2016 IEEE 23rd International Conference on High Performance Computing

High performance Horizontal Diffusion Calculations in Ocean Models on Intel® Xeon Phi™ Coprocessor Systems

[1]Aketh TM, [1]Sathish Vadhiyar, [2]PN Vinayachandran, [2]Ravi Nanjundiah
[1]Department of Computational and Data Sciences, Indian Institute of Science, Bangalore, India
[2]Centre for Atmospheric and Oceanic Sciences, Indian Institute of Science, Bangalore, India
aketh.tm@gmail.com, vss@cds.iisc.ac.in, vinay@caos.iisc.ernet.in, ravi@caos.iisc.ernet.in

***Abstract*—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. However, the use and demonstration of these coprocessors for climate modeling are limited. In this work, we have developed a comprehensive set of novel techniques for efficient use of Intel Xeon Phi coprocessors for ocean modeling. In particular, we focus on one of the most time consuming routines, namely, horizontal diffusion in tracers ($hdifft$). Our techniques include explicit and implicit fusion for data locality and vectorization, choice of coarse-grained over fine-grained parallelism, offloading $hdifft$ function for asynchronous and simultaneous executions on CPU and accelerator cores, and effective data management to minimize CPU-accelerator data transfer overheads. Our comprehensive set of techniques has resulted in about 17-23% improvement in simulation throughput of the entire ocean model code. Our optimization strategies exhibit good scaling with the use of more Intel Xeon Phi processors. Among our optimization techniques, the use of our novel look-ahead asynchronous execution strategy on Intel Xeon Phis resulted in maximum benefit yielding about 36% improvement in performance.**

***Keywords*—Intel Xeon Phi co-processors; offloading; ocean modeling; horizontal diffusion**

I. Introduction

The importance of climate to energy usage and agriculture has made it a prominent field of study. Climate study is carried out based on the mathematical models of the physical processes. Climate models predict temperature changes, winds, radiation, relative humidity and other such factors. 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 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.

One such climate model is the Community Earth System Model [1], 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.

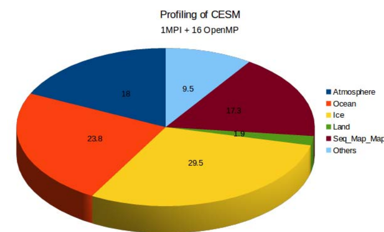


Fig. 1. Execution Profile of CESM

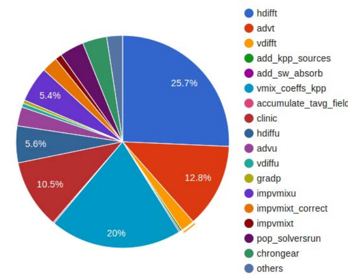


Fig. 2. Execution Profile of the POP model

Along with the atmosphere component, the ocean is the most-time consuming model in CESM, as shown in Figure 1. The execution profile is obtained by running CESM on a single 16-core Intel Xeon node with 16 threads. The model used for ocean is the Parallel Ocean Program version 2 (POP2) [2], [3]. Figure 2 shows the breakdown of the execution time of the ocean model in CESM when executed on a cluster of 8 nodes of 16-core Xeon processors with 128 threads. We find that one of the time consuming routines in the ocean model is the horizontal diffusion in tracers ($hdifft$), which occupies about 26% of the total execution time. This routine performs the Gent-McWilliams parameterization of horizontal diffusion of tracers to generate temperatures and salinity fields in the ocean [4]. Ocean codes are typically major bottlenecks in climate models since ocean processes are slower

978-1-5090-5411-4/16 $31.00 © 2016 IEEE
DOI 10.1109/HiPC.2016.29

IEEE computer society

relative to other components. Further, ocean codes typically require longer spinup (100+ simulated years) than the other components leading to higher computational requirements. Hence, optimizing ocean code is of major importance.

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 [5]–[7]. However, the use and demonstration of these coprocessors for climate modeling are limited.

In this work, we have developed a comprehensive set of novel techniques for efficient use of Intel Xeon Phi coprocessors for horizontal diffusion in tracers in the ocean model of CESM. Our techniques include explicit and implicit fusion for data locality and vectorization. In this optimization, we convert the WHERE blocks with implicit indices into explicit loops and combine multiple such blocks. We also perform a novel look-ahead asynchronous offloading strategy in which the $hdifft$ computations for the next step are offloaded for asynchronous executions on Xeon Phi while the CPU proceeds with the current time step calculations. We also performed effective data management using *nocopy* and *allocatables* options for data to minimize CPU-coprocessor data transfer overheads.

We show that the use of Intel Xeon Phi with our optimizations described in this work yield about 25-42% reduction in execution time over the CPU-only run for the baroclinic phase containing the $hdifft$ routine that we optimize in this work and about 17.4-23% reduction in execution time for the entire POP model. We also show that our optimization strategies exhibit good scaling with the use of more Intel Xeon Phi processors. Among our optimization techniques, the use of our novel look-ahead asynchronous execution strategy on Intel Xeon Phis resulted in maximum benefit yielding about 36% performance improvement over the synchronous strategy. With our simultaneous execution model in which the results of the horizontal diffusion routines for a time step are used in the subsequent time steps, we also show error growths and demonstrate reasonable accuracy.

