

Scaling the Unscalable: A Case Study on the AlphaServer SC *

Patrick H. Worley[†]

Abstract

A case study of the optimization of a climate modeling application on the Compaq AlphaServer SC at Pittsburgh Supercomputer Center is used to illustrate tools and techniques that are important to achieving good performance scaling.

1 Introduction

If we define scalability to be the ability to use an arbitrarily large number of processors productively, then no code solving a fixed size problem “scales”. At some point, there is no work left to assign to additional processors, and processor scaling necessarily stops. Typically, parallel overhead costs dominate computation costs long before this point is reached, limiting the number of processors that can be used to a much smaller number.

While no fixed size problem scales in this asymptotic sense, a scalability analysis, determining performance over a range of problem sizes and processor counts, can be useful in:

- diagnosing performance problems;
- optimizing assignment of processors to tasks in a parallel workload;
- choosing between parallel or numerical algorithms, when all else is equal.

A code’s scaling behavior can also be viewed as an aspect of its performance portability, that is, its ability to make use of available resources. A code’s scalability will be enhanced if it can be easily tuned for a given architecture, problem instance, and processor count. While this tuning may involve a change of algorithm, alternative implementations of a fixed algorithm can be just as important, as many who have ported codes between vector and nonvector systems can testify.

In this study, we look at the performance and scalability of an application benchmark used in the global climate modeling community on the 3000 processor Compaq AlphaServer SC system at the Pittsburgh Supercomputing Center (PSC). The approach used to improve scalability includes supporting multiple parallel algorithms and parallel algorithm implementations in the application code, using separate kernels codes for optimization studies, and using application-specific performance instrumentation and analysis tools to diagnose observed performance.

2 CCM/MP-2D

CCM/MP-2D is a variant of version 3.6.6 of the National Center for Atmospheric Research (NCAR) Community Climate Model (CCM) [9] that uses a 2D domain decomposition of the 3D computational

*This research was sponsored by the Atmospheric and Climate Research Division and the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

[†]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367 (worleyph@ornl.gov)

domain [3]. The production version of the CCM uses a 1D domain decomposition, which severely limits the number of processors that can be used. CCM/MP-2D was developed to demonstrate the performance gains available from introducing additional parallelism. The algorithms developed within CCM/MP-2D are currently being implemented in the Community Atmospheric Model (CAM), the successor to the CCM.

CCM/MP-2D is a spectral atmospheric general circulation model, modeling atmospheric flows on the sphere using a (longitude, vertical, latitude) tensor product computational grid. Performance is determined by that of two distinctly different computational kernels, the dynamical core (“dynamics”) and the physical parameterizations (“physics”). The dynamics use a spectral transform algorithm to advance the winds and temperature [6, 11], and a semi-Lagrangian transport algorithm to advect moisture [12]. The physical parameterizations include cloud physics and solar radiation.

The parallel algorithms are based on mapping the computational domain onto a 2D logical processor grid. A tensor-product longitude/latitude decomposition is used. The number of row processors, decomposing longitude, and the number of column processors, decomposing latitude, are compile-time options.

The physical parameterizations are independent between vertical columns (fixed longitude/latitude coordinates). By using a 2D longitude/latitude domain decomposition, the physics is perfectly parallel and requires no interprocessor communication. The performance of the physics is affected by the decomposition all the same, both from load imbalances and changes in serial performance as the amount of work assigned to each processor changes.

The spectral transform algorithm requires the computation of Fast Fourier Transforms (FFT) over longitude for each latitude and vertical level index and a Legendre transform (approximated using Gauss quadrature) over latitude for each wavenumber (was longitude) and vertical index. Two different parallel algorithms for the FFTs are supported in the code. The first is a distributed algorithm, mixing interprocessor communication and computations. The second is a transpose-based algorithm that changes the decomposition from 2D longitude/latitude to 2D vertical/latitude, computes FFTs using a serial algorithm, then transposes back to the original decomposition. For both algorithms, interprocessor communication occurs only between processors in the same processor row. The parallel Legendre transform computes local contributions to the transform coefficients, then completes the transform using the allreduce collective communication operator. For this algorithm, interprocessor communication occurs only between processors in the same processor column. The parallelization of the semi-Lagrangian transport requires the exchange of boundary data between neighboring row and column processors. The parallel FFT, Legendre transform, and semi-Lagrangian transport algorithms each have numerous parallel implementation options. The options are differentiated by their message counts, message volumes, and MPI communication protocols. For more details on the parallel algorithms used in CCM/MP-2D see [3, 5].

The parallel algorithms for both the spectral transform and semi-Lagrangian transport require significant interprocessor communication. For example, the total amount of data communicated per simulation day for CCM/MP-2D as a function of the number of processors is described in Fig. 1. As can be seen, 1D decompositions are preferable (by this metric) for small processor counts, but 2D decompositions have performance advantages even before the 1D decompositions reach their maximum processor counts.

The target problem resolution for CCM/MP-2D is T42L18, which represents a 128x64x18 computational grid. While not currently valid for science studies, we also have data sets corresponding to a problem size of T170L18, representing a 512x256x18 computational grid. In this study we examine the performance of CCM/MP-2D for both problem sizes.

3 Methodology

Often the difficult aspect of a scaling study is understanding what is observed. Is poor scaling intrinsic to the numerical algorithm or is it an artifact that can be removed by tuning? Serial and parallel kernels extracted from the application code can be used in experiments to understand aspects of code scaling that are difficult to examine, otherwise. Extracting kernels is a nontrivial exercise, but, once done, the payoff can be significant.

