Coupled Climate Models on Grids *

Sivagama Sundari Sathish Vadhiyar
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore, India
{sundari@rishi.,vss@}serc.iisc.ernet.in

Ravi Nanjundiah
Centre for Atmospheric & Oceanic Sciences
Indian Institute of Science
Bangalore, India
ravi@caos.iisc.ernet.in

Abstract

Community Climate System Model (CCSM) is a Multiple Program Multiple Data (MPMD) parallel global climate model comprising atmosphere, ocean, land, ice and coupler components. The simulations have a time-step of the order of tens of minutes and are typically performed for periods of the order of centuries. These climate simulations are highly computationally intensive and can take several days to weeks to complete on most of today's multi-processor systems. Executing CCSM on grids could potentially lead to a significant reduction in simulation times due to the increase in number of processors. However, in order to obtain performance gains on grids, several challenges have to be met. In this work, we describe our load balancing efforts in CCSM to make it suitable for grid enabling. We also identify the various challenges in executing CCSM on grids. Since CCSM is an MPI application, we also describe our current work on building a MPI implementation for grids to grid-enable CCSM.

1. Introduction

Community Climate System Model (CCSM) [3] is a coupled global climate model developed at National Center for Atmospheric Research (NCAR), with contributions from external research groups, funded by the National Science Foundation, Department of Energy (DOE) and National Aeronautics and Space Administration (NASA). It is a multi-component application and consists of four climate subsystem models, namely atmosphere, ocean, land and ice, and a coupler component through which these models interact. CCSM is implemented as a multiple program multiple data (MPMD) programming model where each component is a separate parallel program by itself.

CCSM is a highly computationally intensive application that allows climate researchers to conduct fundamental research into the earth's past, present and future climate states through simulations. Each simulation run could span a few years to a few thousand years of the earth's climate depending upon the scientific issue to be studied, while the simulation time-steps are typically less than an hour. Such simulations could take a few days to a few weeks to execute on most of today's multi-processor systems. Moreover, as there are continuous research efforts to increase the numerical grid resolutions and include more scientific processes, the computational demands of the application are expected to increase in the near future. Therefore, it is very important to explore the possibility of significantly reducing simulation times by executing the application on grid systems.

In this work, we identify the various challenges of executing CCSM on grids and our proposed solutions. We also describe our load balancing strategies to improve the performance of CCSM. Finally, we describe our MPI implementation to enable the execution of CCSM on grids.

2. CCSM on Grid

Grid computing enables computing on multi-institutional, dynamic, heterogeneous resource pools, virtually expanding the bounds of resources available to scientific applications. However, the performance benefit incurred depends largely on application structure, usage requirements and on intelligent resource allocation.

Some of the benefits of executing highly computationally intensive and long-running applications like CCSM on a grid are:

1. **Increased Access.** While researchers in some institutes have access to their own powerful supercomputers, researchers in a large number of educational and research institutes can only access such high performance computing systems through the grid. Executing CCSM on grids could therefore enable these climate researchers to perform high-resolution global simulations.
2. **Increased Resource Availability.** Even for users who have access to supercomputing facilities or dedicated clusters, executing CCSM across clusters or supercomputers enables them to use larger number of processors, thereby potentially reducing the simulation time.
3. **Lower Queue Wait Times.** Most supercomputing facilities have a batch queuing systems through which users submit their jobs. The queue wait times in the batch sys-

* This work is supported by Ministry of Information Technology, India, project ref no. DIT/R&D/C-DAC/2(10)/2006 DT.30/04/07

tems increase significantly with the number of processors requested. Executing CCSM across multiple queues enables the submission of multiple small requests to different batch systems, thereby decreasing the overall queue wait times.

However, several issues have to be addressed before these benefits can be incurred:

1. **Performance** The application scalability, the component model scalability, size and frequency of communications, load-imbalances are issues that could affect the application performance on grid.
2. **Implementation** All communications in CCSM are performed through MPI. One major challenge is to enable the MPI processes started on different platforms to seamlessly connect and communicate, if possible without any modification to the application.
3. **Validation** The climate model has currently been scientifically validated for only a few configurations and on some homogeneous platforms. The simulations are highly sensitive to various platform and compiler dependent features. Multi-platform executions have to be carefully validated.

In the following section we describe the challenges involved in load-balancing CCSM and our work to address some of the large load-imbalances.

