Large System Performance of SPEC OMP2001 Benchmarks

Hideki Saito (Intel Corporation), [1]
Greg Gaertner (Compaq Computer Corporation),
Wesley Jones (Silicon Graphics Incorporated),
Rudolf Eigenmann (Purdue University),
Hidetoshi Iwashita (Fujitsu Limited),
Ron Lieberman (Hewlett-Packard Corporation),
Matthijs van Waveren (Fujitsu European Centre for IT Limited),
and Brian Whitney (Sun Microsystems, Incorporated)

SPEC High-Performance Group

Abstract. Performance characteristics of application programs on large-scale systems are often significantly different from those on smaller systems. SPEC OMP2001 is a benchmark suite intended for measuring performance of modern shared memory parallel systems. The first component of the suite, SPEC OMPM2001, is developed for medium-scale (4- to 16-way) systems. We present our experiences on benchmark development in achieving good scalability using the OpenMP API. This paper then analyzes the published results of SPEC OMPM2001 on large systems (32-way and larger), based on application program behavior and systems' architectural features. The ongoing development of the SPEC OMP2001 benchmark suites is also discussed. Its main feature is the increased data set for large-scale systems. We refer to this suite as SPEC OMPL2001, in contrast to the current SPEC OMPM2001 (medium data set) suite.

Keywords. SPEC OMP2001, Benchmarks, High-Performance Computing, Performance Evaluation, OpenMP

1 Introduction

SPEC (The Standard Performance Evaluation Corporation) is an organization for creating industry-standard benchmarks to measure various aspects of modern computer system performance. SPEC's High-Performance Group (SPEC HPG) is a workgroup aiming at benchmarking high-performance computer systems. In June of 2001, SPEC HPG released the first of the SPEC OMP2001 benchmark suites, SPEC OMPM2001. This suite consists of a set of OpenMP-based application programs. The data sets of the SPEC OMPM2001 suite (also referred to as the medium suite) are derived from state-of-the-art computation on modern medium-scale (4- to 16-way)

[1] Address: SC12-301, 3601 Juliette Lane, Santa Clara, CA 95052 U.S.A.
E-Mail: hideki.saito@intel.com, Phone: +1-408-653-9471, Fax: +1-408-653-5374

shared memory parallel systems. Aslot et al. [1] have presented the benchmark suite. Aslot et al. [2] and Iwashita et al. [3] have described performance characteristics of the benchmark suite.

As of this writing, a large suite (SPEC OMPL2001), focusing on 32-way and larger systems, is under development. SPEC OMPL2001 shares most of the application code base with SPEC OMPM2001. However, the code and the data sets are modified to achieve better scaling and also to reflect the class of computation regularly performed on such large systems.

Performance characteristics of application programs on large-scale systems are often significantly different from those on smaller systems. In this paper, we characterize the performance behavior of large-scale systems (32-way and larger) using the SPEC OMPM2001 benchmark suite. In Section 2, we present our experiences on benchmark development in achieving good scalability using the OpenMP API. Section 3 analyzes the published results of SPEC OMPM2001 on large systems, based on application program behavior and systems' architectural features. The development of SPEC OMPL2001 is discussed in Section 4, and Section 5 concludes the paper.

2 Experiences on SPEC OMPM2001 Benchmark Development

2.1 Overview of the SPEC OMP2001 Benchmark

The SPEC OMPM2001 benchmark suite consists of 11 large application programs, which represent the type of software used in scientific technical computing. The applications include modeling and simulation programs from the fields of chemistry, mechanical engineering, climate modeling, and physics. Of the 11 application programs, 8 are written in Fortran, and 3 are written in C. The benchmarks require a virtual address space of about 1.5 GB in a 1-processor execution. The rationales for this size were to provide data sets significantly larger than those of the SPEC CPU2000 benchmarks, while still fitting them in a 32-bit address space.