For CCM/MP-2D, we have available two kernels, the Parallel Spectral Transform Shallow Water Model (PSTSWM) and the Column Radiation Model (CRM). PSTSWM solves the shallow water equations on the sphere using a parallel spectral transform algorithm very similar to that used in the CCM/MP-2D dynamical

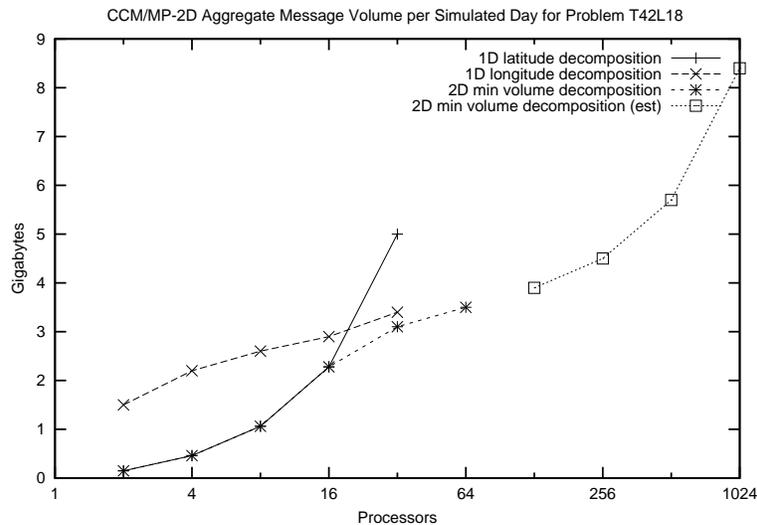


Figure 1: CCM/MP-2D message volume.

core [4, 14]. CRM is a standalone version of the CCM column radiation model, which is the most expensive part of the CCM physics [10]. These codes have vastly different computational “signatures”. Over 99% of the floating point operations in PSTSWM are multiplies and adds, while over 6% of the floating point operations in CRM are `sqrt`. PSTSWM moves through large data arrays with unit or constant stride and little operand reuse. In contrast, CRM is compute-intensive, making much smaller demands on memory.

PSTSWM and CRM are both used in studies for determining the effect of scaling on serial performance. PSTSWM is further used in parallel studies, to determine which parallel implementation options to use with CCM/MP-2D. As PSTSWM is not identical to CCM/MP-2D, it is only used to eliminate poor choices. CCM/MP-2D is used directly in experiments to identify the best options from this restricted set.

CCM/MP-2D has also been instrumented to collect profile and/or MPI trace data. Careful consideration of the nature of the code allows us to identify the major contributors to performance loss

4 Platforms

This study focuses on the performance optimization and evaluation of CCM/MP-2D on the AlphaServer SC at PSC. The PSC system is made up of 750 ES45 4-way symmetric multiprocessor (SMP) nodes and a Quadrics “fat-tree” interconnect. Each node has two ports into the network. The processors in the ES45 are the 1GHz Alpha 21264 (EV68).

Some of the more interesting results in the experiments described later are the ways in which the AlphaServer SC is not subject to certain common performance problems. To illuminate these positive results, we also include performance data from other platforms:

- IBM p690 at Oak Ridge National Laboratory (ORNL): 32-way SMP node using POWER4 processors running at 1.3 GHz.
- Compaq AlphaServer SC at ORNL: 64 ES40 4-way SMP nodes and a Quadrics switch. Each node has one port into the network. The processors are the 667MHz Alpha 21264a (EV67).
- IBM SP at the National Energy Research Supercomputer Center (NERSC): 184 Nighthawk II 16-way SMP nodes and SP Switch II (Colony). Each node has two ports into the network. The processors are 375MHz Power3-II processors.
- IBM SP at ORNL: 176 Winterhawk II 4-way SMP nodes and SP Switch. Each node has one port into the network. The processors are 375MHz Power3-II processors.

- Cray Research T3E-99 at NERSC: 644 Alpha EV5 processors running at 450 MHz.

5 Serial Benchmarks

An important aspect of scaling for fixed size problems is the impact on per processor “serial” performance. While performance can sometimes improve as the per processor problem granularity decreases, producing the oft observed superlinear speedup effect [7], performance can also decrease.

Figure 2 describes the serial performance of PSTSWM as the horizontal resolution changes for a fixed number of vertical levels (18), and as the number of vertical levels changes for a fixed horizontal resolution (T42). The first experiment approximates the change in problem granularity as the number of processors increases when using the distributed FFT algorithm. The second experiment approximates the change in problem granularity when using the transpose-based parallel algorithm. Note that two curves are given for the ES45 system, one using FFT routines from the Compaq Extended Math Library (CXML) and one not. From these data, there is a strong trend (across all platforms) of improved performance as the number of vertical levels decreases. This should improve the scaling behavior of the transpose algorithm. The performance on the ES45 when not using CXML FFT routines is relatively insensitive to changes in horizontal resolution. In contrast, when using CXML FFT routines, performance decreases with decreasing horizontal resolution. This sensitivity to the use of the CXML library led us to include both CXML and non-CXML runs when doing the final CCM/MP-2D benchmarking.

Single processor experiments illuminate the effect of the memory hierarchy on the performance when scaling. However, all of the target systems are clusters of SMP nodes, and multiple MPI processes within a node can contend for node memory bandwidth. Figure 3 describes the average per processor performance when all processors in the SMP node are computing identical, independent problems. Thus, on the ES45, 4 instances are running simultaneously, while on the p690, 32 instances are running. From these data, the trend of improving performance with decreasing vertical resolution continues to hold. However, the slope and achieved performance change (dramatically) for some of the platforms. For the most part, memory contention is not an issue on the ES45. Similarly, the performance sensitivity with respect to horizontal resolution is almost the same whether using one or all processors in the ES45. The same is not true on the p690, where performance degradation due to contention is significant, and greater parallelism (smaller granularity) can mitigate this loss.

As mentioned earlier, the radiation calculations in CRM are vertical column based, and computations between vertical columns are independent. The index ordering in the field arrays is (longitude, vertical, latitude) and the basic loop structure is

```
DO J=1,NLAT
  DO K=1,NVER
    DO I=1,NLON
      (radiation calculation)
    ENDDO
  ENDDO
ENDDO
```

Figure 4 describes the serial performance of CRM as a function of the length of the inner loop (NLON), representing the effect on performance of decomposing the longitude dimension. If NLON is large, then it may be possible to pipeline the expensive intrinsic functions (`sqrt`, `log`, `exp`, `pow`). If NLON is small, then dependence on memory performance is minimized.

