

Performance Portability in the Physical Parameterizations of the Community Atmospheric Model *

P. H. Worley [†]

J. B. Drake [‡]

Oak Ridge National Laboratory

Abstract

Community models for global climate research such as the Community Atmospheric Model must perform well on a variety of computing systems. Supporting diverse research interests, these computationally demanding models must be efficient for a range of problem sizes and processor counts. In this paper we describe the data structures and associated infrastructure developed for the physical parameterizations that allow the Community Atmospheric Model (CAM3) to be tuned for vector or nonvector systems, to provide load balancing while minimizing communication overhead, and to exploit the optimal mix of distributed MPI processes and shared OpenMP threads.

1 Introduction

Atmospheric global circulation models are 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 [4]. 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 timestep advancing the model simulation time. The number and order of these transitions depend on the numerical algorithm.

In the Spring of 2000, a group of atmospheric scientists and computer scientists met at the National Center for Atmospheric Research (NCAR) to outline the design of the Community Atmospheric Model (CAM) [4], the proposed successor to the Community Climate Model (CCM) [13, 15]. One of the decisions made at this meeting was that CAM would support multiple *dynamical cores* (*dycores*) using one set of physical parameterizations. Three dycores were targeted, the spectral Eulerian solver used in CCM, a spectral semi-Lagrangian solver [24], and a finite volume semi-Lagrangian solver [17]. While all three dycores use tensor product longitude-latitude grids, they do not use the same grids, have the same placement of variables on the grid, or support the same domain decompositions in their parallel implementations. Moreover, the expectation was that dycores would be supported in the future that do not use longitude-latitude grids. An explicit interface was defined between the dynamics and the physics, and the physics data structures and parallelization strategies were no longer required to be identical with those used in the dynamics. Instead, a dynamics-physics coupler (`dp_coupling`) would be used to move data between data structures representing the dynamics state and the physics state. In previous work [8, 10, 11, 9], significant effort had been expended to determine data structures and domain decompositions that work well with both the dynamics and the

*This research was sponsored by the Climate Change Research Division of the Office of Biological and Environmental Research, 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. 5600, Oak Ridge, TN 37831-6016 (worleyph@ornl.gov)

[‡]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5600, Oak Ridge, TN 37831-6016 (drakejb@ornl.gov)

physics, in order to minimize memory requirements, to avoid the cost of buffer copies, and/or to avoid the cost of interprocess communication when execution moves between the dynamics and the physics during each timestep of the algorithm. With the decision to decouple physics and dynamics data structures, a global design was no longer necessarily advantageous. Developers were free to optimize performance within the physics, and to investigate a number of different approaches to minimizing the overheads introduced by the decoupling of the physics and dynamics data structures.

This paper begins with a description of the design and implementation of the physics data structures and associated parallel algorithms used in CAM. It then describes the performance that we have observed. One of the important features of the design has been support for compile- and runtime tuning of the physics data structures, improving the ability to port and to optimize the model on different architectures and for different problem sizes and processor counts (i.e., improving the *performance portability* of the code).

2 Design

We refer to all grid points in a three-dimensional grid with a given horizontal location, thus differing only in the vertical coordinate, as a *vertical column*, or just *column*. The current physical parameterizations in CAM are based on vertical columns, with dependencies in the vertical direction only, and computations are therefore independent between columns.

The data structures used in the original CCM physics are identical to the data structures used in the spectral Eulerian dynamics, and target efficient performance on vector machines [16]. The CCM computational domain is a tensor product (longitude x latitude x vertical) grid covering the sphere. To exploit the vectorization possibilities, the CCM data structures define the domain as (`nlon`, `nver`, `nlat`). i.e. with the vertical coordinate as the second index. The basic loop structure in the physics follows the index ordering:

```
do j=1,nlat
  do k=1,nver
    do i=1,nlon
      (physical parameterizations)
    enddo
  enddo
enddo
```

Here

- `nlon` is the number of longitudes;
- `nver` is the number of vertical levels;
- `nlat` is the number of latitudes.
- `nlon = nlon + 1 + 2 · nexpt`, where `nexpt` defines the size of a *halo* used in the semi-Lagrangian advection algorithm [25]. Data is copied into the halo to eliminate boundary condition logic in the advection algorithm, improving performance on vector systems. `nexpt` is typically one, supporting cubic interpolation.

Because computation in the physics is independent between vertical columns, the inner loop over longitude is vectorizable. Coarser grain parallelism is exploited in the outer loop over latitude, via either MPI [12] or OpenMP [6].

To exploit vectorization, it is important to bundle the computation of multiple columns. Vectorization can also be important in modern cache-based processor architectures to expose fine-grain parallelism for long-instruction-word architectures, improve effectiveness of memory prefetching, and enable latency hiding for expensive intrinsics such as `exp` or `sqrt`.) However, working with too many columns simultaneously can cause cache misses. To support performance on both vector and nonvector systems, we generalized the physics data structure as a collection of (arbitrary) subsets of columns that we call *chunks*. Grid points in a chunk are referenced by (`local column index`, `vertical index`). A “chunked” array is declared as (`pcols`, `nver`, `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

- `ncols(j)` is the number of columns allocated to chunk j ;
- `nver` is the number of vertical levels;
- `nchunks` is the number of chunks;
- `pcols` is the maximum number of columns allocated to any chunk (specified at compile time).

Thus, $pcols \cdot nchunks \geq nlon \cdot nlat$ 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 inner loop is again vectorizable, and the outer loop is the MPI or OpenMP parallel direction. CAM is a Fortran code, so the inner loop also runs sequentially over contiguous memory locations. As the chunk size (`pcols` and `ncols`) decreases, the cache locality increases and the parallelism exploitable at the outer loop level increases. In contrast, as the chunk size increases, the vectorization opportunities increase. Note that the original CCM physics data structure is also a chunk data structure where there are `nlat` chunks each with `nlon` columns and `pcols` \equiv `nlon`. Thus it is possible to reproduce the original CCM parallelization configuration using the new data structure.

This design was motivated by our previous experiences in the development of PCCM [8, 9], a research version of CCM that implemented a two-dimensional domain decomposition of the dynamics and physics in order to improve scalability for large processor counts, by extensive experimentation [26, 28, 27] using the CCM column radiation model (CRM) [3], and by numerous reports in the literature on the utility of cache blocking and other cache efficient programming techniques, e.g. [7, 22, 23]. We were also motivated by the successful introduction of a performance tunable data structure in the Integrated Forecast System (IFS) [1], developed at the European Centre for Medium-Range Weather Forecasting. IFS is an atmospheric model with a spectral dycore similar to that used in CAM. It is used for weather forecasting, rather than climate modeling, and with much higher resolution grids than are typically used with CAM. IFS performance is dominated by the cost of the dynamics, in contrast to CAM in which the physics tends to be the dominant cost at the resolutions of interest. IFS is also a more uniform design, without as clear a separation between the dynamics and physics. However, the basic data structure in IFS, *NPROMA blocks*, is very similar to chunks. In the IFS, blocks are not arbitrary collections of columns. Rather, lines of constant latitude in the two-dimensional longitude-latitude grid are concatenated into a one-dimensional data structure, and blocks are defined as columns whose horizontal coordinates are contiguous in the one-dimensional data structure. This has proven to be a good choice for the IFS dynamics. The size of the block, *NPROMA*, is then used to maximize vector lengths or improve cache locality.

This paper deals strictly with the design of the physics data structures. The physics/dynamics split allowed us to develop these new data structures in relative isolation from other changes in the model. However, the intent was always to improve the performance of both the physics and dynamics in CAM. Significant progress has been made in performance optimization and scalability of the finite volume dycore [18]. We plan on adopting the best ideas from the designs used in IFS, PCCM, and the finite volume dycore to improve the performance of the spectral dycores. As will be shown, decoupling the performance engineering of the physics and dynamics has not hurt the overall performance of CAM. In our experience, it has also made the development process less costly and easier to manage and the code easier to maintain than earlier “monolithic” efforts such as PCCM.

