LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-206816

A scalable implementation of a finite-volume dynamical core in the Community Atmosphere Model

A. A. Mirin, W. B. Sawyer

September 27, 2004

International Journal of High Performance Computing Applications

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

A scalable implementation of a finite-volume dynamical core in the Community Atmosphere Model

Arthur A. Mirin
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California 94550
USA

William B. Sawyer
Swiss Federal Institute of Technology (ETHZ)
Rämistrasse 101, 8092 Zurich
Switzerland

Proposed running head: Scalable Atmospheric Modeling

Corresponding author:

Arthur A. Mirin
Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550
USA
925 422 4020
925 423 2993 (fax)
mirin@llnl.gov

Additional author:

William B. Sawyer
Swiss Federal Institute of Technology
Rämistrasse 101
8092 Zurich
Switzerland
011 41 41 790 0015
011 41 1 350 0285 (fax)
sawyer@env.ethz.ch

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48, and NASA's Goddard Space Flight Center, NASA Task# 00-9103-01a. This is LLNL Report UCRL-JRNL-206816.

Summary

A distributed memory message-passing parallel implementation of a finite-volume discretization of the primitive equations in the Community Atmosphere Model is presented. Due to the data dependencies resulting from the polar singularity of the latitude-longitude coordinate system, we employ two separate domain decompositions within the dynamical core – one in latitude/level space, and the other in longitude/latitude space. This requires that the data be periodically redistributed between these two decompositions. In addition, the domains contain halo regions that cover the nearest neighbor data dependencies. A combination of several techniques, such as one-sided communication and multithreading, are presented to optimize data movements. The resulting algorithm is shown to scale to very large machine configurations, even for relatively coarse resolutions.

Introduction

Atmospheric general circulation models (AGCMs) are key tools for weather prediction and climate research. They also require large computing resources: even the largest current supercomputers cannot keep pace with the desired increases in the resolution of these models. AGCMs consist, roughly speaking, of the *dynamics*, which calculates the atmospheric flow, and the *physics*, in which parameterizations for subgrid phenomena such as long- and short-wave radiation, moist processes, and gravity wave drag, are approximated. The physical parameterizations will not be discussed further here. We concentrate on the finite-volume solver of the dynamics (referred to as the *dynamical core*) in the Community Atmosphere Model (CAM; Collins, et al., 2004; McCaa, et al., http://www.cgd.ucar.edu/csm/models.atm-cam).

The finite-volume dynamical core, or *FV dycore* for short, was originally developed during the 1990's by Lin and Rood at the NASA Goddard Space Flight Center (Lin and Rood, 1996; Lin and Rood, 1997; Lin, 2004). Until that time most prominent general circulation models used either a finite-difference method or a spectral method. Finite-difference techniques, when applied on a latitude-longitude mesh, are limited in their allowable time step due to the polar singularity, or alternatively have to perform costly polar filtering to damp out unwanted modes (Arakawa and Lamb, 1977; Wehner, et al, 1995). The pure spectral approach, due to its costly transforms, is known to not scale well with horizontal resolution. The finite-volume approach discussed here addresses both of these issues while maintaining the traditional latitude/longitude grid. It additionally conserves dry air mass and the mass of transported atmospheric constituents, which is particularly important for climate simulations due to their long integration times, and imperative for

accurate representation of chemical interactions in climate/chemistry simulations. It should be noted that there are ongoing efforts to address both the time step limitation and wasted computation due to unnecessary polar resolution, by way of reduced grids and alternate grids such as the cubed sphere (Rancic, Purser and Mesinger, 1996).

The finite-volume dycore uses a Lagrangian vertical coordinate, where the Lagrangian surfaces are in fact material surfaces that bound the control volumes used for the integration. The initial coordinate is terrain-following, and the finite volumes are free to float, compress or expand as dictated by the hydrostatic dynamics (Lin, 2004). This reduces the dimensionality of the main dynamics from three to two. The resulting system closely resembles the shallow water equations, with pressure thickness (which is proportional to the mass) playing the role of fluid depth. A system of equations for mass, constituent concentration, heat, and horizontal momentum within each control volume is discretized using a conservative flux-form semi-Lagrangian algorithm (Lin and Rood, 1996). Smaller time steps are required for the semi-Lagrangian algorithm to solve the dynamical equations as compared to the constituent transport. A mass, momentum and total energy conserving algorithm is used to remap the prognostic variables from the floating Lagrangian control volumes to an Eulerian terrain-following coordinate, thereby preventing undue distortion of the Lagrangian surfaces. The remapping typically occurs on a time scale several times longer than that of the constituent transport. After development at NASA Goddard, this methodology was installed in the NCAR-based Community Atmospheric Model.

The efficient parallelization of the finite-volume dynamical core is not trivial, in part due to the latitude-longitude nature of the underlying grid. The convergence of meridians at the poles brings

not only well-known numerical challenges, but creates software challenges as well. We will see that for the method presented here to scale to a reasonably large processor set, it is necessary to redistribute the data on a regular basis. We present a variety of techniques using advanced features of the current Message-Passing Interface standard MPI-2 (http://www.mpi-forum.org), and the OpenMP standard for multithreading (Dagum and Menon, 1998), to handle the redistribution efficiently. Many of these techniques are new in the field of atmospheric modeling, and we believe an extensive analysis is key to attaining the highest possible performance of CAM in a production environment.

The message-passing parallelization with both one- and two-dimensional decompositions is treated in Sec. 1, with a discussion of the underlying communication primitives in Sec.2. In Sec. 3 the optimizations for improved performance on large parallel computers are presented. Results are presented in Sec.4, where it is seen that the approach scales well to large machine configurations. Additional conclusions and future directions are presented in Sec. 5.

1. Parallelization

The solution procedure for the FV dynamical core is divided into two portions: (1) dynamics and transport within each control volume (referred to as *cd_core* for the main dynamics and *trac2d* for the tracer advection) and (2) remapping from the Lagrangian frame back to the original vertical coordinate (referred to as *te_map*). Time-integrations within control volumes are two-dimensional in nature and are vertically decoupled, except for the solution of the hydrostatic equation, which is inherently a vertical integration (referred to as *geopk*). Dynamics is subcycled with respect to tracer transport, and solution of the hydrostatic equation occurs on the faster time scale. All of these

components were first parallelized with the OpenMP shared memory multitasking paradigm, and respectable performance was obtained on up to 16-32 CPUs on an SGI Origin 2000 (Sawyer, 1999).

The FV dynamical core, however, contains considerably more inherent parallelism. It is clear that the *cd_core* calculation at each level is independent of all others (with the exception of *geopk*). There is no reason, beyond convenience, to parallelize the vertical calculation with shared memory parallelism alone. If there are enough levels, these can be separated into contiguous sets, much like packages of sliced cheese (see Fig. 1). Within each package it is still possible to employ OpenMP parallelism on the small number of local levels or with respect to latitude. This approach is employed on clusters of shared memory nodes, such as the IBM SP, which was used (along with the SGI Origin) for the subsequent performance tests.

