Dynamic Component Extension: a Strategy for Performance Improvement in Multicomponent Applications

Sundari M. Sivagama, Sathish S. Vadhiyar and Ravi S. Nanjundiah

International Journal of High Performance Computing Applications 2009; 23; 84
DOI: 10.1177/1094342008101364

DYNAMIC COMPONENT EXTENSION: A STRATEGY FOR PERFORMANCE IMPROVEMENT IN MULTICOMPONENT APPLICATIONS

Sivagama Sundari M.[1]
Sathish S. Vadhiyar[1]
Ravi S. Nanjundiah[2]

Abstract

Multicomponent application paradigms have gained prominence in many significant multidisciplinary scientific applications. In this work, we propose a software strategy called dynamic component extension for multicomponent applications to improve application performance by minimizing processor idling. In this strategy, the processor space of a component is dynamically extended to include the processors of other components during certain computationally intensive phases of the component. We demonstrate its use in improving the performance of one of the most prominent multicomponent applications, the community climate system model (CCSM). In this application, we dynamically extend the atmosphere component to minimize the idling in other components caused by large periodic temporal load imbalances in the atmosphere component. By means of experiments on different parallel platforms with different numbers of processors, we show that using our strategy can lead to about 15% reduction and savings of several days in execution times of CCSM for 1000-year simulation runs.

Key words: multicomponent applications, temporal load imbalances, idling, load balancing, coupled climate models

1 Introduction

With major advances in high-performance computing, the scientific community is moving toward multidisciplinary multicomponent models to accurately model interacting physical processes or phenomena. Examples of such applications include models of climate, space weather, solid rockets, fluid–structure interaction, heart disease, cancer, ocean plankton population, and nuclear energy (Collins et al. 1998; Lefantzi and Ray 2003; Toth et al. 2003; Coveney et al. 2005; Delgado-Buscalioni et al. 2005; Larson, Jacob, and Ong 2005). Typically, these applications involve long-running simulations of constituent model components with periodic communication or coupling between the components.

While each of the constituent model components is generally parallel, the two basic software strategies used to couple these components are sequential and concurrent. The sequential strategy, illustrated in Figure 1(a), involves a driver that for each coupling cycle or time step, sequentially invokes each of the constituent models on the same sets of processors. The concurrent strategy, illustrated in Figure 1(b), typically follows the multiple program multiple data (MPMD) paradigm and involves concurrent execution of the models on different sets of processors with coupling often performed through a dedicated coupler. For example, the parallel climate model (PCM[1]) involving multiple coupled components is a sequential SPMD version while the community climate system model (CCSM[2]) is a concurrent MPMD version of coupled climate models.

The strategy used depends on various factors including amount of exploitable concurrency, target platform, memory footprint, development plan, model scalability, and so forth. The sequential strategy is adopted when there is not much exploitable concurrency among components of the application. The concurrent strategy is favorable for complex multidisciplinary coupled applications because the models can be built, developed, and tested as stand-alone applications by independent teams. Hence, the concurrent strategy is widely used for various large-scale applications (CCSM,[2] see Coveney et al. 2005). However, when the model components have complex interacting patterns and dependencies, the concurrent strategy can lead to large processor idling. Idling of processors executing certain components can also be the result of large temporal load imbalances caused by highly computationally intensive phases in a component

[1]SUPERCOMPUTER EDUCATION AND RESEARCH CENTRE, INDIAN INSTITUTE OF SCIENCE, BANGALORE, INDIA (SUNDARI@RISHI.SERC.IISC.ERNET.IN)

[2]CENTRE FOR ATMOSPHERIC & OCEANIC SCIENCES, INDIAN INSTITUTE OF SCIENCE, BANGALORE, INDIA

The International Journal of High Performance Computing Applications, Volume 23, No. 1, Spring 2009, pp. 84–98
DOI: 10.1177/1094342008101364

Figures 1–5, 9–11 appear in color online: http://hpc.sagepub.com

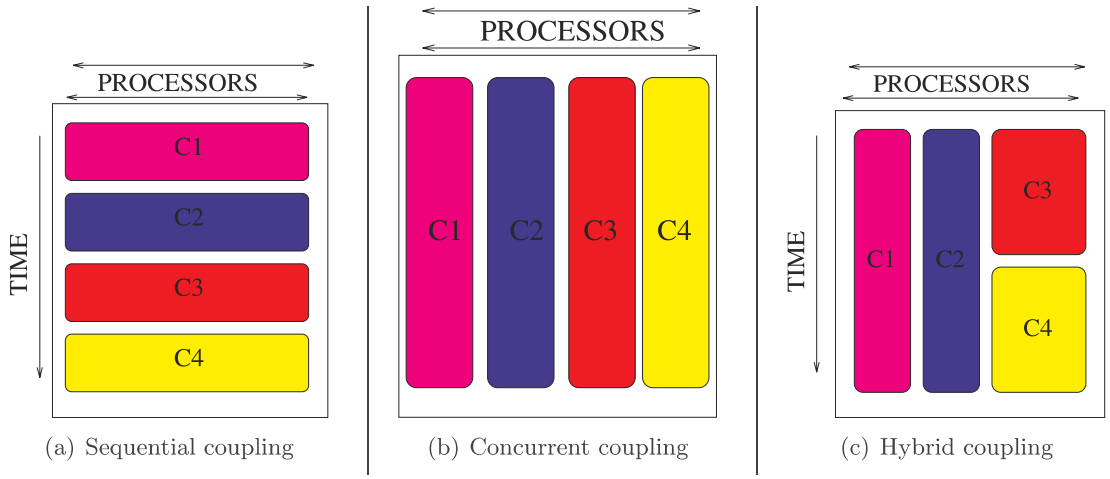


Fig. 1 Sequential, concurrent, and hybrid coupling.

within a time step or coupling period. Hybrid strategies can be used to alleviate the processor idling in concurrent strategy. In this strategy, a subset of components with dependencies is executed sequentially as illustrated in Figure 1(c). An example of such an implementation of the climate system model is the fast atmosphere–ocean model (FOAM[3]). While the hybrid strategy can address the idling resulting from component dependencies, it cannot address the idling arising from temporal load imbalances in a component.

In this work, we propose a software strategy called dynamic component extension to minimize idling arising from temporal load imbalances and improve performance in concurrent and hybrid versions of multicomponent applications. In this strategy, the processor space of the components containing temporal load imbalances is extended to include the processors executing other components during the computationally intensive phases of a time step or coupling period. The strategy is illustrated in Figure 2. The extension is dynamic since it takes into account the ready times of the processors for sharing the computational loads. The strategy is intended for multicomponent application execution in a cluster of homogeneous processors. We demonstrate the potential of the strategy to minimize idling and improve performance in a classic and foremost example of a multicomponent application, the community climate system model (CCSM). Our work is directly applicable only for those multicomponent applications where only one component has temporal load imbalance, is irregular, and executes longer than other components in some coupling intervals. We have not considered applications where two or more components can have longer execution times than the remaining components.