The performance of CRM varies by approximately 10% as a function of longitude dimension on the ES45, with a slightly greater decrease for large NLON when using all 4 processors. The optimum value is approximately NLON=4. In contrast, performance on the IBM POWER processors degrades dramatically when NLON < 16, which will limit performance for large processor counts. When NLON is large, per processor performance on the p690 is approximately 30% less when using all of the processors than when using one, but the performance sensitivity to NLON is qualitatively the same.

Figure 5 examines two other performance issues that can arise. The first looks at the effect of compiler options on the previous results. The second looks at the effect of setting loop bounds and/or array decla-

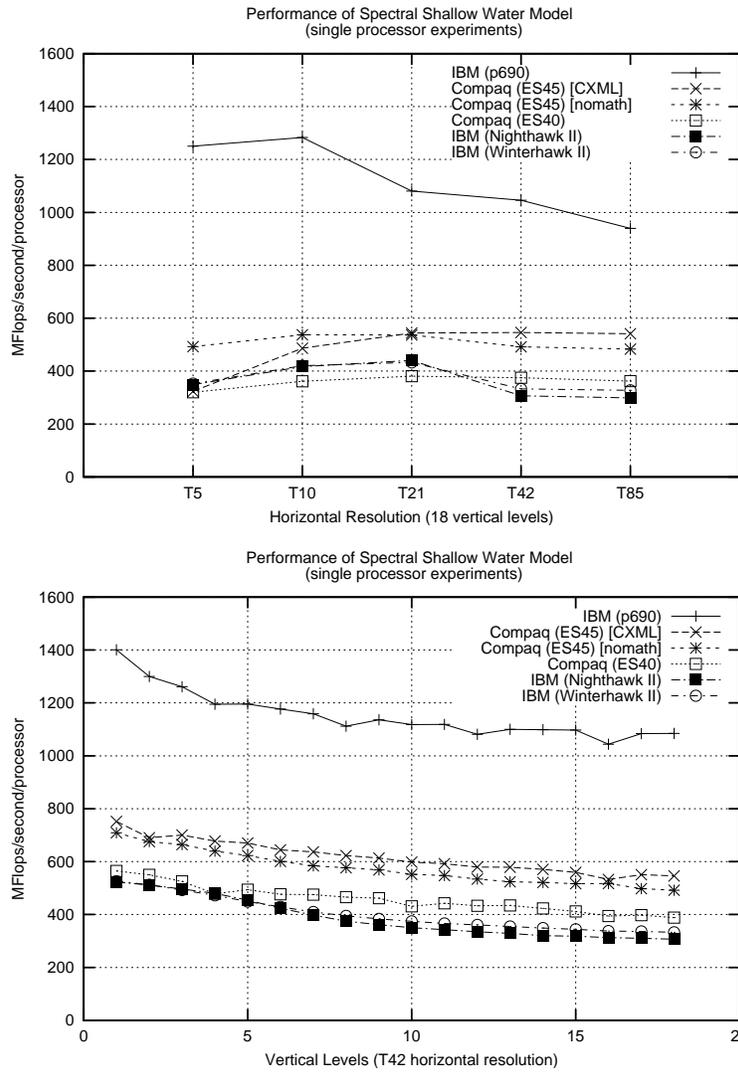


Figure 2: PSTSWM serial computational rate when using one processor.

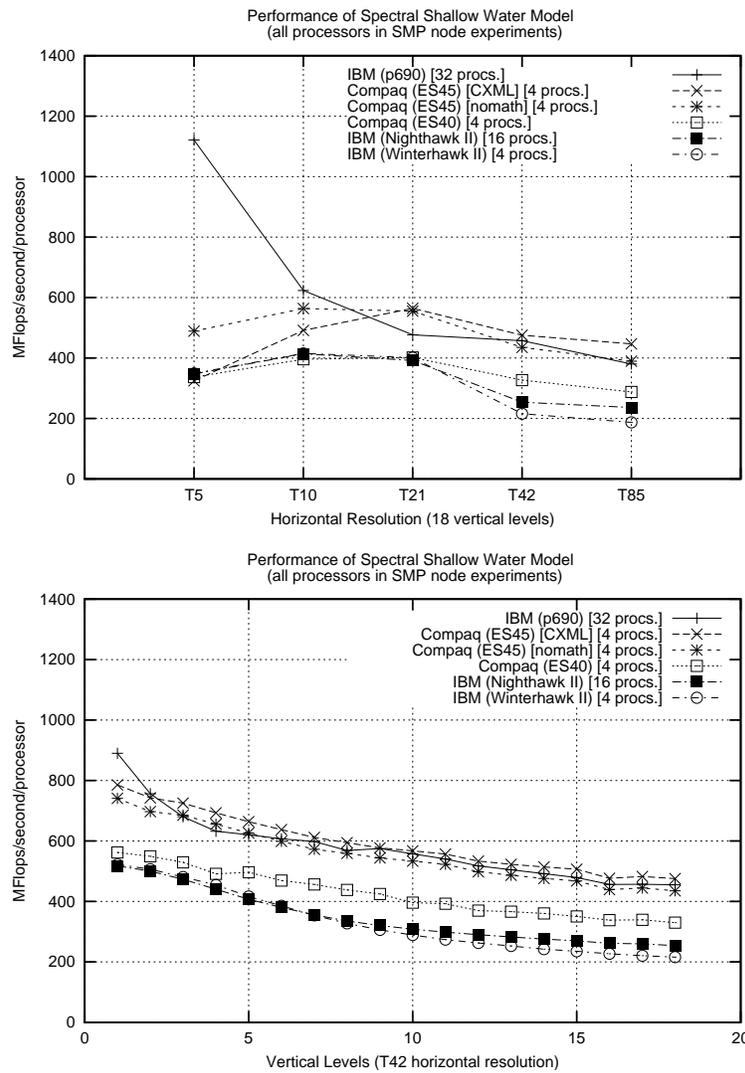


Figure 3: PSTSWM serial computational rate when using all processors.