The stencil of points needed for one finite-volume time-iteration is determined by the spatial accuracy order of the algorithm and the geographical separation of the grid points, as dictated by the dimensionless Courant numbers:

$$C^{\lambda} = u\,\Delta t / (R\,\Delta\lambda \cos\theta) \qquad C^{\theta} = v\,\Delta t / (R\,\Delta\theta)$$

where λ and θ are longitude and latitude, resp., u and v are the zonal (longitudinal) and meridional (latitudinal) velocities, resp., and R is the earth's radius. The Courant number represents the number of mesh widths that a signal travels in one time step. If the difference scheme uses just nearest neighbors, the Courant number must generally be less than unity. Hence, use of larger time

steps requires the domain of dependence of the difference scheme to extend beyond nearest neighbors. In latitude the geographical separation $\Delta\theta$ is constant. Therefore, if Δt is chosen appropriately, and the wind speeds, u and v, remain in an atmospherically realistic range, only the accuracy order of the algorithm is significant, and there are limited north-south neighbor dependencies (1, 2, or 3 lines of latitude) on each level. A similar statement for the *cd_core* calculation in λ is not possible; even if the longitudinal separation $\Delta\lambda$ is constant, the cos θ term goes to zero at the poles. In order to solve the *pole problem* of converging meridians near the pole, a semi-Lagrangian approach is employed to determine the fluxes in λ. The quantity C^{λ} will typically become large, and the semi-Lagrangian method will have dependencies on grid points that are at geographical distances dictated by the departure point for a given Δt. Near the poles, this set goes well beyond the immediate east-west neighbors.

In the one-dimensional domain decomposition algorithm the domain is cut into latitude slabs, with each slab (or more abstractly *decomposition element*, subsequently referred to as a *DE*) maintaining a *halo* (or *ghost*) region on both the north and south, as illustrated in Fig.1. The horizontal calculation can be performed in a distributed memory setting: the halo regions are first exchanged using message-passing, and the latitude-slab calculation is then performed independently on all *DE*'s. In the vertical, shared memory parallelism is still utilized, resulting in a hybrid-parallel model. The one-dimensional domain decomposition is applicable to both *cd_core* (including *geopk*) and *te_map*.

In the two-dimensional domain decomposition formalism, *cd_core* is decomposed in latitude and level. Unfortunately this decomposition is no longer appropriate for *te_map* and the vertical

integration for the calculation of the pressure term (known in atmospheric modeling as ρ^{κ}) in *geopk*; here the dependencies are vertical instead of horizontal. For these components the domain is best decomposed in latitude and longitude (see Fig. 2). This requires the constituent arrays to be redistributed, or *transposed,* from a latitude-level to a longitude-latitude decomposition before the vertical calculation and back thereafter. The communication pattern for the transposes can be quite involved. However, if the latitudinal sub-decomposition is the same for both two-dimensional decompositions, then the communication pattern corresponds to that of shifting between one-dimensional decompositions in longitude/vertical, for each latitudinal subdomain. We have applied this multi-two-dimensional domain decomposition technique, putting much emphasis on a highly optimized transpose operation. In all cases a single MPI task is assigned to a subdomain.

2. Communication Primitives

At the lowest level, we have based the parallelization on the *pilgrim* (Sawyer and Messmer, 2002) and the *mod_comm* (Putman, Lin and Shen, 2004) libraries. The former is designed for general, unstructured domain decompositions while the latter was originally designed for the types of structured communication, in particular halo exchanges, needed for one- and two-dimensional rectangular domain decompositions. *Mod_comm* optionally invokes one-sided communication based on the MPI-2 standard, and combines this with OpenMP multithreading; this is supported on a number of architectures, such as the SGI Origin.

We have added facilities for the types of irregular communication inherent to the transpose, namely exchange of unequally sized data chunks between decomposition elements (*DE'*s). The following

Fortran-90 *type* is used to describe a set of non-uniform blocks by their memory displacements and sizes:

```
TYPE BlockDescriptor
  INTEGER                :: method
  INTEGER                :: type
  INTEGER, POINTER       :: displacements(:)
  INTEGER, POINTER       :: blocksizes(:)
  INTEGER                :: partneroffset
  INTEGER                :: partnertype
END TYPE BlockDescriptor
```

Within each communication paradigm (MPI-1 or MPI-2), more than one communication *method* is available. Depending on which *method* is used, the *type* may contain an MPI derived datatype; otherwise the *displacements* and *blocksizes* may be used directly. The *partneroffset* is used to specify the target location, and the *partnertype* the target datatype, when invoking MPI-2. Different communication methods can be used for different code sections; such specification can be made at run-time, although the choice of whether to invoke MPI-2 must be made at compile time. Various communication methods are discussed in the following section.

Pilgrim defines *communication patterns* using the *ParPatternType*, which contains two arrays of extent [1: #DE's] - the *send descriptor*, an array of block descriptors corresponding to data to be sent, and the *receive descriptor*, an array of block descriptors corresponding to data to be received.

```
TYPE ParPatternType
  INTEGER                              ::   Comm
  INTEGER                              ::   Iam
  INTEGER                              ::   Size
  TYPE(BlockDescriptor), POINTER :: SendDesc(:)
  TYPE(BlockDescriptor), POINTER :: RecvDesc(:)
END TYPE ParPatternType
```

A given data array *A* may have two different decompositions, *D1* and *D2*. The redistribution, or transpose *R* is the communication pattern

$$R: A_{D1} \rightarrow A_{D2}$$

That is, redistributions are a function only of the data decompositions and can be calculated once at initialization and stored. The *mod_comm* library has been extended to handle these irregular datatypes through the routines *mp_sendirr* and *mp_recvirr*, which are analogous in functionality to the regular variants *mp_send* and *mp_recv*. The *mp_sendirr* primitive requires specification of both the send and receive descriptors (both are needed to support one-sided communication), while *mp_recvirr* requires only the receive descriptor.

3. Optimizations

In our original MPI-1 implementation, data was gathered into a contiguous send buffer, communicated using the nonblocking *MPI_Isend* and *MPI_Irecv* primitives, and then scattered to its target destination. Our first optimization was to define MPI derived data types in order to circumvent the temporary contiguous buffers. As MPI performance varies with the local implementation, both approaches continue to be supported.

The enhanced MPI-2 standard offers one-sided communication through the *MPI_Put* primitive. One-sided communication requires the definition of an MPI-2 *window*, which designates the areas of memory that can receive remote data. As with MPI-1, one can specify either contiguous data or

MPI derived data types. In addition, the MPI-2 standard provides for multithreading of communications.

These choices led us to consider four methods for one-sided communication (see Fig. 3). In the most basic method (denoted as method A), data is gathered into a contiguous buffer, written to a contiguous target using *MPI_Put*, and scattered to its final destination. The memory associated with the target window is dedicated to receiving data from *MPI_Put* and is allocated either statically or with *MPI_Alloc_mem*. The target window itself is defined at initialization and remains in place throughout the calculation. The communicated data is partitioned into blocks to enable multithreading using OpenMP.

In method B, each contiguous block of source data is instead written directly into the target window using *MPI_Put*, with multithreading carried out with respect to the blocks of source data. The tradeoff is between the gather step of method A and the extra *MPI_Put* operations of method B.

In method C, the source data is defined by a single MPI derived type and directly written into the target using *MPI_Put*. Threading is carried out with respect to the target destinations. Compared to method B, there are two tradeoffs – multiple *MPI_Put* operations of contiguous blocks of source data versus a single *MPI_Put* of an MPI derived type, and threading over the source blocks versus over the target destinations.

In method D, the source data is defined by a single MPI derived type (as with method C), but the data is instead placed directly into its target destination using *MPI_Put*, bypassing the dedicated

target window. This method requires the definition of a target window for each target array. It would seem that these target windows could be defined at initialization, thereby saving the window creation overhead. However, some of the arrays that undergo communication operations are temporary arrays that are allocated and deallocated each time step, and their location in memory will therefore vary throughout the course of the calculation. Since memory windows are defined in terms of locations rather than syntactic array names, method D might therefore incur a significant windowing overhead. An additional problem with method D is that some machines (such as the SGI Origin) restrict the storage types associated with target windows, with ordinary dynamically allocated memory not qualifying.

