Transitioning from MPP to SMP: Experiences with a Navier-Stokes solver

Chris N. Hill
Center for Meteorology and Physical Oceanography, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. (e-mail: cnh@plume.mit.edu)

Andrew Shaw
Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. (e-mail: shaw@abp.lcs.mit.edu)

High-end, mainstream symmetric multiprocessor (SMP) computers are beginning to offer numerical modelers realistic alternatives to traditional vector or massively parallel processing (MPP) supercomputers. We report on the implementation of the MIT ocean-atmosphere dynamical kernel on a DEC 8400 SMP. The SMP implementation is contrasted with an existing optimized version of the same model on the CM-5 parallel vector supercomputer. We discuss the numerical methods and programming styles appropriate to the two machines. To characterize the differences in the architectures, we compare the performance of key kernels of the model which are typical of those in many computational fluid dynamics (CFD) codes. To render the model readily accessible to individual researchers and students, it is written in High Performance Fortran, which emphasizes both high performance and a high-level programming environment. Our study highlights cache memory architecture and compilation systems as particularly critical technologies in determining the overall utility of a mainstream SMP system for the oceanographic and meteorological community. Notwithstanding these caveats, we demonstrate that the performance of "high-end mainstream" SMP's is rapidly converging on that of "low-end" computer center facilities for real scientific applications.

1 Introduction

During the last twenty years, the application of supercomputers to the numerical study of fluids, in particular the atmosphere and ocean, has been an unquestionable success, paving the way for vast improvements in the fidelity of computational fluid simulations. Throughout this period, low-volume, high-end vector supercomputers have clearly led in absolute performance and vector computers have been an indispensable tool in meeting the seemingly insatiable demands of computational fluid dynamics (CFD). Recently, however, commodity microprocessor-based multiprocessors have reached a level of performance where they can be seriously considered as an affordable alternative to specialized, purpose-built, and hence generally expensive, vector supercomputers.

Today there is a widely articulated viewpoint that mainstream symmetric multi-processor (SMP) architectures, based on the same commodity micropro-

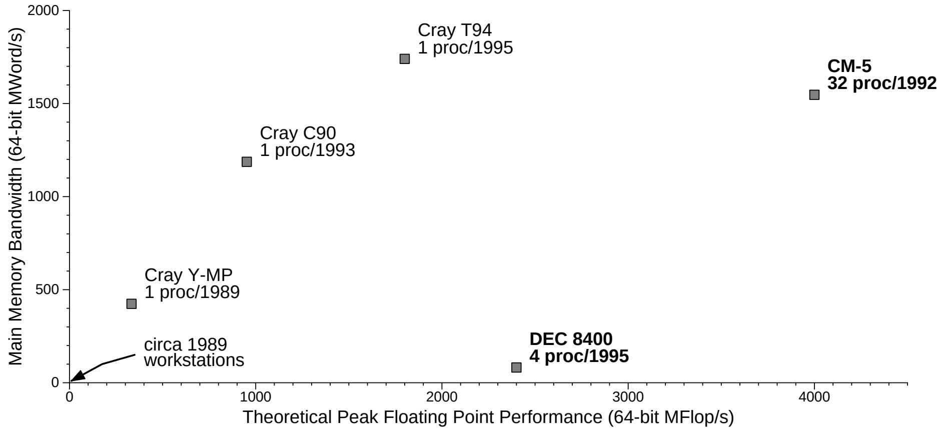

Figure 1: Peak floating point performance versus main memory bandwidth. Peak floating point performance is the maximum floating point operations which can be issued in a cycle multiplied by the cycles per second. Main memory bandwidth is measured using the STREAM benchmark.

cessor technology used in desktop workstations, will become the building blocks for a new generation of high-end, scalable supercomputer platforms. Computer architects now enthuse over the possibility of multiprocessor supercomputers based on commodity microprocessors delivering significant performance to numerical applications at a fraction of the cost of traditional vector supercomputers. Skeptics question the soundness of this conviction, arguing that vector processing is a natural paradigm for core supercomputer fields such as computational fluid dynamics, and that many legacy applications do not perform well on current generation commodity platforms[1,2,3]. The skeptic's argument is compelling. Current generation commodity microprocessor platforms all rely on hierarchical cache-based memories to give them high performance – however, this memory organization performs poorly with the vector-centric numerical procedures that lie at the heart of many supercomputer applications, including most meteorological and oceanographic applications.

Figure 1 illustrates one important architectural difference between vector, massively parallel processor (MPP), and mainstream workstation/SMP platforms. The Cray vector computers (Y-MP, C90, and T94) are a yardstick for the performance of vector supercomputers. They show a close to one-to-one balance between main memory bandwidth (as measured by the STREAM benchmark[3]) and peak floating point performance, with both increasing proportionately over time. Such a balance makes it possible for Cray and other similar vector supercomputers to fetch an array element and to perform a floating point operation on an array element each cycle.

The two platforms we consider in this paper, a 32-processor CM-5 and a 4-processor DEC 8400/300, are also shown on Figure 1. The CM-5 is a vector MPP supercomputer, and accordingly it shows a one-to-three balance between main memory bandwidth and floating point performance – not quite as evenly balanced as the Cray vector computers, but significantly more balanced than cache-based workstations or SMP's. As the CM-5 scales to larger configurations, the balance remains roughly fixed. The DEC 8400, on the other hand, shows comparable peak floating point performance to even the current-generation of Cray vector machine, but has significantly lower main memory bandwidth. The DEC 8400 has about a one-to-thirty balance between floating point and main memory performance.