Our work, to the best of our knowledge is the first method for this type of ocean model. It should be noted that an ocean model is a complex code running into about 1,20,000 lines of code and needs representation of many complex processes such as advection, turbulence, diffusion etc. The optimizations described in this work can be easily generalized to other ocean models with horizontal diffusion modules and can also be used in any physical model which includes multiple processes. While the results are specific to diffusion calculations in particular in climate models, similar techniques can be applied in other fields including CFD that contain opportunities for vectorization, and asynchronous executions.

In Section II, we give the overall structure of the ocean model and $hdifft$, and a brief background on Intel Xeon Phi accelerators. Section III covers related work in the areas of using accelerators and co-processors for climate and weather models. In Section IV, we describe our different optimization techniques for Intel Xeon Phis. Section V presents experiments and results. Section VII gives conclusions and presents scope for future work.

II. Background

A. Ocean Model

Figure 3 gives the overall structure of the ocean model. The overall ocean grid is divided into blocks of multiple vertical levels. As the figure shows, the ocean model contains two primary phases, namely *baroclinic* and *barotropic*. POP uses mode splitting method for solving momentum equations. The calculations of the vertically averaged velocity in the ocean is commonly referred to as the barotropic mode and the part that depends on the vertical co-ordinates is called the baroclinic mode. The baroclinic mode describes the three-dimensional dynamic and thermodynamics processes, and the barotropic mode solves the vertically-integrated momentum and continuity equations in two dimensions. The barotropic mode travels faster and therefore requires a smaller time-step compared to the baroclinic. Separating the mode thus helps in using larger time-steps for the baroclinic modes.

The baroclinic mode consists of two main phases related to updates of tracers and solving momentum equations. The baroclinic phases are performed for all vertical levels. The horizontal diffusion in tracers is calculated in the function $hdifft$ as part of the tracer update phase of the baroclinic driver. The closure of equations of tracer balance in ocean models requires diffusion of tracers in both the horizontal and vertical directions. In addition, the diffusion terms help to regulate the numerical noise generated. In the deep ocean, mixing along surfaces of constant density is found to be stronger than that in the vertical co-ordinate. In addition, significant amount of mixing and diffusion is achieved by eddies. The Gent-McWilliams parameterization of horizontal diffusion has been found to generate realistic temperature and salinity fields in the ocean as well as contribute to reducing the long-term drift in non-eddy resolving version of CESM [4]. In the Matsuno time-stepping scheme used in POP (a Euler forward-backward procedure), a forward prediction (pass 1) is first carried out and the variables are saved as *mixtime tracer*. In the next step, the mixing terms are calculated (pass 2) using the mixtime tracers and the new variables are labeled *current time tracers*. Parallelism of the ocean model is challenging since many functions have low arithmetic intensity.

B. Intel Xeon Phi Architecture

Intel's first generation Many Integrated Core (MIC) architecture 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 upto 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

```
1  foreach time step do
       /* BAROCLINIC DRIVER                          */
       /* Tracer update loop                         */
2      do iblock=1,nblocks_clinic
3          do k=1,nverticals
4              vmix_coeffs (...) ;
5              hdifft (...) ;
6              advt (...) ;
7              vdifft (...) ;
8          end
9      end
       /* Momentum equations solver loop             */
10     do iblock=1,nblocks_clinic
11         do k=1,nverticals
               /* Solve momentum equations           */
12         end
13     end
       /* BAROTROPIC DRIVER                          */
14     do iblock=1,nblocks_clinic
15         ...
16     end
       /* Other functions                            */
17 end
```

Fig. 3. Pseudocode for Ocean Model

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 that provide additional speedup for workloads that have these computations.

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 small. 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.

III. 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 [8] used GPUs to improve the performance of the Weather Research and Forecast (WRF) model. Their work resulted in in 5-20x speed-up for the computationally intensive routine WSM5. In the work by Govett et al. [9], the non-hydrostatic icosahedral (NIM) model was ported to the GPU [9]. 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 [10]. 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 [11] 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.

Related to high performance for ocean modeling, there have been efforts on both distributed memory architectures and GPUs. In a recent work, Hu et al. [12] implemented a pre-conditioned Chebyshev-type iterative method in POP with reduced global reductions. They demonstrate 5.2x speedup on high resolution POP when implemented on distributed memory systems. Bleichrodt et al. [13] implemented a numerical solver for the barotropic vorticity equation on a GPU. Werkhoven et al. [14] proposes a new distributed computing approach with block partitioning scheme and hierarchical load balancing for the POP model on multiple GPU clusters. They primarily focus on two components of POP, namely, equation of state and vertical mixing coefficients. Their load balancing schemes take into account communication hierarchy in the clusters, and their GPU approach considers asynchronous data transfers and memory mapping. In our work, we consider a stronger model with horizontal mixing. While these efforts focused on barotropic phase of POP, our work focuses on the baroclinic phase, in which horizontal diffusion in tracers, the primary focus of our work, is one of the major performance bottlenecks.