The geopotential calculation *geopk* is a vertical integration within the dynamics calculation that occurs on the fast dynamics time scale. This is at a point in the calculation when the domain is decomposed in the latitude and vertical directions. For a one-half degree mesh, the fast dynamics time scale is typically 16 times faster than that of the physics and remapping; hence it is imperative that the *geopk* communications be optimal. Our original approach to computing these *indefinite* integrals was to transpose the relevant arrays before and after this operation. We have instead developed an alternative approach that obviates the need for transposes (see Fig. 4). To compute an upward indefinite integral, *local* partial sums are computed within each subdomain; each *complete local* sum is then communicated to all *higher* subdomains; the *complete local* sums and *local* partial sums are added together accordingly to give *global* partial sums. This method does not require transposes - only the communication of the partial sums to all *higher* subdomains. The partial sum method gives round-off variations with respect to the number of vertical subdomains,

due to differences in the order of summation. But a quadruple precision mode to ensure bit-wise reproducibility over all possible parallel configurations is available for debugging purposes.

4. Results

The FV dynamical core was tested at 0.5^o x 0.625^o, 1^o x 1.25^o and 2^o x 2.5^o horizontal resolutions, corresponding to horizontal grid sizes of 576 x 361, 288 x 181 and 144 x 91, respectively. Twenty-six (26) vertical levels were used for the majority of the cases; some simulations were run using 66 levels. Unless otherwise specified, the partial sum method was used for the geopotential calculation, and MPI-1 with derived types was used for the communications. The target platforms were the SGI Origin 3800 *Chapman* (at NASA Ames) with 1024 MIPS R14000 CPUs @ 600 MHz, and the IBM SP *Seaborg* (at DOE LBNL/NERSC) with 380 Nighthawk nodes, each with 16 Power-3 CPUs @ 375 MHz. All runs used version 3 of CAM. For timing purposes cases were generally run for one simulated day and repeated, with output time excluded from the throughput figures.

The one-dimensional decomposition was extensively evaluated at the 1^o x 1.25^o x 26L resolution on the SGI Origin with various numbers of latitude slabs per subdomain and OpenMP threads per slab, using different communication paradigms. Table 1 gives an overview of the timing results for the entire FV dynamical core in CAM using MPI-1 with intermediate buffers, MPI-1 with derived data types, and MPI-2 with intermediate buffers (method A). Figure 5 shows some of these results graphically. As the number of OpenMP threads increases, the performance using MPI-2 becomes correspondingly better than that using MPI-1. Because the performance of the computational

portions is independent of the communication paradigm, the increase in relative performance with MPI-2 (versus MPI-1) with thread count must be attributable to the multithreading in the halo exchange communication. In fact, closer investigation of the communication timings indicates excellent speedup in the halo exchange. These results are in line with those of Putman, Lin and Shen (2004).

Figure 6 compares the timing percentiles of various components of CAM on the IBM SP when run with the FV dynamical core at the 0.5^o x 0.625^o x 26L resolution, both at small (32) and large (2944) processor count. The 32-processor case corresponds to a 4x1 latitude/vertical decomposition and the 2944-processor case to a 92x4 decomposition, each with 8 OpenMP threads per task. The latter case runs approximately 25 times faster than the former (see Fig. 8), which corresponds to a relative parallel efficiency of about 27%. Some of the components scaling the worst and best are part of the physical parameterizations, which is separate from the dynamical core. Some of these *physics* components scale well because they are communication-free. The land-surface model scales by far the worst and is a known bottleneck at very large processor count; this is due primarily to an insufficient computational load per land point. All components of the dynamical core scale reasonably well to large processor count. For example, the main FV dynamics runs about 30 times faster. At 2944 processors the transposes required for the remapping (*transposes*) account for only 3% of the overall computation. Even the geopotential routine *geopk*, which has been the Achille's heel (scaling-wise) of FV dycore due to its vertical coupling and frequent execution, now scales reasonably well; its fractional time within the dycore increases from 21% to just 26% in going from 32 to 2944 processors.

Figure 7 illustrates the overall scalability of CAM on the IBM SP at the 1^o x 1.25^o x 26L resolution, in simulated days per day of wall-clock time. This includes both the dynamical core and the column physics. We consider two-dimensional decompositions having up to 48 subdomains in latitude and 7 subdomains in the vertical direction. All cases have 4 OpenMP threads. The relative parallel efficiency over this processor count range is approximately 26%. Even for this relatively modest resolution, the two-dimensional hybrid-parallel implementation can exploit parallelism up to 1320 processors.

Figure 8 examines the scalability of CAM on the IBM SP at the 0.5^o x 0.625^o x 26L resolution. We run 2 processes per node with 8 OpenMP threads per process. At 26 levels, we are able to consider only up to 4 subdomains in the vertical direction because of the OpenMP usage in the vertical. Through use of 92 subdomains in latitude, we are able to effectively use 2944 machine processors. The overall throughput is only twice as great as with the finest one-dimensional decomposition, due most likely to limitations of the scaling of the land-surface model.

There is a special extension of CAM designed to study coupled chemistry and climate, known as the Whole Atmosphere Community Climate Model (WACCM). Required conservation of the chemical constituents mandates use of the finite-volume dynamical core. We examine the scalability of CAM on the IBM SP for a 2^o x 2.5^o x 66L WACCM configuration with 51 constituents (see Fig. 9). The larger vertical mesh allows more effective use of the two-dimensional decomposition method, which allows us to realize a four-fold increase in performance as compared to the one-dimensional decomposition.

We next examine performance of the two-dimensional decomposition methodology using MPI-2 one-sided communications on the SGI Origin. Table 2 shows the overall transpose times for the 0.5^o x 0.625^o x 26L resolution using 4 vertical subdomains and 4 threads per process, for MPI-1 with derived types, and MPI-2 methods A and B. We would have liked to include MPI-2 methods C and D in the comparison, but MPI-2 communications with derived data types have not been supported on the Chapman SGI Origin. The performance gains for the transpose are more modest than those for the halo exchange, but show a marked improvement of method B over both method A and the MPI-1 default. The fact that one-sided communication is of less benefit to the transpose calculation is under investigation.

The partial sum optimization of *geopk* mentioned earlier also achieved a notable performance improvement. As indicated in Table 3, the partial sum method (with roundoff error) performs consistently as well or better than the transpose approach. One would expect the partial sum method to be superior particularly for small numbers of vertical subdomains due to the smaller communication stencil.

We have also implemented nested OpenMP constructs in the FV dycore. The motivation for this is that, with the two-dimensional decomposition, one of the decomposition directions (vertical) is the same as the primary OpenMP direction, and, with only 26 vertical levels, the degree of attainable OpenMP parallelism is therefore limited. Nested OpenMP is presently supported on HP/Compaq and IBM platforms, although IBM's present implementation is non-standard and not well publicized. We apply the nested constructs to the vertically independent phase of *cd_core*. The outer loops are with respect to the vertical direction, and the inner loops thread over latitude.

Because there are so many more inner loops than outer loops, nested OpenMP parallelism does incur an overhead. Even so, nesting can pay off for large thread count.