Generally, oceanographic and atmospheric models, either grid-point or spectral, are relatively straightforward to map efficiently to architectures like the Crays, with their even balance between main memory bandwidth and floating point performance. Typically, the models use low-order finite difference schemes, which can be expressed naturally using vector-processor-friendly constructs with a roughly equal ratio of floating point operations to memory references. As Figure 1 shows, such constructs are not well-suited to the DEC 8400 or other cache-based microprocessor platforms, because the main memory bandwidth soon becomes the limiting factor.

Nevertheless, there are numerical codes for which the performance of a 4-processor DEC 8400 is comparable with a top-of-the-line vector platform. The LINPACK[4,5] benchmark rates a 4-processor DEC 8400/300 at 1351 MFlop/s and a 1-processor Cray T94 at 1603 MFlop/s, for the hand-optimized LINPACK 1000 test. A workstation of the same era as the Cray Y-MP, on the other hand, had both low floating point and low main memory performance, and would have been unsuitable for any form of large-scale computational fluid dynamics. The rapidly increasing performance of mainstream microprocessor computers, along with their relatively low costs, makes it tempting to investigate the role they might play in large-scale meteorological and oceanographic modeling, and to anticipate how that role might evolve in the future.

In this paper, we report on the design, implementation and evaluation of the hydro-dynamical kernel of the MIT ocean-atmosphere model on a mass-market SMP platform, the DEC 8400 5/300. We compare this implementation with an existing implementation on a CM-5 MPP currently used for daily production runs of the MIT model. A vision of "personal supercomputers" as effective research and teaching tools underlies this study, and so we limit our experiments to high-level programming environments that would be readily accessible to researchers and students who are not necessarily expert scientific programmers or numerical modelers. Our study employs a hierarchy of problems, culminating in the MIT model hydro-dynamical kernel, to probe the strengths and weaknesses of the platforms, and to identify promising implementation strategies.

2 The MIT Model

The MIT ocean-atmosphere model is built around a versatile incompressible Navier-Stokes solver developed at MIT for the numerical study and analysis of rotating fluids. A hydro-dynamical kernel, designed to be sensitive to the transition from non-hydrostatic to hydrostatic dynamics, lies at the heart of the model, providing computational efficiency across a broad spectrum of simulation regimes. A review of the model's utility for research in a range of problems related to ocean and climate dynamics can be found in Marshall et.al.[6]. Implementations of the model targeted to the MPP CM-5 and to CRAY vector supercomputers are being actively used by researchers at MIT[7,8,9,10,11,12,13].

The discrete algorithm uses a finite volume[14], time-stepping formulation which is rich in data parallelism and can handle the intricate, order one variations in geometry of ocean basins. The CM-5 code achieves sustained floating point throughput of over 10 GFlop/s on a 512 node CM-5. Daily production runs of the model sustain between 500 MFlop/s and 1 GFlop/s on a 32 node CM-5, and between 100 MFlop/s and 200 MFlop/s on a CRAY Y-MP.

2.1 The numerical scheme

The numerical procedure used in the model is described in detail in Marshall et.al.[15] The scheme is a variant on the theme set out by Harlow and Welch[16], in which a pressure correction to the velocity field is used to guarantee non-divergence.

The numerical kernel is a discrete form of the Boussinesq incompressible Navier Stokes fluid equations: it can be written in semi-discrete form to second order in time Δt thus:

$$\frac{\underline{v}^{n+1} - \underline{v}^{n}}{\Delta t} = \underline{G}_v^{n+\frac{1}{2}} - \nabla p^{n+\frac{1}{2}} \tag{1}$$

$$\nabla . \underline{v}^{n+1} = 0 \tag{2}$$

Equations 1 and 2 describe how the fluid evolves ($\underline{v} = (\underline{v}_h, w)$ is the velocity in the horizontal and the vertical) in response to forcing due to $\underline{G}$ (representing inertial, Coriolis, metric, gravitational, and forcing/dissipation terms) and to the pressure gradient ∇p. The same kernel algorithm is used by us to study both the atmosphere and the ocean - the anelastic approximation as described in Brugge et.al.[17] and in Miller[18] is used to render the same model suitable for the study of both fluids.

The pressure field required in Equations 1 and 2 is found by separating p into hydrostatic, surface and non-hydrostatic parts. Vertical variations in p can be computed hydrostatically yielding p_{hy} using only information in each column of the fluid. Horizontal variations in p_s are diagnosed by solving an elliptic equation which ensures non-divergent depth integrated flow:

$$\nabla_h . H \nabla_h p_s = \nabla_h . \overline{\underline{G}_{v_h}^{n+\frac{1}{2}}}^H - \nabla_h . \overline{\nabla_h p_{hy}{}'}^H \tag{3}$$

($\overline{\quad}^H$ indicates the vertical integral over the local depth, H, of the domain and the subscript $_h$ denotes horizontal). In the non-hydrostatic limit a three-dimensional elliptic equation is inverted for the non-hydrostatic pressure. In this study we consider the hydrostatic case involving the two-dimensional problem, Equation 3. More details of the numerical procedure can be found in Marshall et.al.[6,15].

2.2 Program structure

Figure 2 shows the high-level program structure of the MIT model. Almost all of the execution time is spent in the time-stepping loop, which is divided into two parts – **Forward_Step** and **Inversion_Step**. The **Forward_Step** comprises a series of nearest-neighbor computations involving two time levels of the model state variables and their time derivatives. This stage computes explicit time derivative terms for the velocity and tracer equations, steps them forward in time and updates the density field through an equation of state. There is considerable latitude for different orderings of the computations within **Forward_Step**, and, as discussed in Shaw et.al.[19], this stage contains an abundance of both data parallelism and functional parallelism. The **Inversion_Step** requires global communication to solve a set of simultaneous equations to ensure that the flow remains non-divergent.

