A Comparison of the Intel iPSC/860 and the Suprenum-1 Parallel Computers

Oliver A. McBryan

CU-CS-499-90   November 1990

A COMPARISON OF THE INTEL iPSC/860 AND THE SUPRENUM-1 PARALLEL COMPUTERS

Oliver A. McBryan*

Department of Computer Science
University of Colorado
Boulder, CO 80309

ABSTRACT

We compare the Intel iPSC/860 and SUPRENUM-1 parallel computers using a well-known scientific application algorithm. The algorithm, the Shallow Water Equations, is frequently used as a model for both oceanographic and atmospheric circulation. We describe the steps involved in implementing the algorithm on the iPSC/860 and on the SUPRENUM-1, and we provide details of performance. Surprisingly we have found that the SUPRENUM-1 provides better performance on both a single node and multiple node basis, despite the much higher theoretical peak rate of the i860 nodes.

Using the latest Intel PGI compiler we have measured 4.63 Mflops per node on the iPSC/860, with efficiencies of 92.7% on 16 nodes, and 91.6% on 128 nodes. Earlier measurements with the standard Greenhills compiler yielded only 2.7 Mflops per node on the iPSC/860.

With an early version of the SUPRENUM-1 compiler we have measured 5.11 Mflops (64-bit arithmetic) for single node performance, and efficiencies of 88.3% with 16 nodes (72 Mflops aggregate performance). While a 256 node SUPRENUM-1 was not yet available for measurement, we provide a simple static load-balancing algorithm for hierarchical systems which effectively extends the observed single-cluster efficiency for grid algorithms to large systems.

* Research supported in part by the Air Force Office of Scientific Research, under grant AFOSR-89-0422

1. INTRODUCTION

The Shallow Water Equations are a standard model for atmospheric and oceanographic processes and implementations of the algorithm have been used as benchmarks for vector and parallel supercomputer performance for many years[1234]. The Shallow Water algorithm is very memory intensive, involving 14 variables per grid point, and accesses these using nine-point stencils, non-linear expressions and essential divisions. The combined effect provides a decidedly non-trivial test of any computer system. We have recently implemented the benchmark on the Intel iPSC/860 and SUPRENUM-1 MIMD parallel supercomputers and report on our experiences in this paper. The goal of the study was to estimate effective system performance for typical scientific users. For this reason we used only pure Fortran software, and have also minimized modifications to that software.

The Intel iPSC/860 hardware is described briefly in section 2. We ran our tests on a 128 node iPSC/860 at NASA-Ames Research Center, using both the Greenhills and the Portland Group (PGI) Fortran compilers. Since the PGI compiler delivered code that ran at almost twice the speed of the Greenhills code, we will present details only for PGI-compiled runs.

The SUPRENUM-1 architecture is described briefly in Section 3. We ran our tests on the SUPRENUM-1 prototype hardware at SUPRENUM GmbH which was running the Peace 3.0 operating system software. The prototype has only 16 processors, although a 256 processor system will be available in November 1990.

While the underlying hardware of the iPSC/860 and of the SUPRENUM-1 are quite different, the software environments for parallel programming are quite similar. In fact SUPRENUM supports iPSC communication library calls, allowing iPSC programs to run unchanged. We developed the SUPRENUM version of the application in this way, using an iPSC implementation that had been developed previously on an iPSC simulator[5].

As we will see, the Shallow Water equations ran at high parallel efficiency on both the iPSC/860 and SUPRENUM-1. Therefore single-node performance suffices to characterize overall performance. The Shallow Water code ran on a single iPSC/860 node at 4.63 Mflops and at 5.11 Mflops on a SUPRENUM-1 node (see Table 1). The performance on 16 nodes was 68.62 Mflops for the iPSC/860 and 72.14 Mflops for SUPRENUM-1. We also measured 542.64 Mflops on the full 128 node iPSC/860. Corresponding parallel efficiences were 92.7% and 91.6% for the iPSC/860 and 88.3% for the SUPRENUM-1. While we had access to only a 16-node SUPRENUM-1, a 256-node version will be available shortly.

