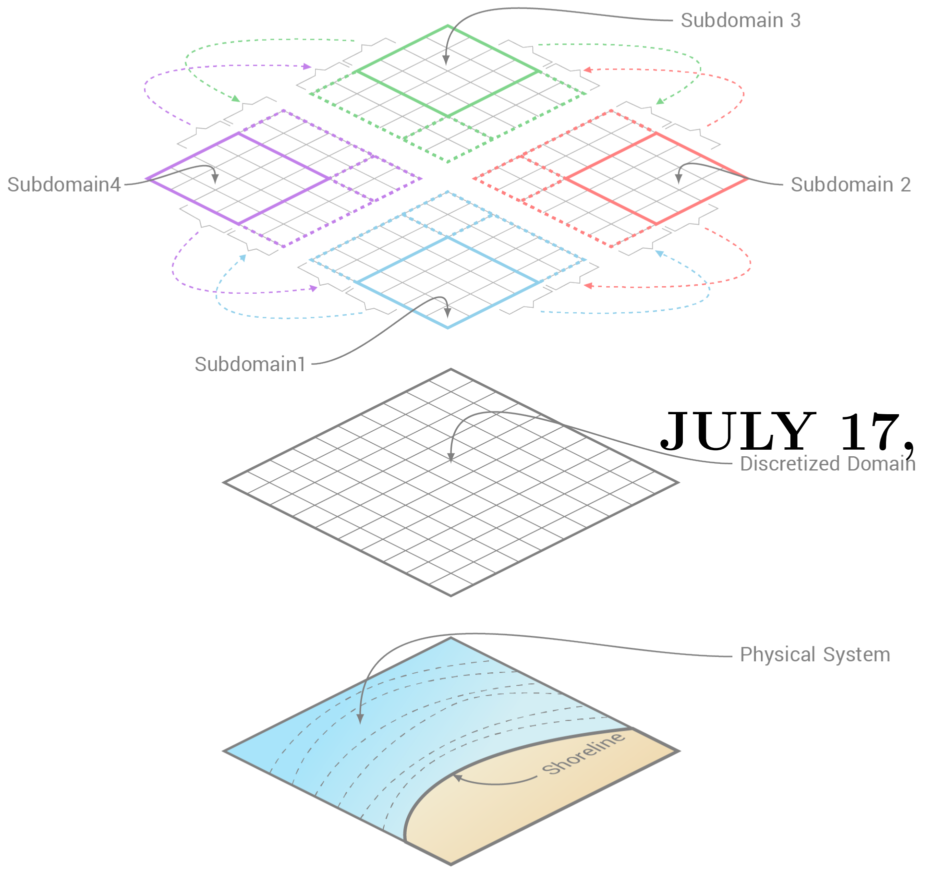
EFDC+ DOMAIN DECOMPOSITION: MPI-BASED IMPLEMENTATION



JULY 17, 2020

Edmonds, WA USA

`www.ds-intl.biz`

EXECUTIVE SUMMARY

The aim of this paper is to describe the domain decomposition approach based on Message Passing Interface (MPI), which has recently been implemented in Environmental Fluid Dynamics Code Plus (EFDC+). With this feature, EFDC+ can run across large cluster systems as well as multi-core desktop computers with greater speed than before. With this additional capability comes increased complexity in specifying the computer configuration that can provide optimal performance for each application. This paper provides a description of EFDC+'s domain decomposition approach and the computer settings to achieve faster run times. Extracting maximum performance for each user's application will require the user to follow some guidelines when preparing and executing an EFDC+ run with MPI.

To give users a sense of this new EFDC+ capability, the run time performance of a large model of the Chesapeake Bay is analyzed. Prior to this version of EFDC+, only multi-threading using Open Multi-Processing (OMP) was available to improve performance. For this Chesapeake Bay model, the previous version of EFDC+ could only achieve a speedup of 2.5, whereas the current hybrid OMP/MPI version of EFDC+ with the domain decomposition approach achieved a speedup of nearly 17 with 32 compute cores. Using a cluster configured on Amazon Web Services (AWS), a strong scaling study was conducted, which resulted in a speedup of nearly 25 while using 96 compute cores. Overall, the domain decomposition approach using MPI provides the ability to run across a cluster and offers substantially improved performance compared to previous versions of EFDC+.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AWS	Amazon Web Services
CPU	Central Processing Unit
DSI	DSI, LLC
EE	EFDC_Explorer
EEMS	EFDC_Explorer Modeling System
EFA	Elastic Fabric Adapter
EFDC+	Environmental Fluid Dynamics Code Plus
Gbps	Gigabits per second
LPT	Lagrangian Particle Tracking
MPI	Message Passing Interface
NOAA	National Oceanic and Atmospheric Administration
OMP	Open Multi-Processing
PDE	Partial Differential Equation
SPMD	Single Program Multiple Data
USGS	United States Geological Survey

1 INTRODUCTION

Grid based hydrodynamic models can take significant amounts of time to complete model simulations. DSI, LLC (DSI) implemented OMP in EFDC+ to reduce model run times by parallelizing several tasks. DSI released this capability in EFDC+8.0 (DSI 2017). EFDC+ with OMP increased the model speed by two to three times. However, this speed up may not be enough for large models with thousands or even millions of cells that represent large areas or complex natural phenomena.

To reduce model run times further, DSI is pleased to introduce a domain decomposition approach for EFDC+. This is accomplished by splitting up the domain of a model into several smaller ones, referred to as subdomains (Fainchtein 2014). Each of these subdomains executes like a traditional EFDC+ run, except that each subdomain exchanges information with its neighboring subdomain at each time step (Gropp et al. 2014). This information exchange is accomplished by leveraging Intel's version of MPI to communicate between domains. To better understand the details of the MPI implementation, an overview of computer architecture and relevant terms is provided.

To assess the performance of the domain decomposition approach, a spatially refined model of Chesapeake Bay is analyzed. This model contains nearly one million computational cells, and simulates typical hydrodynamic processes as well as temperature and salinity changes. A comparison is made between the run times achieved with the domain decomposition approach and the previous version of EFDC+, which only used OMP. Additionally, since the domain decomposition approach can scale across a cluster, a strong scaling study is conducted, where the model is run on up to 96 compute cores.

2 PARALLEL COMPUTING

The essence of EFDC+'s domain decomposition approach is the use of parallel computing to solve chunks of a problem simultaneously on multiple computer processors, instead of solving the problem in sequential steps on a single processor. The most important reason for developing an application to run in parallel is that there are limits on how fast a single processor can efficiently run due to limitation in heat dissipation. Another issue is that a given simulation may be so large it does not fit within the memory of a single computer. A program that runs in parallel can alleviate these issues by splitting up the problem into smaller chunks, each with a smaller memory footprint, thereby allowing the simulation to run as several smaller simulations on multiple processors. With the rapid decrease in processor costs, several modest processors connected may be cheaper than the newest, top-of-the-line single processor.

2.1 Hardware Terminology

From a user's point of view, it is important to have an understanding of commonly used parallel computing terms and distinctions to correctly leverage the available hardware. The basic hierarchy of a modern computer system is provided in Figure 2.1 and the terms represented in the figure are defined below.

- Cluster: Several computers (nodes) connected together by a local network, thus enabling them to behave jointly like a single unified machine.
- Node: Each of the computers in a cluster is referred to as a node; a node may contain multiple Central Processing Unit (CPU)s.
- CPU: The primary component responsible for carrying out instructions; today, a CPU is likely to contain several cores.
- Core: An individual processor within a CPU.
- Process: An instance of a program with a unique memory space.
- Thread: An execution unit within a process (note: a process may contain multiple threads).
- Network (Interconnect): A local high speed network (which may also be referred to as an interconnect) that allows communication to occur between nodes.
- Disk: Refers to a magnetic or solid state medium to store persistent data.

This overview of computer architecture is limited but should provide sufficient insight and consistency to understand the performance metrics and implementation changes made to EFDC+ described later on.

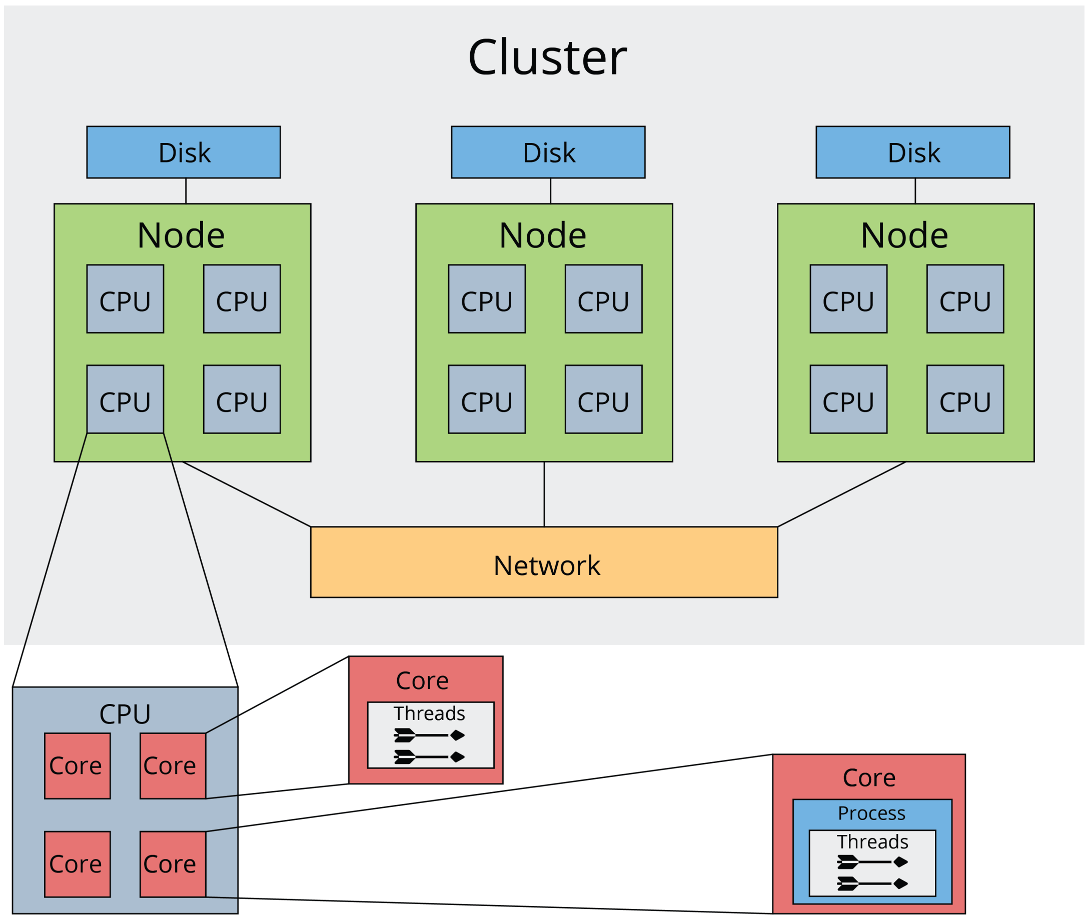


Figure 2.1: Overview of modern computer hierarchy.

