Transitioning from MPP to SMP: Experiences with the MIT Ocean Model

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

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

High-end mainstream symmetric multiprocessor (SMP) computers are beginning to offer numerical modelers realistic alternatives to traditional vector or massively parallel (MPP) super-computers. In this paper we examine how, and to what extent, the meteorological and oceanographic numerical community could usefully employ such SMP's. We describe our efforts to port and implement a ocean-atmosphere model on a DEC 8400 SMP and on a Sun Ultra Enterprise 5000 SMP. These implementations are contrasted with an existing optimized version of the same model targeted to the CM-5 vector super-computers. The numerical methods and programming styles for each implementation are tailored to the individual computer architectures. To characterize the differences in the architectures, we compare the performance of key linear algebra primitives which are used in many computational fluid dynamics (CFD) models. In keeping with a research oriented philosophy we focus on a flexible model implementation directly accessible to individual researchers and students, emphasizing a high-level programming environment in addition to high-performance. Our study demonstrates 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

The application of specialised, "limited-edition", supercomputers to large-scale meteorological and oceanographic computational modeling has been an unquestionable success story, paving the way for vast improvements in numerical simulation skill over the last twenty years. Through-out this period vector supercomputers have led clearly in absolute performance for meteorological applications and vector computing has proved an indispensable tool for tackling the virtually insatiable demands of meteorology's underlying numerical algorithms. Recently, however, innovations in commodity microprocessor-based multiprocessor computer designs have brought their performance to the level where they can seriously be considered as a substitute for some tasks previously requiring supercomputer-level power.

Contemporary computer architects now enthuse over the possibility of multiprocessor commodity microprocessor based supercomputers, delivering significant performance to numerical applications at a fraction of the cost of traditional supercomputers and with much broader availability. In particular there is a widely articulated viewpoint that mainstream symmettric multi-processor (SMP) architectures based on the same commodity microprocessor technology as desktop workstations will inevitably form the building blocks of future high-end "supercomputer" platforms. However, skeptics doubt 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. The skeptical argument is compelling. Modern commodity microprocessor platforms have universally adopted scalar memory architectures that perform poorly with the vector-centric numerical procedures that lie at the heart of many supercomuter applications (including most meteorological and oceanographic applications).

The STREAMS benchmark, which measures the sustainable main memory bandwidth of a computer, highlights the difference between vector and scalar memory architectures. The benchmark shows that typical state of the scalar memory computers have bandwidths of about 2-300MB/sec, a factor of about fifty worse than typical state of the art vector supercomputers (see table 1). However, this metric measures only one dimension of performance, in particular the benchmark reflects the performance of a code constructed solely out of a rigid sequence of long-vector operations. A different picture emerges from the LINPACK benchmark (see table 1). In this case the distinct architectures perform comparably. Unlike STREAMS, LINPACK can be mapped efficiently to both present day scalar memory architectures and vector architectures.

The STREAM and LINPACK benchmarks are both numerical codes, but they work with different categories of linear algebra primitives. The LINPACK benchmark is based on dense matrix problems which can readily exploit the fast and small cache memory found in commodity architectures. STREAM is based on long-vector operations and there are no algorithms which permit such operations capitalise on cache memory to the extent that LINPACK does. The difference in "cache friendliness" between the two benchmarks accounts for the different performance. The difference between the two benchmarks also demonstrates the challenge faced in mapping meteorological and oceanographic applications to commodity microprocessor platforms platforms. Meteorological and oceanographic codes are typically closer to the STREAM benchmark than to the LINPACK benchmark, and so tend to be difficult to implement efficiently on cache based machines. Nonetheless the increasing availability and ever diminishing cost of mainstream microprocessor computers along with their tantalising increases in performance highlighted by table 1 make it tempting to investigate what role they might play in present day large-scale meteorological and oceanographic modeling and to examine how that role might evolve in the furture.

In this paper we explore ways the meteorological numerical modeling community might start to exploit the abundant medium scale parallelism that state of the art commodity microprocessor SMP architectures purport to offer today. We report on the design, implementation and evaluation of the hydrodynamical kernel of the MIT ocean-atmosphere model on two state of the art SMP platforms (the DEC 8400 5/300 and the Sun UltraEnterprise 5000); comparing the SMP implementations with an existing implementation on a CM-5 massively parallel processor (MPP) currently used for daily production runs of the MIT model. A vision of near desktop '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 born numerical modelers. We study the performance of a hierarchy of three problems, culminating in the MIT model hydro-dynamical kernel described in section 2, to probe the strengths and weaknesses of the SMP platforms and to identify promising implementation strategies.

