Performance Tradeoffs in Multi-tier Formulation of a Finite Difference Method

Scott B. Baden[1] and Daniel Shalit[1]

University of California, San Diego
Department of Computer Science and Engineering
9500 Gilman Drive, La Jolla, CA 92093-0114 USA
`baden@cs.ucsd.edu,dshalit@cs.ucsd.edu`
`http://www.cse.ucsd.edu/users/{baden,dshalit}`

Abstract. *Multi-tier* platforms are hierarchically organized multicomputers with multiprocessor nodes. Compared with previous-generation *single-tier* systems based on uniprocessor nodes, they present a more complex array of performance tradeoffs. We describe performance programming techniques targeted to finite difference methods running on two large scale multi-tier computers manufactured by IBM: NPACI's Blue Horizon and ASCI Blue-Pacific Combined Technology Refresh. Our techniques resulted in performance improvements ranging from 10% to 17% over a traditional single-tier SPMD implementation.

1 Introduction

Multi-tier computers are hierarchically organized multicomputers with enhanced processing nodes built from multiprocessors [13]. They offer the benefit of increased computational capacity while conserving a costly component: the switch. As a result, multi-tier platforms offer potentially unprecedented levels of performance, but increase the opportunity cost of communication [8,1,4]

We have previously described *multi-tier* programming techniques that utilize knowledge of the hierarchical hardware organization to improve performance [2]. These results were obtained on SMP clusters with tens of processors and hence did not demonstrate scalability. In this paper, we extend our techniques to larger-scale multi-tier parallelism involving hundreds of processors, and to deeper memory hierarchies. We describe architecture-cognizant policies needed to deliver high performance in a 3D iterative finite difference method for solving elliptic partial differential equations. 3D Elliptic solvers are particularly challenging owing to their high memory bandwidth requirements. We were able to improve performance over a traditional SPMD implementation by 10% to 17%.

The contribution of this paper is a methodology for realizing overlap on large-scale multi-tier platforms with deep memory hierarchies. We find that uniform partitionings traditionally employed for iterative methods are ineffective, and that irregular, multi-level decompositions are needed instead. Moreover, when reformulating an algorithm to overlap communication with computation, we must avoid even small amounts of load imbalance. These can limit the ability to realize overlap.

2 Motivating Application

2.1 A Finite Difference Method

Our motivating application solves a partial differential equation–Poisson's equation in three dimensions. The solver discretizes the equation using a 7-point stencil, and solves the discrete equation on a 3-d mesh using Gauss-Seidel's method with red-black ordering. We will refer to this application as *RedBlack3D*.

We assume a hierarchically constructed multicomputer with N processing nodes. Each node is a shared memory multiprocessor with p processors. When $p = 1$ our machine reduces the degenerate case of a *single-tier* computer with a flattened communication structure. For $p > 1$ we have a *multi-tier* computer.

Our strategy for parallelizing an iterative method is to employ a blocked hierarchical decomposition, reflecting the hierarchical construction of the hardware [1,2]. Fig. 1 shows hierarchical decomposition. The first-level subdivision (Fig. 1a) splits the computational domain into N uniform, disjoint blocks or subdomains. The second level (Fig. 1b) subdivides each of the N blocks into p disjoint sub-blocks. Each first-level block is buffered by a surrounding *ghost region* holding off-processor values.

The calculation consists of successive steps that compute and then communicate to fill the ghost cells. After communication of ghost cells completes, control flow proceeds in hierarchical fashion, passing successively to node-level and then processor-level execution.

Each node sweeps over its assigned mesh, enabling its processors to execute over a unique sub-block. Once the processors finish computing, control flow lifts back up to the node level: each node synchronizes its processors at a barrier, and the cycle repeats until convergence.

Under this hierarchical model, nodes communicate by passing messages on behalf of their processors. Since ghost cells are associated with nodes rather than processors, processors on different nodes do not communicate directly.

2.2 Overlap

Communication delays are long on a multi-tier computer because multiple processors share a communication port to the interconnection network. To cope with long communication delays, we reformulate the iterative method to overlap communication with computation by pre-fetching the ghost cells [14].

As illustrated in Fig. 1(b), we peel an annular region from surface of each node's assigned subdomain, and defer execution on this annulus until the ghost cells have arrived. We initiate communication asynchronously on the ghost cells, and then compute on the interior of the subdomain, excluding the annular region. This is shown in Fig. 1(b). After computation finishes, we wait for communication to complete. Finally, we compute over the annular region.