2.2 MPI vs. OMP

When selecting tools to parallelize a code base, the two most widely used tools are MPI and OMP. Each of these can accomplish similar goals but have important differences. On the one hand, MPI is a library specification containing standardized subroutines and functions for handling communication (Gropp et al. 2014), whereas on the othr hand OMP is a share memory model and an add-on to a compiler. Functionally, MPI is a message passing model, primarily concerned with message passing between processes (Figure 2.2). The programmer must specify this communication explicitly and carefully within the code, which typically requires significant changes to the code base. Processes are executed in parallel and, as discussed in Section 2.1, have their own address space. This translates to each process having access to its own private data that other processes cannot implicitly access.

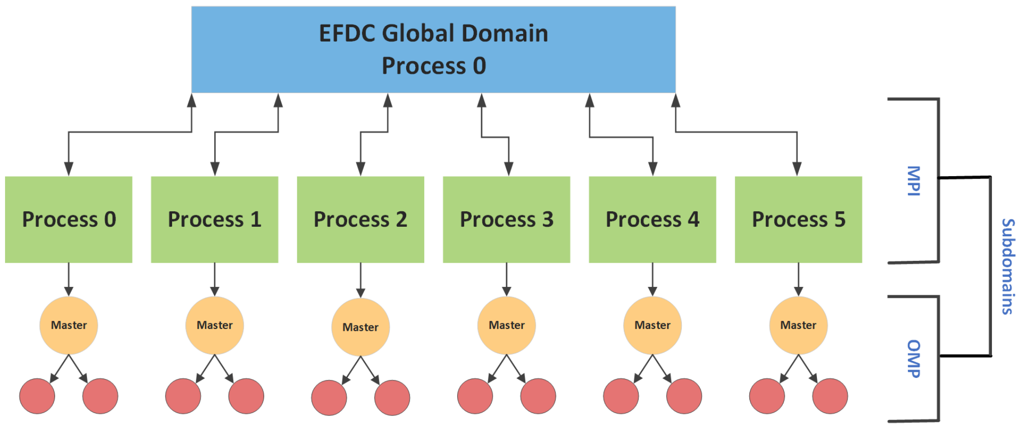


Figure 2.2: Overview of EFDC+'s Hybrid MPI/OMP domain decomposition.

In OMP, In-line directives to the code are added by the programmer to specify when to spawn multiple threads and use them on a section of code. It operates by spawning multiple threads in a fork-join model (Figure 2.3) to accomplish parallel tasks. The creation of threads is simpler from the programmer's point of view, since the only responsibility of the programmer is to specify which sections of the code should have multiple threads running on them. Internal specifications of OMP decide the best way to parallelize a selected portion of code. This makes adoption of OMP into a codebase orders of magnitude easier than using MPI. An additional difference is that, because OMP follows a shared memory model of parallel programming, all threads have access to the memory on a given CPU. Thus, a single process (or each subdomain of an MPI application) can use some or all of the cores within a single computer.

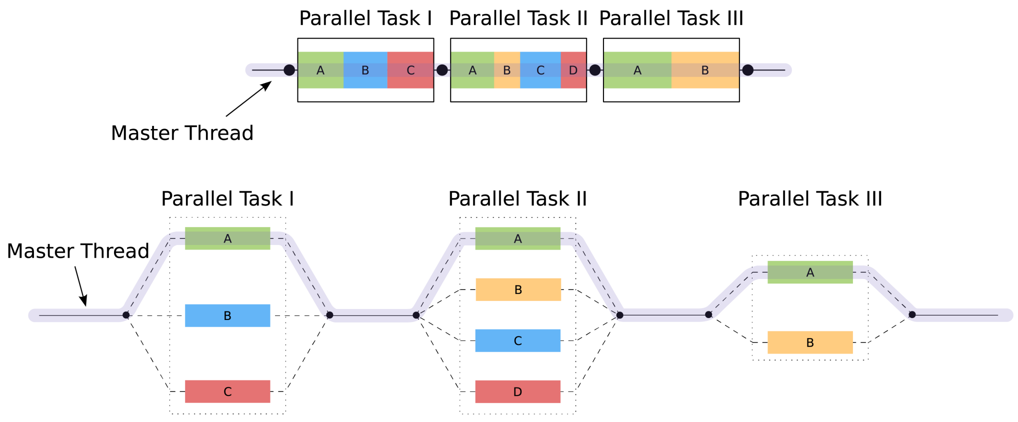


Figure 2.3: Illustration of the Fork-Join Parallel Programming Model.[1]

A code that uses both OMP and MPI is referred to as a hybrid parallel code. The new version of EFDC+ is such a hybrid code (see Figure 2.2). Using a blend of OMP and MPI makes it possible to obtain optimal performance on a given hardware platform. It should be noted, however, that both the OMP's fork-join process and the MPI communication can take significant overhead. Each application will require some testing to determine the optimal blend of the number of MPI subdomains and the number of OMP cores to be assigned. Another key to efficiently using MPI for EFDC+ is to balance the work between each subdomain.

In developing the current MPI/OMP hybrid approach for EFDC+, the work conducted by O'Donncha and James (O'Donncha et al. 2016) was studied in detail. This code provided insights into how MPI could be most effectively implemented in EFDC+. In the O'Donncha study OMP was not found to provide significant speed gains when used in conjunction with MPI. However, as will be shown in the examples below, the current hybrid approach resulted in substantial reductions in simulation time.

[1] `https://commons.wikimedia.org/wiki/File:Fork_join.svg`

3 DOMAIN DECOMPOSITION

The primary purpose of EFDC+ is to solve the governing Partial Differential Equation (PDE)s for fluid flow and constituent transport. This is accomplished by solving these PDEs on a discretized grid representing the geometry of the problem (DSI 2020b). For systems with a large domain or a high degree of spatial grid refinement, the grid may become quite large, requiring significant computation and memory usage, all leading to increases in the run time of a model. To alleviate this problem and take advantage of high core-count computers or high performance clusters, the domain decomposition method has been implemented in EFDC+. Domain decomposition refers to a broad set of techniques to solve PDEs based on splitting the spatial domain of a problem into smaller subdomains (Chan and Mathew 1994). The same governing equations are solved on a smaller grid, referred to as a subdomain, by separate computer cores. Naturally, solving a smaller problem reduces the memory requirements and the solution time on each subdomain. However, the solutions on subdomains are not isolated from one another. For instance, in hydrodynamic and sediment transport modeling, the water and constituents must be allowed to flow across the entire domain, so information must be communicated at the edges of subdomain boundaries. Therefore, an important aspect of implementing domain decomposition into any codebase is a means of exchanging information between subdomains. Within EFDC+, MPI is used for this purpose. Details of how model values are communicated across subdomain boundaries and other implementation details will be discussed in the following section.

There are two components of an efficient domain decomposition; code efficiency and even division of work among subdomains. Code efficiency for domain decomposition is primarily about minimizing the communication required between subdomains, because inter-subdomain communication is usually a run time bottleneck and entails an overhead cost. The second component is dividing the computational work among subdomains as evenly as possible. A balance must be struck between the benefits of dividing the main application domain into smaller and smaller subdomains and the communication costs that increase as the number of subdomains increases.

3.1 Load Balancing

An important consideration to minimize run times is to ensure that the computational loads on each process are as similar as possible. This is critical because the processes have to sync up periodically and exchange data. This synchronization procedure implies that the entire algorithm is constrained by the slowest process. Even if 9 out of 10 processes run quickly, performances gains are lost when they all have to wait for the slowest process to synchronize for collective communication. Thus, care should be taken by the user to verify that the domain is decomposed in such a way that the load is well balanced.

Balancing work with EFDC+ can be difficult for some applications (e.g., when using EFDC+'s ability to activate/deactivate individual cell processes; using Lagrangian Particle Tracking (LPT); and/or varying the number of vertical layers from cell to cell). Fortunately for the user of EFDC+, EFDC_Explorer Modeling System (EEMS) provides tools to assist the user in balancing the work when applying domain decomposition to an application. EEMS guidance is provided in the EEMS user guide (DSI 2020a).

During the model development process, typically a model is run numerous times over a representative time period. At the end of each run, EFDC+ reports the total run time and CPU time used for each sub-process. These run times are stored in the TIME.LOG file. The user should review these timing results and adjust the domain decomposition using EEMS to test if improvements can be made. As experience is gained with the model performance and characteristics of the model, run times can be reduced.

3.2 Subdomain Communication

Creating an efficient code using the domain decomposition approach requires careful consideration of how information is to be exchanged across subdomain boundaries. In order for energy and mass to be exchanged between adjacent subdomains, two additional rows and/or columns of cells belonging to the respective adjacent process are added to the border of each subdomain. These cells are referred to in the literature as ghost or halo cells. Without ghost cells, if the adjacent cell needed for this calculation resided on a different subdomain, then the derivative could not be calculated and the solution would fail. Values for the ghost cells are communicated (i.e., values exchanged between subdomains) at the end of each time step for use at next time step.

To illustrate this ghost cell and subdomain approach, a simplified example of the transition from a global grid to subdomains including ghost cells will be presented. The common term used to describe this domain decomposition approach used by EFDC+ is Single Program Multiple Data (SPMD). Functionally, this means the EFDC+ model grid (the data) is broken up into subdomains (multiple data groups), each of which contains a portion of the entire grid. The major changes to the algorithmic structure is the addition of the ghost cells and the communication of information between subdomains. Figure 3.1 shows a simplified grid representing a global domain in an EFDC+ model. The domain decomposition occurs in the two-dimensional horizontal plane only, which are denoted by the I and J axes in the figures. Figure 3.1 is then broken down (i.e., decomposed) into subdomains, as represented in Figure 3.2. Figure 3.3 illustrates the addition of the two rows/columns of ghost cells to each subdomain and the direction of communication from the source domain to adjacent subdomain ghost cells (cell colors indicate the source subdomain).

