GPU-enabled Efficient Executions of Radiation Calculations in Climate Modeling

Sai Kiran Korwar[1], Sathish Vadhiyar[1], Ravi S Nanjundiah[2]
1 Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India.
[2] Centre for Atmospheric & Oceanic Sciences , Indian Institute of Science, Bangalore, India.
sai@ssl.serc.iisc.in, vss@serc.iisc.in, ravi@caos.iisc.ernet.in

***Abstract*—In this paper, we discuss the acceleration of a climate model known as the Community Earth System Model (CESM). The use of Graphics Processor Units (GPUs) to accelerate scientific applications that are computationally intensive is well known. This work attempts to extract the performance of GPUs to enable faster execution of CESM and obtain better model throughput. We focus on two major routines that consume the largest amount of time namely, *radabs* and *radcswmx*, which compute parameters related to the long wave (infra-red) and short wave (visible and ultra-violet) radiations respectively. We propose a novel asynchronous execution strategy in which the results computed by the GPU for the current time step are used by the CPU in the subsequent time step. Such a technique effectively hides computational effort on the GPU. By exploiting the parallelism offered by the GPU and using asynchronous executions on the CPU and GPU, we obtain a speed-up of about 26× for the routine *radabs* and about 5.6× for routine *radcswmx*.**

I. Introduction

The importance of climate to energy usage and agriculture has made it a prominent field of study. Climate models predict temperature changes, winds, radiation, relative humidity and other such factors. Climate study is carried out based on the mathematical models of the physical processes. These models generally have a lot of computational intensive routines that require a lot of computing power.

Climate models simulate the interaction of the various components of the climate such as atmosphere, land, ocean and ice. These different components interact with each other by the exchange of energy, momentum and matter. There are a variety of models that vary in their degree of complexity. Physical processes like radiation, circulation and precipitation interact with chemical and biological processes to form a complex dynamic system. The length and time scales involved are diverse. The climate models use the equations of conservation of mass, momentum, energy and species to model the various components of the climate. Numerical methods are used extensively to solve these equations.

A. Model Details

One such climate model is the Community Earth System Model [1]. This is developed and maintained by the National Center for Atmospheric Research (NCAR). CESM consists of five geophysical component models, i.e., atmosphere, land, ocean, sea-ice and land-ice. There is also a coupler component that coordinates the interaction and the time evolution of the component models as shown in Figure 1 .

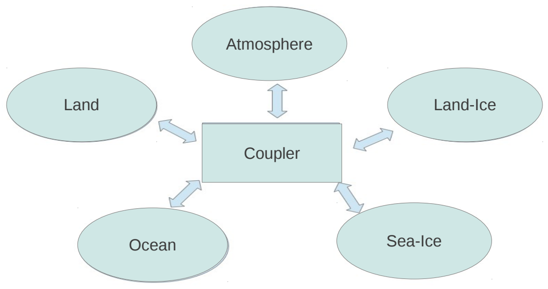

Fig. 1. CESM Model Interactions

The model used for atmosphere is Community Atmosphere Model (CAM) [2]. CAM consists of two computational phases, namely, *dynamics* and *physics*. The *dynamics* advances the evolutionary equations for the flow of atmosphere and the *physics* approximates sub-grid phenomena including clouds, long and short wave radiations, precipitation processes and turbulent mixing. The default core for the *dynamics* is a finite volume method that uses a longitude × latitude × vertical level computational grid over the sphere. The *physics* in CAM is based upon vertical columns whose computations are independent from each other. The parallel implementation of the *physics* is based on the assignment of columns to MPI processes and using OpenMP threads within a process to compute the columns assigned to a process.

CESM can be run with a variety of combinations for the different components. Each of the components may be either active, data, dead or stub and the choice of model physics and dynamics can be prescribed, leading to a combination called *compset*. There are a variety of *compsets* that are supported by CESM. The CESM grids can also be specified by setting an overall model resolution. The various resolutions used in this study are shown in table (I). The number of processors to be used overall and for each component can also be set individually.

B. Use of Computational Accelerators in HPC

Graphics Processing Units (GPUs) have been used to accelerate computationally intensive scientific applications in different domains including molecular dynamics [7], computational finance [4], computational chemistry [5], fluid dynamics [8], medical imaging [6] and climate science [9]. GPUs provide

978-1-4799-0730-4/13/$31.00 ©2013 IEEE

high compute performance per unit of power consumption, thus providing attractive options for future exascale computing systems. GPUs are used as accelerator devices that offload intensive computations from the host CPUs. The GPUs use a SIMT (Single Instruction Multiple Thread) instruction model where large number of threads perform the same operation on different data. Applications like climate science that use numerical methods can exploit this execution model effectively. This heterogeneous environment can be used effectively if both the CPU and the GPU perform computations asynchronously. This is an important idea that can lead to large magnitude of speed-ups in the applications. Thus the use of SIMT parallelism along with MPI and OpenMP will provide significant performance benefits for climate models.

