LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-412240

Improving the Performance Scalability of the Community Atmosphere Model

A. A. Mirin, P. H. Worley

April 16, 2009

Supercomputing 2009
Portland, OR, United States
November 14, 2009 through November 20, 2009

Disclaimer

Improving the Performance Scalability of the Community Atmosphere Model

A. A. Mirin *

Lawrence Livermore National Laboratory

P. H. Worley †

Oak Ridge National Laboratory

April 11, 2009

Abstract

The Community Atmosphere Model (CAM), which serves as the atmosphere component of the Community Climate System Model (CCSM), is currently the most computationally expensive CCSM component in typical configurations. Improving performance scalability in CAM has been a challenge, due largely to algorithmic restrictions necessitated by the polar singularities in its latitude-longitude computational grid. Nevertheless, through a combination of exploiting additional parallelism, implementing improved communication protocols, and eliminating scalability bottlenecks, we have been able to more than double the maximum throughput of CAM on production platforms. We describe these improvements and present results on the Cray XT4/XT5, IBM BG/P, and an Opteron/Infiniband cluster. This improved performance will enable the CCSM research community to use its computing resources more effectively, allowing additional and more computationally expensive experiments to be run in support of the upcoming Intergovernmental Panel on Climate Change (IPCC) fifth assessment.

1 Introduction

The Community Climate System Model [3, 1] is one of the world's leading global climate models. It was an important contributor to the Fourth Assessment Report of the Intergovernmental Panel on Climate

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550 (mirin@llnl.gov)

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

Change [15] and is expected to play an important role in the upcoming fifth assessment. CCSM contains several model components interconnected through a coupler: the Community Atmosphere Model (CAM), the Community Land Model (CLM), the Parallel Ocean Program (POP), and the Community Ice Code (CICE). Contributions to the fourth assessment were typically run with computational grids at an atmosphere/land resolution of 1.4 degrees and an ocean/sea-ice resolution of 1 degree.

CCSM has been undergoing rapid improvement, both in the breadth of its science and as a computational science tool. Through the improved representation of atmospheric aerosols, ocean biogeochemistry and the associated emissions, and land biogeochemistry within a dynamic vegetation model, CCSM is evolving from a climate system model into an earth system model. There is the desire as well to increase both the horizontal and vertical resolution of the grids used in climate simulations. With additional process representation and increased resolution comes increased cost - up to orders of magnitude or more, depending on scenario. The challenge is to improve the computational science capability of CCSM so that by taking advantage of increased capabilities of the evolving computational platforms, the same level of throughput (roughly 5 simulated years per computing day) can be maintained. With increases in raw processor speed approaching technical limits, we look toward being able to utilize a much greater number of computational processors - up to hundreds of thousands.

The least scalable model component of the Community Climate System Model is the atmosphere [5]. Reasons for this will be discussed below, but a contributing factor is the polar singularities of the traditional latitude-longitude grid - whereas the ocean and sea-ice models use modified grids whose poles are within the land areas and hence not part of the computational domain. New numerical methods based on grids that do not suffer from the scalability disadvantages of the latitude-longitude are under consideration for the atmosphere[10, 12], but these will not be available for use in the fifth assessment mentioned above. In consequence, it is vital that the performance scalability of the current version of CAM be improved as much as possible.

Our focus is on the performance and scalability of the Community Atmosphere Model (CAM), which when run in stand-alone mode includes the land component CLM. In section 2 we give an overview of CAM and discuss CAM's parallelization approach and how scalability has been limited heretofore. In section 3 we describe the versions of CAM, benchmark configurations, and computing platforms used in this work. In sections 4, 5, and 6 we present our recent improvements to performance and scalability of CAM. Empirical studies documenting the impact of these modifications are described in section 7. Section 8 provides additional discussion and looks toward the future.

2 Community Atmosphere Model

The Community Atmosphere Model (CAM) [2] has been developed at the National Center for Atmospheric Research (NCAR), with contributions from external National Science Foundation (NSF), Department of Energy (DOE), and National Aeronautics and Space Administration (NASA) funded researchers. CAM is characterized by two computational phases: the dynamics, which advances the evolutionary 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. The dynamics assumes the hydrostatic approximation, which allows a partial decoupling into a two-dimensional horizontal portion and a one-dimensional vertical (columnar) portion. The approximations in the physics are referred to as the physical parameterizations and are columnar in nature. Control moves between the dynamics and the physics during each model simulation timestep.

CAM includes multiple options for the dynamics, referred to as dynamical cores or dycores, one of which is selected at compile-time. Three dycores are currently supported, and several additional dycores are currently undergoing development and testing and may be included in the future. In this study, we discuss only the dycore that will be used in the fifth assessment: a finite-volume flux-form semi-Lagrangian (FV) dynamical core that uses a tensor product latitude $\times$ longitude $\times$ vertical-level grid over the sphere. An explicit interface exists between the dynamics and the physics, and the physics data structures and parallelization strategies are independent from those in the dynamics. A dynamics-physics coupler moves data between data structures representing the dynamics state and the physics state.

The finite-volume dycore [8] was constructed originally by Lin and Rood when at NASA Goddard Space Flight Center in the late 1990s. A Lagrangian vertical coordinate is used to define flux volumes, within which the horizontal dynamics evolve. Vertical transport is modeled through evolution of the geopotential along each vertical column. A conservative Lagrangian surface remap is performed each model time step. A flux form semi-Lagrangian approach to the horizontal dynamics overcomes the stringent Courant stability condition in the neighborhood of the polar singularities, with polar filtering limited to a handful of intermediate variables; no prognostic variables are filtered.