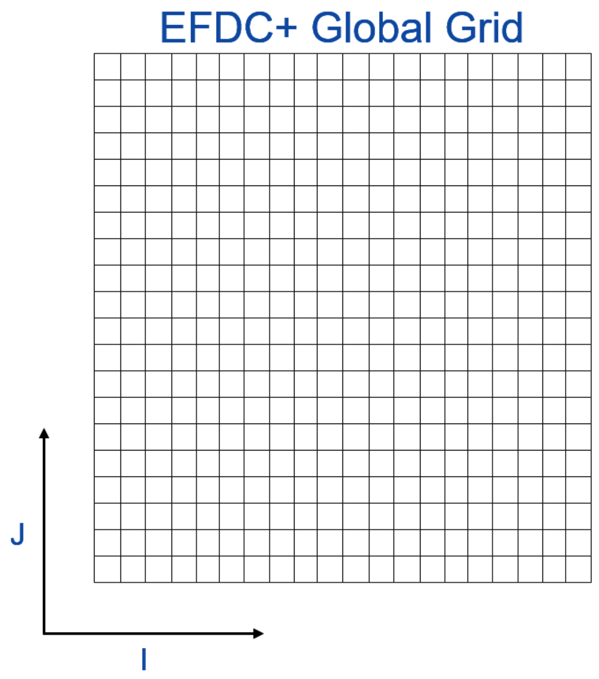


Figure 3.1: Sample EFDC+ grid.

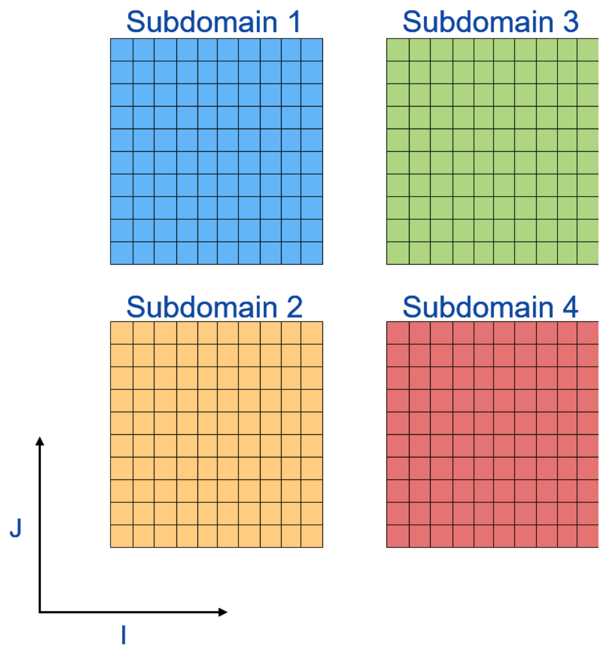


Figure 3.2: Decomposition of the global EFDC+ grid.

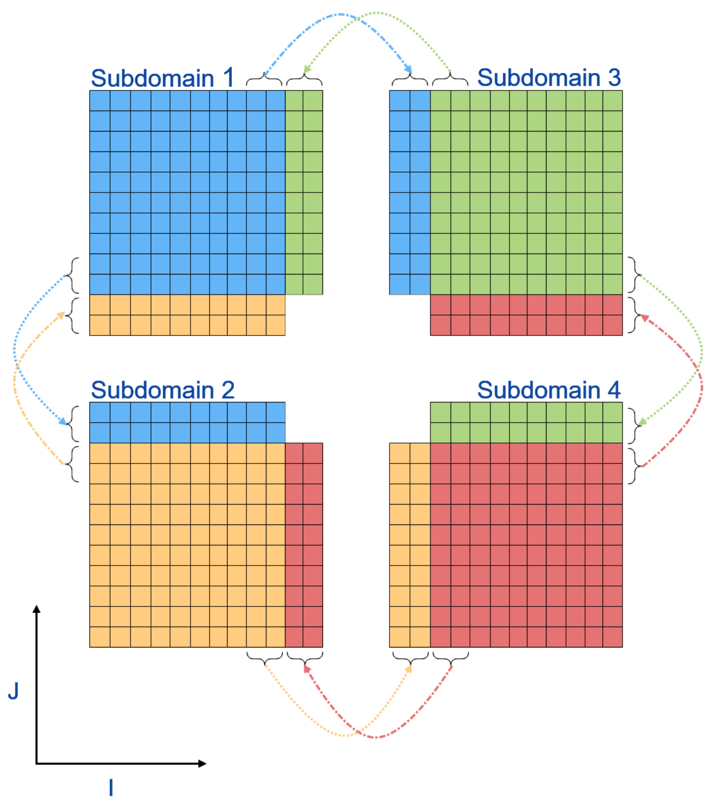


Figure 3.3: Subdomain grids with ghost cells.

In addition to communication of boundary/ghost cell information, several other aspects of communication within the EFDC+ calculation scheme must be considered. Of importance is how to preserve the quality of the iterative conjugate gradient method to solve the external pressure solution. Communication is required during iterations of the conjugate gradient method so that the total residual from all subdomains can be determined, and thus the correct global solution can be obtained on all processes. Similarly, when dynamic time stepping is used, each subdomain will compute its own respective time step. Depending on the dynamics of the system, these time steps between subdomains can be substantially different. To ensure the time steps for each subdomain are equivalent, the minimum time step of all the locally computed values is determined and communicated back to all subdomains. The global minimum time step is then used within each subdomain.

3.3 Input/Output Files

EFDC+ uses a fixed file naming approach for input files needed (DSI 2020a). The domain decomposition information is contained in the file DECOMP.INP. This file specifies the number of subdomains in the I and the J directions, along with the number of cells along the I and J axes for each subdomain. Due to the potential complexity of a domain and the corresponding EFDC+ cell map, some of these IJ subdomains may not include any active cells (i.e., empty subdomains). Empty domains are ignored by EFDC+.

With implementation of domain decomposition in EFDC+, the need to address spatially dependent input files required a significant change to EFDC+ input file handling. Fundamentally, the decomposition requires mapping of the user-input global grid data to each subdomain grid. This holds for any spatially dependent boundary conditions like inflows, outflows, hydraulic structures, Lagrangian particle release locations, etc. The mapping between the starting "global" domain to the subdomains is handled internally within EFDC+.

Finally, the model results are computed on each subdomain. To use these results, however, the user needs global domain results. Therefore, subdomain solution values are collected, mapped back to the global grid, and written out. The model results contained in the output files generated by EFDC+ are based on the global grid and are outputted in the same format with or without domain decomposition. Thus, the user has the ability to process, view, and analyze the results as soon as the model run finishes.

3.4 General Performance Considerations

The computational performance of any given EFDC+ simulation run is dependent on many factors, such as how the model grid is configured, what physics are simulated, and the available computing hardware. This section provides some guidance on how to make the best use of the domain decomposition algorithm to reduce run times.

As discussed in Section 3.1, computational work, not cell count, is what needs to be evenly divided. Running a simulation that leverages the MPI-based domain decomposition algorithm requires care to extract maximum performance. The fundamental consideration is whether the decomposition of the geometry can be set up so that the time spent computing on each subdomain is much greater than the time spent communicating among all of the subdomains. This issue arises due to the current situation with computer architecture, in that the CPU is very fast, but everything else is comparatively slow, especially any communication of data between processes across a network.

Generally, larger domains (in terms of total number of active cells) will get the biggest benefit from decomposing the domain. Conversely, a small domain will not benefit from decomposing the domain into several very small subdomains. This is largely because the cost of communicating the information across ghost cells and collecting the global solution will be much greater than the time it takes to perform calculations on each very small subdomain.

The frequency of snapshots written out to the global EEMS linkage files or NetCDF files will also impact the total elapsed run time. Each output snapshot requires EFDC+ to gather the required variables from all of the subdomains and write the files. The communication time needed to gather and process the information into a global framework is proportional to the number of subdomains.

EFDC+ has been a multi-threaded application using OMP since 2008. With the addition of MPI, EFDC+ is now an OMP/MPI hybrid code. Without OMP, the main run time considerations would be how many subdomains there are and how many cells each subdomain contains. However, certain processes in EFDC+ use OMP very efficiently (e.g., LPT), so the user will need to test a combination of the number of subdomains and the number of threads used for each subdomain to achieve optimal performance.

Another consideration, particularly when running MPI on a single node, is memory usage. Each MPI process has its own address space and allocates memory for its subdomain. If the problem is decomposed such that there are close to the same number of processes as there are physical cores and the EFDC+ model is large, the computer may not have enough memory for all of the processes. This will result in significant slowdown due to an increase in page faults (i.e., the program looks for a value in memory but it is no longer there).

4 EXAMPLE APPLICATION AND BENCHMARK TESTS

To assess the performance of the MPI implementation for EFDC+, a detailed three dimensional hydrodynamic model of the Chesapeake Bay was created. This model was constructed to contain a high degree of spatial resolution to allow for performance testing.

4.1 Chesapeake Bay Model

The Chesapeake Bay watershed is located on the east coast of the United States and drains over 166,530 km^2 spread across five states. The bay, shown in Figure 4.1, has a surface area of more than 14,700 km^2. Details of Chesapeake Bay, its drainage area, and other models, are readily available online[1].

4.1.1 Data Acquisition

Bathymetry data in the Chesapeake Bay were collected from the Chesapeake Community Modeling Program Website. Flows and water surface elevation data in the Chesapeake Bay were collected from various sources, including the CH3D Model (Sheng 1989), the United States Geological Survey (USGS), and the National Oceanic and Atmospheric Administration (NOAA). These datasets helped develop the conceptual model for the bay, and were then used to define the hydrodynamic model boundary conditions.

4.1.2 Model Development

A hybrid cartesian-curvilinear grid for Chesapeake Bay was built using CVLGrid (DSI 2020a), with orthogonal cells used in the bay and curvilinear cells in the ocean region. The hydrodynamic model used for the benchmark test had approximately 204,000 curvilinear cells with an average cell dimension of 256m $\times$ 270m. The model was then set to have four vertical layers, which resulted in approximately 800,000 computational cells. The model has 13 inflow boundaries and one open boundary, as shown in Figure 4.2. The daily inflow boundaries of the James River, York River, Rappahannock River, Potomac River, Susquehanna River and C&D canal were inherited from the CH3D Model. The other daily inflows from the Patapsco River, Chester River, Choptank River, Naticoke River, Pocomoke River, Big Elk Creek, and Dragon Swamp were obtained from USGS stations. In some cases, these stations combine several flow stations on their tributaries, such as the inflow at the Patapsco River boundary, which is the sum of flows from the stations

