Geosci. Model Dev., 16, 1445–1458, 2023
https://doi.org/10.5194/gmd-16-1445-2023

Geoscientific Model Development
Open Access

Porting the WAVEWATCH III (v6.07) wave action source terms to GPU

Olawale James Ikuyajolu[1,2], Luke Van Roekel[3], Steven R. Brus[4], Erin E. Thomas[3], Yi Deng[1,2], and Sarat Sreepathi[5]

[1]Earth and Atmospheric Sciences, Georgia Institute of Technology, Atlanta, GA, USA
[2]Program in Ocean Science and Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[3]Fluid Dynamics and Solid Mechanics (T-3), Los Alamos National Laboratory, Los Alamos, NM, USA
[4]Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA
[5]Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA

Correspondence: Olawale James Ikuyajolu (oikuyajolu3@gatech.edu)

Received: 2 June 2022 – Discussion started: 10 June 2022
Revised: 2 December 2022 – Accepted: 24 January 2023 – Published: 3 March 2023

Abstract. Surface gravity waves play a critical role in several processes, including mixing, coastal inundation, and surface fluxes. Despite the growing literature on the importance of ocean surface waves, wind–wave processes have traditionally been excluded from Earth system models (ESMs) due to the high computational costs of running spectral wave models. The development of the Next Generation Ocean Model for the DOE's (Department of Energy) E3SM (Energy Exascale Earth System Model) Project partly focuses on the inclusion of a wave model, WAVEWATCH III (WW3), into E3SM. WW3, which was originally developed for operational wave forecasting, needs to be computationally less expensive before it can be integrated into ESMs. To accomplish this, we take advantage of heterogeneous architectures at DOE leadership computing facilities and the increasing computing power of general-purpose graphics processing units (GPUs). This paper identifies the wave action source terms, `W3SRCEMD`, as the most computationally intensive module in WW3 and then accelerates them via GPU. Our experiments on two computing platforms, Kodiak (P100 GPU and Intel(R) Xeon(R) central processing unit, CPU, E5-2695 v4) and Summit (V100 GPU and IBM POWER9 CPU) show respective average speedups of 2× and 4× when mapping one Message Passing Interface (MPI) per GPU. An average speedup of 1.4× was achieved using all 42 CPU cores and 6 GPUs on a Summit node (with 7 MPI ranks per GPU). However, the GPU speedup over the 42 CPU cores remains relatively unchanged ($\sim 1.3\times$) even when using 4 MPI ranks per GPU (24 ranks in total) and 3 MPI ranks per GPU (18 ranks in total). This corresponds to a 35 %–40 % decrease in both simulation time and usage of resources. Due to too many local scalars and arrays in the `W3SRCEMD` subroutine and the huge WW3 memory requirement, GPU performance is currently limited by the data transfer bandwidth between the CPU and the GPU. Ideally, OpenACC routine directives could be used to further improve performance. However, `W3SRCEMD` would require significant code refactoring to make this possible. We also discuss how the trade-off between the occupancy, register, and latency affects the GPU performance of WW3.

1 Introduction

Ocean surface gravity waves, which derive energy and momentum from steady winds blowing over the surface of the ocean, are a very crucial aspect of the physical processes at the atmosphere–ocean interface. They influence a variety of physical processes such as momentum and energy fluxes, gas fluxes, upper-ocean mixing, sea spray production, ice fracture in the marginal ice zone, and Earth albedo (Cavaleri et al., 2012). Such complex wave processes can only be treated accurately by including a wave model into Earth system models (ESMs). The first ocean models neglected the existence of ocean waves by assuming that the ocean surface is rigid to momentum and buoyancy fluxes from the atmospheric boundary layer (Bryan and Cox, 1967). Currently, most state-of-the-science ESMs are still missing some or all of these wave-induced effects (Qiao et al., 2013), despite the growing literature on their importance in the simulation of weather and climate.

Recent literature has shown that incorporating different aspects of surface waves into ESMs leads to improved skill performance, particularly in the simulation of sea surface temperature, wind speed at 10 m height, ocean heat content, mixed-layer depth, and the Walker and Hadley circulations (Law Chune and Aouf, 2018; Song et al., 2012; Shimura et al., 2017; Qiao et al., 2013; Fan and Griffies, 2014; Li et al., 2016). Yet, only two climate models that participated in the Climate Model Intercomparison Project Phase 6 (CMIP6), i.e., the First Institute of Oceanography Earth System Model Version 2 (FIO-ESM v2.0; Bao et al., 2020) and the Community Earth System Model Version 2 (CESM2; Danabasoglu et al., 2020), have a wave model as part of their default model components. However, for CMIP6, only FIO-ESM v2.0 employed a wave model. Wind–wave-induced physical processes have traditionally been excluded from ESMs due to the high computational cost of running spectral wave models on global model grids for long-term climate integrations. In addition to higher computing costs due to longer simulation times, adding new model components also increases resource requirements (e.g., the number of central processing units/nodes). The next-generation ocean model development in the US Department of Energy (DOE) Energy Exascale Earth System Model (E3SM) project partly focuses on the inclusion of a spectral wave model, WAVEWATCH III (WW3), into E3SM to improve the simulation of coastal processes within E3SM. To make WW3 within E3SM feasible for long-term global integrations, we need to make it computationally less expensive.

Computer architectures are evolving rapidly, especially in the high-performance computing environment, from traditional homogeneous machines with multicore central processing units (CPUs) to heterogeneous machines with multinode accelerators such as graphics processing units (GPUs) and multicore CPUs. Moreover, the number of CPU–GPU heterogeneous machines in the top 10 of the TOP500 increased from two in November 2015 (https://www.top500.org/lists/top500/2015/11/, last access: 30 November 2022) to seven in November 2022 (https://www.top500.org/lists/top500/2022/11/, last access: 30 November 2022). The advent of heterogeneous super-computing platforms as well as the increasing computing power and low energy-to-performance ratio of GPUs has motivated the use of GPUs to accelerate climate and weather models. In recent years, numerous studies have reported successful GPU porting of full or partial weather and climate models with improved performance (Hanappe et al., 2011; Xu et al., 2015; Yuan et al., 2020; Zhang et al., 2020; Bieringer et al., 2021; Mielikainen et al., 2011; Michalakes and Vachharajani, 2008; Shimokawabe et al., 2010; Govett et al., 2017; Li and Van Roekel, 2021; Xiao et al., 2013; Norman et al., 2017; Norman et al., 2022; Bertagna et al., 2020). A GPU programming model is different from CPU code, so programmers must recode or use directives to port codes to GPU. Because climate and weather models consist of million of lines of code, a majority of the GPU-based climate simulations only operate on certain hot spots (most computationally intensive operations) of the model while leaving a large portion of the model on CPUs. In recent years, however, efforts have been made to run an entire model component on a GPU. For example, Xu et al. (2015) ported the entire Princeton Ocean Model to GPU and achieved a 6.9× speedup. Similarly, the entire E3SM atmosphere including SCREAM (Simple Cloud-Resolving E3SM Atmosphere Model) is running on a GPU (https://github.com/E3SM-Project/scream, last access: 30 November 2022). Taking advantage of the recent advancements in GPU programming in climate sciences and the heterogeneous architectures at DOE leadership computing facilities, this study seeks to identify and move the computationally intensive parts of WW3 to GPU through the use of OpenACC pragmas.

The rest of the paper is structured as follows: Sect. 2 presents an overview of the WW3 model and its parallelization techniques, give an introduction to the OpenACC programming model, describe the hardware and software environment of our testing platforms, and finally present the test case configuration used in this study; the results section, Sect. 3, presents the WW3 profiling analysis on CPU, discusses the challenges encountered, outlines GPU-specific optimization techniques, and compares the GPU results with the original Fortran code; and Sect. 4 concludes the paper.

2 Model description and porting methodology

2.1 WAVEWATCH III

WW3 is a third-generation spectral wave model developed at the US National Centers for Environmental Prediction (NOAA/NCEP) (WAVEWATCH III® Development Group, 2019) from the WAve Model (WAM) (The Wamdi Group, 1988). It has been widely used to simulate ocean waves in many oceanic regions for various science and engineering applications (Chawla et al., 2013b; Alves et al., 2014; Cornett, 2008; Wang and Oey, 2008). To propagate waves, WW3 solves the random-phase spectral action density balance equation, $N(\phi, \lambda, \sigma, \theta, t)$, for wave number direction spectra. The intrinsic frequency (σ) relates the action density spectrum to the energy spectrum (F), $N = \frac{F}{\sigma}$. For large-scale applications, the evolution of the wave action density in WW3 is expressed in spherical coordinates as follows:

$$\frac{\partial N}{\partial t} + \frac{\partial (C_\phi N)}{\partial \phi} + \frac{\partial (C_\lambda N)}{\partial \lambda} + \frac{\partial (C_\sigma N)}{\partial \sigma} + \frac{\partial (C_\theta N)}{\partial \theta} = \sum_i S_i . \quad (1)$$

Equation (1) is solved by discretizing in both physical space (λ, ϕ) and spectral space (σ, θ). Here, ϕ is the longitude, λ is the latitude, σ is the relative frequency, θ is the direction, t is time, and S represents the source and sinks terms. The net source–sink terms consist of several physical processes responsible for the generation, dissipation, and redistribution of energy. The net source–sink terms available in WW3 are wave generation due to wind (S_{in}), dissipation (S_{ds}), nonlinear quadruplet interactions (S_{nl}), bottom friction (S_{bt}), and depth-limited breaking (S_{db}); triad wave–wave interactions (S_{tr}); scattering of waves by bottom features (S_{sc}); wave–ice interactions (S_{ice}); and reflection off shorelines or floating objects (S_{ref}). Details of each source term can be found in the WW3 manual (WAVEWATCH III® Development Group, 2019). The primary source–sink terms used in this work are S_{in}, S_{ds}, S_{nl}, S_{bt}, and S_{db}. Several modules are used for the calculation of source terms. However, the `W3SRCEMD` module manages the general calculation and integration of source terms.

