Load-Balancing Algorithms for Climate Models

Ian T. Foster and Brian R. Toonen

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

Abstract

Implementations of climate models on scalable parallel computer systems can suffer from load imbalances due to temporal and spatial variations in the amount of computation required for physical parameterizations such as solar radiation and convective adjustment. We have developed specialized techniques for correcting such imbalances. These techniques are incorporated in a general-purpose, programmable load-balancing library that allows the mapping of computation to processors to be specified as a series of maps generated by a programmer-supplied load-balancing module. The communication required to move from one map to another is performed automatically by the library, without programmer intervention. In this paper, we describe the load-balancing problem and the techniques that we have developed to solve it. We also describe specific load-balancing algorithms that we have developed for PCCM2, a scalable parallel implementation of the Community Climate Model, and present experimental results that demonstrate the effectiveness of these algorithms on parallel computers.

1 Introduction

In recent years, several groups have worked to adapt the atmospheric circulation models used for weather and climate modeling for execution on scalable parallel computers (e.g., see [5]). Typically, these models use a regular three-dimensional grid to represent the state of the atmosphere, on which are performed distinct "dynamics" and "physics" computations. Parallel algorithms are derived by employing a static, two-dimensional, horizontal domain decomposition of this grid.

[0]To appear in: *Proc. 1994 Scalable High Performance Computing Conf.*, IEEE Computer Science Press.

Dynamics computations are concerned with the fluid dynamics of the atmosphere. They are typically (although not always) well load balanced. However, they have strong horizontal data dependencies and hence significant communication requirements on a parallel computer. The development of efficient parallel communication algorithms has been a major thrust of research efforts in this area.

Physics computations simulate processes such as clouds, radiation, moist convection, the planetary boundary layer, and surface processes that are not not directly concerned with fluid dynamics. These processes typically have no horizontal data dependencies and, hence, can be parallelized easily. However, they can be subject to significant spatial and temporal variations in the computational load per grid point. For example, the National Center for Atmospheric Research (NCAR)'s Community Climate Model (CCM2) performs radiation computations only in grid points that are in sunlight and gravity wave calculations only in grid points that are over land [4]. The resulting load imbalances can reduce overall efficiency by 10–20 per cent with current physics and are expected to become more significant in future models [6, 7]. As climate model runs are typically of extremely long duration, solutions to this load-balancing problem are of considerable importance.

Different climate models incorporate a variety of physics modules with different computational load characteristics. Hence, in addition to requiring that solutions to the load-balancing problem in climate models provide good performance, we require that they be as general purpose as possible. We have addressed these two requirements by developing a flexible schema-driven load-balancing library. Schemas, which may be specified prior to a simulation or computed on the fly, specify the physics data distributions that are to apply at different time steps. The load-balancing library handles automatically the communication required to move from one schema to another.

This is effectively a *run-time compilation* approach to load balancing [8], in that schemas are translated at run time into communication schedules that can then be reused many times.

We have used this load-balancing library to apply a range of different load-balancing techniques to a parallel climate model. These include one well-known technique (recursive bisection [3]) and others specialized toward the particular requirements of climate models. The specialized algorithms provide significantly better performance. We attribute this to the distinctive characteristics of climate models: in particular, the use of relatively small grids, rapid changes in load, and (in some cases) regular and predictable changes in load patterns.

The rest of this paper is as follows. Section 2 describes the structure of the parallel climate model that we use to evaluate our load-balancing techniques and the computational load imbalances found in this model. Section 3 describes the algorithms that we have developed to correct these load imbalances and the structure of the library developed to implement these algorithms. Finally, Section 4 presents performance results for the various algorithms, and Section 5 presents our conclusions.

2 PCCM2

While much of the work reported in this paper is independent of any particular climate model, our implementation work and empirical studies have been performed in the context of PCCM2, a parallel implementation of the National Center for Atmospheric Research (NCAR)'s Community Climate Model (CCM2) developed at Argonne and Oak Ridge national laboratories [1]. Hence, we provide a brief introduction to the structure of this model.