In the work by Xu et al. [15], the authors redesign mpiPOM with GPU's. They covert the code from Fortran to CUDA C and optimize on GPUs using methods like loop and function fusion, improved utilization of read only data cache and L1 cache, optimizing communication among multiple GPUs, I/O optimizations between GPUs through overlapping I/O and computation. Garcia et al. [16] 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, and utilizing the GPU's texture and shared memory. Fuhrher et al. [17] 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 [5]–[7]. 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. 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 1.8x over the original code on Intel Xeon Phi 7120P and on dual socket

configuration of eight core Intel Xeon E5-2670. In another work [19], 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 [20]. They observe a speedup of 1.3x over the original code on Xeon Phi 7120P.

Betro et al. [21] 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 a variety of programming paradigms to fit the needs of all applications developer.

Michalakes et. al [22] 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 1.3x over the original code on Xeon Sandybridge and 3x over the original code on Intel Xeon Phi.

To our knowledge, ours is the first effort on accelerating POP model on Intel Xeon Phi clusters. 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 CPU and coprocessor cores.

IV. METHODOLOGY

In our work, we perform various optimizations for speeding up the tracer calculations using Intel Xeon Phi. These include loop fusion for cache locality, asynchronous offloads of computations for simultaneous executions on Xeon Phi, and efficient data management strategies.

A. Explicit and Implicit Loop Fusion

Most of the loops in the Fortran codes of horizontal diffusion in tracers ($hdifft$) have WHERE conditional blocks and implicit indexing as shown in Figure 4. The effect of the WHERE block is that each statement within the block is converted to a loop by the compiler. Figure 5 shows the equivalent translation of the WHERE block shown in Figure 4. As shown in the figure, this transformation results in a very inefficient code with each statement in the WHERE block expanded as a loop with separate conditionals. This also leads to poor cache efficiency since the data elements corresponding to a large array are swapped out of and in to the cache across subsequent loops. We found that even the use of $-O3$ optimization does not improve this compiler transformation. The implicit indices in the array also prevent parallelization of the WHERE blocks by the OpenMP threads.

To improve cache efficiency and promote parallelism, we converted each WHERE block in the horizontal diffusion in tracers to DO loops with explicit indices and converted the arrays to use these indices. The transformation is shown in Figure 6 for the code shown in Figure 4. We refer to this transformation as *implicit loop fusion* since it implicitly combines the multiple loops corresponding to the separate statements of a WHERE block into a single loop block with a single conditional. The implicit loop fusion results in improved cache locality since the elements corresponding to the same indices of large array are referred in subsequent statements, as illustrated by the use of $WORK1$ array in the figure. We also perform *explicit loop fusion* in which multiple subsequent WHERE blocks are combined into a single loop block.

```
1 WHERE LMASK
2   WORK1(:,:,kk) = KAPPA_THIC(:,:
    ,kbt,k,bid) * SLX(:,:,kk,kbt,k,bid) * dz(k) ;
3   WORK2(:,:,kk) = c2 * dzwr(k) * (WORK1(:,:
    ,kk) - KAPPA_THIC(:,:,ktp,k+1,bid) * SLX(:,:
    ,kk,ktp,k+1,bid) * dz(k+1)) ;
4 end
```

Fig. 4. Code with WHERE block and implicit indexing

```
1  do j=1,n
2    do i=1,n
3      if LMASK(i,j) then
4        WORK1(i,j,kk) =
         KAPPA_THIC(i,j,kbt,k,bid) *
         SLX(i,j,kk,kbt,k,bid) * dz(k) ;
5      end
6    end
7  end
8  do j=1,n
9    do i=1,n
10     if LMASK(i,j) then
11       WORK2(i,j,kk) =
         c2 * dzwr(k) * (WORK1(i,j,kk) -
         KAPPA_THIC(i,j,ktp,k+1,bid) *
         SLX(i,j,kk,ktp,k+1,bid) * dz(k+1)) ;
12     end
13   end
14 end
```

Fig. 5. Equivalent Code of Code shown in Figure 4

```
1 do j=1,n
2   do i=1,n
3     if LMASK(i,j) then
4       WORK1(i,j,kk) =
        KAPPA_THIC(i,j,kbt,k,bid) *
        SLX(i,j,kk,kbt,k,bid) * dz(k) ;
5       WORK2(i,j,kk) =
        c2 * dzwr(k) * (WORK1(i,j,kk) -
        KAPPA_THIC(i,j,ktp,k+1,bid) *
        SLX(i,j,kk,ktp,k+1,bid) * dz(k+1)) ;
6     end
7   end
8 end
```

Fig. 6. Implicit Loop Fusion for the code in Figure 4

The combined use of implicit and explicit fusion results

Version	Exec. time (msecs)
Original code	40.8
Implicit fusion (1 thread)	13.5
Explict fusion (1 thread)	10.2
Implicit fusion (16 threads)	6.06
Explicit fusion (16 threads)	2.76

TABLE I. BENEFITS OF EXPLICIT AND IMPLICIT LOOP FUSION FOR *merged_streamfunction* ON INTEL XEON

in large improvement in performance due to reduced looping overheads and conditionals, and increased cache efficiency and parallelism. For example, Table I shows the single and multi-thread performance on Xeon CPU with explicit and implicit loop fusions for a function named *merge_streamfunction*, one of the functions in $hdifft$. As can be seen, implicit fusion gives 3x performance improvement, while explicit fusion gives a further 24% improvement. We performed such loop fusions for 8 major functions within the $hidfft$ function.