The computational fluid dynamics applications are APPLU, APSI, GALGEL, MGRID, and SWIM. APPLU solves 5 coupled non-linear PDEs on a 3-dimensional logically structured grid, using the Symmetric Successive Over-Relaxation implicit time-marching scheme [4]. Its Fortran source code is 4000 lines long. APSI is a lake environmental model, which predicts the concentration of pollutants. It solves the model for the mesoscale and synoptic variations of potential temperature, wind components, and for the mesoscale vertical velocity, pressure, and distribution of pollutants. Its Fortran source code is 7500 lines long. GALGEL performs a numerical analysis of oscillating instability of convection in low-Prandtl-number fluids [5]. Its Fortran source code is 15300 lines long. MGRID is a simple multigrid solver, which computes a 3-dimensional potential field. Its Fortran source code is 500 lines long.

SWIM is a weather prediction model, which solves the shallow water equations using a finite difference method. Its Fortran source code is 400 lines long.

AMMP (Another Molecular Modeling Program) is a molecular mechanics, dynamics, and modeling program. The benchmark performs a molecular dynamics simulation of a protein-inhibitor complex, which is embedded in water. Its C source code is 13500 lines long.

FMA3D is a crash simulation program. It simulates the inelastic, transient dynamic response of 3-dimensional solids and structures subjected to impulsively or suddenly applied loads. It uses an explicit finite element method [6]. Its Fortran source code is 60000 lines long.

ART (Adaptive Resonance Theory) is a neural network, which is used to recognize objects in a thermal image [7]. The objects in the benchmark are a helicopter and an airplane. Its C source code is 1300 lines long.

GAFORT computes the global maximum fitness using a genetic algorithm. It starts with an initial population and then generates children who go through crossover, jump mutation, and creep mutation with certain probabilities. Its Fortran source code is 1500 lines long.

EQUAKE is an earthquake-modeling program. It simulates the propagation of elastic seismic waves in large, heterogeneous valleys in order to recover the time history of the ground motion everywhere in the valley due to a specific seismic event. It uses a finite element method on an unstructured mesh [8]. Its C source code is 1500 lines long.

WUPWISE (Wuppertal Wilson Fermion Solver) is a program in the field of lattice gauge theory. Lattice gauge theory is a discretization of quantum chromodynamics. Quark propagators are computed within a chromodynamic background field. The inhomogeneous lattice-Dirac equation is solved. Its Fortran source code is 2200 lines long.

2.2 Parallelization of the Application Programs

Most of the application programs in the SPEC OMPM2001 benchmark suite are taken from the SPEC CPU2000 suites. Therefore, parallelization of the original sequential program was one of the major efforts in the benchmark development. The techniques we have applied in parallelizing the programs are described in [1]. We give a brief overview of these techniques and then focus on an issue of particular interest, which is performance tuning of the benchmarks on non-uniform memory access (NUMA) machines.

The task of parallelizing the original, serial benchmark programs was relatively straightforward. The majority of code sections were transformed into parallel form by searching for loops with fully independent iterations and then annotating these loops with OMP PARALLEL DO directives. In doing so, we had to identify scalar and array data structures that could be declared as private to the loops or that are involved in reduction operations. Scalar and array privatization is adequately supported by OpenMP. When part of an array had to be privatized, we created local arrays for that purpose. Parallel array reduction is not supported by the OpenMP Fortran 1.1 specification [9], and thus these array operations have been manually transformed. As we studied performance and scalability of the parallelized code, we realized that LAPACK routines and array initialization should also be parallel. Note that parallel array reduction is supported by the OpenMP Fortran 2.0 specification [10].

In several situations we exploited knowledge of the application programs and the underlying problems in order to gain adequate performance. For example, linked lists were converted to vector lists (change of data structures). In GAFORT, a parallel shuffle algorithm we decided to use is different from the original sequential one (change of algorithms). In EQUAKE, the global sum was transformed to a sparse global sum by introducing a mask. Classical wisdom of parallelizing at a coarser grain also played a role. We have fused small loops, added NOWAIT at the end of applicable work-sharing loops, and even restructured big loops so that multiple instances of the loop can run in parallel. When memory could be traded for a good speedup, we allowed a substantial increase in memory usage.