2 The M.I.T Ocean-Atmosphere Model

The M.I.T ocean-atmosphere model is built around a versatile incompressible Navier-Stokes solver developed at M.I.T for the numerical study and analysis of rotating fluids. A hydro-dynamical kernel, carefully 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 very broad spectrum of simulation regimes. Currently implementations of the model targeted to the MPP CM-5 and to CRAY vector supercomputers are being actively used by researchers at M.I.T. The discrete algorithm uses a finite volume, time-stepping formulation which is rich in data parallelism and that can handle the intricate, order one variations in geometry of ocean basins. The CM-5 code can achieve sustained floating point throughput of 10+GFlop/s on a 512 node CM-5. Daily production runs of the model routinely run at between 500 MFlop/s and 1GFlop/s on a 32 node CM-5, and at about 100-200 MFlop/s sustained on a CRAY Y-MP. A review of the model's utility for research into a range of problems related to ocean and climate dynamics can be found in Marshall et. al[1].

Table 1: Standard benchmark results for "high-performance" computer platforms. The results show the dramatic improvement in numerical performance for "cache friendly" algorithms (like the LINPACK dense matrix benchmark code) on mainstream microprocessor platforms. The Apollo DN100010 is a circa 1990 $100,000 - $300,000 machine while the DEC 8400 5/300 is a circa 1995 $100,000 - $1,000,000 platform. The LINPACK performance improvements are not reflected in the the STREAM benchmark which measures performance for a code composed of a rigid sequence of long vector-vector operations. The performance of the state of the art scalar platforms is still well below that of the previous generation vector platform (the circa 1989 CRAY Y-MP) for such vector operations.

Machine	Linpack (1000x1000)	STREAMS (MBytes/s)	STREAMS (Equiv. MFlop/s)	Vector/ Scalar
CRAY C90 (1. proc 2.2ns)	902	9500	791	Vector
CRAY YMP (1. proc 6.0ns)	332	3396	283	Vector
CM-5 32 nodes	1900 (r_{max})	12372	1031	Vector
DEC8400 5/8400 (4. proc 2.9ns)	1351	832	55	Scalar
Apollo DN10010	6	54	5	Scalar

2.1 The Numerical Procedure

The numerical procedure used in the model is described in detail in Marshall et. al [2]. The scheme is a variant on the theme set out by Harlow and Welch [3], in which a pressure correction to the velocity field is used to guarantee non-divergence. In our scheme we write the Boussinesq incompressible Navier Stokes equations describing our ocean model in semi-discrete form to second order in the time Δt, in which only time is discretized:

$$\frac{\underline{v}_h^{n+1} - \underline{v}_h^n}{\Delta t} = \underline{G}_{v_h}^{n+\frac{1}{2}} - \nabla_h \left\{ p_s + p_{hy} + q p_{nh} \right\}^{n+\frac{1}{2}} \tag{1}$$

$$\frac{w^{n+1} - w_n}{\Delta t} = \widehat{G}_w^{n+\frac{1}{2}} - \frac{\partial p_{nh}^{n+\frac{1}{2}}}{\partial z} \tag{2}$$

$$\frac{\partial w^{n+1}}{\partial z} + \nabla_h . \underline{v}_h^{n+1} = 0 \tag{3}$$

Equations (1)–(3) describe the time evolution of the flow; $\underline{v} = (\underline{v}_h, w)$ is the velocity in the horizontal and the vertical, and the $\underline{G}$ terms incorporate inertial, Coriolis, metric, gravitational, and forcing/dissipation terms. Equations for temperature and salinity are also stepped forward to find, through an equation of state, the density ρ. In (1) the pressure is separated into surface, hydrostatic and non-hydrostatic components:

$$p(\lambda, \phi, z) = p_s(\lambda, \phi) + p_{hy}(\lambda, \phi, z) + q p_{nh}(\lambda, \phi, z) \tag{4}$$

where (λ, ϕ, z) are the latitude, longitude and depth respectively. The first term, p_s, is the surface pressure (the pressure due to sea surface height variations); it is only a function of horizontal position. The second term, p_{hy}, is the hydrostatic pressure defined in terms of the weight of water in a vertical column above the depth z,

$$\frac{\partial p_{hy}}{\partial z} + g' = 0 \tag{5}$$

where g' is the "reduced gravity" $g\frac{\delta\rho}{\rho_{ref}}$, and $\delta\rho$ is the departure of the density from a reference profile, and ρ_{ref} is a constant reference density. The third term is the non-hydrostatic pressure, p_{nh}. In (2) the balancing terms involving the hydrostatic pressure and gravity (5) have been canceled (and the remaining terms represented by $\widehat{G}_w$) to ensure that it is well conditioned for prognostic integration. In hydrostatic experiments, q is zero and (2) is not used; rather w is diagnosed from the continuity relation (3). In the hydrostatic limit only p_{hy} and p_s are required in (4). Equation (5) is used to find p_{hy} and p_s is found by solving a two-dimensional elliptic problem for the surface pressure that ensures non-divergent depth-integrated flow:

$$\nabla_h . H \nabla_h p_s = \nabla_h . \overline{\underline{G}_{v_h}^{n+\frac{1}{2}}}^H - \nabla_h . \overline{\nabla_h p_{hy}}^H \tag{6}$$

($\overline{\quad}^H$ indicates the vertical integral over the local depth, H, of the ocean and the subscript $_h$ denotes horizontal).