3 The computer platforms

We compare the implementation of the MIT model on two very different platforms: the CM-5 vector MPP, and the DEC 8400 cache-based SMP. The CM-5

```
BEGIN
  Initialize
    Define topography, initialize flow field, tracer distributions
  FOR each time step DO
    Forward_Step
      Calculate velocity time derivatives
      Calculate tracer field time derivatives
      Step forward state variables
      Update density field
    Inversion_Step
      Solve elliptic problem to ensure non-divergent flow
  END FOR
END
```

Figure 2: High-level structure of the MIT model. The model iterates repeatedly over a time-stepping loop, comprised of two main blocks, **Forward_Step** and **Inversion_Step**.

is well-suited for scientific simulation applications, and it, along with the Cray Y-MP, is the mainstay of our day-to-day production runs. We will be testing a 32-processor configuration of the CM-5.

The DEC 8400 is a very high-end mainstream SMP system, designed for use in a broad variety of both technical and non-technical situations. Digital has already sold several thousand 8400 systems, while in contrast, Thinking Machines shipped less than 100 CM-5 systems. We will be testing a 300 MHz, 4-processor configuration of the DEC 8400.

3.1 Machine Architectures

Both systems have theoretical peak floating point performance well in excess of 100 MFlop/s, as illustrated in Figure 1. However, the two systems have radically different architectures, which we discuss here in more detail.

The CM-5 MPP[20] was one of the first parallel computers which could routinely out-perform traditional vector supercomputers for real scientific problems. The machine is built from relatively modest vector processors connected through two communications networks, as shown in Figure 3. Each vector unit runs at 16 MHz and can issue a multiply-add instruction each cycle, giving it a peak performance of 32 MFlop/s. Each CM-5 node contains 4 vector units, yielding a peak performance of 128 MFlop/s per node. In real applications, however, sustained performance of 10-30 Mflop/s per node is more typical.

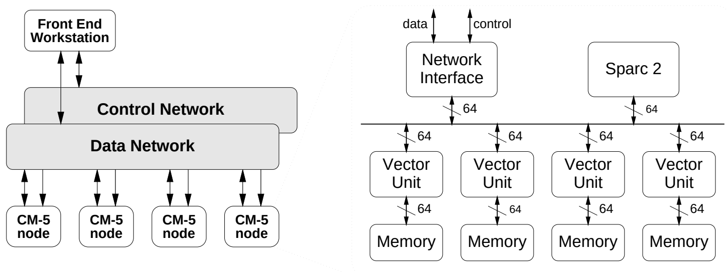

Figure 3: High-level architecture of the CM-5

The key to exploiting the CM-5's vector units is to express computation in terms of data-parallel vector operations, which is a natural paradigm for regular finite difference/volume models. The vector units act as both memory controllers and vector floating point units – each vector unit can read or write one 64-bit word of memory every cycle, as well as performing one floating point operation.

Main memory on the CM-5 is distributed across the vector units and so any sort of inter-processor communication on the CM-5 is relatively expensive, motivating the programmer to structure codes to minimize inter-processor communication. Often, this entails a tradeoff of extra memory use for less communication.

The architecture of the DEC 8400 [21] is typical of the current generation of microprocessor-based SMP's. As shown in Figure 4, the DEC 8400 consists of nodes connected by a shared system bus. Sharing of data between the processors can only occur through reads and writes to memory. Each node of the 8400 consists of an Alpha 21164 microprocessor [22] with three levels of cache – the first two levels of cache are on-chip, and the third level of cache is off-chip, but on the same board. The caches are small, fast-access memories which mirror sections of the computer's larger, slower access main memory. Computation using data from cache is much faster than using data from main memory. As the caches are situated farther from the processor core, they are successively larger and slower.

Other SMP's and even successive generations of DEC SMP's may have different sized caches, or even a different number of levels of caches. However, all microprocessor-based SMP's have one or more levels of cache and a shared view of main memory.

The 8400 and other mainstream SMP's rely on caches to get good perfor-

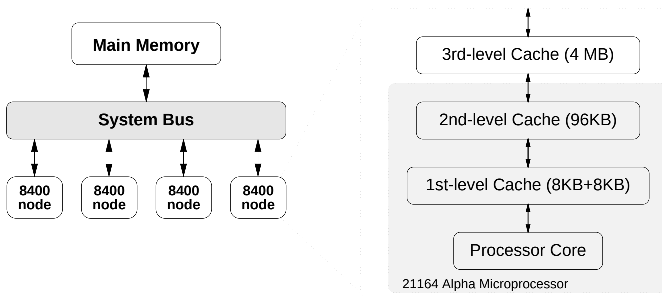

Figure 4: High-level architectures of the DEC 8400 SMP

mance. If a memory reference from the processor is returned by the 1st-level cache, the processor can continue working on the next cycle. References which are returned by the 2nd or 3rd level cache cause the processor to wait longer, and references to the main memory can cause the processor to wait for tens of cycles. Each Alpha microprocessor runs at 300 MHz, and if all of the operands are available, the Alpha can perform one floating point operation (which may be a multiply-add) per cycle, thus having a peak performance of 600 MFlop/s.

The shared memory architecture of the 8400 allows for lower communication costs than on the CM-5. However, if the problem does not fit into the 8400's caches, performance may be very bad, so it is important to minimize memory use. It should be noted, however, that main memory itself can be very large.