Intel iPSC/860 parallel efficiencies are higher than for SUPRENUM due to the lower message startup costs of the iPSC/860. The new Intel PGI compiler (beta release, October 1990) showed a dramatic improvement over the Greenhills compiler, but the compiler is having real difficulties coping with the heavy memory access patterns of the

Shallow Water benchmark. In fact we tested very small grids on a single node - small enough to accommodate all variables at all grid points entirely in cache - but did not obtain significantly better performance. We did not attempt to develop assembly language routines for the critical subroutines, which would likely be the best way to achieve better performance.

SUPRENUM-1 performance of over 5.1 Mflops per node was quite surprising, especially as the code was not vectorized in any way. Two SUPRENUM characteristics that we had expected to affect performance are the message startup cost (2-3 msecs.) and the fact that a division requires 11 cycles. We conclude that the SUPRENUM compiler is doing an excellent job of locating vectorizable statements, and of generating efficient pipelined vector instructions to implement such statements. Numerical results agreed to high precision with those from other machines. We expect that even higher per-node performance could be achieved by utilizing explicit optimizations, and by coding computationally intensive segments using SUPRENUM Fortran's array extensions (Fortran 90).

Based on the performance observed in the above experiments, we would extrapolate SUPRENUM-1 performance to 1150 Mflops on a 16 cluster, 256 processor system. We comment on the validity of this extrapolation in section 8 where we provide a simple load-balancing algorithm to reduce inter-cluster communication overhead by a factor of 8. This would exceed iPSC/860 performance by more than a factor of 2, because with the current iPSC generation of hypercubes, the largest systems are constrained to 128 processors. Intel has however announced plans to develop rectangular grid arrays with much larger numbers of processors (Touchstone Delta).

2. THE SUPRENUM-1 SUPERCOMPUTER

The German SUPRENUM-1 computer couples up to 16 processor clusters with a network of 200 Mbit/sec busses. The busses are arranged as a rectangular grid with 4 horizontal and 4 vertical busses (*global busses*). Each cluster consists of 16 processors connected by a fast bus, along with I/O devices for communication to the global bus grid and to disk and host computers. There is a dedicated disk for each cluster. Individual nodes can deliver up to 20 Mflops (64-bit chained) or 10 Mflops (64-bit unchained) of computing power and support 8 Mbytes of memory. Each node includes both a Motorola 68020 and a Weitek-based vector processor. The high bandwidth of the bus network makes this an interesting machine for a wide range of applications, including those requiring long-range communication. No more than three communication steps are ever required between remote nodes.

SUPRENUM supports a send/receive model of communication which is not unlike that supported by Intel. The primary difference is that SUPRENUM Fortran is an extension of standard Fortran, in which task control and communication are incorporated into the language, rather than being implemented through library calls as on the iPSC. SUPRENUM also supports Fortran 90 array extensions which avail of the vector hardware. However we have not used the vector extensions in our experiments.

SUPRENUM software is characterized by the best support for scientific applications to be found among the various distributed memory MIMD vendors. The effort invested in development of libraries of high-level grid and communication primitives greatly eases the effort of moving applications to the computer, and also provides substantial high-level portability to other systems, since the communication library can be implemented in terms of low level primitives on any distributed system. In order to evaluate the effectiveness of both systems in the same way we have not used the SUPRENUM grid library in the studies reported on here.

3. THE INTEL iPSC/860 SUPERCOMPUTER

In fall 1989 Intel announced an i860-based version of the well-known iPSC/2 hypercube. The iPSC/860 systems are basically standard iPSC/2 hypercubes with the node processors replaced by Intel i860 processors. In terms of raw floating point performance the peak rate is thereby increased by a factor of 10 over even the VX vector board available with the iPSC/2 - to 60 Mflops. In practice it is unlikely that more than 40 Mflops can be realized due to the memory model used by the i860. Some simple vector type kernels, hand-coded in assembler, are currently running at from 28 to 38 Mflops/node. Well-designed Fortran programs currently yield about 5-10% of peak due to the poor state of the i860 Fortran compilers. Several major i860 compiler efforts are underway and will undoubtedly improve substantially on the early results. Because the communication facilities of the iPSC/860 are those of the iPSC/2, the system is constrained to a maximum of 128 nodes.