Table II shows the improvements obtained in the various functions in $hdifft$ by adopting the implicit and explicit loop fusions (shown as I+E). The results correspond to a single node 16-core Intel Xeon runs.

B. Aggregation and Asynchronous Executions on Xeon Phi

One option to accelerate $hdifft$ computations in Xeon Phis is to offload each invocation of $hdifft$ in the tracer loop and parallelize the computations of the loops inside the $hdifft$ function across the Xeon Phi threads. However, the loops inside $hdifft$ are fine-grained loops with low arithmetic intensity. This approach also increases the number of offloads, and hence the corresponding offload and data movement overheads. These result in poor performance for Xeon Phi executions. For example, for a single-node execution, the execution time for a 5-day simulation run increased from 235 seconds on a 16-core Intel Xeon with 16 threads to 253 seconds on Intel Xeon Phi with 240 threads. Thus, we attempt coarse-level parallelism on the Xeon Phi to offset the offloading overheads.

In the tracer loop shown in Figure 3, the $vmix_coeffs$ function sets the global variables needed for all subsequent calls to $hdifft$ in the first iteration corresponding to the first vertical level. By dependency analysis, we also found that the results of the post $hdifft$ routines for a vertical level in the tracer loop shown in Figure 3 is not used by $hdifft$ in the subsequent iterations. Thus, all the vertical levels can be aggregated such that $hdifft$ invocations in all the iterations corresponding to the vertical levels can be potentially offloaded to Xeon Phis, and the different vertical level $hdifft$s can be performed by the different threads on the Xeon Phi cores. The data needed for all the invocations of $hdifft$ can be aggregated and a single-time transfer of the aggregated data to Xeon Phi can be performed for offloading the $hdifft$ computations. This model can exploit sufficient amount of offloading and parallelism on Xeon Phi and can also minimize CPU-Xeon Phi data transfer latencies.

However, we found that the functions invoked by $hdifft$ have pseudo-dependencies between the vertical level iterations. This is illustrated in Figure 7 which shows a segment of the code inside the $hdifft$ function. Here, variable $FZTOP_SUBM$ set in the previous iteration for the previous vertical level, $vl-1$ is used in the current iteration, vl. This hinders the planned coarse-grained parallelism across the different vertical levels. By careful analysis, we found that this variable for an iteration corresponding to vl can be calculated using other variables set in the same iteration. Specifically, the variable is purely based on a set of global variables which do not change across vertical levels. We remove these pseudo-dependencies thereby making the iterations independent.

```
1 do j = this_block%jb, this_block%je
2   do i = this_block%ib, this_block%ie
3     WORK1(i,j) = ... ;
4     fz = ... * (WORK1(i,j) + WORK2(i,j)) ;
5     GTK(i,j,n) = ...+
      FZTOP_SUBM(i,j,n,bid) - fz) *... ;
6     FZTOP_SUBM(i,j,n,bid) = fz ;
7   end
8 end
```

Fig. 7. Illustration of Pseudo-dependency

While the $hdifft$ invocations for different independent vertical levels can be offloaded and parallelized on Xeon Phi, we found that the performance improvement obtained on about 200 light-weight Xeon Phi cores is negligible when compared to the performance obtained on 16 cores of Xeon. The arithmetic intensity of the code is too low to obtain performance benefits in a synchronous offload model, in which the CPU waits for the offloaded CPU computations. Hence, we explore asynchronous offloading of $hdifft$ in which the CPU proceeds with its computations after offloading $hdifft$ to Xeon Phis, and both the resources execute different computations simultaneously.

We propose a novel *look-ahead asynchronous offloading strategy* in which we offload the computations for the next timestep in the current timestep to Xeon Phi, and make the CPUs proceed with its computations in the current timestep, and pick up the results from Xeon Phi when it reaches the next time step. There exists two types of tracers in the ocean code, mixtime tracer ($tmix$) and current time tracer ($tcur$), as mentioned in Section II. One of the primary parameters passed to $hdifft$ is the mixtime tracer, $tmix$. We noticed that at the end of every time step, the mixtime tracer and the current time tracer values are being swapped. Therefore, the value of $tmix$ for a timestep is the same as the value of the current time tracer, $tcur$, in the previous time step. Thus, replacing $tmix$ with $tcur$ in the call to $hdifft$ at a timestep t will result in the output produced by $hdifft$ corresponding to timestep $t+1$. Thus, the CPU can offload $hdifft$ invocations with $tcur$ at timestep t for asynchronous executions on Xeon Phi and proceed with the rest of its calculations. When the CPU reaches the next time step, $t+1$, the Xeon Phis would have completed the $hdifft$ output needed for $t+1$, which the CPU can pick up for its calculations. Thus, in this asynchronous mode of execution, the $hdifft$ calculations are completely masked in the Xeon Phi cores, and the time taken for these calculations is completely subtracted from the total time, thereby resulting in significant benefits.