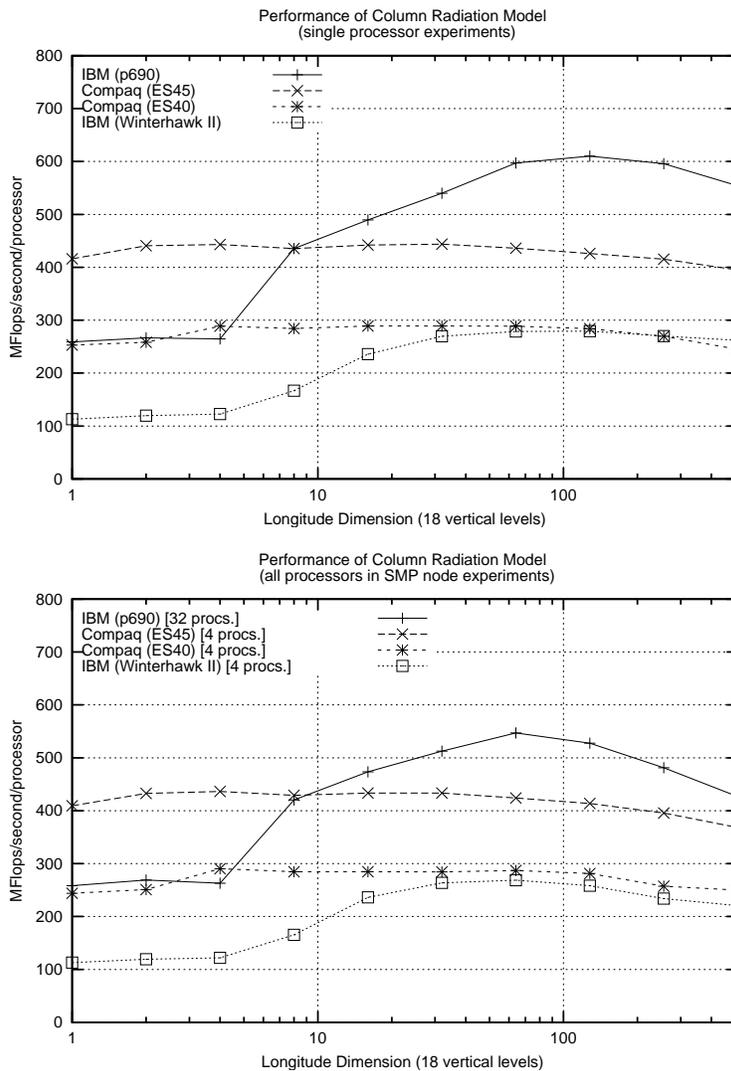


Figure 4: CRM serial computational rate.

rations at runtime. The ES45 system is not particularly sensitive to the compiler options used beyond the default optimization levels. In contrast, the p690 (and other IBM systems) are very sensitive. If higher order transformations (`hot`) are not enabled, then performance on the p690 is not sensitive to the longitude dimension, but then the performance is poor relative to what can be achieved if `hot` is specified.

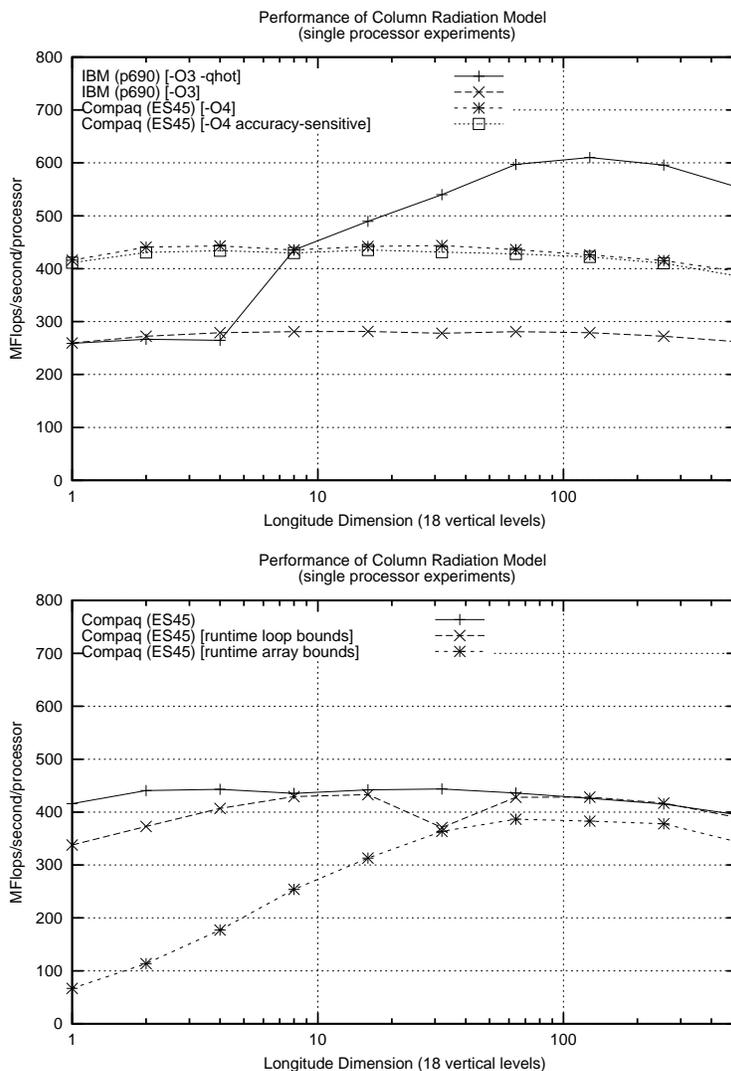


Figure 5: CRM serial computational rate.