2.1 Model structure

Both dynamics and physics operate on a set of three-dimensional data structures with size $N_{glat} \times N_{glon} \times N_{gver}$, where N_{glat}, N_{glon}, and N_{gver} are the number of grid points in the latitudinal, longitudinal, and vertical directions, respectively. The parallel implementation uses domain decomposition techniques to decompose these data structures and associated computation in the two horizontal dimensions [1, 2]. Some of these data structures are used only by dynamics or only by physics; others are shared by the two components. At each time step, a subset of these data structures are passed between the two components of the model, which are executed one after the other. Hence, it is most efficient in the absence of load imbalances to decompose physics data structures in the same way as dynamics data structures.

Dynamics data structures are decomposed using a static, uniform decomposition onto $P_{lat} \times P_{lon}$ processors, where P_{lat} is the number of processors in the latitudinal direction and P_{lon} is the number of processors in the longitudinal direction. In order to exploit symmetries in the dynamics computations, each processor receives data from $N_{glon} \times N_{glat}/(2 \times P_{lat})$ latitudes from the north and symmetrically the same number of latitudes from the south, for a total of $N_{llat} = N_{glat}/P_{lat}$ latitudes. In the longitudinal direction, each processor receives data from $N_{llon} = N_{glon}/P_{lon}$ longitudes.

2.2 Load imbalances

In the current release of PCCM2, PCCM2.1, physics load imbalances account for 8.1 per cent of total execution time at T42 resolution on the 512-processor Intel DELTA computer. This proportion is expected to increase substantially as other components of the model are optimized. (Currently, some components of dynamics are quite inefficient, a fact that of course tends to decrease the impact of load imbalances on performance.)

Load distribution (and hence load balancing) within PCCM2 is complicated by the fact that there are three quite different types of physics time step: no radiation, partial radiation, and full radiation [4].

In the current model, the time step is 20 simulated minutes. Every time step, various computations concerned with precipitation, gravity wave calculations, convective processes, etc., are performed. These may lead to both spatial and temporal imbalances (e.g., land/sea imbalances, seasonal cycle, and weather patterns such as convection over the Indian subcontinent during the monsoon); however, in the current CCM2 the magnitude of these imbalances is quite small [7].

Every hour (every 3 time steps) an additional "partial radiation" module is invoked to perform time-consuming shortwave radiation calculations in grid points exposed to sunlight. This produces a significant load imbalance, as illustrated in Fig. 1. This figure shows, for each of the 512 processors of the Intel DELTA, time spent in physics during a single partial radiation time step. The top part of the figure shows the spatial distribution of computational load) on the 16×32 processor processors, with dark shading representing more computation; the histogram shows the

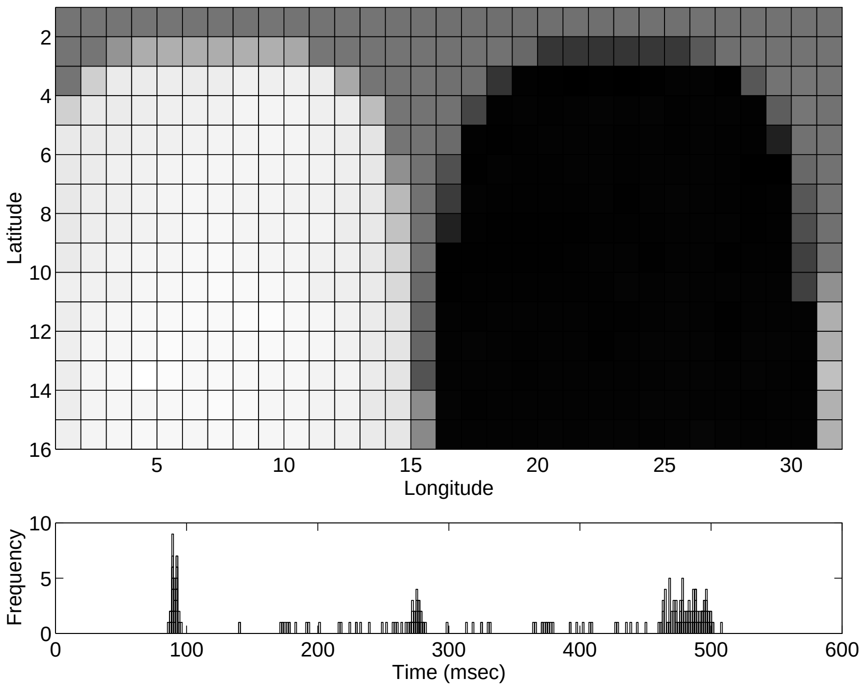

Figure 1: Physics Computation Time: 16×32 Processor Mesh on Intel DELTA, No Load Balancing

distribution of times. Notice that there is a factor of 5 variation between the least and most loaded processor.

