The SPEC OMP2001 Benchmark on the Fujitsu PRIMEPOWER System

Hidetoshi Iwashita, Eiji Yamanaka, and Naoki Sueyasu
Strategic Planning Division, Software Group
Fujitsu Ltd.
140 Miyamoto
Numazu-shi, Shizuoka 410-0396, Japan

Matthijs van Waveren
Fujitsu European Centre for Information Technology Ltd.
Hayes Park Central, Hayes End Road
Hayes UB4 8FE, UK

Kenichi Miura
Computer Systems Group
Fujitsu Ltd.
4-1-1 Kamikodanaka, Nakahara-ku,
Kawasaki-shi, Kanagawa 211-8588, Japan

Abstract

The SPEC OMP2001 benchmark suite has recently become available. This suite can be used as a yardstick for measuring and comparing shared-memory systems, which support the OpenMP API. We describe the separate benchmark components in the benchmark suite. We present the results of this benchmark suite on the PRIMEPOWER 2000 system, as officially submitted to SPEC in June 2001. We have run the benchmark on this flat SMP System with up to 128 processors. We analyze the benchmark results. We give some recommendations for an OpenMP programming style.

1. Introduction

The OpenMP Application Programming Interface has emerged as a de-facto standard for expressing shared-memory parallel programs. The EPCC microbenchmark suite [1,2] measures overheads due to synchronization and loop scheduling. However, no adequate yardstick exists for a comprehensive measurement and comparison of shared-memory platforms, which support this API. With the recent release of the SPEC OMP2001 benchmark suite, such a yardstick has become available. Aslot et al [3] have presented the benchmark suite, and have described issues in creating the OpenMP benchmarks.

The suite measures multiprocessor systems performance using code derived from actual applications and with parallel processing capabilities based on the OpenMP standard for shared-memory parallel processing. The suite is targeted at system vendors, software vendors, and customers of high-performance computing systems. It includes a wide range of application benchmarks covering disciplines from computational chemistry to finite element crash simulation and shallow water modeling. The input data sets use up to 1.6 GB of virtual memory and the runs take approximately four hours to run on a 200 MHz single-processor reference machine.

We have executed the components of the SPEC OMP2001 benchmark on the Fujitsu PRIMEPOWER 2000 system [4] with the Parallelnavi software package. The benchmark was run on this flat SMP system with up to 128 processors. The Parallelnavi software is a development and job execution environment for parallel programs on the PRIMEPOWER. It is based on software technologies for Fujitsu's VPP series of distributed-memory parallel vector computers [5, 6].

The benchmarks in the SPEC OMP2001 benchmark suite are described in section 2, the PRIMEPOWER system is described in section 3, and the Parallenavi software package is described in

section 4. The benchmark results are discussed from different points of view in section 5.

2. SPEC OMP2001 Benchmarks

The SPEC OMP2001[1] 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 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 [7]. 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 [8]. 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 Modelling Program) is a molecular mechanics, dynamics, and modelling 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 [9]. 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 [10]. 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 modelling 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 [11]. 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.

3. PRIMEPOWER 2000 System

The Fujitsu PRIMEPOWER 2000 [4] is a parallel computation server supporting up to 128 CPUs and 512 gigabytes of memory. The CPU used is SPARC64GP (563MHz), which conforms to the SPARC International V9 architecture and loads the Solaris 8 operating system. Figure 1 shows the maximum configuration of the system. Each cabinet (node) has 8 system boards and each system board has four CPUs, 16 gigabytes of memory, six PCI cards and a first level (L1) crossbar switch. All system boards, within and between nodes, are connected with the second level (L2) crossbar switch.

A distinguishing feature of PRIMEPOWER 2000 is its behavior as a flat Symmetric Multi-processing (SMP) server. Two-layered crossbar networks allow every CPU to access all memory in the system and guarantee coherency.

[1] SPEC OMP2001 [tm] is a trademark of the Standard Performance Evaluation Corp.