Our parallelization effort resulted in good theoretical and actual scaling for our target platforms [1]. In all but one code, over 99% of the serial program execution is enclosed by parallel constructs. In GALGEL the parallel coverage is 95%. Figure 1 shows the speedups computed from Amdahl's formula.

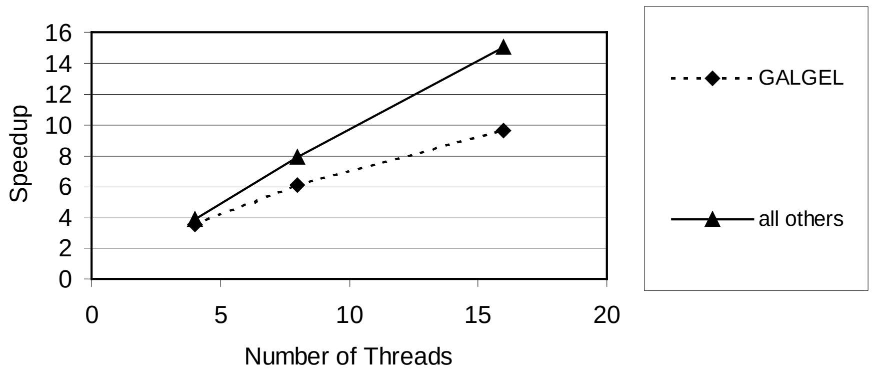

Figure 1: Amdahl's Speedup for 4, 8, and 16 threads.

2.3 Parallelization of the SWIM Application Program with Initialization for NUMA Architectures

SWIM represents one of the more simple benchmarks in the SPEC OMPM2001 suite and makes an easy example of the techniques used to parallelize the applications in the SPEC OMPM2001 suite using the OpenMP Standard. It also exemplifies the use of the parallelization of the initialization of arrays and memory to get good, scalable performance on a number of cache-coherent non-uniform memory access (NUMA) architectures.

Swim has 3 subroutines that contain more than 93% of the work as measured at runtime using a performance analysis tool. Each of these subroutines contained loops, one of which contained most of the work of that subroutine. These are two-dimensional loops, indexed as U(I,J). So, in the initial parallelization of SWIM, these three loops were parallelized over the J-index to reduce false sharing of cache lines and provide enough work in the contiguous I-direction. The application scaled well to 8 CPUs.

Beyond eight CPUs, a number of problems occurred. We will discuss the 32 CPU case. One of the remaining, serial routines took about 6% of the time on one CPU. By Amdahl's law, this routine would consume over 60% of the execution time on 32 CPUs. A performance analysis tool running on all 32 CPUs with the application indicated that most of this time was spent in one loop. We were able to parallelize this loop over the J index with the help of an OpenMP reduction clause.

At this point the application ran reasonably well on 32 CPUs. But, considering that nearly all of the work had been parallelized, a comparison of the 8-CPU time with the 32-CPU time indicated that the application was not scaling as well as expected. To check how sensitive the application was to NUMA issues, we ran 2 experiments on the SGI Origin3000. In the first, we used the default "first touch" memory placement policy where virtual memory is placed on the physical memory associated with the process that touched the memory first, usually via a load or store operation. In the second experiment we used a placement policy where the memory is spread around the system in a round robin fashion on those physical memories with processes that are associated with the job. In the "first touch" case we found load imbalance relative to the "round robin" case. We also found that some parallelized loops ran much slower with the "first touch" relative to "round robin" memory placement, while others ran better with the "first touch" memory placement. A profile of the parallel loops indicated that the arrays that were affected the most in going from "first touch" to "round robin" were the UOLD, VOLD and POLD arrays. Looking at the code, we found that most of the arrays had been initialized in parallel loops along the J index in the same way that they were parallelized in the sections that did most of the work. The primary exceptions were the UOLD, VOLD and POLD arrays, which were initialized in a serial section of the application. After parallelizing the loop that initialized these arrays, we found that the application scaled very well to 32 CPUs.

