J. Parallel Distrib. Comput. 102 (2017) 199–212

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Asynchronous and synchronous models of executions on Intel® Xeon Phi™ coprocessor systems for high performance of long wave radiation calculations in atmosphere models

Amlesh Kashyap [a], Sathish S. Vadhiyar [a,*], Ravi S. Nanjundiah [b], P.N. Vinayachandran [b]

[a] Department of Computational and Data Sciences, Indian Institute of Science, Bangalore-560012, India
[b] Centre for Atmospheric and Oceanic Sciences, Indian Institute of Science, Bangalore-560012, India

HIGHLIGHTS

- Asynchronous and synchronous executions models for radiations on Intel Xeon Phi.
- Techniques including loop rearrangement and early placement, low-cost substitutions.
- Performance improvement of up to 45% due to our asynchronous model.
- Wall-clock time savings of up to 2.25 years for multi-century climate simulations.
- Performance improvement of up to 70% due to our synchronous model.

ARTICLE INFO

Article history:
Received 1 September 2016
Received in revised form
1 November 2016
Accepted 15 December 2016
Available online 3 January 2017

Keywords:
Intel Xeon Phi co-processors
Offloading
CAM
Long-wave radiations

ABSTRACT

Long Wave Radiation Calculations are one of the most time-consuming calculations in atmosphere modeling. In this work, we explore two models for executions of these calculations on Intel® Xeon Phi™ Coprocessor Systems. In the *asynchronous* model, we offload the radiation calculations to the coprocessors and simultaneously execute calculations on the coprocessors along with the other atmosphere model calculations in the CPU cores. In the *synchronous model*, the CPU cores after offloading, wait for the results, and use the results in the same time step. We developed various techniques to complete these synchronous executions in minimal time, including loop rearrangement and low-cost interpolations. Using our experiments on an Intel Xeon Phi cluster, we show that our asynchronous execution model results in savings of many months in wall-clock execution time for multi-century climate simulations. Our synchronous execution model results in performance improvements of up to 70% in long-wave radiation calculations.

1. Introduction

Climate change has become a prominent field of study in the present time due to its immense impact on all walks of life as diverse as energy usage, health, agriculture, etc. Modeling the climate facilitates the prediction of temperature changes, winds, radiation, relative humidity and other factors. These are non-linear models containing computationally intensive routines that require a lot of computing power [7]. Climate models are mathematical representations of the climate based on the physical, chemical and biological principles. The complexity of the equations, which include the equations of conservation of mass, momentum, energy and species, necessitates the use of numerical methods in solving them. Climate models simulate the interaction of the various components of the climate such as atmosphere, land, ocean and ice.

One such climate model is the Community Earth System Model [4], developed and maintained by the National Center for Atmospheric Research (NCAR). CESM consists of several component models, e.g. physical climate, chemistry, land ice, whole atmosphere, etc., that can be coupled in different configurations. In all cases, geophysical fluxes across the components are exchanged via a central coupler module. A large number of simulations with CESM have been conducted, some of which are available for community analysis [4].

* Corresponding author.
E-mail addresses: vss@cds.iisc.ac.in (S.S. Vadhiyar), ravi@caos.iisc.ernet.in (R.S. Nanjundiah), vinay@caos.iisc.ernet.in (P.N. Vinayachandran).

http://dx.doi.org/10.1016/j.jpdc.2016.12.018
0743-7315/

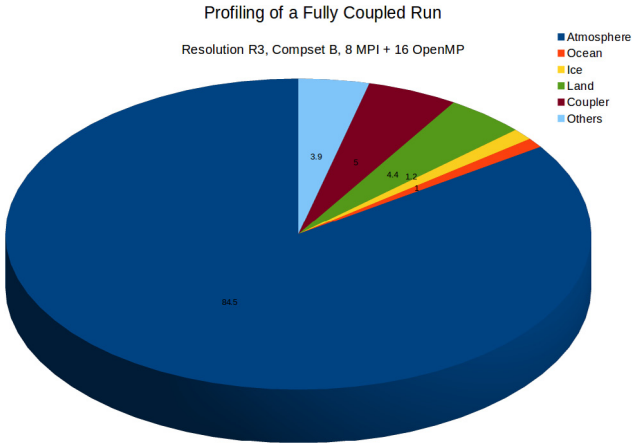


Fig. 1. Execution profile of CESM.

The atmosphere is the most time-consuming model in CESM, as shown in Fig. 1. We obtained such execution profiles using Intel VTune Amplifier and HPCToolkit profiler [11]. The execution profile is obtained by running CESM v 1.2.2 with $f02\_g16$, the highest resolution[1] and B compset (fully-coupled run) on eight-node 128-core Intel Xeon processors with 8 MPI processes and 16 OpenMP threads per process for a total of 128 threads. The model used for atmosphere is the Community Atmosphere Model (CAM4) [3,5]. The computational phases in CAM are dynamics and physics. The evolutionary equations for the flow of atmosphere are advanced by the dynamical core (henceforth dynamics) and the subgrid phenomena including clouds, long and short wave radiations and others are approximated through physics. Finite Volume was adopted as the default core for the physics [5] which uses a longitude $\times$ latitude $\times$ vertical level computational grid over the sphere. The physics in CAM is based upon vertical columns whose computations are independent from each other. In a typical parallel implementation of physics, columns are assigned to MPI processes and within a process, OpenMP threads are used to compute the columns assigned to that process.

Most of the work to accelerate climate science routines have concentrated on the dynamics. In this work, we chose to accelerate the physics routines as they were observed to consume significant times. In our experiments with different resolutions, we found that the physics routines that compute the long and short wave radiations consume about 15%–35% of the total time spent in the atmosphere routines. In fact, the time taken by a full radiation time step is about 3$\times$ the time for the regular time step. The routine *radabs*, corresponding to long-wave radiations, takes the largest percentage of the overall run time (14%–21% for high resolutions and 50% for the lowest resolution) when compared to any other single routine. The cost of such long-wave computations is significant for other climate modeling applications and on other architectures as well. *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 vapor 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.

Accelerators and co-processors are widely prevalent and have been used to provide high performance for many scientific applications. Intel® Xeon Phi™ coprocessors have been gaining ground to provide speedups for advanced scientific applications [9,12,13]. However, the use and demonstration of these coprocessors for climate modeling are limited. In this work, we explore two models for executions of these calculations on Intel® Xeon Phi™ coprocessor systems. In the *asynchronous* model, we offload the radiation calculations to the coprocessors and simultaneously execute calculations on the coprocessors along with the other atmosphere model calculations in the CPU cores. While this model completely masks these calculations in the coprocessors thereby providing significant speedups, it could appear that since the data is used in the subsequent time step, the simulation results could be less accurate. However, due to complete masking and increased performance, the frequency of computing these radiation related calculations can be significantly increased and the diurnal cycle computed in an improved fashion which would lead to improved climate simulations. We also explore a *synchronous model*, in which the CPU cores after offloading, wait for the results, and use the results in the same time step. We developed a suite of techniques to complete these synchronous executions in minimal time. These techniques include loop rearrangement and fission for vectorization, and substituting the costly math functions with low-cost interpolation functions by employing efficient look-up table data structures and vectorization-promoting search techniques.

Using our experiments on an Intel Xeon Phi cluster, we show that our asynchronous execution model results in performance improvement of up to 89% in long-wave radiation calculations and 45% in the atmosphere model for the lowest resolution (13% for high resolutions), and savings of many months in wall-clock execution time for multi-century climate simulations. Our synchronous execution model results in performance improvements of up to 70% in long-wave radiation calculations and 10% in the atmosphere model. We also show that individually our vectorization strategies, low-cost approximations of transcendental functions using interpolation functions and lookup table methods, and use of advanced compiler flags provided significant improvements in performance.

Following are the primary contributions and novelty of our work:

1. This is the first work, to our knowledge, that uses domain knowledge to promote asynchronous hybrid CPU-Intel Xeon Phi executions, in which a slow-varying function is offloaded to Intel Xeon phi in a time step and its results are used in the subsequent time step.
2. The performance-accuracy tradeoffs of the approach of asynchronous computations of slow-varying functions are comprehensively evaluated by comparing with a synchronous execution model in which the results of the function are used in the same time step. This is the first kind of such tradeoff analysis conducted in the context of coupled climate models.
3. This is also the first work, to our knowledge, that uses dependency analysis and code reorganization for efficient hybrid executions in a large-scale legacy climate modeling application, namely CESM. Such reorganizations lead to efficient overlapping of CPU and Xeon Phi executions resulting in high performance for the synchronous execution model.

