Benchmarking using the Community Atmospheric Model

Patrick H. Worley[1]
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831–6016
Email: worleyph@ornl.gov

***Abstract*—The Community Atmospheric Model (CAM) is a global atmosphere model developed for the weather and climate research communities. CAM also serves as the atmospheric component of the Community Climate System Model (CCSM). As a community model, it is important that CAM run efficiently on different architectures, and that it be easily ported to and optimized on new platforms. The current version of CAM contains a number of performance portability features - compile-time or runtime parameters that can be used to optimize performance for a given platform, problem or processor count. The large number of tuning options can make benchmarking an arduous task. The paper describes these options and how optimization is managed to make it feasible for evaluation of early systems. The paper also describes some of the performance sensitivities of selected platforms to the different tuning options.**

I. Introduction

Oak Ridge National Laboratory (ORNL) computer scientists have been evaluating “early systems” since the mid 1980s [9], [10], [12], [11], [13], [14], [15], [16], [20], [28], [32], [34], including, most recently, the SGI Altix, Cray XD1, Cray XT3, and Cray X1. Early systems are low serial number or pre-commerical release versions of computer systems, often delivered with unoptimized or missing system software. The goal of the evaluation is to comment on the promise of the system in a fast, fair, and open way. Conducting evaluations that are both fast and fair is difficult in practice, and compromises are often required. However, if a new architecture provides a novel feature that standard benchmarks do not examine, simply running the standard benchmarks provides no insight into the utility of that feature. Therefore, we develop and use custom benchmarks, as appropriate, as well as use standard benchmarks.

We typically take a hierarchical approach, using microkernels to measure subsystem performance, kernels to compare programming paradigms and to measure whole system performance, and compact or full application codes to estimate the performance in terms meaningful to the application users. The microkernel and kernel results are used to help understand the full application performance. They are also used to estimate what the performance might be if the application codes were modified to exploit the unique features of the system being evaluated. Unfortunately, it is rarely feasible within the time frame of an early evaluation to make significant modifications to the applications codes in order to verify these predictions.

We have also participated in the development of some of the application codes in our benchmark suites, helping introduce “performance portability” features into the codes. These features are compile-time or runtime tuning options that can be used to quickly optimize a code on a new platform, including, for example, support for different vendor-supplied math libraries or messaging layers and interfaces for adding support for new ones. While not perfect, these performance portable codes make it easier to perform a fair evaluation. This paper describes one such application code, the Community Atmospheric Model (CAM), and how it is used in our benchmarking.

CAM is a global atmosphere model developed at the National Science Foundation’s National Center for Atmospheric Research (NCAR) with contributions from researchers funded by the Department of Energy (DOE) and by the National Aeronautics and Space Administration (NASA) [2], [3]. CAM is used in both weather and climate research. In particular, CAM serves as the atmospheric component of the Community Climate System Model (CCSM) [1], [4]. As a community model, it is important that CAM run efficiently on different architectures, and that it be easily ported to and optimized on new platforms. CAM contains a number of compile-time and runtime parameters that can be used to optimize performance for a given platform, problem or processor count. When benchmarking with CAM it is important that the code be optimized to approximately the same level as for a production run, but no more. For example, production usage requires that the results be invariant to the number of processors used. This “reproducibility” requirement can disallow some compiler optimizations.

A consortium of DOE-funded mathematicians and computer scientists, including researchers at ORNL, began working with the Community Climate Model (CCM) [19], [23], the predecessor to CAM, in the early 1990s, developing a massively parallel version (CCM/MP-2D) in which many of the performance portability techniques later implemented in CAM were investigated [6], [7], [8]. As part of this effort we developed a parallel algorithm testbed code called the Parallel Spectral

[1]The work of this author was sponsored by 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.

Transform Shallow Water Model (PSTSWM) [31], [35], [36]. PSTSWM was our first attempt at developing a performance portable benchmark. PSTSWM was later included in PARKBENCH [21]. The importance of PSTSWM is that it includes a superset of the tuning options supported in CCM/MP-2D, and PSTSWM can be used to identify a subset of candidate options, minimizing the number of required CCM/MP-2D tuning runs. While less important for CAM, as PSTSWM is not as faithful a representation of the parallel algorithms in CAM, it is still used for this purpose.

II. Community Atmospheric Model

CAM is a mixed-mode parallel application code, using both the Message Passing Interface (MPI) [18] and OpenMP protocols [5]. CAM is characterized by two computational phases: the dynamics, which advances the evolution equations for the atmospheric flow, and the physics, which approximates subgrid phenomena such as precipitation processes, clouds, long- and short-wave radiation, and turbulent mixing [3]. The approximations in the physics are referred to as the physical parameterizations. Control moves between the dynamics and the physics at least once during each model simulation timestep. The number and order of these transitions depend on the numerical algorithm in the dynamics.

CAM includes multiple *dynamical cores (dycores)*, one of which is selected at compile-time. Three dycores are currently supported: the spectral Eulerian solver from CCM [23], [27], a spectral semi-Lagrangian solver [29], and a finite volume semi-Lagrangian solver [24]. The three dycores do not use the same computational grid. An explicit interface exists between the dynamics and the physics, and the physics data structures and parallelization strategies are independent from those in the dynamics. A dynamics-physics coupler moves data between data structures representing the dynamics state and the physics state.

The development of CAM was (and continues to be) a large community-wide effort. ORNL researchers are responsible for many of the performance portability features in the spectral Eulerian and spectral semi-Lagrangian dycores, most ported from CCM/MP-2D, and in the physics [33]. The finite volume dycore was originally developed at NASA, and most of the initial performance portability features were developed at NASA Goddard and at Lawrence Livermore National Laboratory [25], [26]. However, many individuals from many organizations have contributed to the software engineering of CAM, notably the CCSM Software Engineering Group at NCAR and David Parks of NEC Solutions America, who is responsible for the initial vectorization of many of the routines.

III. Performance Portability Features

We cannot do justice in this paper to the large number of tuning options currently supported in CAM. For more details, see [25], [26], [33]. Instead, we focus on those options that have proved most useful in recent benchmarking exercises.