[1]Chesapeake Community Modeling Program Website

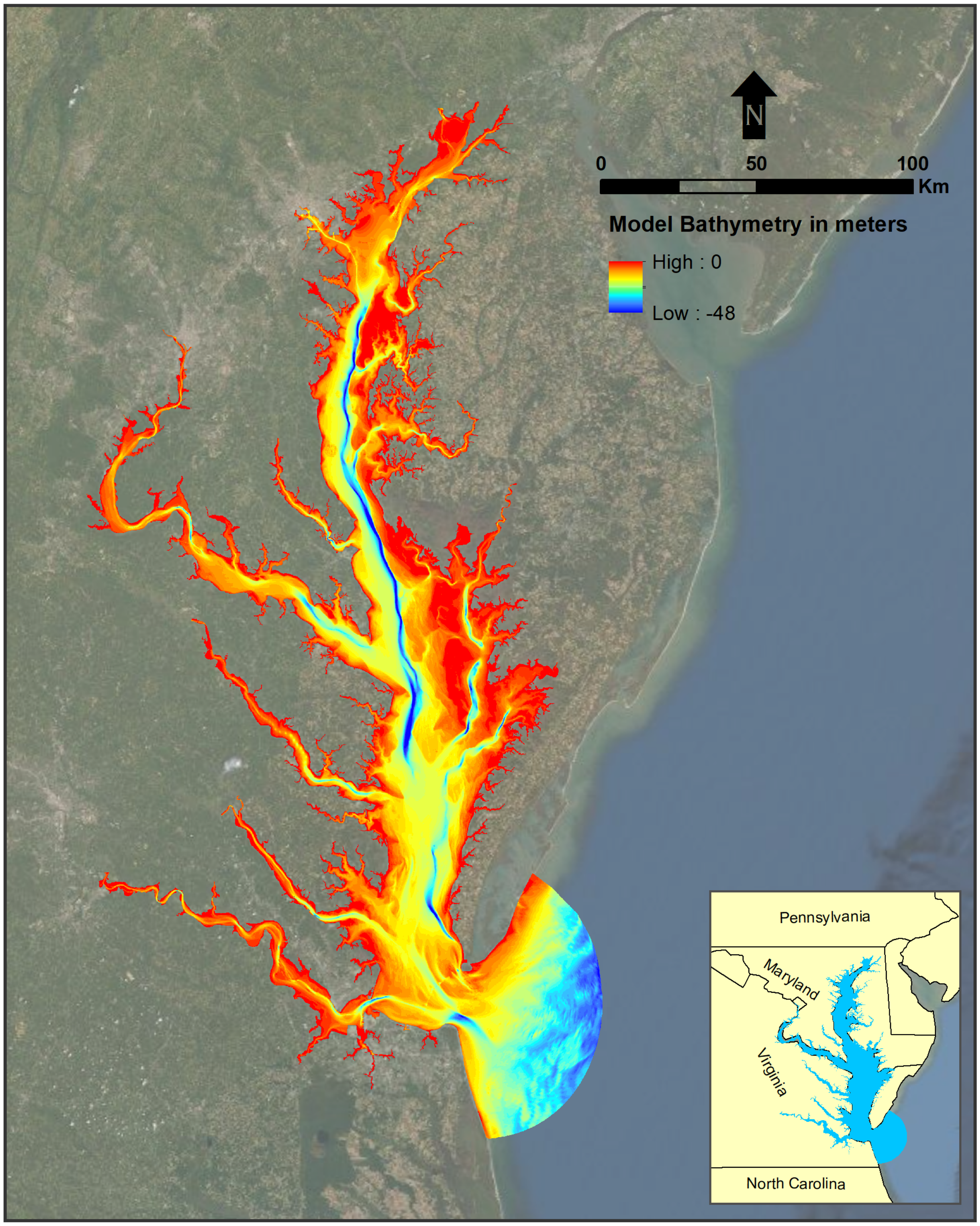


Figure 4.1: Chesapeake Bay elevation grid.

USGS 01589352, USGS 01589035 and USGS 01589100. The major inflow boundaries are the Susquehanna River, Potomac River, James River, Rappahannock River and York River, with average daily flow rates of 1146, 347, 178, 46 and 33 m^3/s, respectively. The daily flow time series for these major rivers series in 2011 are shown in Figure 4.3. Water surface elevation from the Chesapeake Bay Bridge Tunnel station was assigned to the open boundary of the model, and the time series is shown in Figure 4.4.

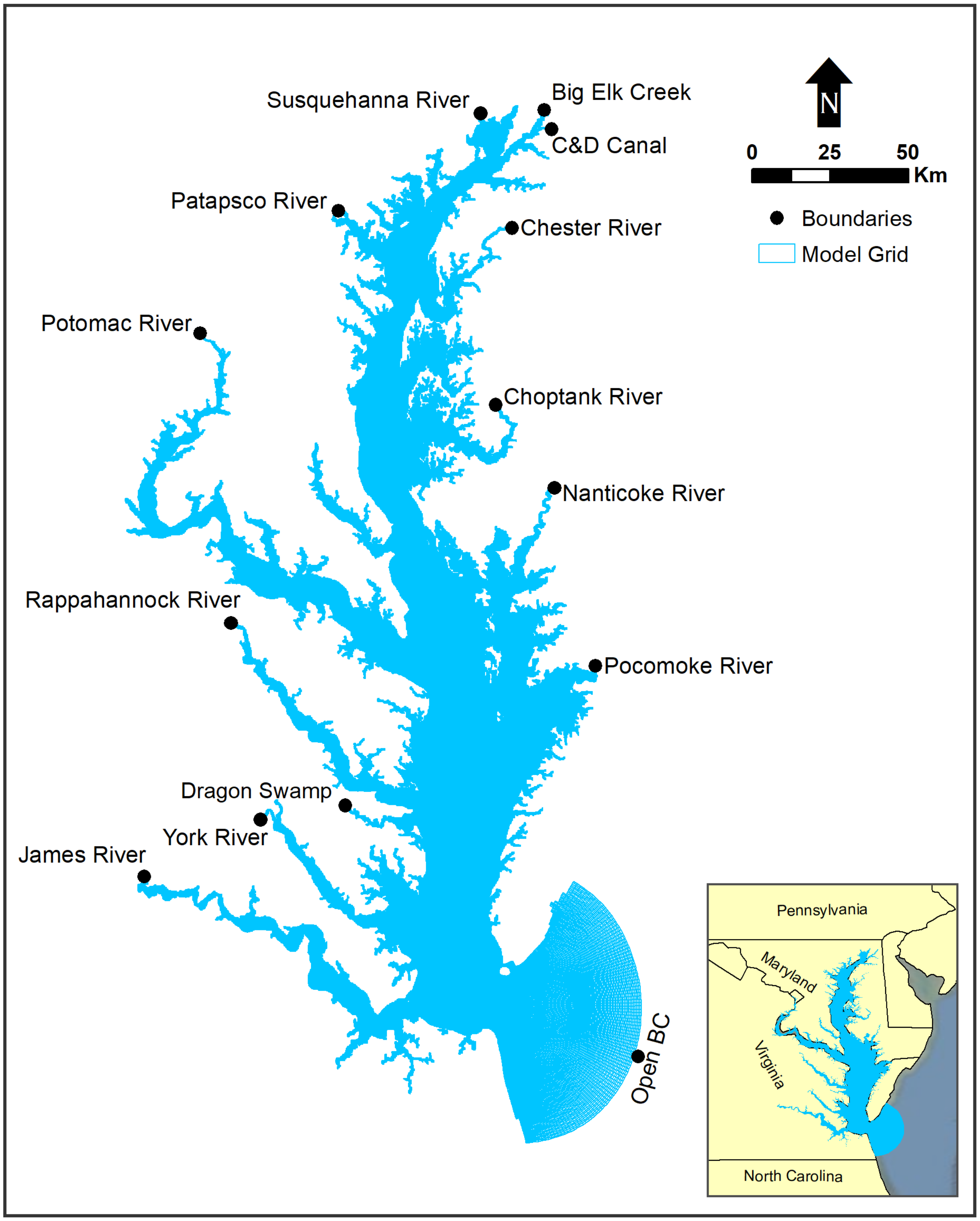


Figure 4.2: Flow and open boundaries used for the Chesapeake Bay Model.

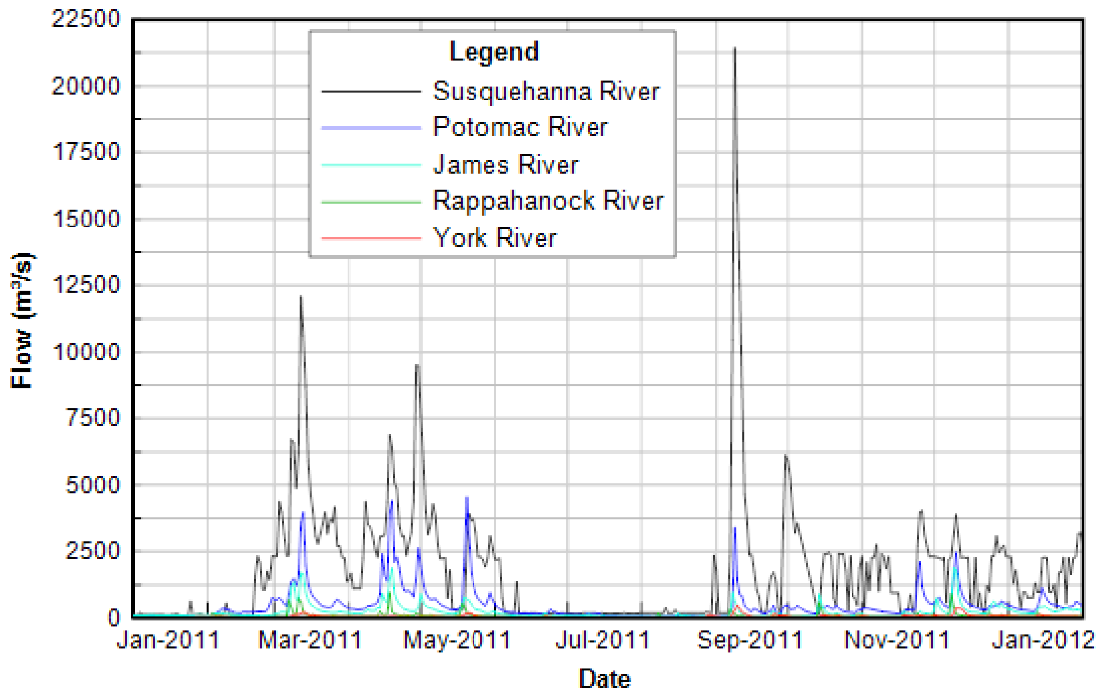


Figure 4.3: Average daily flow time series of the five largest rivers for 2011.