For the first time step, the $hdifft$ calculations are performed on the CPU with $tmix$ variable, and the result is

Function	1-thread Exec. time of Original code (msecs)	1-thread Exec. time of Code with I+E (msecs)	16-thread exec. time of code with I+E (msecs)
merged_stream_function part 1	40.7	10.3	2.5
merged_stream_function part 2	33.0	11.6	4.53
transition layer	30.1	12.8	6.3
apply_vertical_profile_to_isop_hor_diff	13.2	5.8	0.77
buoyancy_frequency_dependent_profile	30.0	28.2	Limited where statements
vertical averages of horizontal buoyancy differences within the mixed layer	6.7	1.7	0.81
compute horizontal length scale	5.3	1.7	0.50
computing streamfunction due to submesoscale parameterization	14.0	3.9	1.1

TABLE II. BENEFITS OF IMPLICIT AND EXPLICIT LOOP FUSIONS (I+E) FOR DIFFERENT FUNCTIONS

used in the same timestep. The entire pseudocode related to asynchronous executions of $hdifft$ in Xeon Phi is shown in Figure 8.

```
1  foreach time step do
       /* Tracer update loop                          */
2      do iblock=1,nblocks_clinic
3          do k=1,nverticals
4              vmix_coeffs (...) ;
5              if first timestep then
6                  hdifft (...,tmix,...,WORK,...) ;
7              else
8                  Wait for Xeon Phi to return from hdifft
                   offloaded in the prev time step ;
9                  WORK = WORK_PHI; /* Copy
                       output of the offloaded
                       hdifft from Xeon Phi           */
10             end
11             begin offload
12                 do kk=1,nverticals
13                     hdifft (...,tcur,...,WORK_PHI,...) ;
14                 end
15             end
               /* Use WORK data                       */
16             advt (...) ;
17             vdifft (...) ;
18         end
19     end
20 end
```

Fig. 8. Pseudocode for Asynchronous Executions of Ocean Model

C. Minimizing Data Transfers and Converting to Allocatables

We also observed that many arrays that are used within $hdifft$ are initialized to zero at the beginning of $hdifft$ and are not used outside the scope of $hdifft$. This implies that these arrays are local to the $hdifft$ computations and hence need not be copied from CPU to the coprocessor on every offload. Many of these arrays (at least six) are large 5-6 dimensional arrays. Thus, use of the $nocopy$ option to the offload for these arrays can result in significant performance benefits. In our experiments, this resulted in at least 18% reduction in data copying time.

We also found that copy of the allocatables are about 10% faster than copy of the static variables from CPU host to the coprocessor. Hence, we promoted many of the static arrays to heap global arrays, and used $allocatables$ option for these arrays. In our experiments, we found that these optimizations results in reduction of data transferred from 470 MBytes to only 55 MBytes for a single timestep. In addition, we perform one-time allocation and one-time copy of global data that needs to reside on the Xeon Phi during only the first time step.

V. EXPERIMENTS AND RESULTS

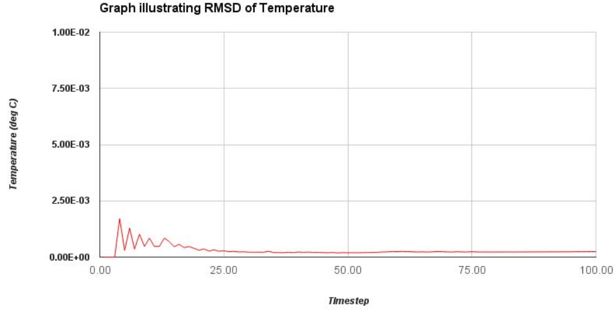
In our experiments, we used the POP ocean model with nominal resolution of approximately 1° with a horizontal grid of 320 x 384 grid points with approximately 100 km spacing between consecutive points, and the number of vertical levels as 60. The experiments were conducted on a cluster containing 8 nodes 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. We show results using 8 or 16 MPI tasks on the 8 nodes. We use the Xeon Phi cards depending on the experiment. In each Xeon Phi, 60 threads were executed. The experiments were conducted for 100 simulation timesteps corresponding to 5-day simulation runs. The timings of the model were obtained from the logs generated by default CESM executions, using GPTL library. Timings over specific regions of the code were using OpenMP's omp_get_wtime.

A. Results on Correctness

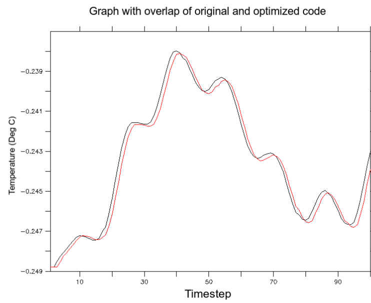
We first demonstrate the correctness of our code modifications due to various optimizations, including the asynchronous executions on Xeon Phi in which some of the parameters for $hdifft$ calculations for the next time step are passed in the current time step.

We verified the accuracy of the results by finding the root mean square of the differences (RMSD) of the temperature values of the ocean grid points produced in the original code and our optimized code. The result was obtained for a 5-day simulation run using 8 MPI tasks on 8 nodes with 4 OpenMP threads on each node for a total of 32 threads, and one Intel Phi card in each node for a total of 8 Intel Xeon Phi coprocessors. The original code uses only the CPU cores while our optimized version offloads $hdifft$ computations to Xeon Phi.