A. General

Along with the usual tuning option of compiler flags, CAM has a number of code fragments, delimited by `cpp` directives, that are enabled only for certain systems. For example, there are a few routines for which we were unable to develop a single version that runs well on both vector and nonvector systems. `cpp` is used to choose either the vectorizable or the cache-friendly versions of these routines. `cpp` is also used to choose between math library routines with different calling sequences, FFT routines primarily.

B. Physics

The physics uses the same computational grid as the dynamics. All three dycores employ a tensor product longitude-latitude-vertical ($\mathtt{nlon} \times \mathtt{nlat} \times \mathtt{nver}$) grid covering the sphere. We refer to all grid points in this three-dimensional grid with a given horizontal location, differing only in the vertical coordinate, as a *vertical column*, or just *column*. The current physical parameterizations in CAM are based on vertical columns, and physics computations at a given timestep are independent between columns. The basic data structure in the physics is the *chunk*, an arbitrary collection of vertical columns. Grid points in a chunk are referenced by ($\mathtt{local\ column\ index}, \mathtt{vertical\ index}$). A "chunked" array is declared as ($\mathtt{pcols}, \mathtt{nver}, \mathtt{nchunks}$) and the loop structure is

```
do j=1,nchunks
  do k=1,nver
    do i=1,ncols(j)
      (physical parameterizations)
    enddo
  enddo
enddo
```

Here

- $\mathtt{ncols}(j)$ is the number of columns allocated to chunk j;
- `nchunks` is the number of chunks;
- `pcols` is the maximum number of columns allocated to any chunk (specified at compile time).

Thus, $\mathtt{pcols} \cdot \mathtt{nchunks} \geq \mathtt{nlon} \cdot \mathtt{nlat}$ for a tensor-product longitude-latitude grid, but there are no other assumptions about the composition of a chunk. In particular, the columns bundled in a given chunk may not be geographically contiguous.

The primary physics tuning options are as follows.

1) The first option is the compile-time parameter `pcols`. This determines the maximum number of columns assigned to a chunk. (Depending on the number of processors and number of columns, `ncols(j)` can be less than `pcols`.) Large `pcols` generates long inner loops, which improve vectorization. Small `pcols` decreases the size of the basic computational unit, which improves cache locality, and increases the number of chunks. While increasing the number of chunks can increase loop overhead, it also exposes additional OpenMP parallelism and may help in load balancing between

OpenMP threads. The specific value of `pcols` will also determine the memory alignment of elements in the chunked arrays, which has performance implications on most systems.

2) The second option is the load-time specification of the number of MPI processes and OpenMP threads for a given total number of processors. The number of OpenMP threads determines the number of chunks to create. The same number of chunks are (usually) assigned to all threads spawned by a given MPI process. Approximately the same number of columns are assigned to these chunks, so each thread is also assigned approximately the same number of columns.
3) The third option is the runtime assignment of columns to chunks. The time required to process a column is a function of geographical location and simulation time, and excellent static load balancing schemes are known. However, the best load balancing scheme is at odds with the domain decompositions utilized by the dycores (described below), thus requiring significant interprocessor communication to implement. Four different load balancing schemes are supported: balance load between chunks assigned to the same MPI process (no interprocess communication, load balancing only between local OpenMP threads), between two MPI processes (only pairwise interprocess communication required), between MPI processes assigned to the same SMP node (only intranode interprocess communication required), and the best load balancing scheme (requiring all processes to communicate with all other processes).
4) The fourth option is the runtime selection of communication protocol used to implement the interprocess communication required by the load balancing scheme. Possibilities include MPI collectives, 19 different MPI two-sided point-to-point implementations, an MPI one-sided point-to-point implementation, and a Co-Array Fortran one-sided point-to-point implementation.

C. Spectral Dynamics

The spectral dycores currently support only a one-dimensional decomposition of the computational grid, over latitude initially. OpenMP parallelism can be exploited to employ more processors in the physics than in the dynamics when there is insufficient MPI parallelism to achieve a performance goal. Each call of the spectral dynamics moves back and forth between the longitude-latitude-vertical grid point space and the spectral coefficient space. The dependencies in these transforms require changing the decomposition from one-dimensional over latitude to one-dimensional over longitude and back again. A halo update is also needed in the semi-Lagrangian advection scheme [30], and `gather` or `allgather` collective communication operators are needed to compute a number of diagnostic quantities. The primary tuning option in the spectral dynamics is the choice of communication protocols for these interprocess communications. The options are the same as for physics load balancing, described earlier, but different choices can be made for the physics and for the dynamics.

D. Finite Volume Dynamics

The finite volume dycore employs a two-dimensional block decomposition of the computational grid. There are two computational phases in this dycore. In one, the longitude and latitude dimensions are decomposed. In the other, the latitude and vertical dimensions are decomposed. Interprocess communication is needed for remapping between the two decompositions and for halo updates in the semi-Lagrangian advection scheme.

The primary dynamics tuning options for this dycore are as follows.

1,2) The first option is the runtime specification of the virtual two-dimensional processor grid that defines the blocks for the latitude-vertical domain decomposition. The second is the runtime specification of the virtual two-dimensional processor grid that defines the blocks for the longitude-latitude domain decomposition. While independent, the two choices should be considered together as they determine the communication overhead when remapping between decompositions. For example, it is generally best to use the same number of processors to decompose the latitude dimension in both decompositions. Note that a one-dimensional decomposition over latitude is best for small processor counts as it eliminates the need for a remap.

3) The third option is the load-time specification of the number of MPI processes and OpenMP threads for a given total number of processors. As OpenMP and MPI parallelism apply to the same loops in this dycore, the optimum is a function of the OpenMP overhead and the MPI communication overhead. The same settings are used in both the physics and the dynamics, and the performance impact in both will determine the optimum.