The second graph in Fig. 5 examines the effect of specifying at runtime parallel algorithm options that impact loop bounds and array declarations. Runtime declarations are a convenience when running scaling experiments and other sorts of algorithm tuning, as far fewer executables are needed. This graph shows the danger of doing so. Specifying the longitude loop bounds at runtime (with a sufficiently large compile-time specification of array sizes) leads to some performance degradation for small NLON, and an array alignment problem for NLON=32 that the compiler was able to avoid when the bound was specified at compile-time. Specifying both the longitude loop bound and array dimension at runtime degrades performance for small NLON significantly and would adversely impact performance for large processor counts. These results led us to specify the virtual processor array and problem size at compile-time when benchmarking CCM/MP-2D on the PSC system.

6 Parallel Benchmarks

CCM/MP-2D has a large number of parallel algorithm options, too many to conveniently investigate using the full application code. PSTSWM includes all of the parallel algorithm options available in CCM/MP-2D for the spectral transform. Even using PSTSWM, it is too expensive to examine all algorithm options for all logical processor grid configurations. Instead, we isolate the parallel FFT and parallel Legendre algorithm tuning by running a sequence of 1D experiments, decomposing only longitude or only latitude. We run experiments using 8, 16, and 32 processors, examining approximately 800 algorithm options for each processor count and problem size. These 1D experiments are followed by three sets of 2D experiments, utilizing processor grids of size 4x16, 8x8, and 16x4.

Because the 1D experiments isolate individual parallel algorithms, they emphasize differences more clearly than the 2D experiments and are the most convenient for choosing candidate algorithms. However, the 1D experiments are run across contiguous nodes, while in a production run a processor row or processor column may not involve contiguous nodes. On most recent systems, the processor topology has not changed the performance ranking of the parallel algorithms - what is good or bad on one topology is good or bad on another. On the AlphaServer SC, the MPI collective commands `MPI_ALLREDUCE` and `MPI_ALLTOALLV` use hardware features in the Quadrics switch to achieve excellent performance, but this is only available if the collectives run over contiguous nodes. In consequence, the parallel implementations using `MPI_ALLREDUCE` and `MPI_ALLTOALLV` were the best performers in almost all of the 1D experiments. This skewed our initial selection of candidate algorithms and forced us to revisit this phase of the algorithm identification process halfway through the next phase.

We will not attempt to do more than quickly summarize the results of this extensive tuning exercise. In practice, we do not examine the results closely, simply running each experiment 3 times to eliminate atypical perturbations in the timing, then ranking the results to pick a set of candidate parallel algorithm and implementation options.

Other than the MPI collectives, the candidate algorithms on the system at PSC are not strikingly different from good algorithms identified on other recent systems.

- For small granularities (small problems, large processor counts), it is best to use algorithms that minimize message counts and to use protocols that use `MPI_SENDRECV`.
- For moderate to large granularities, it is best to use algorithms that minimize message volume and to use protocols that hide latency and load imbalance by preposting receive requests using `MPI_Irecv`.
- For the largest granularities, it is best to use algorithms that simplify the demands on the topology, e.g., ring-based algorithms. Good protocols either precisely control flow (MPI synchronous commands) or loosen dependencies (preposting `MPI_Irecv` requests).

Thus “scalability” requires that this code be adaptable or reconfigurable in order to run efficiently for a range of problem sizes and processor counts.

Note that on the earlier ES40-based AlphaServer SC, there was a strong preference for *not* using bidirectional communication [13]. For example, performance in a swap operation was improved if the send from one process was completed before the send from the other process was initiated. The ES45-based system, with its dual ports into the Quadrics network, does not show this bias.

7 CCM/MP-2D Tuning

Using the PSTSWM results, we selected 15 candidate parallel algorithms. This translates into 30 experiments per problem size and processor grid as each candidate algorithm is run with both row-major (processors in a row are contiguous) and column-major (processors in a column are contiguous) processor grid configurations. Both distributed and transpose-based FFT algorithms were included, as the PSTSWM experiments do not allow us to accurately discriminate between the two approaches. Fifteen candidate algorithms are more than we typically carry into the second phase of the tuning, but the uncertainties introduced by the hardware acceleration of the MPI collectives required it.

For each processor count (2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048), we examined the performance of CCM/MP-2D for a number of different processor grid aspect ratios. For example, for the 16 processor experiments we examined logical processor grids of size 16x1, 8x2, 4x4, 2x8, and 1x16. Testing was selective, as it was often clear after running experiments on a few grids what the trend was and that certain aspect ratios would not be efficient. For these tuning experiments, CCM/MP-2D was run for 32 timesteps. This was sufficient to exercise the parallel algorithms, but avoided the most expensive radiation calculations. The scatterplots in Fig. 6 indicate the performance variability that arises in these experiments, both between the candidate algorithms and for different processor grid aspect ratios. Note that the variability should not be unexpected. Each algorithm was chosen because it was expected to be a good choice for a particular range of problem sizes and processor counts. There was no requirement that it perform well outside of this range. Still, a factor of 2 or 3 difference in performance is significant and indicates the importance of tuning the parallel algorithms.