In Section 2, we give the overall structure of the radiation calculations. Section 3 covers related work in the area of high performance of climate and weather models on accelerators and coprocessors. In Section 5, we describe our asynchronous model of execution of radiation calculations on Intel Xeon Phi core simultaneously with executions of other computations on the CPU cores. Section 6 describes our synchronous execution model, in which we speedup the radiation calculations on the Xeon Phis. This section describes various optimizations to achieve speedup including intelligent placement of the radiation calculations in the overall CAM execution, use of low-cost interpolations, and

[1] See Table 3 for resolutions.

vectorization strategies including loop fission. Section 7 presents experiments and results. In Section 8, we derive general principles from our optimizations on the climate modeling application. Section 9 gives conclusions and presents scope for future work.

2. Background

2.1. Radiation calculations

The computational grid for the atmosphere is divided into latitudes, longitudes in the east–west (x) and north–south (y) directions 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 every vertical column can be computed independent of the other columns giving rise to data parallelism that can be exploited on multi and many core architectures. 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 contiguous set of vertical columns i.e. columns from neighboring grid points may not be in a single chunk. These chunks are distributed among the MPI processes and the OpenMP threads. Every physics routine is called for each chunk. The pseudo-code for physics calculations is shown in Fig. 2.

One of the most important components of the radiation routine is the *radclwmx* subroutine that computes long wave heating rates. 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. As shown in Fig. 2, all radiation calculations including *radabs* are invoked for each chunk. In this work, we focus on providing high performance for the *radabs* routine. *radabs* consumes at least 14% of the total time spent in the atmosphere run, and the full radiation time steps that include the *radabs* computations takes about 3× more time than the regular time steps. In the default execution mode it is run at a 12 h frequency. It would however be preferable to conduct these executions more frequently for better representation of fluctuations of emissivities and absorptivities due to diurnal fluctuation of water vapor.

2.2. Intel Xeon Phi architecture

Intel's first generation Many Integrated Core (MIC) architecture [10,19,20] codenamed Knights Corner, is an x86 based system that contains up to 61 in-order processing cores, and offers peak single precision performance of nearly 2.1 TFLOPS. Each of these cores supports up to 4-way hardware multithreading and features a vector unit which uses wide 512 bit registers. Each core features fully coherent L1 and L2 caches, with a bidirectional ring that provides fast access to L2 caches of other cores.

The Xeon Phi also offers high memory bandwidth of nearly 160 GBPS, but the L1 and L2 caches offer much higher memory bandwidth than the memory, and using them effectively is the key to approaching peak performance. Apart from thread parallelism, it is crucial to properly utilize the wide vector unit; more so than on the Xeon which has only 256 bit wide registers. Although the Xeon Phi offers gather and scatter operations for vectors, vectorization is significantly more effective if data is contiguous in memory with unit stride accesses. Data alignment will provide even better vector performance. The Xeon Phi also features low precision hardware support for certain math functions like power, logarithm, etc. apart from FMA (Fused Multiply Add) that provide additional speedup for workloads that have these computations.

```
1 foreach time step do
2     foreach chunk do
3         stratiform () ;
4         convection () ;
5         ...
6         radiation () ;
7         ...
8     end
9 end
```

Fig. 2. Pseudocode for physics calculations in CAM.

Programming the Xeon Phi is similar to programming any other x86 machine—the same programming model is applicable with a host of popular libraries like MPI and OpenMP that are supported. The same optimizations that apply on the Xeon also apply without change on the Xeon Phi. In general, development time of a parallel application on the Xeon Phi is short. The Xeon Phi offers three modes of operation—native, offload and symmetric. We use the offload model of execution in our study, where the host offloads a portion of computation to the Xeon Phi either synchronously or asynchronously with simultaneous CPU executions.

3. Related work

In this section, we present previous efforts that explored the use of accelerators and co-processors for climate and weather models. There have been a number of efforts in using GPUs for climate and weather models. Michalakes and Vachharajani [15] used GPUs to improve the performance of the Weather Research and Forecast (WRF) model. Their work resulted in 5–20× speed-up for the computationally intensive routine WSM5. In the work by Govett et al. [8], the non-hydrostatic icosahedral (NIM) model was ported to the GPU. 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. The Oak Ridge National Laboratory (ORNL) ported the spectral element dynamical core of CESM, HOMME, to the GPU [2]. 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 climate model ASUCA [23] is a production weather code developed by the Japan Meteorological Agency. 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. 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.

In the work by Schalkwijk et al. [22], the authors have utilized the GPUs for Large Eddy Simulation (LES) models, which has allowed them to provide turbulence-resolving numerical weather forecasts over a region the size of the Netherlands, at 100 m resolution. Garcia et al. [25] accelerate a Cloud Resolving Model (CRM) by implementing the MPDATA algorithm on GPU using CUDA. They perform optimizations like data reuse on GPU for

saving transfer time, coalesced memory accesses on GPUs, and utilizing the texture memory and shared memory on the GPU. Fuhrer et al. [6] optimize the atmospheric model COSMO by rewriting the dynamical core using STELLA DSEL and porting the remaining parts of the Fortran code to the GPUs using OpenACC compiler directives.

Intel Xeon Phi processors have been used to provide high performance for different scientific domains [9,12,13]. There have been recent efforts in porting weather and climate models on Intel Xeon Phi accelerators. Mielikainen et al. have a number of efforts on optimizing Weather Research Forecast Model (WRF) on Intel Xeon Phi architecture [16–18]. In [18], the authors have optimized the Thomspson cloud microphysics scheme, a sophisticated cloud microphysics scheme. They have used optimization techniques such as modifying the tile size processed by each core, using SIMD, data alignment, memory footprint reduction, etc. to achieve a speedup of 1.8× over the original code on Intel Xeon Phi 7120P and 1.8× over the original code on dual socket configuration of eight core Intel Xeon E5-2670. In another work [17], the authors optimize the longwave radiative transfer scheme of the Goddard microphysics scheme of the WRF model, for Intel MIC architecture. Their optimization yields a speedup of 2.2x over the original code on Xeon Phi 7120P. They also optimize the updated Goddard shortwave radiation of the WRF model for Intel Xeon Phi [16]. They observe a speedup of 1.3× over the original code on Xeon Phi 7120P.

Betro et al. [1] highlight experiences and knowledge gained from porting such codes as ENZO, H3D, GYRO, a BGK Boltzmann solver, HOMME-CAM, PSC, AWP-ODC, TRANSIMS, and ASCAPE to the Intel Xeon Phi architecture running on a Cray CS300-AC Cluster Supercomputer named Beacon. Most of these were ported by compiling with the flag -mmic. They conclude that accelerator based systems are the wave of the future based both on their power consumption and variety of programming paradigms to fit the needs of all application developers.

Michalakes et al. [14] optimize a standalone kernel implementation of Rapid Radiative Transfer Model of the NOAA Nonhydrostatic Multiscale Model (NMM-B). They apply methods such as dynamic load balancing, lowering inner loops, avoiding vector remainders, trading computation for data movement, prefetching, etc. and obtain a speedup of 1.3× over the original code on Xeon Sandy Bridge and 3× over the original code on Intel Xeon Phi.

While existing efforts on accelerators and co-processors focused on data management and vectorization, in addition to these optimizations, our work proposes novel asynchronous execution model for simultaneous executions on both the CPU and coprocessor cores.

4. Aggregation of *radabs* for offloading to Intel Xeon Phis

As shown in Section 2, the *radabs* routine is invoked for each chunk containing a set of columns. A simple strategy to provide high performance for the *radabs* routine on Intel Xeon Phi systems is to offload the calculations for each chunk to Intel Xeon Phi and let each column of a chunk to be computed by a single Intel Xeon Phi thread. However, by default, the number of columns in each chunk is limited to load balance for the other physics computations. Hence this strategy results in a large number of offloads, incurring high offload overheads, and limited parallelism for each offload. In our experiments, we found 10× degradation in performance of *radabs* with this strategy. Another option is to configure CESM to divide the domain into a small number of chunks, each containing a large number of columns. However, as mentioned above, this results in load imbalance in the other physics routines and also affects the OpenMP parallelism of the chunk-level loop for the other computations (e.g., convections) on the CPU cores for a MPI process.

We propose two models of offloading: an **asynchronous model** in which the *radabs* computations are offloaded in the current time step and its results are used only in the next step, and a **synchronous model** in which the results are used in the same time step. For both the models, we aggregate all the chunks into contiguous data and offload the *radabs* computations to Intel Xeon Phis for the entire aggregated chunks. This strategy maximizes the parallelism in the Xeon Phi and drastically reduces the number of offloads, and hence the offloading overheads. For aggregation of data of different variables needed for *radabs*, we used the Array of Structures (AoS) since it provides good locality characteristics. In our experiments, we found the aggregation overheads to be negligible.