We now have the basis for building an efficient iterative method on a multi-tier computer. We next discuss the performance programming techniques required to implement the strategy.

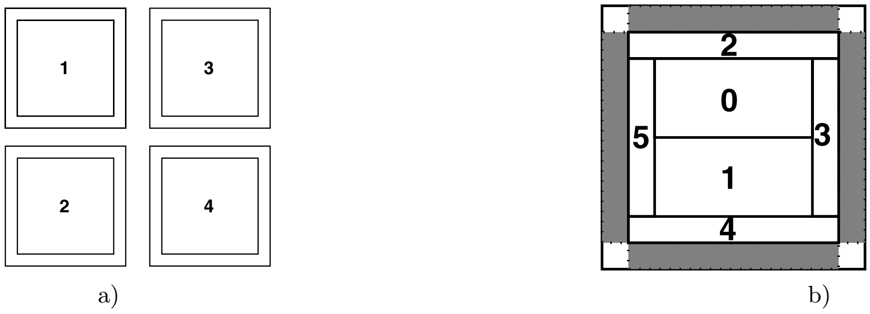

Fig. 1. (a) Cross section of a 3D problem partitioned across 4 nodes, showing the halo region; and (b) the node-level partitioning on dual-processor nodes. The halo is shaded in this depiction. The annular region abuts the halo, and is subdivided into pieces labeled 2 through 5. Points on the interior are labeled 0 and 1. This decomposition is duplicated on each node.

3 Testbeds

3.1 Hardware

We ran on two platforms, both developed by IBM: NPACI's Blue Horizon system[1], located at the San Diego Supercomputer Center, and the ASCI Blue Pacific Combined Technology Refresh (CTR)[2], located at Lawrence Livermore National Laboratory. The two platforms differ significantly in their respective on-node memory hierarchies. Blue Horizon provides significantly lower node bisection bandwidth than CTR relative to processor performance. The nodes are over an order of magnitude more powerful and have twice the number of processors. Blue Horizon's shared memory is multi-ported and employs a cross-bar interconnect rather than a bus. The cache lines are longer.

Blue Horizon contains 144 POWER3 SMP High Nodes (model number 9076-260) interconnected with a "Colony" switch. Each node is an 8-way way Symmetric Multiprocessor (SMP) based on 375 MHz Power-3 processors, sharing 4 Gigabytes of memory, and running AIX 4.3. Each processor has 1.5 GB/sec bandwidth to memory, an 8 MB 4-way set associative L2 cache, and 64 KB of 128-way set associative L1 cache. Both caches have a 128 byte line size.

Blue Pacific contains 320 nodes. Each node is a model number 9076-WCN 4-way SMP based on 332 MHz Power PC 604e processors sharing 1.5 GB memory and running AIX 4.3.1. Each processor has 1.33 GB/sec of bandwidth to memory, a 32 KB 4-way set associative L1 data cache with a 32 byte line size, and a 256KB direct-mapped, unified L2 cache with a 64 byte line size.

We used KAI's C++ and Fortran 77 compilers. These compilers are translators, and employ native IBM compilers to generate object code. C++ code was compiled `kai_mpCC_r`, with compiler options `--exceptions -O2`

[1] `http://www.npaci.edu/BlueHorizon/`

[2] `http://www.llnl.gov/asci/platforms/bluepac/`

`-qmaxmem=-1 -qarch=auto -qtune=auto --no_implicit_include`. Fortran 77 was compiled using `guidef77,` version 3.9, with compiler options `-O3 -qstrict -u -qarch=pwr3 -qtune=pwr3.`[3]

3.2 Performance Measurement Technique

We collected timings in batch mode: *Distributed Production Control System (DPCS)* on ASCI Blue Pacific, *loadleveler* on NPACI Blue Horizon. We report wall-clock times obtained with `read_real_time()` on Blue Pacific, and `MPI_Wtime()` on Blue Horizon.

The timed computation was repeated for a sufficient number of iterations to ensure that the entire run lasted for tens of seconds. Times were reported as the average of 20 runs, with occasional outliers removed. We define an outlier as running at least 25% more slowly than the average time of the other runs. In practice, we encountered outliers once or twice in each batch of twenty runs.

3.3 KeLP Software Testbed

The applications were written in a mix of C++ and Fortran 77 and used a multi-tier prototype of the KeLP infrastructure [1,2,4]. KeLP calls were made from C++, and all numerical computation was carried out in Fortran. A discussion of the KeLP API is out of the scope of this paper. The interested reader is referred to the above references for more information.