4) The fourth option is the communication protocol used for the remap and halo update communications. The finite volume dycore is built on top of a communication layer called `Pilgrim` [26], which supplies different options than those in the spectral dynamics and in the physics. Six options are supported: two MPI two-sided point-to-point implementations using either
   - temporary contiguous send and receive buffers, or
   - sending from and receiving into MPI derived types,

   and four MPI one-sided point-to-point implementations using one of
   - mpi_puts between temporary contiguous buffers on source and target,
   - direct mpi_puts of contiguous segments into a temporary contiguous window, with threading over the segments,
   - mpi_puts of mpi derived types into a temporary contiguous window, with threading over the target, or
   - mpi_puts between mpi derived types at source and target, with threading over the target

   (Some versions of `Pilgrim` also support SHMEM [17], and this has been introduced into the most recent version of CAM.)

IV. TUNING

The goal of the CAM benchmarking is to generate performance scaling curves as a function of processor count for one or more dycores and for one or more problem specifications. As the optimal tuning setings can vary with dycore, problem specification, and processor count, the large number of tuning options can make fair benchmarking an arduous task. Here we describe our approach to tuning CAM when benchmarking.

First, we use performance data from representative kernels or from CAM on a small number of processors to reduce the number of options that need to be examined when benchmarking. In particular, we identify appropriate compiler options, math library routines, communication layer and communication protocols, and chunk size (`pcols`).

Second, we organize the benchmarking in such a way as to prune the search tree when a given option has been shown to be uncompetitive. To eliminate start-up costs that would not be typical of production runs, we time the code when simulating one, two, and three simulation days, taking differences to estimate seconds per day of simulation for longer runs. When runtimes are small, we also verify these estimates with runs for 30 simulation days. We use the one day runs to identify a few good choices of the candidate tuning options and restrict the longer runs to these near-optimal choices. We also use scaling data to identify when options stop being competitive and can be eliminated from further experiments. For example, one-dimensional decompositions in the finite volume dynamics are more efficient for small processor counts than the two-dimensional decompositions. Initial experiments are used to determine the point of crossover, and two-dimensional decompositions are used only for larger processor counts. The search space pruning is applied to both the subset of options identified in the first stage of the tuning and to the other, as yet unexamined, options.

V. EXAMPLE RESULTS

In this section we describe results from recent benchmarking exercises, quantifying the importance of tuning. Data were collected on the following systems.

- Cray X1E cluster at ORNL: 256 4-processor symmetric multiprocessor (SMP) nodes. Each processor is a Multi-Streaming Processor (MSP) comprised of 8 32-stage vector units running at 1130 MHz, 128 64-bit wide, 64-element deep vector registers, and 4 scalar units running at 565 MHz.
- Cray XT3 at ORNL: 5294 single processor nodes and a custom interconnect. Each node is connected to a Cray Seastar router through Hypertransport, and the Seastars are interconnected in a 3D-torus topology. Each processor is a 2.4 GHz AMD Opteron.
- IBM p690 cluster at ORNL: 27 32-processor p690 SMP nodes and an HPS interconnect. Each node has two 2-link network adapters. Each processor is a 1.3 GHz POWER4.
- IBM p575 cluster at the National Energy Research Scientific Computing Center (NERSC): 122 8-processor p575 SMP nodes and an HPS interconnect. Each node has one 2-link network adapter. Each processor is a 1.9 GHz POWER5.
- SGI Altix 3700 at ORNL: non-uniform memory access (NUMA) shared memory system in which two 2-processor SMPs are connected to form a *C-brick*, with an SGI NUMAlink interconnect between the C-Bricks. Each processor is a 1.5 GHz Itanium2. The ORNL system has 256 processors.

The following experiments describe results for both the spectral Eulerian and the finite volume dycores. For the spectral dycore we used version 3.0.p1 of CAM, available from `http://www.ccsm.ucar.edu/models/atm-cam/`, and a problem with a horizontal grid of size 256×128 and 26 vertical levels. This is the same problem resolution and CAM dycore as used in the CCSM coupled climate model for the fourth IPCC (Intergovernmental Panel on Climate Change) assessments [22]. Because the spectral dycore supports only a one-dimensional decomposition over latitude, the number of MPI processes cannot be greater than 128 for this problem. However more than 128 processors can be used if OpenMP parallelism is exploited.

For the finite volume dycore we used version 3.1 of CAM, available from the same URL, and a problem with a horizontal grid of size 576×361 and 26 vertical levels. The finite volume dycore requires that at least three latitudes and three vertical levels be assigned to each MPI process. For a one-dimensional decomposition over latitude, this limits the number of MPI processes to 120. For a two-dimensional decomposition, the maximum number of MPI processes increases to 960.

In the rest of this section we describe some of the performance sensitivities of the different platforms for these problems and dycores. Note that we do not have the data necessary to show how bad performance can be if poor choices are made. The goal of the methodology described earlier is to identify poor choices quickly and eliminate them from further testing. Even some of the data described below could have been eliminated if we had been more careful when collecting data. We also do not show the sensitivities to compiler flags, communication protocol, or runtime environment variables (such as, for example, the memory and task affinity flags on the IBM systems), which are some of the most important optimizations, but are not unique to CAM.

Figure 1 shows the impact of `pcols` on the performance of the physics. The runtime for the physics for one simulation day was measured for a variety of chunk sizes. These runtimes were then normalized with respect to the runtime of the optimal `pcols` setting for each platform. The two graphs contain the same data, but use different axes limits and scalings. As expected, the vector system prefers large `pcols`, while the nonvector systems prefer smaller values. Performance on all of the nonvector systems is relatively insensitive to the exact `pcols` value as long as the extremes, both small and large, are avoided. The optimal value for each system is as follows: 8 for the Altix, 24 for the P690, 34 for the XT3, 80 for the P5-575, and 514 or 1026 for the X1. Note that performance is degraded on many of the systems when `pcols` is a power of two and large.