Table 4 considers nested OpenMP for the computational kernel of the vertically independent phase of the FV dycore on the IBM SP. Cases are carried out at 0.5^o x 0.625^o x 26L resolution using 4 latitudinal subdomains and 7 vertical subdomains, so that each subdomain has either 3 or (typically) 4 points in the vertical direction. A total of 8 OpenMP threads are used. Without nesting, all 8 threads are applied to the vertical direction, but with only 4 vertical points per subdomain, half of the threads are wasted. The optimal situation occurs with 4 threads in the vertical direction and 2 in latitude. The resulting throughput is 50% faster than without nesting. When more threads are applied in latitude instead of the vertical, the throughput declines due to the increased overhead of the additional inner loops. This section of code represents roughly 15% of the overall computation; although the throughput with OpenMP nesting increases by 50%, this translates to just a 7% increase overall.

Virtually no improvement is seen with nested OpenMP constructs when the number of latitudinal subdomains is increased to 48. This is due most likely to the much reduced workload per OpenMP region, making the increased overhead of nesting more significant.

5. Conclusions and Future Work

We have presented a scalable parallel implementation of a finite-volume solver of the primitive equations of atmospheric dynamics. This methodology has been fully integrated into the

Community Atmosphere Model. The approach utilizes two different two-dimensional domain decompositions connected by optimized transposes. Communication is accomplished with either ordinary MPI or one-sided MPI-2 constructs; the latter incorporates OpenMP multithreading as well. Several approaches for incorporating MPI-2 with OpenMP for the interprocess communication have been programmed and evaluated. On machines that support this paradigm, such as the SGI Origin, we find significant improvement over the standard MPI-1 approach. Even when run with MPI-1, we demonstrate that CAM can scale to roughly 3000 processors at the half-degree resolution. We also demonstrate the feasibility of nested OpenMP constructs on the IBM, although the net benefit for this particular application is marginal.

One unexplained phenomenon is the fact that MPI-2 with multithreading is not nearly as much of a benefit to the optimized transposes as it is to the border communications. The reasons for this are under investigation. We also were not able to test all four MPI-2 strategies on the SGI Origin 3800, as those involving MPI derived types encountered system errors.

We recently ported the FV dycore to the Cray-X1 machine and have performed the necessary vectorization. The code is not yet running correctly with MPI-2. We will consider extending the SHMEM option in *mod_comm* to support irregular communcations.

Table 1. Comparison of FV dycore timings on the SGI Origin at 1^o x 1.25^o x 26L resolution for various communication paradigms. The one-dimensional decomposition is used, with up to 45 subdomains in latitude and 9 OpenMP threads. There is a clear increase in performance with MPI-2 with increasing thread count.

*DE*s / Threads	MPI-1 Buffers	MPI-1 Types	MPI-2 Method A
9 / 1	626	545	641
9 / 4	193	194	187
9 / 9	105	111	98
18 / 1	316	312	300
18 / 4	112	111	102
18 / 9	77	79	62
36 / 1	159	162	165
36 / 4	82	84	66
36 / 9	64	67	42
45 / 1	153	142	171
45 / 4	75	74	62
45 / 9	63	68	40

Table 2. Comparison of transpose timings on the SGI Origin at 0.5^o x 0.625^o x 26L resolution for various communication paradigms. We use 4 subdomains in the vertical and 4 OpenMP threads, and up to 36 subdomains in latitude. The performance gains for the transpose are more modest than those for the halo exchange, but show a marked improvement of MPI-2 method B over both method A and the MPI-1 default.

N_{lat}	MPI-1 Types	MPI-2 Method A	MPI-2 Method B
9	113	117	99.5
18	68.8	69.5	60.8
36	46.4	47.4	42.4

Table 3. Timings for the geopotential calculation. The partial sum method is as good or better than the transpose method, particularly for a moderate number of subdomains in the vertical direction.

Geopk method	1 vertical subdomain	4 vertical subdomains	7 vertical subdomains
Transpose	**59.7**	**51.2**	**36.6**
Partial Sum	**60**	**30.1**	**30.3**

Table 4. Results using nested OpenMP constructs at 0.5^o x 0.625^o x 26L resolution on the IBM SP for the computational kernel of the vertically independent phase of the FV dycore, with 4 latitudinal subdomains, 7 vertical subdomains, and 8 OpenMP threads. The case with 4 vertical threads and 2 latitudinal threads is optimal.

Vertical threads	Latitudinal threads	Computer time
8	1	200
4	2	136
2	4	181
1	8	261

Figure Captions

Fig. 1. The one-dimensional algorithm decomposes the latitude-longitude-level domain into a set of latitudinal slabs. Each slab, or decomposition element (*DE*), has a north and south halo region, which covers the latitudinal data dependencies. These halo regions are filled (*halo exchange*) before *cd_core* calculations in a communication phase that can be overlapped with unrelated computation. The calculation on haloed arrays can then take place without further communication.

Fig. 2. The two-dimensional domain decomposition formulation requires different decompositions for different phases of the computation. A decomposition in latitude and level (left) is used for dynamics and transport within a control volume, whereas a decomposition in longitude and latitude (right) is used for the surface remapping. The two decompositions are connected by optimized transposes.

Fig. 3. There are four MPI-2 communication schemes. Method A packs data into a contiguous send buffer, uses multithreaded *MPI_Put* for communication into a dedicated target window, and then unpacks from the receive buffer. Method B performs multithreaded *MPI_Put* over contiguous source segments into a dedicated target window and then unpacks from the receive buffer. Method C uses *MPI_Put* with derived source data types into a dedicated target window, with multithreading over the target process, and then unpacks from the receive buffer. Method D uses derived source and target data types to *MPI_Put* directly from the source to the target; with method D one must make sure that the window continually points to the correct target buffer.

Fig. 4. Partial sum methodology for *geopk*. The atmospheric levels (here illustrated in the horizontal) are distributed over *DE*'s. When integrating vertically upward, each *DE* does a

local integration in parallel and passes its result to all *DE*'s above it. Bitwise reproducibility, if desired, can be ensured through use of high precision arithmetic.

Fig. 5. Wall-clock time for the FV dycore at 1^o x 1.25^o x 26L resolution on the SGI Origin as a function of the number of subdomains, using a one-dimensional decomposition. For 4 and 9 threads per *DE* (upper and lower pair of curves, resp.), the MPI-1 (upper curve in pair) and MPI-2 (lower curve in pair) performance are given. MPI-2, which can take advantage of the multithreading in the communications primitives, can yield as much as a 20% overall reduction in computation time for 4 threads, and a 33% reduction for 9 threads.

Fig. 6. The performance of the overall CAM application on both 32 and 2944 processor configurations using IBM *Seaborg* is broken down by component. The dynamical core components *main dynamics* (which excludes *geopotential*), *geopotential*, *tracer advection* and *Lagrangian remapping* all scale better than average. The *land surface model* has the poorest scaling due to insufficient computational load; the other physical parameterizations *post-coupler physics* and *pre-coupler_physics* scale better than average, in part thanks to their communication-free nature. With the targeted optimizations, the *transpose* and dynamics/physics coupling (denoted *dyn/phys coupling*) sections do not present a performance bottleneck.

Fig. 7. Throughput (in simulated days per computing day) of the 1^o x 1.25^o x 26L resolution on the IBM SP with Nighthawk 16-way nodes, as a function of processor count. This illustrates the scalability of the hybrid-parallel approach at modest resolution on a very large machine. Having 1 subdomain in the Z direction (leftmost curve) allows parallelism to be exploited up to 200 processors (about 4 latitude rows per process), with 4 subdomains in Z (center

curve) up to 780 processors, and with 7 subdomains in Z (longest curve) up to 1320 processors. For all tests, 4 OpenMP threads per process were used.