While the iPSC/860 utilizes the slow iPSC/2 communication hardware and software, communication proves to be much faster on the i860 system than on the iPSC/2. This is because most of the message startup communication overhead is software overhead involved in negotiating the communication protocol. Because the i860 is so much faster than the 80386, the software overhead is correspondingly decreased. The effect is to reduce messaging time by about a factor of three.

Intel has also announced plans to develop a rectangular grid version of the iPSC/860. There will be 8 communication paths per node, allowing 4 bidirectional channels as required for a two-dimensional grid. With the new communication structure, the iPSC will be freed from the constraint of a maximum of 128 nodes. Indeed Intel has announced plans to build a 2048 processor version of the iPSC.

4. THE SHALLOW WATER EQUATIONS BENCHMARK

We will describe the implementation of a standard two-dimensional atmospheric model - the Shallow Water Equations - on the Intel iPSC/860 and SUPRENUM-1 computers. These equations provide a primitive but useful model of the dynamics of the atmosphere. Because the model is simple, yet captures features typical of more complex codes, the model is frequently used in the atmospheric sciences community to benchmark computers[12]. Furthermore, the model has been extensively analyzed mathematically and numerically[67].

The shallow water equations, without a Coriolis force term, take the form

$$\frac{\partial u}{\partial t} - \zeta v + \frac{\partial H}{\partial x} = 0\ ,$$
$$\frac{\partial v}{\partial t} - \zeta u + \frac{\partial H}{\partial y} = 0\ ,$$
$$\frac{\partial P}{\partial t} + \frac{\partial Pu}{\partial x} + \frac{\partial Pv}{\partial y} = 0\ ,$$

where u and v are the velocity components in the x and y directions, P is pressure, ζ is the vorticity: $\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$ and H, related to the height field, is given by: $H = P + (u^2 + v^2)/2$. It is required to solve these equations in a rectangle $a \le x \le b$, $c \le y \le d$. Periodic boundary conditions are imposed on u, v, and P, each of which satisfies $f(x+b,y) = f(x+a,y)$, $f(x,y+d) = f(x,y+c)$.

A scaling of the equations results in a slightly simpler format. Introduce mass fluxes $U=Pu$ and $V=Pv$ and the potential velocity $Z=\zeta/P$, in terms of which the equations reduce to:

$$\frac{\partial u}{\partial t} - ZV + \frac{\partial H}{\partial x} = 0\ ,$$
$$\frac{\partial v}{\partial t} + ZU + \frac{\partial H}{\partial y} = 0\ ,$$
$$\frac{\partial P}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0\ .$$

5. DISCRETIZATION

We have discretized the above equations on a rectangular staggered grid with periodic boundary conditions. The variables P and H have integer subscripts, Z has half-integer subscripts, U has integer and half-integer subscripts, and V has half-integer and integer subscripts respectively.

Initial conditions are chosen to satisfy $\vec{\nabla}\cdot\vec{v} = 0$ at all times. We time difference using the Leap-frog method. We then apply a time filter to avoid weak instabilities inherent in the leap-frog scheme:

$$F^{(n)} = f^{(n)} + \alpha\,(f^{(n+1)} - 2f^{(n)} + f^{(n-1)})\ ,$$

where α is a filtering parameter. The filtered values of the variables at the previous time-step are used in computing new values at the next time-step. For a complete description of the discretization we refer to[1].