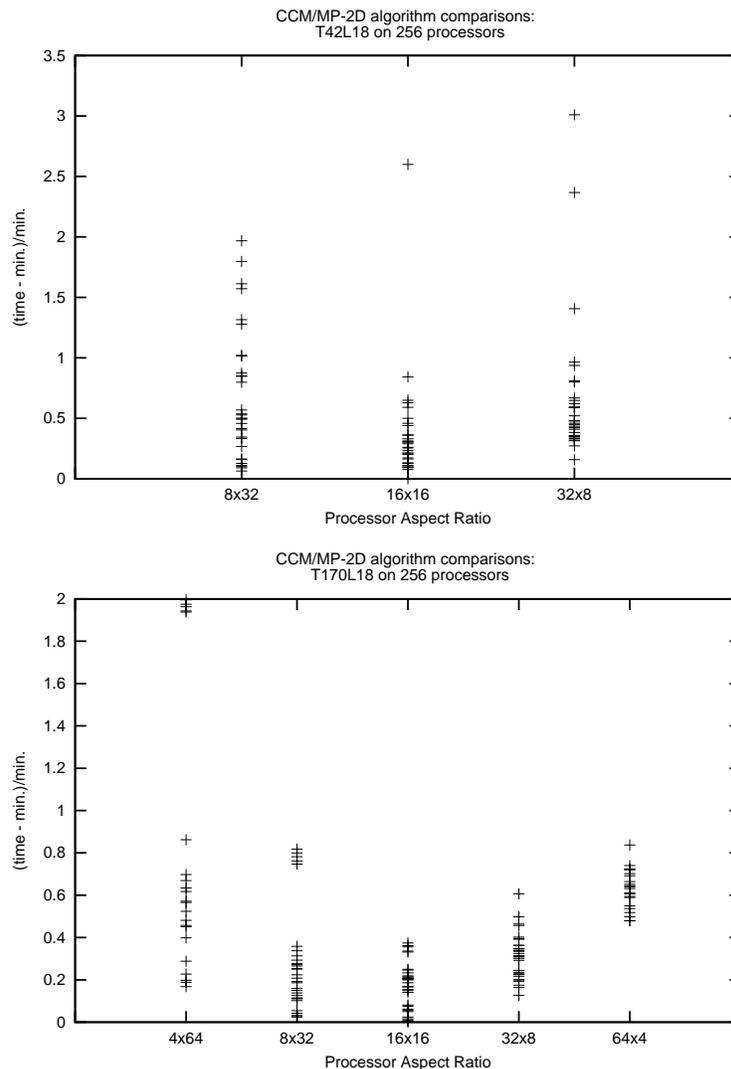


Figure 6: Performance comparison of CCM/MP-2D candidate algorithms.

Table 1 describes, in general terms, the parallel algorithms that achieved the best performance for each processor count and problem size. The `grid` row indicates the aspect ratio of the logical processor grid. The `alg` row indicates whether a distributed (D) or transpose (T) parallel FFT was used. The `coll?` row indicates whether `MPI_ALLREDUCE` was used. (Algorithms using `MPI_ALLTOALLV` were never best.)

From these data, MPI_ALLREDUCE is selected only for the small granularity problems, when minimizing latency is more important than minimizing message volume.

Problem	Processors										
	2	4	8	16	32	64	128	256	512	1024	2048
T42L18											
grid	1x2	1x4	1x8	1x16	2x16	4x16	8x16	16x16	32x16	32x32	-
alg	-	-	-	-	D	T	T	T	D	D	-
coll?	yes	no	no	no	no	no	yes	yes	yes	yes	-
T170L18											
grid	-	-	1x8	1x16	4x8	4x16	8x16	16x16	16x32	16x64	32x64
alg	-	-	-	-	D	T	T	T	T	T	T
coll?	-	-	no	no	no	no	no	no	no	no	no

Table 1: Optimal algorithms for CCM/MP-2D on AlphaServer SC at PSC

8 CCM/MP-2D Performance Analysis

In this section we describe the benchmark results and performance analysis for CCM/MP-2D. For each platform, we use the best observed parallel algorithm and compiler options. Figures 7-8 illustrate the best performance observed for problem sizes T42L18 and T170L18, in terms of both the number of simulation years per day and the per processor computational rate. From these data, CCM/MP-2D shows good performance scaling on the PSC Compaq system up to 256 processors for T42L18 and up to 1024 processors for T170L18. The code shows similar behavior on the other platforms, using the platform-specific optimizations.

Figure 9 describes the sources of performance loss on the PSC system as a function of processor count. CCM/MP-2D was hand-instrumented and run with MPI and user-event profiling. Postprocessing scripts used these data to derive where the code spent time in tasks that do not exist in equivalent serial runs. The overhead categories include time spent in interprocessor communication (**Comm**), the increase in computational time (compared to perfect speedup) due to load imbalances (**Imbal**), time spent in memory copies associated with interprocessor communication (**Copy**), and the increase in computational time due to duplicate computations (**Dupl**). The overhead sources are graphed accumulatively. For example, the curve labelled **plus Dupl** represents the percentage of the total runtime spent in all of the identified overhead tasks. Note, however, that the loss in efficiency is sometimes different, either greater or smaller. As shown with the serial benchmarks, when the computational granularity and memory access patterns change with the processor count, the computational rate can change. This is at least one source of discrepancy in the analysis. As the differences are not large, this validates the qualitative accuracy of the analysis, and confirms the serial benchmark results that computational rate on the ES45 is not very sensitive to problem granularity.

From these data, it is clear that the overwhelming majority of the parallel overhead is due to interprocessor communication. Performance can be improved when using more than 256 and 1024 processors, for T42L18 and T170L18, respectively, by using only one or two processors per SMP node. This decreases contention for access to the network, but also requires using two or four times as many nodes.

The next generation of the model (CAM) will have significantly more physics, decreasing the impact of the communication overhead and improving scalability. The increased scalability comes at the cost of decreased throughput. However, it does imply that the current approach to parallelization will be viable for this next generation code for even larger processor counts.

9 Conclusion

Our experience in tuning CCM3/MP-2D leads us to argue that achieving good scalability is not only a function of the numerical and parallel algorithms, but also of the infrastructure available for optimizing these algorithms. We have found representative kernel codes, support for multiple or hybrid parallel algorithms,