3 Implementation

The chunk generalization of the physics data structures has many performance implications. To motivate the discussion, we first outline the logic used to define chunks in CAM.

Most high performance computing (HPC) systems can usefully be characterized as a cluster of symmetric multiprocessor (SMP) *nodes* with an interconnect linking the nodes. This model is inclusive in that each node could be a single processor system or the system could consist of a single SMP node. An important feature of this characterization is that it views the processors in a multiprocessor system as being partitioned into subsets where communication within a subset of processors is less expensive than communication between processors in different subsets. Thus an essential property of the memory hierarchy is captured.

When partitioning the vertical columns into chunks, we treat the actual parallel system as a virtual cluster of nodes and attempt to assign a fixed number of chunks (`chunks_per_thread`) to each thread of execution. For each node in the cluster, we first determine the number of columns assigned to the node in the domain decomposition used in the dynamics (`num_columns(node)`). We determine the number of OpenMP threads associated with each MPI process assigned to the node, and use this to calculate the total number of threads associated with the node (`num_threads(node)`). (If OpenMP is not enabled, then each MPI process has by definition one thread.) The number of columns in the node is divided by `pcols` to determine the minimum number of chunks for this node:

```
num_chunks(node) = ceiling(float(num_columns(node))/pcols) .
```

This number is increased if it is too small to assign the required number of chunks to each thread. It may also be increased so that the same number of chunks can be assigned to each thread. Finally, we require that each chunk have at least one column, which may require that `num_chunks(node)` be decreased. From this information, we determine the maximum number of columns to assign to each chunk in the current node:

```
maxcol_chunk(node) = ceiling(float(num_columns(node))/num_chunks(node))
```

Columns are assigned to chunks so that all chunks in a node have approximately the same computational cost. This assignment is described in the next section. After the chunks are defined, they are assigned to specific processes in such a way that all threads in a node have the same number of chunks (if possible) and the MPI communication cost when remapping between the dynamics and physics domain decompositions is (approximately) minimized. This last goal is achieved by assigning each chunk to a process that was assigned many of the same vertical columns as part of the dynamics domain decomposition.

The role of the virtual cluster of nodes is to control the amount of MPI communication required in `dp_coupling`. Five options are provided currently, controlled by the runtime parameter `phys_loadbalance`.

- 1) Specify that each node has exactly one process. Then the chunks assigned to a process are made up of the same columns assigned to the process in the dynamics decomposition, and no interprocess communication is required.
- 0) Same as `phys_loadbalance = -1` with respect to the virtual architecture. This option differs from `phys_loadbalance = -1` in the way that columns are assigned to chunks, as described later.
- 1) Specify that virtual nodes correspond to actual nodes. If running on a cluster of single processor nodes, this is identical to option 0. If running on a single shared memory system, this is identical to option 2. If running on a cluster of shared memory systems, all interprocess communication in `dp_coupling` is restricted to communication between processes assigned to the same node.
- 2) Specify that there is only one node and that all processes are assigned to it. Then the load balancing algorithm described in the next section is free to assign any column to any chunk, typically requiring communication between all processes.
- 3) Specify that each node is assigned two processes, defining a partition of the processes into pairs. The chunks assigned to each process pair are made up of the columns assigned to these processes in the dynamics decomposition. Thus, all MPI communication in `dp_coupling` is limited to processes in the same pair. There is a special load balancing algorithm designed for this case, also described in the next section.

Note that each of these options assumes that all the vertical levels for a given horizontal location are assigned to the same node in the dynamics decomposition. For the current dycores the vertical coordinate is not decomposed in the dynamics when calling `dp_coupling`, and all five of the options work. However, this is not a requirement. If the parallel algorithm in a future dycore decomposes over the vertical, only option 2 will be guaranteed to work.

4 Load Balancing

The approach we took for load balancing is to assign vertical columns to chunks so that all chunks (in a node) have approximately the same computational cost. If this is successful, then equidistributing the chunks to threads (within a node) is all that is required in order to balance the load (within a node). The algorithm used to construct load balanced chunks is a generalization of the load balancing algorithm used in PCCM.

Most physical processes are calculated every timestep. However, short-wave radiation, long-wave radiation, and updates to the longwave absorptivity and emissivity of water vapor can be calculated less often. The frequencies can be specified at runtime, but the default is to treat both the short- and long-wave radiation during the same timestep once each simulation hour. We refer to this as a *radiation* timestep. Thus, if the timestep is 10 minutes, then a radiation timestep occurs every sixth timestep. For the absorptivity and emissivity update, the default is to calculate it during a radiation timestep once every twelve hours. We refer to this as an *absems* timestep.

Radiation and absems timesteps are much more expensive than the other *standard* timesteps. And the radiation calculations, in both radiation and absems timesteps, are much more expensive for columns at horizontal locations for which it is day during a given timestep. The magnitude of this difference is a function of the processor architecture, but it is a major contributor to load imbalance in CAM simulations. This will be quantified in Section 8.

To balance the load in the radiation calculation, we partition the columns into sets of column pairs where one column will be at a night location when the other is at a day location. In particular, given a column with longitude-latitude indices (`lon`, `lat`), the column whose longitude coordinate differs by 180 degrees and whose latitude is in the opposite hemisphere but the same distance from the equator, will have the required properties. For the current dycores, this *partner* column has the longitude-latitude indices

$$(\text{mod}((\text{lon} - 1) + (\text{nlon}/2), \text{nlon}) + 1, \text{nlat} + 1 - \text{lat}).$$

There are also other sources of load imbalance in the physics that are characterized by spatial locality. These are addressed by not assigning neighboring columns to the same chunk. In the current implementation a *wrap map* is applied to columns (and their partners), assigning consecutive columns to consecutive chunks, not to the same chunk. This also has the effect of assigning the same number of columns to each chunk, which is the other requirement for constructing load balanced chunks once the day-night load imbalance issue has been addressed. The efficacy of this approach depends on the column ordering defined in the dynamics, the problem size, and the number of chunks.

The load balancing algorithm described above is what is used with `phys_loadbalance` option 2, where all processes are assigned to the same virtual node. However, the partner for a given column is not guaranteed to be in the same node for the other options, and thus would require communication between different virtual nodes when moving from the dynamics domain decomposition, which is not permitted. The second choice is to define the partner as the column whose longitude is 180 degrees different but at the same latitude. This choice is nonoptimal away from the equator because the columns in a pair will both be day locations (night locations) simultaneously at some time during the model day during the summer (winter). If this second choice is not available as a partner, column pairing is not attempted and only the wrap map is used for load balancing.

As mentioned earlier, a special load balancing algorithm is used when communication is limited to process pairs (`phys_loadbalance` option 3). First, process pairs are determined that maximize the likelihood that a column and its partner are assigned to the same process pair in the dynamics decomposition. The standard load balancing algorithm is then applied to each process pair, approximating the load balancing scheme specified by `phys_loadbalance` option 2, but requiring only pairwise MPI communications. However, there

is no guarantee that the two processes in a pair are actually assigned to the same physical SMP node, which would offset some of the performance gains from requiring only pairwise communication. To address this issue, a modified domain decomposition is used in the spectral dycores when specifying `phys_loadbalance = 3` that assigns columns and their partners to successive processes. For example, for a one-dimensional *latitude-slice* decomposition (decomposing only over the latitude dimension), if latitude `lat` is assigned to process i , then latitude $(nlat + 1 - lat)$ is assigned to process $i + 1$ or $i - 1$. A similar approach is being investigated for use with the finite volume dycore.