A final profile of the application indicated that a few small loops that enforce periodic conditions in the spatial variation of the arrays may also be parallelized for an additional improvement in performance on 32 CPUs. However this modification is not part of the current OMPM2001 suite.

3 Large System Performance of SPEC OMPM2001 Benchmarks

Performance characteristics of application programs on large-scale systems are often significantly different from those on smaller systems. Figure 2 shows a scaling of Amdahl's speedup for 32 to 128 threads, normalized by the Amdahl's speedup of 16 threads. The graph now looks notably different from Figure 1.

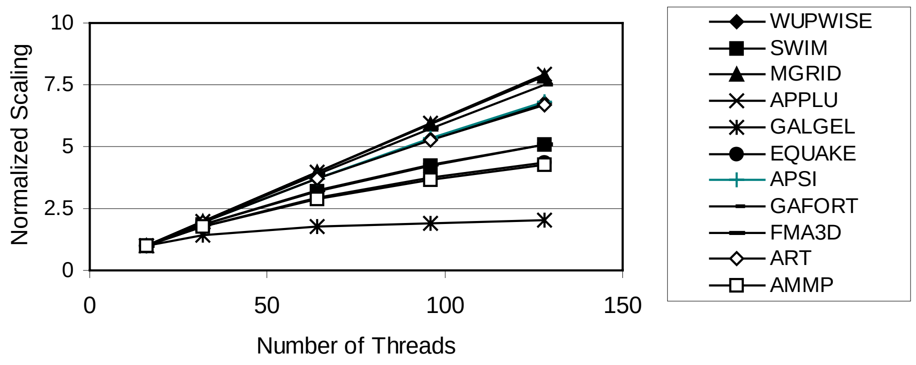

Figure 2: Normalized scaling of Amdahl's Speedup for 16 to 128 threads.

Amdahl's speedup assumes perfect scaling of the parallel portion of the program. Actual programs and actual hardware have additional sources of overhead, which degrade the performance obtained on a real system relative to the upper bound given by Amdahl's law. Figures 3-6 show the scaling data for published benchmark results of SPEC OMPM2001.

3.1 Benchmarks with Good Scalability

The numbers listed in the following figures have been obtained from the results published by SPEC as of December 23, 2001. For the latest results published by SPEC, see http://www.spec.org/hpg/omp2001. All results shown conform to Base Metrics rules, meaning that the benchmark codes were not modified. For better presentation of the graph, we have normalized all results with the lowest published result as of December 23, 2001 for each platform. The results for the Fujitsu PRIMEPOWER 2000 563 MHz system have been divided by the Fujitsu

PRIMEPOWER 2000 18-processor result. The results for the SGI Origin 3800 500 MHZ R14 K system have been divided by the SGI Origin 3800 8-processor result. Finally, the results for the HP PA-8700 750 MHz system have been divided by the HP PA-8700 16-processor result.

The benchmarks WUPWISE, SWIM, and APSI show good scalability up to 128 processors (Figure 3). The scalability of APSI has a dip between 64 and 100 processors, and this is under investigation.

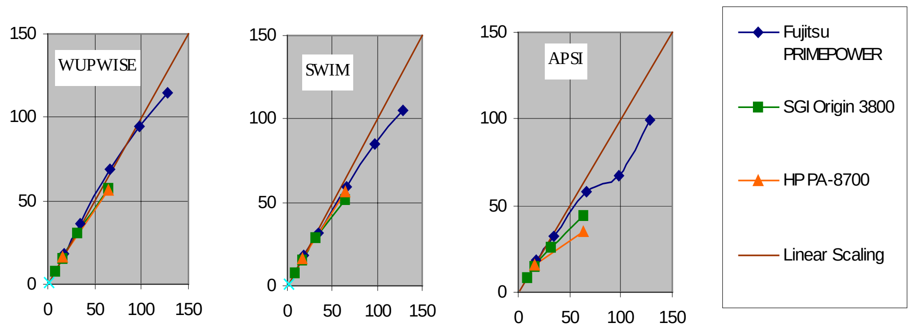

Figure 3: OMPM2001 benchmarks that show good scalability. The X-axis shows the number of processors. The Y-axis shows performance relative to the lowest published result as of Dec. 23, 2001 for each platform.