C. *Execution Profile of CAM*

Figure 2 illustrates the execution profile of CAM. This figure shows that the physics routines that compute the long and short wave radiations are the most time consuming, taking about 32% of the total time spent in the atmosphere routines. The routine *radabs* takes the largest percentage of the overall run time when compared to any other single routine. *Radabs* computes the absorptivities for gases like $H_2O, CO_2, O_3, CH_4, N_2O, CFC11$ and $CFC12$. By default, this is run only every twelve simulated hours because of the expensive computations within this routine. Ideally we would like to run this routine at least once every hour (as water vapour concentration can change significantly on the diurnal scale and significantly change the emissivity and absorptivity of the atmosphere) without unduly increasing the overall time spent in radabs.

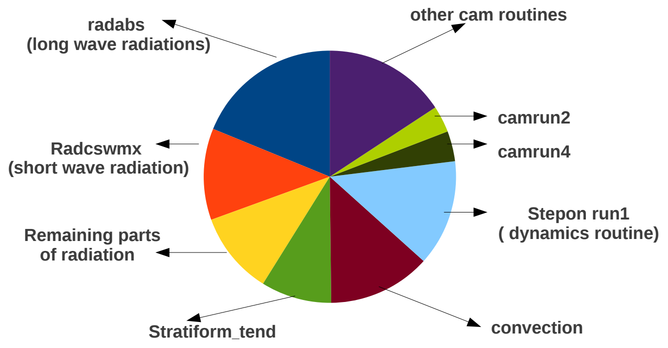

Fig. 2. Execution Time Division of CAM Routines

The next most time consuming routine of the atmosphere was found to be the short-wave radiation routine, *radcswmx*. The purpose of this routine is to compute the heating fluxes due to the solar radiation. It is based upon the Delta-Eddington approximation for solar radiation [3]. The frequency of execution of this routine is 1 hour.

D. *Present Work*

In this work, we have accelerated the CAM model of CESM by asynchronous executions of the radiation routines, *radabs* and *radcswmx*, on GPUs along with the executions of other steps of the model on CPU cores. We have developed a novel asynchronous model in which the result of the asynchronous executions of these routines in a time step are used in the next time step, thereby almost providing *zero-times* for the routines. We have conducted experiments on single core with single GPU, multi-core OpenMP execution with single and multiple GPUs, and multi-node multi-core MPI-OpenMP executions with multiple GPUs. Our experiments show 1.03×-2.3× improvement in the execution times of the entire model with the utilization of GPUs.

Section 2 presents related work on accelerating climate models on GPU and gives the relevance of our work. Section 3 describes the methodology used to accelerate radiation routines in CESM. In Section 4 we present the various experiments and results related to speedups obtained due to acceleration and asynchronous executions of the two physics routines. The last section presents conclusions and gives scope for future work.

II. Related Work and Project Relevance

There have been many efforts to port various climate models to the GPU. Some of the major efforts are discussed in this section. The weather and climate models benefited from increasing processor speed for a large number of years. With the limit on the clock rate being nearly achieved, emerging architectures are being used to improve model performance. There have been many efforts to port various climate models to the GPUs, as GPUs have become popular as high performance computing tools. In this section we discuss some of the previous efforts to accelerate climate models using GPUs.

The use of the fine grained parallelism of GPUs to enhance the performance of the Weather Research and Forecast (WRF) model resulted in a 5-20x speed-up for the computationally intensive routine WSM5 [9]. The WRF single moment 5-tracer (WSM5) which is part of the physics routine was chosen to be ported on the GPU. The Fortran code was converted into CUDA by the use of compiler directives that were developed by the authors. The NVIDIA GTX-8800 GPU used for the WSM5 micro-physics kernel gave a performance of 65 GFlops.

Another attempt at using GPUs for performance in climate modelling was made at the Earth System Research lab (ESRL) in NOAA. The Non-hydrostatic Icosahedral (NIM) model was ported to the GPU [10]. The approach was to run the entire model on the GPU and use the CPU only for model initialization and communication. They developed the Fortran-to-Cuda (F2C-ACC) compiler to convert the Fortran code into CUDA. As most climate models are written in Fortran and there are only a few compilers that support CUDA-Fortran, this tool is extremely useful to accelerate the GPU porting. It does not support all the Fortran 90 constructs and the code must be analyzed and modified by hand. The dynamics portion which is the most expensive part of the NIM model was accelerated using GPU and the speed-up achieved was about 34 times on Tesla - GTX-280 when compared to a CPU (Intel Hapertown).

The Oak Ridge National Laboratory (ORNL) ported the Spectral element Dynamical core of CESM (HOMME) to the GPU [11]. A very high resolution of (1/8) th degree was used as a target problem. Using asynchronous data transfer, the most expensive routine performed three times faster on the GPU than the CPU. This execution model was shown to be highly scalable.

The Swiss Federal Institute of Technology ported to the GPU a model called Consortium for Small scale Modelling (COSMO) which is used for Swiss weather prediction [13]. They used OpenACC directives to accelerate the compute bound physics routines and rewrote the dynamics portion which is memory bound. The speed-up achieved varied from 2-7 for various routines.