6. SERIAL FORTRAN IMPLEMENTATION

The Fortran code implementing the above algorithm involves a 2D rectangular grid with variables: $u(i,j)$, $v(i,j)$, $p(i,j)$, $z(i,j)$, $psi(i,j)$, $h(i,j)$. Here the first index represents the y direction. There are three main loops, two corresponding to the leap-frog time propagation of various quantities, and one for the filtering step. Execution of these three loops completes a single time step, which is then repeated until the desired temporal simulation interval has been achieved. A typical code sequence, used in the updating of the U, V and P variables, is:

```
      do 200 j=1,n
      do 200 i=1,m
        unew(i+1,j) = uold(i+1,j)+
                   tdts8*(z(i+1,j+1)+z(i+1,j))*(cv(i+1,j+1)+cv(i,j+1)+cv(i,j)
                   +cv(i+1,j))-tdtsdx*(h(i+1,j)-h(i,j))
        vnew(i,j+1) = vold(i,j+1)-tdts8*(z(i+1,j+1)+z(i,j+1))
                   *(cu(i+1,j+1)+cu(i,j+1)+cu(i,j)+cu(i+1,j))
                   -tdtsdy*(h(i,j+1)-h(i,j))
        pnew(i,j) = pold(i,j)-tdtsdx*(cu(i+1,j)-cu(i,j))
                   -tdtsdy*(cv(i,j+1)-cv(i,j))
  200 continue
```

Each such loop is followed by code to implement the periodic boundary conditions. Note that there are such loops for both the horizontal and vertical boundaries, and in addition some corner values are copied as single items.

Excluding the boundary computations, the three major loops in a time step involve 65 arithmetic operations per grid point (with division counted as a single operation). The complexity of the memory access patterns required to implement the above loop explain the difficulty the iPSC/860 compilers have in providing good performance[5].

7. iPSC/860 AND SUPRENUM IMPLEMENTATIONS

To speed the implementation effort we decided to test the idea of porting a generic MIMD parallel version of the Shallow Water Equations to the SUPRENUM-1. The work was based on a parallel code developed by McBryan and Pozo[5]. Actually the code was developed for a generic class of MIMD parallel computers, based on the assumption of a single process per node model. The code was developed and tested using a simulator for the generic model[89]. The simulator supports both the Intel iPSC/1 and iPSC/2 communication protocols. The code developed for the simulator was used without change on the iPSC/860.

SUPRENUM supports a library interface allowing both Intel iPSC/1 and iPSC/2 communication interfaces to be utilized. It suffices to declare the main program of both the host and node processes to be SUPRENUM **tasks**, while the rest of each program may remain as a pure Intel iPSC program. This approach greatly reduces code modifications that would be required to develop a complete SUPRENUM-1 implementation from scratch. In fact the code was ported and fully compiled within a few hours. The program ran immediately and gave correct results on the first try. This demonstrates the advantages of developing MIMD codes initially using simulators, and transferring to hardware only when the simulations are running correctly.

Since the algorithm involves rectangular grid arrays, and a nine-point stencil, the parallelization of the code is straightforward. A logical mapping of the processors to a two dimensional array is selected. Thus if $P = P_x P_y$, is a factorization of the number of processors P, then we regard the processors as arranged in a $P_x \times P_y$ logical grid. The large arrays representing physical variables (u, v, etc.) are then decomposed into equal sized blocks, with one block assigned to each processor. For simplicity we assume that the x and y grid dimensions are exact multiples of the corresponding processor numbers P_x and P_y. Each such block is then stored in an array of the same shape, but which has an extra boundary row or column provided on each of the four sides. These extra boundary points are used to maintain copies of the true (i.e. interior) boundary points of the four neighboring processors. The three main loops of the time step are decomposed into equivalent loops performed by each processor on the interior points of the block assigned to that processor. Following each loop, the boundary values are updated by copying the appropriate values from neighboring processors, following a synchronization to ensure that all neighbors have completed changes. Because the current application requires periodic boundary conditions, the logical processor grid is defined to be periodic - i.e. is a torus.

There is an essential simplification that occurs in the case that either P_x or P_y is 1 - in which case the logical rectangular processor array reduces to a line of processors, or a ring in the case of periodic boundary conditions. In this case two of the four communications required within each main loop are not needed, reducing substantially the communication overhead. As mentioned previously, the Shallow Water code uses periodic boundary conditions in each dimension. Normally periodic boundary conditions require copying data between processors at opposite edges of the processor array. In the case that one or other of P_x or P_y is 1, the periodic boundary condition in the corresponding dimension may be implemented by in-memory copying, rather than by communication.