When moving code between different architectures, it is necessary to understand its structure. For the purposes of our study, Fig. 1 shows a representation of the WW3 algorithm structure. WW3 is divided into several submodules, but the actual wave model is `W3WAVEMD`, which runs the wave model for a given time interval. Within `W3WAVEMD`, several modules are called at each time interval to handle initializations, interpolation of winds and currents, spatial propagation, intra-spectral propagation, calculation and integration of source terms, output file processing, etc. In our work, we found that `W3SRCEMD` is the most computationally intensive part of WW3; thus, we focus our attention on this module. According to Fig. 1a, `W3SRCEMD` is being called at each spatial grid point, which implies that the spatial grid loop is not contained in `W3SRCEMD` but rather in `W3WAVEMD`. Figure 1b, which represents `W3SRCEMD`, contains a dynamic integration time loop that can only be executed sequentially. It also calls a number of submodules, such as `W3SRC4MD` for the computation of the wind input and wave breaking dissipation source terms. `W3SRCEMD` consists of collapsed spectral loops (NSPECH = NK × NTH), where NK is the number of frequencies (σ) and NTH is the number of wave directions (θ). Lastly, `W3SRC4MD` (Fig. 1c) consists of only frequency (NK) loops. The structure of other source terms submodules is similar to `W3SRC4MD`.

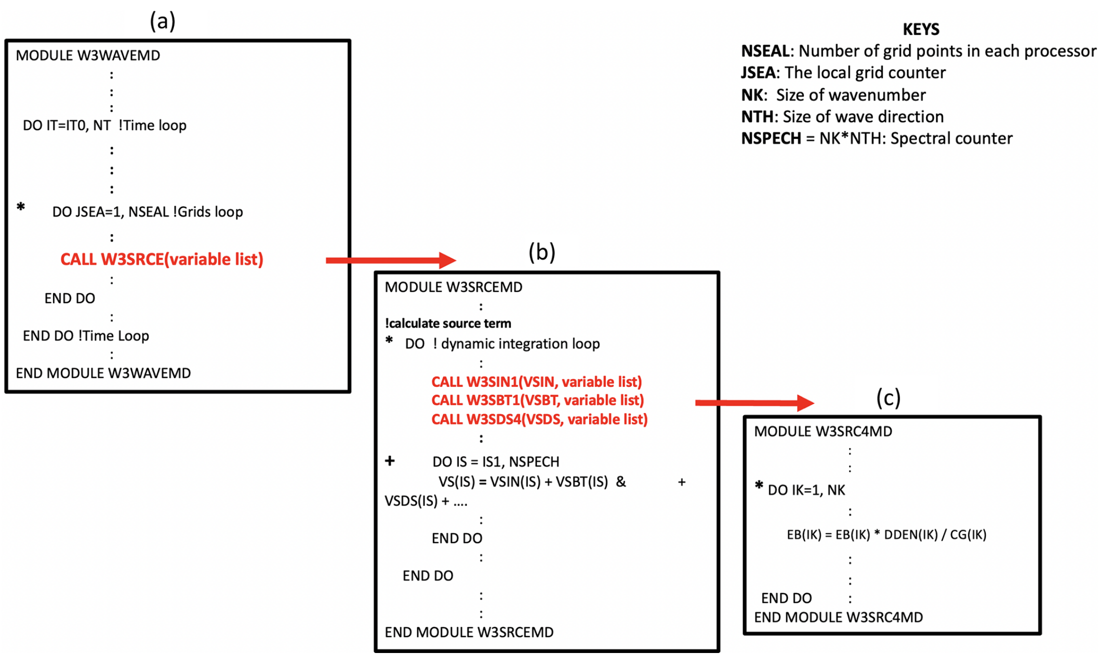
2.2 WW3 grids and parallel concepts

The current version of WW3 can be run and compiled for both single- and multiprocessor Message Passing Interface (MPI) compute environments with regular grids, two-way nested (mosaic) grids (Tolman, 2008), spherical multicell (SMC) grids (Li, 2012), and unstructured triangular meshes (Roland, 2008; Brus et al., 2021). In this study, we ran and compiled WW3 using an MPI with unstructured triangular meshes as the grid configuration. In WW3, the unstructured grid can be parallelized in physical space using either card deck (CD) (Tolman, 2002) or domain (Abdolali et al., 2020) decomposition. Following Brus et al. (2021), we used the CD approach as the parallelization strategy. The ocean (active) grid cells are sorted and distributed linearly between processors in a round-robin fashion using $n = \mathrm{mod}(m-1, N)$, i.e., grid cell m is assigned to processor n. Here, N is the total number of processors. If N is divisible by M (the total number of ocean grids), every processor n has the same number of grids, NSEAL (Fig. 1a). The source term calculation as well as the intra-spectral propagation are computed using the aforementioned parallel strategy, but data are gathered on a single processor to perform the spatial propagation.

2.3 OpenACC

To demonstrate the promise of GPU computing for WW3, we used the OpenACC programming model. OpenACC is a directive-based parallel programming model developed to run codes on accelerators without significant programming effort. Programmers incorporate compiler directives in the form of comments into Fortran, C, or C++ source codes to assign the computationally intensive sections of the code to be executed on the accelerator. OpenACC helps to simplify GPU programming because the programmer is not preoccupied with the code parallelism details, unlike CUDA and OpenCL where you need to change the code structure to achieve GPU compatibility. The OpenACC compiler automatically transfers calculations and data between two different architectures: the host (CPU) and the accelerator device (GPU). OpenACC works together with OpenMP, MPI, and CUDA, supporting heterogeneous parallel environments. Starting from version 4.5, the OpenMP API (application programming interface) specification has been extended to include GPU offloading and GPU parallel directives. Whereas OpenMP and OpenACC have similar constructs, OpenMP is more prescriptive. Prescriptive directives describe the exact computation that should be performed and provide the compiler no flexibility. Our study focuses solely on OpenACC because it has the most mature implementation using the NVIDIA compiler on NVIDIA GPUs at the time of analysis.

OpenACC has three levels of parallelism (Fig. 2), namely vectors, workers, and gangs, corresponding to threadIdx.x, threadIdx.y, and blockIdx.x in CUDA terminology. A gang is a group of workers, where multiple gangs work independently without synchronization. Workers are groups of vectors/threads within a gang, and a vector is the finest level of parallelism operating with single-instruction, multiple thread (SIMT). Gangs, workers, and vectors can be added to a loop region that needs to be executed on a GPU. An example of Fortran code with and without OpenACC directives is shown in Fig. 3. The OpenACC directives are shown in green as comments starting with !$`acc` (e.g., lines 6 and 12 in Fig. 3b). In Fig. 3b, line 6 is a declaration directive for allocating memory for variables on GPU, line 8 is the data region to move data into the GPU, line 11 updates data already present on the GPU with new values from the CPU, line 12



```
MODULE W3WAVEMD
        :
        :
        :
 DO IT=IT0, NT  !Time loop
        :
        :
        :
*    DO JSEA=1, NSEAL !Grids loop
        :
       CALL W3SRCE(variable list)
        :
    END DO
        :
 END DO !Time Loop
        :
END MODULE W3WAVEMD
```

```
MODULE W3SRCEMD
        :
!calculate source term
*  DO  ! dynamic integration loop
        :
       CALL W3SIN1(VSIN, variable list)
       CALL W3SBT1(VSBT, variable list)
       CALL W3SDS4(VDS, variable list)
        :
+      DO IS = IS1, NSPECH
        VS(IS) = VSIN(IS) + VSBT(IS)  &          +
VSDS(IS) + ....
        :
       END DO
        :
   END DO
        :
        :
END MODULE W3SRCEMD
```

```
MODULE W3SRC4MD
        :
        :
* DO IK=1, NK
        :
     EB(IK) = EB(IK) * DDEN(IK) / CG(IK)
        :
        :
  END DO    :
END MODULE W3SRC4MD
```

NSEAL: Number of grid points in each processor
JSEA: The local grid counter
NK: Size of wavenumber
NTH: Size of wave direction
NSPECH = NK*NTH: Spectral counter

Figure 1. A schematic representation of the WAVEWATCH III code structure.

Table 1. GPU hardware specifications.