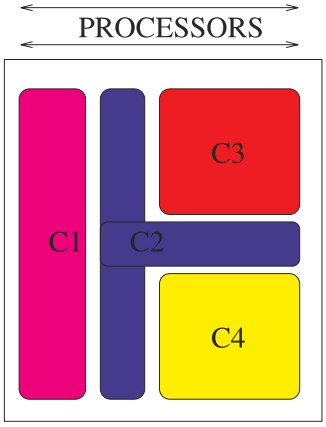


Fig. 2 Component extension.

The community climate system model (CCSM[1]) is a global climate system model from the National Center for Atmospheric Research (NCAR[4]). The CCSM climate model consists of five components, namely, atmosphere, ocean, land, ice, and a coupler component which transforms data and coordinates the exchange of information across the other model components. The CCSM is implemented as a multiple program multiple data (MPMD) programming model where each component is a separate parallel application by itself. Of the five components, atmosphere is the most computationally intensive and is

allocated the highest number of processors (Carr 2005; Carr et al. 2005). The atmosphere model consists of calculations for dynamics and physics. The physics includes calculations corresponding to shortwave (visible and ultraviolet radiation received from the sun) and longwave radiation. The longwave radiation calculations deal with calculations of radiation emitted by the earth–atmosphere system. The most computationally intensive part of these longwave radiation calculations is the estimation of emissivity and absorptivity of various constituents of the atmosphere. Because these coefficients change very slowly with time they are not computed at every time step but once every few simulated hours (typically between 3 and 12 hours). Thus the atmosphere component involves large periodic temporal load imbalances resulting from the computationally intensive emissivity and absorptivity components of longwave radiation calculations (for brevity, we will refer to calculations related to emissivity and absorptivity as longwave radiation calculations even though longwave radiation calculations also consist of other calculations). The temporal load imbalances result in idling of the processors executing non-atmosphere components during the longwave radiation calculations in the atmosphere component.

We apply our strategy to CCSM by dynamically extending the processor space of the atmosphere component (atmosphere processors) to include the idling processors executing the non-atmosphere components during the longwave radiation calculations in the atmosphere. The extension characteristics, including the amount and points of extension, and the processors of other components to include during the extension, are dynamically determined based on the times when the non-atmosphere processors are ready to share work and the different times taken by the different atmosphere processors to start their long-wave radiation calculations. By means of experiments with different processor configurations for two different modeling resolutions on five multiprocessor systems, we show that our dynamic component extension strategy can lead to about 15% reduction and savings of up to 50 days in execution times of CCSM for 1000-year simulation runs.

Section 2 discusses the effect of temporal load imbalance on performance of a coupled application and explains the benefits of the dynamic component extension strategy to improve the performance. In Section 3, we discuss related work on coupled multicomponent applications and load-balancing approaches in climate modeling systems. Section 4 identifies the challenges in load balancing longwave radiation calculations in CCSM. In Section 5, we describe our dynamic component extension (DCE) strategy applied to long-wave radiation calculations in the atmosphere component of CCSM. Section 6 describes the experiments and presents results that illustrate the benefits of using our approach over the existing code for CCSM. Conclusions are presented in Section 7 and future efforts are listed in Section 8.

2 Temporal Load Imbalance and Dynamic Component Extension

Temporal load imbalance or variation in computing load of a component across time steps can have a significant effect on the performance of a concurrent multicomponent system. We illustrate temporal load imbalance and dynamic component extension with a simple multicomponent application consisting of four components executing three time steps as shown in Figure 3(a). We assume that the components couple or synchronize at the end of each time step. The figure shows that the components are perfectly load balanced in the first and the third time steps. The figure also shows that component C2 performs more computations in the second time step than in the other time steps. Let ΔW represent these additional computations. This temporal load imbalance resulting from ΔW in C2 causes large idling and low utilization of the processors executing the other components in the second time step. The time taken for the ΔW computations in C2 and the corresponding idling times in other component processors are denoted as ΔT in the figure. Using a dynamic component extension scheme, C2 can be extended to also execute in other component processors during the ΔW computations in the second time step as shown in Figure 3(b).

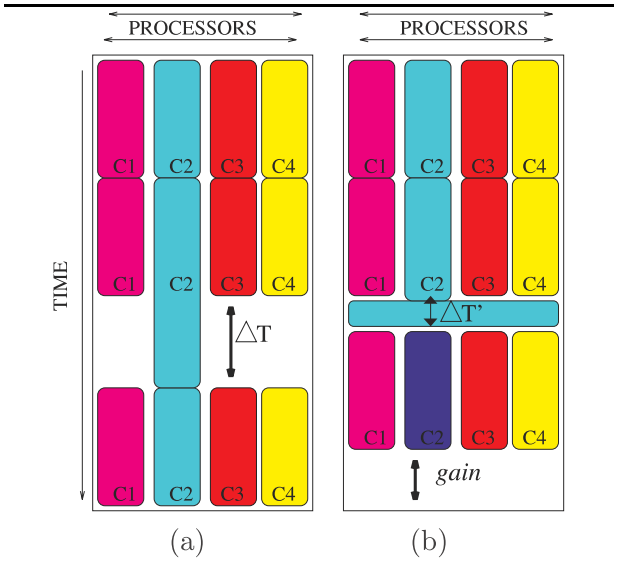


Fig. 3 Effect of (a) temporal load imbalance and (b) component extension on coupled application execution.

Assuming ideal speedup for the computations, the figure shows complete elimination of the processor idling and the resulting decrease in the overall completion time, denoted as *gain*. The time taken for the ΔW computations with the dynamic component extension is denoted as $\Delta T'$.

Generalizing the above example, we consider a multicomponent application consisting of n components, C1, C2, …, Cn with component Ci executing on n(Ci) processors. Component Cj has periodic temporal load imbalance because of additional computations, ΔW, in certain time steps. Assuming that the computations in Cj have ideal speedup and that the components are perfectly load balanced in other time steps, the time taken for the ΔW computations, $\Delta T'$, and the overall reduction in execution time of the application, *gain*, due to dynamic component extension are given by

$$\Delta T' = \Delta T \times n(Cj) / \sum_i n(Ci) \quad (1)$$

$$gain = \Delta T \times \sum_{i \neq j} n(Ci) / \sum_i n(Ci) \quad (2)$$

In general, the gain resulting from dynamic component extension in a multicomponent application is application dependent and depends on various factors including number of processors, distribution of processors to components, scalability of components, extensibility of additional computations in components, component coupling frequency and pattern, impact of the components with temporal load imbalances on the overall execution time and so forth. We demonstrate our strategy with CCSM, an important multicomponent application, and present our experiences in dealing with these factors and our solutions to the generic and application-specific challenges encountered.

3 Related Work

In this section, we first mention some of the work related to software coupling strategies for multicomponent applications. We then describe related efforts on coupled climate models.

3.1 Coupling Strategies in Multicomponent Applications

Chow and Addison (1999) present an overview of the software strategies and the tools and libraries available for multiphysics parallel applications. The work mentions the importance of dynamic load-balancing strategies for concurrent multicomponent applications. Lawrence (1997), in an overview of multidisciplinary aeropropulsion simulations, describes three approaches based on the nature of multidisciplinary coupling: loosely coupled, coupled process, and multiphysics. Our work in this paper applies to the second strategy. Lermusiaux et al. (2004) consider sequential, concurrent, and hybrid strategies to perform coupled physical and biogeochemical ocean simulations. Larson (2006) presents a heuristic set of definitions and organizing principles for coupled models. The work also describes the complexities involved in coupling a multicomponent application using the case study of CCSM. In our work, we also consider the execution time profile of the coupled applications. Larson et al. (2005) have developed MCT, a toolkit that allows construction of coupled multiphysics systems using parallel constituent models and has been used for coupling in a few multicomponent applications (CCSM^2; Michalakes et al. 2004; Collins et al. 2005). MCT supports several static component frameworks including single and multiple executables, and sequential, parallel, and hybrid component executions (Larson et al. 2005).