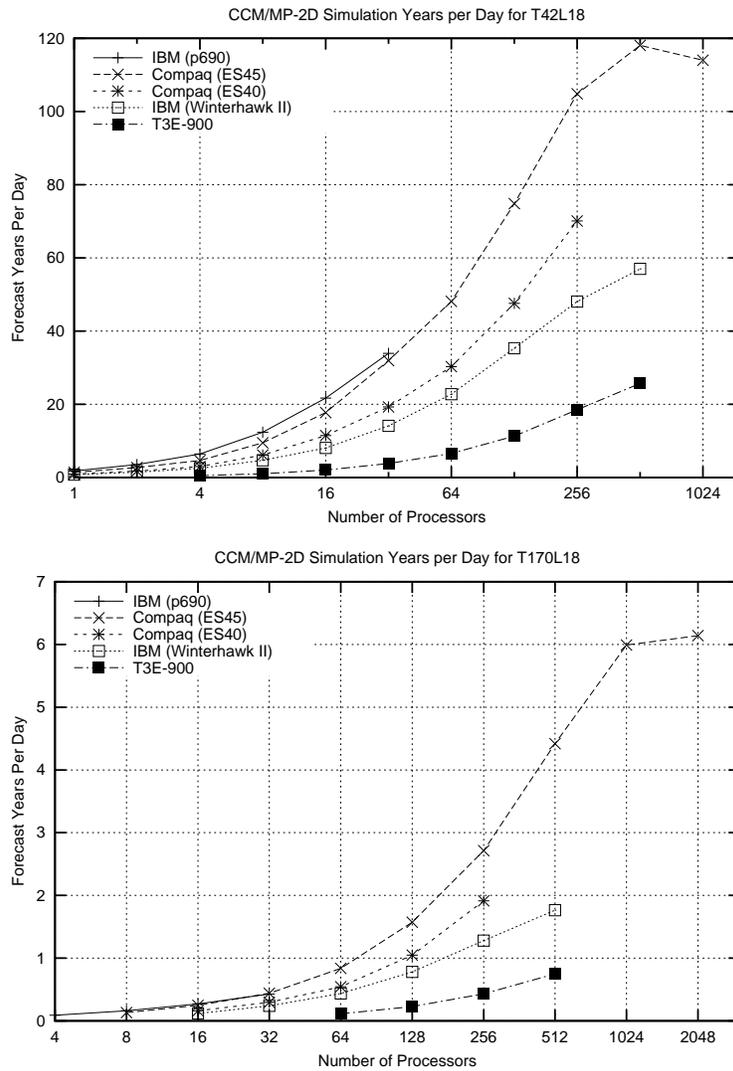


Figure 7: CCM/MP-2D simulation years per day of computation.

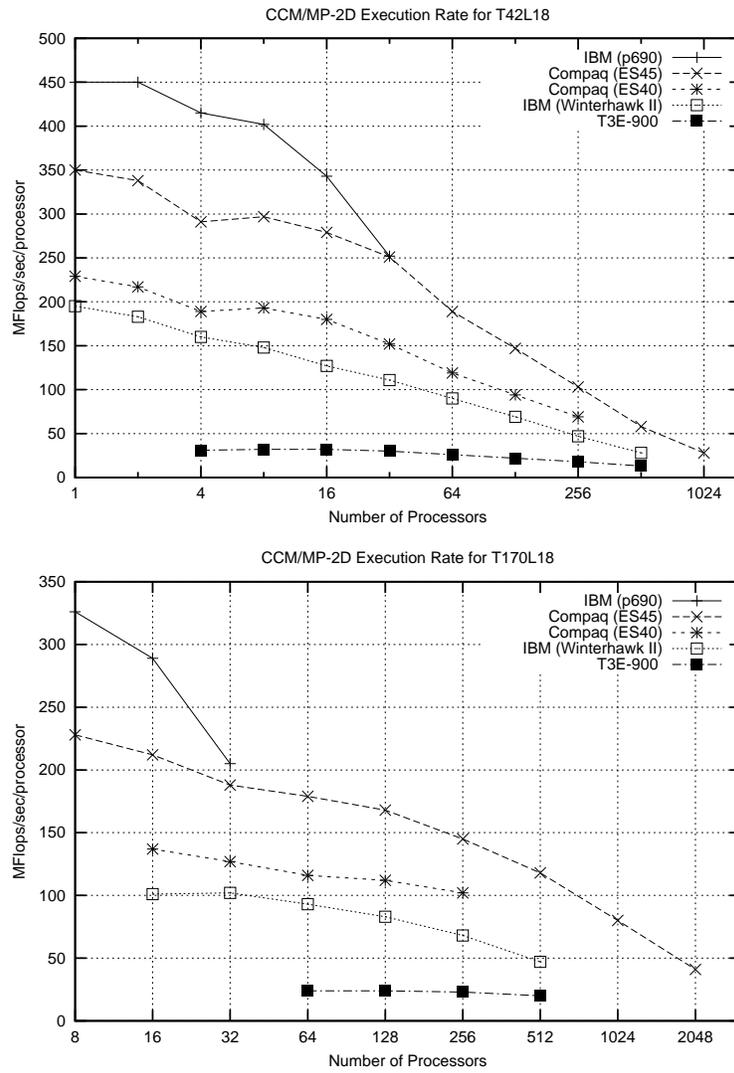


Figure 8: CCM/MP-2D per processor computational rate.