In the non-hydrostatic case q is unity and p_{nh} is found by inverting a three-dimensional elliptic equation ensuring that the local divergence vanishes:

$$\nabla^2 p_{nh} = \nabla . \underline{G}^{n+\frac{1}{2}} - \nabla_h^2 (p_s + p_{hy}). \tag{7}$$

where ∇^2 is a full three-dimensional Laplacian operator.

We form our atmospheric model by exploiting the isomorphism described in Brugge et al[4] between the equations of motion for an incompressible fluid (the ocean) in height coordinates and those of a compressible fluid (the atmosphere) in pressure coordinates. This allows us to employ the same numerical procedure and same program code to simulate either the ocean or the atmosphere. To 'endow' our atmospheric simulations with realistic climatologies suitable for studying feedbacks between the atmosphere and ocean we adopt the technique used by Marshall and Molteni [5] of defining a temperature forcing which persistently holds the model to the long term climatological average. This circumvents the need for elaborate models of the myriad complex physical processes in the atmosphere which have no obvious oceanic analogue; for example cloud physics and radiative-convective processes. The atmospheric model then reduces to a pure hydro-dynamical engine based on the Navier-Stokes equations and the atmospheric numerical procedure is identical in form to the oceanic procedure. We refer the reader to Brugge et. al [4] and the work of Miller [6,7] for a complete discussion of the analogous atmospheric equations which are arrived at by making the transformations:

$$-\, wg\rho_s \rightarrow \omega \tag{8}$$

$$\frac{1}{g\rho_s}\frac{\partial}{\partial h_s} \rightarrow \frac{\partial}{\partial p} \tag{9}$$

$$\frac{p'}{\rho_s} \rightarrow gh' \tag{10}$$

in equations (1)-(7)

3 The computer platforms

The platforms we examine have distinct architectural features and offer a selection of programming environments. The CM-5, is vectorizing multi-processor architecture that caters specifically to scientific simulation applications. This platform (along with the CRAY Y-MP) is the current mainstay of our day to day production runs of the MIT ocean-atmosphere model. The two SMP platforms, the DEC 8400 and the SUN UltraEnterprise 5000, are recently released commercial systems. These latter two computers have been designed for use in a broad variety of both technical and non-technical situations. All the systems we examine come in a wide variety of configurations. The specific configurations we use for our experiments represent the typical configurations readily accessible to research scientists.

3.1 Architectural Differences

Architectural attributes that will affect our application performance are the memory subsystem and the floating point capability. Each of the systems here has theoretical peak floating point performance in excess of 100MFlop/s ,however, the DEC and SUN SMP systems have substantially different memory architectures from the CM-5. Figures 1, 2 and 3 show the high level architectures of each of the platforms.

The CM-5 MPP [8] ,although nearly extinct, serves as one of the first examples of a computer that, through the combination of hundreds or thousands of separate processors and a high-level programming environment, could routinely out-perform state-of-the-art vector supercomputers for real scientific problems. The CM-5 is built from a network of relatively modest vector processors and so needs to exploit the data parallelism common to many numerical simulation codes to achieve multi-gigaflop/sec performance. The contemporary Fujitsu VPP series [9] has a high-level structure similar to the CM-5. The vector units of the CM-5 act as both memory controllers and vectorizing 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 (which may be a multiply-add). Since each vector unit runs at 16 MHz, and each node of the CM-5 contains 4 vector units, the peak performance of one CM-5 node is 128 MFlop/s. 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 grid-point models. 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 application writer to structure codes such that communication is minimized – often, this entails a tradeoff of extra memory use for less communication.

The DEC 8400 [10] consists of nodes connected by a shared bus. Communication between the processors can only occur through reads and writes to the shared main memory. Each node of the 8400 consists of an Alpha 21164 microprocessor [11] 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 faster, but smaller memories which contain copies of sections of the main memory, allowing the microprocessors to read and write memory which is mirrored in the caches much faster than it would if it had to always write to the main memory. The 8400 relies on its caches to get good memory performance – if a referenced memory location is not in any of the three caches, the reference out to main memory can be extremely expensive. Each Alpha runs at 300 MHz, and if all of the operands are available, the Alpha can perform