5. Asynchronous execution model

In this model, the CPU, after offloading *radabs* to Xeon Phi in the current time step, does not wait for the results from Xeon Phi cores to arrive in the same time step, but immediately proceeds to the other computations. Any computations that depend on the offloaded *radabs* use the results corresponding to the previous offloads. The CPU utilizes the Xeon Phi results only when it reaches the dependent computations in the next time step. As shown in our experiments, the Xeon Phi completes the *radabs* calculations before the CPU reaches the dependent computations in the next time step, thereby completely masking these calculations from the CPU executions. 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.

Since only the results of the previous time step are used for the dependent computations in the current time step, such an asynchronous model can be followed if the offloaded function is slow-varying. We harness the domain-specific knowledge related to climate modeling and use the stability of the radiation calculations to follow asynchronous model for the *radabs* calculations. 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 subsequent time step(s). In the default setup, values obtained from *radabs* are used over the next 12 h of simulations. In our asynchronous model, we propose to compute more often. The greatly reduced time in the asynchronous model enables a more frequent *radabs* computation. The frequent execution enabled by the asynchronous model will improve the representation of emissivities and absorptivities and hence improve the quality of climate simulations.

When *radabs* is called for each chunk, the input arguments for each invocation are marshaled into a buffer. After *radabs* is called for the last chunk, the buffer is transferred to the Xeon Phis, and the radiation calculations are executed asynchronously on the Xeon Phis. The pseudo code for the asynchronous executions is shown in Fig. 3.

6. Synchronous execution model

In the synchronous execution model, the CPUs, after offloading the computations to Xeon Phi, wait for the results and use the results in the same time step. For this to be beneficial, the Xeon Phi computations of *radabs* must be faster than the corresponding CPU computations, and the idling time on the CPU due to waiting for the Xeon Phi results should be minimized. We explore various ways to achieve this goal, namely, identify dependencies for *radabs* and perform early starts of the computations that *radabs* depends on, perform good vectorization of the computations on Xeon Phi

```
1  foreach chunk do
2      if do absorp/emiss calc this time step then
3          collect the data necessary for computation ;
4          receive the computed values of previous radabs time step ;
5          if last chunk then
6              copy the necessary data of all columns ;
7              offload to Xeon Phi asynchronously ;
8              continue other computations on the CPU ;
9          end
10     end
11 end
```

Fig. 3. Strategy for asynchronous execution: *Radabs*.

and identify primary bottlenecks resulting in slower computations on Xeon Phi and replace them with low-cost computations. Note that these optimizations are of lesser relevance for asynchronous execution model, since the asynchronous executions even without the optimizations are able to complete by the next time step when the results are needed in the CPU, as shown in our experiments. Once the relevant portions for optimizations were identified, the use of Intel directives to achieve optimized executions on Xeon Phi only needed a maximum of 3 days of work, which conveys the ease of programming when compared to the other programming models.

6.1. Early placement of radabs computations

Fig. 4 shows the original structure of the physics and *radabs* computations. Using data analysis, we found that in each chunk-level iteration, there exist computations on which *radabs* depends on and independent computations that are not related to *radabs*. In the figure, all the functions, namely, *radtpl*, *radoz2*, *trcpth*, *aer_trans_from_od*, and *radems*, perform computations on which *radabs* depends on. Hereon, we refer to these functions as *feeder functions*. The functions, that are marked as *indep_pre_radabs* and *indep_post_radabs* perform computations that are not related to *radabs* computations. The computations which depend on the results of *radabs* are marked as *dep-post-radabs*.

Recalling that all the longwave radiation functions including the feeder functions and *radabs* are independent across various columns and hence across chunks, we can perform all the feeder functions for all the chunks first, making the input data needed for the *radabs* for all the chunks available at the earliest possible instance. This allows an early start of data transfer and offloading of *radabs* to Xeon Phi. By performing *radabs* offloads at the earliest possible instance, the *radabs*-independent computations can be pushed down the code and can be performed on the CPU cores simultaneously with the *radabs* computations on the Xeon Phi cores. This reduces the waiting time incurred by the CPU for the Xeon Phis to complete. We also found that while the shortwave computations are performed before the longwave computations in the original code, not all the computations in the longwave computations depend on the results of the shortwave computations. Only the *dep-post-radabs* in longwave computations depends on the short wave calculations. Hence the shortwave computations can also be pushed down the code after *radabs*, but before *dep-post-radabs*, and can be performed simultaneously on the CPU cores along with the *radabs* computations on the Xeon Phi cores. This reduced the waiting time further. Overall, we found that the code reorganization and early placement of *radabs* offloads resulted in 10% reduction in waiting time by the CPU over offloading *radabs* at its default location in the original code. The modified structure of the physics code is shown in Fig. 5.

6.2. Vectorization of radabs

One of the important techniques to improve performance on Xeon Phi is the vectorization of the loops, where independent iterations are performed in separate vector units on a core. *radabs* contains two large-level loops: the first loop had 300 lines of code and the second loop contained 95 lines. We performed a number of steps for vectorization:

1. Large loop lengths prevent vectorization on the light-weight vector units. Hence, we performed loop fission by dividing the original loop statement into blocks such that minimal or no data dependency exists between the blocks. Loop fission also necessitated the conversion of many scalars in the small loops to equivalent arrays. This helped in vectorization of most of the loops and loop statements.
2. We also extracted some assignments of scalar variables and function calls with these scalar variables as arguments out of the loop and created array variants of these assignments and function calls with implicit indexing. This resulted in the vectorization of these statements.
3. For some of the math functions, we used the corresponding functions in the Intel Math Kernel Library (MKL). This significantly reduced the time taken for these functions and also helped in vectorization.
4. For some statements using math functions, we used the *SIMD* directive that is used to forcefully vectorize a block of statements.

We found that the above vectorization strategies of loop fission, scalar to array conversions, use of Intel MKL and SIMD directives yielded 43%–55% improvement in performance of the individual loops, and 36.1% improvement in performance for the *radabs* computations on Xeon Phi. We also found using our 16-thread experiments on Intel Xeon CPU cores that these vectorization strategies also provide benefits on the CPU cores yielding 17%–35% improvement in performance of the individual loops, and 13% improvement in performance for the *radabs* computations. We find that these vectorization steps yield higher benefits on Xeon Phi due to increased vectorization lengths.

6.3. Use of low-cost interpolations and search methods

We also found that the major bottlenecks in the executions of the *radabs* on Xeon Phi are the many math functions including *log*, *log* and *power* functions in *goffgratch* and *exp*. These were found to consume at least 30% of the overall execution time on Xeon Phi. We explored the use of interpolation functions and lookup tables [24] to replace these costly math functions. Specifically, for each math function, we determined the set of ranges of parameter values passed to the function at different locations in the code using trial runs. We then conducted a series of offline experiments to determine the best method in terms of speed and accuracy for function evaluation in the range. In our offline experiments, we divided each range into fixed size intervals and determined the function values, Y, at discrete input parameter values, X, separated by the fixed size intervals. The array of X and Y formed a lookup table. For the range, we then tried to fit our interpolation functions and chose the function that fits with the least approximation errors. We explored linear, cubic and quartic spline functions for interpolations. Thus, given an x value for a function invoked during

```
1  do i=begchunk, endchunk
2      several physics computations ;
3      shortwaves() ;
       /* longwaves()                                          */
4      radtpl() ;
5      indep_pre_radabs() ;
6      if doabsems then do absorp/emiss calc this time step
7          radoz2() ;
8          trcpth() ;
9          aer_trans_from_od() ;
10         radems() ; /* emissivity                            */
11         radabs() ; /* absorptivity                         */
12     end
13     indep_post_radabs() ;
14     dep-post-radabs() ; /* Post radabs computations        */
15     some updates ;
16 end
```

Fig. 4. Physics and *radabs* computations in CAM.

the *radabs* execution, we can directly invoke the corresponding best interpolation function.

For some ranges, especially for large ranges where interpolation functions gave high approximation errors, we directly search for a particular x value in our lookup table to obtain the corresponding y value. For a range for a function, in which cubic spline interpolation is determined to give the least approximation error, the spline function uses different functions with different coefficients for subranges within the range. Here too, we need to search the lookup table for the subrange in which a given x value falls and use the corresponding coefficients for the spline function. While binary search has smaller time complexity, the loop corresponding to binary search is not vectorizable due to large jumps in accessing the elements of the array that is searched. On the other hand, vectorization of linear search is supported by the Intel Fortran compilers. Thus, the choice of the search method should be dependent on the tradeoffs between benefits due to vectorization and the size of the array.

