from *SIAM News*, Volume 32, Number 9

Combining Message-passing and Directives in Parallel Applications

By Steve Bova, Clay Breshears, Rudolf Eigenmann, Henry Gabb, Greg Gaertner, Bob Kuhn, Bill Magro, Stefano Salvini, and Veer Vatsa

APPLICATIONS ON ADVANCED ARCHITECTURE COMPUTERS
Greg Astfalk, Editor

Developers of parallel applications can be faced with the problem of combining the two dominant models for parallel processing—distributed-memory and shared-memory parallelism—within one source code. In this article we discuss why it is useful to combine these two programming methodologies, both of which are supported on most high-performance computers, and some of the lessons we learned in work on five applications.

All our applications make use of two programming models: message-passing, as represented by the PVM or MPI libraries, and the shared-memory style, as represented by the OpenMP directive standard. In all but one of the applications, we use these two models to exploit computer architectures that include both shared- and distributed-memory systems. Message-passing is used to coordinate coarse-grained parallel tasks across distributed compute nodes, whereas OpenMP exploits parallelism within multiprocessor nodes. One of our applications, SPECseis96, implements message-passing and shared-memory directives at equal levels, which allows us to compare the performance of the two models.

CGWAVE

MPI and OpenMP were used concurrently in the CGWAVE application to improve the performance of harbor analyses. The project, "Dual-level parallel analysis of harbor wave response using MPI and OpenMP," won the Most Effective Engineering Methodology award in the SuperComputing–98 HPC Challenge competition. In this case CGWAVE was used to model the motion of waves in Ponce Inlet on Florida's Atlantic coast [1].

The sea state in a harbor is characterized by a number of incident wave components, which are defined by period, amplitude, and direction. This set of wave components can be regarded as a parameter space; each triplet leads to a separate partial differential equation to be solved on the finite element grid. Parallelism can be exploited at the parameter-space level, with MPI used to distribute the work. Since the execution times for separate wave components can differ by as much as a factor of four, a simple master–slave strategy dynamically balances the workload. For each component calculation, a large sparse linear system of equations is formed. The solution of the system of equations is parallelized with OpenMP. This OpenMP-based parallelization is performed in *each* parameter-space component problem.

Developing this hybrid parallelism was easier than expected, once a few problems had been resolved. Problems occur in a mixed MPI/OpenMP program when `MPI_Init` or other MPI communication routines are called from an OpenMP parallel region. In the parallelization of CGWAVE, we were able to avoid this problem by allowing only one thread to execute message-passing operations.

At the OpenMP level, scalability on the SGI Origin2000 ccNUMA architecture necessitated distribution of the data required by the conjugate gradient solver. CGWAVE takes advantage of the "first touch" rule to distribute the data transparently. Specifically, an item of data resides, i.e., it is co-located, with the processor that first touches it. (This first-touch approach is a feature of the computer system, and not of OpenMP.) The important arrays are initialized in parallel in such a way that the processor initializing the data will be the processor to execute the compute-intensive work on that data.

Also in preparation for use of OpenMP, work arrays in `COMMON` blocks were declared locally in the conjugate gradient subroutine, which eliminated array-access synchronization and coherency overhead. This was the only modification to the original source. The KAP/Pro Toolset (Kuck & Associates, Inc. (http://www.kai.com) was used for parallel assurance testing (Assure) and performance optimization (GuideView).

Figure 1 shows relative performance for computation of the harbor wave problem with different numbers of MPI and OpenMP slaves. Timings were done on a 112-processor SGI Origin2000. Although it is tempting to draw conclusions about the relative performance of MPI and OpenMP, each of the parallelism models was applied to a different level of parallelism in CGWAVE and comparison of MPI versus OpenMP parallelism is therefore not appropriate.