KeLP employs POSIX threads [7] to manage parallelism on node, and MPI [6] to handle communication between nodes. A typical KeLP program runs with one MPI process per node, and unfolds a user-selectable number of threads within each process. The total number of threads per node is generally equal to the number of processors.

KeLP employs a persistent communication object called a *Mover* [5] to move data between nodes. A distinguished master thread in each process is in charge of invoking the Mover, which logically runs as a separate task. Mover provides two entries for managing communication asynchronously: *start()* and *wait().*

KeLP provides two implementation policies for supporting asynchronous, non-blocking communication in the Mover. The Mover may either run as a proxy [12] within a separate thread, or it may be invoked directly by the master thread. In the latter case, asynchronous non-blocking MPI calls `MPI_Isend()` and `MPI_Irecv()` are relied on to provide overlap. However, we found that IBM's MPI implementation cannot realize communication overlap non-blocking asynchronous communication. Thus, we use only the proxy to realize overlap.

4 Results

4.1 Variant Policies

We implemented several variant policies, which are summarized in Table 1. The simplest variant, HAND, is hand-coded in MPI. This variant is typical of how

[3] On Blue Pacific we compiled with options `-qarch=auto -qtune=auto` in lieu of `pwr3`.

most users would implement RedBlack3D, and execution is single-tiered. All other variants were written in KeLP, and used the identical numerical Fortran 77 kernel.

Table 1. A synopsis of the policy variants used in the paper.

Variant	Explanation
HAND	Hand coded MPI (single-tier)
MT(K)	$k > 1$: Multi-tier implementation with k threads per process
MT(1)	Single-tier implementation written in KeLP
OLAP	Communication overlap
!OLAP	No communication overlap
XTRA	Use an extra thread (proxy) to overlap communication
IRR(M,N)	Irregular decomposition assigning work to two processor sets in the ratio m: n

The next variant is MT(P). It supports multi-tier execution using p computation threads per node. With $p = 1$, we flatten out the hierarchical machine interconnection structure. Thus, MT(1) reduces to single-tier execution, running 1 process per processor. When $p > 1$, we obtain a family of multi-tier variants.

We compose the overlap variant with MT(P). As discussed previously, we use a proxy to overlap communication with computation. To signify this overlap variant, we concatenate the policy XTRA using the + sign to indicate variant concatenation. Thus, the policy MT(P)+OLAP+XTRA employs multi-tier execution with p compute threads, and supports communication overlap using an extra thread running a proxy. We will use the variant !OLAP to indicate when we do not employ overlap.

4.2 Experimentation

We first present results for Blue Pacific CTR and then for Blue Horizon. We report all performance figures as the average number of milliseconds per iteration, and ran for 80 iterations. As noted previously, we report the average of 20 runs, ignoring outliers. On Blue Pacific CTR, we ran with a 480^3 domain on 64 nodes (256 processors). On Blue Horizon, we ran with 8 and 27 nodes (64 and 216 processors, respectively), keeping the problem size constant with the number of processors.

Establishing a Baseline. To establish the operating overheads of KeLP, we compare HAND against MT(1)+!OLAP. An iteration of MT(1)+!OLAP completes in 245 ms., including 116 ms of communication wait time. By comparison, HAND completes in 229 ms., including 99.3 ms of communication wait time. KeLP overheads are modest and incurred primarily in communication (15%).

Overall, the application runs just 7% more slowly in KeLP than in MPI. Having determined that KeLP's overheads are low, we will use the single-tier variant written in KeLP, MT(1)+!OLAP, as our baseline for assessing the benefits of multi-tier execution.

Multi-tier execution. We next run with MT(p) using OLAP and !OLAP variants.[4] To peform these runs, we employed the following AIX environment variable settings: `MP_SINGLE_THREAD=yes;AIXTHREAD_SCOPE=S`. Additionally, the OLAP variant ran with `MP_CSS_INTERRUPT=yes`. The !OLAP variant ran with
`MP_CSS_INTERRUPT=no; MP_POLLING_INTERVAL=2000000000`.