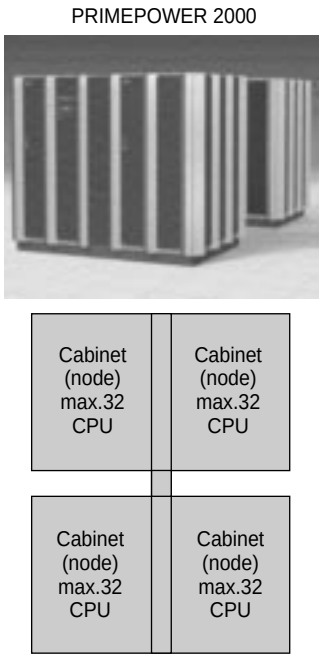

Figure 1: PRIMEPOWER 2000 System

4. Features of Parallelnavi OpenMP

Parallelnavi is a software package for the PRIMEPOWER series server, and includes Fortran and C/C++ compilers, support tools, mathematical libraries, a message passing library (MPI), and a multifunctional job management environment. Parallelnavi Fortran V1.0.2 supports the OpenMP Fortran API Version 1.1 [12] and will support the latest version 2.0 [13]. Parallelnavi C/C++ V1.0.2 supports the OpenMP C/C++ API Version 1.0 [14].

The following parts of this section describe the main features of the Parallelnavi OpenMP language processor.

4.1 Static DO-loop Parallelization

The OpenMP Fortran API specifies two types of DO-loop parallelism: static and dynamic scheduling. Dynamic scheduling requires the system to decide the correspondence of loop indices and threads at runtime. In static scheduling on the other hand, the correspondence can be fixed as a function of the loop parameters and the number of threads at compile time, and does not depend on the runtime status. In our implementation, the loop parameters are represented as expressions for the static scheduling and as library calls for the dynamic scheduling. The difference in cost is clearly shown in Figure 2. Since the cost of static scheduling is small, the cost of DO is mostly BARRIER, which is performed at the exit point of DO block.

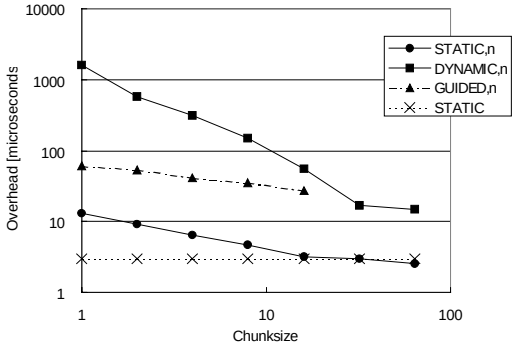

Figure 2: EPCC OpenMP Microbenchmarks Scheduling overheads
Parallelnavi Fortran V1.0. 2 on 8 CPU's

4.2 Software Barrier Synchronization

We have implemented barrier synchronization among threads by using load and store primitives with ceil($\log_2$ N) stages, where N is the number of threads. This method employs N $\log_2$ N barrier flags. Figure 3 illustrates the communication pattern for an example of 4 threads. The barrier flags are indicated as 0 through 7. The first thread, for instance, sets flag 0 on at first, waits for flag 3 to be on, then sets flag 4 on, and waits for flag 6 to be on.

This barrier synchronization is well load-balanced and does not require exclusive execution such as locks and critical sections.