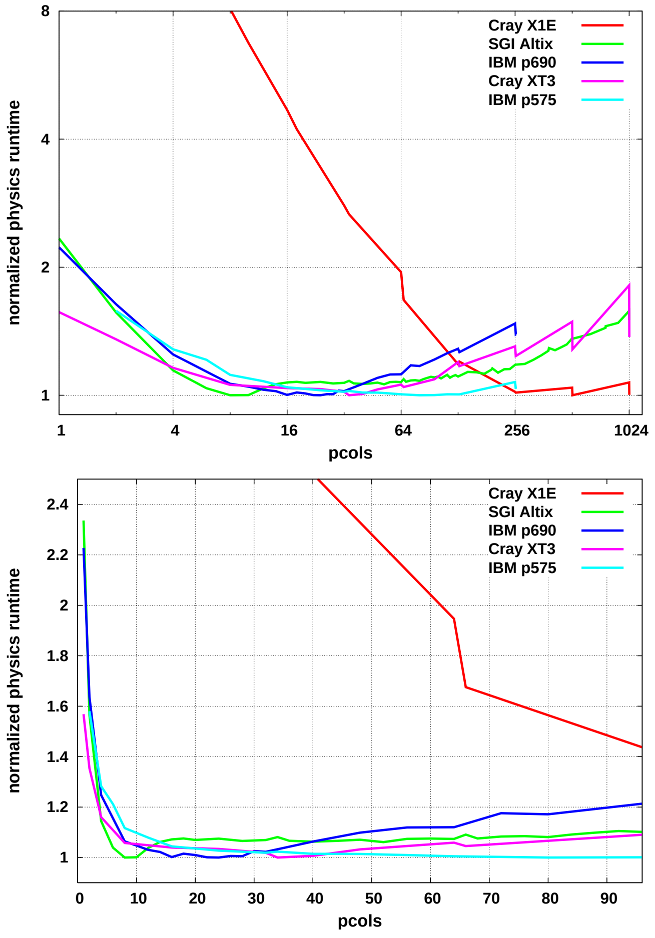


FIGURE 1: Chunk size experiments

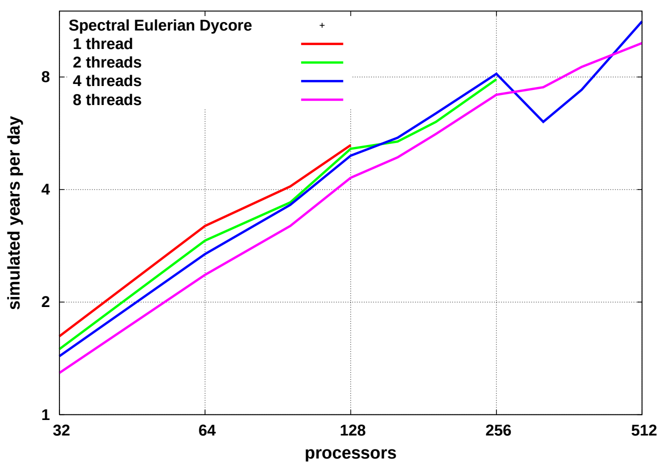


FIGURE 2: OpenMP experiments with spectral Eulerian dycore on IBM p690 cluster

Figures 2 and 3 show the impact of different numbers of OpenMP threads per process as a function of the total number of processors. The total runtime for one simulation day was measured on the IBM p690 cluster for both the spectral Eulerian and finite volume dycores. For the finite volume dycore, results are shown for both a one-dimensional decomposition and a two-dimensional decomposition defined by a virtual processor grid of size $(P/4) \times 4$, where P is the number of MPI processes. Performance is already optimized with respect to communication protocol, chunk size, and load balancing in these experiments. The runtime is used to calculate the computation rate in simulation years per wallclock day. For these experiments the primary utility of OpenMP parallelism is to increase the total number of processors that can be used. For experiments with the spectral Eulerian dycore, utilizing fewer threads is generally faster for a given total number of processors. Exceptions occur when a particular MPI process count leads to an uncorrectable load imbalance. A similar result holds for the finite volume dycore and the two-dimensional decomposition. For the finite volume dycore and the one-dimensional decomposition, there is a crossover point beyond which it is faster to not increase the number of MPI processes, utilizing more OpenMP threads instead. Note that the one-dimensional decomposition achieves a higher computational rate than the two-dimensional decomposition, up to 672 processors and for this problem size.