The climate model ASUCA is a production weather code developed by the Japan Meteorological Agency [12]. By porting their model fully onto the GPU they were able to achieve 15 TFlops in single precision using 528 GPUs. The TSUBAME 1.2 supercomputer in Tokyo Institute of Technology was used to run the model. It is equipped with 170 NVIDIA Tesla S1070 GPU nodes making a total of 680 GPUs. The CPU is used only for initializing the models and all the computations are done on the GPU. There are different kernels for the different computational components.

From these efforts it is clear that the use of GPUs for high resolution climate models is a necessity. We have chosen the CESM model for acceleration using GPUs. Most of the work to accelerate climate science routines have been concentrated on the dynamics. In this work, we choose to accelerate the physics routines as they were observed to be more expensive. Most of earlier works on climate models do not attempt to use asynchronous executions on the CPU and the GPU to avoid CPU idling and increase the model throughput.

There have been efforts on asynchronous executions in other scientific domains including linear algebra [16], N-Body simulations [15], and Adaptive Mesh Refinement applications [17]. However, our asynchronous model of executions of climate modeling routines is unique since it uses the domain-specific knowledge that the results of certain computations performed in a time step need not be used in the same time step. These computations can then be spawned off for GPU executions, while the CPU proceeds with the rest of the computations and the results of the GPU computations can be used in the next time step.

III. GPU Implementation

The main objective of this work is to accelerate the computation intensive routines of CESM that are most expensive in terms of their execution times. The F compset which is the default stand-alone CAM was used. As described earlier these routines were identified by profiling the CESM model using the HpcToolkit profiler [14]. The default dynamical core for the model is the finite volume dynamical core. The physics component of the atmosphere mainly consists of the stratiform, convection and radiation routines. As can be seen from the profiling of CAM (fig 2), the most expensive routines are the radiation routines.

The computational grid for the atmosphere is divided into latitudes, longitudes along the x and y direction and vertical columns along the z direction. The vertical columns extend from the surface up through the atmosphere. The columns are further divided into layers. The characteristic feature of the physics routines is that the computations are vertically independent, i.e., in the z direction. Every column can be computed independent of the other columns giving rise to data parallelism that can be exploited on the GPU. For the purpose of load balancing among the different CPU processes, the columns are grouped into chunks. A chunk is a unit of execution containing a configurable number of columns. A chunk may or may not contain contigous set of vertical columns i.e. columns from neighbouring grid points may not be in a single chunk. This method of combining vertical columns into chunks is used to remove computational imbalances in shortwave radiation routines **??**. These chunks are distributed among the MPI processes and the OpenMP threads. Every physics routine is called for each chunk. The psuedo-code for physics calculations is shown below:

Algorithm 1 Pseudo code for physics calculations

```
for every time step do
    for each chunk do
        call stratiform routine
        call convection routine
        ...
        call radiation routine
        ...
    end for
end for
```

The two most important components of the radiation routine are the *radcswmx* and *radclwmx* subroutines that compute short wave and long wave heating rates respectively. The *radclwmx* subroutine uses broad band absorptivity/emissivity method to compute clear skies and assumes randomly overlapped clouds with variable cloud emissivity to include effects of clouds. It computes the clear sky absorptivity/emissivity at a frequency lower than the model radiation frequency as these computations are expensive. The functions that calculate the absorptivities and emissivities for different gases are *radabs* and *radems* respectively. Out of the two it was observed that *radabs* was the most time consuming routine in-spite of having a lower frequency of calculation. So this was the routine that was selected to be ported to the GPU. *Radcswmx* was the next most time consuming routine. It computes the heating rates due to solar radiation. The following two sections discuss these two routines and the strategies used to port them onto the GPU.

A. Acceleration of Long Wave Radiation

Radabs consumes 10% of the total time spent in the atmosphere run. In the default execution mode it is run at a 12

Algorithm 2 Pseudo code for radiation calculations

```
for every time step do
    for each chunk do
        if do short-wave heating this time step then
            call radcswmx
        end if
        if do long-wave heating this time step then
            if do absorp/emiss calc this time step then
                call radabs
                call radems
            end if
            Long-wave radiation computation
        end if
    end for
end for
```

hour frequency. A simple strategy to parallelize this routine is to let each column of a chunk be computed by a single thread. Hence we can form a GPU kernel that has the same number of threads as the number of columns in a chunk. For each chunk, the *radabs* kernel computes the absorptivities for all the columns. It is clear that in this strategy, the GPU parallelism is limited by the number of columns in a chunk, thereby giving low GPU occupancy. Since this is a configurable parameter, we attempted to increase the number of columns in a chunk to the maximum allowed by the CPU memory. A GPU kernel computed the absorptivities for all columns in a chunk. Since the code is written in Fortran, a Fortran to CUDA compiler (F2CACC) was used to aid in the conversion of the code into CUDA. The OpenACC compiler provided by PGI which automatically converts Fortran or C code to CUDA with the help of compiler directives was also used. The speed-up obtained by this strategy is very negligible as the number of columns per chunk is still low, and the CPU remains idle while the GPU performs computations.