However, none of the existing work on multicomponent applications deal with dynamic optimization strategies involving multiple components. Our work deals with dynamic load balancing across multiple components using component extensions. Although there is a huge amount of literature relating to dynamic load balancing in single component parallel applications and dynamic intra-component load balancing (e.g. Dorneles et al. 2003) in multicomponent applications, we are not aware of any work related to multicomponent applications that addresses temporal load imbalances in components and the resultant inter-component load imbalances.

3.2 Load Balancing Coupled Climate Models

Various efforts have dealt with load balancing in climate models (Michalakes 1991; Foster and Toonen 1994; Michalakes and Nanjundiah 1994; Michalakes et al. 1994; Drake et al. 1995; Ford and Burton 1998; Nanjundiah 1998, 2000; Muszala et al. 2004, 2006). Although all these efforts show improvements in execution times of a single component, they do not address problems related to a coupled climate system model involving load imbalances between multiple components. Efforts that involve the other components in load balancing are based on determining the optimum static allocation of processors to components. These do not involve any algorithmic modifications for load balancing. For instance, the CCSM Load Balancing Workshop document (Carr 2005) outlines the general methodology to be followed for such load balancing. The technique suggested is a trial-and-error method. The largest possible number of processors are allocated to the atmosphere and the remaining processors are allocated to the other components such that the atmosphere component is not delayed. Carr et al. (2005) suggest general strategies for processor mapping of the highest resolution

models based on experiments with CCSM3 on the Cray X1. These static processor allocations only aim at preventing idling of the atmosphere component, while the non-atmosphere component processors will still have large idling times. Further, as the model evolves, the component characteristics can change significantly and hence the static configuration may not remain optimal.

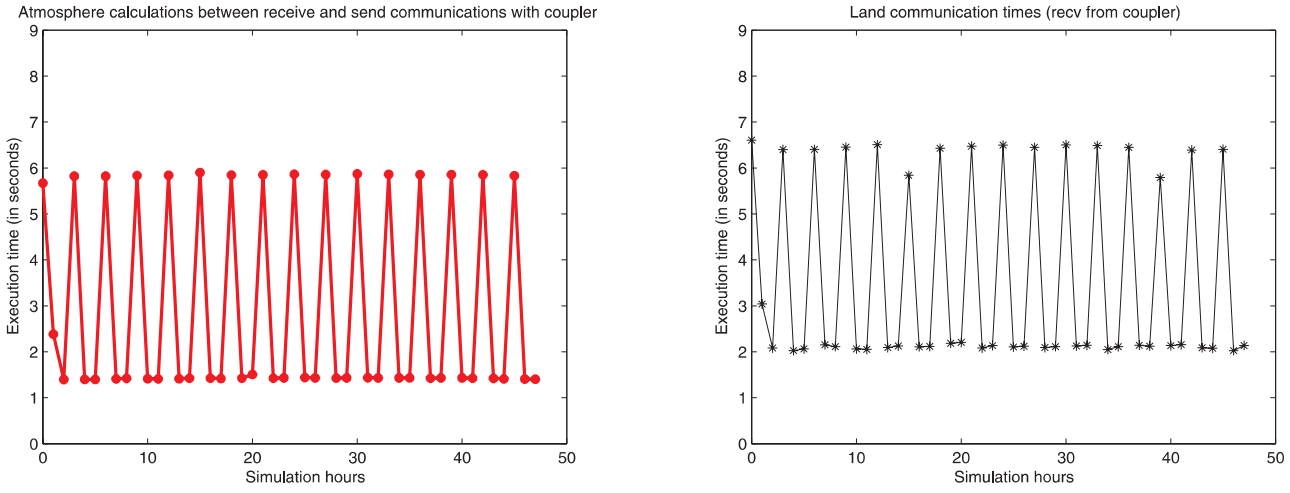
Our work tries to address several of the drawbacks discussed above by dynamically offloading columns of radiation calculations to other components and thereby minimizing component idling and also reducing the total execution time. While we have applied dynamic component extension to reduce temporal load imbalance resulting from longwave radiation calculations in the atmosphere component of CCSM in this work, our strategy is generic and can be applied for any huge temporal load imbalances in a component. Mirin and Worley (2007) have recently studied the scalability of a single component application, the community atmosphere model (CAM[5]) which is the atmosphere component of CCSM. Because of the different parallelization limits of the two phases of CAM, physics and dynamics, the authors use auxiliary processes to support larger parallelism for the physics phase. Thus, in the context of the multicomponent CCSM, using suitable processor distribution to the components, our strategy can be used to support the large parallelism of the physics phase by extending the physics phase of CAM to the idling processors executing other components instead of, or in addition to, the auxiliary processors. While the extension scheme of Mirin and Worley for the single-component CAM application needs the dynamic process management features of MPI-2,[6] our strategy for the multicomponent CCSM employs simple load balancing of the existing processors used for application execution.

4 Load Balancing Challenges in CCSM

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).

4.1 Inter-Component Load Imbalance resulting from Temporal Load Imbalance in the Atmosphere Component

A major percentage of atmosphere calculations are the longwave radiation calculations. For instance, in an experiment with eight processors for atmosphere, four for ocean, two for ice and one each for land and coupler, calculations of absorptivities in longwave radiation consumed 35% of the time for atmosphere calculations. Figure 4(a) shows the times spent by an atmosphere processor performing calculations between receive and send communications with the coupler at different time steps. While the coupler communications have a period of 1 simulated hour, the periodicity of the absorptivity calculations is set to 3 simulated hours for this experiment. The huge spikes in the graph occurring at every third simulated hour correspond to these computations. Thus, these long-wave radi-



(a) Computations in atmosphere

(b) Idling in land

Fig. 4 Computation intensive absorptivity calculations in atmosphere and the corresponding idling in land.

ation calculations cause temporal load imbalance in the atmosphere.

In these time steps, there is a large load imbalance among the components and hence large idling of the processors executing non-atmospheric components. This is reflected as peaks in the times for communications with the coupler for the non-atmospheric model components since they are forced to wait until the coupler finishes its communications with the atmosphere. The coupler in turn is idle waiting to communicate with the atmosphere. Figure 4(b) shows the times spent by the land processor receiving communications from the coupler. We find that corresponding to large computations in the atmosphere processor shown in Figure 4(a), there are large communication times in the land processor as seen in Figure 4(b).

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. Each atmosphere processor performs the physics calculations corresponding to a set of chunks. The chunk formation and assignment to processors are based on a load-balancing option set at compile time. For each chunk, as part of physics calculations, a call is made to the computationally intensive absorptivity calculating function, *radabs*, the source of temporal load imbalance. Inside *radabs*, the long-wave radiation calculations are performed for the columns that constitute the chunk. There are no dependencies between the calculations corresponding to any two columns.