In cases of small ranges for the cubic spline functions, we use linear search of lookup table, instead of binary search which will be faster if the table size becomes large. We have a separate lookup table based methodology for approximation of logarithms (base 10) where the outer loop is not vectorized. This lookup table consists of 10^4 log values corresponding to the integers 1–10 000. For every new incoming parameter upon which log operation is to be performed, we convert the parameter to a form where the mantissa has 4 digits and calculate the corresponding exponent. We then use this 4 digit mantissa for a $O(1)$ lookup of the above table, and add the exponent to it to obtain the approximate log value. This algorithm of mantissa and exponent determination is not vectorizable due to the presence of loop carried dependency, but the lack of vectorization did not have an impact for the lookup table search for this *log* call since the loop in *radabs* containing this *log* call was not vectorizable in the first place.

For the subranges in the cubic spline functions, the sizes of the lookup tables were also tuned using offline experiments. For each subrange, we chose the lookup table size that provided the best tradeoff of performance and accuracy. Specifically, we chose the table size corresponding to the minimum time while satisfying the accuracy requirement of at most 10^{-4} error value. Table 1 illustrates the choices of the best interpolation functions and lookup table sizes for different invocations of math functions at different locations in *radabs*.

6.4. Other optimizations

We also provided speedup of Xeon Phi executions of *radabs* by exploring different floating point options for the compiler. The option "*-fp-model source*" maintains source precision but is slower since it does not use vector units. The option "*-fp-model source-fast-transcendentals*" essentially calculates the values of transcendental functions (math functions) that are less accurate than with "*fp-model source*", but provides better performance since the option produces vectorized versions of these transcendental functions that help vectorize outer loops. The options "*-fp-model fast* = 1" and "*fp-model fast* = 2" in addition to using vectorized transcendentals, also provide faster versions of other arithmetic operations that are less accurate. Besides, we use "*-fimf-precision* = *high*" which tries to improve the precision with these fast models, while "-fimf -precision = medium" is the default. Table 2 shows the performance with different compiler optimizations (-O2 and -O3) and flags on Xeon Phi with 240 threads. We find that the performance improves as fast-related flags are used. We also find that the use of -O3 does not significantly improve performance over -O2. Hence, in all our experiments, we use -O2 due to its higher accuracy.

In addition to the use of advanced flags, on systems with multiple Xeon Phi cards per node, we also divide the *radabs* computations across all chunks equally among the available Xeon Phi cards to provide additional speedups.

7. Experiments and results

7.1. Experimental setup

In our experiments, we used CESM v 1.2.2 with compset F that involves data model for ocean. We used three different

```
1  do i=begchunk, endchunk
2  |  several physics computations ;
3  end
4  do i=begchunk, endchunk
   |  /* longwaves()                                          */
5  |  if begchunk then
6  |  |  do i=begchunk, endchunk
7  |  |  |  radtpl() ;
8  |  |  |  indep_pre_radabs() ;
9  |  |  end
10 |  |  if doabsems then
11 |  |  |  do i=begchunk, endchunk
12 |  |  |  |  radoz2() ;
13 |  |  |  |  trcpth() ;
14 |  |  |  |  aer_trans_from_od() ;
15 |  |  |  |  radems() ;
16 |  |  |  end
17 |  |  |  begin offload
18 |  |  |  |  do i=begchunk, endchunk
19 |  |  |  |  |  radabs() ;
20 |  |  |  |  end
21 |  |  |  end
22 |  |  end
23 |  end
24 |  indep_post_radabs() ;
25 |  shortwaves() ;
26 end
27 do i=begchunk, endchunk
28 |  dep-post-radabs() ;
29 |  some updates ;
30 end
```

Fig. 5. Synchronous model with early placement of *radabs* computations.

Table 1
Interpolation and lookup table choices in *radabs*.

S. No.	Function call and their locations in the model	Interpolation and lookup table usage
1	*log* functions in *trcab* function	Cubic spline with 16-point lookup table
2	*log* and *power* functions in *goffgratch*	Cubic spline with 151-point lookup table
3	*log*-base 10 in loop 1	10,000-point lookup table with $O(1)$ search
4	*exp* calls in *trcab*	2-point interpolation using Lagrange's formula

Table 2
Performance with different optimization levels and flags of *radabs* on Xeon Phis.

S. No.	Compiler switch	-O2 (in sec)	-O3 (in sec)
1	-fp-model source	2.1	2.05
2	-fp-model source -fast-transcendentals -fimf-precision = high	1.14	1.14
3	-fp-model -fast = 1 -fimf-precision = high	0.7	0.7
4	-fp-model -fast = 2 -fimf-precision = high	0.6	0.59

Table 3
Details of resolutions.

Pseudonym	Resolution	lat × lon (deg)	Columns
R1 (default)	f19_g16	1.9 × 2.5	13 824
R2	f05_g16	0.47 × 0.63	221 184
R3	f02_g16	0.23 × 0.31	884 736

resolutions in our experiments. Table 3 shows the parameters for the resolutions used in the atmosphere model. These resolutions are part of the CESM suite and simulate the entire earth. In all cases, 26 vertical levels were used. All calculations including those on Xeon-Phi were using double precision.

The experiments were conducted on a cluster containing 8 nodes of 16-core (dual octo-core) Intel Xeon E5-2670 CPU with a speed of 2.6 GHz. Each node is equipped with two Intel Xeon Phi 7120 PX cards, each with 61 cores. For performance-related experiments, we use single-node with 16 threads on the CPU for the default R1 resolution due to the small size involved. For the higher R2 and R3 resolutions, we use all the 128 CPU cores of the cluster with 8 MPI tasks and 16 OpenMP threads per MPI task. We use one or both the Xeon Phi cards per node depending on the experiment. In each Xeon Phi, 240 threads were used. The performance-related experiments were conducted with 5-day simulation runs. For the fast-related compiler flags, we use "-fimf-precision = high", implying high precision. Each result shown was obtained as a mean of five runs.

Our results on performance include the data transfer times between the CPU and Intel Xeon Phis. The total data transfer per offload is about 870 MB for input and 90 MB for output. The initial one-time overhead for these data transfers is about 0.9–1.2 s. For every subsequent offload corresponding to an invocation of *radabs*, the data transfer time is about 0.26 s. Overall, the total data transfer overheads in the synchronous model is only about 2.5% of the model execution time. Note that most of the data transfers are completely hidden in the asynchronous model.

7.2. Results on correctness

We first demonstrate the correctness of our code modifications due to various optimizations. We verified the accuracy of the results by finding the error growth of the temperature values produced in the code over the simulations [21]. The error growth curves compare the RMS difference between the results of the original code and the results of the modified code due to our optimizations, and the RMS difference between the results from the original code and the results obtained by perturbing the inputs by the least significant bit. We refer to these perturbations to the least significant bit as *induced perturbations*. In general, for an optimization or modification to be acceptable, the error growth curve due to the modification should be smaller than the error growth curve due to the induced perturbations. Our error growth curves were obtained for a 2-day simulation run with the default R1 resolution. For this, we used a single-thread run since our optimizations strategies are not coupled with the parallelization. For a few runs, we also verified that our single-thread errors matched the multi-thread errors.

We first show the error growths with using the advanced compiler flag of "fp-model fast = 2", which is expected to give fast but less accurate results, over the default flag of "fp-model source", which uses source precision. Fig. 6(a) shows the error results on Xeon. We find that the use of the "fast = 2" optimization did not alter the accuracy significantly. Hence the advanced compiler flags can be safely used to potentially obtain high performance without compromising on accuracy.

Fig. 6(b) shows the error growth on Xeon due to our vectorization related efforts including replacing the non-vectorizable math functions with the Intel MKL functions and using SIMD directive to force vectorization at some locations. These options while improving vectorization are known to yield less accurate results. The results show that the error growths are maintained well below the errors due to induced perturbations, thereby implying the stability of these vectorization-related optimizations.

Fig. 6(c) shows the error growth due to our optimization of replacing the expensive *log*, *log* − *base*10, *exp* and *goffgratch* functions with low-cost interpolation functions and lookup table methods. We find that our interpolation and lookup table methods induce notable errors when compared to the induced perturbations. To evaluate the reasonableness of the errors, we compare with a method that replaces all the expensive functions with simple linear spline interpolations, since such linear spline interpolations are used in many other locations in CESM and other scientific applications. For this linear spline interpolation method, we created four lookup tables corresponding to the four math functions, for input values from 0 (minimum range for the inputs) to 325 (maximum range for the inputs) with increments of 1. For every new input value with a math function, a binary search of the corresponding lookup table was performed in order to find the range in which the input value lies. Then, linear interpolation using Lagrange's classic interpolation formulae was applied. Fig. 6(d) shows the error growth due to such linear spline interpolations. We find that the simple linear spline interpolation method that is commonly used for approximations yields huge errors, and in comparison, our method of combination of different low-cost interpolations for different ranges and functions and lookup table search yielded reasonably accurate results. In fact, the use of simple linear spline interpolations gave degradation in performance by 18%–20% on Xeon while our low-cost methods yielded performance improvement of 15%. This is due to the non-vectorizable characteristics of the simple linear spline interpolation function.