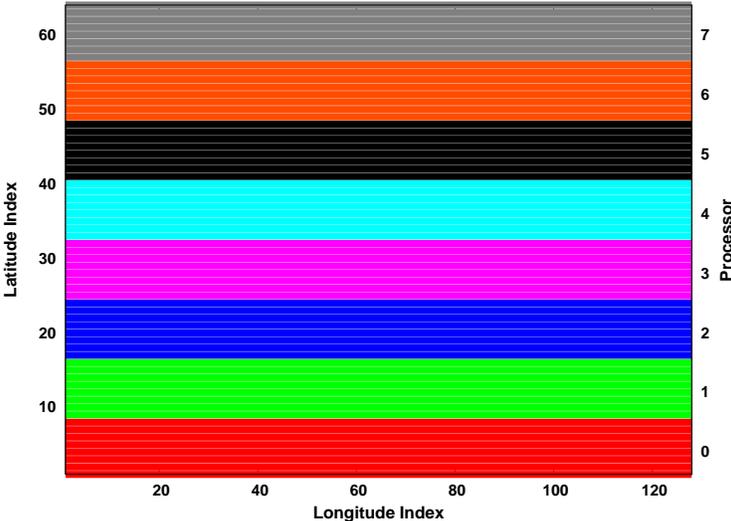


Figure 1: Latitude-slice domain decomposition.

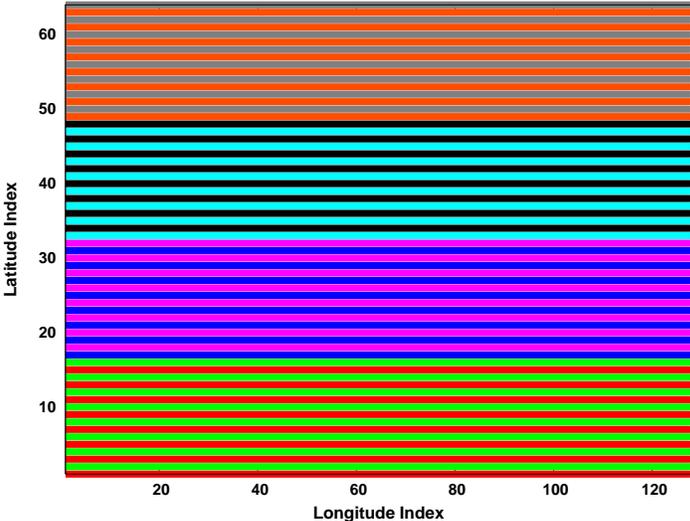


Figure 2: Physics Domain Decomposition: option 1.

Figures 1-3 demonstrate the impact of the different `phys_loadbalance` options on the distribution of columns to processors for an example problem. The problem has a horizontal longitude-latitude grid resolution of (128×64) . The system is a cluster of 2 processor nodes of which we are using 4 nodes (8 processors). Processors 0-1 are on node 0, processors 2-3 are on node 1. etc.. We are not using OpenMP and there are 8 MPI processes, mapped sequentially to the processors. `pcols` is set to 16, so no chunk will have more

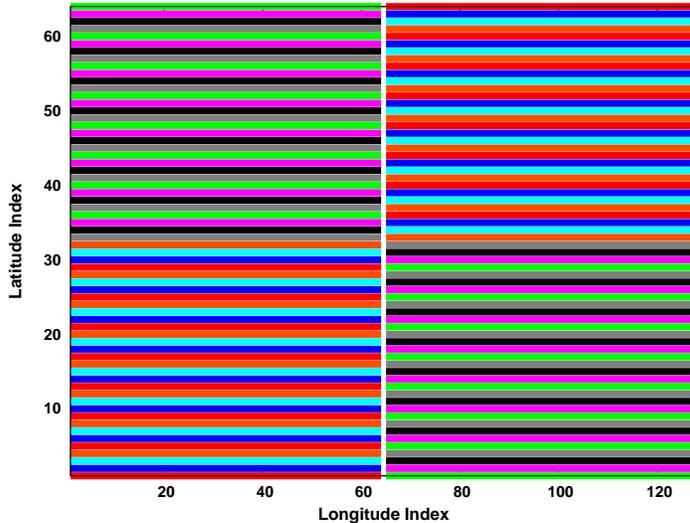


Figure 3: Physics Domain Decomposition: option 2.

than 16 columns. Figure 1 describes the assignment of columns to processors in the standard latitude-slice dynamics decomposition, as well as the processor color code used in all three figures. This is also the physics domain decomposition used when `phys_loadbalance = -1` or 0. Figure 2 describes the assignment when `phys_loadbalance = 1`. Note that latitude lines 1-16 are assigned to the first 2 processors (first node), lines 17-32 are assigned to processors 2 and 3 (second node), etc., and that only intranode communication (between processes 0 and 1, between 2 and 3, etc.) is required when moving from the latitude-slice decomposition. Figure 3 describes the assignment when `phys_loadbalance = 2`. Now latitude line 1 is assigned to processors 0 and 1, latitude line 2 is assigned to processors 2 and 3, etc. With this assignment each process communicates with all of the other processes when moving from the dynamics decomposition.

5 Performance Implications

The chunk data structure as implemented in CAM has a number of useful performance characteristics.

Backward Compatibility. Assuming a latitude-slice domain decomposition in the dynamics for one of the current dycores, CAM can be run in essentially the same fashion as CCM by setting `pcols = nlon + 1 + 2 * nexpt`, forcing `ncols = nlon`, and disabling both column pairing and wrap map in the chunk creation algorithm. This is a supported load balancing option (`phys_loadbalance = -1`). We will use this to estimate the performance improvement due to chunking and load balancing in CAM. Note that memory copies still occur in `dp_coupling` when using this option, but performance timers can be used to measure this cost and subtract it from the performance estimates. This is discussed in Section 8.

Vectorization. By setting `pcols` larger than `nlon`, even longer vector lengths can be achieved than were possible in CCM. Note however that `ncols` will be smaller than `pcols` if there are too many threads of execution. Precedence is given to assigning equal amounts of work to all threads over preserving long vector lengths.

Memory Hierarchy. Setting `pcols` small enough will decrease the sizes of chunks that would normally occur for a given problem size and number of threads. This changes the memory traffic patterns in the computation of a chunk. Given the complexity of the code and the size of the auxiliary arrays, it is unclear whether we can keep all data needed for physics computations in cache even when setting `pcols = 1`. Setting `pcols` too small also decreases opportunities for data reuse. But the amount of memory traffic into the deeper

levels of the memory hierarchy can be controlled somewhat with `pcols`. The optimal setting is determined experimentally on each target platform.

Memory Alignment. Experiments early in the development of the chunk data structure indicated that compilers produce more efficient code when the first dimension (`pcols`) is specified at compile time, somewhat independent of the actual number of columns assigned to a chunk (`ncols`) [26]. `pcols` can also be used to control memory bank or cache line conflicts by being set appropriately. For example, arrays whose sizes are a power of two often suffer performance problems. Since `pcols` is set at compile time, and is independent of the problem size and number of processors, the value can be chosen to take into account the memory structure of the target system.

Parallelization and Scalability. Since `nchunks` is (typically) a multiple of the number of threads of execution, OpenMP parallelism can also be exploited in the physics. OpenMP parallelism is applied at the same level as MPI parallelism in the physics, and is equally effective for modest numbers of threads per MPI process on current SMP cluster architectures.

This is important because the percentage of time spent in MPI communication can become large for large numbers of MPI processes on systems with slow interprocessor communication or when the particular MPI process count leads to load imbalances in the dynamics. It is sometimes not productive to use all of the MPI parallelism that is available to a dycore for a given problem size. Examples of load imbalance in the spectral dycores can be seen in the IBM benchmarks described in Section 8, where using fewer than the maximum number of MPI tasks is more efficient for certain processor counts. Both MPI and OpenMP parallelism are also used to optimize performance when running on large IBM clusters with the finite volume dycore [18].