Figure 9(a) shows the RMSD values for the different timesteps. Figure 9(b) shows the variation of the actual temperature values in °C, for a particular grid point corresponding to approximately 45°N and 135°W, for various time steps in the original and optimized codes. The graphs show that the errors due to our optimizations are reasonably small.



(a) RMSD of Temperature Values of the Ocean Grid



(b) Actual Temperature Values for a Grid Point

Fig. 9. Error Growth due to our Optimizations and Offloading to Xeon Phi

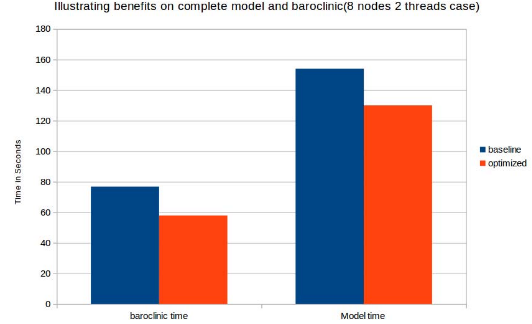


Fig. 10. Overall Performance Improvement Using One Coprocessor per Node

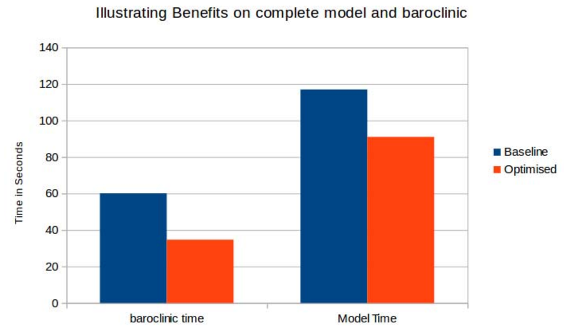


Fig. 11. Overall Performance Improvement Using Two Coprocessors per Node

B. Overall Performance Improvements

In all our performance-related experiments, the comparisons of our optimized code using Intel Xeon Phi cards are made with the parallel execution of the original code using only the CPU cores. In each experiment, our optimized code using (X CPU cores + Y Intel Xeon Phi cards) across N nodes is compared with the original parallel code running on the same X CPU cores across the N nodes.

1) One Coprocessor Per Node Runs: We first show the overall performance improvement using all our optimizations over a baseline run utilizing only the CPU cores. In our first experiment, we execute 8 MPI tasks on 8 nodes with 2 OpenMP threads on each node for a total of 16 threads, and one Intel Phi card in each node for a total of 8 Intel Xeon Phi coprocessors. Figure 10 shows the improvements in both the baroclinic phase containing the $hdifft$ routine that we optimize in our work and the entire POP ocean model.

We find that the use of Intel Xeon Phi with our optimizations described in this work yield about 25% reduction in execution time over the CPU-only run for the baroclinic phase and about 17.4% reduction in execution time for the entire POP model.

2) Two Coprocessors Per Node Runs: Figure 11 shows the executions corresponding to executing 16 MPI tasks on 8 nodes with 2 MPI tasks each. For offloading to Xeon Phis, each of the MPI tasks on a node offloads its computations to a separate Intel Xeon Phi card, thus utilizing both the Xeon Phi cards on the node. The comparisons are with the CPU-only executions. We find that the optimized code results in about 42% performance improvement in the baroclinic phase and about 23% performance improvement in the overall POP model. The use of two Xeon Phis per node thus results in more improvements showing that our strategies scale with more coprocessors.

C. Individual Optimizations

Figure 12 shows the benefits due to our individual optimizations, namely, implicit and explicit fusion, asynchronous executions, and data management techniques. All the results correspond to the execution of the entire POP model with 8 MPI tasks on 8 nodes with 4 OpenMP threads on each node for a total of 32 threads. The first two bars correspond to the benefits with our implicit and explicit fusion only on the Intel Xeon CPUs. We find that our fusion techniques result in 7.7% reduction in time for the CPU execution. The remaining correspond to offloading results using one Intel Phi card in each node for a total of 8 Intel Xeon Phi coprocessors. The middle two bars compare our asynchronous offloading and simultaneous execution strategy on Intel Xeon Phi with a synchronous execution strategy in which the CPU waits for the Xeon Phi computations and use the results in the same time step. We find that we obtain about 36% reduction in execution time due to our asynchronous execution strategy. The last two bars show the benefits due to our data management techniques, including the use of *nocopy* variables and *allocatables*, that minimize the CPU-coprocessor data transfers for offloading. The data management techniques result in 18.16% reduction in execution time.