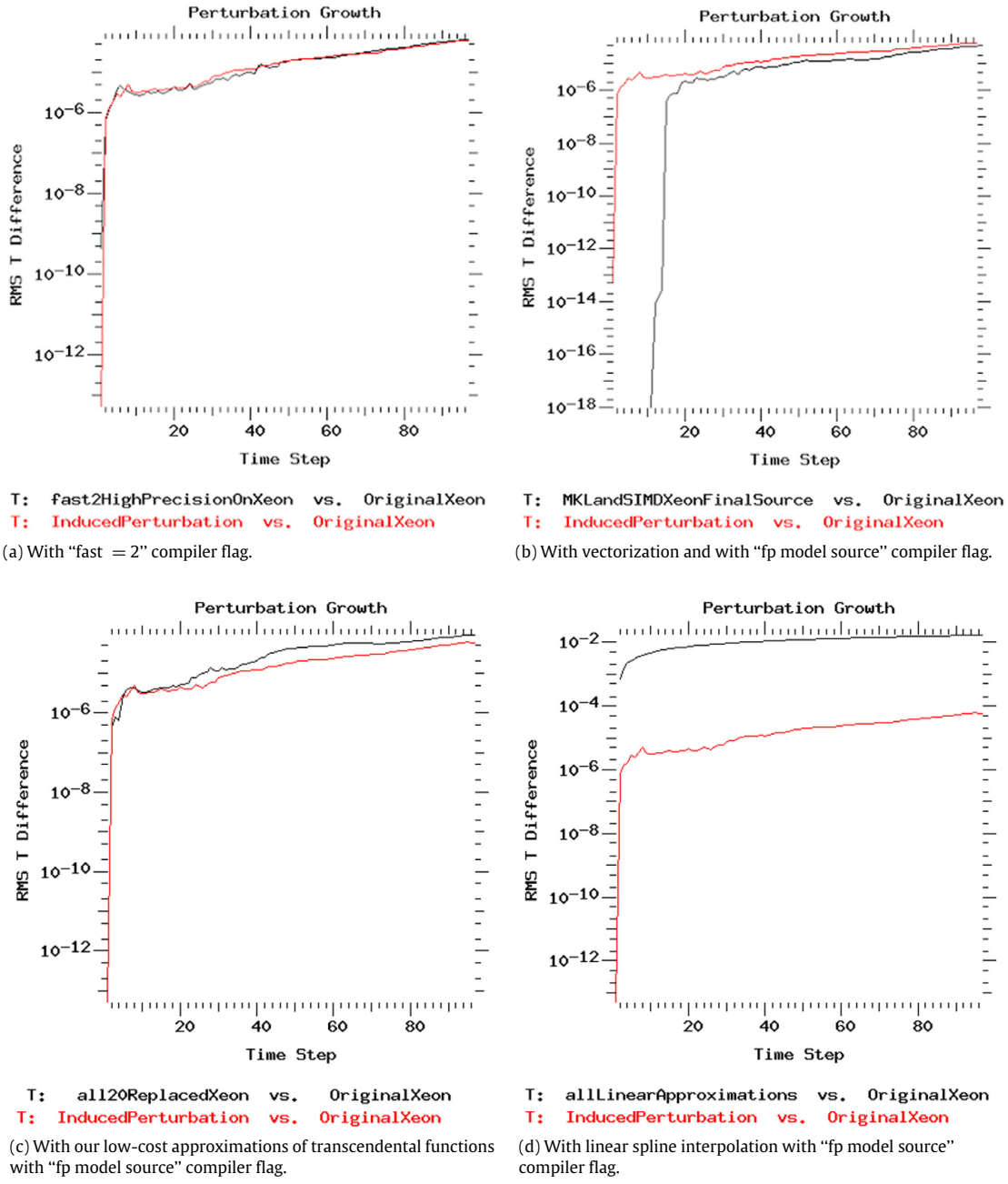
7.3. Performance of asynchronous model

Fig. 7 shows the performance of our asynchronous model of offloading *radabs* to Xeon Phi with respect to the original Xeon CPU executions for different resolutions with increasing frequencies of invocations (*x*-axis). In the asynchronous execution model, we only want the Xeon Phi to complete the *radabs* computations by the time the CPU reaches the next time step and is ready to consume the *radabs* output. By means of using the *offload wait* directive, we confirmed that the waiting time incurred by the CPU is zero in all the cases. We find in the figure, that in all cases, the *radabs* times are almost completely subtracted out from the CPU execution times.

We also find that the performance improvements for the entire atmosphere model increase with increasing *radabs* frequencies. For example, for the default R1 resolution, the performance improved from 3% to 45% for the atmosphere model as the frequency increases from 12 to 1 h. This is beneficial since now the climate models can afford to execute these important, but expensive, radiation calculations at the same frequency as the other calculations. We also find that the performance improvements are significant for all resolutions. For example, for 1 h frequency, the performance improvement of the atmosphere model was 14%–42%.

7.3.1. Savings for multi-century runs in asynchronous model

Multi-century simulation runs are typically of interest in climate models to study the long-term effects on the climate due to various factors including CO_2 levels. We also extrapolated the performance gains obtained in our *radabs* computations due to asynchronous executions on Xeon Phi for a multi-century



(a) With "fast = 2" compiler flag.

(b) With vectorization and with "fp model source" compiler flag.

(c) With our low-cost approximations of transcendental functions with "fp model source" compiler flag.

(d) With linear spline interpolation with "fp model source" compiler flag.

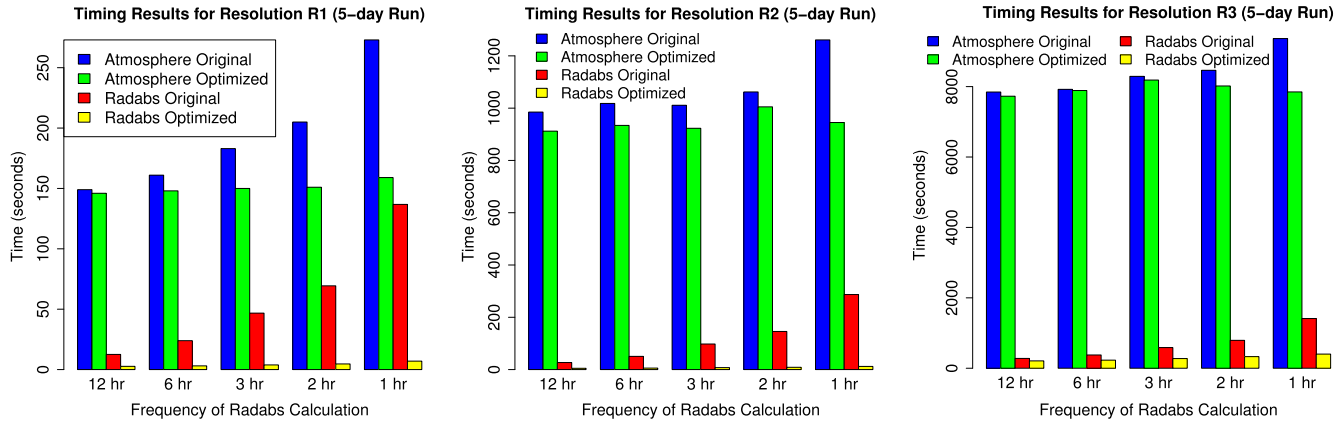
Fig. 6. Error growth curves on Xeon.

Table 4
Original code time and savings in execution days for multi-century runs with asynchronous model and "fp model source" compiler flag.

radabs frequency (h)	R1		R2		R3	
	Original (Days)	Savings (Days)	Original (Days)	Savings (Days)	Original (Days)	Savings (Days)
12	10.20	8.34	22.72	18.59	89.39	61.00
6	20.23	17.64	42.65	37.89	169.66	128.68
3	39.50	36.25	82.63	76.25	329.18	270.12
2	58.60	54.66	123.19	115.84	488.44	407.75
1	115.57	109.66	242.49	232.31	962.69	817.79

simulation run using our runs for limited number of simulation days. Specifically, we obtained the performance gains in seconds due to our asynchronous model over the default CPU execution for a 5-day simulation run, and extrapolated the gains in terms of days for a 1000-year simulation run. Table 4 shows savings in terms of the number of days for execution for different *radabs* frequencies and different resolutions. The table also shows the extrapolated days for execution for the original code.