Fig. 8. Throughput (in simulated days per computing day) of the 0.5^o x 0.625^o x 26L resolution on the IBM SP with Nighthawk 16-way nodes, as a function of processor count. The leftmost curve is with 1 subdomain in the Z direction (up to 736 processors), and the longest curve is with 4 subdomains in the Z direction (up to 2944 processors). All cases use 8 OpenMP threads per process.

Fig. 9. Throughput (in simulated days per computing day) of the 2^o x 2.5^o x 66L resolution with 51 constituents on the IBM SP with Nighthawk 16-way nodes, as a function of processor count. This configuration is typical of the Whole Atmosphere Community Climate Model (WACCM). The two-dimensional domain decomposition increases the throughput four-fold as compared to the one-dimensional decomposition.

References

Arakawa, A. and V. Lamb, 1977. Computational design of the basic dynamical processes of the UCLA general circulation model. Meth. Comput. Phys. 17: 173-265.

Collins, W.D., et al., 2004. Description of the NCAR Community Atmosphere Model (CAM 3.0). NCAR Technical Note NCAR/TN-464+STR. Boulder: National Center for Atmospheric Research.

Dagum, L. and R. Menon, 1998. OpenMP: an industry-standard API for shared-memory programming. IEEE Trans. Comput. Sci. Eng. 5(1): 46-55.

Drake, J., I. Foster, J. Michalakes, et al., 1995. Design and performance of a scalable parallel community climate model. Par. Comput. 21 (10): 1571-1591.

Lin, S.-J., 2004. A "vertically Lagrangian" finite-volume dynamical core for global models. Mon. Wea. Rev. 132 (10): 2293-2307.

Lin, S.-J. and R. Rood, 1996. Multidimensional flux form semi-Lagrangian transport schemes. Mon. Wea. Rev. 124: 2046-2070.

Lin, S.-J. and R. Rood, 1997: An explicit flux-form semi-Lagrangian shallow water model on the sphere. Q. J. R. Met. Soc. 123: 2477-2498.

Putman, W.M., S.-J. Lin and B. Shen, 2004. Cross-platform performance of a portable communication module and the NASA finite volume general circulation model. Int'l. Jour. High Performance Comput. Appl. (this issue).

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

Sawyer, W.B., 1999. A multi-level parallel implementation of the Lin-Rood dynamical core. Eighth Workshop on the Solution of PDE's on a Sphere, San Francisco.

Sawyer, W.B. and P. Messmer, 2002. Parallel grid manipulations for general circulation models. Parallel Processing and Applied Mathematics: 4th International Conference, edited by R. Wyrzykowski, et al., Springer-Verlag, 564-571.

Wehner, M.F., A.A. Mirin, P.G. Eltgroth, et al., 1995: Performance of a distributed memory finite difference atmospheric general circulation model. Par. Comput. 21: 1655-1675.

Acknowledgments

We would like to thank Shian-Jiann Lin and William Putman for allowing us to use the OpenMP-parallel FV dynamical core and the *mod_comm* library as a basis for our development.

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48, and NASA's Goddard Space Flight Center, NASA Task # 00-9103-01a.

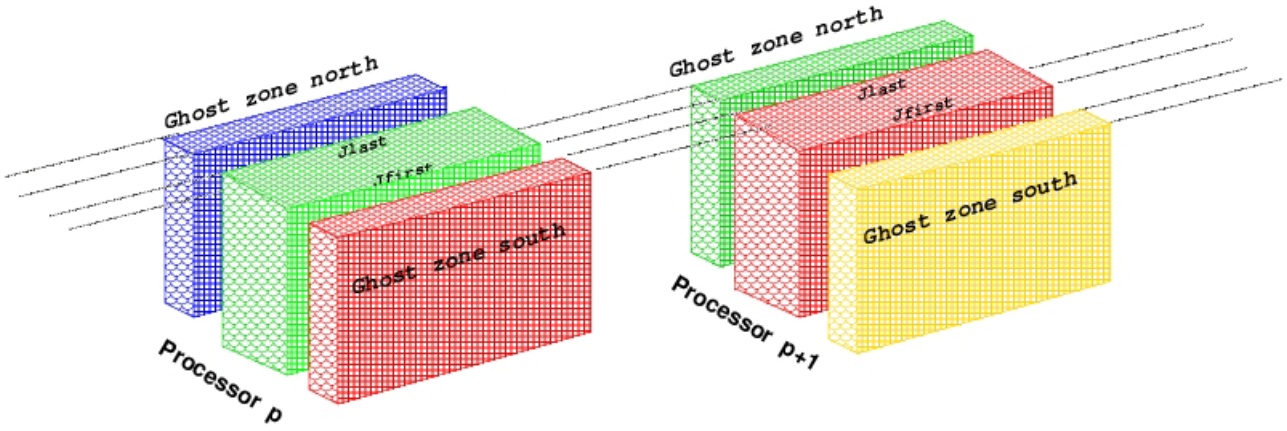

Fig. 1. The one-dimensional algorithm decomposes the latitude-longitude-level domain into a set of latitudinal slabs. Each slab, or decomposition element (*DE*), has a north and south halo region, which covers the latitudinal data dependencies. These halo regions are filled (*halo exchange*) before *cd_core* calculations in a communication phase that can be overlapped with unrelated computation. The calculation on haloed arrays can then take place without further communication.

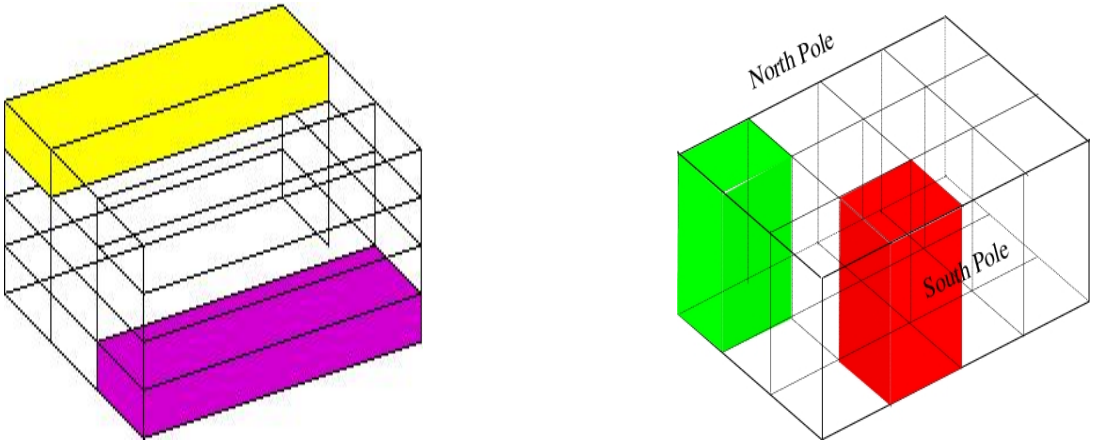

Fig. 2. The two-dimensional domain decomposition formulation requires different decompositions for different phases of the computation. A decomposition in latitude and level (left) is used for dynamics and transport within a control volume, whereas a decomposition in longitude and latitude (right) is used for the surface remapping. The two decompositions are connected by optimized transposes.

