UCRL-PROC-204955

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

A Scalable Implementation of a Finite-Volume Dynamical Core in the Community Atmosphere Model

W.B. Sawyer, A.A. Mirin

June 28, 2004

IASTED Int'l. Conf. on Parallel and Distributed Computing and Systems (PDCS 2004)
Cambridge, MA, United States
November 9, 2004 through November 11, 2004

A SCALABLE IMPLEMENTATION OF A FINITE-VOLUME DYNAMICAL CORE IN THE COMMUNITY ATMOSPHERE MODEL

William B. Sawyer
Swiss Federal Institute of Technology (ETHZ)
Rämistrasse 101, Zürich, 8092 Switzerland
email: william.sawyer@env.ethz.ch

Arthur A. Mirin
Lawrence Livermore National Laboratory
Livermore, California, 94550 USA
email: mirin@llnl.gov

ABSTRACT
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, it is necessary to employ two separate domain decompositions within the dynamical core. Data must 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.

KEY WORDS numerical weather prediction, parallel programming, MPI-2

1 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 (the *dynamical core*) in the Community Atmosphere Model (CAM). The finite-volume dynamical core, or *FVdycore* for short, requires a substantial fraction of the overall computational time.

The efficient parallelization of FVdycore 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. For the method presented here to be scalable, it is necessary to redistribute the data on a regular basis. In this paper, a variety of techniques using advanced features of the current Message-Passing Interface standard MPI-2 [1], and the OpenMP standard [2] for multithreading, are presented 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 1-D and 2-D decompositions is treated in Sec. 2, with a discussion of the underlying communications primitives in Sec. 3. In Sec. 4 the optimizations for improved performance on large parallel computers are presented. Results are presented in Sec. 5, where it is seen that the approach scales well to large machine configurations. Additional conclusions and future directions are presented in Sec. 6.

2 Parallelization

The *FV dynamical core* solves the 3-D *primitive equations* [3]. FVdycore contains a component (referred to hereafter as `cd_core`) which separates the 3-D equations into n 2-D equations by using a Lagrangian (floating) vertical coordinate, and a remapping component (hereafter `te_map`) which consists of the vertical remapping [4] from the Lagrangian frame back to the original vertical coordinate. Finally, in `geopk`, it also solves the hydrostatic equation at each step, which is inherently a vertical integration. All of these components were first parallelized with the OpenMP shared memory multitasking paradigm, and obtained respectable performance on up to 16-32 CPUs on an SGI Origin 2000 [5].

The FV dynamical core, however, contains considerably more inherent parallelism. In the Lagrangian frame it is clear that the `cd_core` calculation at each level is independent of all others. 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. Within each package it is still possible to employ OpenMP parallelism on the small number of local levels. This approach is employed on clusters of shared memory nodes, such as the IBM Nighthawk nodes which were used for the subsequent performance tests.

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

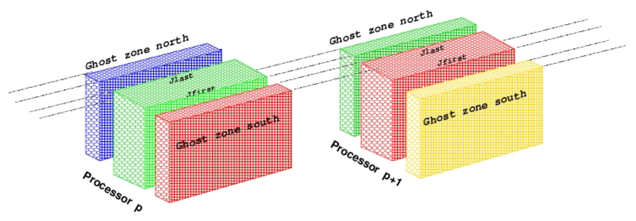


Figure 1. *The 1-D algorithm decomposes the latitude-longitude-level domain into a set of latitude slabs. Each slab, or* decomposition element *(DE), has a north and south halo region, which covers the latitudinal data dependencies. These halo regions are filled ("halo exchange") before* `cd_core` *calculations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

$$C^{\lambda} = \frac{u\Delta t}{R\Delta\lambda\cos\theta} \quad C^{\theta} = \frac{v\Delta t}{R\Delta\theta} \tag{1}$$

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 longitude separation $\Delta\lambda$ is constant, the $\cos\theta$ term goes to zero at the poles. In order to solve the "pole problem" of converging meridians near the pole, a semi-Lagrangian approach [6] is employed to determine the fluxes in λ. C^{λ} can become large, and the semi-Lagrangian method has dependencies on grid points which 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 *1-D* 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 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 with message-passing, and the latitude-slab calculation is then performed independently on all DEs. In the vertical, shared memory parallelism is still utilized, resulting in a hybrid-parallel model. The 1-D domain decomposition is applicable to both `cd_core` and to `te_map`.