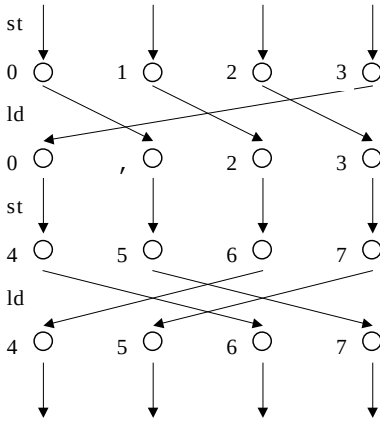

Figure 3: Illustration of software barrier

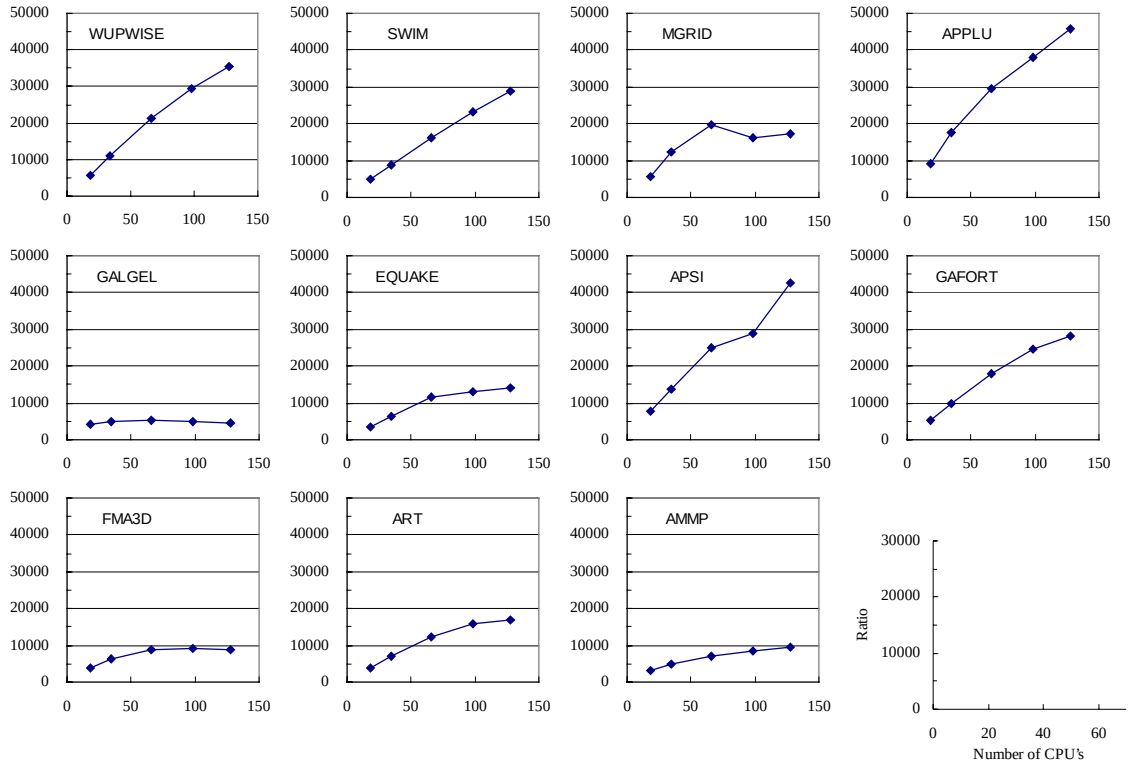

Figure 4: Result of SPEC OMP2001
Parallelnavi Fortran and C/C++ V1.0.2 on PRIMEPOWER 2000 (563MHz)

Cache conflicts are minimized by distributing the barrier flags in a suitable manner.

4.3 Support of Nested Parallelism

The OpenMP Fortran API specifies nested parallelism, in which a thread executing in a parallel region together with other (sister) threads can generate additional (child) threads in a number limited by the runtime environment. Parallelnavi OpenMP supports nested parallelism.

This support, however, enlarges the runtime system and thus affects the performance of other OpenMP applications, which do not use the nested parallelism. So we will prepare another runtime system, that concentrates on non-nested parallel regions and is used only when the job clearly is executed with non-nested parallel regions.

5. Experience on SPEC OMP2001

This section discusses the result of the SPEC OMP2001 Medium data set with the Parallelnavi OpenMP Fortran and C/C++ compilers.

Figure 4 shows the result of all SPEC OMP2001 benchmarks (Medium size) up to 128 CPU's. The numbers listed include all Fujitsu base metric results published by SPEC between 27 June 2001 and 27 July 2001. For the latest results published by SPEC, see http://www.spec.org/hpg/omp2001/. All results conform to Base Metrics rule, meaning that the benchmark codes were not modified.