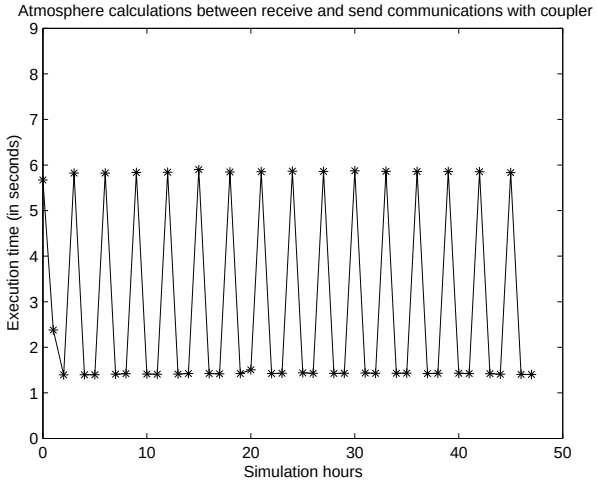
3. Load Balancing Challenges in CCSM

CCSM is an MPMD application with five parallel components: four climate models, each corresponding to one of the climate subsystems viz., atmosphere, land, ice and ocean, and a coupler which transforms data and coordinates the exchange of information across the model components. The periodic coupling of information occurs at different frequencies for different components. For example, the atmosphere exchanges information with other components every simulated hour, while the ocean exchanges information with other components only once in 24 simulated hours (the lower frequency of coupling with ocean is due to the fact that ocean has much higher inertia and the state changes more slowly than the other components). Further, several calculations are performed periodically at different frequencies. For example, the radiation parameter calculations which are part of atmospheric physics are performed only once in 36 time-steps of atmospheric simulation, where each time-step corresponds to 20 simulated minutes.

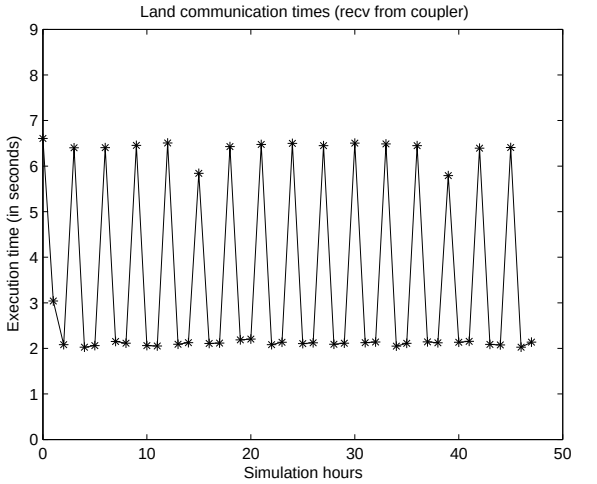
There are two sources of load imbalance in CCSM: (i) Load imbalance across components (inter-component load imbalance), and (ii) Load imbalance across processes of each component (intra-component load imbalance). Load balancing across components by allocating suitable number of processors to each component is complicated because the components are coupled at various frequencies and the computations and communications are interleaved. Moreover, the frequencies of some computation-intensive calculations are user specified. Another difficulty is that there are restrictions imposed by the components on their parallelization. However, the general guideline [1,2] for load-balancing is to give a large number (close to two-thirds) of the available processors to atmosphere and then try by trial and error to minimize idling of atmosphere processors by giving sufficiently many processors to the other components. In spite of following the above guideline for load-balancing, we identified a major load-imbalance, due to atmosphere long-wave radiation calculations, resulting in idling of processors.

The atmosphere component is the most loaded component among all components. Hence, any effort in reducing the time taken for atmosphere component helps in decreasing the overall execution time of CCSM. A major percentage of atmosphere calculations are the long-wave radiation calculations. These involve the calculation of emissivities and absorptivities of infrared radiations. These calculations are highly computation intensive and slowly varying, and hence performed at periodic intervals instead of at every time-step. For instance, in an experiment with 8 processors for atmosphere, 4 for ocean, 2 for ice and one each for land and coupler, calculation of absorptivity is 35% of the total atmosphere calculations. Figure 1(a) shows the times spent by an atmosphere processor performing calculations between receive and send communications with coupler. While the coupler communications have a period of 1 simulated hour, the periodicity of the absorptivity calculations is set to 3 simulated hours in this case. The spikes in the graph occurring at every 3rd simulated hour correspond to these computations. In these time-steps there is a large load imbalance among the components and hence, a large idling of the processors executing non-atmospheric components. This is reflected as peaks in the communication times of the non-atmospheric model components, since they are forced to wait until the coupler finishes its communications with atmosphere. The coupler itself is idle waiting to communicate with the atmosphere. Figure 1(b) shows the receive times of the land processor.