Compared with MT(1), MT(4)+ !OLAP reduces the running time slightly from 245 ms to 234 ms. Computation time is virtually unchanged. Communication time drops about 15%. We attribute the difference to the use of the shared memory cache-coherence protocol to manage interprocessor communication in lieu of message passing. Although Blue Pacific uses shared memory to resolve message passing on-node, communication bandwidth is about 80 Megabytes/sec regardless of whether or not the communicating processors are on the same node. As noted previously, bandwidth to memory is more than an order of magnitude higher: 1.33 GB/sec per processor. We are now running at about the same speed as hand-coded MPI. Our next variant will improve performance beyond the HAND variant.

Overlap. We next ran MT(3)+OLAP+XTRA. Performance improves by about 11% over MT(4)+ !OLAP: execution time drops to 209 ms. We are now running 17% faster than the single-tier variant. Communication wait time drops to 29.6 ms–a reduction of a factor of three. The proxy is doing its job, overlapping most of the communication. Since the proxy displaces one computational thread, we expect an increase in computation time. Indeed, computation time increases from 139 ms to 184 ms. This slowdown forms the ratio of 3:4, which is precisely the increase in workload that results results from displacing one computational thread by the proxy.

Although communication wait time has dropped significantly, it is still nonzero. Proxy utilization is only about 25% so this is not at issue. Part of the loss results from thread synchronization overhead. But load imbalance is also a significant factor. It arises in the computation over the inner annular region. The annulus is divided into six faces, and each face is assigned to one thread. (Faces that abut a physical boundary have 3, 4, or 5 faces.) Because faces have different strides–depending on their spatial orientation– the computation over the annulus completes at different times on different nodes. The resulting imbalances delay communication at the start of the next iteration. The time lag compounds over successive iterations, causing a phase shift in communication. When this phase shift is sufficiently long, there is not sufficient time for com-

[4] We did not run MT(1)+OLAP since the p extra proxy threads would interfere uselessly with one another.

munication to complete prior to the end of computation. We estimate that this phase shift accounts for 1/3 to 1/2 of the total wait time.

Tab. 2 summarizes performance of variants of HAND, MT(1)+!OLAP, MT(4)+!OLAP, and MT(3)+OLAP+XTRA.

Table 2. Execution time break-down for variants of redblack3D running on 64 nodes of ASCI Blue Pacific CTR. Times are reported in milliseconds per iteration. The column labeled 'Wait' reports the time spent waiting for communication to complete. The times reported are the maximum reported from all nodes; thus, the local computation and communication times do not add up exactly to the total time.

Variant	**Total**	**Wait**	**Comp**
HAND	229	99.3	147
MT(1) + !OLAP	245	116	142
MT(4) + !OLAP	234	100	139
MT(3) + OLAP + XTRA	209	29.6	184

Blue Horizon. Blue Horizon has has a "Colony switch," that provides about 400 MB/sec of message bandwidth under MPI for off-node communication, and 500 MB/sec on-node. We used AIX environment variables recommended by SDSC and IBM. For non-overlapped runs we used `#@ Environment = COPY_ALL; MP_EUILIB=us; MP_PULSE=0; MP_CPU_USAGE=unique; MP_SHARED_MEMORY=YES; AIXTHREAD_SCOPE=S; RT_GRQ=ON; MP_INTRDELAY=100;` for overlapped runs we added the settings `MP_POLLING_INTERVAL=2000000000; AIXTHREAD_MNRATIO=8:8`. With single-tier runs, the load leveler variable `tasks_per_node=8`. For MT(P) , $p > 1$, we used a value of 1. The number of nodes equals the number of MPI processes.

We ran on 8 and 27 nodes, 64 and 216 processors, respectively. We maintained a constant workload per node, running with a 800^3 mesh on 8 nodes, and a 1200^3 mesh 27 nodes. This problem size was chosen to utilize 1/4 of the nodes' 4GB of memory. In practice, we would have many more than the 2 arrays used in RedBlack3D (solution and right hand side), and would not likely be able to run with a larger value of N. Tab. 3 summarizes performance.

We first verify that KeLP overheads are small. Indeed, the KeLP (MT(1)+!OLAP) and HAND variants run in nearly the identical amount of time. The multi-tier variant MT(8)+ !OLAP reduces the running time from 732 ms to 713 ms on 8 nodes. Curiously the running time *increases* on 27 nodes, from 773 ms to 824 ms. The increase is in communication time–computation time is virtually unchanged. Possibly, external communication interference increases with a larger number of nodes, and is affecting communication performance. We are currently investigating this effect.