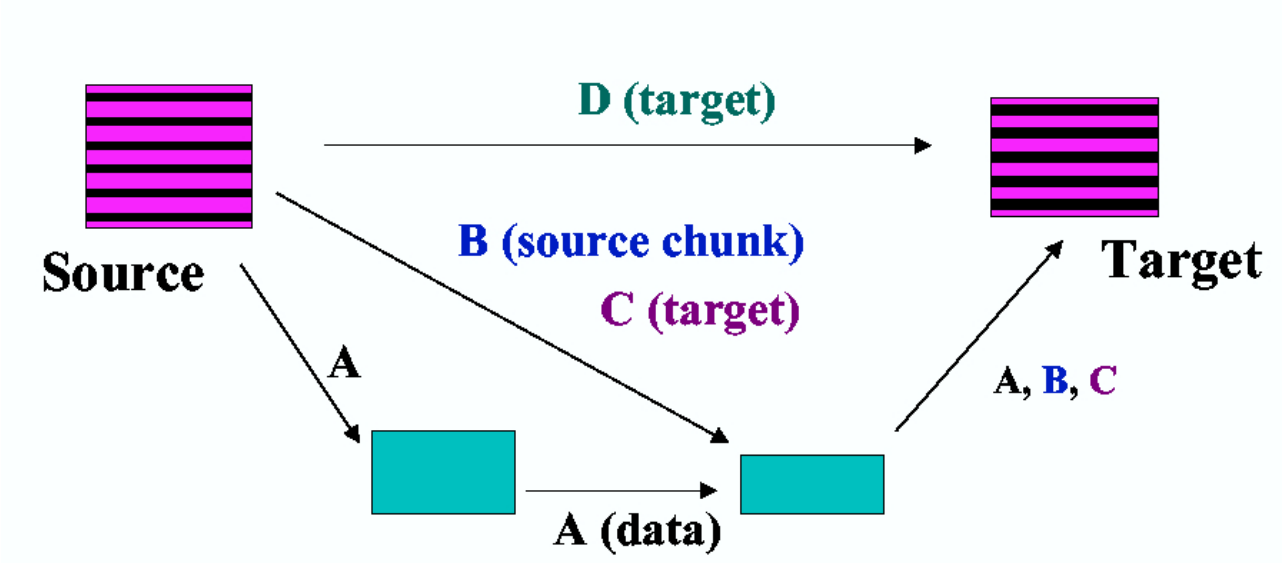

Fig. 3. There are four MPI-2 communication schemes. Method A packs data into a contiguous send buffer, uses multithreaded *MPI_Put* for communication into a dedicated target window, and then unpacks from the receive buffer. Method B performs multithreaded *MPI_Put* over contiguous source segments into a dedicated target window and then unpacks from the receive buffer. Method C uses *MPI_Put* with derived source data types into a dedicated target window, with multithreading over the target process, and then unpacks from the receive buffer. Method D uses derived source and target data types to *MPI_Put* directly from the source to the target; with method D one must make sure that the window continually points to the correct target buffer.

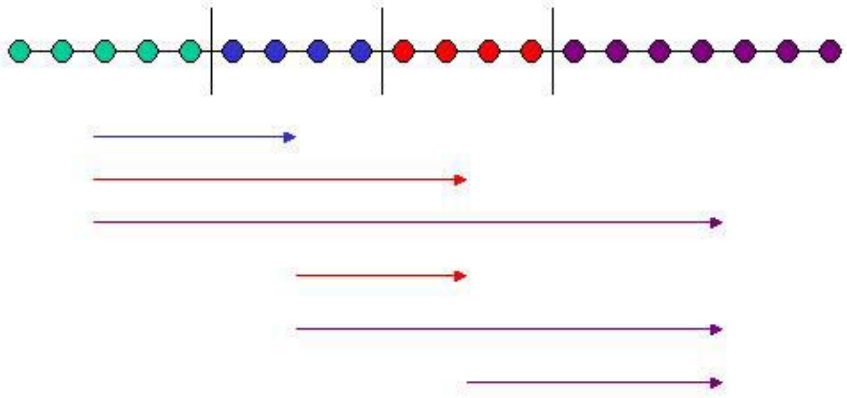

Fig. 4. Partial sum methodology for *geopk*. The atmospheric levels (here illustrated in the horizontal) are distributed over *DE*'s. When integrating vertically upward, each *DE* does a local integration in parallel and passes its result to all *DE*'s above it. Bitwise reproducibility, if desired, can be ensured through use of high precision arithmetic.

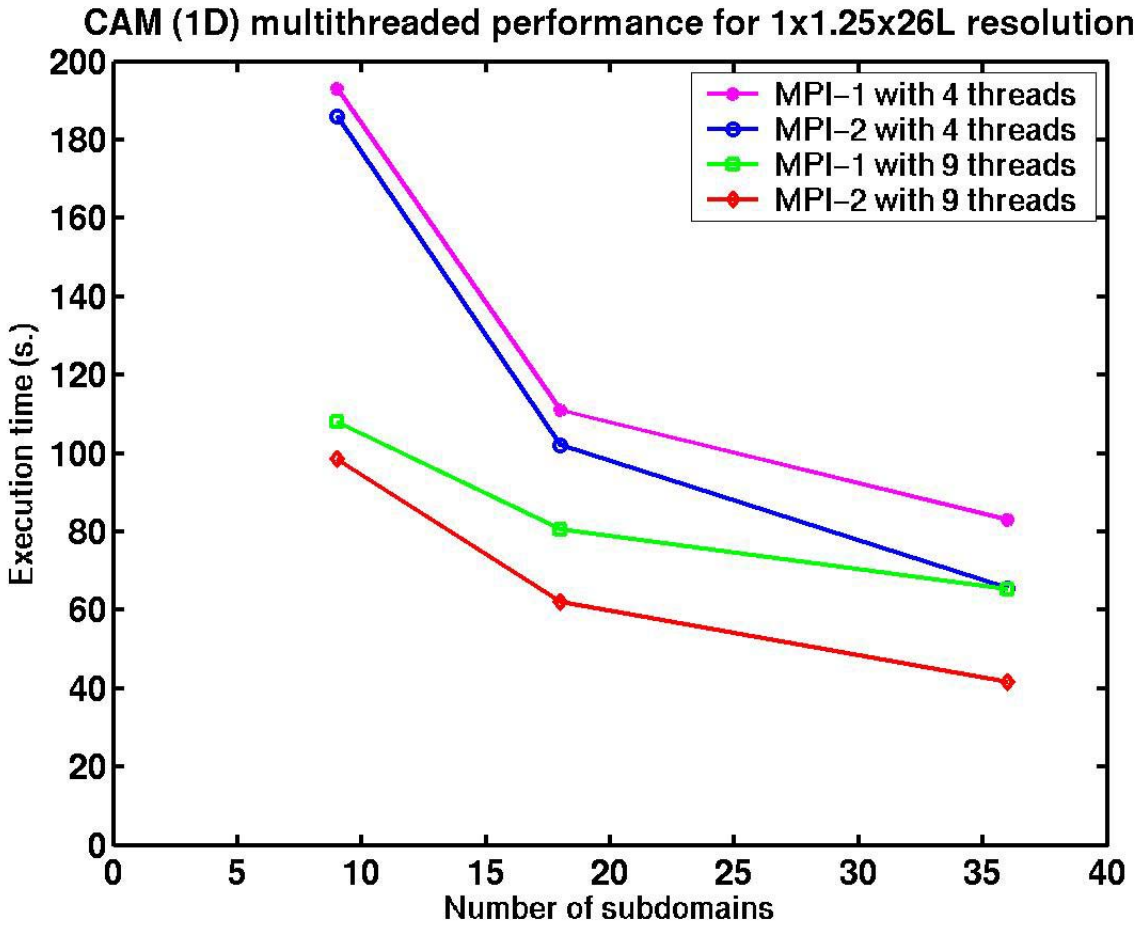

Fig. 5. Wall-clock time for the FV dycore at 1^{o} x 1.25^{o} x 26L resolution on the SGI Origin as a function of the number of subdomains, using a one-dimensional decomposition. For 4 and 9 threads per *DE* (upper and lower pair of curves, resp.), the MPI-1 (upper curve in pair) and MPI-2 (lower curve in pair) performance are given. MPI-2, which can take advantage of the multithreading in the communications primitives, can yield as much as a 20% overall reduction in computation time for 4 threads, and a 33% reduction for 9 threads.