The approach to parallelization is domain decomposition, where each subdomain is assigned to a single MPI [6] process; when available, OpenMP [4] is used for additional parallelization. The dynamics and physics each use separate decompositions. The physics utilizes a fine-grain 2-D latitude-longitude decomposition. Each subdomain, referred to as a chunk, is a collection of vertical columns [17]. Chunk sizes are chosen to optimize cache utilization, or in the case of a vector machine, vector length. While there are no

inherent restrictions on how the vertical columns are partitioned, typical decompositions either minimize communication costs (when transposing to/from the dynamics decomposition) or load imbalance (typically by combining daytime and nighttime regions).

The dynamics utilizes multiple block decompositions. The finite-volume dycore uses a latitude-vertical decomposition for the main dynamics and a latitude-longitude decomposition for the Lagrangian surface remapping and (optionally) geopotential calculation [11].

The various decompositions are connected by transpose routines. In cases where the physics decomposition is not equivalent to the connecting dynamics decomposition, the transposes are accomplished by either collective MPI routines or point-to-point communications, depending on which performs best on the given architecture. The transposes connecting the dynamics decompositions for the finite-volume dycore utilize the Pilgrim and Mod_Comm libraries [13], which were originally based on non-blocking point-to-point communications.

There are a number of limitations to scalability. Climate models are notorious for their coarse meshes, a requirement derived from the need to integrate out to hundreds of simulated years. Present day simulations with CAM and the finite-volume dycore typically use a latitude/longitude/vertical computational grid of size $96 \times 144 \times 26$; this amounts to roughly 360 thousand grid points, which is a very modest number for a 3-D grid. The most ambitious horizontal grid size currently under consideration for upcoming studies, including the fifth assessment, would represent an eight-fold refinement in each horizontal direction, or 768×1152; the number of levels might increase to 30, and, for whole atmosphere chemistry scenarios, to 66. Since the dynamics domain decomposition is grid-based, the relatively modest grid size limits the size of the domain decomposition, hence the number of MPI tasks. The implementation of the finite-volume dynamics requires subdomains to have at least 3 points in latitude, 3 points in longitude, and (until recently) 3 in the vertical; with only 26 levels in the vertical, this has limited the size of the vertical decomposition to 8 subdomains. With the requirement (until recently) that all phases of the calculation utilize the same-sized decomposition, the limitations on dynamics scalability has limited scalability of the overall code.

The left graph in Figure 1 shows throughput on a 1.9×2.5 degree horizontal grid (96×144) for a sequence of ever more expensive physical process options under consideration for use in the fifth assessment. Data were collected in March 2007 on a Cray XT4 with quad-core Opteron processors using CAM development version 3.5.27, and both MPI and OpenMP parallelism were used. The figure shows both the maximum throughput rate and the maximum processor count that CAM was capable of before implementation of the optimizations described in this paper. Note that the new physical processes increase the cost of the model

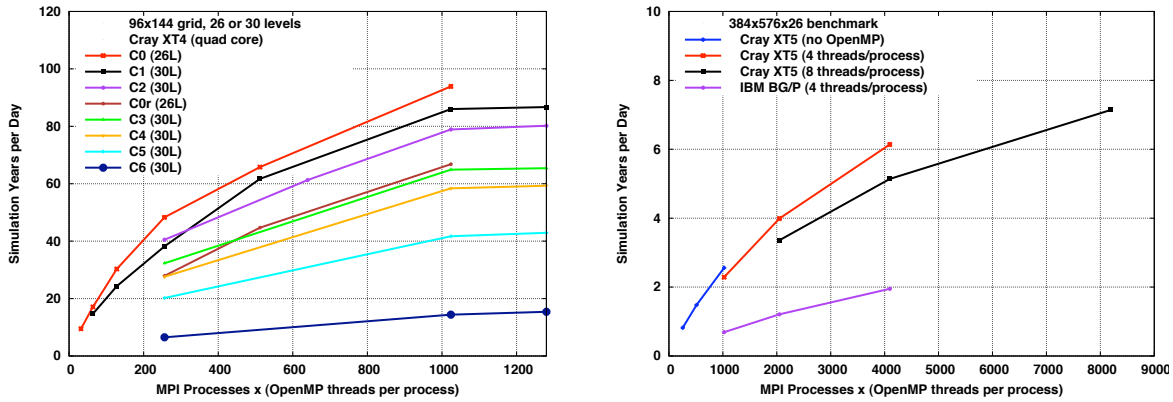

Figure 1: Throughput of the Community Atmosphere Model on a 96×144 grid with a sequence of ever more expensive physical process options, and on a 384×576 grid with the current default physical process options.

signficantly, and throughput rate may be an issue even for this very coarse grid resolution.

The right graph in Figure 1 shows throughput and maximum parallelism on a 0.47×0.63 degree horizontal grid (384×576) with 26 vertical levels, using the current default physical processes. Again, these results demonstrate the capability of the code prior to the performance improvements reported herein. Data were collected on a Cray XT5 with dual quad-core Opteron SMP nodes and on an IBM BG/P, both using CAM development version 3.6.27. Data are presented separately for pure MPI, 4-way OpenMP parallelism, and 8-way OpenMP parallelism, to demonstrate that OpenMP parallelism helps improve scalability, but has its limits for this code. The memory requirements for this problem size were too large to run on the BG/P in pure MPI mode. Some of our optimizations were already implemented in version 3.6.27, so we used a modified version with these optimizations removed.