The *2-D* algorithm employs a domain decomposition for `cd_core` which is decomposed by 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 p^{κ}) in `geopk`; here the dependencies are vertical instead of horizontal. For these components the domain is best decomposed in latitude and longitude. This requires the constituent arrays to be redistributed or *transposed* from a *latitude-level* to a *latitude-longitude* decomposition before the vertical calculation and back thereafter. We have applied this technique, putting much emphasis on a highly optimized transpose operation.

3 Communication Primitives

At the lowest level, we have based the parallelization on the PILGRIM [7] and the `mod_comm` [8] libraries. The former is designed for general, unstructured domain decompositions while the latter is designed for the types of structured communication, in particular halo exchanges, needed for the 1-D and 2-D algorithms. `Mod_comm` takes advantage of one-sided communication from the Message-Passing Interface MPI-2 standard [1], and combines this with OpenMP multithreading of the communication, which is possible on some architectures, e.g. SGI Origin.

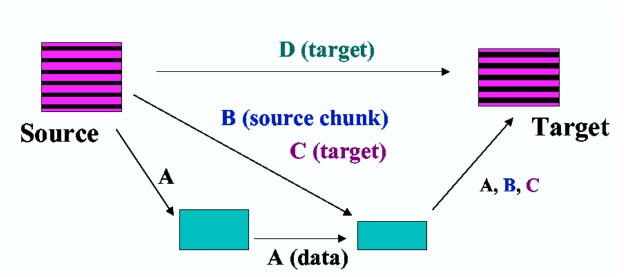


Figure 2. *MPI-2 communication schemes: Method A packs data into a send buffer, uses multithreaded* `MPI_Put` *for communication, then unpacks from a receive buffer; B performs multithreaded* `MPI_Put` *from the source to a dedicated target window, then unpacks into the final destination; C uses* `MPI_Put` *with MPI derived datatypes to move data to a dedicated window, with multi-threading over the target process; D uses* `MPI_Put` *directly into the destination buffer, multithreading over the target process. Method D requires considerable overhead to make sure the window points to the proper target buffer.*

We have added facilities for the types of irregular communication inherent to the transpose, namely exchange of unequally sized data chunks between decomposition elements (DEs). A type to describe a set of non-uniform blocks defined in memory by their displacements and sizes, is as follows:

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

A powerful feature is that the communication `method` can be supplied at run-time, allowing flexibility and inherently supporting different methods for separate communications. 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 a scalar which defines a local offset needed by the MPI-2 implementation.

PILGRIM defines *communication patterns* as two arrays of length 1 . . . # DEs: the *send descriptor*, an array of block descriptors of blocks which need to be sent, and the *receive descriptor*, an array of descriptors of blocks which will 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, D_1 and D_2. The redistribution R is the communication pattern

$$R : A_{D_1} \longrightarrow A_{D_2} \qquad (2)$$

That is, redistributions are a function only of the data decompositions and can be calculated once for all time in the initialization phase. The `mod_comm` library was extended with non-uniform send and receive primitives which take the form:

Function	`mod_comm`	Additions
Begin transfer	`mp_send`	`mp_sendirr`
Complete transfer	`mp_recv`	`mp_recvirr`

The `mp_sendirr` takes both the send and receive descriptor (both needed to support both 1-way and 2-way communication) as arguments while `mp_recvirr` requires only the receive descriptor.

4 Optimizations

The MPI-1 standard [9] supports only two-sided communication. The default MPI-1 implementation uses `MPI_Isend` and `MPI_Irecv` primitives and utilizes both a send buffer into which the data to be transfered are packed, and a receive buffer, from which the data are unpacked. A first optimization required the definition of MPI derived datatypes to define the send and receive descriptors, circumventing the user buffers.

The enhanced MPI-2 standard offers one-sided communication through the `MPI_Put` primitive. One-sided communication requires the definition of MPI-2 *windows* defining the segments of memory that receive remote data. These windows can utilize the MPI derived datatypes described previously.

In addition, the possibility is available on some platforms to multithread the one-sided communication. This can be done in different ways. First, a large block to be sent from a given DE to another is broken into a set of smaller blocks; the delivery of this set with `MPI_Put` is multithreaded. The second possibility is to multithread the delivery of all blocks from one DE with `MPI_Put` over the set of DE targets.