Every twelve hours (36 time steps), an additional "full radiation" time step module is invoked to compute the absorptivity and emissivity of longwave radiation. This is an extremely time-consuming calculation but involves the same amount of computation at every grid point.

It turns out out that in PCCM2, the most significant source of load imbalance is the partial radiation calculation [7]. In addition to the strong spatial load variation illustrated in Fig. 1, this suffers from a substantial temporal variation due to the diurnal and seasonal cycles. The earth's rotation about its axis causes the area exposed to solar radiation to shift continuously westward, passing over the whole mesh once every 24 hours (72 time steps). The revolution of the earth around the sun results in a cyclic annual drift of the solar declination between the summer and winter solstices, causing the latitudes exposed to solar radiation to change over time.

3 Load-balancing algorithms

We are interested in developing general load-balancing techniques that can be used to compensate for load imbalance in PCCM2 and other similar models. These must certainly be able to compensate for load imbalance in PCCM2; in addition, because both future versions of PCCM2 and other models may have quite different load distributions, it is important to design load-balancing techniques with a high degree of generality.

It is almost always the case in atmospheric circulation models that within physics, computation performed at each grid point (vertical column) is independent of that performed at other points. This means that grid points and their associated computation can easily be migrated between processors. Unfortunately, these modified decompositions cannot easily be used (and probably will not be efficient) in dynamics. Hence, the use of load balancing leads to a need for separate dynamics and physics grids. Communication is required at each time step to move data between these grids.

In some parallel programs, load imbalances can be addressed by a static mapping of computation to processors that minimizes total execution time. However, this does not appear feasible in climate models in general and in PCCM2 in particular. The rapid temporal changes in PCCM2 load distribution and the differing load distribution characteristics of the three types of time step would make any static mapping inefficient. Instead, we require load-balancing algorithms that change mapping frequently—in fact, on almost every time step.

3.1 Library

To provide a flexible framework for the investigation of load-balancing algorithms, we have developed a general-purpose, configurable load-balancing library. The programmer need specify just the mapping of physics grid points to processors that is to apply at each time step; the library then organizes the movement of data required to support this mapping. A mapping is specified by a *schema*, a two-dimensional array in which each entry is an integer processor number, and a *program*, which indicates which schema is to apply at each time step. The load-balancing library uses schedule and schema information to generate *communication schedules* which are executed to actually move data between processors. Data movement is required for three purposes:

Input: Grid points modified by dynamics and shared with physics must be communicated prior to calling physics if the dynamics and physics grid distributions place that grid point on different processors.

Output: Similarly, grid points modified by physics and shared with dynamics must be communicated prior to calling dynamics if the dynamics and physics grid distributions place that grid point on a different processor.

State: Grid points used only by physics must be communicated when moving from one physics grid distribution to another.

3.2 Algorithms

The various load-balancing algorithms that we have developed for PCCM2 are primarily intended to compensate for the load imbalance due to the diurnal cycle. Hence, they migrate grid points only within latitudes; there is little purpose in migrating grid points within longitudes, because there is little load imbalance in the north/south direction. The algorithms do not compensate explicitly for land/sea imbalances or for imbalances due to the seasonal cycle. However, the pairing of symmetric north-south latitudes in the initial distribution compensates for most of the latter variation.

In the following, we describe four different load-balancing algorithms. These differ in both migration frequency and migration patterns.

Swapping. The simplest algorithm that we consider swaps every second data column with the processor directly opposite itself within its latitude. This column will be located on the processor that is longitudinally $P_{lon}/2$ processors away. The algorithm causes a contiguous block of grid points exposed to daylight to be dispersed to processors that contain few or no day points and, hence, does an excellent job of load balancing. Because each processor moves half of its data to another processor, it has the disadvantage of always moving a large amount of data. On the other hand, the communication pattern is predictable and hence amenable to optimization.