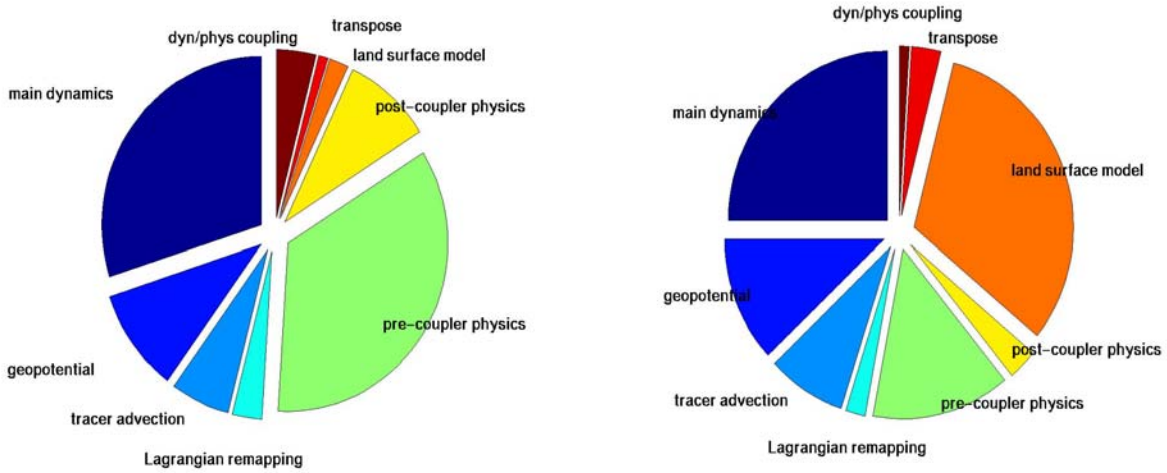

Fig. 6. The performance of the overall CAM application on both 32 and 2944 processor configurations using IBM *Seaborg* is broken down by component. The dynamical core components *main dynamics* (which excludes *geopotential*), *geopotential*, *tracer advection* and *Lagrangian remapping* all scale better than average. The *land surface model* has the poorest scaling due to insufficient computational load; the other physical parameterizations *post-coupler physics* and *pre-coupler_physics* scale better than average, in part thanks to their communication-free nature. With the targeted optimizations, the *transpose* and dynamics/physics coupling (denoted *dyn/phys coupling*) sections do not present a performance bottleneck.

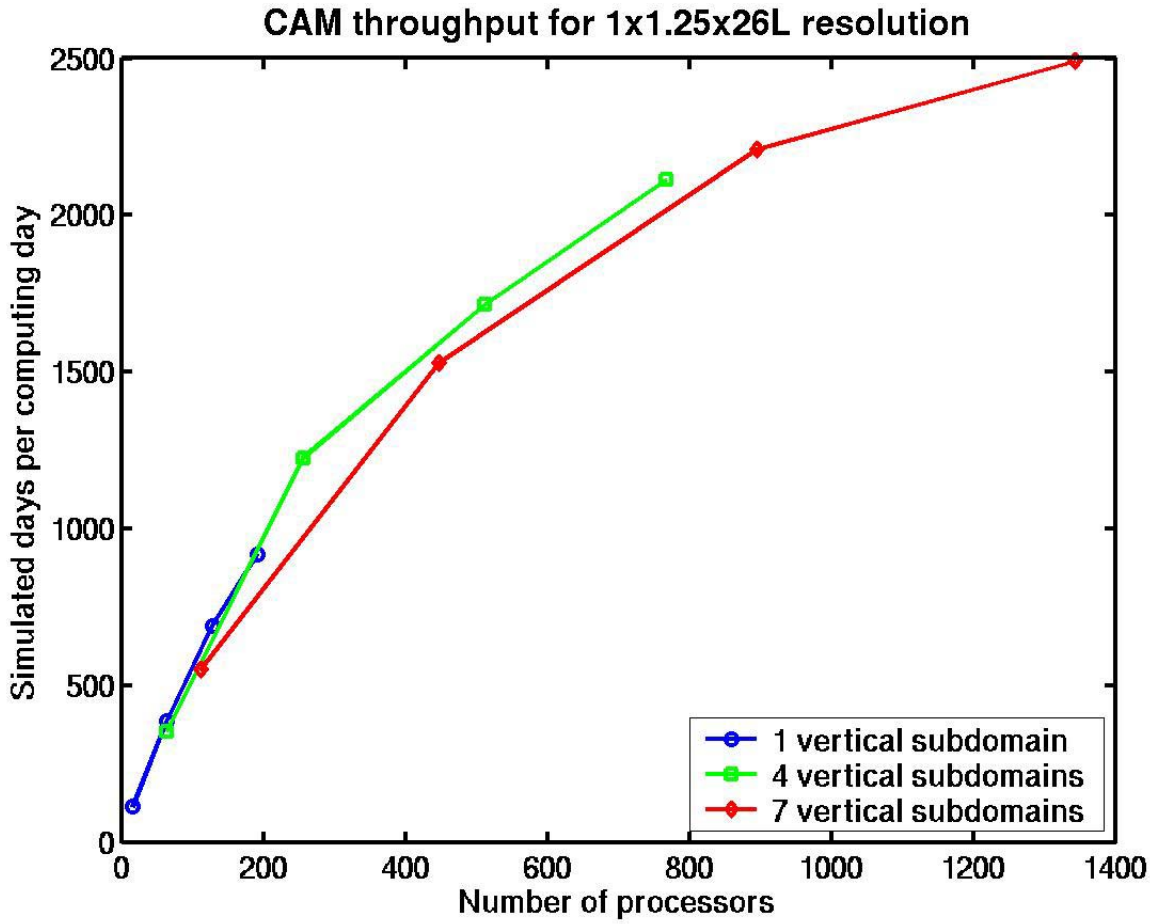

Fig. 7. Throughput (in simulated days per computing day) of the 1^{o} x 1.25^{o} x 26L resolution on the IBM SP with Nighthawk 16-way nodes, as a function of processor count. This illustrates the scalability of the hybrid-parallel approach at modest resolution on a very large machine. Having 1 subdomain in the Z direction (leftmost curve) allows parallelism to be exploited up to 200 processors (about 4 latitude rows per process), with 4 subdomains in Z (center curve) up to 780 processors, and with 7 subdomains in Z (longest curve) up to 1320 processors. For all tests, 4 OpenMP threads per process were used.