The throughput demonstrated in Figure 1 represents a marked improvement over what was possible in even earlier versions of CAM [11, 17]. However, limitations on performance scalability are still evident. In particular, the number of MPI processes for this high-resolution (for climate) 0.47×0.63 degree grid has been limited to roughly 1000, and these limitations cannot be addressed simply by moving to ever larger OpenMP parallelism. Fortunately, we have been able to overcome several of these limitations and extend scalability and capability, as described in Sections 4, 5, and 6.

3 Experiment Particulars

The primary development platforms and target architectures for this work have been a series of Cray XT systems at Oak Ridge National Laboratory (ORNL), IBM BG/P systems at ORNL and at Argonne National Laboratory (ANL), and an Infiniband-connected cluster of 8-way (4 socket, dual-core Opteron) nodes at Lawrence Livermore National Laboratory (LLNL). This latter system is called Atlas. While not an issue on these platforms currently, hybrid MPI/OpenMP parallelism has been problematic in the past. In consequence, optimizing for MPI-only parallelism has been retained as a goal during this work. For large MPI process counts there is also limited OpenMP parallelism to exploit, and efficient MPI-based parallel algorithms and efficient MPI communication protocols are likewise important on the current target systems.

As we are attempting to address performance of CAM as it is to be used in the fifth assessment, we have used internal development versions. Most of the data presented here is based on version 3.6.27, tagged on Feb. 2, 2009. Two modified versions are used, one without the optimizations described in the paper (`previous`), and one with a few optimizations that were added to CAM in versions after 3.6.27 (`current`). CAM continues to be a moving target, and the physical processes that will be used in production with the next release are still to be decided. These choices will impact the quantitative aspects of the results described, but the performance characteristics should remain similar qualitatively.

For simplicity of presentation, the majority of the performance data presented here will be for the 0.47×0.63 degree resolution horizontal grid with 26 vertical levels. Other resolutions, both coarser and finer, have been and are being used during development. The CCSM community intends to include some 0.47×0.63 degree resolution configurations in its submission to the fifth assessment, so this resolution is particularly relevant. The 0.47×0.63 degree resolution is also expensive enough that efficient parallel performance is critical.

Finally, the data described in this paper were from runs of CAM with the output of 2-D and 3-D fields disabled. Most I/O in CAM goes through a single process currently. The cost of I/O from/to disk is approximately constant as the process count increases, and the cost of the associated interprocessor communication between the I/O process and the other processes increases with process count. While not significant at small to medium process counts, this I/O overhead would mask some of the impact of the modifications described here at large process counts. By disabling the I/O, we are also able to use shorter benchmark runs without having to adjust the timings to take into account what would otherwise be incorrect I/O frequencies. The CCSM developer community is actively working on a parallel I/O layer that is expected to reduce the I/O overhead at scale significantly.

horizontal resolution	longitude	latitude	vertical levels	maximum MPI process count
1.9×2.5	144	96	26	256
1.9×2.5	144	96	30	320
0.94×1.25	288	192	26	512
0.94×1.25	288	192	30	640
0.47×0.63	576	384	26	1024
0.47×0.63	576	384	30	1280
0.23×0.31	1152	768	26	2048
0.23×0.31	1152	768	30	2560

Table 1: Maximum MPI parallelism available in CAM previously

4 Improving Algorithmic Scalability

As indicated in section 2, the parallel scalability of CAM has been limited at the algorithmic level in a number of important ways. Table 1 lists the maximum number of MPI processes that could be used for a range of problem resolutions, including the current production resolution of a 1.9×2.5 degree horizontal grid with 26 vertical levels and the largest problem resolution currently under consideration for the fifth assessment.

In contrast, the physics can, algorithmically, support up to longitude $\times$ latitude parallel threads of execution. For the largest problem resolution in Table 1, this is 884,736, or over 300 times larger than what is available in CAM as a whole. OpenMP parallelism can be used to increase the parallelism throughout the code for systems on which it is available. In our experience, this is not a complete solution, nor is it available on all platforms of interest.

The basic approach that we have taken to exploit additional parallelism is based on the concept of inactive and auxiliary processes. Recall the previous limitation that each phase of the code invoke the same number of MPI tasks. That limitation has been relaxed. Processes may be inactive during either the physics phase, the dynamics phase, or both. In the case of the physics, those processes are merely assigned zero chunks. The ability of Fortran 90 compilers to support empty loops and the existence and allocation of zero-sized arrays has been crucial in keeping the source code simple. Support for inactive processes during the dynamics phase has required conditional statements, but fortunately only a limited number. The support for inactive processes then opens the door for those processes to be used as auxiliary processes, to be activated for specific phases of the calculation. The ability to invoke auxiliary processes is crucial to some of the scalability extensions described below.

One of the simplest extensions to scalability has been removal of the constraint that subdomains contain at least 3 vertical levels for the finite-volume dynamics. This required that only several lines of code be

modified. Another extension has been to allow using more processes in the physics than in the dynamics. For purposes of minimizing physics load imbalance, CAM already had the capability to support any physics decomposition provided its size was the same as that for the dynamics. Given the added capability for inactive processes, supporting more physics than dynamics processes required, for the most part, only generalizing the dynamics-physics transpose to allow the domain and range to have different numbers of processes.

Recall that the finite-volume dynamics uses two domain decompositions: a latitude-vertical decomposition for the main dynamics and a latitude-longitude decomposition for the Lagrangian surface remapping and (optionally) geopotential calculation. Those parts of the algorithm that utilize the latitude-longitude decomposition are columnar; hence they support a much greater number of subdomains longitudinally than the vertical decomposition supports vertically. We have added the ability to have a larger latitude-longitude than latitude-vertical decomposition. This required generalizing the transposes that connect the two dynamics decompositions to support unequal-sized decompositions as well as adding conditional execution to certain code sections.