B. Asynchronous Execution of Radiation Computations

The time spent in GPUs can be effectively hidden by asynchronous execution. The technique can be summarized as

- Transfer relevant data to GPU
- Begin computation on GPU
- CPU continues its execution of other parts of the code
- Data is transferred to CPU during the next call to radiation

Since the computations of the columns are independent, for the *radabs* routine the collection of columns into chunks is not necessary. All the columns of all chunks can be combined together and executed at once in one kernel invocation. We also need to ensure that the CPU does not idle while waiting for the GPU to finish its computations. Hence a novel strategy was proposed to achieve both these objectives. Since the *radabs* computations are not performed every time step and the numerical methods involved in the computations are stable, we can use the absorptivities computed in the previous *radabs* time step for the current *radabs* time step. When *radabs* is called for each chunk, the input arguments for each invocation are marshalled into a buffer. After *radabs* is called for the last chunk, the buffer is transferred to the GPU, and the GPU kernel is executed asynchronously while the CPU proceeds with other calculations. During the next invocation of radabs in the next time step, the results of the previous *radabs* computations on GPU are brought back to the CPU and utilized. The psuedo code for asynchronous executions is as follows:

Algorithm 3 Strategy for Asynchronous Execution: *Radabs*

```
for every time step do
    for each chunk do
        if do absorp/emiss calc this time step then
            collect the data necessary for computation.
            receive the computed values of previous
            radabs time step
            if last chunk then
                copy the necessary data of all columns
                launch the kernel asynchronously
                continue other computations.
            end if
        end if
    end for
end for
```

Now since the *radabs* calculations are done asynchronously the time spent in the *radabs* routine is only to collect the relevant data for the next *radabs* computation and copy back the results of the previous *radabs* computation. This greatly reduces the time spent in *radabs* and thus enables a more frequent *radabs* computation. Hence the use of asynchronous executions enables us to make computations, that were done infrequently due to their expensive nature, more frequent.

C. Acceleration of Short Wave Radiations

The *radcswmx* routine calculates the heating due to solar radiation. The solar spectrum is divided into 19 intervals. The frequency of these computations is one hour of simulation time. The chunks are load balanced so that half the columns in the chunk are day columns and the rest are the night columns. The *radcswmx* computations are done only on the columns that are in daylight. The columns are rearranged so that all the day columns are together and all the night columns are together. This makes it easier to compute only the day columns. The *radcswmx* routine can be divided into three stages. The first stage computes the layer reflectivities and transmissivities for each layer using the Delta-Eddington solutions [3]. In the second stage the columns are divided into sets of adjacent layers called regions in which clouds are maximally overlapped. The clouds are randomly overlapped between different regions. This stage determines the number of unique cloud configurations, called streams, within a region.

The first and the second stages are independent of each other. The third stage computes the up and down fluxes and the solar heating rate. This stage is dependent upon the first two stages. The strategies for parallelization of the first and the third routines are similar. A single thread is associated with a single column. The second stage which involves some random calculations is done on the CPU.

The first and the third stage are converted into CUDA kernels in which a single column is computed by a single thread. The strategy for asynchronous executions is similar to that of *radabs*. All the chunks compute the second stage and store the data required for the first and second stages. The last chunk calls the two kernels asynchronously. This method is justified as the short-wave computations are performed every hour and the solar radiation from the previous time step may not vary much from the current time step.

Algorithm 4 Strategy for Asynchronous Execution: *radcswmx*

```
for every time step do
    for each chunk do
        if do short-wave heating then
            receive the previous heating rates
            store the data required for stage I
            perform stage II
            store the data required for stage III
            if last chunk then
                copy the necessary data of all columns
                launch the kernels asynchronously
                continue other computations.
            end if
        end if
    end for
end for
```

For a multi-threaded program, the chunks are divided among the CPU threads. In this case, if we are using a single GPU, one of the CPU threads is used to launch the kernel. The other threads marshal their data into a buffer and a simple synchronization primitive is used to ensure that the data from all the threads has been collected. Similarly for a multi-threaded program with multiple GPUs, the CPU threads are divided into as many groups as there are GPUs available. The threads within a group synchronize among themselves and one thread in the group is used to launch the GPU kernel for a GPU. In the case of multiple nodes, each node is assigned a single MPI process and the OpenMP threads of each process access the GPUs in the same way as a single-node multi-threaded program. The number of columns per chunk was configured to divide the chunks equally among the threads.

All the CUDA kernels used a large amount of registers, so the shared memory was used in such a way that it did not limit the occupancy that could be achieved by the register usage. Most of the arrays had the column index as the fastest varying dimension leading to coalesced accesses to memory. The thread divergence in the kernels is very minimal.