	Summit (V100)	Kodiak (P100)
Compute capability	7	6
Global memory size	16 GB	16 GB
L1 cache	10 MB	1.3 MB
L2 cache	6 MB	4 MB
Shared memory size per SM	Configurable up to 96 kB	49 kB
Constant memory	64 kB	64 kB
Register file size	256 kB (per SM)	256 kB (per SM)
32-bit registers	65 536 (per SM)	65 536 (per SM)
Max registers per thread	255	255
Number of multiprocessors (SMs)	80	56
Warp size	32 threads	32 threads
Maximum resident warps per SM	64	64
Maximum resident blocks per SM	32	32
Maximum resident threads per SM	2048	2048
Maximum threads per block	1024	1024

SM refers to streaming multiprocessor, and L1 and L2 denote Level 1 and Level 2, respectively.

launches the parallel region at the gang level and then dispatches the parallel threads at the worker and vector levels, line 21 updates CPU data with new values from the GPU, and line 28 deletes the data on the GPU after computation. To learn more about all OpenACC directives, the reader is referred to NVIDIA (2017) or Chandrasekaran and Juckeland (2017)

2.4 Test case configuration

In this study, the WW3 model was configured and simulated over the global ocean with an unstructured mesh with a 1° global resolution and a 0.25° resolution in regions with a depth of less than 4 km, e.g., 1° at the Equator and 0.25° in coastal regions (Brus et al., 2021). The number of the unstructured mesh nodes is 59 072 (ranges from 1° at the Equator to 0.5° along coastlines; hereafter 59K). In addition, we demonstrate the effect of scaling the problem size on speedup by using an unstructured mesh with 228 540 nodes (ranges

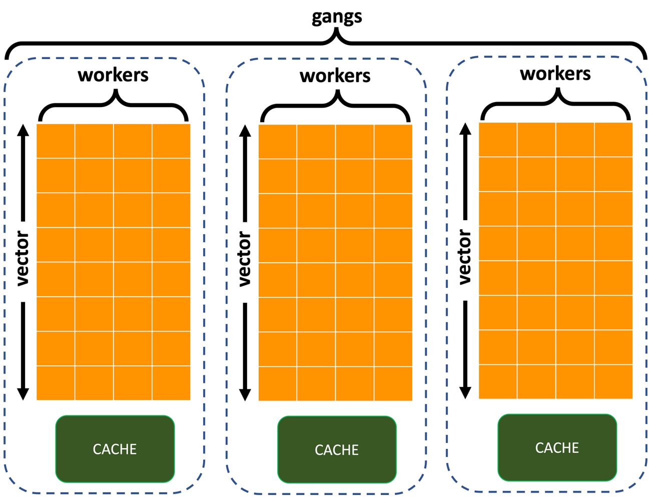


Figure 2. A map of gangs, workers, and vectors (adapted from Jiang et al., 2019).

from 0.5° at the Equator to 0.25° along coastlines; hereafter 228K). Following Chawla et al. (2013a), for both spatial grid configurations, we use a spectral grid that has 36 directions and 50 frequency bands that range exponentially from 0.04 to 0.5 Hz, separated by a factor of 1.1. In WW3, the combinations and types of the source terms in Eq. (1) depend on the research question being answered. WW3 has several source term packages which can be implemented by activating different switches. However, as the goal of this study is purely computational, we selected the following commonly used source–sink term switches: ST4, DB1, BT1, and NL1. The ST4 switch (Ardhuin et al., 2010) consists of the wind input (S_{in}) and wave breaking dissipation (S_{ds}) source terms, the DB1 switch is for the depth-induced breaking (S_{db}) source term, the BT1 switch consists of the bottom friction (S_{bt}) source term parameterizations, and the nonlinear quadruplet wave interactions (S_{nl}) are computed in the model using the NL1 switch.

We develop and test our accelerated WW3 code on GPUs on the following two computational platforms with heterogeneous architectures:

1. the Kodiak cluster from the Parallel Reconfigurable Observational Environment (PROBE) facility (Gibson et al., 2013) of Los Alamos National Laboratory. Kodiak has 133 compute nodes. Each node contains an Intel(R) Xeon(R) CPU (E5-2695 v4), 2.10 GHz with 36 CPU cores, and four NVIDIA Tesla P100 SXM2 general purpose computation on graphics processing units (GPGPUs), each with 16 GB of memory.

2. Summit, a high-performance computing system at Oak Ridge National Laboratory (ORNL). Summit has 4608 nodes; each node contains two IBM POWER9 CPUs and six NVIDIA Tesla V100 GPUs, each with 80 streaming multiprocessors. All are connected together with NVIDIA's high-speed NVLink. Summit is the fastest supercomputer in the US and the second fastest in the world (in 2021).

Table 1 shows the configuration of each compute node for both platforms. For a more accurate comparison of CPU and GPU codes, we used the same compiler. On Kodiak, the CPU Fortran code was compiled using the flags `-g -O3 -acc`. Similarly, the OpenACC code was compiled with flags `-g -O3 -acc -Minfo=accel -ta=tesla,ptxinfo,maxregcount:n`. Likewise, the flags for the CPU code on Summit are `-g -O2`, and those of OpenACC are `-g -O2 -acc -ta=tesla,ptxinfo,maxregcount:n -Minfo=accel`. Options `-ta=tesla:ptxinfo,maxregcount:n` are the optimization flags[1] used in this study (Sect. 3.2). Adding the option `-ta=tesla:ptxinfo` to the compile flags provides information about the amount of shared memory used per kernel (a function that is called by the CPU for execution on the GPU) as well the registers per thread. The flag `-ta=tesla:maxregcount:n`, where "n" is the number of registers, sets the maximum number of registers to use per thread.

As a test case, we performed a 5 h simulation from 00:00:00 to 05:00:00 on 1 June 2005 by forcing WW3 with atmospheric winds derived from the US National Center for Atmospheric Research (NCAR) reanalysis. We verify the GPU model for correctness using significant wave heights (SWHs) from the CPU-only simulation.

3 Results and evaluations

In this section, we first describe the performance of WW3 on a CPU and its computationally intensive sections. Furthermore, we discuss the challenges encountered, how GPU optimization is done, and present the performance result of porting WW3 on GPU. Lastly, we discuss the performance limitation using roofline analysis.

3.1 WW3 profiling analysis on a CPU

With model optimization, an important step is to find the runtime bottlenecks by measuring the performance of various sections in units of time and operations. In order to not waste time nor resources improving the performance of rarely used subroutines, we first need to figure out where WW3 spends most of its time. The technical term for this process is "profiling". For this purpose, we profile WW3 by running the Callgrind profiler from the Valgrind tool and then visualize the output using a KCachegrind tool (Weidendorfer, 2008). An application's performance can be 10× to 50× slower when profiling it with Callgrind; however, the proportions of time

[1]WW3 hangs when the `-O3` optimization flag is used.

Table 2. The speedups and simulation times associated with offloading one MPI rank exclusively to one GPU on Kodiak and Summit with 59K and 228K mesh sizes for MPI ranks ranging from 1 to 32.

		MPI ranks	CPU	GPU	Speedup
Summit	59K	1	6688.97	1429.92	4.7
		2	3347.84	748.39	4.5
		4	1685.4	386.4	4.4
		8	842.91	205.45	4.1
		16	435.5	113.63	3.8
		32	236.84	64.39	3.7
	228K	1	38 982.1	5905.57	6.6
		2	19 827.3	3042.59	6.5
		4	9901.16	1544.22	6.4
		8	3424	778.45	4.4
		16	1720.87	399.59	4.3
		32	878.05	219	4.0
Kodiak	59K	1	5574.38	2477.38	2.3
		2	2856.38	1291.53	2.2
		4	1564.3	700.68	2.2
		8	873.05	373.78	2.3
		16	398.54	195.99	2.0
		32	239.64	114.2	2.1
	228K	1	22 864.27	9636.75	2.4
		2	12 184.45	5592.38	2.2
		4	6313.02	2964.1	2.1
		8	3423.55	1448.44	2.4
		16	1586.75	746.27	2.1
		32	871.22	381.01	2.3

remain the same. Figure 4 shows the call graph obtained by profiling WW3 with 300 MPI ranks. The source term subroutine, `W3SRCEMD`, can easily be spotted as the consumer of ∼ 82 % of the total execution time and resources. Within the `W3SRCEMD` subroutine, the dissipation source term S_{ds} uses more than 40 % of the total runtime because it contains numerous time-consuming spectral loops. In fact, profiling WW3 with another profiler (not shown), Intel Advisor (Intel Corporation, 2021), specifically highlights the time-consuming spectral loops. In WW3, each processor serially runs through its sets of allocated spatial grids (described in Sect. 2.2), with each containing spectral grid points, and `W3SRCEMD` has a time-dynamic integration (Fig. 1b) procedure which can not be parallelized. Looping through spectral grids and the time-dynamic integration procedure are plausible reasons why `W3SRCEMD` is the bottleneck of WW3. We refer to the WW3 model with GPU-accelerated `W3SRCEMD` as "WW3-W3SRCEMD.gpu", and we refer to the CPU-only version as "WW3.cpu".