A final optimization of the communication structure was required to get the peak performance. Before each of the main loops in the algorithm, the boundary data for the various physical variables (P,U,V,Z,H) used in that loop need to be copied from neighboring processors. Typically two or three variables are needed from a specific direction, although the number needed may depend on the direction. Because of the high communication startup cost of SUPRENUM-1 (at least 2 msecs), and the moderately high startup cost on the iPSC/860 (.3 msecs), it is essential to limit the number of individual communication requests. This was accomplished by packaging several communications of different physical variables in a single direction into one large communication packet. For some steps this reduced startup overhead by a factor of three.

In the final SUPRENUM-1 implementation we also replaced the Intel iPSC/2 communication calls by explicit calls to SUPRENUM Fortran equivalents, thereby saving an extra copying of each data array to a communication buffer. SUPRENUM Fortran supports explicit I/O statements for send/receive communication which allow multiple arrays to be communicated in a single operation.

8. AN ALGORITHM TO INCREASE MULTI-CLUSTER EFFICIENCY

Because multi-dimensional grids are easily imbedded in hypercubes, iPSC/860 parallel efficiency remains essentially constant with increasing number of processors. This is borne out by our measurements in the following section.

SUPRENUM-1 is however an hierarchical architecture (with nodes as the bottom level, and clusters as the second level), and consequently one can expect that single-cluster efficiencies will drop as one goes to multi-cluster systems due to the increased communication overhead in traversing the bus grid that interconnects the top level of the hierarchy. We now show that for multi-dimensional rectangular grid algorithms, these extra overheads may be trivially reduced by a factor of $C/2$, where C is the number of nodes in a cluster.

First we note that with strip rather than square 2D subgrids, communication can be restricted entirely to one grid direction (x direction to ensure that the Fortran column direction, y, is within single processors). By assigning processors of each cluster to consecutive intervals of the x direction, only the first and last processor in a cluster will ever send data out of the cluster, while the remaining processors simply exchange data with their left and right neighbors. Thus communication between clusters will be extremely simple, and in fact the first and last processor of each cluster could be assigned a smaller amount of work to perform, to balance for the increased communication time to be expected at those nodes. The same comments apply in 3D, provided that the word "plane" replaces "column".

Consider the horizontal dimension of a grid which has been decomposed so that each cluster contains n columns (or planes), with n_1 columns stored in "interior" nodes of the cluster and n_2 columns in the first and last nodes. Let c_1 denote the communication time to send a column between nodes in a cluster, and c_2 denote the *extra* time required to communicate between clusters. If a denotes the time to perform the required arithmetic computation on one column, then we may describe the per-node elapsed time of interior and boundary nodes (assuming no communication overlap) as:

$$t_1 = an_1 + 2c_1 , \quad t_2 = an_2 + 2c_1 + c_2 ,$$

Imposing load balancing requires that $t_1 = t_2$. In addition we have a constraint on the number of columns in a cluster: $n = (C-2)n_1+2n_2$. These equations lead to the result:

$$t_1 = a*n/C + 2c_1 + 2c_2/C .$$

Note that $t = a*n/C + 2c_1$ is the per-node elapsed time assuming only a single cluster is in use. Thus load balancing allows effective time to be increased by only $2c_2/C$, rather than the c_2 which we would increase by without load balancing. In the case of SUPRENUM-1, where C is 16, this allows a decrease by a factor of 8 in the costs of inter-cluster overhead.