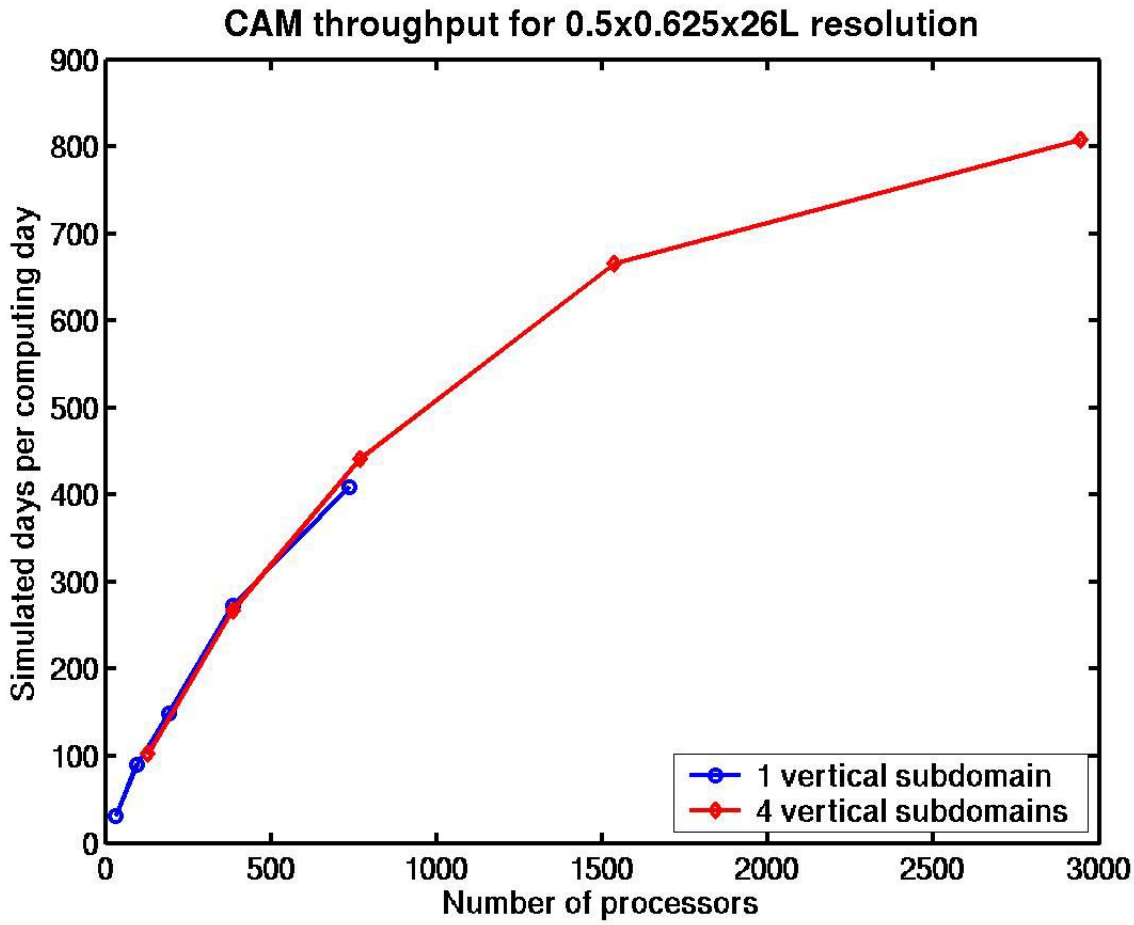

Fig. 8. Throughput (in simulated days per computing day) of the 0.5^{o} x 0.625^{o} x 26L resolution on the IBM SP with Nighthawk 16-way nodes, as a function of processor count. The leftmost curve is with 1 subdomain in the Z direction (up to 736 processors), and the longest curve is with 4 subdomains in the Z direction (up to 2944 processors). All cases use 8 OpenMP threads per process.

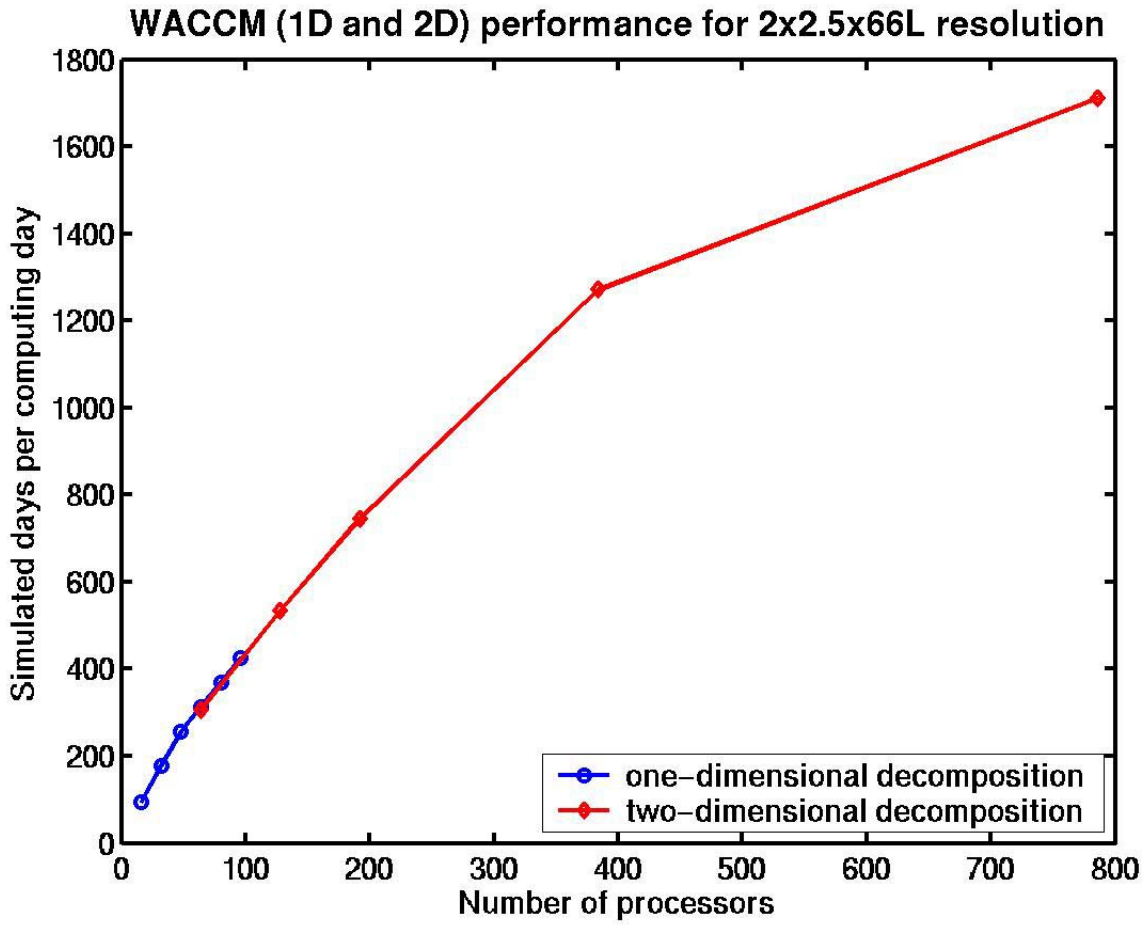

Fig. 9. Throughput (in simulated days per computing day) of the 2^{o} x 2.5^{o} x 66L resolution with 51 constituents on the IBM SP with Nighthawk 16-way nodes, as a function of processor count. This configuration is typical of the Whole Atmosphere Community Climate Model (WACCM). The two-dimensional domain decomposition increases the throughput four-fold as compared to the one-dimensional decomposition.