Fortunately, `W3SRCEMD` does not contain neighboring grid dependencies in the spatial nor spectral grids, i.e., no parallel data transfers; therefore, `W3SRCEMD` can be ported to GPUs with less difficulty. We moved the entire WW3 source term computation to GPU, as shown in Fig. 3b.

3.2 Challenges and optimization

Once the program hot spot is ported to the GPU, the GPU code needs to be optimized in order to improve its performance. Conventionally, the optimization of GPU codes involves loop optimization (fusion and collapse), data transfer management (CPU to GPU and GPU to CPU), memory management, and occupancy. Some of these optimization techniques are interrelated (e.g., memory management and occupancy). The WW3 model contains very few collapsible loops, so loop fusing and loop collapsing did not effectively optimize the code (not shown). To successfully port WW3-W3SRCEMD.gpu and achieve the best performance, two challenges had to be overcome in this study. The first is a data transfer issue caused by the WW3 data structure, while the second is a memory management and occupancy issue caused by the use of many local arrays, sometimes of spectral length, and scalars within `W3SRCEMD` and its embedded subroutines.

3.2.1 Data transfer management

It is important to understand the layout of data structures in the program before porting to a GPU. WW3 outlines its data structures using modules, e.g., `W3ADATMD`, `W3GDATMD`, and

```
 1 MODULE W3WAVEMD
 2     USE CONSTANTS
 3     USE W3GDATMD
 4     !! use other modules !!
 5     REAL        :: Variables
 6
 7     !! ============== !!
 8     !! BUNCH OF CODES !!
 9     !! ============== !!
10
11     DO IT=IT0, NT !Time loop
12         !! ============== !!
13         !! BUNCH OF CODES !!
14         !! ============== !!
15
16         DO JSEA=1, NSEAL !grids loop
17             !! BUNCH OF CODES !!
18             CALL W3SRCEMD(variable list)
19             !! BUNCH OF CODES !!
20         END DO
21
22     !! ============== !!
23     !! BUNCH OF CODES !!
24     !! ============== !!
25     END DO !Time Loop
26     !! ============== !!
27     !! BUNCH OF CODES !!
28     !! ============== !!
29 END MODULE W3WAVEMD
```

(a)

```
 1 MODULE W3WAVEMD
 2     USE CONSTANTS
 3     USE W3GDATMD
 4     !! use other modules !!
 5     REAL        :: Variables
 6 !$acc declare create(needed variables on GPU)
 7     !! BUNCH OF CODES !!
 8 !$acc enter data copyin(variables to GPU)
 9     DO IT=IT0, NT !Time loop
10         !! BUNCH OF CODES !!
11 !$acc update device(GPU data)
12 !$acc parallel copy(NSEAL) num_gangs(NSEAL)
13 !$acc loop gang
14         DO JSEA=1, NSEAL !grids loop
15             !! BUNCH OF CODES !!
16             CALL W3SRCEMD(variable list)
17             !! BUNCH OF CODES !!
18         END DO
19
20 !$acc end parallel
21 !acc update host(needed variables on CPU)
22     !! ============== !!
23     !! BUNCH OF CODES !!
24     !! ============== !!
25     END DO !Time Loop
26     !! BUNCH OF CODES !!
27
28 !$acc exit data delete(all GPU variables)
29 END MODULE W3WAVEMD
```

(b)

Figure 3. A schematic representation of the WAVEWATCH III original Fortran source code for the W3WAVEMD module **(a)** and its OpenACC directives version **(b)**.

W3ODATMD (lines 2 and 3 of Fig. 3a). Depending on the variable required, each subroutine uses these modules. These variables are called global external variables. As it is not possible to move data within a compute kernel in a GPU, all necessary data must be present on the GPU prior to launching the kernel that calls W3SRCEMD. The structure of WW3 requires the use of a routine directive (!$acc routine) to create a device version of W3SRCEMD, as well as other subroutines in it. In Fortran, the routine directive appears in the subprogram specification section or its interface block. Due to the use of routine directives, OpenACC declare directives (!$acc declare create; line 6 of Fig. 3b) were added to the data structures module to inform the compiler that global variables need to be created in the device memory as well as the host memory.

All data must be allocated on the host before being created on a device unless they have been declared device residents. In WW3, however, arrays whose size are determined by spatial–spectral grid information are allocated at runtime, rather than within the data structure modules. Thus, creating data on the device within the WW3 data modules poses a problem. Due to this restriction, it was then necessary to explicitly move all of the required data to the GPU before launching the kernel. For managing data transfers between iteration cycles, we use !$acc update device(variables) and !$acc update self(variables) (lines 11 and 21 in Fig. 3b). However, it would be easier to use unified memory, which offers a unified memory address space to both CPUs and GPUs, rather than tracking each and every piece of data that needs to be sent to the GPU, but the latest OpenACC version does not support the use of unified memory with routine directives. In the future, having this feature would save time spent tracking data transfers in programs with many variables such as WW3.

3.2.2 Memory management and occupancy

Occupancy is defined as the ratio of active warps (workers) on a streaming multiprocessor (SM) to the maximum number of active warps supported by the SM. On Kodiak and Summit, the maximum number of threads per SM is 2048. A warp consists of 32 threads which implies that the total number of possible warps per SM is 8. Even if the kernel launch con-

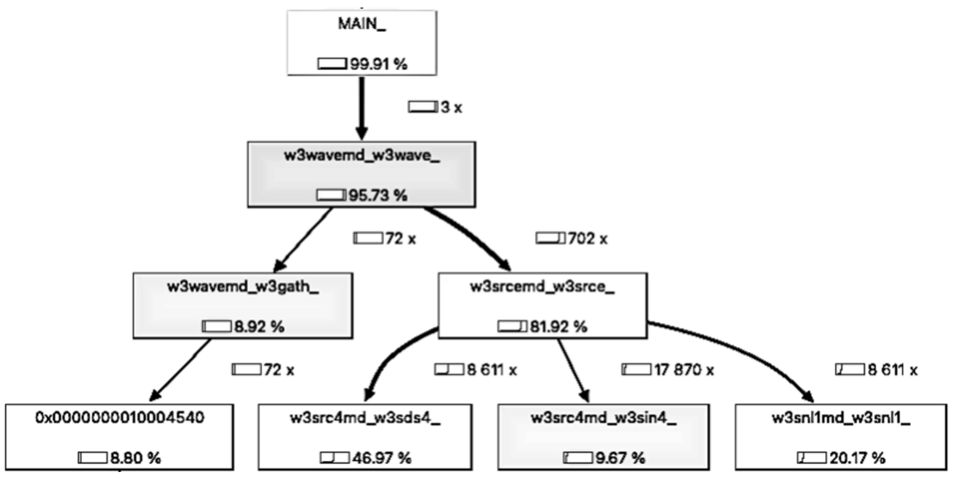


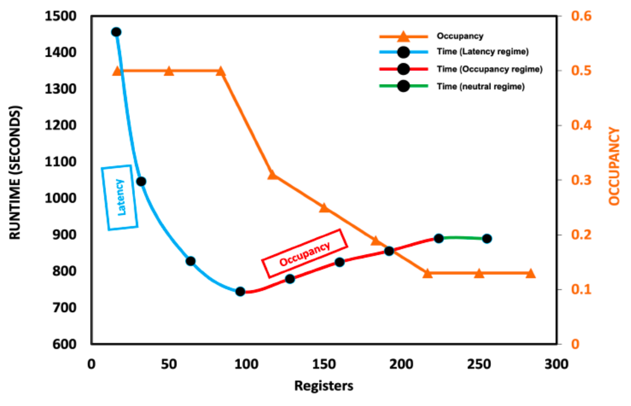
Figure 4. The call graph obtained by profiling WAVEWATCH III with 300 CPU cores on Summit. Each box includes the name of the subroutine and its relative execution time as a percentage.

figuration maximizes the number of threads per SM, other GPU resources, such as shared memory and registers, may also limit the number of maximum threads, thus indirectly affecting GPU occupancy. A register is a small amount of fast storage available to each thread, whereas shared memory is the memory shared by all threads in each SM. As seen in Table 1, the maximum memory per SM for Kodiak and Summit is 256 kB, but the maximum shared memory is 64 kB for Kodiak (P100) and 94 kB for Summit (V100). In terms of total register file size per GPU, Kodiak and Summit have 14 336 and 20 480 kB, respectively.