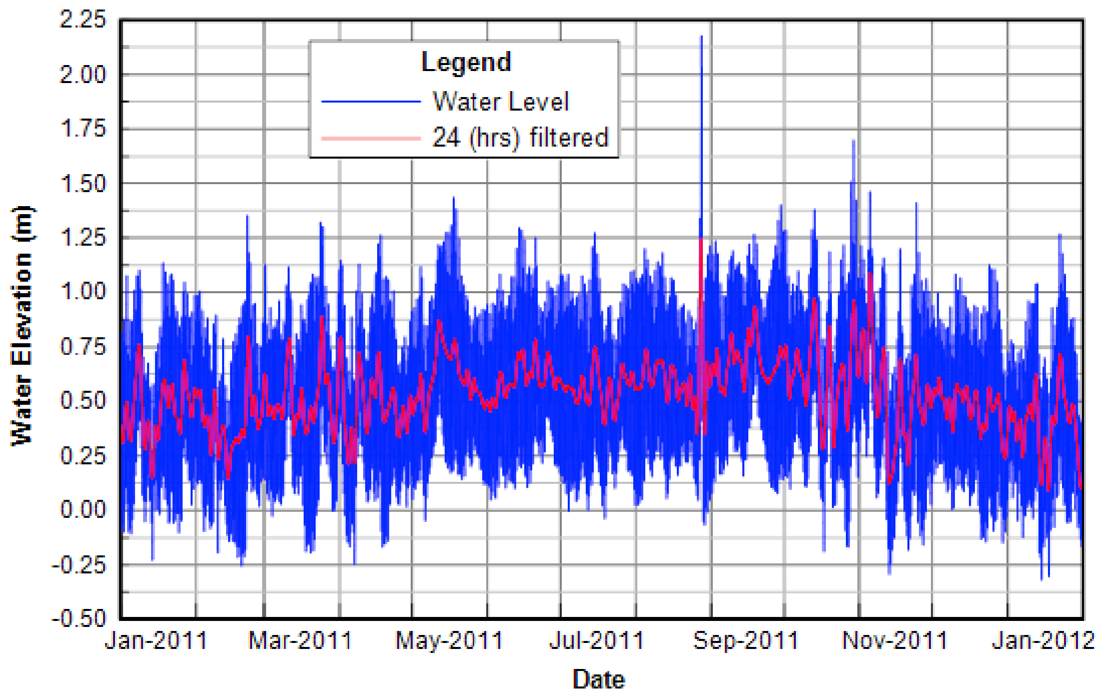


Figure 4.4: Six-minute water surface elevation time series in 2011 from open boundary.

The model was then configured to also simulate salinity and temperature. The 4-day simulation time period was from 2011-07-01 to 2011-07-05. The dynamic time stepping option was employed, with a minimum time step selected as 0.05 seconds and a maximum time step of 10 seconds. The computational performance of the Chesapeake Bay model is discussed in detail in Section 4.2. An example of the model configured with 32 subdomains is shown in Figure 4.5.

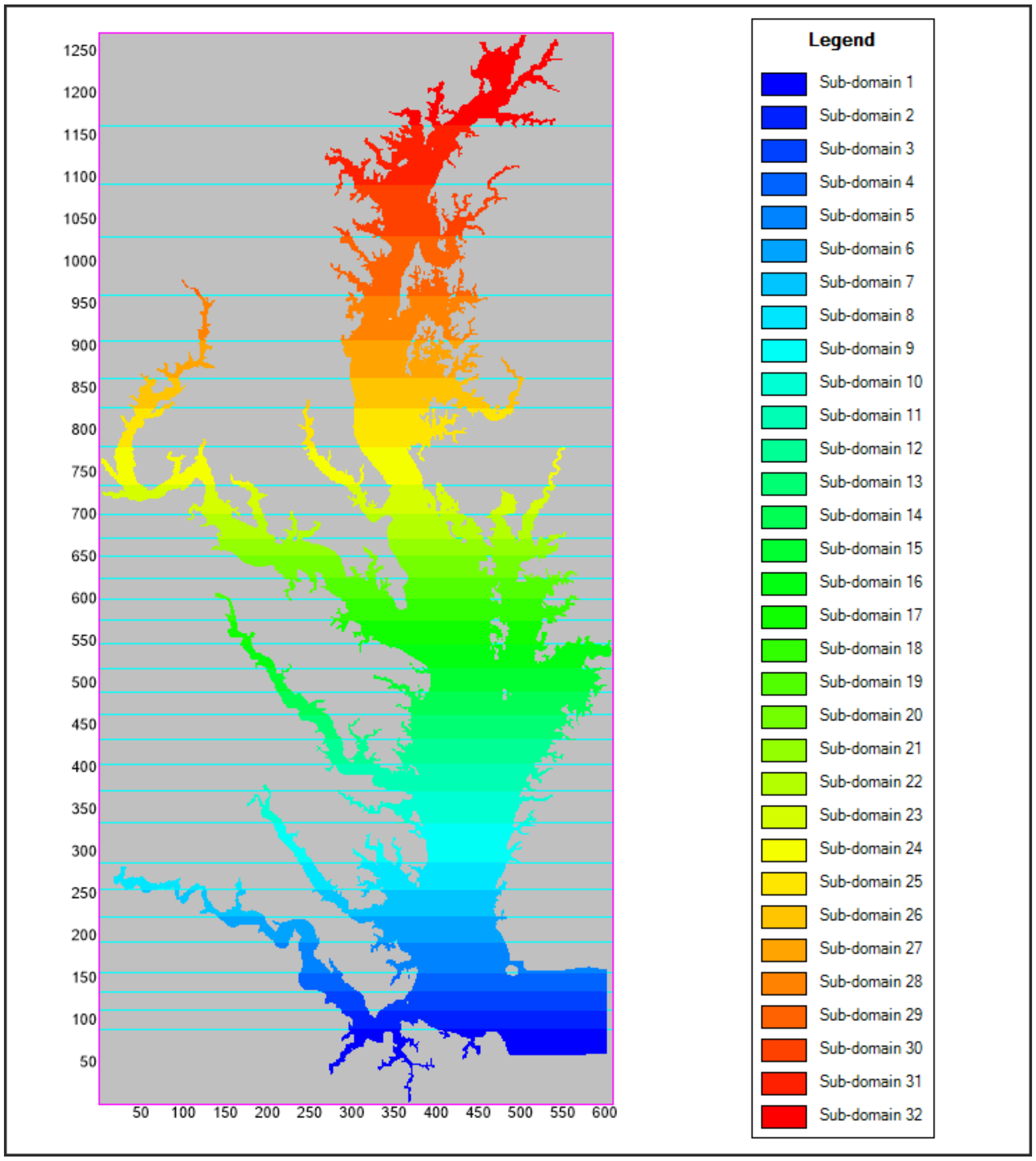


Figure 4.5: Cell Map view of 32 MPI subdomains.

4.1.3 Model Output Comparison

Ouptut time series of water surface elevation and salinity from model runs with one domain, two subdomains and 16 subdomains were compared. Salinity time series were compared at the top layer, depth averaged and bottom layer. These time series were extracted from the same five locations in each model run (Figure 4.6). The comparison indicates that the number of MPI subdomains do not influence the model output.

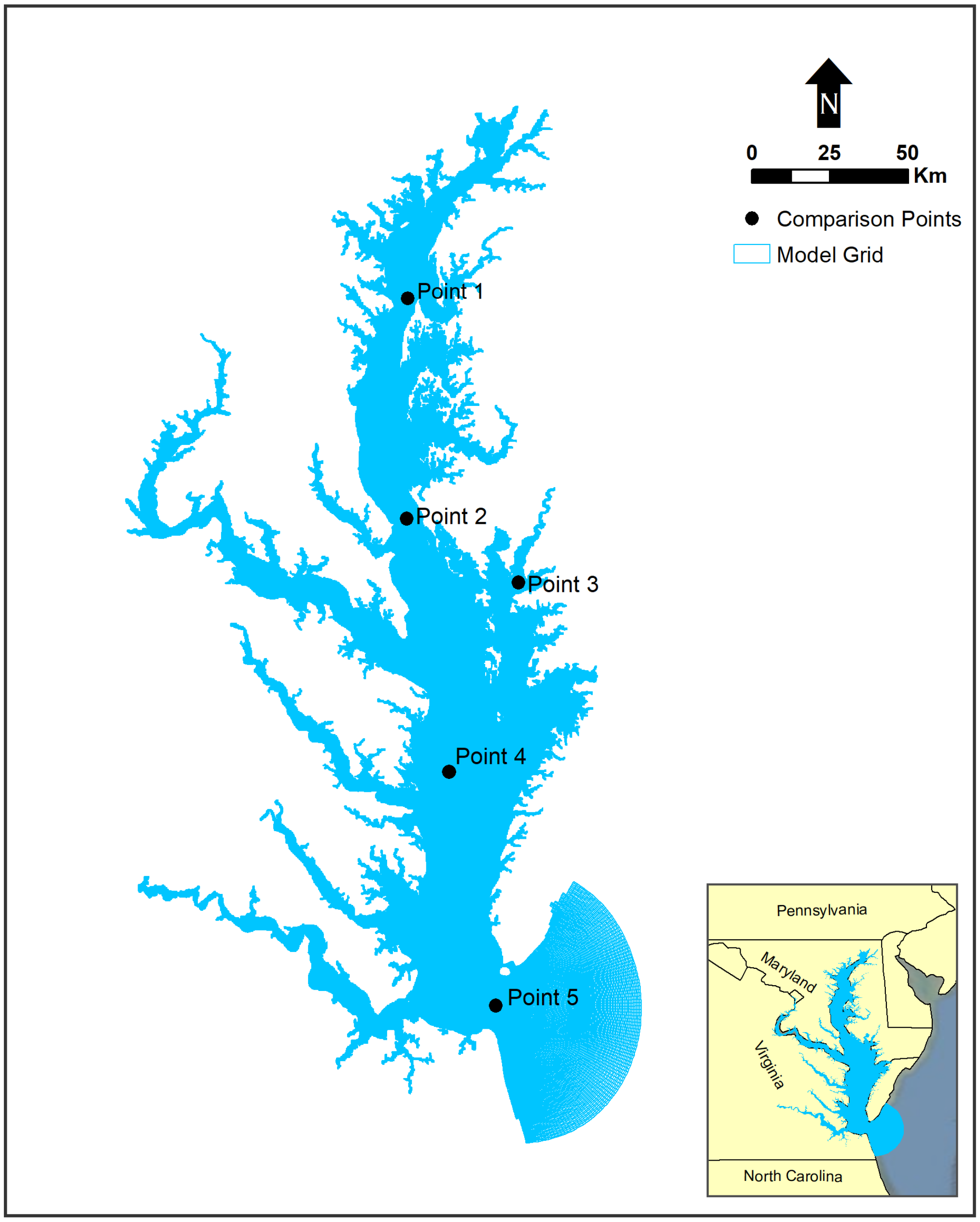


Figure 4.6: Five locations for comparison from model runs.

Water surface elevation comparison from model runs at the locations from Point 1 to Point 5 are summarized in Table 4.1 and the time series plots are shown in Figure 4.7 and Figure 4.8, while salinity comparison at the depth averaged from model runs at the locations from Point 1 to Point 5 are summarized in Table 4.2 and the time series plots are shown in Figure 4.9 and Figure 4.10.

Table 4.1: Water surface elevation (m) statistics at comparison locations.