SNo	Resolution	lat $\times$ lon (grid spacing in degrees)	columns
R1	$f45\_g37$	4 x 5	3312
R2	$f19\_g16$	1.9 x 2.5	13824
R3	$f09\_g16$	0.9 x 1.25	55296
R4	$f05\_g16$	0.47 x 0.63	221184

TABLE I
TABLE OF DIFFERENT RESOLUTIONS OF CESM USED IN OUR EXPERIMENTS

IV. Experiments and Results

A. Experiment Setup

The CESM model was run for various resolutions and various hardware configurations and the results are shown comparing the existing model runs on CPU with our asynchronous execution model. The various resolutions chosen are shown in Table I. All these resolutions use the finite volume grid for atmosphere and displaced pole grid for ocean. The lat $\times$ lon represent the approximate spacing between grid points for the finite volume grid in degrees. For the component sets in CESM [18], [19], we used stand-alone CAM (Community Atmosphere model) containing CAM4 physics with finite volume semi-Lagrangian dynamical core. In these component sets, only atmosphere and land models are active while sea ice and ocean are prescribed from data. The CAM4 physics uses the *radabs* and *radcswmx* routines. Each resolution has a different number of vertical columns. The number of columns is equal to the product of the number of latitude and longitude grid points.

Our experiments were run on two platforms. The first is the Fermi cluster located in our department. The cluster has five nodes connected by Gigabit Ethernet. Each of the first four nodes (Fermi1 - Fermi4) contain four Intel(R) Xeon(R) CPU W3550 processors operating at 3.07 Ghz clock speed and one NVIDIA C2070 card. The fifth node (Fermi5) has 16 Intel(R) Xeon(R) CPU ES-2660 processors operating at 2.20 Ghz clock speed and three NVIDIA M2090 GPU cards. The first four nodes have 16 GB main memory each and Fermi5 has 64 GB main memory with Intel 12.0.4 compiler. The second platform, *Pune cluster*, is located in the NVIDIA development center in Pune, India. This has 4 nodes connected by IB 40Gb/s. Each node has 8 Core Intel i7 operating at 3.07 GHZ clock speed, and 2 GPUs which are a mixture of S2050, C2070 and K20 GPU cards. Each node has 24GB RAM with the PGI 13.4-0 and OpenACC accelerator compilers.

The experiments were performed with four execution configurations. These are:

- Single core and Single GPU
- Multi-core OpenMP with Single GPU
- Multi-core OpenMP with Multiple GPUs
- Multi-node MPI with OpenMP and Multiple GPUs

B. *Results*

C. *Verification*

In our method of asynchronous calculations, the results of radiation calculations at a particular time step are used only in the next time step. The resulting simulation output changes but since the absorptivities and emissivities are slowly varying functions of time (hence they were calculated once every 12 hours and assumed to be constant for the next twelve hours) using the next time step does not lead to significant degradation in the quality of simulations (since the frequency of these qualifications can now be increased to 1 hour, we believe the simulations will be superior vis-a-vis the default configuration). We verified the accuracy of the results by finding the error growth and root mean square (RMS) difference of the temperature values produced in the original and the modified code after one day of simulations. We found that the RMS difference was very small and of the order of 10^{-2} to 10^{-3}, thus demonstrating that the accuracy of the modeling is not compromised due to our asynchronous executions.

Single core and Single GPU: For both the *radabs* and *radcswmx*, experiments were performed using a single CPU core and a single GPU. Those resolutions for which the data, required for the kernel computations, could fit on a single GPU were run. For the *radabs* kernel, the experiments were run with the R1, R2 and R3 resolutions and for the *radcswmx* kernel, the experiments were run with the R1 resolution. For the R1 and R2 resolutions, every experiment was run for five model days and for the R3 and R4 resolutions,the experiments were run for a single model day. Figure 3 shows the speed up values achieved for *radabs* for different radiation frequencies and resolutions. The speed ups varied from 3.4 to 26.4 ×. Figure 4 shows the speed ups achieved for the entire model run, the atmosphere run and physics run for the three resolutions by the acceleration of the *radabs* routine. For each of these routines, the speed up achieved is limited by the percentage of time occupied by radabs in them. From these two figures, we can observe that the speed-up obtained for the resolution R2 is maximum. This is because the number of columns in resolution R1 are too less to exploit the SIMT parallelism in the GPU effectively. The speed up for resolution R3 is not higher than for resolution R2 because for the first and the last time step the *radabs* calculations are done on the CPU. For a higher resolution, *radabs* takes a longer time to compute on the CPU. In Figure 3 and Figure 4, we observe that the highest speed up is achieved for the 1 hour frequency as there are more number of *radabs* invocations at this frequency. The speed up achieved reduces as we decrease the *radabs* frequency of computation.