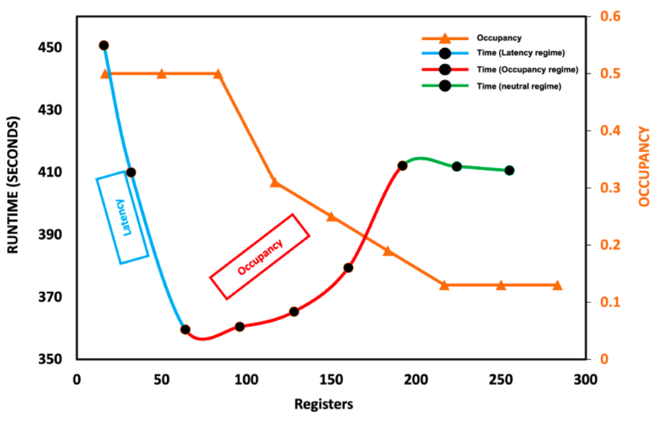
To estimate GPU usage, we use the CUDA occupancy calculator available at https://docs.nvidia.com/cuda/cuda-occupancy-calculator/CUDA_Occupancy_Calculator.xls (last access: 30 November 2022). In this study, the kernel launch configurations consist of NSEAL gangs, where NSEAL refers to the number of grids on each node, and each gang has 32 vector lengths (or threads). With this configuration, the achievable GPU occupancy (regardless of other resources) is 50 %. Adding `-ta=tesla:pxtinfo` to the compiler flags provides information about the size of registers, memory spills (movement of variables out of the register space to the main memory), and shared memory allocated during compilation. Kodiak and Summit both allocate the maximum 255 kB register per thread, reducing the GPU occupancy to 13 %. In addition, a full register leads to the spilling of memory into the Level 1 (L1) cache (shared memory). A spill to cache is fine, but a spill to the global memory will severely affect performance because the time required to get data from the global memory is longer than the time required to get data from a register. This phenomenon is known as latency: the amount of time required to move data from one point to the other. Therefore, an increase (decrease) in register size causes two different things simultaneously: a decrease (increase) in occupancy and a decrease (increase) in latency. There is always a trade-off between register, latency, and occupancy, and the trick is to find the spot that maximizes performance. One can set the maximum number of registers per thread via the `-ta=tesla:maxregcount:n` flag, where "n" is the number of registers.

Figure 5 illustrates how the trade-off between latency and occupancy affects the GPU performance based on the number of registers. Our analysis was only based on the performance of running a 228K mesh configuration with 16 MPI tasks. For Summit, as the number of registers increases from 16 to 64, the GPU occupancy remains constant at 50 % and GPU performance improves due to the movement of more variables to the fast memory. As indicated by the blue part of the line, this is a latency-dominant region. However, as the number of register increases from 64 to 192, GPU occupancy gradually decreases from 50 % to 13 %. This degraded performance despite moving more data into the fast memory. Therefore, this is an occupancy-dominant region, as indicated by the red part of the line (Fig. 5b). From 192 to the maximum register count, the occupancy remains constant at 13 % and GPU performance remains relatively constant. As occupancy is constant, latency is expected to dominate this region, which is represented by the green part of the line in Fig. 5b. However, we observe no memory spill for registers between 192 and 255 and, thus, the latency effect remains unchanged. Therefore, constant latency and occupancy result in constant performance in this region.

Figure 5b shows that 64 registers produced the best performance (minimum runtime) on Summit. For brevity, the previously described trade-off analysis also applies to Kodiak, and a 96 register count produced the best performance (Fig. 5a). On Kodiak, the register count that achieved the best performance is higher than on Summit, probably due to Summit's larger L1 and Level 2 (L2) caches (Table 1). With larger L1



(a)



(b)

Figure 5. The WW3-W3SRCEMD.gpu runtime (primary vertical axis) and occupancy (secondary vertical axis in orange) based on register counts on Kodiak **(a)** and Summit **(b)**. For the runtime, the light blue part of the line represents the latency-dominant region, the red part represents the occupancy-dominant region, and the green part represents the neutral region. We analyzed the result of running a 228K mesh size on 16 GPUs and CPUs.

and L2 caches, more data can be stored, reducing memory spillover to global memory and, thus, reducing latency. Optimizing the register count increases the GPU performance by approximately 20 % on Kodiak and by 14 % on Summit.

3.3 GPU-accelerated `W3SRCEMD`

This section compares the performance of WW3.cpu and WW3-W3SRCEMD.gpu. Note that the speedups in this section are achieved by parallelizing the local grids' loop, which calls the `W3SRCEMD` function, using the OpenACC parallel directives (Fig. 3). In `W3SRCEMD` and its dependent subroutines, we introduced the OpenACC routine directive (`!$acc routine`) which instructs the compiler to build a device version of the subroutine so that it may be called from a device region by each gang. In addition, at the start of the time integration, we moved the required constants' data to the GPU (line 8 of Fig. 3). Using the average over the last 2 h simulation, Fig. 7 compares the output results of CPU and WW3-W3SRCEMD.gpu codes as well as their relative difference for significant wave height (SWH). According to the validation results, the SWH is nearly identical, and the error is negligible and acceptable. It is possible that the error stems from the difference in mathematical precision between the GPU and CPU.

For simplicity, we start by mapping one MPI rank to one GPU. Comparing the performance of a single GPU with a single CPU core (Table 2) on Kodiak, 2.3× and 2.4× speedups were achieved for the 59K and 228K meshes, respectively. Here, the speedup is relative to a single CPU core. Similarly, we achieved speedups of 4.7× and 6.6× on Summit for the 59K and 228K meshes, respectively. On Summit, the GPU performance of 228K nodes is better because the CPU gets extremely slow. Summit's speedup is greater than Kodiak's because the Tesla V100 SXM2 16GB GPU is faster than the Tesla P100 PCIe 16GB GPU (NVIDIA, 2017). Due to the reduction in GPU workload, speedups gradually decline as the number of MPI ranks increases (Table 2).

3.3.1 Fair comparison using multiprocess service (MPS)

The following sections focus exclusively on Summit's results. Each Summit compute node is equipped with 6 GPUs and 42 CPU cores. As an initial step, we used only six MPI ranks on each node so that each process could offload its work to one GPU. In order to properly compare CPUs and GPUs on a single node, we must use all of the CPU and GPU resources on a node. The NVIDIA Volta GPUs support multiprocess services. When running with a mesh size greater than 59K and assigning multiple MPI ranks to a single GPU, the GPU heap size overflows with an auto-allocation problem. The auto-allocation problem is solved by setting the environment variable "PGI_ ACC_CUDA_HEAPSIZE" to a minimum allowed number based on the mesh size and MPI ranks per GPU. By mapping seven MPI ranks per GPU on Summit, a speedup of 1.36× (1.41×) was achieved for the 59K (228K) mesh size (Fig. 6) over all of the CPU cores. We also ran four, three, and two MPI ranks per GPU configuration. The results of the different multiprocess configurations and the two mesh sizes are shown in Table 3. Note that the speedups of all the GPU configurations are compared with the full 42-cores CPU run. By varying the number of MPI ranks per GPU, it can be seen that the 4 and 3 MPI ranks per GPU (24 and 18 MPI ranks in total) have higher speedups than 7 MPIs per GPU for the 59K mesh size. Even with 2 MPI ranks per GPU (total of 12 ranks), the speedup is 1.23×. The speedup for the 228K gradually decreases from 1.41× to 1.12× as the number of MPI ranks per GPU decreases from seven to two. The workload (distributed grids) per MPI rank decreases as the number of MPI ranks increases. As a result

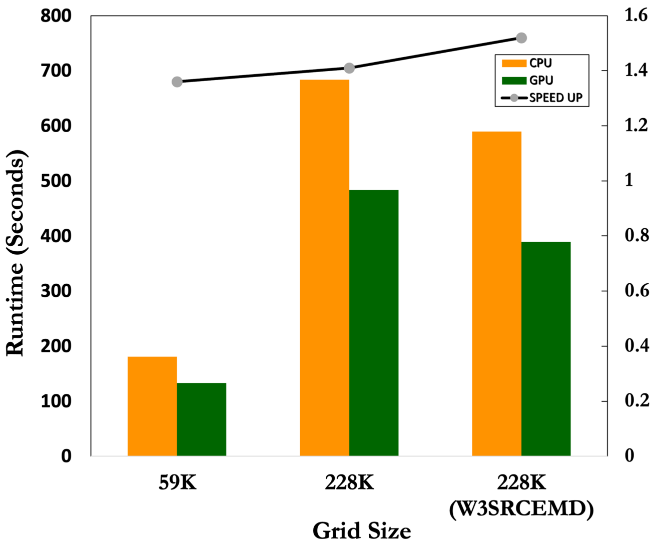


Figure 6. The runtime of WW3.cpu (orange) and WW3-W3SRCEMD.gpu (green) for 42 CPU cores and 7 MPI ranks per GPU on a Summit node with 59K and 228K meshes. The right-hand bar chart shows the runtimes for only the GPU-accelerated subroutine, `W3SRCEMD`, on a 228K grid size. The black line represents the speedup of WW3-W3SRCEMD.gpu over WW3.cpu.

of the reduced workload, GPU utilization falls. The overall performance of a GPU depends on both the number of MPI ranks per GPU and the number of the grid size per MPI rank.

Table 4 shows the results of scaling the 228K mesh size over multiple nodes. We used the full GPU configuration here by launching seven MPI ranks on each GPU. In the results, it can be seen that the speedup is relatively uniform across multiple nodes. Likewise, as the number of grid points per MPI rank decreases with increasing nodes, the speedup gradually decreases due to reduced GPU utilization. However, the speedup is always the highest whenever the number of grid points per MPI rank is $\sim$ around 2500, e.g., the speedup of 2 nodes in Table 4 for 228K and the speedup of 24 MPI ranks for the 59K mesh in Table 3. This is most likely due to the tail effect – load imbalance between SMs. Based on our kernel launch configuration, 64 register and a 32 block size, the number of possible blocks per SM is limited to 32. With 80 SMs on a V100 GPU, the maximum total number of blocks (gangs, i.e., grid points) that could be executed simultaneously is 80×32 (2560).