Location	Model	Min	Median	Max
Point 1	1 Domain	0.47	0.68	1.05
	2 Sub-domain	0.47	0.68	1.05
	16 Sub-domain	0.47	0.68	1.05
Point 2	1 Domain	0.44	0.69	1.07
	2 Sub-domain	0.45	0.69	1.07
	16 Sub-domain	0.45	0.69	1.07
Point 3	1 Domain	0.35	0.70	1.08
	2 Sub-domain	0.36	0.70	1.08
	16 Sub-domain	0.36	0.71	1.08
Point 4	1 Domain	0.41	0.70	1.07
	2 Sub-domain	0.41	0.70	1.07
	16 Sub-domain	0.41	0.70	1.07
Point 5	1 Domain	0.07	0.64	1.08
	2 Sub-domain	0.07	0.65	1.09
	16 Sub-domain	0.07	0.66	1.08

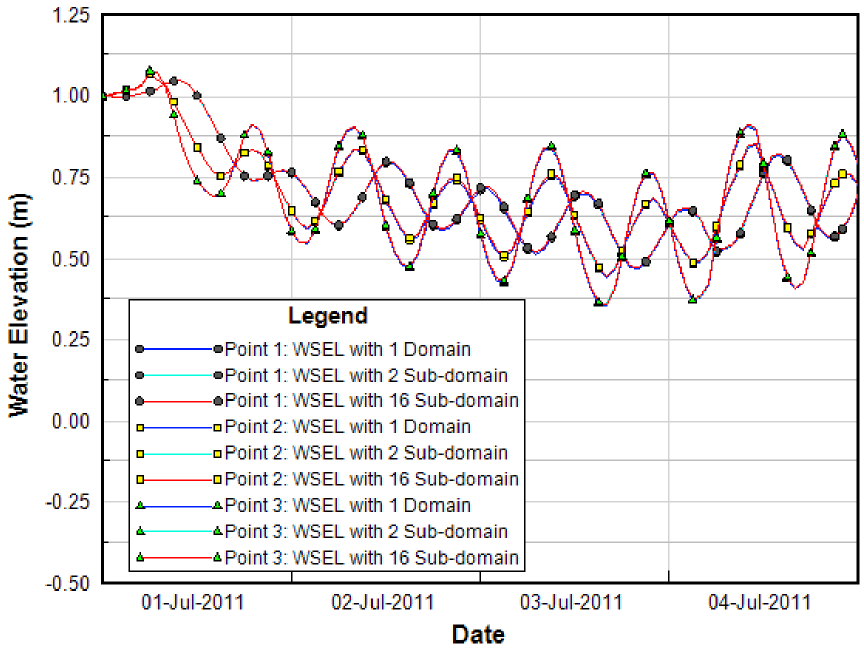


Figure 4.7: Water Surface Elevation comparison from model runs at Point 1, 2 and 3.

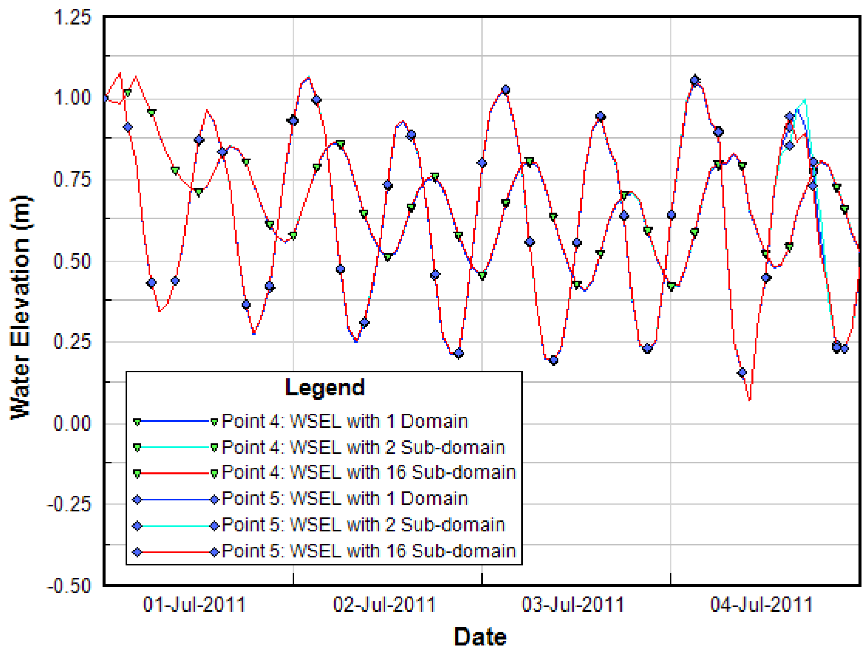


Figure 4.8: Water Surface Elevation comparison from model runs at Point 4 and 5.

Table 4.2: Salinity (ppt) at depth averaged statistics at comparison locations.

Location	Model	Min	Median	Max
Point 1	1 Domain	0.00	0.00	0.00
	2 Sub-domain	0.00	0.00	0.00
	16 Sub-domain	0.00	0.00	0.00
Point 2	1 Domain	0.00	0.64	2.27
	2 Sub-domain	0.00	0.64	2.27
	16 Sub-domain	0.00	0.64	2.27
Point 3	1 Domain	1.28	1.45	2.17
	2 Sub-domain	1.28	1.45	2.17
	16 Sub-domain	1.28	1.45	2.17
Point 4	1 Domain	7.21	7.83	8.64
	2 Sub-domain	7.21	7.84	8.64
	16 Sub-domain	7.21	7.84	8.64
Point 5	1 Domain	15.35	19.10	25.84
	2 Sub-domain	15.49	19.07	25.46
	16 Sub-domain	15.48	19.06	25.47

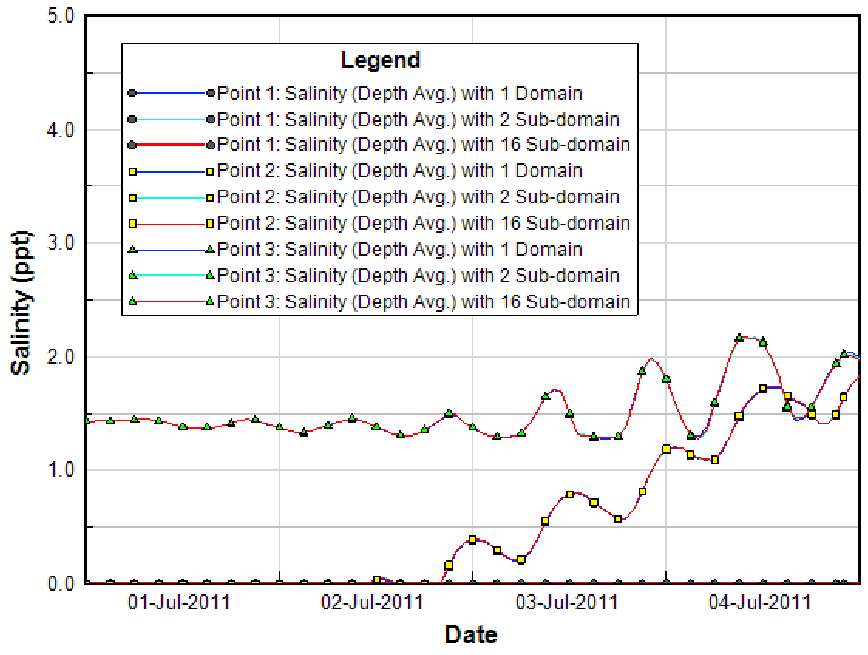


Figure 4.9: Salinity at depth averaged comparison from model runs at Point 1, 2 and 3.

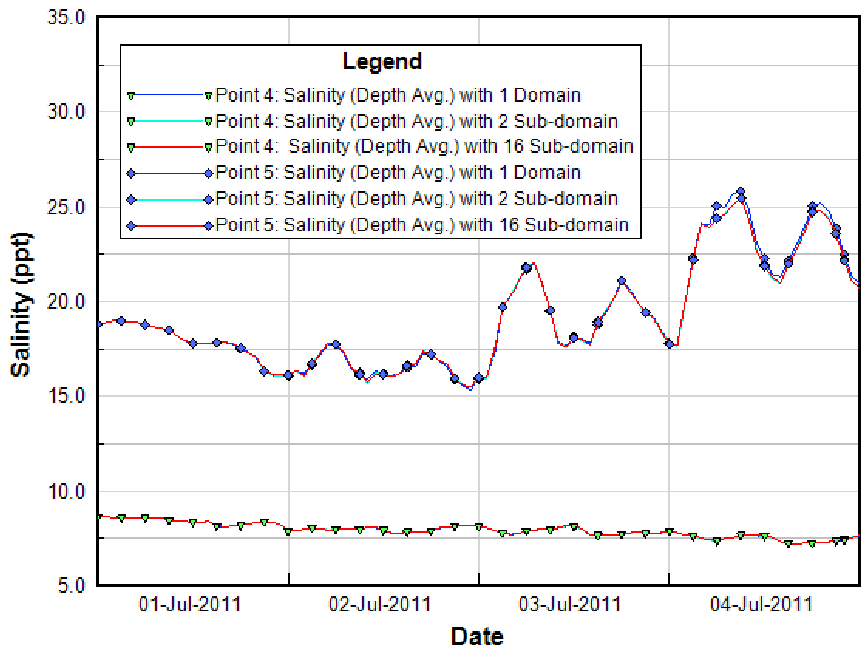


Figure 4.10: Salinity at depth averaged comparison from model runs at Point 4 and 5.

4.2 Sample Application Performance

In this section, we examine the computational performance of the Chesapeake Bay model described in Section 4.1. An overview of the computer configuration that was used for running this model is provided in the following section.

4.2.1 Computing Environment

The only requirement to run an EFDC+ model with MPI implementation is to have multiple cores available on the computer running the model. The configuration of that computer is of importance to understand run times and the maximum attainable performance. Most desktop computers have CPUs with multiple cores, which will allow MPI applications to run on that single computer. However, if it desirable to run a problem on more cores than are available on a single computer, this naturally requires several computers to be connected together. This is accomplished by running EFDC+ on what is commonly referred to as a cluster, as described in Section 2.1. The primary benefit of MPI-based algorithms is they can be run on a cluster; this is in contrast to multi-threaded programs based on something like OMP, which on their own are limited to running on a single computer. This distinction is important, because CPU clock speeds have largely stagnated, in part due to Moore's Law (Theis and Wong 2017). Increased computational performance now primarily comes from having additional cores available in a given computing environment and writing software with parallelization in mind.