3.2 Benchmarks with Superlinear Scaling

The benchmark APPLU shows superlinear scaling on HP PA-8700, on the SGI Origin 3800, and on the Fujitsu PRIMEPOWER 2000 due to a more efficient usage of the cache as more processors are used (Figure 4). HP's PA-8700 processor has 1.5MB primary data cache (and no secondary cache), and the R14000 in SGI's Origin 3800 has an 8MB off-chip unified secondary cache. The cache of the 8-processor Origin 3800 system (64MB unified L2 cache) is insufficient to fit the critical data, but the cache of the 16 processor system holds it. The cache of the 64 processor PA-8700 system (96MB L1 data cache) is large enough to hold the critical data. Preliminary results on the Fujitsu PRIMEPOWER 2000 system, which also has an 8MB off-chip secondary cache, show superlinear scaling when going from 8 processors with a 64MB L2 cache to 16 processors with a 128MB L2 cache, and to 22 processors with a 176MB L2 cache.

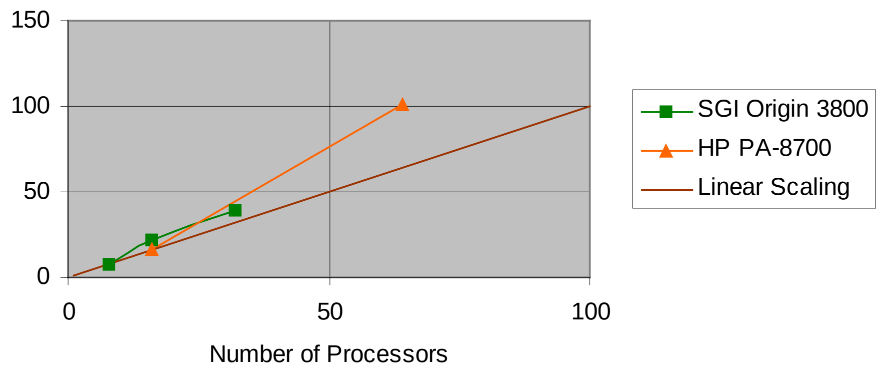

Figure 4: OMPM2001 benchmark APPLU, which shows superlinear scalability. The Y-axis shows the speedup relative to the lowest published result as of Dec. 23, 2001 for each platform.

3.3 Benchmarks with Good Scaling up to 64 Processors

The benchmarks EQUAKE, MGRID, and ART show good scaling up to 64 processors, but poor scaling for larger number of processors (Figure 5). This has been shown on a Fujitsu PRIMEPOWER 2000 system, and the results are normalized by the 18-processor result. MGRID and EQUAKE are sparse matrix calculations, which do not scale well to large number of processors. We expect that the large data set of the OMPL suite, described in the next section, will have better scaling behavior to a large number of processors.

3.4 Benchmarks with Poor Scaling

The benchmarks GALGEL, FMA3D, and AMMP show poor scaling for all number of processors beyond 8 or 16 CPUs (Figure 6). GALGEL uses the QR algorithm, which has been known to be non-scalable.

4 SPEC OMPL2001 Benchmark Development

As of this writing, the development of a new OpenMP benchmark suite, also referred to as SPEC OMPL2001, is underway. In contrast to the SPEC OMPM2001 suite, the target problem size (working set size and run time) is approximately 4x to 8x larger than SPEC OMPM2001. The entire suite should take no more than two days to complete on a 16-CPU 350MHz reference machine. On a reference system, we expect the maximum working set size would be around 6GB, which will require a support for a 64-bit address space[2]