For atmospheric physics calculations, the atmospheric grid, consisting of latitudes on one axis and longitude on another, is divided into chunks, where each chunk is a collection of a fixed number of columns. A column represents all the vertical levels corresponding to a latitude-longitude pair (since the coupling between various variables is very tight in the vertical direction hence parallelization is not generally done in the vertical direction). Each atmosphere processor performs the physics calculations corresponding to a set of chunks. For each chunk, as part of its physics calculations, a call is made to the absorptivity calculating function, *radabs*. Inside *radabs*, the radiation calculations are performed for the columns that constitute the chunk. There are no depen-



(a) Computations in Atmosphere



(b) Idling in Land

Figure 1. Simulation times showing load imbalance due to long-wave radiation calculations

dencies between the calculations corresponding to any two columns.

We have developed a multi-level dynamic load-balancing scheme that addresses the imbalance due to long-wave radiation calculations without introducing delays in any of the components. We reduce the inter-component load imbalance by offloading the radiation calculations of atmosphere with idling processors of other components. Each atmosphere processor offloads or sends some columns of each chunk to processors of other components. Then, each processor of each component including atmosphere performs radiation calculations on the columns it possesses. After calculations, the atmosphere processors receive the results corresponding to the columns it offloaded to other components. The amount of offloading and the timesteps when the offloading is performed are dynamically determined based on the times the non-atmosphere processors are ready to share work and the different times taken for the different atmosphere processors time to start their long-wave radiation calculations.

The performance gain due to our load balancing scheme is illustrated in Figure 2. The experiments were conducted on a cluster of 8 dual-core AMD Opteron 1214 based 2.21 GHz Sun Fire servers running CentOS release 4.3 with 2 GB RAM, 250 GB Hard Drive and connected by Gigabit Ethernet. The execution time reported is for a climate simulation for 30 days using the lowest resolution data. The performance benefits are very high for lower number of processors. In this case, all components execute on only one or two processors. The ratio of the number of atmosphere to other processors is approximately 1:4. Thus, each processor will have to perform only around one-fifth of the original radiation calculations. As the number of processors available for the simulation increases, the number of atmosphere processors increases faster than those of the other components. This is because atmosphere is more computationally intensive and processors have to be distributed among the components in proportion to their computational load so as to obtain the lower simulation times. When sufficiently many processors are available, the ratio of atmosphere processors to all other processors is around 2:1 and each processor will have to perform two-thirds of the original radiation calculations.

4. Execution of CCSM across clusters

An important issue that dictates the performance of an application on a Grid system is the application scalability across clusters. There are two possible ways of executing CCSM across clusters:

1. executing one or more components on each cluster, with each component executed on only one cluster, and
2. executing some components across clusters.

The first option seems promising from the communication point of view since the components are loosely coupled

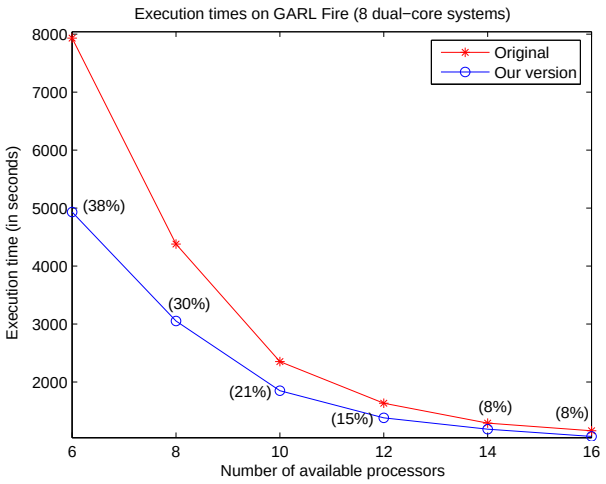


Figure 2. Load Balancing Results

and communicate with other components only once in several time-steps of their own model simulation. However, the performance gain is severely limited by the fact that the atmosphere model is more computationally intensive than the other components. Limiting each component to a single cluster implies that the atmosphere model can execute on only as many processors as available in the largest cluster. If all the components of CCSM are executed within the largest cluster, the atmosphere component would have been allocated two-thirds of the processors. Thus the performance gain due to using multiple clusters, with each component executed on only one cluster, will correspond to the negligible performance gain obtained due to executing atmosphere on all processors of the largest cluster over executing atmosphere on two-thirds of the processors of the largest cluster. Moreover, in the first option, the maximum number of clusters in the grid that can be used by the applications will be equal to the number of components of the application.