To test how many cores could efficiently be used by implementation of domain decomposition, a cluster was configured on AWS using their ParallelCluster service (Services 2020a). This allows a remote (cloud-based) cluster to be configured on demand. For this study, a cluster containing four compute nodes and one head node was set up. Each compute node (single computer) contains a 36-core Intel Xeon Platinum 8000-series CPU. To minimize communication latency between nodes, a high-speed interconnect was set up using the Elastic Fabric Adapter (EFA) interface (Services 2020b), which enables up to 100 Gigabits per second (Gbps) of throughput. A high-speed file system based on FSx for Lustre was used to minimize time spent writing to the disk during a simulation. The description reported here of the cluster used for the simulations includes sufficient detail to allow the user to feasibly recreate the computing environment and the timing results.

4.2.2 Performance Comparison with the Previous Version of EFDC+

An obvious benchmark for performance improvement is the comparison of EFDC+ with an MPI implementation compared to the OMP-only implementation available in previous versions. Prior to this release of EFDC+, only multi-threading was available to reduce model run times. This approach yielded significant benefits, but as mentioned in Section 4.2.1, there are limits to this approach.

To highlight this, the Chesapeake Bay model was run on the same computer with equivalent computational resources using the OMP and MPI implementations of EFDC+. A comparison of the model run times as a function of the number of cores used is shown in Figure 4.11. Of note, these reported times consider the setup of the problem and writing out the full model output to the disk 120 times. This is a typical output frequency, which was chosen to provide a realistic sense of the model performance one could expect using EFDC+.

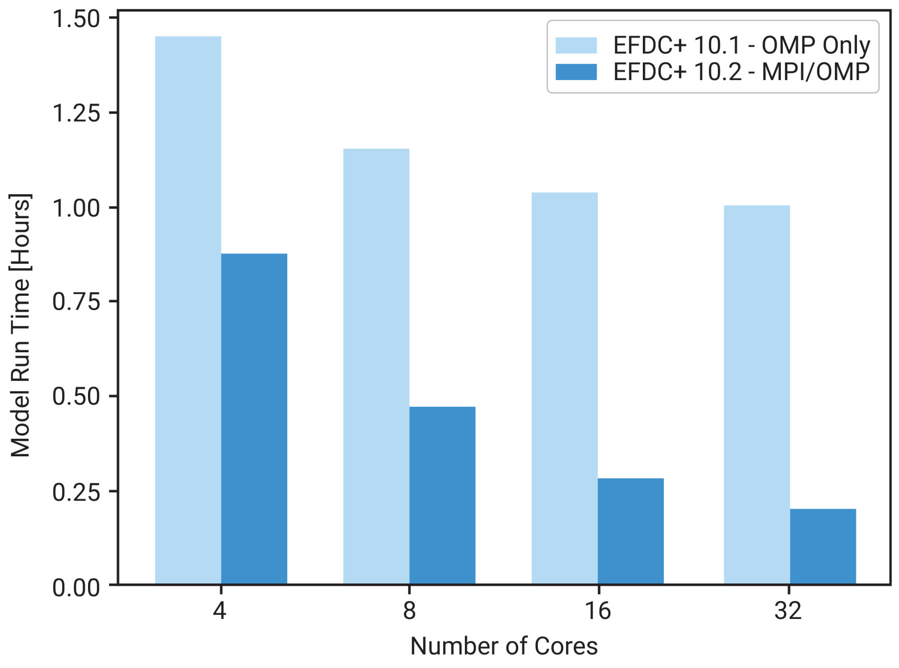


Figure 4.11: Elapsed time for each model shown as a function of computer cores used.

In Figure 4.11, it is clear in all cases that the MPI/OMP version of EFDC+ is faster. An additional observation is that the OMP-only version improves the run times with additional cores, but the improvements are minor, especially beyond 16 cores. Conversely, with the MPI/OMP version, there is still noticeable speedup from 16- to 32 cores. This fact is better highlighted by comparing the improvements in run time as a function of how much faster a given model run is (speedup) compared to running on a single core. Running on a single core is the expected performance of legacy versions of EFDC not developed by DSI.

For the same model runs shown in Figure 4.11, a similar plot is produced in Figure 4.12, except in each case, the speedup is reported. The speedup, S, is defined as

$$S = \frac{T_1}{T_C} \tag{4.1}$$

where T_1 is the model run time spent in a single core and T_C is the time spent by a model run with a C number of cores.

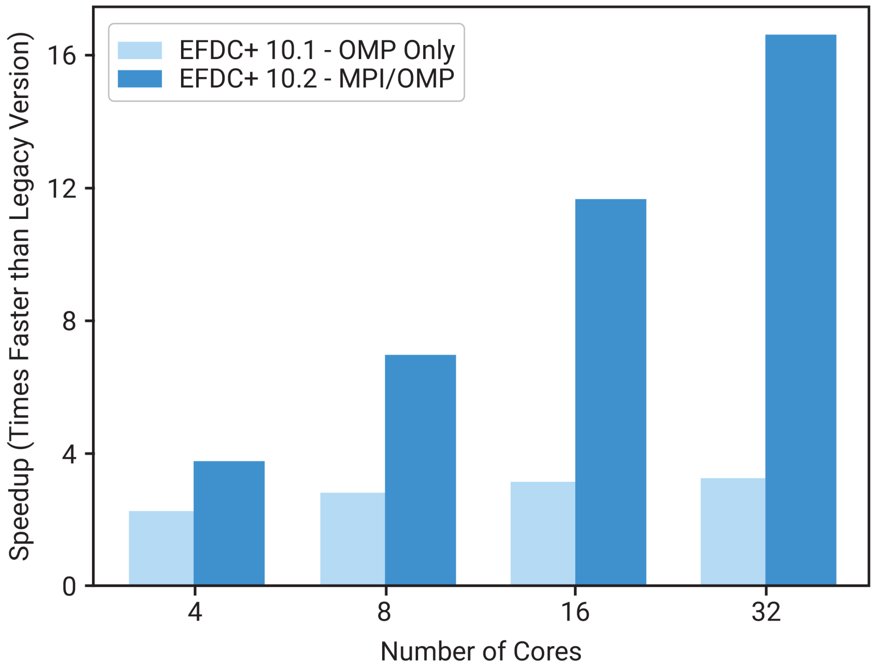


Figure 4.12: Speedup for each model run performed with either OMP or MPI versions of EFDC+.

Upon review of Figure 4.12, it is clear that the OMP-only version achieves a speedup of only about 2.5 times the legacy run times, whereas the MPI/OMP version has a speedup of about 17 times the legacy speeds. The run time improvements with MPI do not increase linearly with the number of cores because of the need for increased communication synchronization between processes, as per Amdahl's Law. Amdahl's Law says that for a large number of cores, C, the speedup does not scale with C but rather to a $1/w$, where w is the fraction of the computer work not run in parallel (Amdahl 2013). This is because the additional time spent communicating is the time not spent calculating and that limits the amount of computer work that can be done in parallel.

Overall, MPI implementation offers a significant improvement in run times compared to OMP-only versions of EFDC+. This is particularly evident for models such as the Chesapeake Bay model, which contain a large number of grid cells and thus represent a good use case for the domain decomposition approach. There are limits to the parallelization efficacy of the domain decomposition approach, however, due primarily to increased communication overhead and unavoidable serialization in parts of the code.

Next, in Section 4.2.3, the trade-offs in run time versus additional parallelization will be explored in greater detail.

4.2.3 Strong Scaling Study

As mentioned, an important consideration when running an MPI-based program is to determine at what point increased parallelization begins to yield diminishing returns. In the interest of providing results and guidelines that will be useful for a typical user, a strong scaling study (which focuses on a problem of fixed size or amount of computational use) was conducted on the Chesapeake Bay model. This is in contrast to weak scaling, where the amount of computational work is increased in proportion to the increased number of cores a program is run on. These weak scaling studies are often shown to highlight the sophistication of a parallel implementation and its scalability to a large number of cores (1000+). Weak scaling studies are valuable to those running on systems with thousands of cores, but obscure what actual performance may be extracted from a given model of fixed size with modest computational resources.

Results of the strong scaling study are provided in Figure 4.13, where the Chesapeake Bay model was run on up to 96 cores. In all of these cases, the subdomain sizing was done using the simple automatic domain decomposition algorithm provided in EFDC_Explorer (EE). The approach taken for decomposing the domains tries to best balance the number of active cells in the requested number of subdomains. Furthermore, at this time, it only considers decomposing the subdomain in either the I or the J direction of the model, but not both. It is likely that further performance could be extracted by judiciously selecting the subdomain sizing. Nevertheless, in the interest of providing reproducible results, we opted not to adjust the subdomain sizing from what EE provides. For the Chesapeake Bay example, using 96 cores (i.e., 96 subdomains) resulted in a speedup factor of about 22.

It is clear that the rate of performance improvement decreases after 64 cores. As mentioned, this is in part due to Ahmdahl's law. It is exacerbated by the communication time required, because the simulations run with more than 32 cores had to run across several nodes of the cluster and therefore were subject to increased communication time across the network. Generally, communication between cores on a given node is quick, since the latency is minimal. Communicating across a network (i.e., between cores on different nodes) is subjected to increased latency, so any calculation requiring communication across a network experiences an increased performance penalty. This is highlighted in Figure 4.14, which shows the fraction of time spent communicating for each simulation shown earlier in Figure 4.13.

In Figure 4.14, a clear jump in communication time occurs when using 48 cores; the communication percentage surges to nearly 20%, compared to about 5% with 32 cores. This is due to the communication penalty taken for communicating across the InfiniBand network.

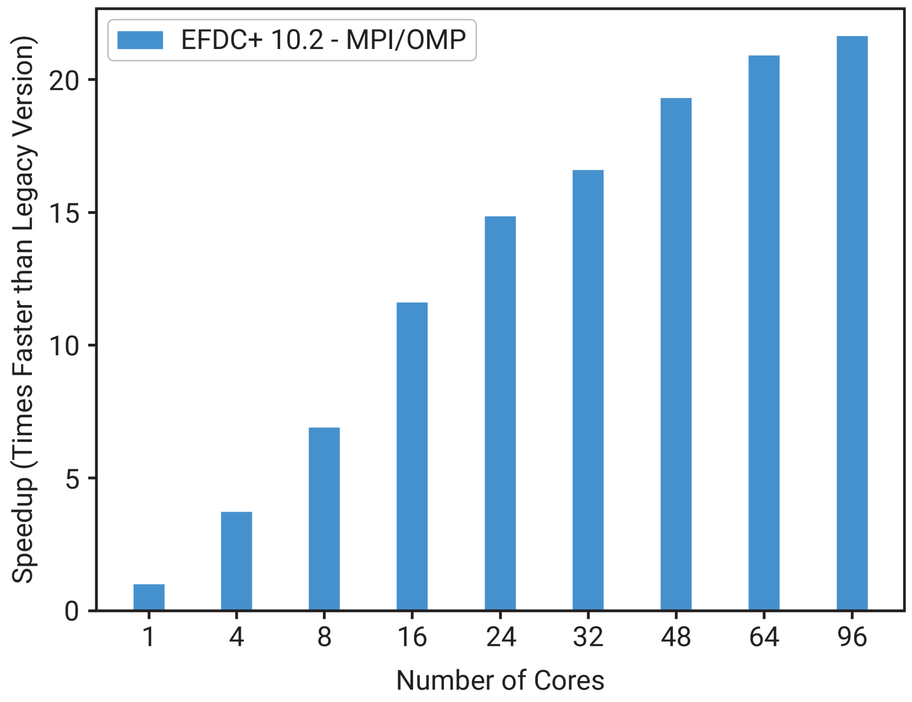


Figure 4.13: Strong scaling study results highlighting the speedup using up to 96 cores.