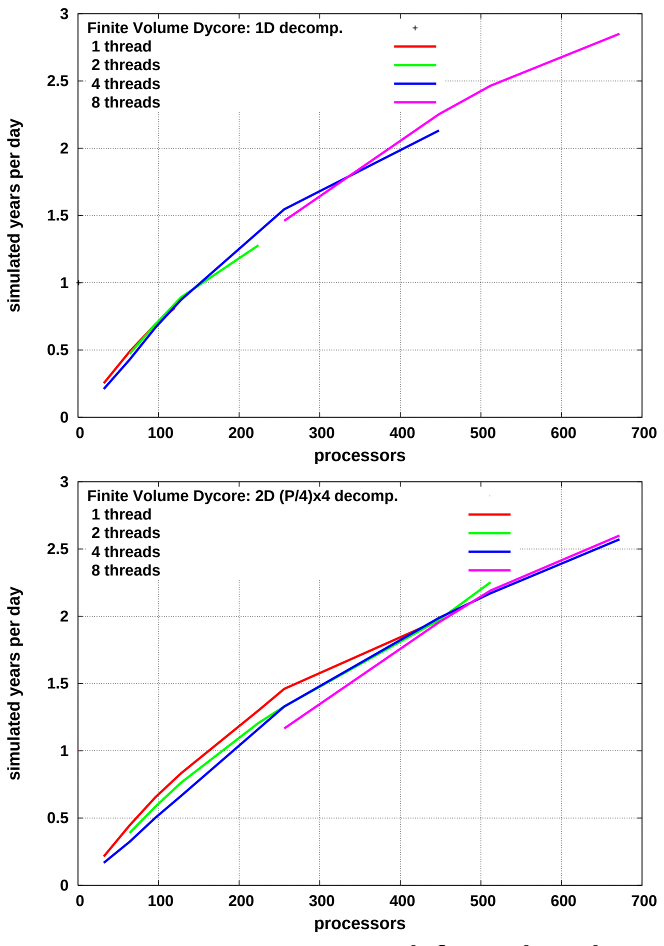


FIGURE 3: OpenMP experiments with finite volume dycore on IBM p690 cluster

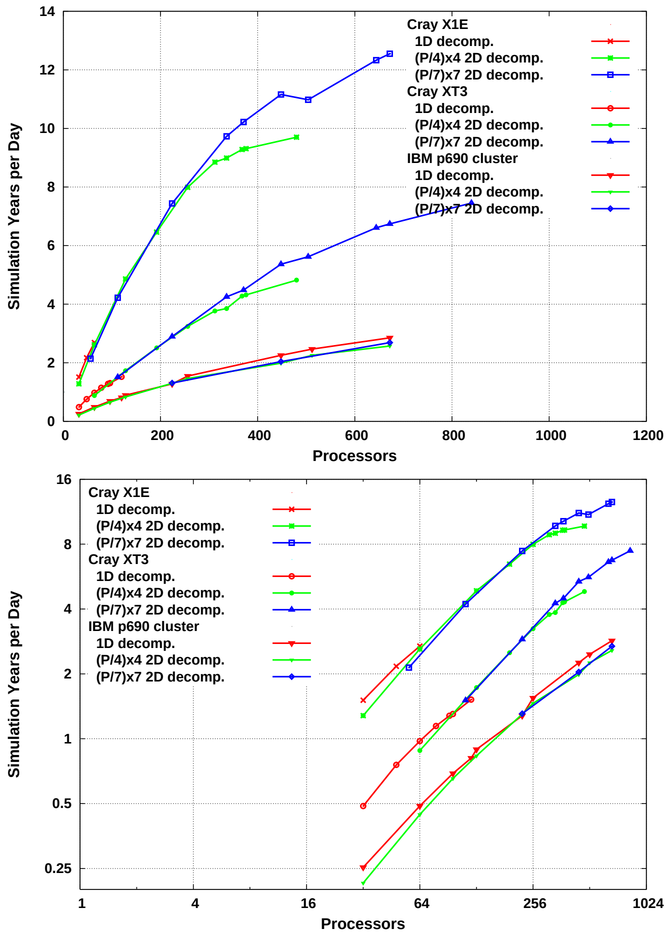


FIGURE 4: Domain decomposition experiments

Figure 4 compares the performance of the one-dimensional domain decomposition with two-dimensional domain decompositions defined by virtual processor grids of size $(P/4) \times 4$ and $(P/7) \times 7$, respectively, for the finite volume dycore. Performance is already optimized with respect to communication protocol, chunk size, load balancing, and number of OpenMP threads in these experiments. Results are presented for three systems: the Cray X1E, the Cray XT3, and the IBM p690 cluster. The computational rate for one simulation day is plotted versus the number of processors. The two graphs contain the same data, but use different axes scalings. For the two Cray systems the two-dimensional decompositions extend scalability and improve performance compared to the one-dimensional decomposition when the one-dimensonsal decomposition employs more than approximately 70 MPI processes to decompose the latitude dimension. This latitude process limit also determines when the $(P/7)\times 7$ two-dimensional domain decomposition begins to outperform the $(P/4)\times 4$ two-dimensional domain decomposition. For smaller latitude process counts, the performance of the two-dimensional decompositions are approximately the same. Experiments on the Cray systems did not use OpenMP parallelism. In contrast, OpenMP parallelism was employed in the IBM p690 experiments. OpenMP parallelism allows the number of MPI processes to be kept no larger than 84 in the p690 cluster experiments, and the one-dimensional decomposition results are always superior to the two-dimensional decomposition results. Again, the two two-dimensional decompositions demonstrate approximately the same performance.

Figure 5 compares the performance of the different load balancing schemes on the Cray XT3 and on the IBM p690 cluster for the finite volume dycore. Performance is already optimized with respect to communication protocol, chunk size, domain decomposition, and number of OpenMP threads in these experiments. The graphs show the ratio of the runtime for a given load balancing scheme to that of the minimum runtime over all load balancing schemes for a given processor count. On the p690 cluster, full load balancing is always best, but no load balancing is at most 8 percent slower. On the Cray XT3, full load balancing is usually best, but pairwise exchange load balancing is never much worse. Here, no load balancing is as much as 12 percent slower.

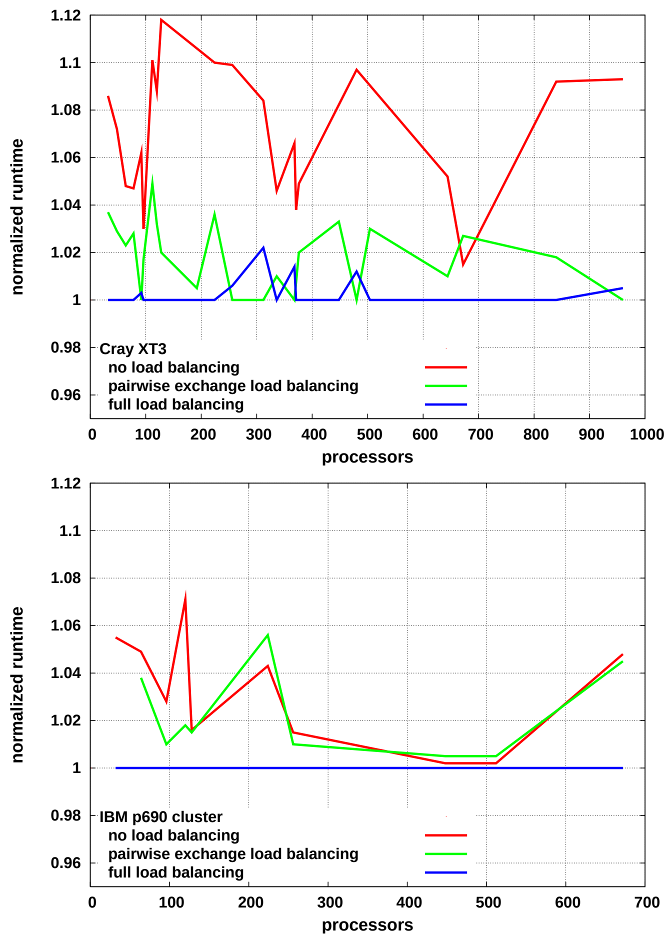
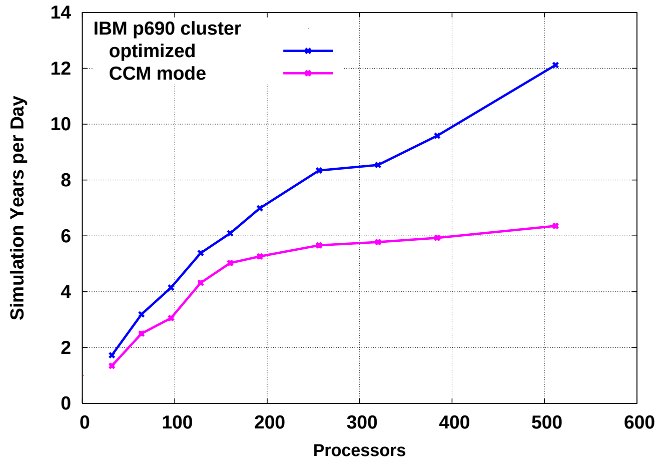