4.2 Intra-Component Load Imbalance Resulting from Shortwave Radiation Calculations

The physics calculations for each chunk has a call to a fairly computation intensive short-wave radiation calculation function *radcswmx* before the call to *radabs*. This function, performing calculations only for the grid points in the day region, is a cause of very high intra-component load imbalance between the atmosphere processors. Although the standard 1-D decomposition of the grid along the longitudes results in the processors with latitudes close to the equator getting equal number of day and night grid points, those at the poles may have all grid points corresponding to day (or night) depending upon the season. This is illustrated by the graph in Figure 5 that shows the times spent by the atmosphere processors in *radcswmx* at different time steps. The graph also shows that the load imbalance between the atmosphere processors varies over time.

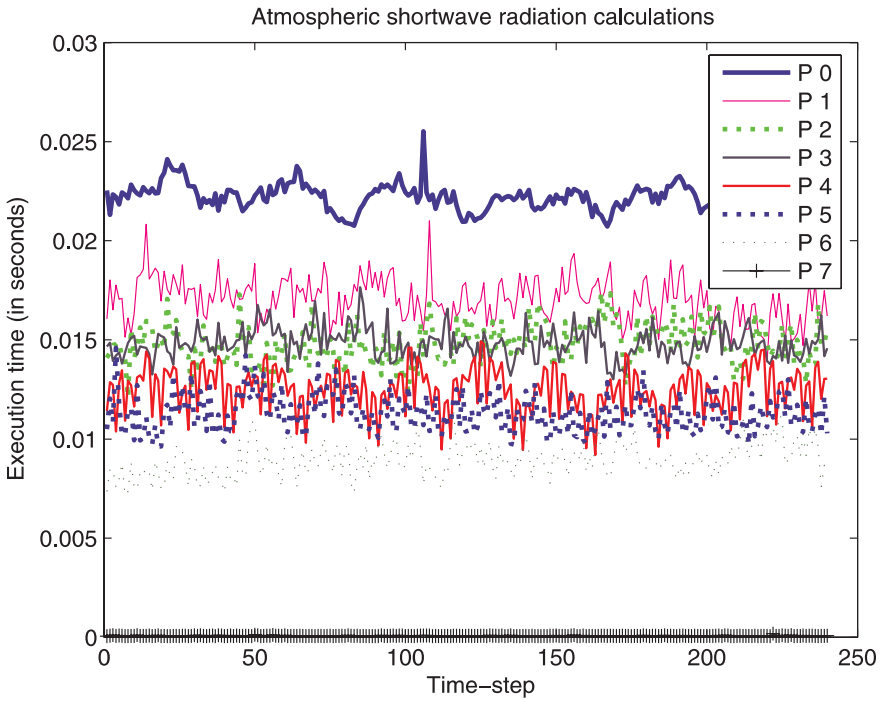


Fig. 5 Load imbalance resulting from *radcswmx*.

```
1  Algorithm:Pseudo-code for radiation calculations
2  for each time-step do
3  |  :
4  |  for 1 to nchunks do
5  |  |  :
6  |  |  if radcswmx time-step then
7  |  |  |  :
8  |  |  |  call radcswmx /* Shortwave radiation calculations.  Source of
   |  |  |     intra-component load imbalance                          */
9  |  |  |  :
10 |  |  |  if radabs time-step then
11 |  |  |  |  for 1 to ncol do
12 |  |  |  |  |  call radabs /* Longwave radiation calculations.  Source of
   |  |  |  |  |     temporal load imbalance in atmosphere and inter-component
   |  |  |  |  |     load imbalance                                       */
13 |  |  |  |  end
14 |  |  |  end
15 |  |  |  :
16 |  |  end
17 |  |  :
18 |  end
19 end
```

Fig. 6 Pseudo-code for radiation calculations.

CCSM allows a "load balanced" chunk formation scheme which addresses the load imbalance in the shortwave radiation calculations. However, the use of this scheme results in an increase in the time taken for remapping the dynamic grid to the physics grid at every time step. The pseudo-code involving short- and longwave radiation calculations is given in Figure 6. By default, the *radcswmx* time step is every 3 hours and the *radabs* time step is every 12 hours. However, the work by Pauluis and Emanuel (2004) describes the numerical instabilities that result from infrequent radiation calculations. In our calculations, we perform *radabs* calculations every 3 simulated hours without loss of generality.

5 Dynamic Extension of the Atmosphere Component

Our component extension method tries to address both the above imbalances without introducing delays in any of the components. We reduce the inter-component load imbalance by offloading the absorptivity calculations of atmosphere to idling processors of other components. The amount of offloading by each atmosphere processor is in turn determined from the amount of intra-component imbalance in shortwave radiation calculations. In our scheme, 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 absorptivity calculations on the columns it possesses. After calculations, the atmosphere processors receive the results corresponding to the columns it offloaded to other components.

The first step of communication that involves sending *radabs* input from atmosphere to other components is performed before shortwave radiation calculations. By placing the sends before *radcswmx*, the *radabs* calculations in non-atmosphere components can be overlapped with the shortwave radiation calculations in atmosphere and part of the intra-component load imbalance can also be reduced. In order to determine the amount of extension or the amount of sharing of *radabs* calculations of the atmosphere processor with each non-atmosphere proc-

essor, we periodically execute CCSM without modifications for a simulated day and collect the times when each processor of each component is ready to share *radabs* calculations. These ready times are then used for atmosphere extension or load balancing in other simulated days. Thus the extension is performed dynamically based on the ready times collected periodically. Three primary issues have to be addressed for extending longwave radiation calculations:

1. At what time steps should atmosphere offload radiation calculations to a non-atmosphere component?
2. In a given time step, should the *radabs* calculations of all the chunks be offloaded?
3. How many columns of a chunk should an atmosphere processor offload to each of the other non-atmosphere processors?

We perform a three-level extension strategy to address the above issues.

5.1 Time Step Level

At this level, the time steps (one time step = 20 simulated minutes) at which atmosphere radiation calculations can be extended to non-atmospheric processors have to be decided. Atmosphere radiation calculations cannot be offloaded from atmosphere to non-atmosphere processors at all time steps since the non-atmosphere processors may be performing their own calculations at certain time steps when the atmosphere processors are performing radiation calculations. This typically happens when the coupling frequency of non-atmosphere components is lower than the frequency of radiation calculations as in the case of ocean. Ocean communicates with coupler only once in 24 simulated hours whereas radiation calculations resulting from *radabs* occur every 3 simulated hours. Ocean processors may be busy with their own calculations for the first 12 simulated hours of the simulated day and may become idle only from the 13th simulated hour. In this case, the atmosphere processors will not be able to offload *radabs* calculations to ocean processors during simulated hours 3, 6, 9, and 12 and will be able to offload or share only from the 15th simulated hour.