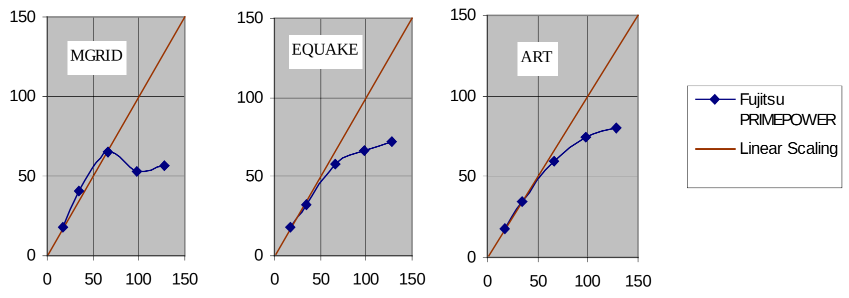

Figure 5: OMPM2001 benchmarks that show good scaling to 64 CPUs. The X-axis shows the number of processors. The Y-axis shows the speedup relative to the 18-CPU results as of Dec. 23, 2001.

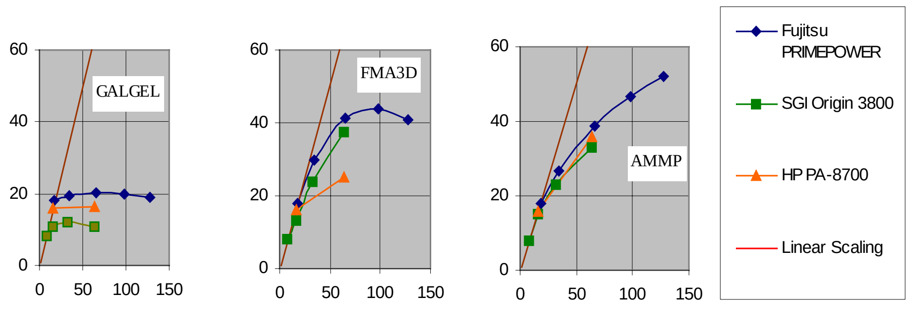

Figure 6: OMPM2001 benchmarks that show poor scaling. The X-axis shows the number of processors. The Y-axis shows the speedup relative to the lowest published results as of Dec. 23, 2001 for each platform.

There are still many unknowns in the final outcome of the SPEC OMPL2001 benchmark suite. We have set the following design goals for the new suite. SPEC OMPL2001 will exercise more code paths than SPEC OMPM2001, necessitating additional parallelization efforts. On some benchmark programs, I/O became a

[2] More memory would be needed if more CPUs were used.

bottleneck in handling larger data sets. When I/O is an integral part of the application, parallel I/O would be called for. Alternatively, output can be trimmed down. ASCII input files are most portable and performance-neutral, but the overhead of converting large ASCII input files to floating-point binary can impact the execution time significantly. Furthermore, the C code will be made C++ friendly, so that C++ compilers can also be used for the benchmarks.

Figure 7 shows the scaling of the working set size and the execution time for a 32-processor system. For example, WUPWISE in SPEC OMPL2001 uses 3.5 times more memory than SPEC OMPM2001, and it takes three times longer to execute. This experiment is based on the latest SPEC OMPL2001 benchmark development kit, and thus the code and the data set are subject to changes before the final release of the benchmark.

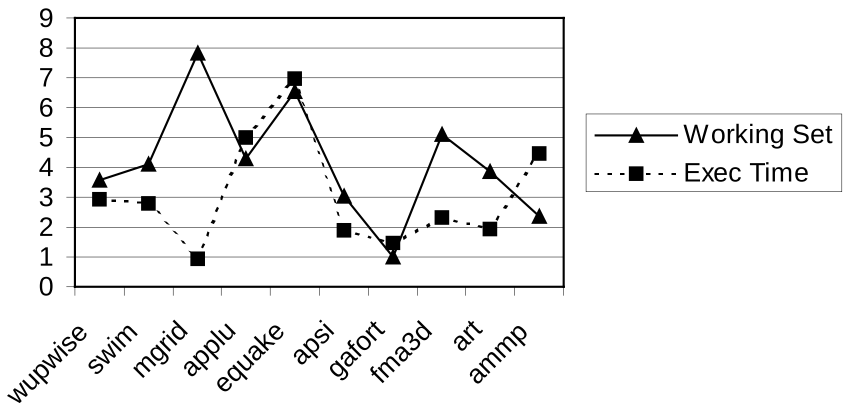

Figure 7 Working set and execution time scaling of a preliminary version of the SPEC OMPL suite