The other algorithms that we consider seek to achieve a similar quality of load balance while moving fewer points. To achieve this, they require an estimate of the amount of computation per grid point. We base this estimate on whether grid points are in sunlight or darkness.

Swapping 2. Given information about which grid points are exposed to solar radiation, a second swapping algorithm can be considered. This algorithm determines the difference d in the number of day points on a given processor and the processor $P_{lon}/2$ processors to the west. Then, $d/2$ day points from the more heavily loaded processor are exchanged for an equal number of night points from the opposing processor. This method reduces the number of grid points being transmitted but requires that a separate mapping be generated for each radiation time step. These mappings must either be cached or computed on the fly. While caching is sufficient for the trial runs associated with this study, it is infeasible for the extended runs common to climate models.

Movement. Slightly more complex is the algorithm that moves grid points rather than simply swapping them. This requires that we estimate computation costs associated with a column, as follows. First, we determine the ratio of the average computation times for day and night grid points. We refer to this ratio as the *day-night ratio* or d/n. This ratio is computed in a calibration step, prior to running the model. Using this information and the exposure information discussed previously, we can estimate the cost of computation associated with the grid points on every latitude and processor. The cost for any given latitude and processor is simply $(d/n) \times N_{day} + 1 \times N_{night}$, where N_{day} is the number of points exposed to daylight and N_{night} is the number of points in complete darkness.

The movement algorithm also requires that the data arrays used within physics be extended to provide room for more than N_{llon} points per latitude. Because PCCM2 dynamics assume N_{llon} points per latitude and cannot easily be modified, data must be copied from dynamics arrays to new "extended arrays" prior to calling physics. This requirement represents additional overhead not required in the swapping algorithms.

The movement algorithm attempts to move grid points between a given processor and the processor offset 180 degrees in longitude, as this was found to compensate well for the diurnal cycle imbalance. Specifically, day points are moved from the more heavily loaded processor until cost difference between the two processors is minimized. Then, the same technique is used for night points, thus providing a fine-grain adjustment. Since physics requires the grid points on each processor to be contiguous, points may need to move locally. To minimize this local movement, the points selected for transportation to opposing processors are chosen from right to left in the data arrays.

Recursive Bisection. For comparison purposes, we also implemented a version of the well-known recursive bisection algorithm. Using the same per-column cost estimates as the movement algorithm, this recursively divides the grid points within a latitude into two groups with approximately equal costs. The recursion continues until each processor has been assigned a contiguous set of points.

Since all four algorithms are designed to deal with the diurnal cycle, they are applied only during radiation time steps: that is, once every hour. During nonradiation time steps, physics data structures remain in their initial decomposition. Hence, the first swapping algorithm alternates between two mappings (schemas): the initial mapping and a swapped mapping. The other algorithms use a different mapping for each radiation time step.

4 Empirical studies

We now describe the empirical studies performed to evaluate the effectiveness of our load-balancing algorithms in PCCM2.

4.1 Method

We measured the performance of our four algorithms using an instrumented version of PCCM2. This instrumented version is based on an early release of PCCM2 in which the dynamics algorithms are quite inefficient. These inefficiencies reduce the proportion of total time taken in physics and hence the apparent impact of the load-balancing algorithms, but do not invalidate the algorithm comparison. In Section 4.4, we present the performance results when the first swapping algorithm is incorporated into a more optimized PCCM2.

Initially, we performed several calibration runs to determine the daypoint-nightpoint ratio required for the movement and recursive bisection algorithms. These runs were performed for a range of processor counts. Schema sets along with schedules were then generated and stored for each algorithm.

Performance runs were then performed for each algorithm and for a range of processor counts, from 32 to 512. Each run was performed once without instrumentation to measure overall performance, and once with instrumentation to gather information about load balancing overhead. The repeated runs ensured that instrumentation costs did not effect overall performance measurements. Because there is a certain variability in the times measured, however, a disadvantage of this approach is that the total times and measured overheads are not always consistent.