While the coupling and radiation calculation frequencies are static, the time step at which a component is ready depends upon the number of processors allocated to the component and can vary dynamically. Hence, the ready time step has to be determined at run-time and updated periodically. As explained earlier, corresponding to each *radabs* call there is a send-ready time in the atmosphere and a recv-ready time in each non-atmosphere component. These ready times are determined by periodically executing unmodified CCSM for a simulated day and observing the times for various phases of execution. Our objective at this level is to determine the first time step of a simulated day for which some of the *radabs* calls can be offloaded to a non-atmosphere component without the atmosphere being delayed. This time step is the one immediately before the time step for which the send-ready time of atmosphere, corresponding to the first *radabs* call in the time step, is greater than the recv-ready time of the component, and is calculated by each non-atmosphere component.

The *radabs* time step after which offloading can take place is noted as the component's time step sharing index and is broadcast to all the processors. At every *radabs* time step of each simulated day of the simulation with offloading, the atmosphere includes in its destination processor set, all the processors of every component that has (a) a lower coupling frequency than atmospheric *radabs* frequency, and (b) a time step sharing index less than the current *radabs* time step.

5.2 Chunk Level

When the coupling frequency of a component is greater than or equal to the *radabs* frequency, the idling of the component occurs at every time step of atmospheric *radabs* calculation. Thus, the offloading of atmosphere to this component can occur at all *radabs* time steps. However, as seen in Figure 6, for a given time step, the *radabs* function is invoked for every chunk. Thus, for a given time step, the non-atmosphere component may become idle and may be able to share *radabs* with the atmosphere processors calculations only after the atmosphere processors have finished processing some chunks in the time step. Thus, the chunk at which *radabs* calculations can be shared for a given time step has to be precisely calculated.

For each time step, all processors maintain another array of send-ready times and recv-ready times, where each array element corresponds to a chunk within the time step. The array index where the send-ready time exceeds the recv-ready time determines the number of chunks for which radiation calculations can be offloaded at that time step. We calculate the maximum of these numbers, across the time steps, as the component's chunk-sharing index. Thus the chunk-sharing index is the starting chunk number from which the atmosphere can offload radiation calculations to processors of a particular component.

After the component root processor calculates its chunk-sharing index as outlined above, it broadcasts it to all processors. In the following simulated days, the atmosphere, when processing each chunk, includes in its destination processor set, all the processors of the component that has (a) a higher coupling frequency than the *radabs*

frequency, and (b) a chunk-sharing index less than the index of the current chunk in the current time step. Thus the points of extension to the coupler, land, and ice components are dynamically determined.

5.3 Column Level

This step involves the finest level of decision making. For a given time step and chunk, a set of non-atmosphere processors may be ready to share *radabs* calculations. However, the number of columns within the chunk that will be shared by each atmosphere processor with each non-atmosphere processor has to be calculated. As mentioned earlier, the times at which each atmosphere processor enters *radabs* calculation can vary between atmosphere processors because of differences in times taken by the processors for shortwave radiation calculations. Thus, the shortwave radiation calculation time of each processor should be taken into account to calculate the number of columns that each atmosphere processor can offload to each of its destination non-atmosphere processor, so that all atmosphere and non-atmosphere processors finish the *radabs* calculations at about the same time.

Since the input data for *radabs* is sent before the shortwave radiation calculations, the time taken for the shortwave radiation calculations by each atmospheric processor in the previous time step is used to determine the availability times of the processor for performing *radabs* calculations. In each *radabs* time step consisting of multiple *radabs* calls as shown in Figure 6, these values are sent to a non-atmosphere component by an atmosphere processor just before the first *radabs* call for which the component processors are included in the atmosphere processor's destination processor set. Based on these availability times, a component processor determines the number of columns in the current chunk that it will process and the source atmosphere processors containing these columns.

Each processor that is involved in the extension for the current *radabs* time step calls a column-processor mapping algorithm, with the array of availability times, the number of atmosphere processors, and the number of other processors as inputs, and obtains as results the details of the communications it has to perform during the extension. The mapping is determined in two steps. The first step determines the number of columns to be given to each processor. For each column of the total available columns, we determine the processor which will complete processing it in the minimum time and assign the column to the processor. The availability time of this processor is now updated to its expected completion time. In the first step, we had assumed that all the columns are available as a single pool. However, some of the participating processors (atmosphere processors) already possess some columns. Therefore, in the second step, a mapping for senders to receivers has to be determined. The number of columns a processor has to send or receive is the difference between the number of columns it originally possesses and the number of columns assigned to it in the first step. A simple greedy technique is followed to determine the processor communications. While there are senders or receivers, we transfer the maximum number of columns from the current sender to the current receiver, update the sender's and receiver's remaining number of columns to be communicated, and if the sender and/or receiver meets its target number of columns, we move to a new sender and/or receiver. Note that the maximum number of columns in a transfer is the minimum of the number of columns that a sender can send and the number of columns a receiver can receive. Using this strategy, each processor determines the number of columns (amount of extension) and the processors (targets of extension) with which it has to communicate. For each chunk, it then involves in input communications, *radabs* computations, and output communications, using the same mapping until the number of sharing processors changes.

The pseudo-codes of the modified algorithm described in this section are given in Figures 7 and 8, and can be compared with the pseudo-code of the original version given in Figure 6.

5.4 Summary of Dynamic Component Extension in CCSM

We dealt with several key aspects of the dynamic component extension strategy while extending the atmosphere component of CCSM to reduce the temporal load imbalance in the atmosphere component resulting from computationally intensive longwave radiation calculations. We first identified atmosphere as the component containing temporal load imbalance by conducting profiling runs. We then modified the codes of the atmosphere and other components to share parts of the heavy computations in atmosphere. By using a three-level load-balancing strategy, we dynamically determined various parameters of extension at different granularities. By periodically executing the CCSM application without our load balancing and obtaining the times spent by the different components in different phases, we automatically and dynamically identified the idling periods in processors of other components and the times when the component containing temporal load imbalance is ready to extend its heavy computations to other component processors. At periodic intervals, we turn off (using a conditional-if statement) the load balancing part of the code and turn on the profiling part of the code. After executing in this phase for one simulated day, we turn off the profiling and turn on the load balancing parts of the code. The load-balancing strategy will then be using the profiled information for all simulated days until

```
1  Algorithm:Modified algorithm
2  for each time-step do
3  |  atmosphere model execution
4  |  for each chunk do
5  |  |  if radabs time-step then
6  |  |  |  determine/update destination processor set
7  |  |  |  if change in destination processor set or in time-step then
8  |  |  |  |  send availability times to new component(s); calculate column mapping.
9  |  |  |  end
10 |  |  |  send radabs input data
11 |  |  |  availability timer start
12 |  |  end
13 |  |  if radcswmx time-step then
14 |  |  |  call radcswmx
15 |  |  end
16 |  |  if radabs time-step then
17 |  |  |  availability timer stop
18 |  |  |  call radabs with own column range; receive radabs output data
19 |  |  end
20 |  end
21 |  compute average availability time
22 |  atmosphere model execution
23 end
```

Fig. 7 Pseudo-code for modified atmosphere component.

```
1  Algorithm:Modified algorithm for components
2  for each time-step do
3  |  component model execution
4  |  if radabs time-step then
5  |  |  recv availability times from atmosphere
6  |  |  determine number of chunks using chunk sharing index
7  |  |  for each chunk do
8  |  |  |  determine/update destination processor set
9  |  |  |  if change in destination processor set or in time-step then
10 |  |  |  |  calculate column mapping.
11 |  |  |  end
12 |  |  |  recv radabs input data; call radabs with own column range; send radabs output data
13 |  |  end
14 |  end
15 |  communication with coupler
16 |  component model execution
17 end
```

Fig. 8 Pseudo-code for modified non-atmosphere component.