All previous speedups are based on the whole WW3 code. However, only `W3SRCEMD` is accelerated on the GPU, whereas the rest of the code is run on the CPU. Thus, when comparing 24 MPI ranks in the GPU configuration with the 42 CPU cores, 24 MPI ranks are being used for the CPU section of WW3-W3SRCEMD.gpu against 42 MPI ranks for the WW3.cpu. Therefore, it is necessary to also consider only the speedup of `W3SRCEMD` on the GPU over the CPU. The last row of Table 3 (also Fig. 6) shows the runtimes and the achieved speedup of the `W3SRCEMD` subroutine for a 288K mesh size on a Summit node. In comparison with the speedup based on the whole WW3 code, the `W3SRCEMD` speedup increases for all configurations with a maximum speedup of 1.61×.

3.4 `W3SRCEMD` roofline plot

The roofline model helps us understand the trade-off between data movement and computation, so we can find out what is limiting our code and how close we are to it. The roofline (Fig. 8) indicates that, as expected from a memory-intensive model, the kernel is limited by the data transfer bandwidth between the CPU and the GPU. Most of the kernel's time is spent executing memory (load/store) instructions. It is worth mentioning that the device achieved a compute throughput and a memory bandwidth that are both below 40 % of its peak performance. Therefore, it appears that, while the computation is waiting for the GPU to provide required data, the GPU is waiting for the CPU to transfer data. Similarly, the roofline shows that the kernel memory bandwidth is approximately equal to the NVLink bandwidth. Thus, the `W3SRCEMD` kernel is limited by the bandwidth between the CPU and the GPU memory.

The kernel also has a very low arithmetic intensity for both single- and double-precision floating-point (Fig. 8) computations, performing only very few flops for every double and integer loaded from dynamic random access memory (DRAM). Arithmetic intensity is a measure of floating-point operations (FLOPs) performed relative to the amount of memory accesses in bytes that are needed for those operations. For our 5 h simulation, the kernel is being launched 21 times; therefore, there are 42 data movements between the host and the device for nonconstant variables. Unfortunately, the large arrays need to be updated on both the device and host at each time step. As an example, VA, the spectra storage array, is approximately 5 GB (20 GB) for a spatial mesh size of 59 000 (228 000) and a spectral resolution of 50×36. In summary, the W3SRCE subroutine is simply too big and complicated to be ported efficiently using OpenACC routine directives and, therefore, requires significant refactoring.

4 Conclusions

Climate science is increasingly moving towards higher-spatial-resolution models and additional components to better simulate previously parameterized or excluded processes. In recent decades, the use of GPUs to accelerate scientific problems has increased significantly due to the emergence of supercomputers with heterogeneous architectures.

Wind-generated waves play an important role in modifying physical processes at the atmosphere–ocean interface. They have generally been excluded from most coupled Earth system models, partly due to the high computational cost of

Table 3. Node baseline comparison runtime (seconds) with different MPI ranks per GPU configuration on a Summit node. The numbers in square brackets are the speedup relative to the whole CPU on a Summit node (42 MPI ranks). In the last row, we present the speedups and simulation times when comparing only the GPU-accelerated subroutine, `W3SRCEMD`.

	CPU	GPU			
Mesh size	42 ranks	42 ranks (7 per GPU)	24 ranks (4 per GPU)	18 ranks (3 per GPU)	12 ranks (2 per GPU)
59K	180.72	133.18 [1.36×]	124.28 [1.45×]	127.17 [1.42×]	146.34 [1.23×]
228K	683.66	483.22 [1.41×]	523.91 [1.30×]	533.08 [1.24×]	612.60 [1.12×]
228K (`W3SRCEMD`)	589.56	389.12 [1.52×]	407.54 [1.45×]	366.00 [1.61×]	395.31 [1.49×]

Table 4. The speedup of multiple nodes for the 228K mesh size on Summit. We used all 42 CPU cores and 6 GPU on each node with a configuration of 7 MPI ranks per GPU.

Nodes	Grid points (per rank)	CPU	GPU	Speedup
1	5441	683.66	483.22	1.41×
2	2721	354.12	224.69	1.58×
3	1814	238.02	171.15	1.39×
4	1360	195.59	141.44	1.38×
5	1088	154.17	126.84	1.22×

modeling them. However, the Energy Exascale Earth System Model (E3SM) project seeks to include a wave model (WW3), and introducing WW3 to E3SM would increase the computational time and usage of resources.

In this study, we identified and accelerated the computationally intensive section of WW3 on a GPU using OpenACC. Using the Valgrind and Callgrind tools, we found that the source term subroutine, `W3SRCEMD`, consumes 78 % of the execution time. The `W3SRCEMD` subroutine has no neighboring grid point dependencies and is, therefore, well suited for implementation on a GPU. Using two different computational platforms, Kodiak with four P100 GPUs and Summit with six V100 GPUs on each node, we performed 5 h simulation experiments using two global unstructured meshes with 59 072 and 228 540 nodes. On average, running `W3SRCEMD` by offloading one MPI per GPU gives an approximate 4× (2×) speedup over the CPU version on Summit (Kodiak). Via a fair comparison, using all 42 CPU cores and 6 GPUs on a Summit node, a speedup of 1.4× was achieved using 7 MPI ranks per GPU. However, the GPU speedup over the 42 CPU cores remains relatively unchanged (∼ 1.3×), even when using 4 MPI ranks per GPU (24 ranks in total) and 3 MPI ranks per GPU (18 ranks in total). The GPU performance is heavily affected by the data transfer bandwidth between the CPU and the GPU. The large number of local scalars and arrays within the `W3SRCEMD` subroutine and the huge amount of memory required to run WW3 is currently hurting the GPU utilization and, thus, the achievable speedup. Too many constants in WW3 occupy the register (fast memory) and then spill over to the L1 and L2 caches or the GPU global memory. To increase the GPU performance, the grid loop counter within W3WAVEMD must be pushed into the `W3SRCEMD`, thereby moving the gang-level parallelization into `W3SRCEMD`. This requires major code refactoring, starting with modification of WW3 data structures. There are other parts of the WW3 code that can be ported to GPU, such as spatial and spectral propagation.

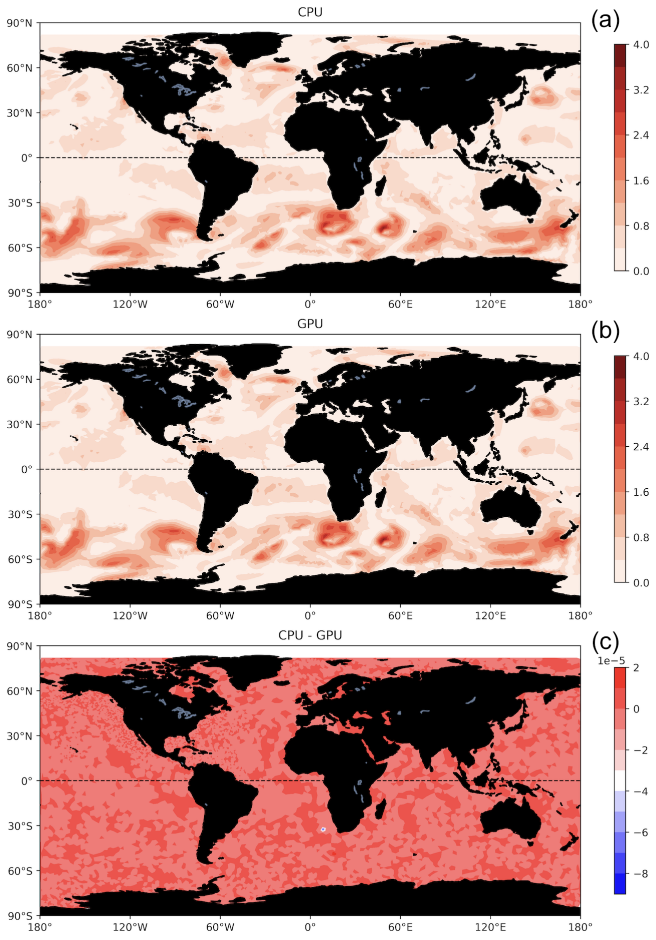


Figure 7. The average of WW3.cpu **(a)** and WW3-W3SRCEMD.gpu **(b)** last 2 h simulations and their differences **(c)** for significant wave height.