3.2 Programming Environments

Our goal is to make a model that is accessible to the general scientist, as well as to specialist scientific programmers and numerical modelers. A high-level, intuitive scientific programming environment is therefore of interest to us.

Motivated by this goal, we implemented the model on the CM-5 in CM Fortran (CMF) [23], which is a high-level, data-parallel dialect of Fortran. For the DEC 8400, we implemented the model in High Performance Fortran (HPF) using Digital's native High Performance Fortran (HPF) compiler.[24] HPF is very similar to CMF because the design of the HPF language standard was strongly influenced by CMF.

HPF compilers are available for most SMP platforms, provided either by the hardware vendor or third-party compiler vendors. However, the quality of HPF compilers can vary widely – although HPF compiling technology is fairly mature in general, some hardware vendors have not made a full commitment in manpower and other resources to develop high-quality HPF compilers for their platforms. Furthermore, in our experience, third-party compiler vendors are at a disadvantage in developing high-quality compilers for any particular platform because (1) they must support several platforms, and (2) details of the hardware necessary to obtain maximum performance are usually only well-understood by the hardware vendor.

The state of affairs is improving, but any decision to move to a new platform should be made only after evaluation of both hardware capabilities as well as software support.

4 Performance Profiling

To obtain high performance for the MIT model on a new computer platform, we take an incremental approach, beginning with performance profiling of elementary kernels, and moving towards profiling of the full application. At each stage, we gain a better quantitative understanding of the architecture, programming environment, and algorithmic match to the machine. The successive tests give us a step-by-step guide to the design of the full application.

4.1 **DAXPY** *Test*

The **DAXPY** test measures the performance of a system for vector-style operations. **DAXPY** is a primitive of the BLAS1 [25] linear algebra library which performs the operation $\underline{y} = a\underline{x} + \underline{y}$, where $\underline{y}$ and $\underline{x}$ are 64-bit floating point arrays.

Figure 5 shows the floating point performance as a function of the length of the input arrays for the 32 node CM-5, and for different processor counts of the DEC 8400. Examining this graph, we clearly see the impact of the cache size on performance. For the 8400 system, performance tails off sharply when cache size is exceeded, which is indicated at the "cache shoulders". As the number of processors increases, the total amount of cache memory also increases, as seen by the cache shoulder moving towards the right as more processors are used in the 8400.

For the CM-5, the performance increases with the size of the problem, because the CM-5 can more fully use its vector units on larger sized problems. In comparing the absolute performance of the CM-5 to the 8400, we should try to determine what region of interest is delineated by the curves. Once the size of the problem is too large to fit into the caches of the 8400, the performance is uncompetitive with the CM-5, or like machines.

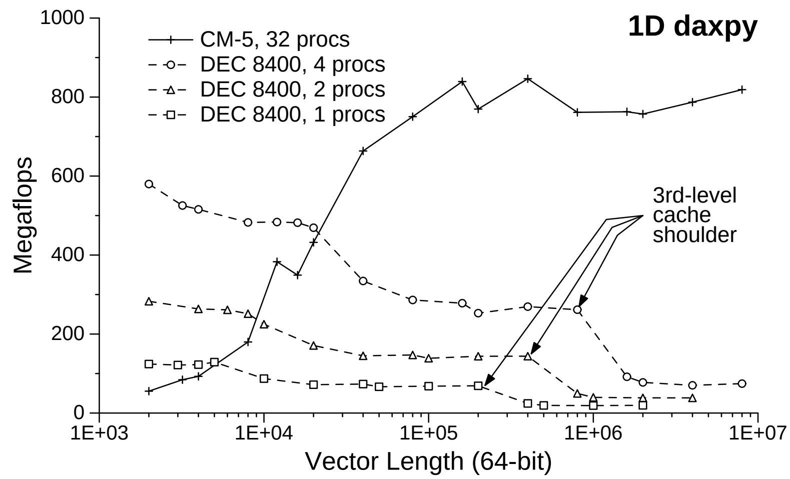

Figure 5: Performance of **DAXPY** BLAS1 primitive for different lengths of vector on one, two and four processors of a DEC 8400, and on a 32 node CM-5 computer.

In some ways, **DAXPY** is the worst-case situation for the SMP, because no data is reused, and therefore, the caches are poorly exploited. We can attempt to alleviate the effects of computations like **DAXPY**, by reusing data that is in the cache. However, traditional vector supercomputers favored applications which were structured as compositions of DAXPY-like operations, so programming and compiling for microprocessor-based SMP's requires a re-thinking of the mapping of an algorithm to the computer.

4.2 **3D DAXPY** *Test*

3D DAXPY is a three-dimensional version of **DAXPY**. Although the **3D DAXPY** computation seems extremely straightforward, there are actually some subtleties about programming even such a simple piece of code. Figure 6 shows some of the results obtained when we ordered the axes of the arrays differently, and when we used different HPF data layout directives for the same code using a fixed problem size and a fixed number of processors. We ran the code with arrays of size $200 \times 100 \times 10$, which is representative of runs we do daily on the CM-5.

There is almost a factor of 5 difference between the performance of Version 1 and Version 5. Version 2 actually uses the same axis order and layout as the CM-5 version of the MIT model, but it is not optimal on the 8400. It

3D DAXPY performance on DEC 8400			
Version	**Axis Order**	**Data Layout**	**Mflops**
1	(z,x,y)	(BLOCK,*,*)	60
2	(z,x,y)	(*,BLOCK,BLOCK)	210
3	(x,y,z)	(BLOCK,BLOCK,*)	260
4	(y,x,z)	(BLOCK,BLOCK,*)	230
5	(x,y,z)	(*,BLOCK,*)	270

Figure 6: The performance of **3D DAXPY** on the DEC 8400 under the DEC HPF compiler is highly dependent on the axis order declaration and data layout directives. The 3D arrays are of extents $X_{len} = 200, Y_{len} = 100, Z_{len} = 10$, and runs were on a 4 processor 8400.

is clear that Version 5 gives the maximum possible performance, because its performance is about the same as the one-dimensional **DAXPY** test.

