), which can be used for an explicit data migration. In our experiment we measured the Stream kernels using `valarrays` after the data has been migrated by a "next-touch" mechanism using the *madvise* runtime function, which clearly improves parallel performance (see table 5).

This little test demonstrates how sensitive the Opteron architecture reacts to disadvantageous memory allocation and how a "next-touch" mechanism can be employed beneficially.

On the USIV+guide and USIV+ss10 platforms we were able to exploit the MPO feature of Solaris to improve the performance of DROPS, but currently there is no C++ compiler available for Solaris on Opteron capable of compiling the parallel version of DROPS.

Stream kernel	**Data structure**	**Initialization method**	**1 Thread**	**2 Threads**	**3 Threads**	**4 Threads**
assignment	valarray	implicit	1.60	1.84	1.94	1.79
	valarray	implicit+madvise	1.60	3.19	4.78	6.36
	C-array	explicit parallel	1.69	3.35	5.00	6.64
scaling	valarray	implicit	1.51	1.81	1.93	1.78
	valarray	implicit+madvise	1.50	2.98	4.47	5.94
	C-array	explicit parallel	1.62	3.22	4.81	6.38
summing	valarray	implicit	2.12	2.16	2.16	2.03
	valarray	implicit+madvise	2.12	4.20	6.22	8.22
	C-array	explicit parallel	2.19	4.34	6.42	8.49
saxpying	valarray	implicit	2.11	2.16	2.15	2.03
	valarray	implicit+madvise	2.10	4.18	6.20	8.20
	C-array	explicit parallel	2.15	4.26	6.30	8.34

Table 5. Stream benchmark, C++ (valarray) vs. C, memory bandwidth in GB/s on OPT+ss10

On the whole the linear equation solvers scale reasonably well given that frequent synchronizations in the CG-type linear equation solvers are inevitable. The modified preconditioner takes more time than the original recursive algorithm for few threads, but it pays off for at least four threads. Table 6 shows the runtime of the solvers.

code	**serial**	**serial**	**parallel (omp_block)**					**parallel (jac0)**				
	original	**tuned**	**1**	**2**	**4**	**8**	**16**	**1**	**2**	**4**	**8**	**16**
XEON+icc81	939	894	746	593	780	n.a.	n.a.	837	750	975	n.a.	n.a.
OPT+icc81	1007	839	823	590	496	n.a.	n.a.	699	526	466	n.a.	n.a.
USIV+guide	2682	2727	2702	1553	1091	957	878	1563	902	524	320	232
USIV+ss10	2741	2724	2968	1672	1162	964	898	2567	1411	759	435	281
POW4+guide	398	428	815	417	333	1171	18930	747	267	308	12268	37142

Table 6. C++ + OpenMP: linear equation solvers

5.3 Total Performance

Table 7 shows the total runtime of the DROPS code on all platforms for which a parallel OpenMP version could be built. Please note that we didn't have exclusive access to the POW4 platform. Table 8 shows the resulting total speedup.

code	**serial original**	**serial tuned**	**parallel (omp_block) 1**	**2**	**4**	**8**	**16**	**parallel (jac0) 1**	**2**	**4**	**8**	**16**
XEON+icc81	2643	1723	2001	1374	1353	n.a.	n.a.	2022	1511	1539	n.a.	n.a.
OPT+icc81	2517	1761	2081	1431	1093	n.a.	n.a.	1962	1382	1048	n.a.	n.a.
USIV+guide	7551	5335	5598	3374	2319	1890	1796	4389	2659	1746	1229	1134
USIV+ss10	7750	5198	6177	3629	2488	2001	1782	5683	3324	2067	1457	1151
POW4+guide	5252	2819	3467	3310	4534	7073	26037	3290	2871	4338	17465	43745