5.1 Fine Scalability -- WUPWISE, SWIM, APPLU, and GAFORT

SWIM and APPLU have simple parallel regions with work-sharing directives. WUPWISE uses parallel versions of the LAPACK routines. GAFORT underwent a quite major transformation during parallelization [3].

These four benchmarks show good scalability up to 128 CPU's as shown in Figure 4. This is as expected for SWIM, APPLU, and WUPWISE. The result for GAFORT shows that the extensive parallelization work of [3] has been quite successful. While the Medium data set of SPEC OMP2001 is specified for SMP systems of about 10 CPU's, we

verified the scalability of the SMP system PRIMEPOWER up to 128 CPU's.

5.2 Automatic Arrays on Stack -- APSI

In APSI, large arrays are allocated frequently at the top of subroutines in the dynamic extent of the parallel region. Such allocation performed in parallel often causes partial serialization and lock conflict. To avoid such conflicts, the arrays should be managed locally to each thread without interruption among threads. Our compiler succeeded in allocating such automatic arrays onto the local stack of each thread at the entrance of subroutines, using the `-Nautoobjstack` compiler option. The effectiveness of the option can be shown in Figure 5. The result is that APSI also shows quite scalable behavior, as shown in Figure 4.

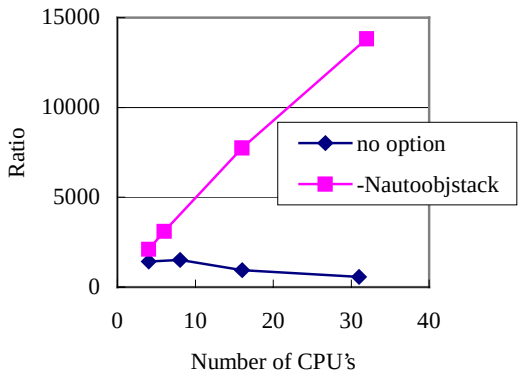

Figure 5: Optimization of memory allocation on APSI

5.3 Inefficient PARALLEL DO -- GALGEL

GALGEL is one of the hardest benchmark in the suite to get high performance. It includes about 90 PARALLEL DO blocks and all of them enclose only a few assignment statements without nested-DO loops, as shown in Figure 6 for instance. This kind of PARALLEL DO block cannot be executed effectively because the cost of thread fork/join is relatively high compared to the parallel computation inside the block. Therefore, a naive implementation could cause even lower performance than the serial execution.

A typical PARALLEL DO block:

```
!$OMP PARALLEL DO
      DO i = 1,NUMNE
        NNPSETS(i) = 0
      ENDDO
```

One of the largest PARALLEL DO block:

```
!$OMP PARALLEL DO PRIVATE(SUM)
      DO 140 J = 1, N
        SUM = V1*C( 1, J ) + V2*C( 2, J ) +
     $        V3*C( 3, J ) +
     $        V4*C( 4, J ) + V5*C( 5, J ) +
     $        V6*C( 6, J ) +
     $        V7*C( 7, J )
        C( 1, J ) = C( 1, J ) - SUM*T1
        C( 2, J ) = C( 2, J ) - SUM*T2
        C( 3, J ) = C( 3, J ) - SUM*T3
        C( 4, J ) = C( 4, J ) - SUM*T4
        C( 5, J ) = C( 5, J ) - SUM*T5
        C( 6, J ) = C( 6, J ) - SUM*T6
        C( 7, J ) = C( 7, J ) - SUM*T7
  140   CONTINUE
```

Figure 6: Examples of PARALLEL DO directives in GALGEL

To avoid such a situation, the compiler selects a serial version of the execution code for the PARALLEL DO block statically or dynamically, instead of the faithful implementation of the PARALLEL DO directive. When the serial execution is selected at the compile time or runtime, the performance of the PARALLEL DO block will be guaranteed to be at least as high performance as the serial execution. According to our experience, the performance goes slightly up as a function of the number of CPU's from 4349 for 18 CPU's up to 5149 for 66 CPU's and then down to 4740 for 128 CPU's.