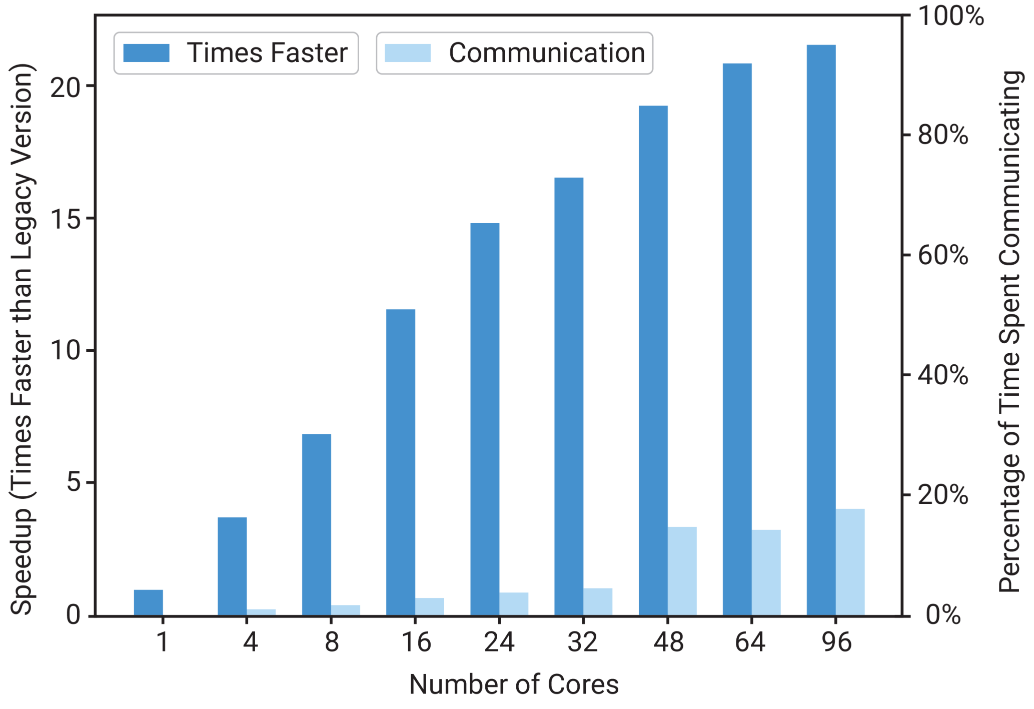


Figure 4.14: Speedup with varying numbers of cores is shown along with the fraction of total computational time spent in communication for each instance.

Apart from decomposing the domain further, another approach to gaining additional performance is to use additional threads on each subdomain. Deciding on the number of threads to use per subdomain is challenging and likely will differ from model to model. In this case, the best performance was achieved by allowing for 2 threads per subdomain, with a total of 48 subdomains. This requires a total of 96 cores to run efficiently. It should be noted that this cluster was configured such that hyperthreading was not allowed. Hyperthreading allows for two threads to run on a single physical core, but in our experience, this has not provided any substantial run time improvements, and in many cases, has degraded performance. Therefore, we allow a single thread to occupy an entire physical core. Several of the simulation speedup results using greater than 48 cores are shown in Figure 4.15. In Figure 4.15, using 48 threads and 2 threads per subdomain provided a speedup of about 24.

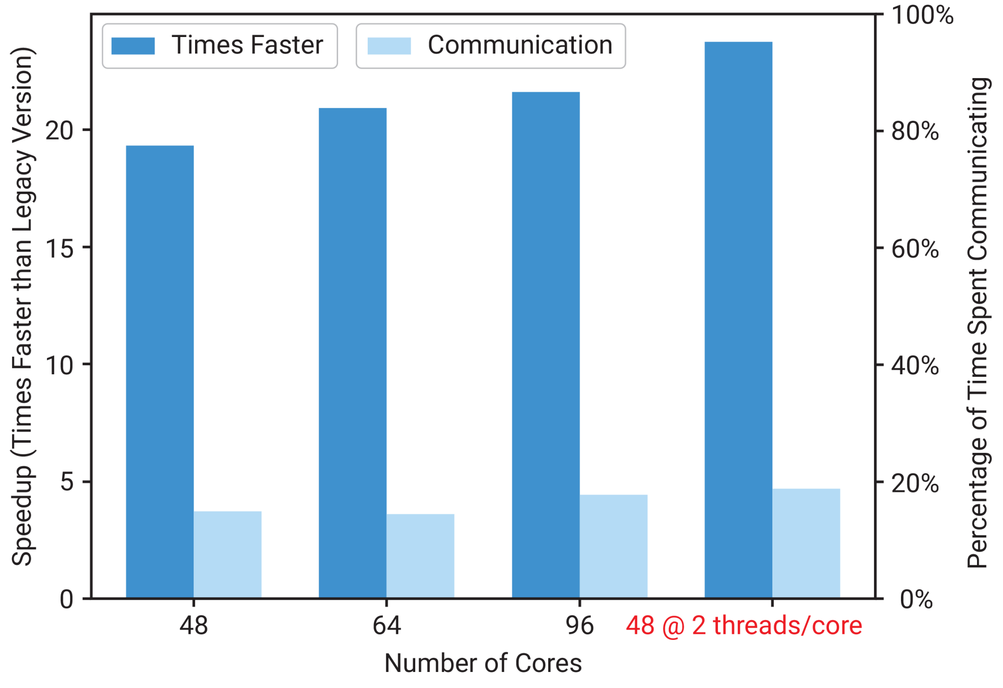


Figure 4.15: In-depth look at maximum core counts and the percentage of time spent communicating in each case.

In this case, the fraction of time spent communicating increased slightly, but the entire speedup compared to using 96 subdomains on the 96 cores was improved. Although it is not clear, we believe the increase in time spent communicating when using additional threads occurs because the MPI implementation may use extra available resources in its internal subroutines to package and unpackage data in preparation for communication routines. In this case, using two threads per subdomain yielded the best compromise. Other numbers of subdomains and/or threads per subdomain were tried, but did not improve speedup.

It is generally difficult to evaluate the performance of a scientific code like EFDC+, due to variations in model setups and available computational resources. The goal of this study was to present a large model with realistic parameters and have all simulations run on a computer system that is publicly available (AWS). The results presented here highlight that speedups of nearly 25 are possible with fewer than 100 cores. As the number of cores increases, the primary limitation on performance is the communication overhead across the network. Therefore, when setting up a cluster system, the choice of bandwidth and latency are the most important factors for achieving optimal computational performance. Another consideration is how many threads should be run per subdomain; in this model, two threads per subdomain provided the best performance. Other models, such as those that simulate many constituents, may do better with additional threads.

5 CONCLUSION

By decomposing an EFDC+ model domain into smaller subdomains, it is possible to run the model across a cluster computer or multi-core desktop. Running these models is simple to do and requires no user pre- or post-processing. The domain decomposition approach implemented in the latest version (10.2) of EFDC+ appears to provide substantial decreases in run time compared to prior versions of EFDC+. A detailed scaling study has shown the limits of the parallelization approach, due primarily to Amdahl's law, but also highlighted that a speedup of nearly 25 is possible on the sample model analyzed. This performance was achieved using a simple decomposition of the model, employing the automated subdomain selection tool available in EE. As described in this paper, there are a variety of considerations to achieve the maximum performance, and additional improvement in performance is likely possible with tuning of the subdomain numbers and sizing. Considerations have been provided as to how this tuning may be conducted to extract maximum performance.

BIBLIOGRAPHY

[1] Gene M Amdahl. “Computer Architecture and Amdahl’s Law”. In: *Computer* 46.12 (2013), pp. 38–46.

[2] Tony F Chan and Tarek P Mathew. “Domain decomposition algorithms”. In: *Acta numerica* 3 (1994), pp. 61–143.

[3] DSI. *EFDC+8.0*. Version 8.0. DSI, LLC, Edmonds, WA, USA., 2017.

[4] DSI. *EFDC_Explorer 10 User Guide*. DSI, LLC, Edmonds, WA, USA., 2020. URL: `https://eemodelingsystem.atlassian.net/wiki/spaces/EK/overview?mode=global` (visited on 04/13/2020).

[5] DSI. *EFDC+ Theory Version 10.2*. 2020. URL: `https://www.eemodelingsystem.com/?ddownload=12205` (visited on 04/13/2020).

[6] R. Fainchtein. *Intermediate MPI:Domain Decomposition*. MIT Press, 2014. URL: `https://edoras.sdsu.edu/~mthomas/docs/mpi/nasa.tutorial/mpi2.pdf`.

[7] William Gropp et al. *Using advanced MPI: Modern features of the message-passing interface*. MIT Press, 2014.

[8] Fearghal O’Donncha et al. “On the Efficiency of Executing Hydro-environmental Models on Cloud”. In: *Procedia Engineering* 154 (2016), pp. 199–206. ISSN: 18777058. DOI: `10.1016/j.proeng.2016.07.447`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S1877705816318367` (visited on 06/29/2020).

[9] Amazon Web Services. *AWS Parallelcluster User Guide*. 2020. URL: `https://docs.aws.amazon.com/parallelcluster/index.html` (visited on 04/13/2020).

[10] Amazon Web Services. *Elastic Fabric Adapter*. 2020. URL: `https://aws.amazon.com/hpc/efa/` (visited on 04/13/2020).

[11] Y. Peter Sheng. “Evolution of A Three-Dimensional Curvilinear-Grid Hydrodynamic Model for Estuaries, Lakes and Coastal Waters: CH3D”. In: *Estuarine and Coastal Modeling*. Estuarine and Coastal Modeling. Newport, Rhode Island, United States: American Society of Civil Engineers, Nov. 15, 1989. ISBN: 978-0-87262-758-1 (ISBN-13).

[12] Thomas N Theis and H-S Philip Wong. “The End of Moore’s Law: A New Beginning for Information Technology”. In: *Computing in Science & Engineering* 19.2 (2017), pp. 41–50.