We find that the use of our asynchronous model results in highly significant savings in execution days. As shown above, the performance and hence the savings increase with increase in *radabs* frequencies and for increasing resolutions. For the desired frequency of one hour and highly needed highest resolution for



(a) R1 resolution. (b) R2 resolution. (c) R3 resolution.

Fig. 7. Asynchronous model for different frequencies and resolutions with "fp model source" compiler flag.

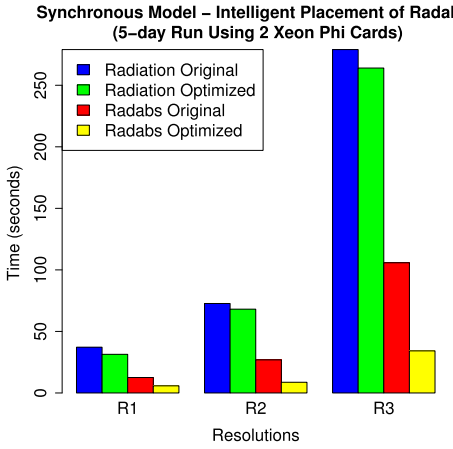


Fig. 8. Synchronous model—2 Xeon Phis with "-fp-model fast = 2" compiler flag.

modeling studies, the use of our asynchronous model results in savings of up to 817 days or 2.24 years in execution. These are highly significant savings and imply not only improved performance, but also savings in power consumption, electricity and maintenance costs.

7.4. Performance of synchronous model

Our synchronous model primarily consists of our optimization of early placement of *radabs* in the code for early offloads. Fig. 8 shows the performance benefits obtained with synchronous model for *radabs* and radiation calculations. The results correspond to execution with 2 Xeon Phi cards. We find that when compared to the original code, the synchronous model results in 50%–70% performance improvements in *radabs* and 6%–20% improvements in total radiation calculations.

7.4.1. Compilation flags

Fig. 9 shows the performance of *radabs* calculations with synchronous model when using different compiler flags on both Xeon and Xeon Phi for different resolutions. We find that the use of advanced flags like "fast = 2" improved the performance from 25%–30% on Xeon, and from 62%–70% on Xeon Phi. Thus, the compiler flags play a much more significant effect on Xeon Phi than on Xeon.

7.4.2. Vectorization and low-cost approximations of transcendental functions

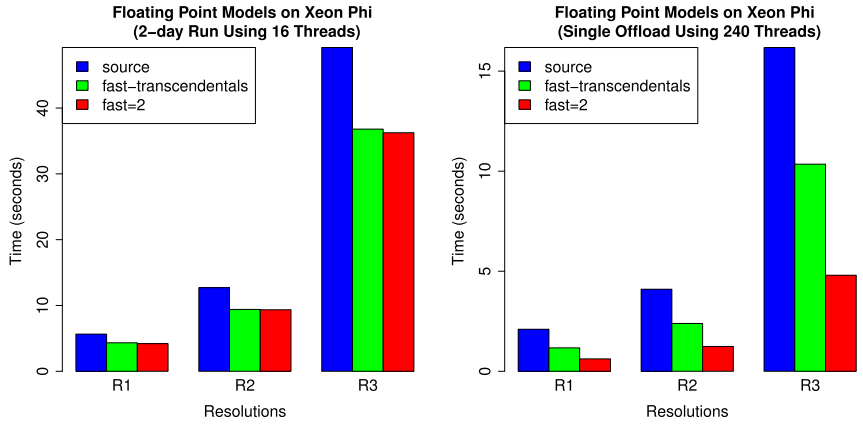
Fig. 10(a) shows the performance with synchronous model due to our vectorization related optimizations on both Xeon and Xeon Phi. We find that our vectorization-related optimizations result in 11%–20% improvement in performance on Xeon, and 25%–30% improvement on Xeon Phi. Vectorization provides higher benefits in Xeon Phi due to wider vector units. Thus vectorization can serve as an important optimization strategy in future architectures where large vector units are envisaged.

Fig. 10(b) shows the performance on both Xeon and Xeon Phi with synchronous model due to our optimizations related to low-cost substitutions, namely, use of interpolation functions and lookup table methods. We find that our optimizations due to these low-cost substitutions result in 12%–20% improvement in performance on Xeon, and 15%–20% improvement on Xeon Phi.

7.5. Asynchronous vs. synchronous models

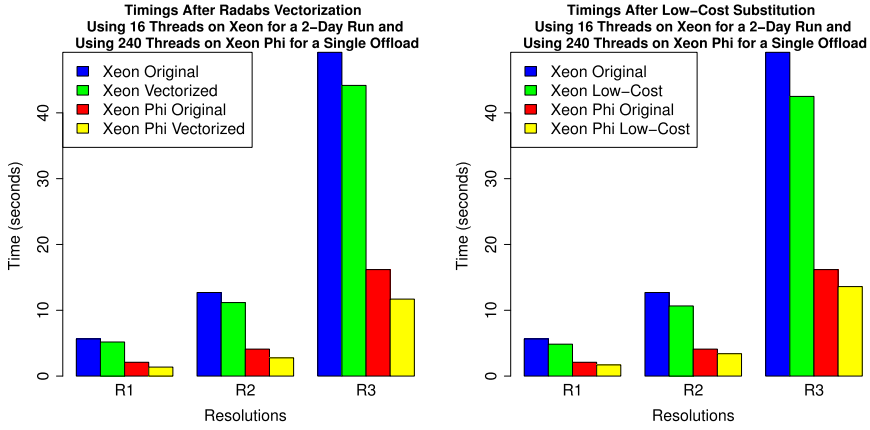
In this section, we compare both the models of execution in different aspects. In general, the model to choose for a particular application depends on factors including performance, accuracy of results, code complexity and other factors.

1. Scalability: Fig. 11 shows the execution times for the atmosphere model with both the synchronous and asynchronous models for 32, 64 and 128 cores for the R2 resolution for the 12 h *radabs* frequency. We find that both the models provide good scalability. For example, the speedup of execution on 128 cores with respect to the execution on 32 cores is about 3 for both the synchronous and asynchronous models.
2. Accuracy vs. Performance Trade-off: The asynchronous execution model is expected to provide better performance since the *radabs* calculations are completely masked in the Xeon Phi while the CPU proceeds with the rest of the calculations. This is illustrated in Fig. 12 which shows the times spent by the CPU for the *radabs* calculations on the Xeon Phi in both the models for 12 h *radabs* frequency. As can be seen, the asynchronous execution model in which the CPU spends time only for the offloading overheads, provides 3X–5X performance improvement over the synchronous model.
   However, as mentioned earlier, the asynchronous model could be considered to give lesser accuracy if evoked at the same frequency as the original, since the *radabs* result for a time step is only used in the next time step. Fig. 13(a) and (b) show the error growths for the asynchronous and synchronous execution models, respectively, for a 12 h frequency of long-wave



(a) Xeon. (b) Xeon Phi.

Fig. 9. *radabs* timings with synchronous model for different flags.



(a) Vectorization. (b) Low-cost substitutions.

Fig. 10. Synchronous model—benefits due to vectorization and low-cost substitutions on both Xeon and Xeon Phi with "fp model source" compiler flag.

radiation calculations. The synchronous execution model is the baseline model consisting of only the intelligent placement of *radabs* calculations, and not the other optimizations. Both the results were obtained on Xeon Phi with "fp model source" compiler flag. We find that while the asynchronous execution model exhibits errors in calculations since the results of *radabs* are used only in the next time step, the errors are reasonably small. We find that even these errors are avoided in the synchronous execution model, where the error growth matches well with the induced perturbations.

We also directly compare the accuracy of synchronous and asynchronous execution models. Fig. 13(c) shows the comparison of the errors between the synchronous model with 12 and 1 h frequency of radabs calculations, and the errors between the synchronous model and the asynchronous model both with 1 h frequency. We find that the errors between the asynchronous and synchronous models for 1 h frequency are significantly smaller than the errors between the synchronous model for the default 12 h frequency and the synchronous model for 1 h frequency. This implies that it is advantageous to execute the asynchronous model at the highest frequency of 1 h since it results in less errors than the synchronous model at the default frequency of 12 h. And, as seen in the earlier performance results, executing the asynchronous model at the highest frequency gives higher performance than executing the synchronous model at the same highest frequency.