4.3 **Forward_Step** *Test*

The **Forward_Step** part of the code consists of updates to a three dimensional grid of double-precision floating point numbers representing the state of the fluid. In our study, in keeping with our goal of a highly intuitive and readable code, the grid is mapped directly to three-dimensional arrays. Figure 7 shows the performance of a representative fragment from **Forward_Step**, which calculates the G terms in Equation 1. Problem sizes were chosen to be wide and thin, which is representative of geometries we use for our research. Times were gathered by extracting the section of code from the model, and using the optimal axis ordering and data layout as determined by the **3D DAXPY** test.

The results in Figure 7 represent the performance after modest optimizations were made to the DEC 8400 code relative to the CM-5 code. The CM-5 version actually uses about 40 arrays, some of which are used to minimize communication. With the DEC 8400 shared-memory architecture it is more important to reduce the overall memory usage, so some of these buffer arrays are removed. Memory requirements were further reduced by storing time invariant terms as 32-bit rather than 64-bit values.

Note that the performance of the DEC 8400 does not drop off as much as for **DAXPY** with this more representative code extract, but that the drop-off occurs at an earlier point. The improvement in asymptotic performance reflects the higher ratio of floating point operations to memory references of the **Forward_Step** test (which has a ratio of $\approx 26:8$) relative to the **DAXPY** tests (where the ratio is $1:1$). This higher ratio translates into increased op-

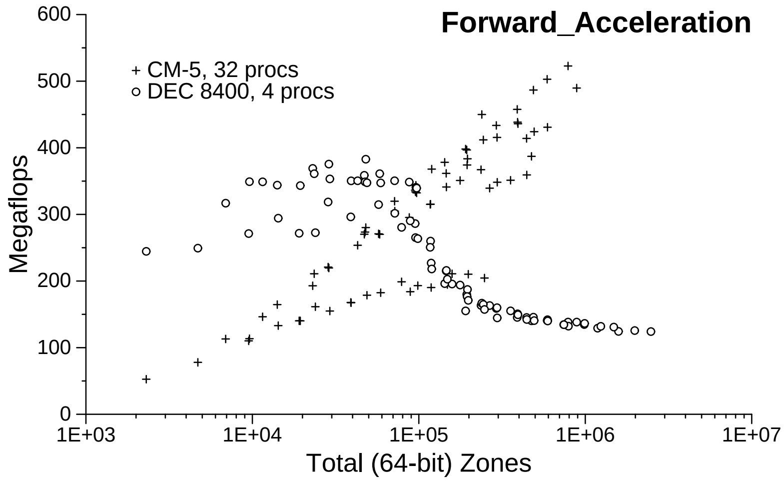

Figure 7: Performance of the **Forward_Step** test as a function of problem size. The shoulder for the DEC 8400 has moved to about 10^5 elements, because many more arrays are used for **Forward_Step** than for **DAXPY**.

portunities for reusing data loaded from main memory into cache. The degree to which this is exploited is highly dependent upon optimizations performed by the compiler and the way in which the code is written by the user.

The earlier drop-off point relative to **DAXPY** is the result of using more arrays – whereas **DAXPY** uses 2 arrays, this code segment uses the equivalent of 24 64-bit arrays, and accordingly, the drop-off point is about a factor of 12 earlier than for **DAXPY**. The full MIT model requires about the same number of arrays (to hold six state variables, their time derivatives at two time levels and geometric terms), so we expect that the rest of the **Forward_Step** code should behave as this code fragment does. To put this problem size in perspective, it is about equivalent to a global simulation with a four degree horizontal grid and twenty vertical layers.

4.4 **Inversion_Step** *Test*

The other important stage in the model algorithm is the **Inversion_Step** which solves an elliptic problem to ensure non-divergent flow. In discrete form this inversion can be written:

$$\underline{\underline{A}}\underline{p} = \underline{f} \tag{4}$$

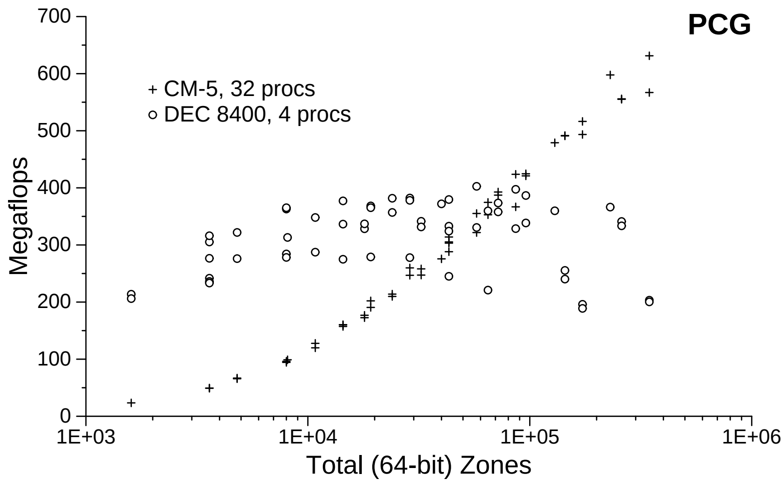

Figure 8: The preconditioned conjugate gradient algorithm used to solve for the pressure is competitive to the CM-5 over the region of interest (10^4 to 10^5).

in which, for hydrostatic simulations, $\underline{\underline{A}}$ is the horizontal Laplacian operator, a matrix with five diagonals, and $\underline{p}$ and $\underline{f}$ represent the surface pressure and the right-hand side term of Equation 3 respectively.