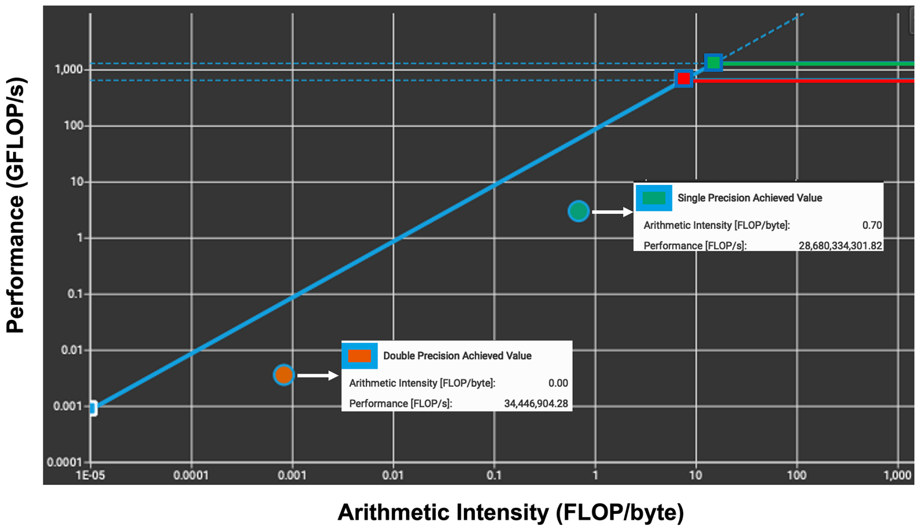


Figure 8. Roofline model for the `W3SRCEMD` kernel from NVIDIA Nsight Compute. The red and green dots are the double- and single-precision values, respectively.

Coupling the CESM with WW3 at low resolution, Li et al. (2016) found a 36 % and 28 % increase in computational cost for ocean-wave-only and fully coupled simulations when running WW3 on a 3.2° × 4° latitude–longitude grid with 25 frequency and 24 directional bins with a 3° resolution ocean model and T31 atmosphere. Likewise, in a one-way coupling of WW3 to the E3SM atmospheric component, WW3 increases the number of processors by ∼ 35 % and its runtime is 44 % longer than the ocean model. From our first attempt at GPU-based spectral wave modeling, the runtime decreased by 35 %–40 % and resource usage decreased by 40 %–55 %. Thus, leveraging heterogeneous architectures reduces the amount of time and resources required to include WW3 in global climate models. It is important to note that WW3 will become a bottleneck as the ocean and atmosphere models in E3SM move towards heterogeneous architectures. Consequently, WW3 needs huge refactoring to take advantage of GPU capabilities and be fully prepared for the Exascale regime.

After refactoring the WW3 code, we also need to investigate how different WW3 setups (grid parallelization method, source term switches, propagation schemes, etc.) affect GPU performance. The performance of GPU-accelerated WW3 using OpenMP should also be considered for future work. The success of this work has laid the foundation for future work in global spectral wave modeling, and it is also a major step toward expanding E3SM's capability to run with waves on heterogeneous architectures in the near future.

Code and data availability. The model configuration and input files can be assessed at https://doi.org/10.5281/zenodo.6483480 (Ikuyajolu et al., 2022a). The official repository of the WAVEWATCH III CPU code can be found at https://github.com/NOAA-EMC/WW3 (last access: 30 November 2022). The new WW3-W3SRCEMD.gpu code used in this work is available at https://doi.org/10.5281/zenodo.6483401 (Ikuyajolu et al., 2022b).

Author contributions. OJI was responsible for the code modifications, simulations, verification tests, and performance analyses. LVR developed the concept for this study. OJI provided the initial draft of the manuscript. LVR, SRB, EET, YD, and SS contributed to the final manuscript.

Competing interests. The contact author has declared that none of the authors has any competing interests.

Acknowledgements. This research was supported as part of the Energy Exascale Earth System Model (E3SM) project, funded by the US Department of Energy, Office of Science, Biological and Environmental Research program. This research used resources provided by the Los Alamos National Laboratory (LANL) Institutional Computing Program, which is supported by the US Department of Energy via the National Nuclear Security Administration (contract no. 89233218CNA000001). This research also used resources from the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the US Department of Energy's Office of Science (contract no. DE-AC05-00OR22725). We wish to thank Phil Jones at LANL for assistance and comments on early drafts of the paper. Lastly, the authors would like to thank Kim Youngsung at ORNL for technical assistance with the revised manuscript.

Financial support. This research has been supported by the US Department of Energy's Office of Science (ESMD-SFA).

Review statement. This paper was edited by Julia Hargreaves and reviewed by three anonymous referees.

References

Abdolali, A., Roland, A., van der Westhuysen, A., Meixner, J., Chawla, A., Hesser, T. J., Smith, J. M., and Sikiric, M. D.: Large-scale hurricane modeling using domain decomposition parallelization and implicit scheme implemented in WAVEWATCH III wave model, Coast. Eng., 157, 103656, https://doi.org/10.1016/j.coastaleng.2020.103656, 2020.

Alves, J.-H. G. M., Chawla, A., Tolman, H. L., Schwab, D., Lang, G., and Mann, G.: The operational implementation of a Great Lakes wave forecasting system at NOAA/NCEP, Weather Forecast., 29, 1473–1497, 2014.

Ardhuin, F., Rogers, E., Babanin, A. V., Filipot, J., Magne, R., Roland, A., van der Westhuysen, A., Queffeulou, P., Lefevre, J., Aouf, L., and Collard, F.: Semiempirical Dissipation Source Functions for Ocean Waves. Part I: Definition, Calibration, and Validation, J. Phys. Oceanogr., 40, 1917–1941, 2010.

Bao, Y., Song, Z., and Qiao, F.: FIO-ESM Version 2.0: Model Description and Evaluation, J. Geophys. Res.-Oceans, 125, e2019JC016036, https://doi.org/10.1029/2019JC016036, 2020.

Bertagna, L., Guba, O., Taylor, M. A., Foucar, J. G., Larkin, J., Bradley, A. M., Rajamanickam, S., and Salinger, A. G.: A Performance-Portable Nonhydrostatic Atmospheric Dycore for the Energy Exascale Earth System Model Running at Cloud-Resolving Resolutions, SC '20, IEEE Press, https://doi.org/10.1109/SC41405.2020.00096, 2020.

Bieringer, P. E., Piña, A. J., Lorenzetti, D. M., Jonker, H. J. J., Sohn, M. D., Annunzio, A. J., and Fry, R. N.: A Graphics Processing Unit (GPU) Approach to Large Eddy Simulation (LES) for Transport and Contaminant Dispersion, Atmosphere, 12, 890, https://doi.org/10.3390/atmos12070890, 2021.

Brus, S. R., Wolfram, P. J., Van Roekel, L. P., and Meixner, J. D.: Unstructured global to coastal wave modeling for the Energy Exascale Earth System Model using WAVEWATCH III version 6.07, Geosci. Model Dev., 14, 2917–2938, https://doi.org/10.5194/gmd-14-2917-2021, 2021.

Bryan, K. and Cox, M. D.: A numerical investigation of the oceanic general circulation, Tellus, 19, 54–80, https://doi.org/10.3402/tellusa.v19i1.9761, 1967.

Cavaleri, L., Fox-Kemper, B., and Hemer, M.: Wind Waves in the Coupled Climate System, B. Am. Meteorol. Soc., 93, 1651–1661, https://doi.org/10.1175/BAMS-D-11-00170.1, 2012.

Chandrasekaran, S. and Juckeland, G.: OpenACC for Programmers: Concepts and Strategies, 1st Edn., Addison-Wesley Professional, ISBN 978-0134694283, 2017.

Chawla, A., Spindler, D. M., and Tolman, H. L.: Validation of a thirty year wave hindcast using the Climate Forecast System Reanalysis winds, Ocean Model., 70, 189–206, 2013a.

Chawla, A., Tolman, H. L., Gerald, V., Spindler, D., Spindler, T., Alves, J.-H. G. M., Cao, D., Hanson, J. L., and Devaliere, E.-M.: A multigrid wave forecasting model: A new paradigm in operational wave forecasting, Weather Forecast., 28, 1057–1078, 2013b.

Cornett, A. M.: A global wave energy resource assessment, in: The Eighteenth International Offshore and Polar Engineering Conference, International Society of Offshore and Polar Engineers, ISOPE-I-08-370, 2008.

Danabasoglu, G., Lamarque, J.-F., Bacmeister, J., Bailey, D. A., DuVivier, A. K., Edwards, J., Emmons, L. K., Fasullo, J., Garcia, R., Gettelman, A., Hannay, C., Holland, M. M., Large, W. G., Lauritzen, P. H., Lawrence, D. M., Lenaerts, J. T. M., Lindsay, K., Lipscomb, W. H., Mills, M. J., Neale, R., Oleson, K. W., Otto-Bliesner, B., Phillips, A. S., Sacks, W., Tilmes, S., van Kampenhout, L., Vertenstein, M., Bertini, A., Dennis, J., Deser, C., Fischer, C., Fox-Kemper, B., Kay, J. E., Kinnison, D., Kushner, P. J., Larson, V. E., Long, M. C., Mickelson, S., Moore, J. K., Nienhouse, E., Polvani, L., Rasch, P. J., and Strand, W. G.: The Community Earth System Model Version 2 (CESM2), J. Adv. Model. Earth Sy., 12, e2019MS001916, https://doi.org/10.1029/2019MS001916, 2020.

Fan, Y. and Griffies, S. M.: Impacts of Parameterized Langmuir Turbulence and Nonbreaking Wave Mixing in Global Climate Simulations, J. Climate, 27, 4752–4775, https://doi.org/10.1175/JCLI-D-13-00583.1, 2014.