The benefits of multi-tier parallelism come with the next variant: communication overlap. MT(7) + OLAP runs faster than MT(1) + !OLAP, reducing

Table 3. Execution time break-down for variants of redblack3D running on 8 and 64 nodes of NPACI Blue Horizon, with N=800 and 1200, respectively. The legend is the same as the previous table. Threads were unbound except for MT(7) + Olap + Xtra + Irr(44:50). We were unable to run the HAND variant on 8 nodes due to a limitation in the code. We were unable to get speedups in the Irr variant on 27 nodes.

Variant	Total	Wait	Comp	Total	Wait	Comp
HAND	719	144	575	-	-	-
MT(1)+!Olap	732	163	569	773	195	578
MT(8)+!Olap	713	141	566	824	230	581
MT(7)+Olap+Xtra	655	20.5	608	693	42.5	669
MT(7)+Olap+Xtra+Irr(44:50)	626	9.23	607	-	-	-

execution time to 655 ms on 8 nodes, and 693 on 27 nodes. Overlap significantly reduces the wait time on communication, which drops from 141 ms to 20.5 ms on 8 nodes, and from 230 ms to 42.5 ms on 27 nodes. Our multi-tier overlapped variant MT(7) + Olap is about 10% faster than the single-tier variant MT(1) + !Olap. Although our strategy increases computation time, more significantly, it reduces the length of the critical path: communication.

An additional level of the memory hierarchy. Although we have reduced communication time significantly, there is still room for improvement. Upon closer examination, the workload carried by the computational threads on the interior of the domain is imbalanced. This imbalance is in addition to the imbalance within the annulus, which was discussed above.

The reason why is that the Power3 high node's shared memory is organized into groups of four processors and each group has one port to memory. Thus, when we run with seven compute threads, four of the threads sharing one port of memory see less per-CPU bandwidth than the other three threads sharing the other port. The uniform partitionings we used are designed to divide floating point operations evenly, but not memory bandwidth requirements.

The thread scheduler does a good job of mitigating the load imbalance, but at a cost of increased overheads. We can reduce running time further by explicitly load balancing the threads' workload assignments according the available per-processor bandwith. We use an irregular hierarchical partitioning. The first level divides the inner computational domain into two parts, such that the relative sizes of the two parts correspond to an equal amount of bandwidth per processor. We determined experimentally that a ratio of 44:50 worked best. That is, 44/94 of the 504 planes in the domain were assigned contiguously to 4 processors, and the remainder to the other 3 processors.

The irregular hierarchical improve performance, cutting the communication wait time in half to 9.2 ms. Overall computation time drops to 626 ms. We have now improved performance by 14.4% relative to the single-tier implementation.

As with ASCI Blue, it appears that the remaining losses result from thread synchronization overheads and from load imbalances arising within the annulus computation. The latter effect is more severe on Blue Horizon, which has 8 way nodes, than with Blue Pacific CTR, which has only 4-way nodes. To avoid large memory access strides in the annulus computation, we were limited to two-dimensional data decompositions. (Long strides, comprising thousands of bytes, penalize computation severely on unfavorably oriented faces–by a factor of 20!) No node received more than 4 annular faces. We can only utilize about half the 7 processors on Blue Horizon when computing on the annulus. The load imbalance due to the annulus computation introduces a phase lag of about 3% into the iteration cycle. Communication within the proxy consumes about 18%. Thus, after about 25 iterations, we can no longer overlap communication. Our runs were 40 cycles long.

5 Conclusions and Related Work

We have presented a set of performance programming techniques that are capable of reducing communication delays significantly on multi-tier architectures that employ a hierarchical organization using multiprocessor nodes. We realized improvements in the range of 10% to 17% for a 3D elliptic solver. A drawback of our approach–and others that employ *hybrid* programming–is to introduce a more complicated hierarchical programming model and a more complicated set of performance tradeoffs. This model has a steeper learning curve than traditional SPMD programming models, but is appropriate when performance is at a premium. Our data decompositions were highly irregular, and we were constantly fighting load imbalance problems. We suspect that dynamic workload sharing on the node would be easier to program and more effective in dealing with the wide range of architectural choices faced by users of multi-tier systems.

Other have incorporated hierarchical abstractions into programming languages. Crandall *et. al* [10] report experiences with dual-level parallel programs on an SMP cluster. Cedar Fortran [9] included storage classes and looping constructs to express multiple levels of parallelism and locality for the Cedar machine. The pSather language is based on a cluster machine model for specifying locality [11], and implements a two-level shared address space.