Each run was performed for forty time steps on the Intel DELTA at T42 resolution (a 64×128 grid). All

Table 2: Load-Balancing Overhead on 16×32 Intel DELTA Processors

Phase	Algorithm	Time (msec) Full	Part.	None	Mean
Input	bisection	9.0	38.3	2.2	13.4
	movement	4.0	19.0	2.5	7.6
	swapping	20.0	19.4	2.2	8.0
	swapping2	18.0	19.4	2.2	7.9
Output	bisection	8.0	44.2	2.0	15.1
	movement	3.0	19.7	2.0	7.5
	swapping	20.0	20.0	2.0	8.0
	swapping2	19.0	19.6	2.0	7.9
State	bisection	10.0	55.0	30.4	37.4
	movement	4.0	29.3	13.2	17.8
	swapping	26.0	27.1	14.5	18.6
	swapping2	26.0	26.5	13.9	18.1
Extend	bisection	8.0	8.5	8.4	8.4
	movement	8.0	8.9	8.4	8.5
	swapping	0.0	0.0	0.0	0.0
	swapping2	0.0	0.0	0.0	0.0
Total	bisection	35.0	146.0	43.0	74.3
	movement	19.0	76.9	26.1	41.4
	swapping	66.0	66.5	18.7	34.6
	swapping2	63.0	65.5	18.1	33.9

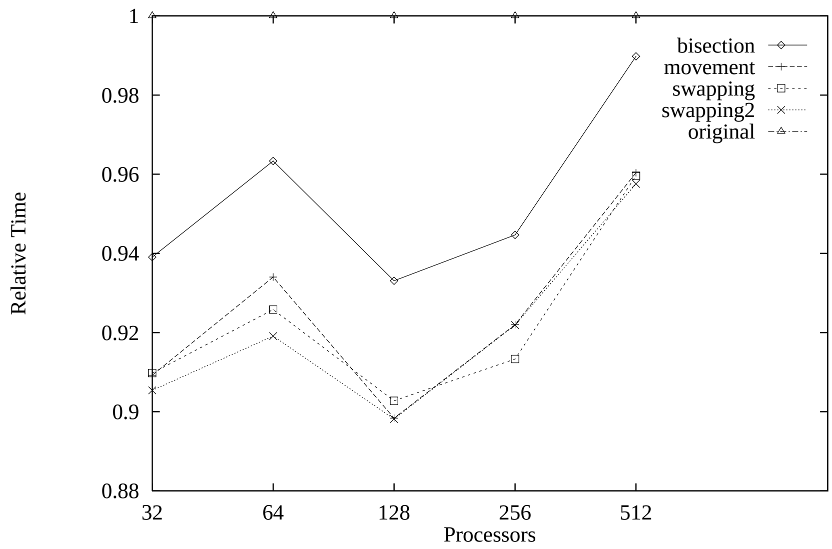

Figure 2: Relative Overall Performance on the Intel DELTA

algorithms were specified by schema sets and schedules loaded from files. A forty-step run was chosen because it encompasses all types of time steps and a full diurnal cycle, without requiring that an excessive number of layouts be buffered in memory. It does not, however, allow us to observe the effects of the seasonal cycle.

4.2 Results

Despite additional overhead from load balancing, we found that load balancing succeeded in reducing total execution time. From Fig. 2, we see that all load-balancing algorithms reduced total execution time. The bisection algorithm was significantly less efficient than the movement or swapping algorithms. The second swapping algorithm was somewhat more efficient than the movement or other swapping algorithm in most cases.

As can be seen in Fig. 2, the best load-balancing algorithm improves overall performance by ten per cent on 128 processors, eight per cent on 256 processors, and 4.3 per cent on 512 processors. (Recall that these results are with inefficient dynamics; results with an optimized model are given in Section 4.4 below.) On 512 processors, idle time due to load imbalance is reduced from 6.8 per cent to 0.8 per cent of total execution time; physics execution time, excluding the overhead of the load-balancing system, is reduced by 22.8 per cent. This almost total elimination of load imbalance is apparent in Fig. 3. The difference between the 6.8 per cent reduction in idle time and the 4.3 per cent reduction in execution time corresponds to load-balancing overhead.