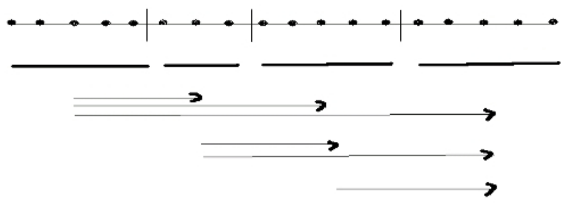


Figure 3. *Partial sum* `geopk`*: The atmospheric levels (here illustrated in the horizontal) are distributed over DEs. When integrating vertically upward, each DE does a local integration in parallel and passes its result to all DEs above it. Bitwise reproducibility, if required, can be ensured by performing local calculations in high precision arithmetic.*

In addition, an extensive effort was made in MPI-2 to reduce the amount of buffering needed. Fig. 2 explains four methods for the MPI-2 implementation of an asynchronous transfer. All methods were supported by IBM SP, but multithreading of messages on that platform does not enhance performance. Only methods A and B (which do not involve MPI derived datatypes) are currently supported on the SGI Origin, which also supports multithreading of messages.

The geopotential calculation `geopk` is a vertical integration within the dynamics calculation taking place at a point where the 2-D domain is decomposed in the latitude and the vertical dimension. The original approach was to transpose the necessary arrays before and after this operation. As an additional optimization, a parallel algorithm was developed which constructs and sends partial sums “upward” as indicated in Fig. 3. This method does not require a transpose — only the communication of the partial sum to all ’higher’ subdomains. The partial sum method gives round-off differences in the results with different DE configurations, due to the varying order of additions. But a quadruple precision mode is available for debugging purposes to ensure bit-wise reproducibility over all possible parallel configurations.

5 Results

The FV dynamical core was tested at both $0.5^o \times 0.625^o$ and $1^o \times 1.25^o$ horizontal resolutions, containing 576 x 361 and 288 x 181 grid points, respectively. 26 vertical levels were used for both resolutions. The target platforms were the SGI Origin 3800 *Chapman* (at NASA GSFC) with 1024 CPUs @ 600 MHz, and an IBM SP *Seaborg* (at DOE

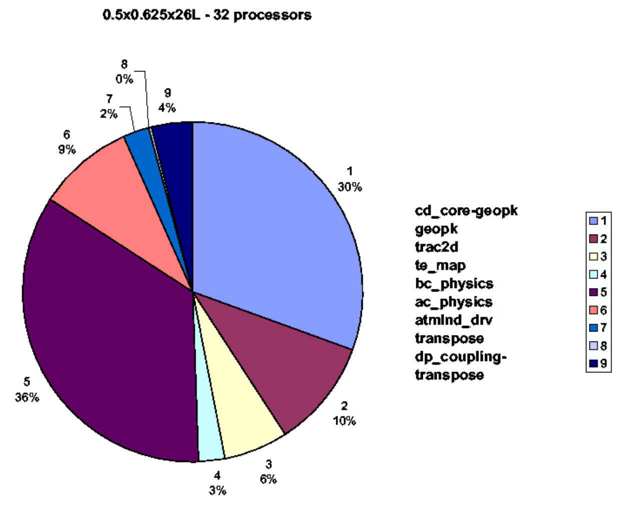
	MPI-1		MPI-2
DEs / Threads	Buffers (s.)	Types (s.)	Method A (s.)
9 / 1	626	545	641
/ 4	193	194	187
/ 9	105	111	98
18 / 1	316	312	300
/ 4	112	111	102
/ 9	77	79	62
36 / 1	159	162	165
4	82	84	66
9	64	67	42
45 / 1	153	142	171
4	75	74	62
9	63	68	40

Table 1. *The FV dynamical core (1-D decomposition) timings are given for a one-day CAM simulation at* $1^o \times 1.25^o \times 26L$*, run on configurations with 9, 18, 36, 45 DEs, each running with 1, 4 or 9 OpenMP threads. The MPI-1 methods using send and receive intermediate buffers or derived datatypes, are compared with MPI-2 Method A. The results indicate overheads for using MPI-2 one-sided communication with 1 thread, but better scalability for 4 and 9 threads than MPI-1. The OpenMP multithread performance of individual computation-only components is not affected by the communication paradigm. The increase in MPI-2 performance is thus attributable to the multithreading in the halo exchange communication.*

NERSC) with 380 Nighthawk nodes, each with 16 CPUs @ 375 MHz.

The 1-D decomposition was extensively evaluated with various numbers of latitude slabs and OpenMP threads per slab, using different communication primitives. Tab. 1 gives an overview of the timing results for the entire FV dynamical core in CAM for MPI-1 with intermediate buffers, MPI-1 with derived datatypes, and MPI-2 method A. The benefits of MPI-2 multithreaded communication are alluded to already in this comparison. Closer investigation of the communication timings indicates excellent speedup in the halo exchange. These results are in line with those found in [8].