Table 7. C++ + OpenMP: total runtime

Version	**USIV+guide**		**USIV+ss10**		**OPT+icc**	
	omp_block	jac0	omp_block	jac0	omp_block	jac0
serial (original)	1.00	—	1.00	—	1.00	—
serial (tuned)	1.42	—	1.49	—	1.43	—
parallel (1 Thread)	1.35	1.72	1.26	1.36	1.21	1.28
parallel (2 Threads)	2.24	2.84	2.14	2.33	1.76	1.82
parallel (4 Threads)	3.26	4.32	3.11	3.75	2.30	2.40
parallel (8 Threads)	3.99	6.14	3.87	5.32	—	—
parallel (16 Threads)	4.20	6.66	4.35	6.73	—	—

Table 8. Speedup for the USIV+guide, USIV+ss10 and OPT+icc platforms

6 Summary

The compute intense program parts of the DROPS Navier-Stokes solver have been tuned and parallelized with OpenMP. The heavy usage of templates in this C++ program package is a challenge for many compilers. As not all C++ compilers support OpenMP, and some of those which do fail for the parallel version of DROPS, the number of suitable platforms turned out to be quite limited.

We ended up with using the guidec++ compiler from KAI (which is now part of Intel) and the Sun Studio 10 compilers on our UltraSPARC IV-based Sun Fire servers (platform USIV+guide) and the Intel compiler in 32 bit mode on our Opteron-based Linux cluster (OPT+icc).

The strategy which we used for the parallelization of the Finite Element Method implemented in DROPS was straight forward. Nevertheless the obstacles which we encountered were manifold, many of them are not new to OpenMP programmers.

Finally the USIV+guide and USIV+ss10 versions exhibit some scalability. The best effort OpenMP version runs 6.7 times faster with 16 threads than the original serial version on the same platform. But as we improved the serial version during the tuning and parallelization process the speed-up compared to the tuned serial version is only 4.7.

As an Opteron processor outperforms a single UltraSPARC IV processor core it only takes 3 threads on the Opteron-based machines to reach the same absolute speed. On the other hand Opteron processors are not available in large shared memory machines. So shorter elapsed times are not attainable.

As tuning is a never ending process, there still is room for improvement. Particularly the data locality has to be improved for the ccNUMA-architecture of the 4-way Opteron machines.

Acknowledgements
The authors would like to thank Uwe Mordhorst at University of Kiel and Bernd Mohr at Research Center Jülich for granting access to and supporting the usage of their machines, an SGI Altix 3700 and an IBM p690.

References

1. Arnold Reusken, Volker Reichelt, Multigrid Methods for the Numerical Simulation of Reactive Multiphase Fluid Flow Models (DROPS), http://www.sfb540.rwth-aachen.de/Projects/tpb4.php
2. Sven Gross, Jörg Peters, Volker Reichelt, Arnold Reusken, The DROPS package for numerical simulations of incompressible flows using parallel adaptive multigrid techniques, ftp://ftp.igpm.rwth-aachen.de/pub/reports/pdf/IGPM211_N.pdf
3. GNU Compiler documentation, http://gcc.gnu.org/onlinedocs/
4. Intel C/C++ Compiler documentation, http://support.intel.com/support/performancetools/c/linux/manual.htm
5. Guide-Compiler of the KAP Pro/Toolset, http://support.rz.rwth-aachen.de/Manuals/KAI/KAP_Pro_Reference.pdf, http://developer.intel.com/software/products/kappro/
6. PGI-Compiler documentation, http://www.pgroup.com/resources/docs.htm
7. KCC-Compiler, component of guidec++, http://support.rz.rwth-aachen.de/Manuals/KAI/KCC_docs/index.html
8. Pathscale-Compiler, http://www.pathscale.com
9. Sun Analyzer of Sun Studio 9, http://developers.sun.com/tools/cc/documentation/ss9_docs/
10. Intel Threading Tools, http://www.intel.com/software/products/threading/
11. Sven Karlsson, Mats Brorsson, OdinMP OpenMP C/C++ Compiler, http://odinmp.imit.kth.se/projects/odinmp