3. Asynchronous model can be a more generic model for climate modeling codes, since they contain processes which have different timescales due to the different computational complexities. In such cases, slower time-scale processes can be performed asynchronously on Xeon Phis. However, the synchronous model may be hard to apply as it depends on the existence of independent computations which can be overlapped in order to get minimum waiting time.
4. Code changes for synchronous model can involve complex movements. Most of the code for asynchronous model was directly usable for synchronous execution. Further complexity is added by the optimizations related to vectorization and low-cost substitutions.
5. Utilization of Xeon in case of asynchronous model is more than that in the case of synchronous model, as the CPUs wait for Xeon Phi computations to complete in the synchronous model.
6. In applications, where sufficient data can be collected at the beginning of the time step, synchronous model of execution can be the first choice to explore. However, in applications where the data slowly builds over the time step, the asynchronous model is preferable since the data can be aggregated and offloaded at the end of the time step so that it can be used in the next time step.

8. General principles for other scientific applications

While we had primarily worked on and demonstrated our results for a climate modeling application, we can derive a number of general principles that can be applied for other scientific applications.

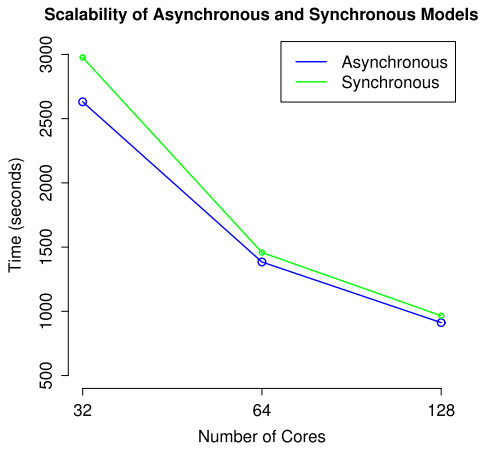


Fig. 11. Execution times for atmosphere model for different cores (R2 resolution with 12 h *radabs* frequency).

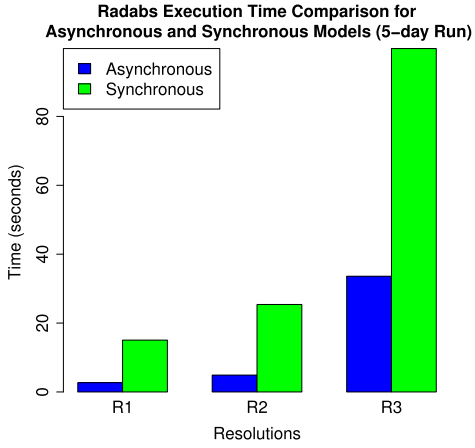
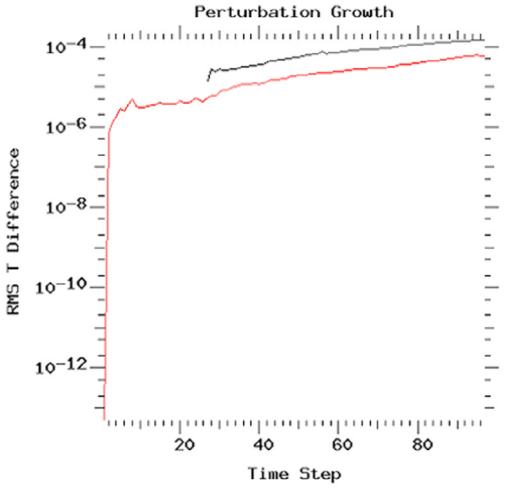
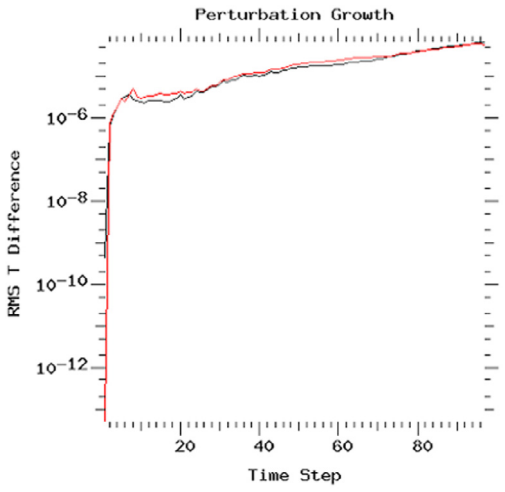


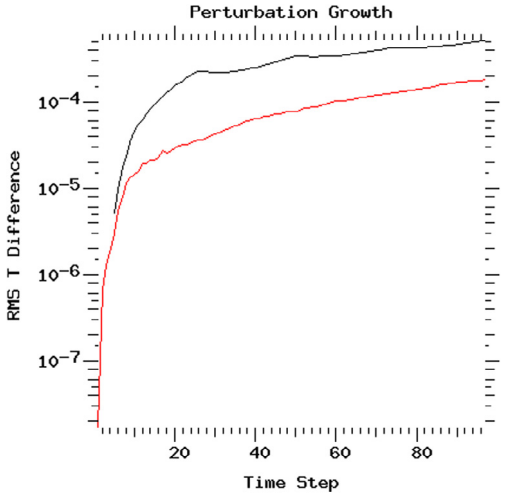
Fig. 12. Execution times of *radabs* for different resolutions (12 h *radabs* frequency).



(a) Asynchronous execution model with "fp-model source" flag.



(b) Synchronous execution model with "fp-model source" flag.



(c) (Async and Sync 1 h errors) vs. (Sync 12 h and Sync 1 h errors).

Fig. 13. Error growth with synchronous and asynchronous execution models on Xeon Phi.

When a candidate function for offloading is invoked in a loop, the simplest strategy is to insert offloading directives above the function. However, this can result in limited parallelism for the computations in the function on Xeon. This will also lead to multiple offloads corresponding to multiple iterations, resulting in large offloading and data transfer overheads. While this fine-level parallelism and offloading can be adopted if each function execution has sufficient scope of parallelism, in general, aggregating computations and data across multiple function calls in the loop and offloading the aggregated computations can expose significant parallelism on Xeon Phi and result in smaller offloading overheads.

As noted above, asynchronous execution model is an attractive model for completely masking the offloaded computations with the CPU computations for slowly-varying functions. Domain knowledge can be used to identify such functions for which the results of the computations in the current time step may be used in the subsequent time steps without considerable loss in accuracy.

When considering candidate functions for offloading, care should be taken such that tuning the application parameters for efficient offloading of a function does not impact the performance of the other application modules executing on the CPU. This was seen in our climate modeling example, where an easier way for offloading *radabs* would have been to promote large parallelism for a single *radabs* invocation by increasing the columns per chunk parameter in CESM. However, large chunk sizes result in load imbalances in the other modules of CESM, namely in the physics routines. Hence, we adopted the aggregation strategy for increasing the parallelism for *radabs* offloading.

The computations for offloading should be offloaded to Xeon Phi as early as possible in a time step so as to maximize the computations that can be performed on the CPU simultaneously with the offloaded computations. This involves careful identification and segregation of the other computations into feeder computations on which the offloaded computations depend and independent computations. The feeder computations, if present in a loop, should be extracted and performed for all iterations at once by performing loop fission such that all the data needed for the computations that will be offloaded are available at the earliest.

Use of architecture-tuned and highly optimized native math libraries and functions like the Intel MKL will have to be encouraged especially when encountering math functions that are profiled to be expensive. Our error plots show that these libraries help in increased performance without much loss in accuracy. The speedup and accuracy provided by these libraries will have to be compared with the default math functions in the original code using different compiler flags with varying optimization levels.

In general, transcendental functions that are widely prevalent in many large scientific applications can act as major bottlenecks in performance as shown in our results. While most modern compilers provide optimization flags to provide both high performance and accuracy for these transcendentals, we have demonstrated that these transcendentals can also be approximated with combination of different low cost interpolation functions and lookup table methods with reasonable accuracy. Performance-accuracy tradeoffs of compiler flags and optimizations for these transcendentals vis-a-vis the low-cost approximations will have to be carefully evaluated for optimal selections.

We found loop fission of large loops as an important technique in our strategies, primarily for many of the vectorizations of the offloaded codes as well as early placement of computations to be offloaded. Such large loops are present in many scientific codes. While our method involved a combination of careful analysis of the blocks of the loop and experiments to identify the appropriate locations in the loops for loop fission, in general, building a generic tool that automatically identifies such locations for loop fission primarily for improving vectorization will be highly useful for the scientific community. Building such a tool can be part of our future work.

9. Conclusions and future work

In this work, we successfully offloaded the time-consuming long-wave radiation calculations in the atmosphere model of CESM to Xeon Phi. We performed both asynchronous and synchronous models of executions, and implemented several optimizations including intelligent reorganization of code, early start of offloads, vectorization techniques and use of low-cost interpolation functions in place of expensive math functions.