Increasing the number of subdomains (and the MPI parallelism) in the latitude-longitude decomposition also increases the amount of parallelism in the physics. For certain problems and computer systems, this improves performance over that of using additional MPI processes only in the physics. In other instances it can be slower. This latter situation is due to an increase in communication costs that outweighs the improved throughput during the latitude-longitude decomposition execution phase. Since load balancing is implemented during the communication between the physics and the dynamics, the additional communication cost in the transposes between the latitude-longitude and latitude-vertical decompositions does not necessarily decrease the overhead of communication in the transposes between the dynamics and the physics.

The final parallelism enhancement is relevant for finite-volume dynamics with large numbers of advected quantities, such as when simulating chemistry scenarios. We refer to these advected quantities as *tracers*. Chemistry scenarios will be important for the fifth assessment, and recent versions of CAM have a chemistry package with 25 tracers enabled by default. Without a chemistry package, CAM requires only 3 tracers. Full tropospheric chemistry requires use of 103 or more tracers.

Each tracer is advected within the dynamics each timestep. While partially a function of the chemistry package, our experience has been that each chemistry package tracer adds between 2% and 2.5% to the overall run time. Hence as few as 40 tracers will double the overall run time compared to running without a chemistry package. Approximately one-third of this cost increase is due solely to tracer advection. The next largest portions are in the dynamics-physics coupling and the physics itself.

We have added the capability to decompose the tracer advection with respect to tracer index - that is, to advect multiple tracers simultaneously. We decompose the tracer population into T groups. We define T-1 auxiliary latitude-vertical decompositions; this requires (T-1)*M auxiliary processes, where M is the size of the latitude-vertical decomposition. In other words, we hold in reserve an additional set of (T-1)*M inactive processes. These processes are brought in as auxiliary processes during the tracer advection phase. The tracer advection requires Courant numbers, mass fluxes, pressure thicknesses, and tracer values. These quantities are communicated to the auxiliary processes using nonblocking communications, overlapping computation to the extent possible.

To summarize, additional parallelism has been exposed by decoupling the parallel algorithms in the different phases of the code (the physics, and the latitude-vertical and longitude-latitude decompositions in the dynamics). In the cases where processes are idle during the latitude-vertical decomposition, these can be used to decompose over the tracer index, introducing another direction of domain decomposition. None of these modifications change the asymptotic nature of the parallel algorithms, but, as will be shown, they do lead to practical improvements in performance. Before describing the performance improvements, two other categories of performance optimizations must be described. Exploiting additional parallelism is not productive unless the communication overhead can be controlled. Similarly, unscalable algorithms that can be ignored for small process counts will prevent the efficient exploitation of large numbers of processes.

5 Communication Optimizations

The MPI communication protocols used in the physics/dynamics transposes [17] and in the Pilgrim and Mod_Comm libraries [13] have been optimized over a number of years and proven adequate up until now. However, MPI communication at increased scale, especially when between phases with significantly different numbers of active processes, sometimes performs poorly, and sometimes fails, on our target systems. In response to this, we re-examined the existing algorithms, reworking some of the implementations and adding new optimization options to better support the wide variety of problem and machine configurations for which CAM will be used.

MPI collectives. With one exception, the transposes between the different decompositions are not full all-to-all communication patterns. Rather, each MPI process sends to a subset of the processes and receives from a (usually different) subset of processes. On some systems and for some of these transposes, an implementation of this communication pattern using MPI point-to-point commands achieves better performance

than one calling MPI_Alltoallv. Conversely, MPI_alltoallv is the most efficient on other systems or for other transposes. The important issue here is that the performance difference between the point-to-point and MPI_Alltoallv implementations can be very large, and that one choice is not suitable in all situations. To address this, we systematically added support for both point-to-point and MPI collective implementations to each of these transposes, and the choice is made at runtime. We have, however, determined and implemented reasonable defaults based on our experiences.

As will be explained below, we also support both point-to-point and MPI collective implementations of the gather and allgather operators.

Combining. The original implementation of communication in the FV dynamics utilized nonblocking point-to-point commands and attempted to overlap communication with computation. Overlap has proven difficult to achieve on many systems, especially at scale when the amount of intervening work is small and the communication costs are large. In these situations, combining communication requests, in order to minimize the number of communication requests and to possibly improve the efficiency of underlaying memory copies, can be more effective. We have added options to combine communication requests where possible, in particular when advecting tracers. The amount of combining to use is a runtime parameter and reflects a tradeoff between latency minimization and memory usage.

Flow control. The original implementation of communication in the FV dynamics preposted all receive requests, issued all (nonblocking) send requests, then waited for the receive requests to be satisfied. At scale, this has the potential of overwhelming any given process with messages for which it has not yet posted receive requests. This can cause failures if the system can not allocate sufficient system buffer space to handle all of the requests, and will degrade performance in any case with all of the additional buffer copying. With the introduction of different numbers of active processes in different phases of the code, the likelihood of this situation to occur has increased significantly. It also is a common problem with the gather collective, and can even affect performance and robustness of MPI_Gather for some vendor implementations.

To address this, we added support for handshaking messages. These are used to eliminate all unexpected messages of size greater than zero. After each nonblocking receive is posted, a zero-byte message is sent to the source process. Upon receipt of this signal, the source process can send the message (using MPI_Rsend or MPI_Irsend, since it is now safe to use the *ready* variant of the MPI send command). This has proven more efficient than using MPI_Sendrecv or the synchronous variants of the MPI send commands. Again, the use of handshaking is enabled at runtime, and can be enabled for use in specific transposes or gathers.