Efficient OpenMP parallelism in the physics is especially important when using the spectral dycores. MPI parallelism is more efficient than OpenMP parallelism in the spectral dycores, and little performance improvement is gained in the spectral dycores from OpenMP parallelism once MPI parallelism is exhausted. Also, the spectral dycores are currently limited to one-dimensional domain decompositions, which severely limits their scalability. Because the runtime is dominated by the physics for current problem sizes, it is often worthwhile exploiting OpenMP parallelism when MPI parallelism is exhausted, even though many of the processors are effectively idle during the dynamics. This can also be seen in the IBM benchmarks described in Section 8. (Note that the finite volume dycore is not limited in the same fashion as the spectral dycores. In particular it supports a two-dimensional decomposition, enabling both more MPI parallelism and efficient support for OpenMP parallelism.)

Load Balancing and Communication Costs. Load balancing was discussed in detail in the previous section. Load balancing comes at the cost of communication overhead, and the best load balancing option is sensitive to the communication and computation characteristics of the target system.

6 Communication Optimizations

In order for load balancing to be effective, the communication overhead in the dynamics-physics coupling needs to be minimized. Load balancing option 3 attempts to minimize communication overhead by limiting communication to process pairs, while option 1 limits communication to processes within the same nodes, taking advantage of the low latency and high bandwidth typically available in shared memory implementations of MPI. We also introduced four runtime parameters for further tuning the communication performance: `phys_alltoall`, `swap_comm_order`, `swap_comm_protocol`, and `swap_comm_maxreq`.

The parameter `phys_alltoall` has four options controlling interprocess communication in `dp_coupling`.

- 0) Use `mpi_alltoallv`.
- 1) Use point-to-point MPI-1 two-sided commands to exchange data between pairs of processes using an exclusive-or ordering of the process pairs.
- 2) Use point-to-point MPI-2 one-sided commands to write directly into remote memory.

	Processor	MHz	L1 cache	L2 cache (per proc. if shared).	L3 cache (per proc. if shared)	Peak Processor Performance (GFlop/sec)
p690	POWER4	1300	32KB	0.72MB	16MB	5.2
p655	POWER4+	1700	32KB	1.44MB	32MB	6.8
Altix	Itanium2	1500	32KB	.25MB	6MB	6.0
X1	Cray MSP	800	16KB per scalar unit	2MB	–	12.8

Table 1: Test Platforms

3) Use Co-Array Fortran [19] to write directly into remote memory.

For `phys_alltoall` options 1-3, communication occurs between processes only if there is data to exchange. Thus, depending on how `mpi_alltoallv` is implemented, options 1, 2, and 3 can be faster than option 0 for `phys_loadbalance` options 1 and 3. Note that if MPI-2 and/or Co-Array Fortran are not supported on a given platform, options 2 and 3 default to an implementation using MPI-1 two-sided commands.

The other three runtime parameters control which of the many MPI-1 two-sided protocols to use when `phys_alltoall = 1`. More than 19 protocols are supported currently. These are generalizations of similar algorithms used in PCCM and will not be described here.

7 Test Systems

We used the following architectures to collect performance data.

- IBM p690. 32-processor SMP. Each processor is a 1.3 GHz POWER4.
- IBM p655. 4-processor SMP. Each processor is a 1.7 GHz POWER4+.
- SGI Altix. Non-Uniform Memory Access (NUMA) shared memory system in which two 2-processor SMPs are connected to form a *C-brick*, and some number of C-Bricks are then connected to form the larger system. Each processor is a 1.5 GHz Itanium2.
- Cray X1. 4 processor SMP. Each processor is a Multi-Streaming Processor (MSP) comprised of 8 32-stage vector units running at 800 MHz, 128 64-bit wide, 64-element deep vector registers, and 4 scalar units running at 400 MHz.

Details for these architectures are summarized in Table 1. For full application benchmarking we used the following systems.

- IBM p690 cluster at Oak Ridge National Laboratory (ORNL): 27 32-processor p690 nodes and an HPS interconnect. Each node has two 2-link network adapters.
- Cray X1 cluster at ORNL: 128 4-processor nodes.

8 Performance Results

Chunk Size and Performance. Figure 4 describes the sensitivity of serial performance to chunk size. CAM was run for two simulation days on a problem with 26 vertical levels and a horizontal grid of size (64×32) , i.e., 2048 vertical columns. Performance data for one simulation day were also collected and the differences between the one day and two day simulation data were examined to verify that no atypical startup overhead contaminated the two day simulation data. Two processors were used in the experiments. (MPI-enabled CAM can not be run on one processor, and all subsequent experiments use MPI.) The two processes were assigned to processors that do not share L2 caches (or L3 caches on the Altix), and all processors sharing cache with these two were idle.

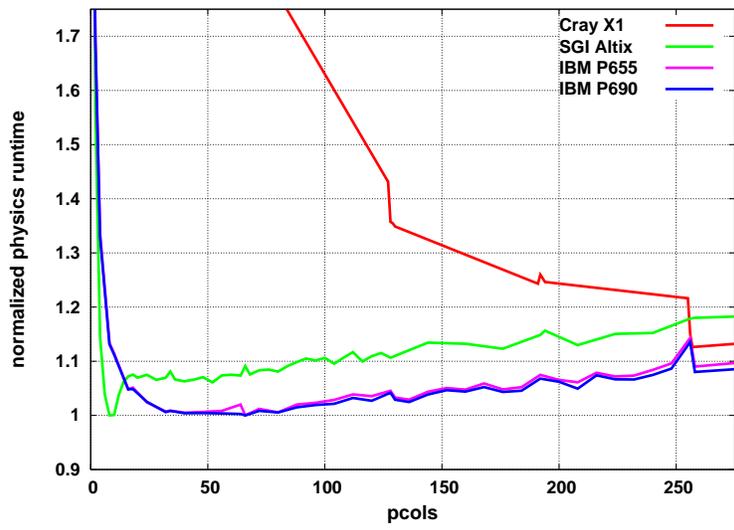
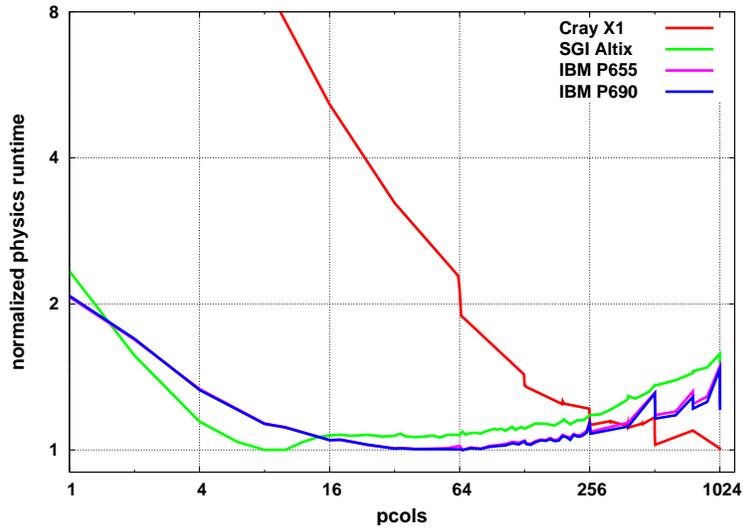


Figure 4: Chunk size experiments (2 processes).