In the CM-5 implementation of the model, Equation 4 is solved using an iterative pre-conditioned conjugate gradient (PCG) scheme. PCG is well suited to data parallel implementations and the same algorithm was implemented in HPF on the DEC 8400.

Figure 8 shows the performance of the PCG algorithm on the CM-5 and DEC 8400. Note that because we are solving a two-dimensional problem instead of a three-dimensional problem (required in the non-hydrostatic limit formulation of the model), the problem sizes we consider are much smaller, and thus the performance of the DEC 8400 does not show the characteristic cache shoulder of the previous tests.

In addition, we perform further optimizations to minimize memory use. In the CM-5 implementation of PCG, to reduce communication we duplicate data structures that hold the diagonals of $\underline{\underline{A}}$ shifted by one point in x and y. On the DEC SMP system, this duplication is not necessary, and we exploit the symmetry of $\underline{\underline{A}}$ to eliminate another array by recalculating its value at run-time. Finally, we also store $\underline{\underline{A}}$ in 32-bit precision rather than 64-bit. In total, we were able to reduce the memory required to represent $\underline{\underline{A}}$ by a factor

Skyline (Direct) vs. PCG (Iterative) solvers on DEC 8400					
Ocean	**Skyline** (1 proc)			**PCG** (4 proc)	
Size	**Factor**	**Solve**	**Megaflops**	**Solve**	**Megaflops**
100×100	1.1 s.	0.1 s.	9.1	0.1 s.	350
200×100	2.4 s.	0.2 s.	8.7	2.7 s.	380
200×200	23.0 s.	0.9 s.	8.8	1.6 s.	467
400×200	49.7 s.	1.9 s.	8.4	4.6 s.	480
400×400	238.0 s.	10.0 s.	6.4	25.8 s.	230
800×400	633.3 s.	20.9 s.	6.1	83.5 s.	210

Figure 9: The direct solution to $Ax = b$ is divided into a factorization of A and the solve for x. Factorization only occurs once during the run of a program, and its cost is amortized out on long runs. The Skyline direct solver is faster in wall clock time than the PCG iterative solve, even though it has a lower Megaflops rating.

of 5.

The PCG iterative solution can require anywhere between 10 and 200 iterations to converge – the exact number will depend on the physics of the particular simulation. For a domain with a surface of size $NX \times NY$, if we assume that convergence requires *nits* iterations, the number of floating point operations executed is:

$$nits \times 32 \times NX \times NY$$

where 32 is the number of floating point operations performed per grid point per iteration in our PCG implementation.

An alternative approach to an iterative PCG solver is a direct solver. Because $\underline{\underline{A}}$ is symmetric and sparse, the LU decomposition can be stored within the profile of the $\underline{\underline{A}}$ matrix. For any $\underline{\underline{A}}$ representing a closed physical domain, this requires about $NX \times NX \times NY$ elements of storage. Once the LU factorization has been accomplished, we can find $\underline{p}$ using forward-backward substitution. Solving for $\underline{p}$ using this *direct* method requires the following number of floating point operations:

$$3 \times NX \times NX \times NY$$

where 3 floating point operations are performed per word of the LU factorization for the forward-backward substitution.

Because of the way the problem is posed, we can choose to appropriately switch NX and NY to arrive at a problem with a smaller LU decomposition. We should note, however, the lateral aspect ratio of typical geometries is not usually greater than $2:1$, so that the direct method has an algorithmic complexity $O(n^3)$, whereas the iterative methods generally require about $O(n^2)$.

Despite the higher order complexity of the direct method, the high constant factors in the iterative method make the direct method competitive for a certain range of problem sizes. Suppose that a 100×100 sized problem required 100 PCG iterations to converge – the PCG algorithm would require $100 \times 32 \times 100 \times 100 = 32,000,000$ floating point operations, whereas the direct method would only require $3 \times 100 \times 100 \times 100 = 3,000,000$ floating point operations. However, even though the direct method may execute fewer floating point operations, each floating point operation will be much slower than the direct method because the direct method must stream through the entire LU factorization, performing only 3 operations per element, leading to a fairly bad ratio of floating point operations to memory operations.

To test the effectiveness of a direct approach versus the iterative approach, we solved the same 2D problem using a sparse direct solver included with the DEC DXML scientific library [26]. The solver uses the skyline algorithm, which assumes matrices have elements which are primarily close to the main diagonal. This particular algorithm does not assume any other structure (other than symmetry) within the $\underline{\underline{A}}$ matrix, and currently only runs on 1 processor of the 8400 system – a more specialized or parallel direct solver may be faster than this very general-purpose, single-processor algorithm.

Nevertheless, the results in Figure 9 show that a direct solver could be a competitive solution method for certain problems. For example, Hill and Marshall[27] make a non-hydrostatic model competitive with a hydrostatic model, even in the presence of complex geometry, by separating out, and solving to high-accuracy, the two-dimensional surface pressure problem we examine here. Because the direct solver returns a solution to Equation 4 which is accurate to machine precision, the solver would be ideal for accelerating a non-hydrostatic model.