GAMESS

The traditional Hartree–Fock, self-consistent field (SCF) approach to computing wavefunctions, using a finite basis set of Gaussian-type functions, has been a mainstay of ab initio quantum chemistry since digital computers first became sufficiently powerful to tackle polyatomic molecules. GAMESS-US is a well-studied example of a computation of this type [2].

A key reason for the targeting of both distributed-memory and shared-memory programming for GAMESS is illustrated by current parallel processing technology. Consider the Compaq AlphaServer architecture. A typical clustered system would consist of AlphaServer 8400 or AlphaServer 4100 servers, connected with Memory Channel. Memory Channel provides a high-bandwidth connection between Compaq Alpha systems. In principle, use of the SMP bus, as compared with Memory Channel, can improve performance by a factor of five.

Both models of parallelism have been applied to GAMESS. In GAMESS, four nested parallel loops select electron orbitals for integration. At an outer-loop level, message-passing works well and allows the use of distributed-memory architectures. At an inner-loop level, OpenMP's finer-granularity dynamic scheduling can provide better load balancing.

Table 1 shows the performance of parallel GAMESS on a cluster of four Compaq Alpha 8400 systems, with eight EV5 Alpha processors in each system, connected by Memory Channel. A speedup of greater than 5, relative to the four-processor time, is shown for 32 processors (with the obvious limit being a relative speedup of 8).

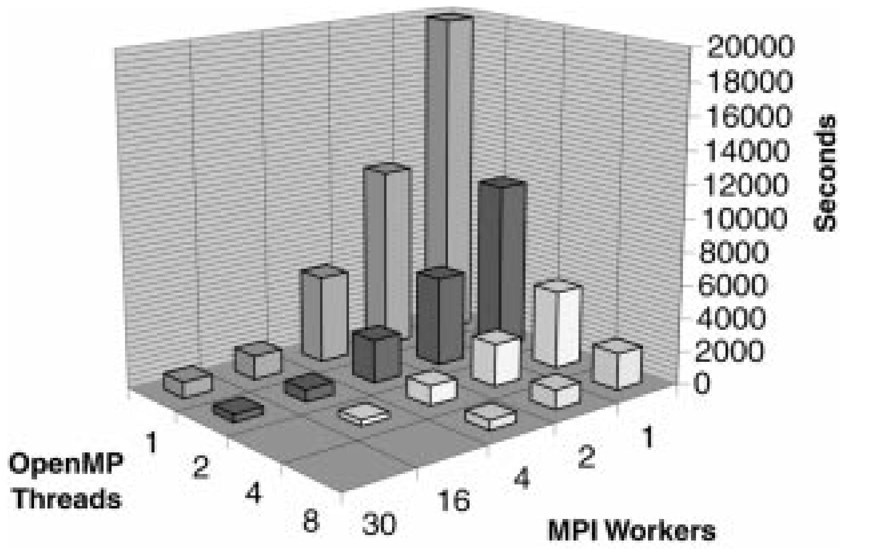

Figure 1. *Wall-clock time on a 112-processor Origin2000 for various combinations of MPI and OpenMP "slaves."*

Linear Algebra Study

As part of an ongoing collaboration, NAG Ltd. and the Albuquerque High Performance Computer Center (AHPCC) analyzed the feasibility of mixed-mode parallelism on a Model F50-based IBM SP system [3]. Each Model F50 SMP node has four processors connected by an enhanced interconnect.

In linear algebra operations, dynamic load balancing is particularly difficult for message-passing paradigms: Any approach based on migration of data across nodes would entail significant communication cost and code complexity. For a fixed problem size, as the number of nodes increases, computation time decreases while communication costs and load imbalance both tend to increase.

Message-passing efficiency can be increased with hybrid parallelism. If N_t is the total number of processors and N_{smp} the number of processors per node, message-passing will occur between only N_t / N_{smp} communicating entities. In other words, communication costs and overhead will be comparable to those of a smaller message-passing system. Load imbalance will also be reduced. Furthermore, if communication is introduced within the regime of dynamic load balancing, communication and computation can be overlapped, referred to as "communication hiding," within each SMP node, reducing communication costs by up to a factor N_{smp}. On the IBM SP used in this study, communication costs were reduced by up to 75%.