On the *Pune cluster* we compared the performance of OpenACC and our asynchronous code using a single CPU and a single GPU for the *radabs* routine. The model was run for five simulation days with R1 and one simulation day with R2 resolutions. The *radabs* calculations were performed at a 1 hour frequency. Figure 5 shows the execution times of the

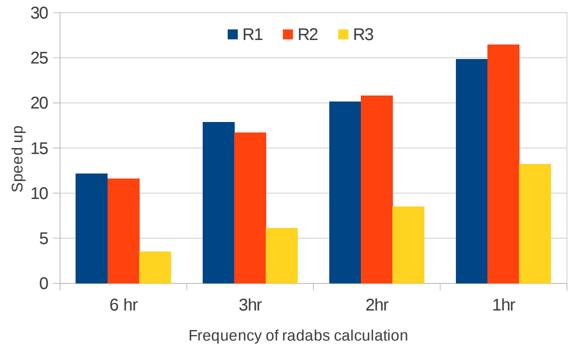

Fig. 3. Speed-up of *radabs* in the Single CPU Single GPU Case on Fermi Cluster

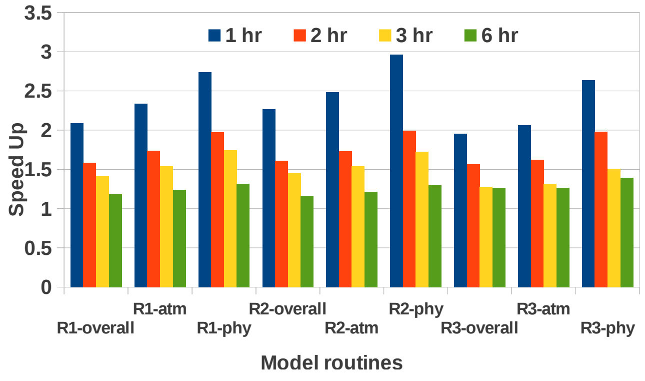

Fig. 4. Speed-up for Different Model Routines in the Single CPU Single GPU case on Fermi Cluster

entire model for the CPU based code, the OpenACC accelerated code and our asynchronous algorithm. The speed up obtained by using the OpenACC directives is 2.07× whereas for asynchronous execution the speed up obtained is 2.68× for resolution R1. The reasons for the better performance of our strategy are asynchronism and more data parallelism. The OpenACC implementation calls a GPU kernel for each chunk of columns resulting in increased data latency, whereas our implementation invokes a GPU kernel for all the chunks simultaneously.

For the *radcswmx* routine, we ran the model on the Fermi cluster for the resolution R1. Since the frequency of *radcswmx* calculations is 1 simulation hour, we did not attempt to increase the frequency of calculation. A five day model run was conducted and the speed up obtained for the *radcswmx* by our strategy was 5.66× and the overall model speed up was about 1.15×. This speed up is not as much as the speed up obtained for the *radabs* routine for the same resolution since *radcswmx* occupies 16% of the total execution time while *radabs* occupies 55%.

A run was performed in which both the *radabs* and *radcswmx* routines were accelerated using a single CPU core and two GPUs, one for each of the routines, on the Fermi cluster for the resolution R1. The *radabs* frequency was set to

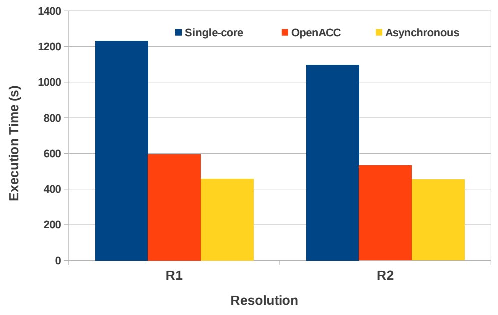

Fig. 5. Execution Times of the Entire Model in Pune cluster for the Single CPU and Single GPU

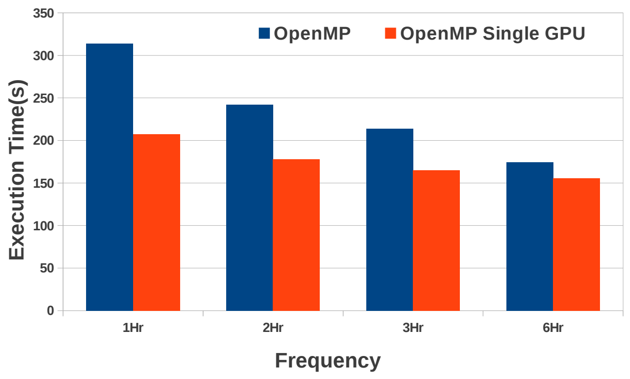

Fig. 6. Execution Times for the entire model with OpenMP executions with and without GPU computations for *radabs* with resolution R1 on Fermi cluster

1 hour and the model was run for five simulation days. The speed up achieved for the overall model using this approach is 2.38× when compared to 2.09× obtained with only the *radabs* acceleration. The performance improvement obtained by accelerating both the routines over the acceleration of only the *radabs* is 14%.

Multi-core OpenMP with Single GPU: In this configuration we used OpenMP with four CPU cores and a single GPU. These runs were performed on the Fermi cluster. For *radabs*, the experiments were performed with R1, R2 and R3 resolutions for which a single GPU is sufficient to hold the buffers for all the columns. The speed up values were obtained by comparing with the OpenMP executions on 4 cores. About 50-80% reduction in the execution time for the entire model and 2.3×-3.4× speedup for the *radabs* routine were obtained with a 1 hour frequency. Figure 6 shows the execution times of the model for the resolution R1. As expected the frequency of 1 hour gives the best performance.