5 Conclusion

In this paper we have analyzed the performance characteristics of published results of the SPEC OMPM2001 benchmark suite. We have found that many of the benchmark programs scale well up to several tens of processors. We have also found a number of codes with poor scalability. Furthermore, we have described the ongoing effort by SPEC's High-Performance Group to develop a new release of the OpenMP benchmark suites, SPEC OMPL2001, featuring data sets up to 6GB in size. SPEC/HPG is open to adopt new benchmark programs. A good candidate program would represent a type of computation that is regularly performed on high-performance computers.

Acknowledgement

The authors would like to thank all of those who developed the application programs and data sets used in the benchmark. Benchmark development would have been impossible without their hard work.

References

[1] Vishal Aslot, Max Domeika, Rudolf Eigenmann, Greg Gaertner, Wesley B. Jones, and Bodo Parady, SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance, In Proc. Of WOMPAT 2001, Workshop on OpenMP Applications and Tools, Lecture Notes in Computer Science, vol. 2104, pages 1-10, July 2001.

[2] Vishal Aslot and Rudolf Eigenmann, Performance Characteristics of the SPEC OMP2001 Benchmarks, in Proc. of the Third European Workshop on OpenMP (EWOMP'2001), Barcelona, Spain, September 2001.

[3] Hidetoshi Iwashita, Eiji Yamanaka, Naoki Sueyasu, Matthijs van Waveren, and Ken Miura, The SPEC OMP2001 Benchmark on the Fujitsu PRIMEPOWER System, in Proc. of the Third European Workshop on OpenMP (EWOMP'2001), Barcelona, Spain, September 2001.

[4] E. Barszcz, R. Fatoohi, V. Venkatkrishnan and S. Weeratunga, Solution of Regular Sparse Triangular Systems on Vector and Distributed-Memory Multiprocessors, Rept. No: RNR-93-007, NASA Ames Research Center, 1993.

[5] Gelfgat A.Yu., Bar-Yoseph P.Z. and Solan A, Stability of confined swirling flow with and without vortex breakdown, Journal of Fluid Mechanics, vol. 311, pp.1-36, 1996.

[6] Key, S. W. and C. C. Hoff, An Improved Constant Membrane and Bending Stress Shell Element for Explicit Transient Dynamics, Computer Methods in Applied Mechanics and Engineering, Vol. 124, pp 33-47, 1995.

[7] M.J. Domeika, C.W. Roberson, E.W. Page, and G.A. Tagliarini, Adaptive Resonance Theory 2 Neural Network Approach To Star Field Recognition, in Applications and Science of Artificial Neural Networks II, Steven K. Rogers, Dennis W. Ruck, Editors, Proc. SPIE 2760, pp. 589-596(1996).

[8] Hesheng Bao, Jacobo Bielak, Omar Ghattas, Loukas F. Kallivokas, David R. O'Hallaron, Jonathan R. Shewchuk, and Jifeng Xu, Large-scale Simulation of Elastic Wave Propagation in Heterogeneous Media on Parallel Computers, Computer Methods in Applied Mechanics and Engineering 152(1-2):85-102, 22 January 1998.

[9] OpenMP Architecture Review Board, OpenMP Fortran Application Programming Interface Version 1.1, November 1999 (http://www.openmp.org/specs/mp-documents/fspec11.pdf)

[10] OpenMP Architecture Review Board, OpenMP Fortran Application Programming Interface Version 2.0, November 2000 (http://www.openmp.org/specs/mp-documents/fspec11.pdf)