The performance results illustrate the importance of communication hiding, which was implemented and measured in the linear algebra study quite simply: Communications were performed first outside and then inside the parallel region.

OpenMP directives requiring the dynamic scheduling of a `DO` loop were easily used to hide communication costs—the matrix block broadcast was treated as one special iteration of the `DO` loop. In the alternative, an adaptive load-balancing scheme, the matrices were subdivided into column-blocks, one for each processor in the SMP node. The column-block accessed by the processor performing the communication was narrower than the others, with its width determined by a set of ad hoc cost parameters.

The IBM SP system supports POSIX threads, and their MPI libraries are thread-safe, which allows the coexistence of the two modes of parallelism. Not all implementations of MPI provide the same level of thread-safety. Thread-safety should be checked by users of mixed-mode parallelism.

No. of Processors	Elapsed Time (sec)	Speedup
4	327	1
8	178	1.84
16	101	3.24
24	76	4.30
32	64	5.11

Table 1. *Cluster performance of four eight-processor systems on GAMESS.*

Table 2 shows the performance of a QR factorization for various cluster configurations, with and without communication hiding (the "Hide" and "No Hide" columns, respectively), and for dynamic versus adaptive strategies. The performance data are shown in Mflops; all execution times were measured using the system's wall-clock timer. The configurations tested are $N_n \times N_{smp}$, where N_n is the number of nodes and N_{smp} is the number of processors per node. No attempt was made to optimize the routines manually. For comparison, performance of 1060 Mflops was reported for the 1×4 configuration (i.e., pure four-way SMP) with the NAG library routine `F68AEF`, $n = 2000$ [3]. Performance of 743 Mflops was measured with the LAPACK routine `DGEQRF`.

TLNS3D

TLNS3D, developed at NASA Langley, is a thin-layer Navier–Stokes solver used in computational fluid dynamics analysis. The program is capable of handling models composed of multiple blocks connected via various boundary conditions.

To simplify flow modeling of complex objects, a typical input data set contains multiple blocks, motivated by the geometry of the physical model. These blocks can be computed concurrently; MPI is used to divide the blocks into groups and assign each group to a process. The block assignment is static for the duration of the run because distinct data files must be created for each slave.

This approach is quite effective for models in which the number of blocks greatly exceeds the number of slaves, since TLNS3D can generally group the blocks in such a way that each MPI slave does roughly the same amount of work. Unfortunately, as the number of MPI slaves increases, the potential for static load balancing diminishes.

To address the limitations of parallelization at the MPI level, OpenMP directives were added to exploit the inherent parallelism within each block. Each block is represented as a three-dimensional grid, and most of the computations on that grid are in the form of loops that can be performed in parallel. Because there are many such parallel loops in TLNS3D, the OpenMP directives were carefully tuned to maximize cache affinity between loops and to eliminate unnecessary global synchronization among threads.

This application may seem similar to the linear algebra study and GAMESS. We found, however, that a ccNUMA architecture like the SGI Origin2000 allowed us to configure "virtual" SMP nodes that did not have a hardware limited number of processors for the OpenMP level. The mixed-parallel version of TLNS3D achieves load balancing by first partitioning the blocks across MPI slaves to achieve the best possible static load balance. A group of threads, equal in number to the number of processors to be used, is then partitioned among blocks such that the number of grid points per thread is approximately equal. For example, a block containing 60,000 grid points would have roughly twice as many threads at its disposal as a block containing 30,000 grid points. This nonuniform allocation of threads achieves a second form of load balancing that is effective for runs on very large numbers of processors.