Tables 1 and 2 give execution and overhead times measured on 512 processors, averaged over forty-step runs. Times are given both for each of the three types of time step and for an average time step. It should be noted that the latter value is not simply the average of the other three columns but is a weighted average, where the weights are based on the number of time steps of each type executed within a 24-hour period.

The overhead data in Table 2 breaks down load-balancing costs into four categories. The first three correspond to the input, output, and state communication operations described in Section 3.1, while the fourth is the copying required when moving data from the dynamics arrays to the extended arrays used in physics in the nonswapping algorithms. Note that the swapping algorithms do not incur these latter costs.

The recursive bisection algorithm improved overall performance by only 1.0 per cent. There are two reasons for this algorithm's poor performance. First, it moves more grid points: typically, most grid points will move. Second, the large spatial imbalance resulting from the diurnal cycle is not easily removed without reordering points within a latitude. Not having the ability to intersperse night points among the day points, this algorithm fails to make the fine-grain adjustments necessary to balance the radiation calculations.

Table 1: Load-Balancing Results on 16×32 Intel DELTA Processors

Section	Algorithm	Time (msec)				Relative
		Full	Partial	None	Average	Speedup
Overall	bisection	3589.0	1179.5	858.6	1032.5	1.010
	movement	3592.0	1103.5	847.3	1001.8	1.041
	swapping	3489.0	1111.6	846.7	1001.0	1.042
	swapping2	3473.0	1102.8	848.2	998.9	1.044
	original	3740.0	1276.5	823.9	1043.2	1.000
Physics	bisection	2829.0	346.5	63.7	226.9	1.185
	movement	2826.0	294.1	63.8	210.9	1.275
	swapping	2702.0	306.4	62.9	210.6	1.276
	swapping2	2700.0	296.5	63.0	207.6	1.295
	original	2998.0	504.5	47.0	268.8	1.000

In theory, one would expect the movement algorithm to outperform the swapping algorithms; however, the empirical data shows it to be less effective than expected. Although this algorithm communicates less data than either of the swapping algorithms, it has the additional overhead of extending the physics arrays. It also proves to be slightly less effective in balancing load, because it expects the computational costs associated with a day or night grid point to be constant. However, physics contains other imbalances besides the diurnal cycle. Although smaller in magnitude, they do have an impact which is compensated for by swapping but not by movement.

4.3 Other issues

The algorithms utilized in this study switch to an identity mapping during nonradiation time steps. Hence, physics state data is reorganized both before and after each radiation time step. In principle, this data could be cached on each processor in the swapping algorithm, avoiding the need for the reorganization. As Table 2 shows, the state data reorganization is a large proportion of total communication time. Hence, this is expected to have a significant impact on performance. We will investigate this optimization in future work.

Extended arrays are a source of overhead in the PCCM2 implementation of the data movement algorithms, because data must be copied to and from the extended arrays at each time step. This overhead could be avoided in a climate model that used extended arrays throughout both physics and dynamics. Our results suggest, however, that the swapping algorithms would still outperform the data movement algorithm.

While it would be possible to devise a new movement algorithm that was aware of the other imbalances that result in inefficiencies in the current movement algorithm, the additional overhead associated with this awareness would likely cancel out any improvements. In addition, we would then encounter the problem of generating and storing the associated communication schedules.

4.4 Optimized PCCM2

Further enhancements were made to PCCM2 independent of the version used for development of the load-balancing libraries. Many modifications were made in the dynamics portion of the code, but the physics remained relatively untouched. The modifications made to the newer version of the code have resulted in a substantial performance improvement within dynamics and thus have made the physics imbalances more significant to the overall execution time. Trial runs using the first swapping algorithm indicate an overall improvement of 5.9 per cent as a result of the load-balancing code's being added to the current version of the model.