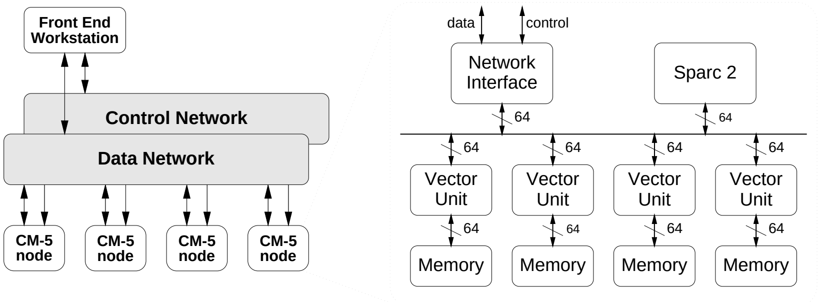

Figure 1: High-level architecture of the CM-5

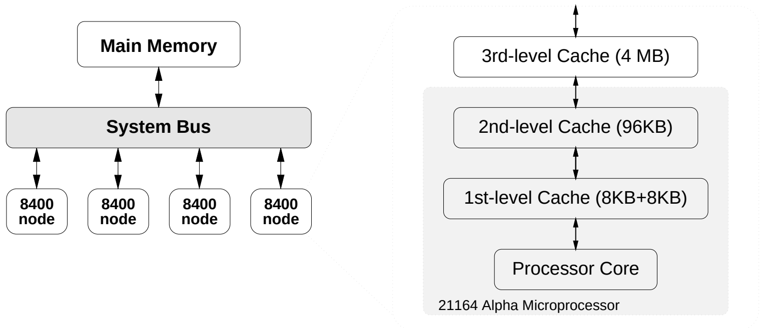

Figure 2: High-level architecture of the DEC 8400

one floating point operation (which may be a multiply-add) per cycle, thus having a peak performance of 600 MFlops. The key to exploiting the Alpha microprocessor is referencing memory in such an order that those memory locations will be in cache. On the 8400, communication costs are not as high as the CM-5 because of the shared memory, but any extra memory use causes poor utilization of the cache, so it is important to minimize memory use, even though main memory itself may be very large.

The Enterprise 5000 SMP system is a shared memory multi-processor system which can be configured with up to 30 processors. Like the DEC 8400 this machine has a scalar memory subsystem which relies on a small fast cache memory exploiting the locality of reference in an algorithm for performance (see figure 3). The UltraSPARC-I processor [12] operates at 167MHz and can perform up to two floating point instructions per clock cycle, making for a theoretical peak floating-point performance of 333MFlop/s.

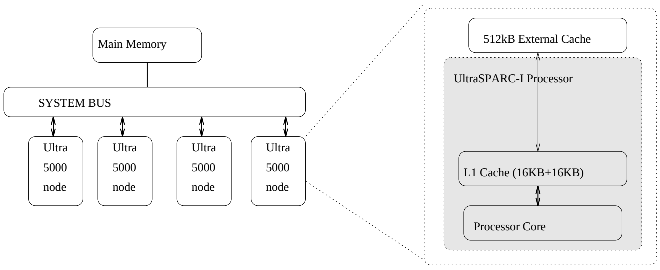

Figure 3: High-level architecture of the ULTRA 5000

3.2 Programming Environments

The aim of our study was to identify prospective near-desktop 'supercomputers' as research and teaching tools. Ideally such tools should be accessible to the general scientist as well as to specialist scientific programmers and numerical modelers, suggesting that a high-level and intuitive scientific programming environment is important. Motivated by this goal we chose to look at implementations in High Performance Fortran (HPF) on the SMP platforms. The DEC system has a native HPF compiler whilst for the Sun SMP we used a third party product from Thinking Machines Corporation (the developers of the CM-5) called TM-HPF. The CM-5 programming environment to which we compare is very similar to HPF and we have found from experience that the CM-5 code is readily understood by end-users with relatively little assistance.

4 Experiments

In this report we study three separate problems ,of increasing sophistication and size, to examine performance and explore implementation strategies best suited to each particular platform. The first two problems are representative of computational aspects of the building blocks which go to make up the third. The third problem is the full hydro-dynamical kernel from the M.I.T model.

4.1 Problem 1: DAXPY

We use the DAXPY problem to measure the performance of each system when performing arithmetic on vector operands. DAXPY is a primitive of the BLAS1 linear algebra library which performs the operation

$$\underline{y} = a\underline{x} + \underline{y}$$

In this test we are interested in seeing at what point the penalty for referencing data in main memory rather than in cache becomes prohibitive for the SMP architectures. Figures 4, 5 and 6 show curves of floating point throughput as a function of the length of the vector $\underline{x}$ for the different platforms and for different processor counts for the SMP platforms, a curve showing DAXPY performance on a 32 node CM-5 is also shown for comparison. Because communication costs are minimal on the SMP platforms the results from this test are a pointer to what performance we can expect from the time-stepping part of the model algorithm, equation 1 and its counterparts for the thermodynamic variables. Examining these figures we see clearly the impact of the cache size on performance. For both SMP platforms performance tails off sharply when cache size is exceeded. For a four processor Ultra 5000 the tail off occurs at a problem size eight times smaller than the tail off point for the four processor DEC 8400, reflecting the relative cache sizes on these two machines.