FIGURE 5: Load balancing experiments

We did not include a load balancing comparison for the Cray X1E as full load balancing is always significantly better and we did not collect the required data. While full load balancing is typically best on these platforms, this is not true on all platforms. In particular, systems for which interprocessor communication bandwidth is low compared to processor speed will likely not find full load balancing useful.

The final figure and tables are attempts to capture the impact of all of the performance tuning options. Such a comparison is difficult in that it requires defining the "default" options.

It is easy to make CAM run poorly on any given system, for example, by setting `pcols` inappropriately. Figure 6 compares optimal CAM performance for the spectral Eulerian dycore with the performance when running CAM in the same way as CCM, CAM's predecessor, on the IBM p690 cluster and on the Cray X1E. In *CCM mode* each latitude line (subset of vertical columns with a common latitude index) is a chunk (`pcols` $\equiv 258$), OpenMP is not enabled, and no load balancing is used.



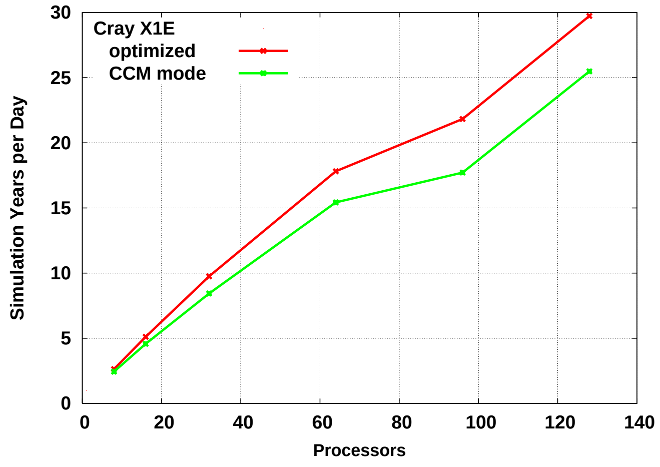


FIGURE 6: Performance comparison between optimal and default settings for spectral Eulerian dycore.

The optimal settings for the IBM p690 cluster are described in Table 1. The load balancing and chunk size optimizations result in significant performance improvements. Both load balancing and increased serial performance due to the smaller chunk size contribute to the performance enhancement for small processor counts. For large processor counts, the small chunk sizes (16, 24, 32 vertical columns) increase the amount of OpenMP parallelism available compared to 256 column chunks, improving scalability compared to runs with the CCM settings. Note that full load balancing is not optimal in many instances in these experiments.

The optimal settings for the Cray X1E are described in Table 2. As performance for the vector length in CCM mode is close to optimal, the differences in performance are primarily due to load balancing. OpenMP parallelism was not exploited in these runs. Partly this is because OpenMP has not been tested extensively in the current port of CAM to the X1E.

Proc.	MPI processes	threads per process	`pcols`	load balance	improv. vs. CCM alg.
32	32	1	16	full	28%
64	64	1	16	pairwise	28%
96	96	1	24	full	36%
128	128	1	16	pairwise	25%
160	40	4	24	full	21%
192	48	4	24	pairwise	33%
256	128	2	16	full	47%
320	40	8	32	none	48%
384	48	8	24	pairwise	62%
512	128	4	16	pairwise	91%

TABLE I
OPTIMAL PERFORMANCE SETTINGS FOR IBM P690 CLUSTER FOR SPECTRAL EULERIAN DYCORE

Proc.	MPI processes	threads per process	`pcols`	load balance	improv. vs. CCM alg.
8	8	1	1026	full	8%
16	16	1	1026	full	19%
32	32	1	1026	full	16%
64	64	1	514	full	15%
96	96	1	514	full	23%
128	128	1	258	full	17%

TABLE II
OPTIMAL PERFORMANCE SETTINGS FOR CRAY X1E FOR SPECTRAL EULERIAN DYCORE

However, we do not expect OpenMP parallelism to increase scalability much beyond 128 processors for this problem size. For example, using 256 processors, say 128 MPI processes and 2 OpenMP threads per process, will halve the number of columns per chunk to 128. This decreases the vector length of many loops in the physics to 128, which is half the vector length of the X1E processor. Thus, while we would be using twice as many processors, the performance of each processor would be approximately halved in the physics.

There is no older version of CAM with the finite volume dycore that can be used to define an experiment similar to that described above, nor do we have sufficient performance data for any default settings. Instead we simply report optimal settings.