The arguments above are independent of the dimensionality of the grid, provided it is decomposed along only one direction. Of course such a decomposition typically requires more data transfer than for sub-square or sub-cube decompositions, although this may be balanced by the efficiencies of longer vector length and fewer communication startups. Finally we note that a similar load-balancing could be used for grids that are decomposed into sub-squares or sub-cubes, but to much less effect. In the sub-square case 12 of the 16 processors in a SUPRENUM cluster would incur the increased overhead, allowing only a minor improvement in performance from load balancing. For very large clusters, one could gain a factor of $O(\sqrt{C})$ decrease in inter-cluster overhead for sub-squares, or $O(C^{1/3})$ for sub-cubes, using similar arguments.

9. PERFORMANCE RESULTS: iPSC/860 vs. SUPRENUM-1

All measurements were performed on a 128 processor iPSC/860 and on a 16-processor SUPRENUM-1 cluster (no larger SUPRENUM system was available). The Shallow Water code was exactly the standard sequential code, modified only to take account of communication as described in the previous section.

On SUPRENUM-1, no attempt was made to introduce Fortran 90 vectorization constructs, or to otherwise adapt the code to known features of the SUPRENUM compiler. The code was compiled with both the vectorizer and optimizer switches (options "-opt -vec -strength -strip_length"). Similarly, on the iPSC/860 we did not attempt to write i860 assembler code. The Fortran was compiled with the i860 PGI compiler (beta release) using the options "-O3 -W0,nodepchk -Mvect". All arrays were arranged to be fully visible to the compiler - i.e. the compilers were aware of the static dimensions.

We present the measured results in Table 1. The table indicates the number of processors P, their arrangement as a logical $P_x \times P_y$ rectangular processor array, the computational domain size $M_x \times M_y$ and the resulting per processor sub-grid domain size $N_x \times N_y$.

The iPSC/860 performed at 4.63, 68.62 and 543.64 Mflops respectively on 1, 16 and 128 processors. Corresponding multiprocessor efficiencies were 92.7% and 91.6%.

The SUPRENUM-1 performed on a 128×128 grid at 4.10 Mflops on a single processor. Performance on a 16-node cluster was measured at 72.14 Mflops on a 512×512 grid. This corresponds to approximately 108% efficiency relative to linear scaling of the single processor results. Thus we appear to have the unusual situation of super-linear speedup.

The reason we have observed superlinear speedup is that the standard Shallow Water benchmark is a square grid program. Thus the obvious parallel test would be to solve the equations on as large a square grid as the machine accommodates. On a single processor there is only one possibility: a 128×128 grid. However on 16 processors, while total grid size is restricted to the 512×512 square, one now has some leeway as to how to decompose the grid: as horizontal, or vertical strips, or sub-squares for example. Depending on the choice, the sub-grids stored in the individual processors may be non-square. The most favorable sub-grids for SUPRENUM-1 are vertical strips, since these provide the longest vector length within individual processors. Indeed we see from the table that the best single-node performance is realized for that case: 32×512 sub-grids. Vertical strips also have another advantage: they require communication in only two rather than four directions.

To make a really fair comparison to a single processor one should therefore consider rectangular grids for the single processor case: in the second SUPRENUM-1 line of the table we include the 1 processor performance for a grid of the same dimensions as the optimal subgrids for 16 processors. The resulting 1-grid performance increases to

5.11 Mflops. If this value is used to measure 16-processor efficiency one arrives at a quite respectable, but sublinear, value of 88.3% efficiency.

Alternatively one could make the comparison of 1 and 16 processor results using square *sub-grids* in both cases. As seen from the table the resulting performance is 56.89 Mflops, corresponding to an efficiency relative to the 1 processor square grid of 86.6%, which is totally consistent with the conclusion based on skewed grids.

For the iPSC/860 we also investigated the effect of varying the single-processor domain shape. We found that the best performance occurred with grids that were close to square. In fact long vertical strips of the type that are optimal for SUPRENUM-1 gave very poor performance on the iPSC/860 (off by a factor of 2.5). The iPSC/860 nodes therefore definitely do not appear as vector processors with the current compilers. For a more detailed discussion of these issues we refer to[5].