Table 1
Platform specifications.

Platform name	Number of procs.	Specifications
Fire-16	16	8 dual-core AMD Opteron 1214 based 2.21 GHz Sun Fire servers, CentOS release 4.3, 2 GB RAM, 250 GB hard drive and connected by Gigabit Ethernet
Fire-48	48	12 × 2 dual-core AMD Opteron 2218 based 2.64 GHz Sun Fire servers, CentOS release 4.3, 4 GB RAM, 250 GB hard drive and connected by Gigabit Ethernet
Regatta	16	AIX, 16-way SMP IBM pSeries 690 node with 16 POWER4 CPUs running at 1.3 GHz and 160 GB memory with IBM AIX
Regatta	32	32-way SMP IBM pSeries 690 node with 34 POWER4 CPUs running at 1.1 GHz and 64 GB memory with IBM AIX
Param	216	AIX, 54 4-way SMP IBM pSeries 630 nodes, each having 4 POWER4 CPUs running at 1 GHz and 8 GB memory with IBM AIX

the next profiling phase. Thus we dynamically determine the load imbalances across components and the source of the imbalances using profiling phases of unmodified CCSM at periodic intervals. The idling periods and ready times were then used to determine the points, amounts and target processors of extension. Similar steps can be followed when applying our dynamic component extension strategy to other multicomponent applications to improve the application performance.

6 Experiments and Results

We used five different platforms to evaluate our strategy and compare its performance with that of the original version of CCSM. The specifications of these platforms are given in Table 1. The first three systems are located in the Indian Institute of Science (IISc) while the last two systems are located in the Centre for Development of Advanced Computing (CDAC), Bangalore, India. On CDAC Param, out of the 54 systems, we used 16 systems of 64 processors for our experiments.

All the experiments were conducted with NCAR's most recent release of CCSM, CCSM3.0.1beta14. We have ensured that our modifications do not affect the scientific validity of the results, and verified that the results of the modified version and those of the original version match bit-by-bit. CCSM has an option for the users to enable a load-sharing mechanism for the shortwave radiation calculations. On each platform, we executed four versions of CCSM on various numbers of processors. The four versions are: (a) the original version which by default has the shortwave load balancing turned off, (b) the original version with shortwave load balancing turned on, (c) our modified version with the shortwave load balancing turned off, and (d) our modified version with shortwave load balancing turned on. Each of our experiments corresponded to a 30-day climate simulation. For each experiment, execution times were noted for each of the four versions. We mostly show results for versions (a) and (c). Similar results were obtained for versions (b) and (d).

Table 2 shows the percentage reduction in execution time of *radabs* as a result of dynamic component exten-

Table 2
Results on various platforms.

Platform	Procs.	Original *radabs* time (s) [version (a)]	DCE *radabs* time (s) [version (c)]	Gain in *radabs* due to DCE	Savings for a 1000-year simulated run
Fire-16	16	355.06	268.68	24%	13.63 days
Fire-48	48	144.598	102.676	29%	4.37 days
IISc Regatta	16	515.71	360.82	30%	19.3 days
CDAC Regatta	32	307.94	250.28	19%	3.02 days
CDAC Param	32	338.62	274.69	19%	4.36 days

Table 3
Results for the 16-processor configuration (Atmosphere-8,Ocean-4,Ice-2,Land-1,Coupler-1) on various platforms.

Platform	Original execution time [version (a)]	DCE execution time [version (c)]	Percentage gain	Savings for a 1000-year simulated run
Fire-16	1315.426	1112.820	15%	28.14 days
Fire-48	1129.155	964.375	15%	22.89 days
IISc Regatta	1750.586	1459.620	17%	40.41 days
CDAC Regatta	2056.061	1704.762	17%	48.79 days
CDAC Param	2477.55	2095.74	15%	53 days

sion (DCE) for different platforms and different numbers of processors.[7] The last column denotes the savings in execution time of CCSM for a 1000-year simulated run and is calculated by multiplying the savings obtained for a 30-day simulated run by 12,000. The results in the last column are significant because CCSM is commonly executed for such multi-century runs. We find that using our strategy can lead up to 30% reduction in execution time of *radabs* and result in savings of 3–19 days in execution time of CCSM for a 1000-year simulations.

In order to compare the gain resulting from DCE gain across all platforms, we executed CCSM on 16 processors, the largest number of processors available on all platforms. The results on various platforms for executions on 16 processors are shown in Table 3.[8] The results show that the percentage gains are almost constant across all platforms.

As mentioned earlier, the frequency of radiation computations is once every 3–12 hours of simulation. Latest studies (Pauluis and Emanuel 2004) show that radiation computations should be done as frequently as possible, preferably at the same frequency as the rest of the model. Figure 9 shows the effect of increasing radiation frequency on various execution times for a 30-day simulation obtained on the Fire-16 cluster with the same configuration as in Table 3. As can be seen in the figure, dynamic component extension results in decrease of *radabs*, physics, and total execution times in all cases. Another interesting observation was that while the DCE execution times are better than the original execution times for all frequencies, the gain is significantly higher for the highest frequency. Our investigations showed that this is because more radiation calculations were extended to other component processors for large radiation frequencies.

Figure 10(a) illustrates the gain obtained with dynamic component extension in CCSM. The result was obtained with 30-day runs on the IISc Regatta cluster with the same configuration as in Table 3. The figure shows that

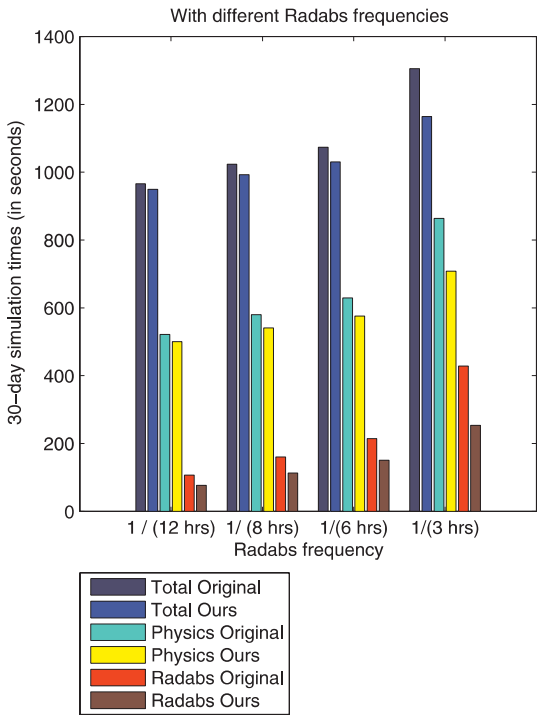
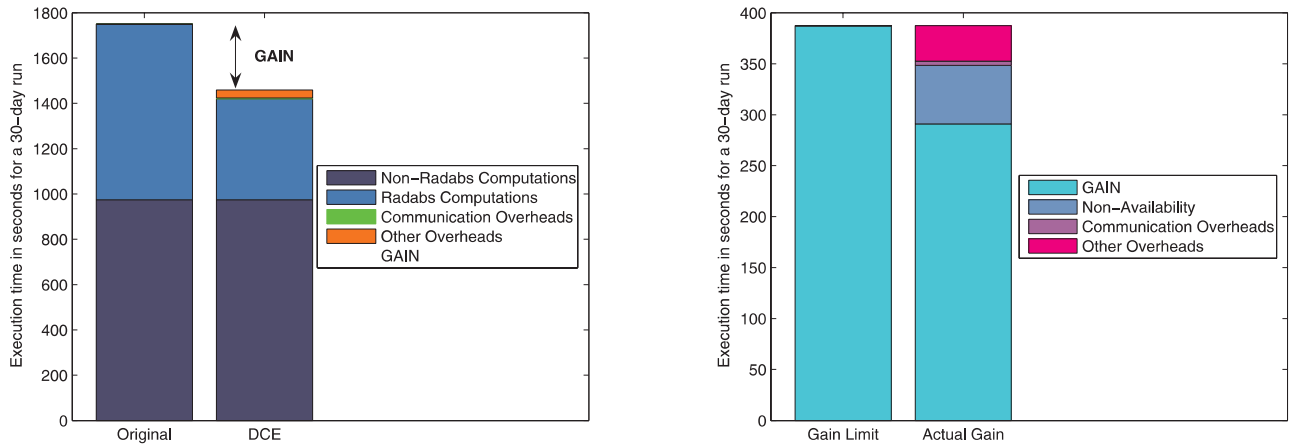