Fig. 4 compares the timing percentiles of various components of the Community Atmosphere Model (CAM, part of the Community Climate System Model [10]) in which the FV dynamical core is embedded. The figure indicates that the components scaling the worst and best are part of the physical parameterizations, which are outside of the dynamical core. Some "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. All components of the dynamical core scale reasonably to 2944 CPUs, including



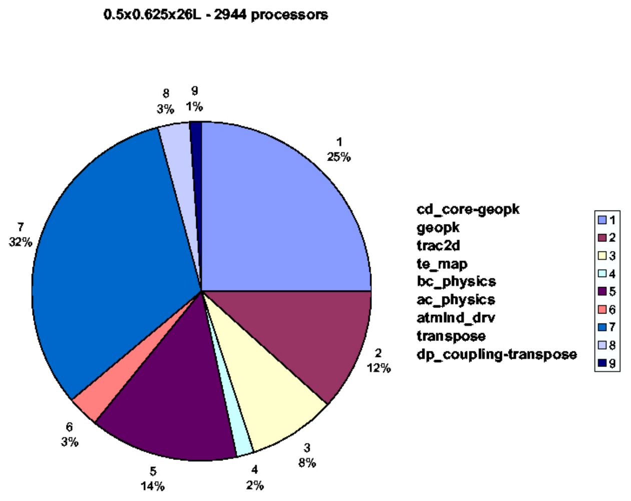


Figure 4. *The performance of the overall CAM application on both a 32 and 2944 CPU configuration (IBM "Seaborg") is broken down by component. The dynamical core components* `cd_core` *(without* `geopk`*),* `geopk`*,* `trac2d`*, and* `te_map` *all scale better than average. The land surface parameterization* `atmlnd_drv` *has the poorest scaling due to insufficient computational load; the other physical parameterizations* `bc_physics` *and* `ac_physics` *scale better than average, in part thanks to their communication-free nature. With the targeted optimizations, the* `transpose` *and* `dp_coupling-transpose` *do not present a performance bottleneck.*

the transpose, which consists entirely of communication. The worst performer is the geopotential calculation, while the best is the `cd_core` routine (without `geopk`).

Fig. 5 illustrates the overall scalability of the CAM run, in simulated days per day of wallclock time. This includes all components illustrated in Fig. 4. Even for

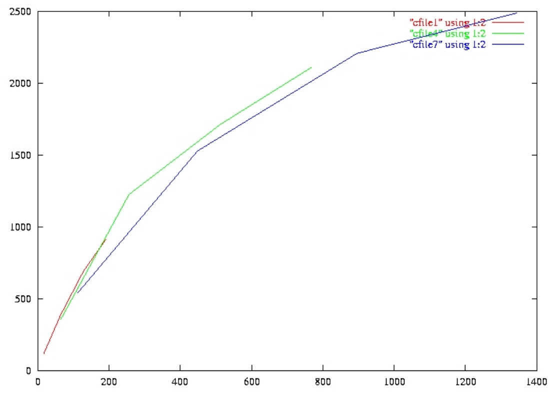


Figure 5. *IBM SP with Nighthawk 16-way nodes: the $1^o \times 1.25^o \times 26L$ resolution illustrates the scalability of the hybrid-parallel approach on a very large machine. Even though the resolution of this simulation can be considered low, the 2-D domain decomposition with 1 thread per DE (leftmost curve) allows parallelism to be exploited up to 200 CPUs, with 4 threads per DE (center curve) up to 780 CPUs, and with 7 threads (longest curve) up to 1320 CPUs.*

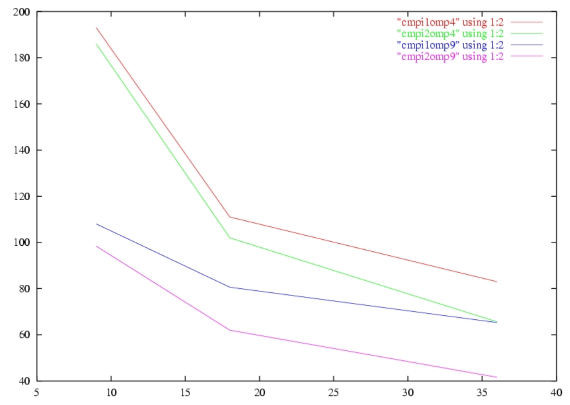


Figure 6. *SGI Origin 3800 results: the wall-clock time for the overall FV dynamical core (1-day simulation at $0.5^o \times 0.625^o \times 26L$ resolution) is given as a function of the number of subdomains (DEs) using a 1-D decomposition. For 4 and 9 threads per DE (upper and lower pair of curves, respectively), the MPI-1 (upper curve in pair) and MPI-2 (lower curve in pair) performances 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.*

the relatively low resolution of $1^o \times 1.25^o \times 26L$ the 2-D hybrid-parallel implementation can exploit parallelism up to a large extent of the machine.