TABLE 1: SHALLOW WATER PERFORMANCE ON iPSC/860 AND SUPRENUM-1

SYSTEM	P	Px	Py	Mx	My	Nx	Ny	Mflops
iPSC/860	1	1	1	128	128	128	128	4.502
iPSC/860	1	1	1	256	64	256	64	4.628
SUPRENUM-1	1	1	1	128	128	128	128	4.104
SUPRENUM-1	1	1	1	32	512	32	512	5.107
iPSC/860	16	16	1	512	512	32	512	62.763
iPSC/860	16	2	8	512	512	256	64	68.616
SUPRENUM-1	16	4	4	512	512	128	128	56.889
SUPRENUM-1	16	8	2	512	512	64	256	64.103
SUPRENUM-1	16	16	1	512	512	32	512	72.142
iPSC/860	128	16	8	2048	1024	128	128	542.638

The SUPRENUM-1 lines in the table above involve at most 16 processors. Based on these numbers, we extrapolate to a performance of about 1150 Mflops (64-bit) on a full 256 processor SUPRENUM-1. In practice performance may drop somewhat relative to the above extrapolation due to the need to access the global communication bus when using more than 16 processors. As pointed out in the previous section, such overheads are easily reduced by a factor of 8 by using a static load-balancing approach.

Furthermore since message startup costs are dominating the communication overhead, we suspect that the slower speed of the global bus will not have a major impact. While this may prove overly optimistic, there will be potential gains from improvements to the SUPRENUM Fortran which may well balance any communication losses (in fact a new SUPRENUM compiler was released while this paper was being prepared, but was not used here). Another feature enhancing potential 256 processor performance is the fact that when a larger 2048×2048 grid is decomposed, the optimal decomposition by strips of size 8×2048 will result in better vectorization of the node programs due to longer vector lengths.

ACKNOWLEDGEMENTS

We would like to thank NASA-Ames Research Center for providing access to their iPSC/860. We would also like to thank Dr. S. Griffin and Dr. D. Scott of Intel Scientific Computers for helpful discussions about iPSC/860 optimization of the Shallow Water equations.

We would like to thank SUPRENUM GmbH for providing access to the SUPRENUM-1 prototype, and especially Dr. R. Vogelsang, of SUPRENUM GmbH, for facilitating access to the Intel iPSC/1 library emulation on the SUPRENUM-1. We also thank M. Lemke and W. Krotz-Vogel of SUPRENUM GmbH for helpful comments concerning efficiency issues.

References

1. G.-R. Hoffman, P.N. Swarztrauber, and R.A. Sweet, ''Aspects of using multiprocessors for meteorological modeling,'' in *Multiprocessing in Meteorological Models*, ed. D. Snelling, pp. 126-195, Springer-Verlag, Berlin, 1988.
2. O. McBryan, ''New Architectures: Performance Highlights and New Algorithms,'' *Parallel Computing*, vol. 7, pp. 477-499, North-Holland, 1988.
3. O. McBryan and R. Pozo, ''Performance Evaluation of the Myrias SPS-2 Computer,'' CS Dept Technical Report, University of Colorado, Boulder, 1990.
4. O. McBryan and R. Pozo, ''Performance Evaluation of the Evans and Sutherland ES-1 Computer,'' CS Dept Technical Report, University of Colorado, Boulder, 1990.
5. O. McBryan and R. Pozo, ''Performance of the Shallow Water Equations on the Intel iPSC/860 Computer,'' CS Dept Technical Report, University of Colorado, Boulder, 1990.
6. R. Sadourny, ''The dynamics of finite difference models of the shallow water equations,'' *JAS*, vol. 32, pp. 680-689, 1975.

7. G.L. Browning and H.-O. Kreiss, ''Reduced systems for the shallow water equations,'' *JAS*, to appear.

8. O. McBryan and E. Van de Velde, *Hypercube Algorithms and Implementations*, SIAM J. Sci. Stat. Comput., 8, pp. 227-287, 1987.

9. O. McBryan, ''Software Issues at the User Interface,'' in *Frontiers of Supercomputing II: A National Reassessment*, ed. W.L. Thompson, MIT Press, to appear.