	Dynamic Versions ($N_n \times N_{smp}$)							
	1×1		1×4		2×4		4×4	
n	Hide	No Hide	Hide	No Hide	Hide	No Hide	Hide	No Hide
500	218	208	611	494	656	618	628	
1000	225	229	732	678	1128	912	1231	1131
2000			746	773	1310	1185	1963	1579
4000							2467	2124

	Adaptive Versions ($N_n \times N_{smp}$)							
	1×1		1×4		2×4		4×4	
n	Hide	No Hide	Hide	No Hide	Hide	No Hide	Hide	No Hide
1000	229		700		1225		1796	
2000			713		1409		2507	
4000							2758	

Table 2. *QR factorization performance, in Mflops, on various cluster configurations.*

Parallel processing tools were found to be very effective in analyzing TLNS3D. For MPI, the VAMPIRTRACE and VAMPIR programs from Pallas (http://www.pallas.com) analyze the message-passing performance to identify where delays occur. Block load imbalance can be identified in this way. On the OpenMP side, GuideView from KAI identifies OpenMP performance problems. The KAI Assure tool was also used to find shared variables needing synchronization, a potential problem in the conversion of distributed-memory programs (i.e., MPI) to directives, given that shared variables can be touched by any processor.

SPECseis96

SPECseis96 is a seismic processing benchmark used by the Standard Performance Evaluation Corporation (SPEC) (http://www.spec.org). It is representative of modern seismic processing programs used in the search for oil and gas [4].

The benchmark was originally in message-passing form. The motivation for the development of an OpenMP version was the increasing availability of shared-memory parallel systems. In developing this version, we started from the message-passing variant. Unlike the other applications discussed in this article, SPECseis96 does not combine the two models, but rather uses either PVM/MPI libraries or OpenMP directives for a particular run. Because the two variants exploit the same level of parallelism, we can use this code to compare the two programming paradigms.

The message-passing variant starts directly in SPMD mode; that is, all processes start by executing exactly the same program. During the initialization phase, which must be executed by the master processor only, the other processes are explicitly waiting. In contrast, the OpenMP variant must start in sequential mode before opening an SPMD parallel section, which in this case encompasses the rest of the program.

In both versions, all data are kept local to the processes and are partitioned in the same way. In PVM or MPI programs, all data are always local to the processes, whereas OpenMP programs give explicit locality attributes (the default is globally shared data). The only data elements that are declared shared in SPECseis are the regions for exchanging data between the processes. These regions are used in a mode similar to that of implementations of message-passing libraries on SMP systems. The "sending" thread copies data to the shared buffer, and the "receiving" thread copies the data into its local data space. This scheme can be improved by allocating in shared-memory all data that will need communication, although we have not done so in this version of SPECseis.

Many might perceive message-passing to be more scalable than directive-based parallelism. However, our message-passing and OpenMP variants use the same parallelization scheme, the same data partitioning, and the same high-level parallelism, and they achieve the same scalability. Table 3 shows the results obtained on an SGI

	Processors			
	1	2	4	8
PVM	1	1.7	2.8	5.3
OpenMP	1	1.9	3.6	5.6

Table 3. *SPECseis speedups on the SGI Power Challenge using both PVM and OpenMP.*

Power Challenge. Although the OpenMP variant runs slightly faster than the PVM version, we have seen that this slight difference disappears if we increase the data set size. Hence, we attribute it to the higher message-passing costs for the exchange of small data sections.

Summary

All five of the applications described in this article were successfully developed into high-performance programs that use both message-passing and directive-based parallel models. For the most part, multiple levels of parallelism were used, with distributed-memory programming (MPI) for the coarser-grained levels and shared-memory programming (OpenMP) for the finer-grained. This can be achieved without loss of performance.

Table 4 summarizes what we learned in each application.

The application developers have found it relatively easy to use both parallelism models in the same application, in part because of the existence of strong standards, such as MPI and OpenMP, and their efficient implementations on many systems. In addition, we have found that strong standards stimulate the development of good parallel programming tools for each of the models. Therefore, we believe that combining MPI and OpenMP is a very feasible programming paradigm that can be used in many applications.