The MPI-2 multithreading capabilities of the code can also provide improved performance if these facilities are supported by the target platform. Fig. 6 illustrates a non-negligible performance increase for the overall FV dynamical core. The performance gains for the transpose (Tab. 2) were more modest than those for the halo exchange, but showed 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 in Sec. 4 also achieved a notable performance improvement. As indicated in Tab. 3, the partial sum method (with round-off error) performs consistently as good or better than the transpose approach.

We have also implemented nested OpenMP constructs in the FV dycore. The motivation for this with the 2-D decomposition is that 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. We find that for certain high-resolution configurations and thread counts, IBM performance of the vertically-independent phase of the dynamical core can improve significantly. Nested OpenMP is presently supported on HP/Compaq and IBM platforms, although IBM's present implementation is non-standard and not well publicized.

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

Table 2. *Timings in seconds are given for the overall transpose times in a 1-day $0.5^o \times 0.625^o \times 26L$ simulation on the SGI Origin 3800 with 4 vertical subdomains and N$_{lat}$ bundles of latitudes (i.e., # DEs = 4 x N$_{lat}$). MPI-2 multithreading can lead to higher performance than the best MPI-1 method: MPI-2 method A is comparable to MPI-1 (using derived datatypes). Method B consistently outperforms both. Methods C and D are not currently supported in the SGI MPI-2 implementation.*

6 Conclusions and Future Work

We have presented a scalable parallel implementation of a *FVdycore* finite-volume solver of the primitive equations. This has been fully integrated into the Community Atmosphere Model. The optimizations presented in this paper utilize two different 2-D spatial decompositions of the domain as well as multithreading primitives on the subdomain local to a node of shared-memory processors. Several techniques incorporating both MPI-2 and OpenMP primitives for efficient redistributions between nodes have been programmed and evaluated. These efforts now allow the

	Threads pe DE		
	1	4	7
Transpose	59.7	51.2	36.6
Partial sum	60.0	30.1	30.3

Table 3. *Timings in seconds for the geopotential calculation in* `geopk`*: the partial sum method is as good or better than the transpose method, particularly if multiple threads per DE are employed.*

FVdycore to scale to large machine configurations, even for simulations with relatively modest resolution.

Some additional optimizations to this implementation are planned. We hope to port the code to the Cray X1, utilizing both vector parallelism and the SHMEM library for communication. Our primitives for irregular communication will thus be extended to use SHMEM as an option to MPI-2.

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.

This is LLNL Report UCRL-PROC-204955.

References

[1] MPI Forum. MPI-2: Extensions to the Message-Passing Interface. MPI Forum, 1996.

[2] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Transactions on Computational Science and Engineering*, 5(1), 1998.

[3] Eugenia Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, London, 2003.

[4] Shian-Jiann Lin. A 'Vertically Lagrangian' Finite-Volume Dynamical Core for Global Models. *Monthly Weather Review*, 2004. Accepted for publication.

[5] William Sawyer. A Multi-level Parallel Implementation of the Lin-Rood Dynamical Core. Presentation at *8th Workshop on the Solution of Partial Differential Equations on the Sphere*, 1999.

[6] Shian-Jiann Lin and Richard B. Rood. Multidimensional Flux-Form Semi-Lagrangian Transport Schemes. *Monthly Weather Review*, 124(9):2046–2069, September 1996.

[7] W. Sawyer and P. Messmer. Parallel Grid Manipulations for General Circulation Models. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics: 4th International Conference*, pages 564–571. Springer-Verlag, 2002. LNCS 2328.

[8] William M. Putman, Shian-Jiann Lin, and Bowen Shen. Cross-Platform Performance of a Portable Communication Module and the NASA Finite Volume General Circulation Model. Submitted to *Parallel Computing*, 2004.

[9] MPI Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, Univ. of Tennessee, 1994.

[10] 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. Hurrell. The Community Climate System Model. *BAMS*, 82(11):2357–2376, 2001.