The comparison of the SMP DAXPY with the CM-5 is stark and illustrates clearly that producing an SMP implementation of the M.I.T model that can approach the efficiency of a 32 node CM-5 is a challenging goal. However, we can see that for the four processor DEC 8400 performance of 300MFlop/s is possible for a problem involving two million 64-bit floating point numbers. This is encouraging on two counts. First, because for some problems that people are currently studying our numerical procedure requires fewer than two million numbers. Our finite volume, grid point algorithm requires uses 24 three dimensional fields to step forward the model state and carry the domain geometry. With this scheme a global model with a four degree grid and up to twenty vertical layers involves fewer than two million 64-bit floating point numbers. A second reason to be encouraged is that this result suggests that if we could reorder the computations in the time-stepping code so that we worked extensively on one region of the grid evaluating all the terms for that region before working on the next region we could improve the cache utilization of our algorithm.

4.2 Problem 2: Inverter

The DAXPY test highlighted the importance cache utilization will play in determining the performance of our final algorithm. It suggested that we might be able to get significant performance provided we ordered the computations for the prognostic equations intelligently. In this next test we examine the performance of the elliptic inverter that diagnoses the surface pressure field by solving equation 6). In our model this inversion takes the form

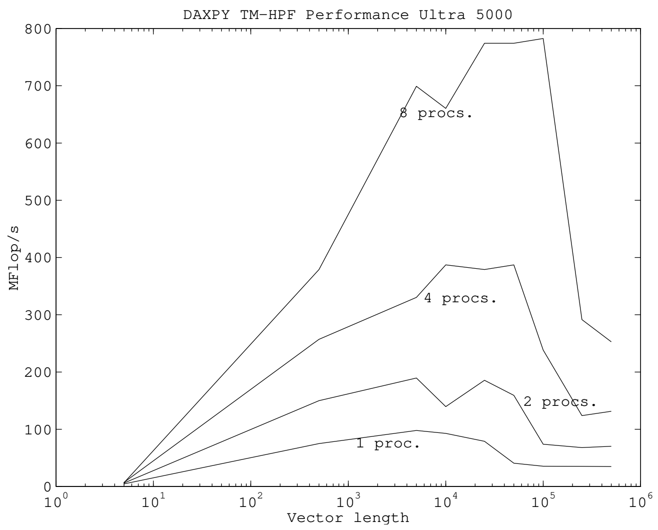

Figure 4: Performance of DAXPY BLAS1 primitive for different lengths of vector and for different processor counts of a SUN UltraEnterprise 5000 computer.

$$\underline{\underline{A}}\underline{p} = \underline{f} \tag{11}$$

in which $\underline{\underline{A}}$ is the Laplacian operator, a matrix with five diagonals, and $\underline{p}$ and $\underline{f}$ represent the surface pressures and the right-hand side term of equation 6 at each point on the model grid respectively. In the CM-5 implementation of the model equation 11 is solved using an iterative pre-conditioned conjugate gradient (PCG) scheme. PCG is well suited to data parallel implementations and we implemented the same algorithm in HPF on the DEC 8400 and the Sun Enterprise 5000. Unlike the DAXPY test PCG algorithm requires communication and synchronization between processors. In that regard it is representative of the finite volume operations in the full model prognostic code. However, unlike the prognostic code, PCG requires operations to adhere to the strict sequence outlined in figure 7. This leaves little room for an implementation which can improve cache utilization so we also examine the application of a different algorithm to the solution of equation 11. In this second approach we take advantage of the shared memory nature of the SMP architecture in contrast to the CM-5 which enables us to implement a banded matrix direct solver.

Figure 8 shows the performance of the of our PCG algorithm for a problem

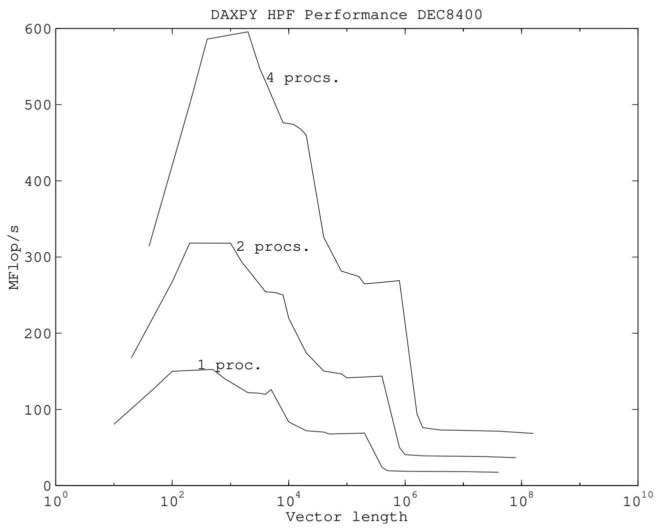

Figure 5: Performance of DAXPY BLAS1 primitive for different lengths of vector and for different processor counts of a DEC 8400 5/300 computer.