There is still a potential problem in preposting more nonblocking receive requests than are supported on a given system (with any given MPI environment variable settings). There may also be a performance impact from having a large number posted, if only in the cost of matching receive requests with the incoming messages. There is also a potential problem of overwhelming a given process with the zero-byte handshaking messages. To address these issues, we have also added a *throttle* parameter, specifying the maximum number of outstanding send and receive requests to allow. Again, this is a runtime parameter, and can be set separately for specific transposes, gathers, and scatters.

For throttling to work, we also implemented a dimensional-exchange ordering of the messages. With this, in a total ordering of possible sends and receives between processes, process i and process j will exchange data at the same step. This ordering does a reasonable job of minimizing contention and hotspots, and is now used for all point-to-point implementations of collective communication within CAM, replacing the original ordering used in the FV dynamics.

The options described so far allow us to address problems within a single collective operation. Problems can also arise from communication demands of a series of collective requests, for example a series of scatter requests, even though any single collective request may not cause a problem. Moreover, invoking a handshaking protocol in a scatter may replace one problem with another. We have found that replacing the nonblocking send requests with blocking sends in the point-to-point implementations can slow down the rate at which message requests are generated, slow enough to avoid problems arising from multiple collective calls. The choice of nonblocking (the default) and blocking sends in the point-to-point implementation of message-passing algorithms CAM is yet another runtime parameter that can be set separately for specific collectives.

Summary. Common, and efficient, communication protocols for the transposes, gathers, and tracer advection communication in CAM can break down at scale, by either performing poorly or failing. The communication logic has been re-engineered to add new options that can be used to avoid these problems. Most of the basic communication algorithms were imported from existing code used in the physics/dynamics tranposes [17], but all were re-implemented and optimized to take into account the specific issues being addressed.

6 Performance Scalability Bottlenecks

In the process of increasing the number of MPI processes that CAM can use, a number of performance bottlenecks were identified (and eliminated). These were typically code fragments with complexity $O(N)$ or $O(N^2)$, where N is the size of the computational grid, $O(P)$, where P is the number of MPI processes, or $O(NP)$. Most of these qualify as "performance bugs" in that much new code is being written at the moment, and the performance implications of the code were simply not examined sufficiently closely. It does point out, however, the importance of testing the code at scale periodically as these performance bottlenecks were not evident in smaller performance benchmark runs. One modification of more significance is described below.

Reproducible distributed sums At least 4 distributed sums, some global and some over specific geographical regions, are computed each timestep. CAM requires reproducibility in its numerics, that is, that the computed solution be invariant to the number of processes and threads used in the computation. Heretofore, the distributed means were calculated first by summing over the undecomposed dimension (if a three-dimensional field), then sending the resulting two-dimensional field to process 0 for final determination. Both at large problem sizes and at large process counts, this serialization degrades performance scalability. (For large problem sizes the memory requirements also become burdensome.) A number of solutions were considered, but the one we implemented is a variable-precision algorithm that calls `MPI_ALLREDUCE` twice, once with the `MAX` operator to determine the correct normalization and required precision and once to sum the integer vector representing local sums. If a reasonable upper bound on the summands is already known, the first call to `MPI_ALLREDUCE` can be eliminated. (Note that the obvious algorithm of computing in quad-precision is not generally viable in that compilers on many of our target systems do not support `REAL*16`. Importing public domain quadruple or higher precision packages presented other difficulties, leading us to implement our own special-purpose algorithm.)

7 Empirical Results

Figure 2 summarizes the impact of our work to date. The left graph in Figure 2 compares (*green*) the maximum throughput when running the `current` version of CAM restricted to the runtime options that are available in the `previous` version with (*blue*) the throughput when using the `current` optimal settings and approximately 4.9 times more MPI processes. Data are presented for three grid resolutions, all with

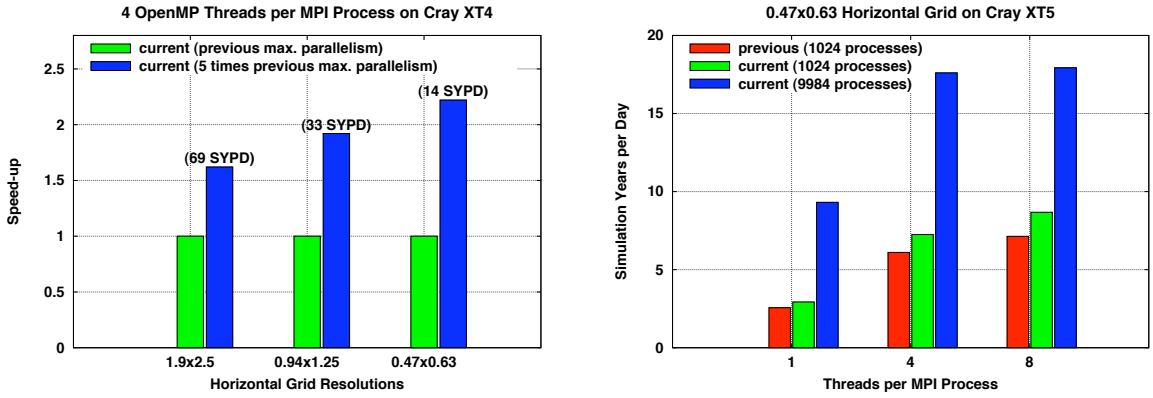

Figure 2: Increased throughput from recent performance optimizations to CAM for model resolutions with 26 vertical levels

26 vertical levels, run on a Cray XT4 with quad-core Opteron nodes. Only one MPI process is assigned to each node, and 4 OpenMP threads are used per process. The MPI process counts for the *green* data are the maxima listed in Table 1.