Table II shows the number of computation days saved for the different resolutions by acceleration of the *radabs* routine in this configuration for a 100 year simulation. This is the number of computation days that would have been saved if the model had been run with our accelerated code when compared

Frequency	R1	R2	R3
1 hour	9	51	169
2 hour	5	21	77
3 hour	4	14	41
6 hour	1	8	29

TABLE II
ESTIMATED NUMBER OF COMPUTATION DAYS SAVED FOR A 100 YEAR SIMULATION FOR THE OPENMP AND SINGLE GPU CASE ON THE FERMI CLUSTER

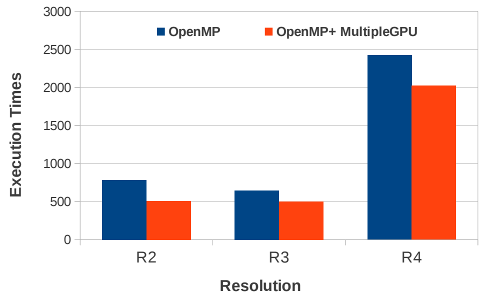

Fig. 7. Execution Times for the Entire Model with 1 hour Frequency for OpenMP and Multiple GPU case on Fermi cluster

to the OpenMP CPU version. The number of days saved in general increases with the increasing resolutions due to the larger amount of computations.

With the acceleration of only the *radcswmx* routine, performance improvement of 7% was obtained over the OpenMP CPU version for the entire model with the R1 resolution. For the same resolution, both the routines *radcswmx* and *radabs* were accelerated together using 4 CPU cores and 2 GPUs, one for each of the routines. The frequency of *radabs* calculations was set to 1 simulation hour. The speed up achieved was 1.95× over the OpenMP CPU version when compared to the speed up of 1.51× obtained with accelerating only the *radabs* routine.

Multicore OpenMP with Multiple GPUs: For these runs, we used 12 CPU cores for OpenMP and 3 GPUs on the Fermi 5 cluster. With more number of GPUs it is possible to run higher resolutions and the use of 12 CPU cores also enables the lower resolution models to run faster. We performed the experiments for R2, R3 and R4 resolutions for the *radabs* routine for the four different frequencies. The speed-up obtained in this configuration is compared to a 12 core OpenMP run. The speed up achieved is 2.2-3.6 × for the *radabs* routine and 1.53-1.19 × for the entire model. Figure 7 shows the executions times of the entire model for a 1 hour frequency for different resolutions.

Since we use asynchronous memory transfers to the GPU, pinned memory is allocated on the host node. The amount of pinned memory for the R4 resolution is 14GB. Since each of the 12 OpenMP threads has its own memory stack, the R4

Experimental Configuration	Without GPUs	With GPU
Single CPU and Single GPU	3807.81	1679.04
4-OpenMP and Single GPU	1317.67	705.37
2-MPI and 4-OpenMP and 2 GPUs	960.4034	714.7923
12-OpenMP and 3GPUs	784.26	510.21

TABLE III
EXECUTION TIMES OF THE MODEL FOR A 1 HOUR FREQUENCY FOR DIFFERENT CONFIGURATIONS ON FERMI CLUSTER FOR *radabs* USING RESOLUTION R3

resolution results in excessive pinned memory. This causes the other routines in the model to run slower than the OpenMP CPU version leading to a degradation in the overall model performance. So, for this resolution, the memory transfers to the GPU were made synchronous without the use of pinned memory. Hence the speed up for the resolution R4 is 19% compared to the 53% speed up seen in resolution R2.

For *radcswmx* the resolution R2 was run for this configuration on the Fermi 5 node. The speed up achieved for this resolution was 12.5% for the entire model and 31.5% for the *radcswmx* routine alone.

Multi-node MPI with OpenMP and GPUs: In this configuration, we used multiple nodes of Fermi cluster with one MPI process per node and OpenMP for the CPU cores within each node and one GPU per node. The speed-up comparison is against a run that does not use GPUs. Our experiments were performed with two nodes of the Fermi cluster for the *radabs* routine for the four different frequencies, with 2 MPI processes and 4 OpenMP threads per process. We ran the experiments with R2 and R3 resolutions. The speed up, achieved for the 1 hour frequency of *radabs* calculation for the entire model is 34% for the R2 resolution and 27.9% for the R3 resolution. Similar to the earlier configuration, pinned memory was not used for the R3 resolution since it degraded the performance. This was not observed when using OpenMP with single GPU (Table II) as those simulations were performed on Fermi 5 which has 64GB main memory whereas the multi-node runs were performed on Fermi 1 and Fermi 2 which have only 16 GB main memory.

Table III gives the execution times of the model after accelerating the *radabs* routine for a 1 hour computation frequency for the different experiment configurations. The execution time for the GPU enabled execution for Multi-node MPI with OpenMP and 2 GPUs is more than the time taken for OpenMP with Single GPU. This is also explained due to the use of pinned memory.