size of 320 by 720 on the CM-5 and on the DEC 8400 for different processor counts. From this test we see that the four processor performance is actually nine times greater than the one processor PCG performance on the DEC 8400. This is because as the number of points per processor decreases the per processor performance moves to the left on the DAXPY performance curve figure 5 out of the main memory bound region and into the significantly cache resident region giving rise to a super-linear speed up. Although the PCG algorithm does not allow for major code reordering we did make some changes relative to the CM-5 implementation. In the CM-5 implementation of PCG it is essential to minimize the interprocessor communication to ensure good performance. For this reason duplicate data structures that hold the diagonals of $\underline{\underline{A}}$ shifted by one point in X and Y are employed. On the DEC SMP system these duplicated arrays would utilize precious cache memory and so instead we put shifted references to diagonals of $\underline{\underline{A}}$ in the iterative loop. In a similar vein we exploit the fact that $\underline{\underline{A}}$ is symmetric and that its main diagonal is minus the sum of the off diagonals to reduce the number diagonals we store for $\underline{\underline{A}}$ to two. A final optimization is made by noting that the values of $\underline{\underline{A}}$ need not be stored in 64-bit precision, so we instead store them as 32-bit values making for a further reduction in memory requirements. The values of the elements

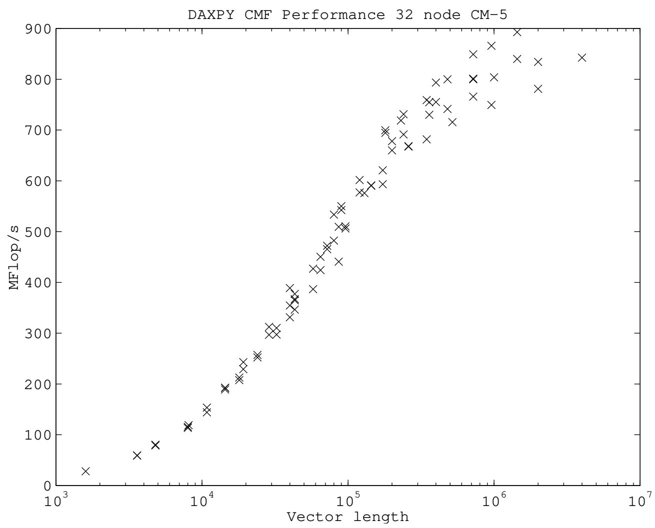

Figure 6: Performance of DAXPY BLAS1 primitive for different lengths of vector on a 32 node CM-5 computer.

of $\underline{\underline{A}}$ are determined by the geometry of the domain in which equation 11 is solved, they are normalized to be $O(1)$ and so in principle could be held in as little as 16-bit precision, however, the DEC 8400 hardware and software does not readily support 16-bit arithmetic operands. The TM-HPF implementation of PCG is not included in this section because it became evident during this test that the TM-HPF compiler was not yet mature enough to give meaningful performance for this code or our full model.

The PCG iterative solution requires a lot of iterations to converge, and consequently many floating point operations are necessary to solve each linear system. For an 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 about:

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

However, because $\underline{\underline{A}}$ is symmetric and sparse, the LU decomposition can be stored within the profile of the $\underline{\underline{A}}$ matrix – that is, the spaces between (and including) the main diagonal and the outermost diagonal. For any $\underline{\underline{A}}$ representing a closed basin, this requires about $NX \times NX \times NY$ elements of storage. Once we find the LU factorization, we can find $\underline{p}$ using forward-backward substitution. This algorithm requires 3 floating point operations per

$$\boxed{\underline{r}_o = \underline{f} - \underline{\underline{A}}\,\underline{p}_o \quad \underline{b}_o = \underline{\underline{\tilde{A}}}^{-1}\underline{p}_o}$$