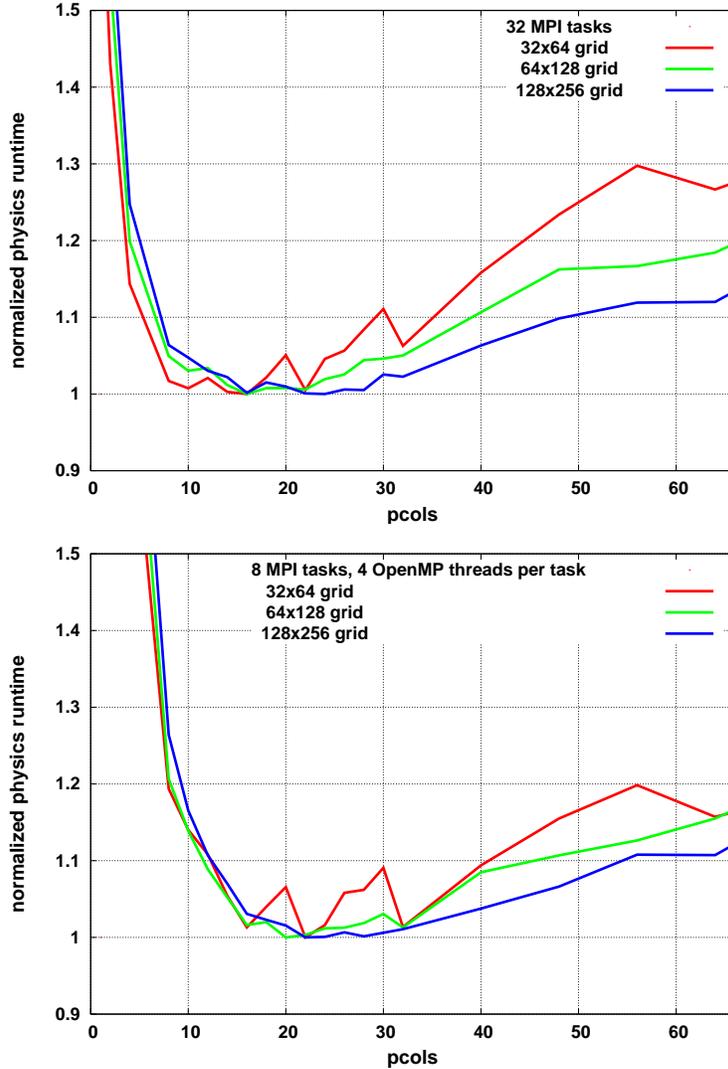


Figure 5: Chunk size experiments (32 processors on IBM p690).

The physics execution times for the two processes were summed, and results for each `pcols` value were normalized with respect to the minimum physics execution time observed over all experiments for a given platform. Experiments were run with both `phys_loadbalance = 0` and `phys_loadbalance = 2`, and the data are similar. The `phys_loadbalance = 2` data are used in Fig. 4. The first graph in Fig. 4 is a log-log plot, while the second graph is a linear-linear plot. The optimal chunk size for this experiment is `pcols = 8` for the Altix, `pcols = 56` for the p690, `pcols = 66` for the p655, and `pcols ≥ 1026` for the Cray vector system. Performance on the IBM systems is near optimal for any `pcols` in the range of 16 to 200, but is degraded for values outside of this range. The optimal `pcols` is better defined on the Altix, but the performance sensitivity is otherwise similar to that observed on the IBM systems. As expected, the Cray vector system prefers long vectors (`pcols` large), but `pcols` can not be set much larger than 1026 because of memory limitations. (`pcols` should also not be set too much larger than the maximum `ncols` in a chunk as this does not increase vector length and it degrades memory performance.) Performance on the X1 is also sensitive to the exact value of `pcols`, preferring non-power-of-two values. Note that the IBM results also show a performance degradation when using a large power-of-two value for `pcols`, e.g. 256, 512, 768, and 1024.

The previous experiment examined issues such as vectorization, cache blocking, and memory alignment.

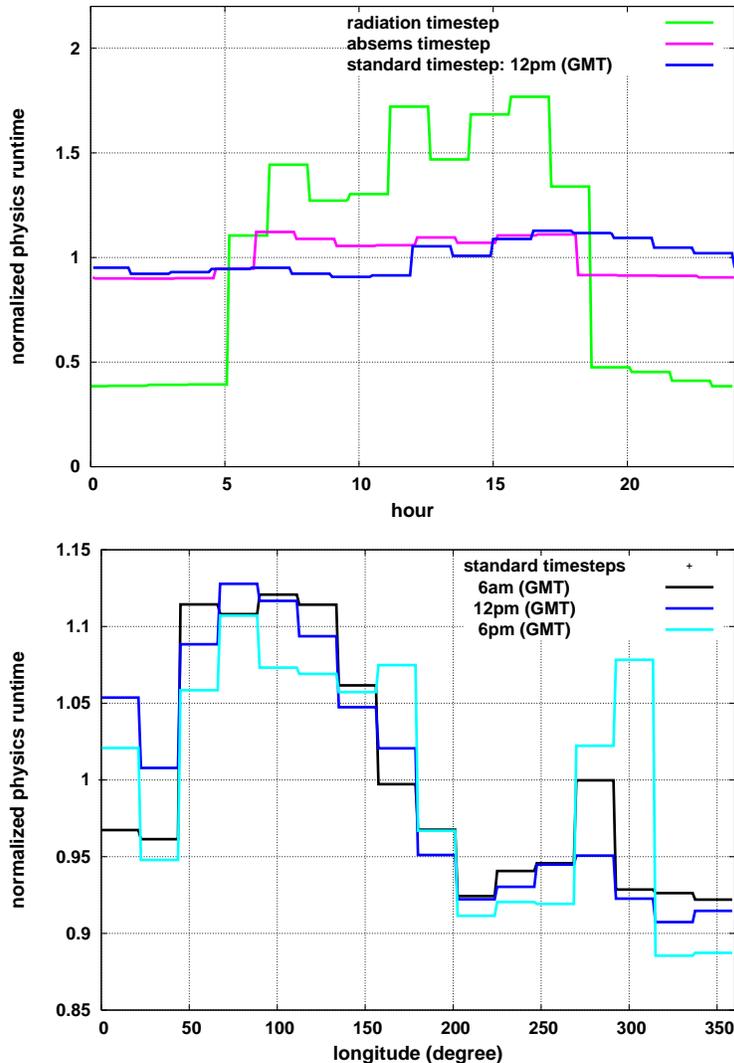


Figure 6: Load variability experiments (single latitude).

However, memory contention can also influence performance. We repeated the previous experiment using all of the processors in an IBM p690 node (32), all of the processors in an Altix C-brick (4), and all of the processors in a Cray X1 node (4). As access to the shared levels of the memory hierarchy change with `pcols`, contention for this access between the different processes will also change. We examined three different problem resolutions: 26 vertical levels and horizontal grids of size (64×32) , (128×64) , and (256×128) , respectively. For the IBM, we also compared the performance for 32 MPI processes and 8 MPI processes with 4 OpenMP threads per process. The results for the Cray X1 and the SGI Altix were unchanged. Figure 5 describes the IBM results. Contention increases the performance sensitivity and decreases the optimal `pcols` value. For these experiments the optimal `pcols` value is between 16 and 32.

Load Balancing and Performance. Figure 6 illustrates the nature of the load variability for the different types of physics timesteps for a single latitude. CAM was run on a problem with 26 vertical levels and a horizontal grid of size (256×128) on 1 node (32 processors) of the IBM p690 cluster with `pcols` set to 16 and using 32 MPI processes. A special load balancing option was used that is equivalent to setting `phys_loadbalance = 0` but with both column pairing and wrap map disabled. Thus consecutive longitudes

were assigned to chunks, and no chunk contained columns from multiple latitudes. The runtime was measured for each chunk for a single radiation timestep, for a single absmes timestep, and for three different standard timesteps. Runtime for individual columns was assumed to be one sixteenth of the runtime for the corresponding chunk. (The performance degradation from using `pcols = 1`, as shown earlier, makes it difficult to measure accurately the cost associated with individual vertical columns.)