	CGWAVE
Motivation	Add performance needed to attack another dimension in the problem
Message-passing	Master–slave applied to wave parameter space
Directives	Sparse solver applied to PDE
Platforms	Multiple SGI Origin2000s
Problems	Calling message-passing routines in OpenMP parallel regions
Parallel software engineering	Used Assure to explore OpenMP parallelism
	GAMESS
Motivation	Flexible use of SMP clusters on problem with high degree of parallelism
Message-passing	Outer coarser-grained parallel loop
Directives	Inner finer grain more variable in size
Platforms	Memory Channel AlphaServer 8400
Problems	Fine granularity in MPI and thread-private efficiency of OpenMP
Parallel software engineering	OpenMP versions sometimes much simpler
	Linear Algebra Study
Motivation	Achieve message-passing scalability /use directives dynamic scheduling
Message-passing	Block-solve matrix system with fixed distribution
Directives	Dynamic or adaptive scheduling of block solution
Platforms	SP2 with F50 nodes
Problems	MPI could not be used within node; incomplete support for OpenMP
Parallel software engineering	Porting and maintaining two levels difficult
	SPECseis
Motivation	Portable benchmark, distributed-memory and shared-memory systems
Message-passing	SPMD: Compute, barrier, communicate, then repeat
Directives	Same parallelism but built with different model
Platforms	SGI, Sun
Problems	Setting up message-passing configuration, thread-safety of libraries
Parallel software engineering	Emulating message-passing in directives
	TLNS3D
Motivation	Assignment of grid blocks left poor load balance for MPI
Message-passing	Group of grid blocks assigned to each slave by master
Directives	More or fewer processors assigned to each slave
Platforms	SGI Origin2000
Problems	Need for flexible clustering of processors to SMP nodes
Parallel software engineering	One expert for MPI, another for OpenMP

Table 4. *Comparison of five applications in which both message-passing and directives were used.*

References

[1] S.W. Bova, C.P. Breshears, C. Cuicchi, Z. Demirbilek, and H.A. Gabb, *Dual-level parallel analysis of harbor wave response using MPI and OpenMP*, Int. J. High Performance Comput., in press, 1999.

[2] R. Kuhn, G. Gaertner, and D. Schneider, *Towards standard supercomputer benchmarks for computational chemistry*, 1996 American Institute of Chemical Engineers Annual Meeting.

[3] S. Salvini, B.T. Smith, and J. Greenfield, *Towards mixed mode parallelism on the new model F50-Based IBM SP system*, Albuquerque High Performance Computing Center, University of New Mexico, Technical Report AHPCC98–003, September 1998.

[4] R. Eigenmann and S. Hassanzadeh, *Benchmarking with real industrial applications: The SPEC High-Perfomance Group*, IEEE Computational Science and Engineering, Spring 1996.

Steve Bova (gabb@ibm.wes.hpc.mil) is the computational fluid dynamics lead at the Engineer Research & Development Center MSRC in Vicksburg, Mississippi, where Clay Breshears (gabb@ibm.wes.hpc.mil) is the scalable parallel tools lead and Henry Gabb (gabb@ibm.wes.hpc.mil) is the director of scientific computing. Rudolf Eigenmann (eigenman@ecn.purdue.edu) is an associate professor at Purdue University. Greg Gaertner (ggg@zko.dec.com) is a principal software engineer at Compaq Computer, in Nashua, New Hampshire. Bob Kuhn (kuhn@kai.com) is the director of products and Bill Magro (magro@kai.com) is a parallel applications engineer at Kuck & Associates, in Champaign, Illinois. Stefano Salvini (stef@nag.co.uk) is the high performance computing group leader at NAG Ltd., in Oxford, United Kingdom. Veer Vatsa (v.n.vatsa@larc.nasa.gov) is a senior research scientist at NASA Langley Research Center in Hampton, Virginia.