$$\downarrow$$

$$\boxed{\begin{array}{c} \alpha_i = \dfrac{[\underline{r}_i, \underline{\underline{\tilde{A}}}^{-1}\underline{r}_i]}{[\underline{b}_i, \underline{\underline{A}}\,\underline{b}_i]} \\ \underline{p}_{i+1} = \underline{p}_i + \alpha_i \underline{b}_i \quad \underline{r}_{i+1} = \underline{r}_i - \alpha_i \underline{\underline{A}}\,\underline{b}_i \\ \beta_i = \dfrac{[\underline{r}_{i+1}, \underline{\underline{\tilde{A}}}^{-1}\underline{r}_{i+1}]}{[\underline{r}_i, \underline{\underline{\tilde{A}}}^{-1}\underline{r}_i]} \\ \underline{b}_{i+1} = \underline{\underline{\tilde{A}}}^{-1}\underline{r}_{i+1} + \beta_i \underline{b}_i \end{array}}$$

$$\downarrow \quad \uparrow n$$

$$\boxed{[\underline{r}_{i+1}, \underline{r}_{i+1}] < \epsilon^2}$$

$$\downarrow y$$

$$\boxed{\text{Stop}}$$

Figure 7: Flow diagram for the pre-conditioned conjugate gradient algorithm. The algorithm uses an approximate inverse, $\underline{\underline{\tilde{A}}}^{-1}$ (called the pre-conditioner) to find a correction to the current estimate of the solution. Repeated iteration converges in a finite number of steps to within a finite tolerance, ϵ. The pre-conditioner is used in each iteration.

element of the LU factorization. Solving for $\underline{p}$ using this *direct*, as opposed to *iterative* method requires the following number of floating point operations:

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

Because of the way the problem is posed, we can choose to switch NX and NY so that we have a problem with a smaller LU decomposition. We should note, however, the 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 $O(n^2)$. Due to the computational constraints imposed by the rest of the program, we are usually considering problem sizes within a certain range, and the constant factors due to the number of iterations to convergence play an important factor.

The direct method will require much more memory than the PCG solution. To store the LU factorization, we need $NX \times NX \times NY$ double-precision floating point numbers. For a 200×100 sized ocean, we require about 16 Megabytes to store the LU factorization. Furthermore, the forward-backward substitution is similar to the daxpy code in that once an array element is used, it will not be used again, so that the computation must stream through the entire factorization, paying the penalty for accessing main memory.

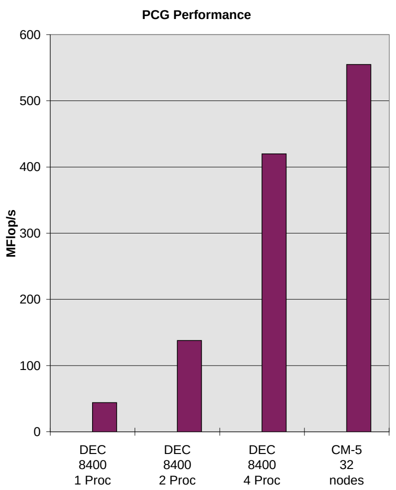

Figure 8: Performance of the PCG solver on a 32 node CM-5 computer and a DEC 8400 5/300 server.

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 [13]. 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. Nevertheless the results in table 4.2 show that this is a competitive solution method for *certain* problems.

4.3 Problem 3: The Full Model

Through the test problems described previously we concluded that it would not be possible to implement a full model implementation on the Sun UltraEnterprise 5000 using TM-HPF so we only discuss the DEC 8400 implementation here. We evaluate this 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 baro-

Table 2: The direct solution to $\underline{\underline{A}}\underline{p} = \underline{f}$ is divided into a factorization of $\underline{\underline{A}}$ and the solve for $\underline{p}$. Factorization only occurs once during the run of a program, and its cost is amortized out on long runs. The direct solve is faster in wall clock time than the iterative solve, even though it has a lower Megaflops rating. The iterative solver here solves to machine accuracy from an initial guess of $\underline{p} = 0$. In actual numerical integrations the accuracy to which $\underline{p}$ is solved is determined dynamically by monitoring the velocity divergence and the value of $\underline{p}$ from the previous time-step is used as an initial guess. This can reduce the time for the iterative solver by as much as ten times. Nevertheless for the larger problem sizes shown here the direct solver timings are likely to be the fastest solution method.

Skyline (Direct) vs. PCG (Iterative) solvers on DEC 8400			
Ocean	**Skyline** (1 proc)		**PCG** (4 proc)
Size	**Factor**	**Solve**	**Solve**
50×50	10.0 s.	0.2 s.	.05 s.
200×100	10.0 s.	0.2 s.	2.7
200×200	10.0 s.	0.2 s.	1.6
400×200	10.0 s.	0.2 s.	4.6
400×400	10.0 s.	0.2 s.	25.8
800×400	210.2	8.8	83.5