The second option is better for grid-enabling CCSM since it executes individual components across clusters. In this option, the maximum number of clusters that can be used is not limited by the number of components, but by the scalability in performance of the components when executed across clusters. This requires developing efficient communication schemes in the various stages of the individual components. Many current research efforts focus on improving scalability of individual components on single multi-processor systems, and scalability to a few thousand processors for most components, including atmosphere, has been reported [4, 5]. Our future research will be to achieve a similar scalability across the grid by developing fault-tolerant versions of different algorithms in the individual components.

5. MPI for Cross-Cluster Communications

CCSM is implemented using the well known Message Passing Interface (MPI). Enabling and executing CCSM across different clusters of the grid requires an implementation of MPI that can interface with the local MPI implementations on the individual clusters. Our current work is to develop a solution for seamlessly interfacing the MPI processes started on different clusters and supercomputers. Although some existing MPI-implementations have been developed for use across multiple computing sites [6, 7], we propose to use the local implementations of MPI, which are highly optimized for their respective systems, for intra-cluster communications.

Another challenge is to enable communications with nodes having private/hidden IP addresses. Most clusters have a front-end/master node and multiple slave nodes. Only the front-end nodes are accessible from outside the clusters. The slave nodes have hidden IP addresses. Hence a MPI communication involving two slave nodes of two different clusters has to be routed through intermediate nodes. Although there are some existing solutions for dealing with hidden IPs [8], some of the assumptions made in the solutions prevent them from being used on real multi-cluster grid environments. For example, PACX-MPI [8] assumes that one of the processors on each site on which an MPI process is executing is accessible to the outside world. This assumption is not true in typical batch systems where the local MPI processes are started only on the slave nodes. The front-end nodes are used only for starting the MPI processes on the slave nodes. Another limitation in existing solutions is due to the assumption that the front-end nodes of all clusters will be able to directly communicate with each other. In many environments, the front-end nodes can communicate only through multiple intermediate nodes.

Our implementation tries to address the above issues by use of communicating daemons. We have three basic daemons: E-Daemon, S-Daemon and T-Daemon. These daemons are started with suitable arguments at various sites in order to establish the communicating system. Our MPI version uses the existing local MPI-implementations for the intra-site communications, and performs inter-site communications through these daemons. The MPI process contacts and connects to the locally accessible S-Daemon to start any inter-site communication. The S-Daemon in turn contacts and connects to the T-Daemon at the intermediate site. The T-Daemon decodes the destination information which is part of the message header and connects to the E-Daemon. The E-Daemon performs file read or write depending upon the MPI call being receive or send. Figure 3 shows the basic structure of these daemons. Note that the design can be easily extended to multiple clusters and complex communication paths merely be starting more daemons. Also, processes dynamically spawned can prevent blocking on communications and allow communications to take place in parallel.

In this framework, communications for MPI_Send and MPI_Recv happen independently. Let us consider a process (p1 in figure) in one site (site-I) posting a MPI_Send and