The optimal settings for the IBM p690 cluster for the finite volume dycore are described in Table 3. In only one of the examined cases did a two-dimensional decomposition (32x7 with one thread) outperform a one-dimensional decomposition (112x1 with two threads), and then only slightly. Note that the optimal value of `pcols` increases with the number of OpenMP threads. The reason for this is not clear, but the performance impact was significant in some cases. Also note that direct interprocessor communication between MPI derived types was faster than copying into/out of temporary buffers.

The optimal settings for the Cray X1E for the finite volume dycore are described in Table 4. For each experiment a `pcols` value was determined that would maximize the number of columns per chunk subject to the restriction that `pcols` ≤ 1026 while also minimizing memory requirements. (`pcols` larger than 1026 sometimes causes memory problems.) A subset of the values was then selected that provided reasonable coverage, simply to limit the number of executables gener-

Proc.	processor grid	threads per process	`pcols`	load balance	derived types
32	32x1	1	16	full	yes
64	64x1	1	16	full	yes
96	48x1	2	16	full	yes
128	64x1	2	16	full	yes
224	32x7	1	24	full	yes
256	64x1	4	24	full	yes
448	56x1	8	32	full	yes
512	64x1	8	32	full	yes
672	84x1	8	32	full	yes

TABLE III

OPTIMAL PERFORMANCE SETTINGS FOR IBM P690 CLUSTER FOR FINITE VOLUME DYCORE

Proc.	processor grid	threads per process	`pcols`	load balance	derived types
32	32x1	1	870	full	no
48	48x1	1	870	full	no
64	64x1	1	870	full	no
96	24x4	1	870	full	no
128	32x4	1	870	full	no
192	48x4	1	990	full	no
256	64x4	1	870	full	no
336	48x7	1	630	full	no
448	64x7	1	570	full	no
672	96x7	1	330	full	no

TABLE IV

OPTIMAL PERFORMANCE SETTINGS FOR CRAY X1E FOR FINITE VOLUME DYCORE

ated for the benchmarking. As mentioned earlier, the one-dimensional decomposition is superior until the number of processes applied to the latitude dimension exceeds approximately 70. A similar rule holds when comparing the two-dimensional decompositions. For example, for 336 processors the 48×7 virtual processor grid delivers better performance than the 84×4 processor grid. On the X1E sending from or receiving into MPI derived data types degrades performance.

The optimal settings for the Cray XT3 for the finite volume dycore are described in Table 5. A slightly larger value of `pcols` than indicated earlier was found to perform best, though any choice between 20 and 40 is likely to work well. Performance is insensitive to the choice of communicating directly between MPI derived data types or not. While full load balancing was always optimal, the pairwise exchange load balancing option performed equally well, in agreement with the earlier discussion.

Benchmark experiments are not yet complete on the p575 cluster at NERSC, and results comparable to those from the other systems were not available in time for inclusion in this paper. Further Altix benchmarks will be run on the newer Altix systems at NASA-Ames rather than on the ORNL system.

VI. CONCLUSIONS

In this paper we described briefly some of the tuning options that make CAM a useful benchmark for evaluating early systems. We also described a methodology that makes it possible to tune CAM quickly when benchmarking. Even with this methodology, tuning CAM for benchmarking is expensive,

Proc.	processor grid	threads per process	`pcols`	load balance	derived types
32	32x1	1	40	full	yes
48	48x1	1	40	full	no
64	64x1	1	40	full	yes
96	24x4	1	40	full	yes
128	32x4	1	40	full	yes
192	48x4	1	40	full	yes
256	64x4	1	40	full	yes
336	48x7	1	40	full	no
448	64x7	1	40	full	yes
672	96x7	1	40	full	yes

TABLE V

OPTIMAL PERFORMANCE SETTINGS FOR CRAY XT3 FOR FINITE VOLUME DYCORE

but we feel that the improved fairness in the performance data justifies the additional cost.

On the systems utilized in this paper, OpenMP and the two-dimensional domain decompositions were primarily useful for extending the number of processors that could be employed. In contrast, load balancing and chunk size were important optimizations for any processor count. While not mentioned in the paper, communication protocol did not play an important role in the sense that the obvious choices (using MPI collectives for collective operations and MPI_SENDRECV for point-to-point operations) performed well. The one exception was that sending from and receving into MPI derived data types performed poorly on the Cray X1E. The insensitivity to the choice of communciation protocol is due to the high performance of the interconnects and messaging libraries available on the target systems. However, communication protocol options have been very important performance enhancers in the past, and we expect them to continue to be important performance portability options in the future.

VII. ACKNOWLEDGEMENTS

This research used resources (Cray X1E, Cray XT3, IBM p690 cluster, and SGI Altix) of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. It also used resources (IBM p575 cluster) of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. We also gratefully acknowledge our collaborators, too numerous to mention here, in the performance engineering of CAM.

REFERENCES

[1] M. B. BLACKMON, B. BOVILLE, F. BRYAN, R. DICKINSON, P. GENT, J. KIEHL, R. MORITZ, D. RANDALL, J. SHUKLA, S. SOLOMON, G. BONAN, S. DONEY, I. FUNG, J. HACK, E. HUNKE, AND J. HURREL, *The Community Climate System Model*, BAMS, 82 (2001), pp. 2357–2376.

[2] W. D. COLLINS, P. J. RASCH, B. A. BOVILLE, J. J. HACK, J. R. MCCAA, D. L. WILLIAMSON, B. P. BRIEGLEB, C. M. BITZ, S.-J. LIN, AND M. ZHANG, *The Formulation and Atmospheric Simulation of the Community Atmosphere Model: CAM3*, Journal of Climate, to appear (2006).

[3] W. D. Collins, P. J. Rasch, and et. al., *Description of the NCAR Community Atmosphere Model (CAM 3.0)*, NCAR Tech Note NCAR/TN-464+STR, National Center for Atmospheric Research, Boulder, CO 80307, 2004.

[4] Community Climate System Model. http://www.ccsm.ucar.edu/.

[5] L. Dagum and R. Menon, *OpenMP: : An industry-standard API for shared-memory programming*, IEEE Computational Science & Engineering, 5 (1998), pp. 46–55.