Fig. 9 Effect of radiation frequency on DCE gain. Fire-16 cluster.

the dynamic component extension during the *radabs* (absorptivity) calculations reduces the execution time of *radabs* calculations by 17%. The figure also shows that the overheads resulting from dynamic component extension are negligible. These overheads include communication of columns and availability times from the atmosphere to component processors, time stamping, computation of availability times, and mapping columns to processors.



(a) Illustration of gain

(b) Actual vs. maximum gain

Fig. 10 Overheads and gain resulting from DCE. IISc Regatta cluster.

Figure 10(b) compares the actual gain obtained as a result of dynamic component extension to the maximum gain for the same configuration and processors. This maximum gain is obtained using simplistic assumptions for extending *radabs* calculations, namely, the entire *radabs* calculations in the atmosphere processor can be extended to the other components and there are zero overheads for extension. The maximum gain is obtained using equation (2) in Section 2. The figure shows that our implementation of dynamic component extension in CCSM is efficient and achieves about 74% of the maximum achievable gain. We also find that the primary reason for not achieving the maximum gain is the non-availability of all components for extension at all *radabs* points.

Based on experimental timings and scalability of various components, we have built an execution model for CCSM. Using this model, we performed simulations to find the execution times for the original and the DCE version and compute the percentage gain resulting from DCE for various application configurations. As mentioned earlier, the CCSM application consists of a dynamics and a physics phase in each time step. Our DCE strategy in this work was applied to the physics phase. Current efforts by the CCSM community (Mirin and Sawyer 2005; Mirin and Worley 2007) are to increase the scalability of dynamics phase. As dynamics becomes more scalable, the percentage execution time of the physics phase increases. Thus our DCE strategy that optimizes the physics execution time will lead to increasing reduction in overall execution time of CCSM. This is illustrated in Figure 11 where the overall gain for the application is shown for 64 and 128 processors for increasing scalability of dynamics.

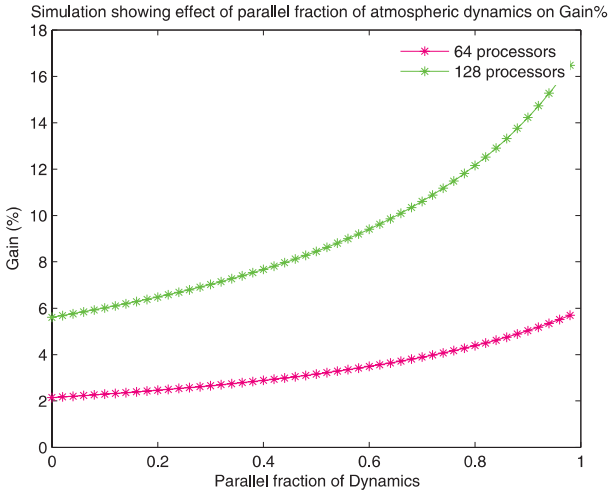


Fig. 11 Simulation results showing effect of scalability in dynamical core on DCE gain.

7 Conclusions

In this work, we proposed a software strategy for multi-component applications called dynamic component extension for minimizing processor idling and improving application performance. We demonstrated the potential of this strategy by applying it to improve the perform-

ance of the most prominent multicomponent application, CCSM. By automatic identification of idling points and dynamic determination of points and amounts of extension, we extended the computation intensive longwave radiation calculations of the atmosphere component to the processors executing other components. By evaluating our strategy on five different parallel platforms for different number of processors, we showed that our strategy can reduce CSSM execution times by about 15% and save several days of execution for 1000-year simulation runs. We also showed that the overheads arising from our strategy are negligible and that our strategy can give increasing benefits with increasing frequency of radiation calculations and increasing scalability of dynamics phase of CCSM. Our strategy can enable climate researchers to conduct multi-century climate runs in a reasonable time even on the moderately sized clusters common in academic institutions.

8 Future Work

The CCSM community has been advocating and working toward increasing the frequency of radiation calculations, adopting highly scalable dynamic cores, and using larger resolutions. These will lead to larger percentages of execution times for longwave radiation calculations and increased significance of our dynamic component extension strategy in optimizing the radiation calculations. We plan to empirically evaluate our strategy for these various scenarios as they become available. We also plan to devise methodologies for efficient execution of CCSM on heterogeneous systems, namely, grid systems.

Acknowledgments

The authors would like to thank the Centre for Development of Advanced Computing (CDAC) for allowing us to experiment on their high performance systems and approving our requests for dedicated resources. The authors are also grateful to the anonymous reviewers for providing valuable comments for improving the quality of the paper. This work was supported by Ministry of Information Technology, India, project reference number DIT/R&D/C-DAC/2(10)/2006 DT.30/04/07.

Author Biographies

Sivagama Sundari M. received her B.E.(Hons) in electronics and instrumentation engineering from the Birla Institute of Technology and Science, Pilani, in 2005. She is now pursuing a Ph.D. at the Supercomputer Education and Research Centre at the Indian Institute of Science, Bangalore. Her current research interests include software architectures and performance of scientific applications, grid computing, and high performance computing.

Sathish Vadhiyar is an assistant professor in the Supercomputer Education and Research Centre, Indian Institute of Science. He obtained his B.E. degree in the Department of Computer Science and Engineering at Thiagarajar College of Engineering, India, in 1997 and received his Masters degree in computer science at Clemson University, USA, in 1999. He graduated with a Ph.D. in the Computer Science Department at University of Tennessee, USA, in 2003. His research areas are in parallel and grid computing with primary focus on performance modeling of parallel applications, scheduling and rescheduling methodologies for grid systems, and grid applications. Dr Vadhiyar is a member of IEEE and has published papers in peer-reviewed journals and conferences. He was a tutorial chair and session chair of escience 2007 and served on the program committees of conferences related to parallel and grid computing including IPDPS, CCGrid, eScience, and HiPC.