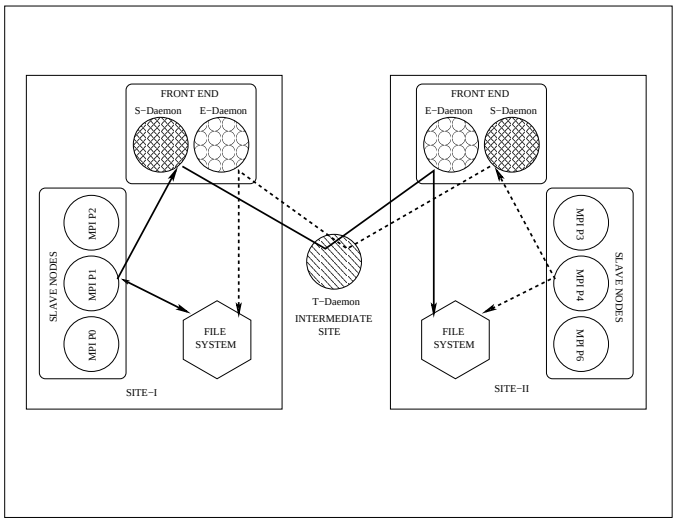


Figure 3. Inter-cluster MPI Framework

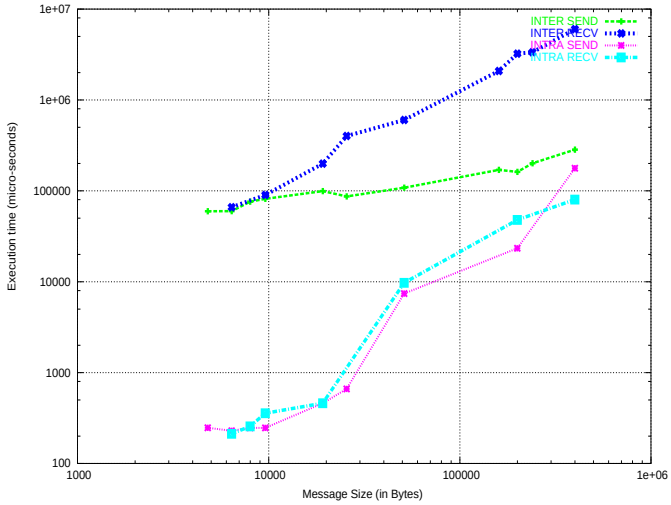


Figure 4. Inter and Intra cluster Communication Times for Different Message Sizes

a process (p4 in figure) in another site (site-II) posting a MPI_Recv. When MPI_Send is posted, p0 writes the data to its local file-system and the header to the file-system at site-II through the daemons as shown by the arrows in the figure. When MPI_Recv is posted, p4 continuously checks for the corresponding header in the local file-system and when the file is found (i.e. after the MPI_Send at the other site has completed), the data from the file-system at site-I is read through the daemons. Note that although the communication path is the same (symmetric with respect to the sites) for both send and recv, the entire message is transferred across the sites only when the recv is posted. Preliminary comparisons of time taken for inter and intra cluster transfers are shown in Figure 4. The experiment involved two clusters in two sites: a Sun cluster in Indian Institute of Science (IISc), Bangalore and an IBM AIX cluster in Centre for Development of Advanced Computing, Bangalore. The intermediate node is the front-end node of an IBM cluster at IISc. The figure clearly shows that with increase in message sizes, the inter and intra cluster transfer times become comparable.

6. Conclusions

In this work, we have explained our load balancing strategy to make CCSM suitable for grid computing. We also enumerated our plans for grid enabling CCSM. We have described our current efforts in developing a MPI implementation for cross-cluster communications.

7. Future Work

Our plan is to continue our implementation of MPI for cross-cluster communications. This MPI will have all the MPI functions used by CCSM. We then plan to execute CCSM across clusters and determine the conditions when grid executions will be useful. We also plan to improve the scalability of the algorithms in CCSM.

References

[1] G. Carr. An Introduction to Load Balancing CCSM3 Components. In *Proceedings of Software Engineering Working Group (SEWG) Meeting, CCSM Workshop, NCAR*, June 2005.

[2] G. Carr, M Cordery, J. Drake, M. Ham, F. Hoffman, and P. Worley. Porting and Performance of the Community Climate System Model (ccsm3) on the Cray X1. In *Proceedings of the 2005 Cray Users Group (CUG) Meeting*, Albuquerque, New Mexico, May.

[3] Community Climate System Model (CCSM). `http://www.ccsm.ucar.edu`.

[4] Arthur A. Mirin and William B. Sawyer. A Scalable Implementation of a Finite-Volume Dynamical Core in the Community Atmosphere Model. *International Journal of High Performance Computing Applications*, 19(3), August 2005.

[5] Arthur A. Mirin and Patrick H. Worley. Extending Scalability of the Community Atmosphere Model. In *Journal of Physics: Conference Series, 78 (Proceedings of SciDAC 2007)*, Boston, MA, June 24-28 2007.

[6] MPICH-VMI2 Teragrid User Manual. `http://vmi.ncsa.uiuc.edu/teragrid/index.php`.

[7] Guillaume Mercier Oliver Aumage. MPICH/Madeleine: a True Multi-Protocol MPI for High Performance Networks. In *15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, 2001.

[8] PACX-MPI. `http://www.hlrs.de/organization/amt/projects/pacx-mpi`.