4.5 Application Performance

We evaluate the full model's DEC 8400 implementation by comparing its performance with the CM-5 implementation for a specific physical simulation. The simulation we chose was an experiment modeling the collapse and break up, through baroclinic instabilities, of an anomalously dense column of fluid in a rotating frame of reference.[10] The chosen test has a computational grid of size $100 \times 100 \times 26$, which is large enough to exceed the cache size of a four processor DEC 8400, but not so large that there are no opportunities for optimization.

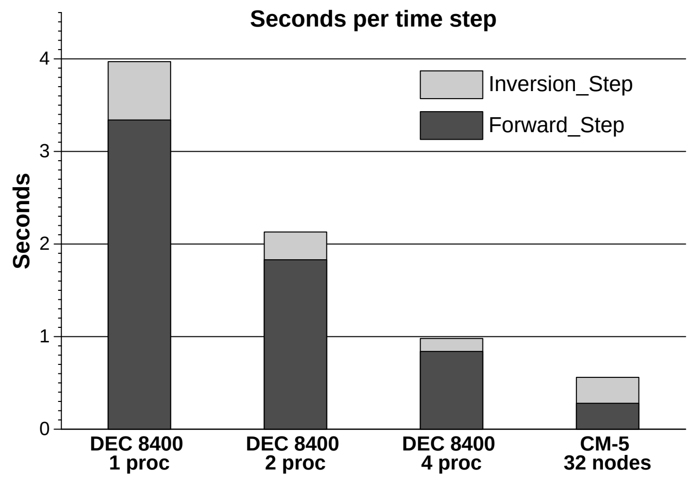

Figure 10: Performance of the full model on a 32 node CM-5 and a DEC 8400 5/300 server.

Figure 10 shows the relative performance of the CM-5 and DEC 8400 for this simulation. The DEC 8400 four processor system is about one half as fast as the CM-5. From our previous tests, we should be able to predict the performance of the DEC 8400 given the problem size. For the **Forward_Step**, referring to Figure 7, the problem size is 2.6×10^5, which shows that the DEC 8400 should be about 2.5 times slower than the CM-5. For the **Inversion_Step**, referring to Figure 8, the problem size is 10^4, so the DEC 8400 should be about 2.5 times faster than the CM-5.

For the problem that we simulated in Figure 10, the time spent in one time step on the CM-5 is about equally divided between the **Forward_Step** and the **Inversion_Step**. This ratio will depend on the physics of the simulation, in particular, the amount of work required by the **Inversion_Step** varies greatly from one problem to another. Notice that our performance prediction was fairly accurate: the time spent in the **Inversion_Step** for the DEC 8400 is indeed about 2.5 times less, and the time spent in the **Forward_Step** is roughly 2.5 times greater than the CM-5 time.

Although the particular integration considered here had a 50/50 balance between forward and inversion steps on the CM-5, other simulations have a different split. For problems that spend a large fraction of time in the inverter, the 4-processor DEC 8400 should out-perform the CM-5.

We have also compared single processor performance of our HPF test codes expressed in Fortran 77, using the DEC compilers. In these checks, the HPF test code performance was always within 10% of its Fortran 77 equivalent. This, along with the linear scaling shown in Figure 10, leads us to conclude that an explicit message-passing version of the MIT model (for instance, coded using PVM or MPI) would have similar performance to our HPF implementation. Furthermore, unlike a message-passing code, we are able to use a global address space throughout the model, making the code much easier to understand and modify.

5 Conclusion

We have implemented a complete hydrodynamical code in HPF on a DEC 8400 SMP. We have shown that its performance on a 4-processor DEC 8400 is competitive with a 32-processor CM-5 for problems we run routinely on the CM-5. We found that the size of a problem which could be feasibly integrated on a mainstream SMP machine like the DEC 8400 is limited by the aggregate cache memory size. For a 4-processor DEC 8400, performance is sufficient for coarse resolution ($\approx 4^\circ$ horizontal resolution) climatological time scale integrations, or for higher resolution (down to $\approx 1^\circ$) decadal simulation.

To obtain good performance on the SMP, we analyzed and optimized a series of isolated kernels that are the building-blocks of the full model. These test codes highlight factors which affect performance of the full application. For example, we see how performance scales with problem size, machine size, and cache size. Each test also reveals strengths and weaknesses of the compiler, and allows us to explore alternative implementations which may better match the system. These alternatives range from simple axis order re-ordering to completely different numerical procedures such as the substitution of a direct linear solver for an iterative one.

Achieving good performance on microprocessor-based SMP's depends upon many factors, including the number of processors in the system, the size of the caches for each processor, the speed of the individual processors, and the maturity of the compilation system. The DEC 8400 represents a successful combination of these attributes for our application, although other applications (e.g. spectral models or elaborate physics packages) may have different requirements. We note that processor counts, cache sizes, and processor speeds for SMP systems can vary significantly from vendor to vendor, although these hardware characteristics are easy to measure. Less easy to measure is compiler technology, which plays an equally important role in attaining good performance – it has been our experience that some systems with adequate hardware do not have compilers which can exploit the hardware to their full potential.

We are optimistic that microprocessor-based SMP platforms will, for real oceanographic and meteorological applications, bridge the gap between frequently over-burdened, centralized supercomputer center facilities and underpowered desktop workstations. However, the widespread adoption of SMP's is contingent on their affordability – currently, the 4-processor DEC 8400 system is significantly more expensive than four individual workstations. Nevertheless, it seems likely that prices will fall as high-performance SMP technology gains popularity in the marketplace, driven primarily by demands of the business community, as well as the scientific and engineering communities.