Figure 6 contains graphs of the approximate runtime for each column from a latitude line next to the equator, normalized by the average runtime over all columns at the same timestep and latitude. (Note that these columns were all assigned to the same processor.) Thus the normalization is different for each timestep. In the first graph the x-axis is the “calendar” hour for each column for the indicated timestep, and a given column will be at different hours for different timesteps. It is clear from this graph that the primary source of load variability between columns for the radiation and absorptivity-emissivity (absmes) timesteps is the additional cost of calculations at day locations. What is not obvious from this graph is that essentially all of the variability between columns in the absmes timestep is due to the radiation calculation also occurring during this timestep. The absorptivity and emissivity update costs approximately the same for all columns.

The second graph in Figure 6 is the runtime for three different standard timesteps. The x-axis is now the longitude coordinate for each column in degrees. While there are differences in the column load variability between the three timesteps, spatial locality appears to have the strongest correlation with the load. The variability is also much lower than that in the radiation timestep.

Table 2 indicates the cost (runtime for physics summed over all columns in the equatorial latitude) and frequency for the different timesteps in this benchmark problem. While the radiation timestep does not dominate the physics runtime, it is a significant part, and addressing the radiation load variability is important for performance even for a single latitude. The importance grows for the global problem because of the impact of seasons on the polar latitudes.

	standard	radiation	absmes
steps per day	120	22	2
seconds per step	.078	.270	1.63
seconds per day (approximate)	9.36	5.94	3.26

Table 2: Cost and frequency of physics timesteps for benchmark problem on IBM p690.

Figures 7 and 8 show the performance impact of load balancing on the physics. CAM was run for 30 days on a problem with 26 vertical levels and a horizontal grid of size (256×128) on 4 nodes (128 processors) of the IBM p690 cluster and 32 nodes (128 processors) of the Cray X1, both with 128 MPI processes. In Figure 7 the simulated days were September 1 through September 30, so seasonal differences between the hemispheres were small. In Figure 8 the simulated days were January 1 through January 30 and seasonal differences between the hemispheres were large. Experiments were run with `phys_loadbalance` set to 0 and 2, and with `pcols` set to 16 on the p690 cluster and 258 on the X1. The runtime spent in physics for each process is normalized by the average time over all processes for the `phys_loadbalance = 0` experiment for a given platform. This normalized runtime is then plotted as a function of MPI process identification number.

Figures 7 and 8 do not include the communication overhead associated with setting `phys_loadbalance = 2`. However, they do indicate that this static load balancing algorithm eliminates much of the load imbalance seen in the physics when using a latitude-slice decomposition, especially when seasonal imbalances are large. Note that, despite the differences in the processor architectures, the nature of the load imbalance and the impact of the load balancing algorithm are very similar on the IBM and Cray systems.

Figure 9 is a repeat of the IBM graphs in Figures 7 and 8, but with the addition of physics runtime data from an experiment with `pcols = 258` and `phys_loadbalance = -1`. This figure indicates the performance gain in the physics from using a more efficient chunk size as well as load balancing on the p690.

Performance Benchmarks. Our final set of experiments compare the performance of CAM using the (empirically determined) optimal `pcols`, `phys_loadbalance`, and `phys_alltoall` settings with the performance using the physics data structure of the original CCM. However, the data structures in the dynamics have also evolved and not all dynamics arrays are declared with `nlon`. To estimate the performance of the

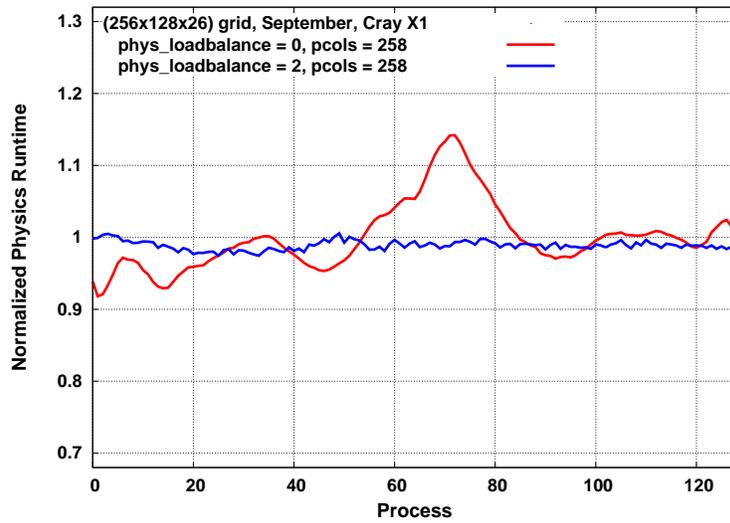
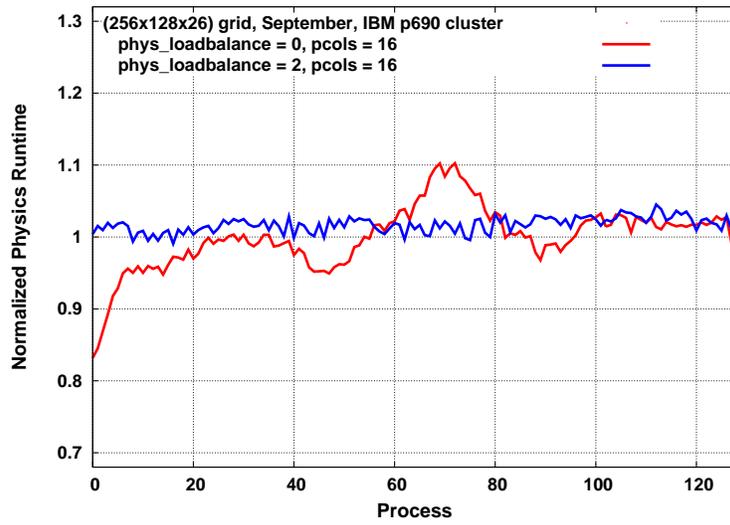


Figure 7: Load balancing experiments (128 processes, September).

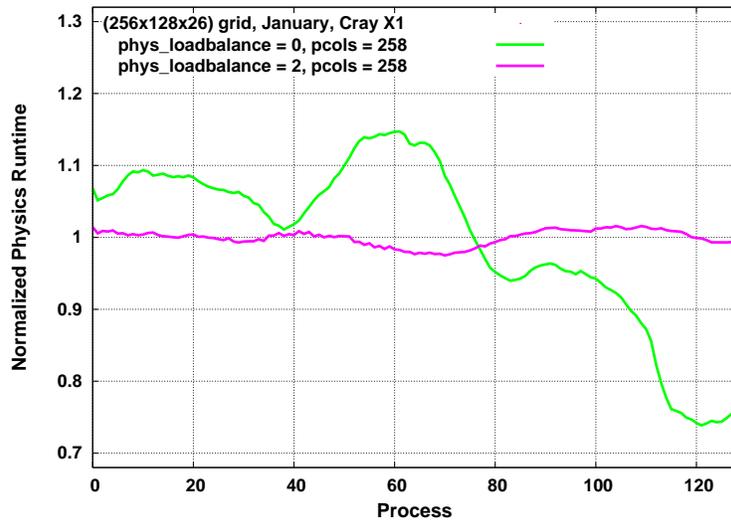
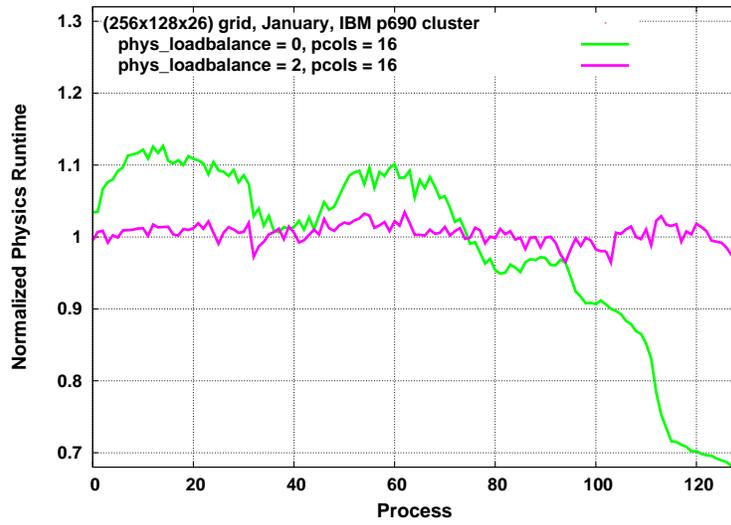


Figure 8: Load balancing experiments (128 processes, January).