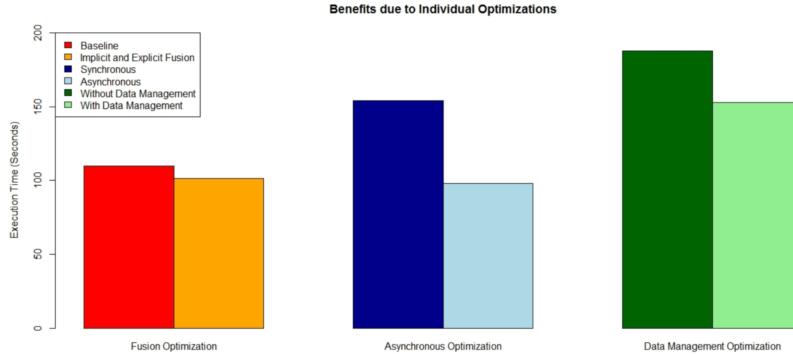


Fig. 12. Benefits due to Implicit and Explict Fusions

Figure 13 compares two modes of offloading the $hdifft$ computations - our coarse-grained model in which we aggregate all the data for all $hdifft$ invocations for the vertical levels, and offload and parallelize all the $hdifft$ iterations and a fine-grained model in which we offload every $hdifft$ invocation and parallelize all the loops contained in $hdifft$ on Xeon Phi. The result was obtained on 8 nodes with 8 MPI tasks. Our results show that we obtain about 26% improvement in performance with coarse-grained offloading model over fine-grained mode. The large improvements with the coarse-grained model is due to the reduced number of offloads and offloading overheads, reduced data transfer latencies and large arithmetic intensity in each Xeon Phi thread.

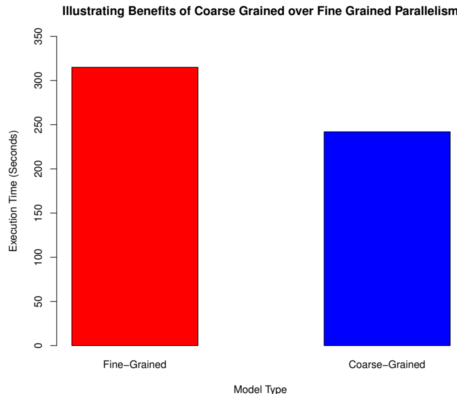


Fig. 13. Comparison of Coarse-Grained and Fine-Grained Offloading Model

VI. Discussion

We can derive a number of valuable lessons from our optimization steps with horizontal diffusion in tracers.

1) Options will have to be carefully weighed when deciding on the portions to offload computations to Xeon Phi. Particularly, large-scale scientific codes will have large loops calling computationally intensive functions, which in turn will have embedded loops. The choice of offloading each function call and parallelizing the inner loops (fine-grained offloading) versus offloading the entire outer-level loop depends on the arithmetic intensity of the inner loops, and the amount of data needed by the function. If the inner loops have large arithmetic-intensity, fine-grained offloading can be adopted, while coarse-grained offloading is beneficial if there is more parallelism at the outer-loop level. This was illustrated in Figure 13 where the coarse-grained parallelism yielded 24% improvement in performance over the fine-grained parallelism.
2) For performing coarse-grained offloading of the outer-loop that invokes the function, the data needed by the function will have to be aggregated efficiently to minimize data-latency overheads. Data dependencies into and out of the function will have to be analyzed. In our work, analyses of the data dependencies and minimizing data transfers resulted in 9% reduction in execution time.
3) If the resulting coarse-grained offloading does not improve performance significantly over the CPU executions, asynchronous execution strategy can be beneficial in which independent computations can be performed simultaneously on both the CPU and Intel Xeon Phi cores. This depends on the program characteristics of the particular application. Look-ahead asynchronous model, similar to our work, can be performed if the code permits. As illustrated in Figure 12, the asynchronous execution strategy resulted in 33% performance improvement over synchronous executions in our work.
4) Large legacy scientific codes can have pseudo dependencies that hinder parallelism. These will have to be carefully removed. Our work also performed this optimization and illustrated in Figure 13.
5) The use of WHERE statements in FORTRAN can hinder parallelism, cache locality and vectorization. Performance benefits will have to be analyzed for potentially converting these blocks to loops with explicit indexing. In our work, conversion to such explicit indexing resulted in significant benefits as shown in Tables I and II, and Figure 12.
6) Large scientific applications also contain many high-

dimensional data arrays needed for the offloaded portions. Not all of them may have to be transferred on every offload. Our strategies of copying only the needed data by analyzing the local data, and use of allocatables for static variables can significantly reduce data transfer times. As shown in Figure 12, our strategies resulted in 21% reduction in data copying time.

VII. Conclusions and Future Work

In this work, we have successfully offloaded the time-consuming horizontal diffusion in tracers in the ocean model on Intel Xeon Phi architecture. We adopted various strategies including replacing WHERE blocks for implicit and explicit fusions for increasing parallelism and cache efficiency, removing pseudo-dependencies, data and function aggregation, use of coarse-grained over fine-grained parallelism, a novel look-ahead asynchronous execution model, and efficient data management techniques including the use of *nocopy* variables and *allocatables* for offloading. With our simultaneous execution model in which the results of the horizontal diffusion routines for a time step are used in the subsequent time steps, we showed error growths and demonstrated reasonable accuracy. The use of Intel Xeon Phi with our optimizations described in this work yielded about 25-42% reduction in execution time over the CPU-only runs for the baroclinic phase and about 17.4-23% reduction in execution time for the entire POP model. Our optimization strategies exhibited good scaling with the use of more Intel Xeon Phi processors. Among our optimization techniques, the use of our novel look-ahead asynchronous execution strategy on Intel Xeon Phis resulted in about 36% performance improvement over the synchronous strategy and the use of our data management for offloading resulted in about 18% performance improvement due to reduction in CPU-coprocessor data transfer times. In future, we plan to adopt similar strategies for other 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 Knight Landing.