V. CONCLUSIONS AND FUTURE WORK

The use of GPUs to accelerate applications that are embarrassingly parallel leads to significant performance improvements. In this paper, we have described a novel way to perform hybrid computing for physics routines of the climate models. The most time consuming routines *radabs* and *radcswmx* were ported from Fortran code to CUDA C. We have performed experiments with different resolutions and different hardware configurations. The use of an auto parallelizer tool OpenACC was also explored. The results show a $26\times$ performance improvement over a single CPU core. Our asynchronous executions also performed better than the OpenACC parallelization. At all resolutions the gains are significant and allows for frequent computations of absorptivities and emissivities, hitherto done at a very coarse temporal frequency of 12 hours. With asynchronous executions on GPU the rate of computation does not suffer and the simulations also gain from an atmospheric science perspective due to better resolution of diurnal cycle of the radiational parameters which could be significantly affected by high frequency fluctuations of water vapour. Due to pinning memory constraints, some of the combinations (such as Multi-node-Multiple GPU with OpenMP vis-a-vis Single GPU and OpenMP) showed lowered efficiencies. Most recent studies using GPUs for climate modeling have been on improving throughput through acceleration of an atmospheric model's dynamical core. Our study complements these and clearly shows that computational accelerators can be very effective in improving model throughput through large improvements in model physics. We further propose to study the acceleration of cloud-convection and stratiform routines. The different dynamical cores such as the finite volume and the spectral element can also be considered for acceleration.

ACKNOWLEDGEMENTS

This research is partly supported by Centre for Development of Advanced Computing, India, project ref. no. CDAC/ESE/STV/0016 and by CUDA Research Centre award from NVIDIA to Supercomputer Education and Research Centre, Indian Institute of Science.

REFERENCES

[1] http://www.cesm.ucar.edu

[2] Conley, Andrew J., et al. "Description of the NCAR Community Atmosphere Model (CAM 5.0)." (2012).

[3] Briegleb, Bruce P. "Delta-Eddington approximation for solar radiation in the NCAR Community Climate Model." Journal of Geophysical Research: Atmospheres (19842012) 97.D7 (1992): 7603-7612.

[4] Kolb, Craig, and Matt Pharr. "Options Pricing on the GPU." GPU Gems 2 (2005): 719-731.

[5] Genovese, Luigi, et al. "Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures." The Journal of chemical physics 131 (2009): 034103.

[6] Smelyanskiy, Mikhail, et al. "Mapping high-fidelity volume rendering for medical imaging to cpu, gpu and many-core architectures." Visualization and Computer Graphics, IEEE Transactions on 15.6 (2009): 1563-1570.

[7] Stone, John E, et.al. "GPU-accelerated computation and interactive display of molecular orbitals. " In Wen-mei Hwu, editor, GPU Computing Gems, chapter 1, pp. 5-18. Morgan Kaufmann Publishers, 2011.

[8] Cohen, J., and M. Jeroen Molemaker. "A fast double precision CFD code using CUDA." Parallel Computational Fluid Dynamics: Recent Advances and Future Directions (2009): 414-429.

[9] Michalakes, John, and Manish Vachharajani. "GPU acceleration of numerical weather prediction." Parallel Processing Letters 18.04 (2008): 531-548.

[10] Govett, M., Jacques Middlecoff, and Tom Henderson. "Running the NIM next-generation weather model on GPUs." Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on. IEEE, 2010.

[11] Carpenter, Ilene, et al. "Progress towards accelerating HOMME on hybrid multi-core systems." In International Journal of High Performance Computing Applications (2012).

[12] Shimokawabe, Takashi, et al. “An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code.” High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for. IEEE, 2010.

[13] Oliver Fuhrer, et al. “GPU Consideration for Next Generation Weather (and Climate) Simulations” ETH.

[14] Adhianto, Laksono, et al. “HPCToolkit: Tools for performance analysis of optimized parallel programs.” Concurrency and Computation: Practice and Experience 22.6 (2010): 685-701.

[15] Jetley,P., et. al. “Scaling Hierarchical N-body Simulations on GPU Clusters.” In Proceedings of ACM/IEEE Supercomputing Conference (SC), pages 1–11, 2010.

[16] Song,F., et. al. ”Enabling and Scaling Matrix Computations on Heterogeneous Multi-core and Multi-GPU Systems.” In the Proceedings of IEEE/ACM Supercomputing Conference (SC), pages 365–376, 2012.

[17] Raghavan H. and S Vadhiyar. “Efficient Asynchronous Executions of AMR Computations and Visualization on a GPU System. In the Journal of Parallel and Distributed Computing (JPDC), Vol 73/6, pp 866-875, 2013.

[18] http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc_1_0_4/a3170.html

[19] http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc_1_0_4/x42.html #ccsm_components

[20] Sundari,S.M et. al. “Dynamic Component Extension: A Strategy for Performance Improvement in Multi-Component Applications.” In the International Journal of High Performance Computing Applications (IJHPCA), Vol. 23/1, pp 84-98, 2009.