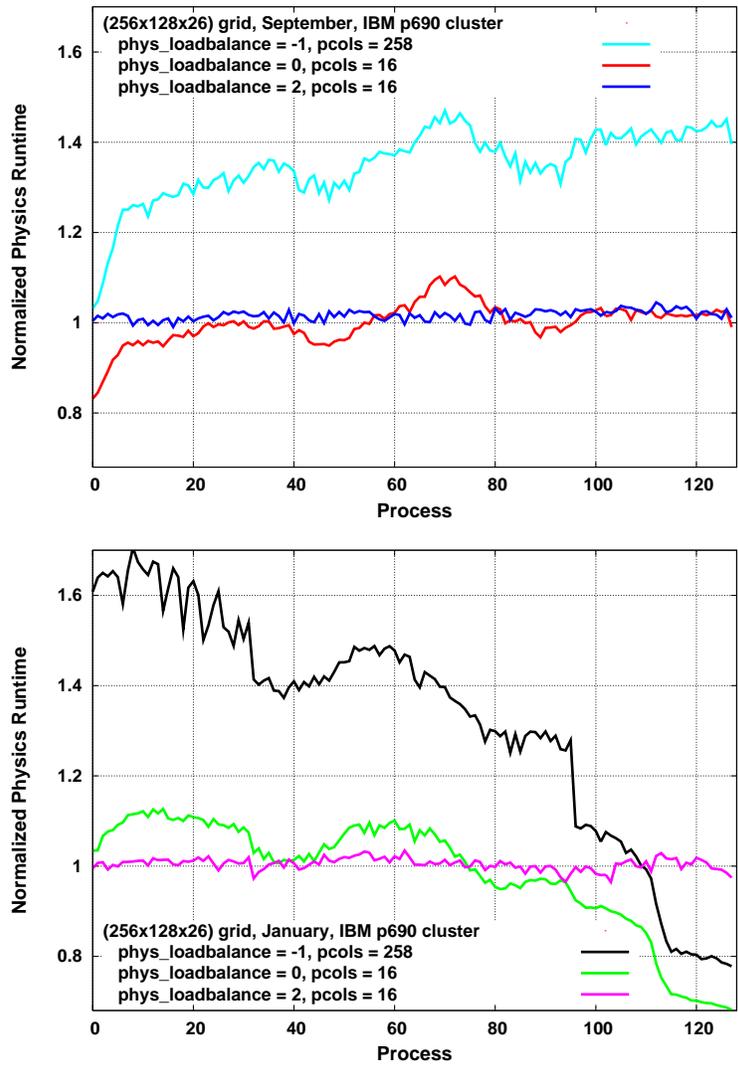


Figure 9: p690 load balancing and chunk size experiments (128 processes).

Proc.	MPI processes	OpenMP threads per process	pcols	phys_loadbalance	phys_alltoall	improvement vs. CCM alg.
32	32	1	16	2	0	28%
64	64	1	16	3	1	28%
96	96	1	24	2	0	36%
128	128	1	16	3	1	25%
160	40	4	24	2	0	21%
192	48	4	24	3	1	33%
256	128	2	16	3	1	47%
320	40	8	32	0	–	48%
384	48	8	24	3	1	62%
512	128	4	16	3	1	91%

Table 3: Optimal Performance Settings for IBM p690 Cluster

CCM data structures and algorithms as they might have been ported into CAM, we present the best results from using `ncols = nlon`, `nlon + 2`, and `nlon + 3`, all using `phys_loadbalance = -1`. We also empirically determine the optimal number of MPI processes and OpenMP threads per process for each processor count for both the CCM and optimal algorithms, presenting only the best results.

We used version 3.0.pl of CAM, available from <http://www.cesm.ucar.edu/models/atm-cam/>, compiled with the internal performance timers disabled. We used the spectral Eulerian dycore, 26 vertical levels, and a horizontal grid of size (256 × 128). This is the same problem resolution and CAM dycore as used in the CCSM coupled climate model [2, 5] for the fourth IPCC (Intergovernmental Panel on Climate Change) assessments [14]. However, we ran with the default problem specifications, characterized by 3 advected constituents, rather than the specifications used in the IPCC scenarios which, in addition, enable greenhouse gas and prognostic sulfur chemistry, use prognostic sulfur in the radiative forcing, and advect an additional 8 constituents.

Optimal settings were determined using one and two day simulations. For the results presented here, runtime was measured for a 30 day simulation, September 1 to September 30, ignoring time spent in model initialization. Performance is reported in terms of simulation years per wallclock day. The results for the IBM p690 cluster and a Cray X1 system at Oak Ridge National Laboratory are described in Figures 10 and 11. As mentioned earlier, memory copies occur in `dp_coupling` even when setting `phys_loadbalance = -1`. In separate runs, we measured the cost of these copies and adjusted the runtimes used to calculate the “CCM settings” curve accordingly. This adjustment was not significant, however, as the cost of the copies was between 1.3% and 1.7% of the total runtime on the p690 cluster and between .8% and 1.2% of the total runtime on the X1.

The optimal settings for the p690 cluster are described in Table 3. 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 columns) increase the amount of OpenMP parallelism available compared to 256 column chunks, improving scalability compared to runs using the CCM settings.

The optimal settings for the Cray X1 are described in Table 4. Here increased vector lengths and load balancing are equally important in improving performance over that achieved using the CCM settings. Note that for 128 processors the vector lengths are identical for the optimal and CCM settings, and the performance difference is due to load balancing only. OpenMP parallelism was not exploited in these runs. Partly this is because OpenMP has not been extensively tested in the current port of CAM to the X1. 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 X1 processor. Thus, while we would be using twice as many processors, the performance of each processor would be approximately halved in the physics. The port of CAM to the X1 is relatively immature, and we expect improved vectorization in the future, which will further emphasize

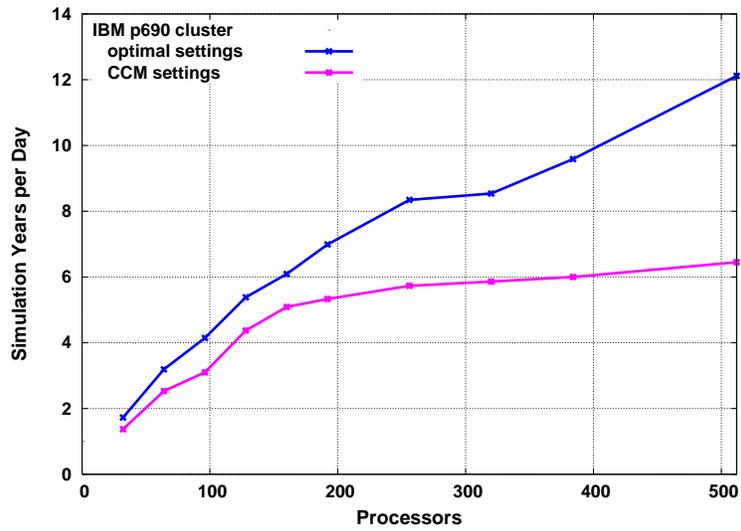


Figure 10: Performance comparison on IBM p690 cluster.