5 Conclusions

The results of this work are encouraging. The swapping algorithms succeeded in significantly reducing the load imbalance, improving the total execution time by 5.9 per cent on 512 processors. The overhead associated with the load-balancing code, however, is still rather high. Some of this overhead is inevitable given

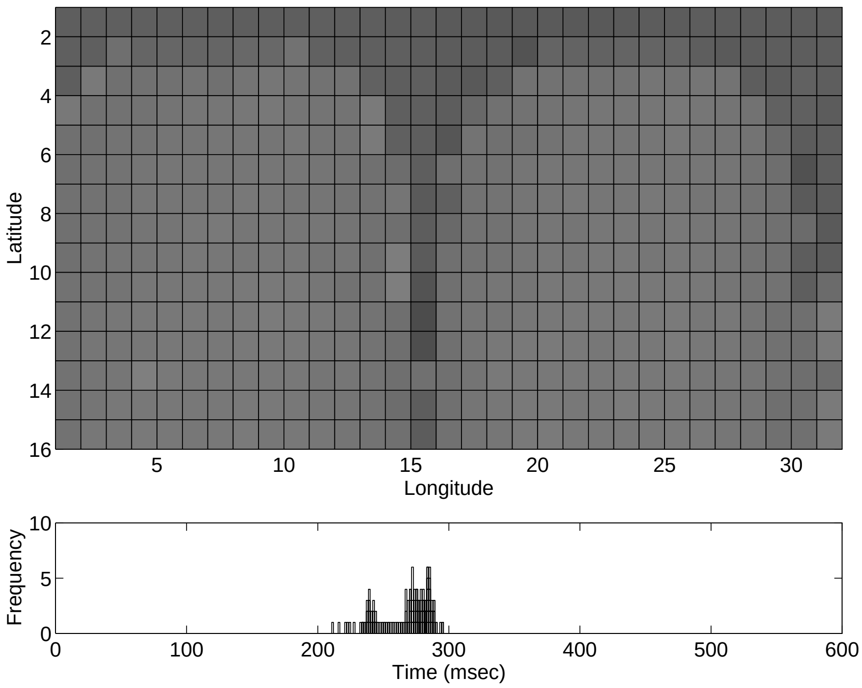

Figure 3: Physics Computation Time after Load Balancing with the "Swapping 2" Algorithm

the use of a general-purpose library; however, part of the overhead can be avoided with tuning. In future work, we will both tune the existing library and investigate alternative structures for the library, with the goal of reducing overhead. For example, we will consider the feasibility of caching state data. We also hope to extend this work to consider other physics modules and other climate models.

Acknowledgments

This work was supported by the Atmospheric and Climate Research Division of the Office of Energy Research, U.S. Department of Energy. We are grateful to members of the CHAMMP Interagency Organization for Numerical Simulation, a collaboration involving Argonne National Laboratory, the National Center for Atmospheric Research, and Oak Ridge National Laboratory, for sharing codes and results.

References

[1] Drake, J., Foster, I., Hack, J., Michalakes, J., Semeraro, B., Toonen, B., Williamson, D., and Worley, P., PCCM2: A GCM Adapted for Scalable Parallel Computers. *Proc. AMS Annual Meeting*, AMS, 1994.

[2] Foster, I., Gropp, W., and Stevens, R., The parallel scalability of the spectral transform method, *Mon. Wea. Rev.*, 125, 835–850, 1992.

[3] Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D., *Solving Problems on Concurrent Processors*. Prentice-Hall. 1988.

[4] J. Hack, B. Boville, B. Briegleb, J. Kiehl, P. Rasch, and D. Williamson, Description of the NCAR Community Climate Model (CCM2), NCAR Tech. Note TN–382+STR, National Center for Atmospheric Research, Boulder, Colo., 1992.

[5] Hoffmann, G.-R., and Kauranne, T. (eds.), *Parallel Supercomputing in the Atmospheric Sciences*, World Scientific, 1993.

[6] Michalakes, J., Analysis of Workload and Load Balancing Issues in the NCAR Community Climate Model, ANL/MCS-TM-144. Argonne National Laboratory, Argonne, Illinois, 1991.

[7] Michalakes, J., and Nanjundiah, R., Computational Load in Model Physics of the Parallel NCAR Community Climate Model, ANL/MCS TM-186, Argonne National Laboratory, Argonne, Illinois, 1994.

[8] Ponnusamy, R., Saltz, J., and Choudhary, A., Runtime-Compilation Techniques for Data Partitioning and Communication Schedule Reuse, *Proc. Supercomputing '93*, 1993.