Ravi S. Nanjundiah is a professor at the Centre for Atmospheric & Oceanic Sciences (CAOS), Indian Institute of Science (IISc). He obtained his B.E. degree in mechanical engineering from Rani Durgavati University (Jabalpur, India) in 1984, his M.E. in mechanical engineering from IISc in 1986, and his Ph.D. in atmospheric science from IISc in 1992. His research areas are the study of monsoons – their variability and change – using climate system models, and application of HPC/grid computing to climate system modeling. He has published papers in peer-reviewed journals and conferences. He is an associate editor of Journal of Earth System Science.

Notes

1 http://www.cgd.ucar.edu/pcm

2 http://www.ccsm.ucar.edu

3 www.mcs.anl.gov/foamA

4 http://www.ncar.ucar.edu

5 http://www.ccsm.ucar.edu/models/atm-cam/

6 Message Passing Interface 2, www.mpi-forum.org

7 CCSM does not scale well for larger numbers of processors since the maximum number of processors on which the atmosphere model can execute is limited by the number of latitudes (48 for low resolution).

8 Results for 16 processors for Fire-16 and IISc Regatta are different in Table 2 and 3 because of the different CCSM configurations used. CCSM configuration of (Atmosphere-10,Ocean-2, Ice-2,Land-1,Coupler-1) was used for the corresponding results in Table 2.

References

Carr, G. (2005). An introduction to load balancing CCSM3 components. In *Proceedings of Software Engineering Working Group (SEWG) Meeting, CCSM Workshop, NCAR*.

Carr, G., Cordery, M., Drake, J., Ham, M., Hoffman, F., and Worley, P. (2005). 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.

Chow, P. and Addison, C. (1999). An overview on current multiphysics software strategies for coupled applications with interacting physics on parallel and distributed computers. In *Proceedings of the 11th International Conference on Domain Decomposition Methods in Sciences & Engineering*.

Collins, W., Bitz, C., Blackmon, M., Bonan, G., Bretherton, C., Carton, J., Chang, P., Doney, S., Hack, J., Henderson, T., Kiehl, J., Large, W., McKenna, D., Santer, B., and Smith, R. (1998). The community climate system model: CCSM3.

Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li., P., Yang, W., Hill, C., and da Silva, A. (2005). Design and implementation of components in the Earth system modeling framework, *International Journal of High Performance Computing Applications*, **19**(3): 341–350.

Coveney, P., Fabritiis, G. D., Harvey, M., Pickles, S., and Porter, A. (2005). On steering coupled models. In *e-Science All Hands Meeting*.

Delgado-Buscalioni, R., Coveney, P., Riley, G., and Ford, R. (2005). Hybrid molecular-continuum fluid models: Implementation within a general coupling framework, *Philos. Trans. Roy. Soc. London Ser. A*, **363**.

Dorneles, R., Rizzi, R., Diverio, T., and Navaux., P. (2003). Dynamic load balancing in PC clusters: An application to a multiphysics model. In *Proceedings of 15th Symposium on Computer Architecture and High Performance Computing*, pp. 192–198.

Drake, J., Foster, I., Michalakes, J., Toonen, B., and Worley, P. (1995). Design and performance of a scalable parallel community climate model, *Parallel Computing*, **21**(10): 1571–1592.

Ford, R. and Burton, P. (1998). Load balancing physics routines. In *Proceedings of the 8th ECMWF Workshop on the Use of Parallel Processors in Meteorology*.

Foster, I. and Toonen, B. (1994). Load balancing algorithms for climate models. In *Proceedings of the Scalable High Performance Computing Conference*, pp. 674–681.

Larson, J. (2006). Some organising principles for coupling in multiphysics and multiscale models. In *Computational Techniques and Applications Conference (CTAC'06)*.

Larson, J., Jacob, R., and Ong, E. (2005). The Model Coupling Toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models, *International Journal of High Performance Computing Applications*, **19**: 277–292.

Lawrence, C. (1997). An overview of three approaches to multidisciplinary aeropropulsion simulations. Technical Report NASA TM-107443, NASA.

Lefantzi, S. and Ray, J. (2003). A component-based scientific toolkit for reacting flows. In *Proceedings of the 2nd MIT Conference on Computational Fluid and Solid Mechanics*, pp. 1401–1405.

Lermusiaux, P., Evangelinos, C., Tian, R., Haley, P., McCarthy, J., Patrikalakis, N., Robinson, A., and Schmidt, H. (2004). Adaptive coupled physical and biogeochemical ocean predictions: A conceptual basis. In *Computational Science, International Conference on Computational Science (ICCS) 2004*, edited by F. Darema.

Michalakes, J. (1991). Analysis of workload and load balancing issues in the NCAR community climate model. Technical Report ANL/MCS-TM-144, Argonne National Laboratory, Argonne, Illinois.

Michalakes, J. and Nanjundiah, R. (1994). Computational load in model physics of the parallel NCAR community climate model. Technical Report ANL/MCS-TM-186, Argonne National Laboratory, Argonne, Illinois.

Michalakes, J., Canfield, T., Nanjundiah, R., Hammond, S., and Grell, G. (1994). Parallel implementation, validation, and performance of MM5. In *Parallel Supercomputing in Atmospheric Science*. River Edge, NJ: World Scientific.

Michalakes, J., Dudhia, J., Gill, D., Henderson, T., Klemp, J., Skamarock, W., and Wang, W. (2004). The weather research and forecast model: Software architecture and performance. In *The Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology*.

Mirin, A. and Sawyer, W. (2005). A scalable implementation of a finite-volume dynamical core in the community atmosphere model, *International Journal of High Performance Computing Applications*, **19**(3).

Mirin, A. and Worley, P. (2007). Extending scalability of the community atmosphere model. In *Journal of Physics: Conference Series, 78 (Proceedings of SciDAC 2007)*, Boston, MA.

Muszala, S., Alaghband, G., Connors, D., and Hack, J. (2004). A VFSA scheduler for radiative transfer data in climate models. In *17th International Conference on Parallel and Distributed Computing Systems*.

Muszala, S., Connors, D., Hack, J., and Alaghband, G. (2006). The promise of load balancing the parameterization of moist convection using a model data load index, *Journal of Atmospheric and Oceanic Technology*, **23**(525b).

Nanjundiah, R. (1998). Strategies for parallel implementation of a global spectral atmospheric general circulation model. In *5th International Conference on High Performance Computing*.

Nanjundiah, R. (2006). Seasonal simulation of the monsoon with the NCMRWF model, *Current Science*, **78**: 869–875.

Pauluis, O. and Emanuel, K. (2004). Numerical instability resulting from infrequent calculation of radiative heating, *Monthly Weather Review*, **132**: 673–686.

Toth, G., Volberg, O., Ridley, A., Gombosi, T., DeZeeuw, D., Hanson, K., Chesney, D., Stout, Q., Powell, K., Kane, K., and Oehmke, R. (2003). A physics-based software framework for Sun-Earth connection modeling. In *Multiscale Coupling of Sun-Earth Processes, Proceedings of the Conference on the Sun-Earth Connection*, pp. 383–397.