5.4 Loop Interchange -- EQUAKE

Similar to GALGEL, the key of the performance in EQUAKE (written in OpenMP C) was a PARALLEL FOR directive which contains trivial computation as shown in Figure 7. In this case, however, the loop interchange technique worked well in the compiler, which hoists the thread fork/join overhead out of the outer loop. So good scalability continued up to 128 CPU's.

```
  for (j = 0; j < numthreads; j++) {
#pragma omp parallel for private(i)
    for (i = 0; i < nodes; i++) {
      w2[j][i] = 0;
    }
  }
```

Figure 7: Critical PARALLEL FOR block in EQUAKE

5.5 General discussions

We met performance problems related to the memory allocation for all the SPEC OMP2001 benchmarks. Not only explicit allocatable and automatic arrays, but also array expressions and array assignment statements of Fortran90 cause dynamic allocation generated by the compiler. For high performance, all of these allocations must be reduced and handled in parallel with the least number of conflicts between the threads. Even if the `malloc` system call is called thread-safe, it often causes exclusive execution or severe cache conflict. As mentioned in the example of APSI, avoiding memory allocation gives the best results.

As shown in GALGEL and EQUAKE, the OpenMP program sometimes contains many PARALLEL DO/FOR directive blocks enclosing a small amount of computation. We would like to recommend a programming style in which many DO/FOR directive blocks are enclosed in a large PARALLEL directive block. Thus the number of thread fork/join can be reduced as much as possible. However it might be difficult to get this programming style accepted by developers, since the former style is easier to use.

6. Conclusion

We evaluated the Medium size SPEC OMP2001 benchmark with the Parallelnavi Fortran and C/C++ compilers on the SMP server Fujitsu PRIMEPOWER2000. Without modification of benchmark codes (Base Metrics rule), the compilers have extracted high and scalable performance for most benchmark codes in SPEC OMP2001.

We can conclude that the PRIMEPOWER 2000 is a suitable platform for the implementation of OpenMP. With respect to the flat SMP hardware, it is shown that the execution performance increases smoothly with the number of CPU's without bending at any special numbers.

References

[1] EPCC Microbenchmarks. http://www.epcc.ed.ac.uk/research/openmpbench/

[2] J.M. Bull. Measuring Synchronisation and Scheduling Overheads in OpenMP. *In Proc. Of EWOMP99, First European Workshop on OpenMP.* (http://www.epcc.ed.ac.uk/research/openmpbench/ewomp.pdf)

[3] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W.B. Jones, and B. Parady. SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance. In *Proc. Of WOMPAT2001, Workshop on OpenMP Applications and Tools, Lecture Notes in Computer Science*, 2104, pages 1-10, July 2001. (http://www.ece.purdue.edu/~eigenman/reports/wompat01spec.pdf)

[4] N. Izuta, T. Watabe, T. Shimizu, and T. Ichihashi. Overview of PRIMEPOWER 2000/1000/800 Hardware. *Fujitsu Sci. Tech. J.* 36(2):121-127, 2000. (http://magazine.fujitsu.com/us/vol36-2/paper03.pdf).

[5] H. Iwashita, S. Okada, M. Nakanishi, T. Shindo, and H. Nagakura. VPP Fortran and Parallel Programming on the VPP500 Supercomputer. In *Proceedings of the 1994 International Symposium on Parallel Architectures, Algorithms and Networks (poster session papers)*, pages 165-172, Kanazawa, Japan, December 1994.

[6] H. Iwashita, N. Sueyasu, S. Kamiya, and M. van Waveren. VPP Fortran and the Design of HPF/JA Extensions, *Concurrency: Practice and Experience*. To be published.

[7] 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*

[8] 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.

[9] 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.

[10] 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).

[11] 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.

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

[13] OpenMP Architecture Review Board. *OpenMP Fortran Application Program Interface Version 2.0*, November 2000. (http://www.openmp.org/specs/mp-documents/fspec20.pdf)

[14] OpenMP Architecture Review Board. *OpenMP C/C++ Application Program Interface Version 1.0*, October 1998. (http://www.openmp.org/specs/mp-documents/cspec10.pdf)