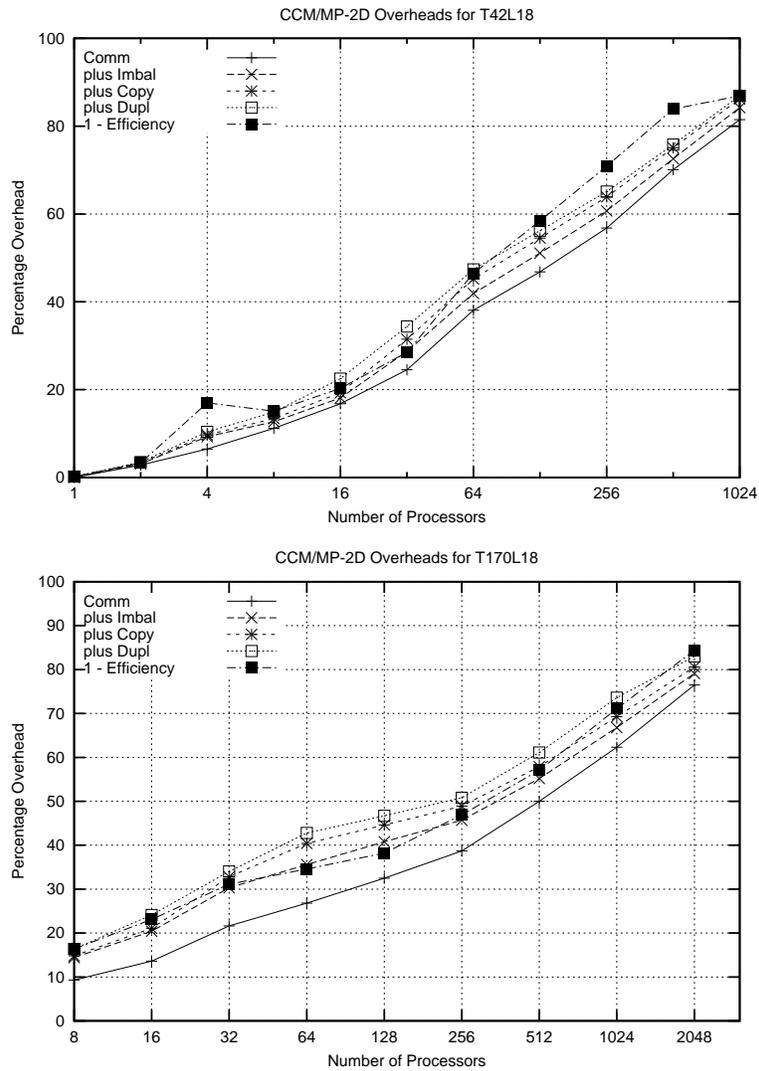


Figure 9: Sources of CCM/MP-2D performance loss on the Compaq AlphaServer SC at PSC.

performance measurement tools, and a carefully thought out optimization methodology to be crucial for improving the scalability of otherwise “unscalable” codes.

Much more could be done than was presented here.

- While we trust the qualitative results of the kernel experiments, we have not yet established that the kernel performance can be used to predict application performance quantitatively. This is less important when porting an existing code, but it can be very important if the performance of a kernel is used to design new code.
- The extensive experimentation used to tune the application is expensive. Accurate performance models could accelerate the optimization process significantly [2].
- Embedding multiple or hybrid parallel algorithms in an application code makes it feasible to implement automatic or runtime optimization. Access to the largest configurations of a system may be difficult a priori. Runtime optimizations would allow the code to adapt to novel performance aspects [8].
- Vendors need to continue to develop tuned hybrid collective communication algorithms [1].

10 Acknowledgements

We gratefully acknowledge the National Science Foundation and the Pittsburgh Supercomputing Center for access to the Compaq AlphaServer SC at PSC, the Center for Computational Sciences at ORNL for access to the IBM p690, IBM SP, and Compaq AlphaServer SC at ORNL, and the National Energy Research Scientific Computing Center for access to the IBM SP and the Cray Research T3E-900 at NERSC.

References

- [1] M. BARNETT, L. SHULER, S. GUPTA, D. G. PAYNE, R. VAN DE GEIJN, AND J. WATTS, *Building a high-performance collective communication library*, in Supercomputing, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 107–116.
- [2] J. BREHM, P. H. WORLEY, AND M. MADHUKAR, *Performance modeling for SPMD message-passing programs*, Concurrency: Practice and Experience, 10 (1998), pp. 333–357.
- [3] J. B. DRAKE, I. T. FOSTER, J. G. MICHALAKES, B. TOONEN, AND P. H. WORLEY, *Design and performance of a scalable parallel community climate model*, Parallel Computing, 21 (1995), pp. 1571–1591.
- [4] I. T. FOSTER, B. TOONEN, AND P. H. WORLEY, *Performance of parallel computers for spectral atmospheric models*, J. Atm. Oceanic Tech, 13 (1996), pp. 1031–1045.
- [5] I. T. FOSTER AND P. H. WORLEY, *Parallel algorithms for the spectral transform method*, SIAM J. Sci. Comput., 18 (1997), pp. 806–837.
- [6] J. J. HACK, B. A. BOVILLE, B. P. BRIEGLEB, J. T. KIEHL, P. J. RASCH, AND D. L. WILLIAMSON, *Description of the NCAR Community Climate Model (CCM2)*, NCAR Tech. Note NCAR/TN–382+STR, National Center for Atmospheric Research, Boulder, Colo., 1992.
- [7] D. P. HELMBOLD AND C. E. MCDOWELL, *Modeling Speedup greater than n*, IEEE Trans. Par. Dist. Sys., 1 (1990), pp. 250–256.
- [8] P. J. KELEHER, J. K. HOLLINGSWORTH, AND D. PERKOVIC, *Exploiting application alternatives*, in Proceedings of the 19th International Conference on Distributed Computing Systems, IEEE Computer Society Press, Los Alamitos, CA, May 1999, pp. 384–392.

- [9] J. T. KIEHL, J. J. HACK, G. BONAN, B. A. BOVILLE, D. L. WILLIAMSON, AND P. J. RASCH, *The National Center for Atmospheric Research Community Climate Model: CCM3*, *J. Climate*, 11 (1998), pp. 1131–1149.
- [10] J. T. KIEHL, J. J. HACK, G. B. BONAN, B. A. BOVILLE, B. P. BRIEGLEB, D. L. WILLIAMSON, AND P. J. RASCH, *Description of the NCAR Community Climate Model (CCM3)*, NCAR Tech. Note NCAR/TN-420+STR, National Center for Atmospheric Research, Boulder, Colo., September 1996.
- [11] W. WASHINGTON AND C. PARKINSON, *An Introduction to Three-Dimensional Climate Modeling*, University Science Books, Mill Valley, CA, 1986.
- [12] D. L. WILLIAMSON AND P. J. RASCH, *Two-dimensional semi-Lagrangian transport with shape-preserving interpolation*, *Mon. Wea. Rev.*, 117 (1989), pp. 102–129.
- [13] P. H. WORLEY, *Performance evaluation of the IBM SP and the Compaq Alphaserver SC*, in Proceedings of the 14th International Conference on Supercomputing, Association for Computing Machinery, New York, NY, 2000, pp. 235–244.
- [14] P. H. WORLEY AND B. TOONEN, *A users' guide to PSTSWM*, Tech. Rep. ORNL/TM-12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.