clinic instability of an anomalously dense column of fluid in a rotating frame of reference. The chosen test has a computational grid $100 \times 100 \times 26$ (of sufficient size that it exceed the cache size of a four processor DEC 8400 but not so large that the are no opportunities for optimization) representing a portion of ocean 200 kilometers square and two kilometers deep. Figure 9 shows the relative performance of the CM-5 and DEC 8400 for this simulation. The DEC 8400 four processor system is about one third as fast as the CM-5. This result was surprising; we had expected the two platforms to perform comparably for this problem. A performance analysis of the experiment revealed that most of the time on the DEC 8400 code was being spent in work associated with the CSHIFT Fortran 90 primitive which performs periodic shifts on entire arrays. This behavior is different from the two test programs and arises because the DEC HPF compiler is not recognizing that shift operations are regular nearest-neighbor communication, for which the compiler can generate highly optimal code by automatically introducing ghost elements around the boundaries of arrays. In the Inverter test problem the compiler was able to recognize and optimize these characteristic communication patterns. Although it is possible to manually introduce ghost regions to fix this problem it is clearly preferable and in the spirit of this study to expect the compiler to generate the code. We anticipate that once this feature is implemented in the compiler the CM-5 and DEC 8400 codes will perform comparably.

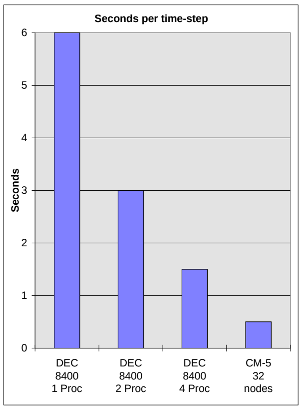

Figure 9: Performance of the full model on a 32 node CM-5 computer and a DEC 8400 5/300 server.

5 Conclusion

We believe that there are many researchers who use high-performance computing in their research as we do. On a daily basis, we run modest-sized problems on a modest-sized 128-processor CM-5, of which we usually have access to 32 processors at a time. Our preliminary results indicate that the DEC 8400 and other machines like it may be a viable alternative to more traditional vector and parallel supercomputers like the CM-5. The wider availability of these machines could be beneficial to research groups many of which could make use of a personal 100-1000 megaflop/s computing facility. The combination of the DEC 8400 hardware and its high-level HPF programming environment is an effective one for our application, although clearly there are opportunities for improving the current compiler technology. Although the Sun 5000 hardware performed well and we its programming environment is not geared toward casual programmers. We also note that we had to re-write most of application kernel to take full advantage of the DEC system, albeit in a high-level lenguage.

Acknowledgments

This research is a joint effort of the MIT Laboratory for Computer Science and the MIT Center for Meteorology and Physical Oceanography. The research of Andrew Shaw is partially funded by ARPA under ONR contract N00014-92-J-1310. The research of Chris Hill is partially funded by ONR contract N00014-95-1-0967, and University of California, Scripps Institute of Oceanography grant PO#10037358.

We would also like to thank Digital Equipment Corporation for running the Technical Computing Contest, and the Pittsburgh Supercomputing Center for giving us access to the DEC 8400 SMP. Experiments run on the CM-5 were done under the SCOUT project at MIT under ARPA contract #MDA972-92-J-1032.

References

1. J. C. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic and non-hydrostatic ocean modeling. to be published *J. Geophys Res.*, 1997.
2. J. C. Marshall, A. Adcroft, C. Hill, and L. Perelman. A finite-volume, incompressible Navier-Stokes model for studies of the ocean on parallel computers. to be published *J. Geophys Res.*, 1997.
3. F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow. *Phys. Fluids*, 8:2182, 1965.
4. R. Brugge, H.L. Jones, and J.C. 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.
5. J.C. Marshall and F. Molteni. Toward a dynamical understanding of planetary-scale flow regimes. *J. Atmos. Sci.*, 50(12):1792–1818, 1993.
6. M.J. Miller and R.P. Pearce. A three-dimensional primitive equation model of cumulonimbus convection. *Quart. J. R. Meteorol. Soc.*, 100:133–154, 1974.
7. M.J. Miller. On the use of pressure as vertical co-ordinate in modelling. *Quart. J. R. Meteorol. Soc.*, 100:155–162, 1974.
8. TMC. *The CM-5 Technical Summary*. Thinking Machines Corporation, Cambridge, MA, 1991.
9. Fujitsu. *VPP500 Vector Parallel Processor*. Fujitsu America Inc., San Jose, CA, 1992.
10. D.M. Fenwick et. al. The alphaserver 8000 series: High-end server plat-

form development. *Digital Technical Journal*, 7(1), 1995.
11. J.H. Edmondson et. al. Internal organization of the alpha 21164, a 300-mhz 64-bit quad-issue cmos risc microporcessor. *Digital Technical Journal*, 7(1), 1995.
12. SUN. *The UltraSPARC Programmer Reference Manual.* Sun Microsystems, Mtn. View, CA, 1995.
13. D.P. Manley C. Kamath, R. Ho. Dxml: A high-performance scientific subroutine library. *Digital Technical Journal*, 6(3), 1994.