The right graph in Figure 2 compares throughput for the 0.47×0.63 horizontal grid with 26 vertical levels on a Cray XT5 with eight processor cores per node (two quad-core Opteron processors). Data are presented for MPI-only runs (8 processes per node), 4 OpenMP threads per process (2 processes per node, 1 process per socket), and 8 OpenMP threads per process (1 process per node). The red bars indicate maximum throughput when using the `previous` version of CAM. The green bars indicate throughput when using the `current` version of the code, but run using only the parallel algorithm options available in the `previous` version. The blue bars indicate throughput when using all of the new optimization options and 9.75 times as many MPI processes.

The data in Figure 2 demonstrate capability. The parallel efficiencies are poor as we move to such extreme process counts. However, performance has not rolled over yet for any of these cases. That is, performance is maximized by using all of the processes in the *blue* data experiments. From these data it is clear that these optimizations are most effective for larger grid resolutions and for fewer OpenMP threads per process. More than 4 OpenMP threads is not efficient at large MPI process counts in any case, so this is not as significant. The `current` version of the code, even when restricted to the `previous` algorithm options and maximum MPI process counts, is between 15% and 20% faster than the `previous` version. Note, however, that these optimizations are most critical at scale. Without them, we could not use additional parallelism effectively.

Figure 3 presents one view of how the additional throughput is attained. In the labels, the first two numbers are the size of the virtual process grid used to decompose the computational grid in the dynamics.

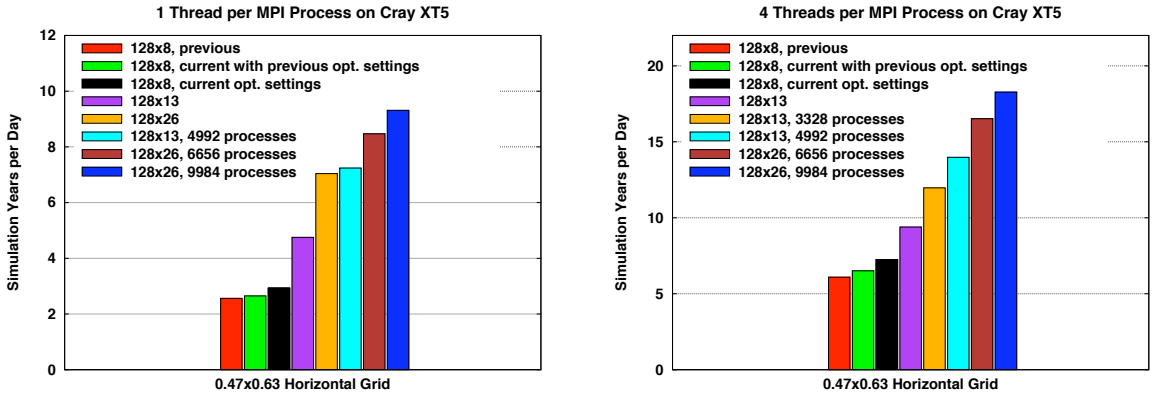

Figure 3: Throughput increase as additional parallelism is exploited for model resolutions with 26 vertical levels

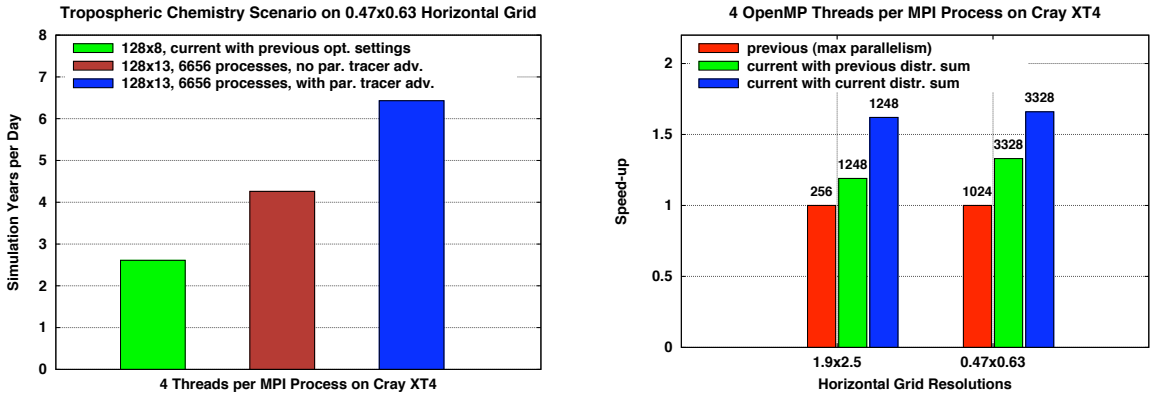

Figure 4: Importance of tracer advection parallelization and of new reproducible distributed sum algorithm, respectively, in improving CAM throughput at scale.

The first number is the number of processes used to decompose the latitude dimension. The second number is the number of processes used to decompose the vertical and longitude dimensions. The third number, when present, indicates the total number of MPI processes when this is larger than the number indicated by the virtual process grid. These auxiliary processes are used to parallelize the tracer advection and to further parallelize the physics. The left graph describes results for MPI-only runs. The right graph describes results when using 4-way OpenMP parallelism. The results are qualitatively very similar, with the only difference being that using a 128×26 process grid was optimal when using a total of 3328 processes for the MPI-only runs, while using 128×13 process grid with auxiliary processes was optimal for the 4-way OpenMP runs. The important point here is that each increment to the number of processes increases throughput a nontrivial amount. We have not yet identified the maximum parallelism than can be used for this problem. This is a promising result when considering scenarios with more expensive physics and/or more tracers.