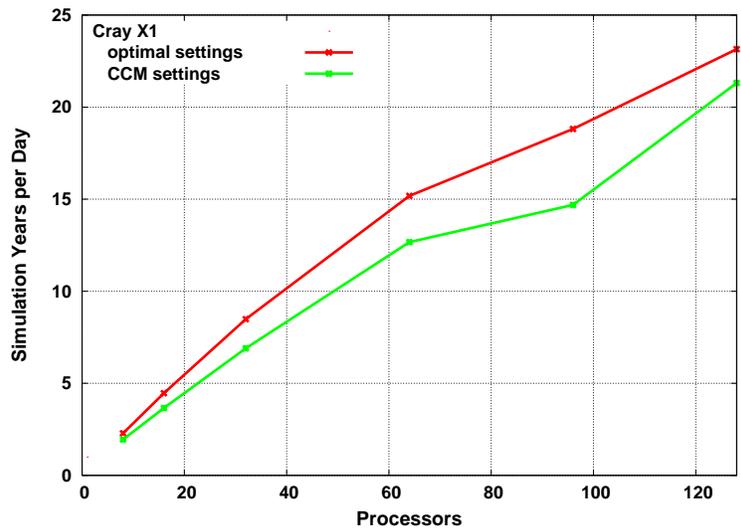


Figure 11: Performance comparison on Cray X1.

Proc.	MPI processes	OpenMP threads per process	pcols	phys_loadbalance	phys_alltoall	improvement vs. CCM alg.
8	8	1	1026	2	0	19%
16	16	1	1026	2	0	23%
32	32	1	1026	2	0	24%
64	64	1	514	2	0	21%
96	96	1	514	2	0	30%
128	128	1	258	2	0	10%

Table 4: Optimal Performance Settings for Cray X1

the importance of preserving vector lengths.

9 Conclusions

The decision to decouple the physics and dynamics data structures incurred copy overhead and required additional memory, but was justified by the ability to support multiple dynamical cores. The increase in memory requirements has not proven to be limiting, as the amount of memory in current HPC systems has been adequate for the problem sizes of interest. In this paper we have shown that careful optimizations of the physics data structures and introduction of load balancing and communication optimization options have led to significant performance improvements, more than offsetting the small amount of copy overhead introduced by decoupling. On nonvector systems, support for additional OpenMP parallelism and efficient load balancing has proved to be as important as improved cache blocking, our initial motivation for the chunk data structure. On vector systems, the ability to increase vector length to longer than the processor vector length has proven to be more significant than expected.

The MPI-2 and Co-Array Fortran communication options did not improve performance of the load balancing communication on the Cray X1, and the MPI-2 communication option did not improve performance on the IBM p690 cluster. However, other researchers have show significant performance improvements from using MPI-2 on other platforms when using the finite-volume dycore [21]. We consider the inclusion of the MPI-2 and Co-Array Fortran options to be *defensive programming*, providing an alternative when implementations of MPI point-to-point or collective routines are not efficient.

The experiments described in this paper used a *full grid*, in which each line of constant latitude has the same number of longitude points. CAM also supports using a *reduced grid* where, for example, fewer longitude points are used near the poles. The chunk data structure works equally well with a reduced grid, and load balancing becomes even more important as the standard domain decomposition in the dynamics is even more load imbalanced in the physics. This is the first example of using the chunk data structure with something other than a longitude-latitude computational grid. Preliminary studies with a spectral element method using a more general finite element-like grid have been promising. However, the current implementation of load balancing is specific to a longitude-latitude grid (full or reduced). While this is not necessary, we have delayed a general implementation until the next dycore is ready to be incorporated into CAM.

10 Acknowledgements

We gratefully acknowledge the support of the DOE Office of Biological and Environmental Research, Climate Change Prediction Program and the Center of Computational Sciences at Oak Ridge National Laboratory for access to the IBM p655 and p690, SGI Altix, and Cray X1 systems used in this paper. We feel fortunate to have been able to contribute to the development of CAM. The success of our efforts has been due to close collaboration with the many software engineers and computational scientists involved in CAM development, especially those in the Climate and Global Dynamics Division at NCAR and our colleagues in the SciDAC [20] project *Collaborative Design and Development of the Community Climate System Model*

for *Terascale Computers*. In particular, we wish to thank Byron Boville and David Williamson of NCAR for leading the initial efforts to decouple the physics and dynamics.

References

- [1] S. R. M. BARROS, D. DENT, L. ISAKSEN, G. ROBINSON, G. MOZDZYNSKI, AND F. WOLLENWEBER, *The IFS model: A parallel production weather code*, *Parallel Computing*, 21 (1995), pp. 1621–1638.
- [2] 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.
- [3] C. ZENDER AND B. BRIEGLEB, *CRM Homepage*. <http://http://www.cgd.ucar.edu/cms/crm/>.
- [4] 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.
- [5] COMMUNITY CLIMATE SYSTEM MODEL. <http://www.cesm.ucar.edu/>.
- [6] L. DAGUM AND R. MENON, *OpenMP: : An industry-standard API for shared-memory programming*, *IEEE Computational Science & Engineering*, 5 (1998), pp. 46–55.
- [7] C. DOUGLAS, J. JU, M. KOWARSCHIK, U. RUEDE, AND C. WEISS, *Cache optimization for structured and unstructured grid multigrid*, *Electronic Transactions on Numerical Analysis*, 10 (2000), pp. 21–40.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] I. T. FOSTER AND P. H. WORLEY, *Parallel algorithms for the spectral transform method*, *SIAM J. Sci. Comput.*, 18 (1997), pp. 806–837.
- [12] W. GROPP, M. SNIR, B. NITZBERG, AND E. LUSK, *MPI: The Complete Reference*, MIT Press, Boston, 1998. second edition.
- [13] 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.
- [14] INTERGOVERNMENTAL PANEL ON CLIMATE CHANGE. <http://www.ipcc.ch/>.
- [15] 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.
- [16] 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.

- [17] S.-J. LIN, *A ‘vertically Lagrangian’ finite-volume dynamical core for global models*, Mon. Wea. Rev., 132 (2004), pp. 2293–2307.
- [18] A. MIRIN AND W. B. SAWYER, *A scalable implementation of a finite-volume dynamical core in the Community Atmosphere Model*, International Journal of High Performance Computing Applications, (2005). (this issue).
- [19] R. W. NUMRICH AND J. K. REID, *Co-Array Fortran for parallel programming*, ACM Fortran Forum, 17 (1998), pp. 1–31.
- [20] OFFICE OF SCIENCE, U.S. DEPARTMENT OF ENERGY, *Scientific Discovery Through Advanced Computing*. (available from http://www.sc.doe.gov/ascr/mics/scidac/SciDAC_strategy.pdf), March 2000.
- [21] 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, (2005). (this issue).
- [22] A. SAWDEY, M. O’KEEFE, AND W. JONES, *A general programming model for developing scalable ocean circulation applications*, in Proceedings of the Seventh ECMWF Workshop on Use of Parallel Processors in Meteorology, Reading, England, December 1996, World Scientific Publishing Co. Pte. Ltd., Singapore, 1997.
- [23] R. C. WHALEY, A. PETITET, AND J. J. DONGARRA, *Automated empirical optimization of software and the ATLAS project*, Parallel Computing, 27 (2001), pp. 3–35. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).
- [24] 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.
- [25] D. L. WILLIAMSON AND P. J. RASCH, *Two-dimensional semi-Lagrangian transport with shape-preserving interpolation*, Mon. Wea. Rev., 117 (1989), pp. 102–129.
- [26] 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.
- [27] ———, *Scaling the unscalable: a case study on the Alphaserver SC*, in Proceedings of the IEEE/ACM SC2002 Conference, Nov. 16-22, 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002.
- [28] 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.