[6] J. B. Drake, R. E. Flanery, I. T. Foster, J. J. Hack, J. G. Michalakes, R. L. Stevens, D. W. Walker, D. L. Williamson, and P. H. Worley, *The message-passing version of the parallel community climate model*, in Parallel Supercomputing in Atmospheric Science: Proceedings of the Fifth ECMWF Workshop on Use of Parallel Processors in Meteorology, G.-R. Hoffman and T. Kauranne, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1993, pp. 500–513.

[7] 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.

[8] J. B. Drake, S. Hammond, R. James, and P. H. Worley, *Performance tuning and evaluation of a parallel community climate model*, in Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC99), Nov. 13-19, 1999, IEEE Computer Society Press, Los Alamitos, CA, 1999.

[9] T. H. Dunigan, Jr., *Hypercube performance*, in Hypercube Multiprocessors 1987, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 178–192.

[10] ———, *Performance of a second generation hypercube*, Tech. Rep. ORNL/TM-10881, Oak Ridge National Laboratory, Oak Ridge, TN, September 1988.

[11] ———, *Communication performance of the Intel Touchstone DELTA Mesh*, Tech. Rep. ORNL/TM-11983, Oak Ridge National Laboratory, Oak Ridge, TN, December 1991.

[12] ———, *Performance of the Intel iPSC/860 and the Ncube 6400 hypercubes*, Parallel Computing, 17 (1991), pp. 1285–1302.

[13] ———, *Kendall square multiprocessor: Early experience and performance*, Tech. Rep. ORNL/TM-12065, Oak Ridge National Laboratory, Oak Ridge, TN, March 1992.

[14] T. H. Dunigan, Jr., M. R. Fahey, J. B. White III, and P. H. Worley, *Early Evaluation of the Cray X1*, in Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC03), Nov. 15-21, 2003, IEEE Computer Society Press, Los Alamitos, CA, 2003.

[15] T. H. Dunigan, Jr., J. S. Vetter, J. B. White III, and P. H. Worley, *Performance evaluation of the Cray X1 distributed shared-memory architecture*, IEEE Micro, 25(1) (January/February 2005), pp. 30–40.

[16] M. R. Fahey, S. Alam, T. H. Dunigan, Jr., J. S. Vetter, and P. H. Worley, *Early Evaluation of the Cray XD1*, in Proceedings of the 47th Cray User Group Conference, May 16-19, 2005, R. Winget and K. Winget, ed., Eagen, MN, 2004, Cray User Group, Inc.

[17] K. Feind, *Shared Memory Access (SHMEM) Routines*, in CUG 1995 Spring Proceedings, R. Winget and K. Winget, ed., Eagen, MN, 1995, Cray User Group, Inc., pp. 303–308.

[18] W. Gropp, M. Snir, B. Nitzberg, and E. Lusk, *MPI: The Complete Reference*, MIT Press, Boston, 1998. second edition.

[19] 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.

[20] M. T. Heath, G. A. Geist, and J. B. Drake, *Early experience with the Intel iPSC/860 at Oak Ridge National Laboratory*, Tech. Rep. ORNL/TM-11655, Oak Ridge National Laboratory, Oak Ridge, TN, September 1990.

[21] R. Hockney and M. B. (Eds.), *Public International Benchmarks for Parallel Computers, Parkbench Committee Report-1*, Scientific Programming, 3 (1994), pp. 101–146.

[22] Intergovernmental Panel on Climate Change. http://www.ipcc.ch/.

[23] 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.

[24] S.-J. Lin, *A 'vertically Lagrangian' finite-volume dynamical core for global models*, Mon. Wea. Rev., 132 (2004), pp. 2293–2307.

[25] A. Mirin and W. B. Sawyer, *A scalable implemenation of a finite-volume dynamical core in the Community Atmosphere Model*, International Journal of High Performance Computing Applications, 19 (2005), pp. 203–212.

[26] W. Putman, S. J. Lin, and B. Shen, *Cross-platform performance of a portable communication module and the NASA finite volume general circulation model*, International Journal of High Performance Computing Applications, 19 (2005), pp. 213–224.

[27] M. Rancic, R. J. Purser, and F. Mesinger, *A global shallow-water model using an expanded spherical cube: gnomic versus conformal coordinates*, Q. J. R. Met. Soc., 122: 959-982 (1996).

[28] J. S. Vetter, S. R. Alam, T. H. Dunigan, Jr., M. R. Fahey, P. C. Roth, and P. H. Worley, *Early Evaluation of the Cray XT3 at ORNL*, in Proceedings of the 47th Cray User Group Conference, May 16-19, 2005, R. Winget and K. Winget, ed., Eagen, MN, 2005, Cray User Group, Inc.

[29] D. L. Williamson and J. G. Olson, *Climate simulations with a semi-lagrangian version of the NCAR Community Climate Model*, Mon. Wea. Rev., 122 (1994), pp. 1594–1610.

[30] D. L. Williamson and P. J. Rasch, *Two-dimensional semi-Lagrangian transport with shape-preserving interpolation*, Mon. Wea. Rev., 117 (1989), pp. 102–129.

[31] P. H. Worley, *MPI performance evaluation and characterization using a compact application benchmark code*, in Proceedings of the Second MPI Developers Conference and Users' Meeting, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 170–177.

[32] ———, *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.

[33] P. H. Worley and J. B. Drake, *Performance portability in the physical parameterizations of the Community Atmosphere Model*, International Journal of High Performance Computing Applications, 19 (2005), pp. 187–202.

[34] P. H. Worley, T. H. Dunigan, Jr., M. R. Fahey, J. B. White III, and A. S. Bland, *Early evaluation of the IBM p690*, in Proceedings of the IEEE/ACM SC2002 Conference, Nov. 16-22, 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002.

[35] P. H. Worley and I. T. Foster, *Parallel Spectral Transform Shallow Water Model: a runtime–tunable parallel benchmark code*, in Proc. Scalable High Performance Computing Conf., J. J. Dongarra and D. W. Walker, eds., IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 207–214.

[36] 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.