Acknowledgments

This project is supported by the Intel® Parallel Computing Centre for Modelling Monsoons and Tropical Climate (IPCC-MMTC), India sponsored by the Intel® Corporation. We would also like to thank Om Sachan and Vinutha V from Intel Software Services Group for their immense help with training and support.

References

[1] P. Worley, A. Mirin, A. Craig, M. Taylor, J. Dennis, and M. Vertenstein, "Performance of the Community Earth System Model," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011.

[2] J. Dukowicz and R. Smith, "Implicit Free-surface Method for the Bryan-Cox-Semtner Ocean Model," *Journal of Geophysical Research*, vol. 99, pp. 7991–8014, 1994.

[3] R. Smith and e. a. P. Jones, "The Parallel Ocean Program (POP) reference manual: Ocean component of the Community Climate System Model (CCSM)," Los Alamos National Laboratory, Los Alamos, USA, Tech. Rep. LAUR-10-01853, 2010.

[4] "The Gent-McWilliams Parametrization: 20/20 Hindsight," *Ocean Modeling*, vol. 39, pp. 2–9, 2011.

[5] Y. Liu, T. Tran, F. Lauenroth, and B. Schmidt, "SWAPHI-LS: Smith-Waterman Algorithm on Xeon Phi coprocessors for Long DNA Sequences," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2014.

[6] S. Heybrock, B. Joo, D. Kalamkar, M. Smelyanskiy, K. Vaidyanathan, T. Wettig, and 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.

[7] R. Luo, J. Cheung, E. Wu, H. Wang, and S.-H. C. et al., "MICA: A Fast Short-read Aligner that takes full advantage of Many Integrated Core Architecture (MIC)," *BMC Bioinformatics*, vol. 16, no. 7, pp. 1–8, 2015.

[8] J. Michalakes and M. Vachharajani, "GPU Acceleration of Numerical Weather Prediction," in *IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, 2008.

[9] M. Govett, J. Middlecoff, and T. Henderson, "Running the NIM Next-generation Weather Model on gpus," in *IEEE/ACM Internation Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010.

[10] I. Carpenter, R. Archibald, K. Evans, J. Larkin, P. Micikevicius, M. Norman, J. Rosinski, J. Schwarzmeier, and M. Taylor, "Progress Towards Accelerating HOMME on Hybrid Multi-core Systems," *International Journal of High Performance Computing Applications (IJH-PCA)*, vol. 27, no. 3, pp. 335–347, 2013.

[11] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, and 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*, ser. SC '10, 2010, pp. 1–11.

[12] Y. Hu, X. Huang, A. Baker, Y. Tseng, F. Bryan, J. Dennis, and G. Yang, "Improving the Scalability of the Ocean Barotropic Solver in the Community Earth System Model," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2015.

[13] F. Bleichrodt, R. Bisseling, and H. Dijkstra, "Accelerating a Barotropic Ocean Model using GPU," *Ocean Modeling*, vol. 14, pp. 16–21, 2012.

[14] B. van Werkhoven, J. Maassen, M. Kliphuis, H. Dijkstra, S. Brunnabend, M. van Meersbergen, F. Seinstra, and H. Bal, "A Distributed Computing Approach to Improve the Performance of the Parallel Ocean Program (v2.1)," *Geoscientific Model Development*, vol. 7, no. 1, pp. 267–281, 2014.

[15] S. Xu, X. Huang, L.-Y. Oey, F. Xu, H. Fu, Y. Zhang, and G. Yang, "POM.gpu-v1.0: a GPU-based Princeton Ocean Model," *Geoscientific Model Development*, vol. 8, no. 9, pp. 2815–2827, 2015.

[16] H. Zhang and 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.

[17] O. Fuhrer, C. Osuna, X. Lapillonne, T. Gysi, B. Cumming, M. Bianco, A. Arteaga, and T. Schulthess, "Towards a Performance Portable, Architecture Agnostic Implementation Strategy for Weather and Climate Models," *Supercomputing Frontiers and Innovations*, vol. 1, no. 1, 2014.

[18] J. Mielikainen, B. Huang, and A. Huang, "Revisiting Intel Xeon Phi optimization of Thompson cloud microphysics scheme in Weather Research and Forecasting (WRF) model," *Proc. SPIE 9646, High-Performance Computing in Remote Sensing V"*, 2015.

[19] ——, "Performance tuning Weather Research and Forecasting (WRF) Goddard longwave radiative transfer scheme on Intel Xeon Phi," 2015.

[20] ——, "Optimizing the updated Goddard shortwave radiation Weather Research and Forecasting (WRF) scheme for Intel Many Integrated Core (MIC) architecture," 2015.

[21] V. Betro, R. Harkness, B. Hadri, H. You, R. Hulguin, R. Brook, and 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.

[22] J. Michalakes, M. Iacono, and D. Berthiaume, "Optimizing Weather Model Radiative Transfer Physics for the Many Integrated Core and GPGPU Architectures," in *Heterogeneous Multi-Core Workshop*, 2014.