Acknowledgments

The authors wish to thank John May and Bronis de Supinski, with the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory, for the many illuminating discussions about the ASCI Blue-Pacific machine and with Dr. Bill Tuel, and David Klepacki, both with IBM, for explaining the subtleties of performance tuning in IBM SP systems.

KeLP was the thesis topic of Stephen J. Fink (Ph.D. 1998), who was supported by the DOE Computational Science Graduate Fellowship Program. Scott Baden is supported in part by NSF contract ACI-9876923 and in part by NSF contract ACI-9619020, “National Partnership for Advanced Computational Infrastructure.” Work on the ASCI Blue-Pacific CTR machine was performed under the auspices of the US Dept of Energy by Lawrence Livermore National Laboratory Under Contract W07405-Eng-48.

References

1. Fink, S. J.: Hierarchical Programming for Block–Structured Scientific Calculations. Doctor dissertation, Dept. of Computer Science and Engineering, Univ. of Calif., San Diego (1998)
2. Baden, S.B. and Fink, S. J.: Communication Overlap in Multi-tier Parallel Algorithms. In Proc. SC ’98, IEEE Computer Society Press (1998)
3. Fink, S. J. and Baden, S.B. Runtime Support for Multi-tier Programming of Block-Structured Applications on SMP Clusters. In: Ishikawa, Y., Oldehoeft, R, Reynders, J.V.W., and Tholburn, M. (eds.): Scientific Computing in Object-Oriented Parallel Environments. Lecture Notes in Computer Sci., Vol. 1343. Springer-Verlag, New York (1997) pp. 1–8
4. Fink, S. J. and Baden, S.B. A Programming Methodology for Dual-tier Multicomputers. *IEEE Trans. on Software Eng.,* 26(3), March 2000, pp. 212–26
5. Baden, S.B. and Fink, S. J., and Kohn, S. R. Efficient Run-Time Support for Irregular Block-Structured Applications. *J. Parallel Distrib. Comput.,* Vol 50, 1998, pp. 61–82
6. MPI Forum: The Message Passing Interface (MPI) Standard. `http://www-unix.mcs.anl.gov/mpi/index.html`, 1995
7. IEEE: IEEE Guide to the POSIX Open System Environment. New York, NY, 1995
8. Gropp, W.W. and Lusk, E. L. A Taxonomy of Programming Models for Symmetric Multiprocessors and SMP Clusters. In Giloi, W. K. and Jahnichen, S., and Shriver, B. D. (eds.): Programming Models for Massively Parallel Computers. IEEE Computer Society Press, 1995, pp. 2–7
9. Eigenmann, R., Hoeflinger, J., Jaxson,G., and Padua, D. Cedar Fortran and its Compiler, *CONPAR 90-VAPP IV, Joint Int. Conf. on Vector and Parallel Proc.*, 1990, pp. 288–299
10. Crandall, P. E., Sumithasri, E. V., Leichtl, J., and Clement, M. A. A Taxonomy for Dual-Level Parallelism in Cluster Computing, Tech. Rep., Univ. Connecticut, Mansfield, Dept. Computer Science and Engineering, 1998
11. Murer, S., Feldman, J., Lim, C.-C., and Seidel, M.-M. pSather: Layered Extensions to an Object-Oriented Language for Efficient Parallel Computation, Tech. Rep. TR-93-028, Computer Sci. Div., U.C. Berkeley, Dec. 1993
12. Lim, B.-H., Heidelberger, P., Pattnaik, P., and Snir, M. Message Proxies for Efficient, Protected Communication on SMP Clusters, in Proc. Third Int’l Symp. on High-Performance Computer Architecture, San Antonio, TX, Feb. 1997, IEEE Computer Society Press, pp. 116–27.
13. Woodward, P.R.. Perspectives on Supercomputing: Three Decades of Change, *IEEE Computer*, Vol. 29, Oct. 1996, pp. 99–111.
14. Sawdey, A. C., O’Keefe, M.T., and Jones, W.B. A General Programming Model for Developing Scalable Ocean Circulation Applications, *Proc. ECMWF Workshop on the Use of Parallel Processors in Meteorology*, Jan. 1997.
15. Somani, A. K. and Sansano, A. M. *Minimizing Overhead in Parallel Algorithms through Overlapping Communication/Computation,* Tech. Rep. 97-8, NASA ICASE, Langley, VA., Feb. 1997