Figure 4 looks at the performance impact of specific optimizations. The left graph compares (*green*) performance of the `current` version run with the old optimizations, (*red*) the `current` version using more processes in the vertical and auxiliary processes in the physics, but not the tracer advection, and (*blue*) the `current` version also parallelizing the tracer advection. These data come from runs using full tropospheric chemistry, requiring almost 4 times as many tracers as in the default chemistry used in the earlier experiments, on the Cray XT4 using 4-way OpenMP parallelism. These data describe the separate contributions from exploiting additional parallelism in the physics and in the tracer advection. They also demonstrate the importance of exploiting additional parallelism for such computationally demanding scenarios, as it is only by increasing the processor count by a factor of over 6 that we achieve the targeted 5 simulated years per day.

The right graph in Figure 4 compares the performance achieved with and without the new reproducible distributed sum. The data were collected on a Cray XT4 using 4-way OpenMP parallelism. The number at the top of each bar is the number of MPI processes used. The `current` experiments for the 1.9×2.5 grid used a 32×13 virtual process grid and 1248 total MPI processes. The `current` experiments for the 0.47×0.63 grid used a 128×26 virtual process grid and no auxiliary MPI processes. It is clear that, at scale, the new distributed sum is a critical component to achieving improved throughput.

As noted previously, posting a large number of send requests during a short period of time runs the risk of overwhelming the target processes with messages that they are not yet ready to receive. This has been observed in both gather and scatter operations on the Cray XT system (using both MPI collective calls and equivalent point-to-point implementations), resulting in runs terminating with error messages indicating that, for example, MPI has "run out of unexpected buffer space" or that an event was "dropped". The first error message occurred on an XT5 during a gather associated with writing a restart file for a problem on a 0.47×0.63 grid. The run used 4-way OpenMP parallelism and 256 MPI processes. The second error message occurred on an XT5 during a series of scatters as part of the initialization for a problem with full tropospheric chemistry on a 0.47×0.63 grid. This was an MPI-only run on 3328 processors.

Setting appropriate MPI environment variables to larger values does eliminate the errors in these examples. However, this is a fragile approach because a sufficient value is a function of the process count and problem size, and is difficult to predict. In some circumstances we have not been able to set the environment variable large enough, as it exhausts the available memory. We are also concerned about the performance and memory impacts of preposting very large numbers of requests, though we have not yet observed this to be a problem. In any case, by specifying handshaking and a maximum number of requests in logical gather

operators and by using blocking send protocols in the scatter operators, we have been able to eliminate the need to modify the default settings of MPI environment variables on the Cray XT4 and XT5.

Similar problems can occur with other irregular communication patterns, such as those arising from transposes connecting different sized decompositions. We first noticed anomalously large communication times when transposing from a latitude-longitude decomposition to a latitude-vertical decomposition that was three times smaller. In this instance, the runtime for the model was 50% to 100% *slower* than when using one-third as many processes (with the same size latitude-vertical and latitude-longitude decompositions). Similar behavior was observed on the Cray XT4, the IBM BG/P, and the Atlas cluster. The performance degradation was traced to extremely large communication times involving one-third of the MPI tasks (5 times larger than the communication times for the other two-thirds). The algorithm in place at that time was that each task would post all of its nonblocking receive requests followed by all of its nonblocking send requests. We believe that early arrival of messages from the otherwise idle two-thirds of the processes at the one-third active processes was causing the performance anomaly. By enabling handshaking - delaying send requests until the receiver was ready to receive them - this performance problem was eliminated. On the BG/P system, use of the MPI_Alltoallv command also eliminated the problem, and is generally somewhat faster than using a point-to-point implementation of these transposes in all situations. On the Cray XT4 and XT5, MPI_Alltoallv decreases the performance anomaly, but the point-to-point implementation with handshaking is still approximately twice as fast for transposes between the two decompositions, resulting in a 10% improvement in model runtime in typical cases as compared to the implementation using the MPI collective.

In contrast, when using 2-way or 4-way OpenMP parallelism on the Cray XT and Atlas systems the handshaking protocol increased model runtime for this same example by 10% to 30% compared to the original algorithm, and the MPI_Alltoallv-based implementation exhibited similarly poor performance compared to the original point-to-point implementation. The BG/P performance was relatively insensitive to the communication protocol when using OpenMP parallelism. So, it has been important for us to retain the flexibility of chosing the communication protocol for different phases at runtime, empiricially determining with short test runs which protocols are most efficient. Note that MPI_Alltoallv on the Cray XT systems does perform well when all processes are sending and receiving from all other processes. It is only in cases when processes are receiving from subsets and sending to subsets that we have observed the point-to-point implementations to be superior, and this could change in the next update to the MPI library.

To the Reviewers: The work described here took place on all three target platforms: XT, BG/P, and the

LLNL Atlas system, and performance data from all three platforms drove our development efforts. We do not as yet have a set of final evaluation studies on the BG/P and Atlas comparable to that for the XT4/XT5. If the paper is accepted, we will add summary performance data for these other systems as well.

8 Discussion and Future Directions

We have made a number of improvements to the performance and scalability of the Community Atmosphere Model. Central to this has been support for inactive and auxiliary computational processes. The ability to have some processes inactive during the dynamics but active during the physics is relevant to all dynamical cores, and its importance increases for scenarios of intense physics, such as atmospheric chemistry and cloud superparameterization [7]. The ability to harness these auxiliary processes to parallelize tracer advection within the finite-volume dycore has also proven to be important, especially for scenarios with significant numbers of tracers such as found with atmospheric chemistry. Another important algorithmic modification was adding the ability to assign fewer vertical levels per subdomain for the finite-volume dynamics.