Finally we note that our adherence to a high-level, global programming model has made it significantly easier to automate the complex data flow analysis required to differentiate our forward model code. As a result we are able to apply the TAMC adjoint model generation tool[28] to our HPF code, and generate adjoint and tangent linear models automatically. Furthermore, through appropriate replication of the data distribution directives from the forward model, the automatically derived codes transparently inherit the parallelism of our forward model. Using this technique we are incorporating our model into a systematic optimization framework with relative ease.

Acknowledgments

This research is a joint effort of the MIT Center for Meteorology and Physical Oceanography and the MIT Laboratory for Computer Science. The research of Chris Hill is partially funded by ONR contract N00014-95-1-0967. The research of Andrew Shaw is partially funded by ARPA under ONR contract N00014-92-J-1310.

John Marshall and Arvind read drafts of this paper. We would also like to thank Digital Equipment Corporation and the Pittsburgh Supercomputing Center for giving us access to the DEC 8400 SMP. Experiments on the CM-5 were done under the MIT SCOUT project, ARPA contract #MDA972-92-J-1032.

References

1. M. Lam et.al. The cache performance and optimizations of blocked algorithms. In *Fourth International Conference on Architectural Support for Programming Languages and Systems*, April 1991.
2. S. McKee, Wm. A. Wulf, and T. C. Landon. Bounds on memory bandwidth in streamed computations. In *Lecture Notes in Computer Science 966*, pages 83–99. Springer-Verlag, Berlin, 1995.

3. John D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, December 1995. http://www.computer.org/tab/tcca/news/dec95/dec95_mccalpin.ps.
4. J. Dongarra. The LINPACK Benchmark: An Explanation. *Supercomputing*, pages 10–14, Spring 1988.
5. J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. Technical Report TR MC-SRD 23, Argonne National Laboratory, March 1988.
6. J. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic and non-hydrostatic ocean modeling. *J. Geophys. Res.*, in press 1997.
7. B. Klinger, J. Marshall, and U. Send. Representation and parameterisation of deep convective plumes by mixing. *J. Geophys. Res.*, 101 (C8):18175–18182, August 1996.
8. K.G. Speer and J. Marshall. The growth of convective plume at seafloor hot springs. *J. Marine Res.*, 53:1025–1057, 1995.
9. M. Visbeck, J. Marshall, and H. Jones. On the dynamics of convective chimneys in the ocean. *J. Phys. Oceanogr.*, 26:1721–1734, September 1996.
10. H. Jones and J. Marshall. Restratification after deep convection. *J. Phys. Oceanogr.*, in press 1997.
11. T. Haine and J. Marshall. Gravitational, symmetric and baroclinic instability of the oceanic mixed layer. *J. Phys. Oceanogr.*, in press 1997.
12. D. Menemenlis, T. Webb, C. Wunsch, U. Send, and C. Hill. Basin-scale ocean circulation from combined altimetric, tomographic and model data. *Nature*, in press 1997.
13. D. Menemenlis and C. Wunsch. Linearization of an oceanic circulation model for data assimilation and climate studies. *Journal of Atmospheric and Oceanic Technology*, in press 1997.
14. A. Adcroft, C. Hill, and J. Marshall. On the treatment of topography in numerical ocean models. *Mon. Wea. Rev.*, in press 1997.
15. J. Marshall, A. Adcroft, C. Hill, and L. Perelman. A finite-volume, incompressible Navier-Stokes model for studies of the ocean on parallel computers. *J. Geophys. Res.*, in press 1997.
16. F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow. *Phys. Fluids*, 8:2182, 1965.
17. R. Brugge, H.L. Jones, and J. Marshall. Non-hydrostatic ocean modelling for studies of open-ocean deep convection. In *Proceedings of the Workshop on Deep convection and deep water formation in the ocean*,

pages 325–340. Elsevier Science Publishers, Holland, 1991.

18. M.J. Miller. On the use of pressure as vertical co-ordinate in modelling. *Quart. J. R. Meteorol. Soc.*, 100:155–162, 1974.
19. A. Shaw, K-C Cho, C. Hill, R. P. Johnson, J. Marshall, and Arvind. A comparison of implicitly parallel multithreaded and data parallel implementations of an ocean model based on the Navier-Stokes equations. Technical Report CSG Memo 364, MIT LCS, 1995.
20. TMC. *The CM-5 Technical Summary.* Thinking Machines Corporation, Cambridge, MA, 1991.
21. D.M. Fenwick et.al. The alphaserver 8000 series: High-end server platform development. *Digital Technical Journal*, 7(1), 1995.
22. J.H. Edmondson et.al. Internal organization of the alpha 21164, a 300-mhz 64-bit quad-issue cmos risc microprocessor. *Digital Technical Journal*, 7(1), 1995.
23. Thinking Machine Corporation. *CM Fortran Language Reference Manual, V2.1*, January 1994.
24. J. Harris et.al. Compiling High Performance Fortran for Distributed-memory Systems. *DEC Technical Journal*, 7(3), 1995. ftp://ftp.digital.com/pub/Digital/info/DTJ/v7n3_03_FORTRAN.ps.
25. C. Lawson, R. Hanson, D. Kincaid, and Krogh F. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.
26. D.P. Manley C. Kamath, R. Ho. DXML: A High-performance Scientific Subroutine Library. *Digital Technical Journal*, 6(3), 1994. ftp://ftp.digital.com/pub/Digital/info/DTJ/v3n6-dxml.txt.
27. C. Hill and J. Marshall. Application of a parallel Navier-Stokes model to ocean circulation. In *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers*, pages 545–552. Elsevier Science Publishers, Holland, 1995.
28. R. Giering and T. Kaminski. Recipes for adjoint code construction. Technical Report 212, Max-Planck-Institute for Meteorology, 1996.