Gibson, G., Grider, G., Jacobson, A., and Lloyd, W.: PRObE: A thousand-node experimental cluster for computer systems research, Usenix ;login, 38, https://www.usenix.org/system/files/login/articles/07_gibson_036-039_final.pdf (last access: 2 June 2022), 2013.

Govett, M., Rosinski, J., Middlecoff, J., Henderson, T., Lee, J., MacDonald, A., Wang, N., Madden, P., Schramm, J., and Duarte, A.: Parallelization and Performance of the NIM Weather Model on CPU, GPU, and MIC Processors, B. Am. Meteorol. Soc., 98, 2201–2213, https://doi.org/10.1175/BAMS-D-15-00278.1, 2017.

Hanappe, P., Beurivé, A., Laguzet, F., Steels, L., Bellouin, N., Boucher, O., Yamazaki, Y. H., Aina, T., and Allen, M.: FAMOUS, faster: using parallel computing techniques to accelerate the FAMOUS/HadCM3 climate model with a focus on the radiative transfer algorithm, Geosci. Model Dev., 4, 835–844, https://doi.org/10.5194/gmd-4-835-2011, 2011.

Ikuyajolu, O. J., Van Roekel, L., Brus, S., Thomas, E. E., and Deng, Y.: Porting the WAVEWATCH III Wave Action Source Terms to GPU – WaveWatchIII configuration files, Zenodo [data set], https://doi.org/10.5281/zenodo.6483480, 2022a.

Ikuyajolu, O. J., Van Roekel, L., Brus, S., Thomas, E. E., and Deng, Y.: Porting the WAVEWATCH III Wave Action Source Terms to GPU – Code Base (1.0.0), Zenodo [code], https://doi.org/10.5281/zenodo.6483401, 2022b.

Intel Corporation: Intel Advisor User Guide Version 2022.0, Intel Corporation, https://www.intel.com/content/www/us/en/develop/documentation/advisor-user-guide/top.html (last access: 30 November 2022), 2021.

Jiang, J., Lin, P., Wang, J., Liu, H., Chi, X., Hao, H., Wang, Y., Wang, W., and Zhang, L.: Porting LASG/ IAP Climate System Ocean Model to Gpus Using OpenAcc, IEEE Access, 7, 154490–154501, https://doi.org/10.1109/ACCESS.2019.2932443, 2019.

Law Chune, S. and Aouf, L.: Wave effects in global ocean modeling: parametrizations vs. forcing from a wave model, Ocean Dynam., 68, 1739–1758, https://doi.org/10.1007/s10236-018-1220-2, 2018.

Li, J.-G.: Propagation of ocean surface waves on a spherical multiple-cell grid, J. Comput. Phys., 231, 8262–8277, https://doi.org/10.1016/j.jcp.2012.08.007, 2012.

Li, Q. and Van Roekel, L.: Towards multiscale modeling of ocean surface turbulent mixing using coupled MPAS-Ocean v6.3 and PALM v5.0, Geosci. Model Dev., 14, 2011–2028, https://doi.org/10.5194/gmd-14-2011-2021, 2021.

Li, Q., Webb, A., Fox-Kemper, B., Craig, A., Danabasoglu, G., Large, W. G., and Vertenstein, M.: Langmuir mixing effects on global climate: WAVEWATCH III in CESM, Ocean Model., 103, 145–160, https://doi.org/10.1016/j.ocemod.2015.07.020, 2016.

Michalakes, J. and Vachharajani, M.: GPU acceleration of numerical weather prediction, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, 14–18 April 2008, Miami, FL, USA, 1–7, https://doi.org/10.1109/IPDPS.2008.4536351, 2008.

Mielikainen, J., Huang, B., and Huang, H.-L. A.: GPU-Accelerated Multi-Profile Radiative Transfer Model for the Infrared Atmospheric Sounding Interferometer, IEEE J. Sel. Top. Appl., 4, 691–700, https://doi.org/10.1109/JSTARS.2011.2159195, 2011.

Norman, M. R., Mametjanov, A., and Taylor, M. A.: Exascale Programming Approaches for the Accelerated Model for Climate and Energy, https://doi.org/10.1201/b21930-9, 2017.

Norman, M. R., Bader, D. A., Eldred, C., Hannah, W. M., Hillman, B. R., Jones, C. R., Lee, J. M., Leung, L. R., Lyngaas, I., Pressel, K. G., Sreepathi, S., Taylor, M. A., and Yuan, X.: Unprecedented cloud resolution in a GPU-enabled full-physics atmospheric climate simulation on OLCF's summit supercomputer, Int. J. High Perform. Co., 36, 93–105, 2022.

NVIDIA: NVIDIA Tesla V100 GPU Architecture, Tech. rep., NVIDIA Corporation,http://www.nvidia.com/object/volta-architecture-whitepaper.html (last access: 2 June 2022), 2017.

Qiao, F., Song, Z., Bao, Y., Song, Y., Shu, Q., Huang, C., and Zhao, W.: Development and evaluation of an Earth System Model with surface gravity waves, J. Geophys. Res.-Oceans, 118, 4514–4524, https://doi.org/10.1002/jgrc.20327, 2013.

Roland, A.: Development of WWM II: Spectral wave modeling on unstructured meshes, PhD thesis, https://www.academia.edu/1548294/PhD_Thesis_Spectral_Wave_Modelling_on_Unstructured_Meshes (last access: 2 June 2022), 2008.

Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N., and Matsuoka, S.: An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code, in: SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 1–11, https://doi.org/10.1109/SC.2010.9, 2010.

Shimura, T., Mori, N., Takemi, T., and Mizuta, R.: Long-term impacts of ocean wave-dependent roughness on global climate systems, J. Geophys. Res.-Oceans, 122, 1995–2011, https://doi.org/10.1002/2016JC012621, 2017.

Song, Z., Qiao, F., and Song, Y.: Response of the equatorial basin-wide SST to non-breaking surface wave-induced mixing in a climate model: An amendment to tropical bias, J. Geophys. Res.-Oceans, 117, C00J26, https://doi.org/10.1029/2012JC007931, 2012.

The Wamdi Group: The WAM model – A third generation ocean wave prediction model, J. Phys. Oceanogr., 18, 1775–1810, 1988.

Tolman, H. L.: Distributed-memory concepts in the wave model WAVEWATCH III, Parallel Comput., 28, 35–52, https://doi.org/10.1016/S0167-8191(01)00130-2, 2002.

Tolman, H. L.: A mosaic approach to wind wave modeling, Ocean Model., 25, 35–47, https://doi.org/10.1016/j.ocemod.2008.06.005, 2008.

Wang, D.-P. and Oey, L.-Y.: Hindcast of waves and currents in Hurricane Katrina, B, B. Am. Meteorol. Soc, 89, 487–496, 2008.

WAVEWATCH III® Development Group: User manual and system documentation of WAVEWATCH III version 6.07, Tech. Note 333, NOAA/NWS/NCEP/MMAB, Tech. rep., College Park, MD, USA, 2019.

Weidendorfer, J.: Sequential Performance Analysis with Callgrind and KCachegrind, in: Tools for High Performance Computing, edited by: Resch, M., Keller, R., Himmler, V., Krammer, B., and Schulz, A., Springer Berlin Heidelberg, Berlin, Heidelberg, 93–113, https://doi.org/10.1007/978-3-540-68564-7_7, 2008.

Xiao, H., Sun, J., Bian, X., and Dai, Z.: GPU acceleration of the WSM6 cloud microphysics scheme in GRAPES model, Comput. Geosci., 59, 156–162, https://doi.org/10.1016/j.cageo.2013.06.016, 2013.

Xu, S., Huang, X., Oey, L.-Y., Xu, F., Fu, H., Zhang, Y., and Yang, G.: POM.gpu-v1.0: a GPU-based Princeton Ocean Model, Geosci. Model Dev., 8, 2815–2827, https://doi.org/10.5194/gmd-8-2815-2015, 2015.

Yuan, Y., Shi, F., Kirby, J. T., and Yu, F.: FUNWAVE-GPU: Multiple-GPU Acceleration of a Boussinesq-Type Wave Model, J. Adv. Model. Earth Sy., 12, e2019MS001957, https://doi.org/10.1029/2019MS001957, 2020.

Zhang, S., Fu, H., Wu, L., Li, Y., Wang, H., Zeng, Y., Duan, X., Wan, W., Wang, L., Zhuang, Y., Meng, H., Xu, K., Xu, P., Gan, L., Liu, Z., Wu, S., Chen, Y., Yu, H., Shi, S., Wang, L., Xu, S., Xue, W., Liu, W., Guo, Q., Zhang, J., Zhu, G., Tu, Y., Edwards, J., Baker, A., Yong, J., Yuan, M., Yu, Y., Zhang, Q., Liu, Z., Li, M., Jia, D., Yang, G., Wei, Z., Pan, J., Chang, P., Danabasoglu, G., Yeager, S., Rosenbloom, N., and Guo, Y.: Optimizing high-resolution Community Earth System Model on a heterogeneous many-core supercomputing platform, Geosci. Model Dev., 13, 4809–4829, https://doi.org/10.5194/gmd-13-4809-2020, 2020.