Our asynchronous execution model resulted in performance improvement of up to 45% in the atmosphere model. This resulted in significant savings in wall-clock execution time of up to 2.25 years for multi-century climate simulations. The time savings also translate to reduction in power consumption, electricity and maintenance costs of the resources. Our synchronous execution model resulted in performance improvements of about 10% in the atmosphere model. Our vectorization strategies provided up to 30% performance improvement on Xeon Phi and is expected to increase with future wider vector units. Our low-cost approximations of transcendental functions using interpolation functions and lookup table methods provided up to 20% performance improvement. Compiler flags and optimization levels were shown to have major impact on performance, and will have to be weighed with our low-cost approximations in accelerating transcendentals for performance-accuracy tradeoffs.

Accelerating large legacy scientific applications on Xeon Phi requires a kernel-by-kernel approach for comprehensive set of optimizations. In this work, we had worked on the most time-consuming physics kernel in CESM. In future, we plan to adopt similar strategies for other kernels and components of CESM and provide large-scale improvements for the entire CESM on Intel Xeon Phi architectures. We also plan to explore our optimizations in the future Intel Xeon Phi architecture of Knights Landing.

Acknowledgments

This work is funded by the Intel Parallel Computing Center (IPCC) India project, Grant No. INTC/MAS/RSN/0001. We would like to thank the anonymous reviewers for their valuable comments that helped to improve the quality of the paper.

References

[1] V. Betro, R. Harkness, B. Hadri, H. You, R. Hulguin, R. Brook, L. Crosby, Performance metrics and application experiences on a cray CS300-AC cluster supercomputer equipped with Intel Xeon Phi coprocessors, in: In Proceedings of the Cray User Group (CUG) Conference, 2013.

[2] I. Carpenter, R. Archibald, K. Evans, J. Larkin, P. Micikevicius, M. Norman, J. Rosinski, J. Schwarzmeier, M. Taylor, Progress towards accelerating HOMME on hybrid multi-core systems, Int. J. High Perform. Comput. Appl. (IJHPCA) 27 (3) (2013) 335–347.

[3] W. Collins, P. Rasch, B. Boville, J. Hack, J. McCaa, et al. The Formulation and Atmospheric Simulation of the Community Atmosphere Model Version 3 (CAM3), Vol. 19, 2006, pp. 2144–2161.

[4] J. Hurrell, et al., The community earth system model: A framework for collaborative research, Bull. Am. Meteorol. Soc. 94 (9) (2013) 1339–1360.

[5] R. Neale, et al., The mean climate of the community atmosphere model (CAM4) in forced SST and fully coupled experiments, J. Clim. 26 (14) (2013) 5150–5168.

[6] O. Fuhrer, C. Osuna, X. Lapillonne, T. Gysi, B. Cumming, M. Bianco, A. Arteaga, T. Schulthess, Towards a performance portable, architecture agnostic implementation strategy for weather and climate models, Supercomput. Front. Innov. 1 (1) (2014).

[7] H. Goose, P. Barriat, W. Lefebvre, M. Loutre, V. Zunz, Introduction to Climate Dynamics and Climate Modelling, Cambridge-University Press, 2010, Online textbook available at http://www.climate.be/textbook.

[8] M. Govett, J. Middlecoff, T. Henderson, Running the NIM next-generation weather model on gpus, in: IEEE/ACM Internation Conference on Cluster, Cloud and Grid Computing, CCGrid, 2010.

[9] S. Heybrock, B. Joo, D. Kalamkar, M. Smelyanskiy, K. Vaidyanathan, T. Wettig, P. Dubey, Lattice QCD with domain decomposition on Intel Xeon Phi co-processors, in: International Conference for High Performance Computing, Networking, Storage and Analysis, SC14, 2014.
[10] J. Jeffers, J. Reinders, Intel Xeon Phi Coprocessor High-Performance Programming, Morgan Kaufmann, 2013.
[11] A. Laksono, et al., HPCToolkit: Tools for performance analysis of optimized parallel programs, Concurr. Comput.: Pract. Exper. 22 (6) (2010) 685–701.
[12] Y. Liu, T. Tran, F. Lauenroth, B. Schmidt, SWAPHI-LS: Smith-Waterman algorithm on Xeon Phi coprocessors for long DNA sequences, in: IEEE International Conference on Cluster Computing, CLUSTER, 2014.
[13] R. Luo, J. Cheung, E. Wu, H. Wang, S.-H. Chan, et al., MICA: A fast short-read aligner that takes full advantage of many integrated core architecture (MIC), BMC Bioinformatics 16 (7) (2015) 1–8.
[14] J. Michalakes, M. Iacono, D. Berthiaume, Optimizing weather model radiative transfer physics for the many integrated core and GPGPU architectures, in: Heterogeneous Multi-Core Workshop, 2014.
[15] J. Michalakes, M. Vachharajani, GPU acceleration of numerical weather prediction, in: IEEE International Symposium on Parallel and Distributed Processing, IPDPS, 2008.
[16] J. Mielikainen, B. Huang, A. Huang, Optimizing the updated Goddard shortwave radiation Weather Research and Forecasting (WRF) scheme for Intel Many Integrated Core (MIC) architecture.
[17] J. Mielikainen, B. Huang, A. Huang, Performance tuning Weather Research and Forecasting (WRF) Goddard longwave radiative transfer scheme on Intel Xeon Phi.
[18] J. Mielikainen, B. Huang, A. Huang, Revisiting Intel Xeon Phi optimization of Thompson cloud microphysics scheme in weather research and forecasting (WRF) model, in: Proc. SPIE 9646, High-Performance Computing in Remote Sensing V.
[19] J. Reinders, J. Jeffers, High Performance Parallelism Pearls, Vol. 1, Morgan Kaufmann, 2015.
[20] J. Reinders, J. Jeffers, High Performance Parallelism Pearls, Vol. 2, Morgan Kaufmann, 2015.
[21] J. Rosinski, D. Williamson, The accumulation of rounding errors and port validation for global atmospheric models, SIAM J. Sci. Comput. 18 (2) (1997) 552–564.
[22] J. Schalkwijk, H. Jonker, A. Siebesma, E.V. Meijgaard, Weather forecasting using GPU-based Large-Eddy simulations, Bull. Am. Meteorol. Soc. 96 (2014) 715–723.
[23] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, S. Matsuoka, An 80-fold Speedup, 15.0 Tflops full GPU acceleration of non-hydrostatic weather model ASUCA production code, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'10, 2010, pp. 1–11.
[24] C. Wilcox, M. Strout, J. Bieman, Mesa: Automatic generation of lookup table optimizations, in: Proceedings of the 4th International Workshop on Multicore Software Engineering, IWMSE'11, 2011.
[25] H. Zhang, J. Garcia, GPU acceleration of a cloud resolving model using CUDA, in: Proceedings of the International Conference on Computational Science, ICCS, 2012, pp. 1030–1038.

Amlesh Kashyap is a member of the Middleware and Runtime Systems Laboratory, at the Department of Computational and Data Sciences, Indian Institute of Science. He obtained his B.Tech. degree in Computer Science and Engineering from the National Institute of Technology, Patna in 2015. His research interests are in high performance computing.

Sathish S. Vadhiyar is an Associate Professor in the Department of Computational and Data Sciences, Indian Institute of Science. He obtained his B.E. degree from the Department of Computer Science and Engineering at Thiagarajar College of Engineering, India in 1997 and received his Master's degree in Computer Science at Clemson University, USA in 1999. He graduated with a Ph.D. from the Computer Science Department at University of Tennessee, USA in 2003. His research areas are building application frameworks including runtime frameworks for irregular applications, hybrid execution strategies, and programming models for accelerator-based systems, processor allocation, mapping and remapping strategies for networks for different application classes including irregular, multi-physics, climate and weather applications, middleware for production supercomputer systems and fault tolerance for large-scale systems.

Ravi S. Nanjundiah is Chair and Professor at the Centre for Atmospheric & Oceanic Sciences (CAOS), Indian Institute of Science (IISc), Bengaluru, India. He is also associated with the Divecha Centre for Climate Change. His research areas are study of monsoons using earth system models, and application of HPC to earth system modeling.

P.N. Vinayachandran is a Professor of Oceanography at the Centre for Atmospheric and Oceanic Sciences, Indian Institute of Science, Bangalore, India. He did his B.Sc. in Physics from Christ College, Irinjalakuda and M.Sc. in Oceanography from Indian Institute of Science. He received his Doctorate Degree in Oceanography in 1996 from Indian Institute of Science. His research areas are study of dynamics of the Indian Ocean, ocean modeling, and physical–biological interactions in the ocean. He is the recipient of the Shanti Swarup Bhatnagar Award in 2008.