Exploiting additional parallelism exposed a number of performance limiters heretofore unrecognized or deemed unimportant. Among these were algorithms with unscalable computational complexities, algorithms with unnecessarily high communication overheads, and communication protocols that were more prone to contention than the alternatives. Addressing each of these was critical for effective utilization of the high process counts demonstrated in this paper.

While this work has focused on increasing the number of computational processes (MPI tasks), utilizing additional processors through OpenMP is relevant as well. The CAM physics utilizes OpenMP at the chunk level; that is, chunks are assigned to threads executing in parallel, and this scales well. The main portion of the finite-volume dynamics advances multiple vertical levels within an OpenMP loop. The additional MPI parallelism introduced in the vertical eliminates almost all of the OpenMP parallelism in the dynamics when running at scale. Several years ago we experimented with nested latitudinal threading within the finite-volume dynamics, but the resulting overhead worsened performance. With only one or two vertical levels per subdomain (when maximizing process count), we are currently revisiting inner OpenMP threading, either as the sole threading within the main dynamics or in conjunction with outer threading, if supported. Even so, this may have its limitations, as at maximum process count there are only 3 latitudes per subdomain.

As mentioned before, our current work is meant to address the immediate performance needs of the fifth assessment. However, we believe that significant additional improvements in performance scalability will

require moving to a more scalable dynamics solver. Three such solvers are currently being evaluated.

The first candidate is a finite-volume dynamical core on a cubed sphere grid [12]. The cubed sphere grid consists of six horizontal logically rectangular patches, connected in a manner that is topologically equivalent to a cube [14]. This approach avoids the polar singularity at the expense of special treatment at the cubes' edges and corners. It has been implemented in the atmospheric model of the Geophysical Fluid Dynamics Laboratory and shown to scale well to large process count [9]. The cubed sphere finite-volume approach has been implemented in NASA's GEOS model.

The cubed sphere is also the geometry of choice for the other two candidate dycores, one based on a spectral element discretization and one utilizing a discontinuous Galerkin approach. Both of these are being developed within a framework called HOMME [16]. HOMME supports two-dimensional horizontal domain decomposition via space-filling curves. The spectral element-based solver has been shown to scale efficiently to tens of thousands of processors for relevant atmospheric resolutions.

9 Acknowledgements

This research was sponsored by the Atmospheric and Climate Research Division and the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC and Contract No. DE-AC52-07NA27344 with Lawrence Livermore National Security, LLC. This work has been authored by contractors of the U.S. Government under contracts No. DE-AC05-00OR22725 and No. DE-AC52-07NA27344, and is released as LLNL Report LLNL-CONF-??????. 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. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. It also used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Prepared by LLNL under Contract DE-AC52-07NA27344.

References

[1] W. D. Collins, C. M. Bitz, M. L. Blackmon, G. B. Bonan, C. S. Bretherton, J. A. Carton, P. Chang, S. C. Doney, J. H. Hack, T. B. Henderson, J. T. Kiehl, W. G. Large, D. S.

McKenna, B. D. Santer, and R. D. Smith, *The Community Climate System Model Version 3 (CCSM3)*, J. Climate, 19 (2006), pp. 2122–2143.

[2] W. D. Collins, P. J. Rasch, B. A. Boville, J. J. Hack, J. R. McCaa, D. L. Williamson, B. P. Briegleb, C. M. Bitz, S.-J. Lin, and M. Zhang, *The Formulation and Atmospheric Simulation of the Community Atmosphere Model: CAM3*, Journal of Climate, 19 (2006), pp. 2144–2161.

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

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

[5] J. Drake, P. Jones, M. Vertenstein, J. White III, and P. Worley, *Software design for petascale climate science*, in Petascale Computing: Algorithms and Applications, D. Bader, ed., Chapman & Hall/CRC, New York, NY, 2008, ch. 7, pp. 125–146.

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

[7] M. Khairoutdinov, D. Randall, and C. Demott, *Simulations of the atmospheric general circulation using a cloud-resolving model as a superparameterization of physical processes*, Journal of Atmospheric Sciences, 62 (2005), pp. 2136–2154.

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

[9] ——, 2008. personal communication.

[10] R. Loft, S. Thomas, and J. Dennis, *Terascale spectral element dynamical core for atmospheric general circulation models*, in Proceedings of the IEEE/ACM SC2001 Conference, Nov. 10-16, 2001, IEEE Computer Society Press, Los Alamitos, CA, 2001.

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

[12] W. Putman and S.-J. Lin, *Finite volume transport of various cubed-sphere grids*, Journal of Computational Physics, 227 (2007), pp. 55–78.

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

[14] M. Rancic, R. J. Purser, and F. Mesinger, *A global shallow water model using an expanded spherical cube: gnomonic versus conformal coordinates*, Quart. J. Roy. Meteor. Soc., 122 (1996), pp. 959–982.

[15] S. Solomon, D. Qin, M. Manning, M. Marquis, K. Averyt, M. Tignor, H. Miller, Jr., and Z. Chen, eds., *Climate Change 2007 - The Physical Basis: Working Group I Contribution to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*, Cambridge University Press, Cambridge, England, 2007.

[16] M. Taylor, J. Edwards, and A. St. Cyr, *Experience with CAM/HOMME: CAM aqua planet simulations using a cubed-sphere grid*, 2008. Presentation at the CCSM Atmospheric Model Working Group Meeting, Boulder CO. See also http://www.ccsm.ucar.edu/csm/working_groups/Atmosphere/Presentations/2008WG.presentations/AMWG-thurs/taylor_amwg08.pdf.

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