Optimizing a Stencil-Based Application for Earthquake Modeling on Modern InfiniBand Clusters

S. Potluri[1], P. Lai[1], K. Tomko[2], Y. Cui[3]
M. Tatineni[3], K. Schulz[4], W. Barth[4], A. Majumdar[3] and D. K. Panda[1]
[1] Department of Computer Science and Engineering, The Ohio State University
{potluri,laipi,panda}@cse.ohio-state.edu
[2] Ohio Supercomputer Center
ktomko@osc.edu
[3] San Diego Supercomputer Center
{mahidhar,yfcui,majumdar}@sdsc.edu
[4] Texas Advanced Computing Center
{bbarth,karl}@tacc.utexas.edu

Abstract

Efficient communication design is essential for HEC applications to scale to thousands and 10s of thousands of processors on current and upcoming systems. Traditionally, non-blocking two-sided semantics from MPI-1 have been used to achieve latency hiding when optimizing communication for HEC applications. The more recently proposed one-sided semantics of MPI-2 can also be used to achieve latency hiding but have not been evaluated at large scale. For this study, we investigate application level optimizations for latency hiding in conjunction with one-sided and non-blocking two-sided implementations of near-neighbor communication in a stencil-based earthquake-induced ground motion simulation code, AWM-OLSEN from the Southern California Earthquake Center. We implement reduced synchronization and communication-computation overlap, two key mechanisms for latency hiding. We use Post-Wait/Start-Complete calls or Fence calls in the one-sided versions and ISend/IRecv/Waitall calls in the two-sided version. Our experiments on TACCś Ranger system show that these mechanisms can provide similar benefits in both the one-sided and two-sided models and demonstrate use of one-sided communication at large scale. We see a 7% improvement in the overall application performance on 4,096 cores.

1 Introduction

Efficient communication design is essential for HEC applications to scale to thousands and tens of thousands of processors on current and upcoming systems. The earthquake modeling code (AWM-Olsen) discussed in this paper is frequently run on 32,000 cores of the TACC Ranger system. Any inefficiency in its implementation or in the implementation of similar HEC codes will result in the consumption of tens of thousands of CPU hours which could be better spend on additional science runs. AWM-Olsen, like most commonly used HEC codes, uses the Message

Passing Interface (MPI) for managing parallel processes and communications among them. For stencil-based codes, the near-neighbor exchanges are commonly implemented with MPI-1 point-to-point semantics. These calls are traditionally optimized for latency hiding using non-blocking two-sided semantics from MPI-1. The more recently proposed one-sided semantics of MPI-2 can also be used to achieve latency hiding but adoption of these semantics in scientific applications has been slow. Due to this slow adoption, there have been very few studies on the performance of MPI applications which use one-sided communications, particularly at large scale.

In this paper we use the highly parallel AWM-Olsen ground motion simulation as a platform to explore the following questions in optimizing the near-neighbor communication pattern in stencil-based codes.

- How much latency hiding can we achieve in non-blocking two-sided communication designs?
- What are the design alternatives to consider when converting a two-sided based code to use one-sided operations?

In exploring the one-sided approach, some additional questions arise.

- What kind of window creation techniques represent best-practice for stencil-based applications?
- What is the impact of different synchronization models in MPI-2 semantics on the overall application performance?

To address these questions, we develop multiple non-blocking implementations using the asynchrous two-sided and the active synchronization-based one-sided communication mechanisms made available in MPI-1 and MPI-2 respectively. We compare our one-sided Fence-based and Post/Wait -Start/Complete-base version with equivalently restructured ISend/IRecv/Waitall-based versions. These designs are presented in-depth to illustrate how an MPI application written with two-sided semantics can be converted to use one-sided semantics. The proposed designs have been evaluated on the TACC Ranger system. Our experiments show that the proposed two mechanisms provide similar benefits to our application, with a 7% decrease in run-time on 4,096 cores. Additionally we demonstrate that one-sided communications with either fence-based or post-wait/start-complete synchronization can be effectively used at large scale for a common communication pattern in HEC software.

The remainder of this paper is organized as follows. Section 2 gives a brief review of the petascale AWM-Olsen application and provides an overview of the one-sided operations provided in MPI-2. In section 3 we present several approaches for reducing synchronization and achieving overlap for the near-neighbor communications in AWM with both asynchronous two-sided and MPI-2 one-sided versions of the sub-grid boundary exchange. In Section 4, we present the detailed analysis of experimental performance. We briefly introduce the related work in Section 5, and present the conclusions and future work in section 6.

2 Background

2.1 AWM Application Overview

AWM-Olsen is a community model [10, 15, 3, 5, 4] used by researchers at the Southern California Earthquake Center (SCEC) for wave propagation simulations, dynamic fault rupture studies, physics-based seismic hazard analysis, and improvement of the structural models. The AWM-Olsen code has been scaled to tens of thousands of processor cores of Ranger at TACC, Kraken at NICS, and BlueGene machines at IBM and ANL. Some of the most detailed simulations to date of earthquakes along the San Andreas fault were carried out using this code, including the well-known TeraShake, SCEC ShakeOut simulations. The ShakeOut simulations, for example, modeled an M7.8 earthquake with $600 \times 300 \times 80$-km3 domain at 100-m resolution for up to four minutes. The most demanding simulation of a wall-to-wall rupture scenario required 32 billion volume elements and 86,000 time steps, ran for 20 hours on 32K processor cores of TACC Ranger, and generated 7 terabytes of surface output and checkpoints.

AWM-Olsen solves the 3D velocity-stress wave equation explicitly by a staggered-grid FD method, fourth-order accurate in space and 2nd-order accurate in time. The code includes a coarse-grained implementation of the memory variables for a constant-Q solid and Q relations validated against data. The code uses Perfectly Matched Layers to implement absorbing boundary conditions on the sides and bottom of the grid, and a zero-stress free surface boundary condition at the top. The 3D volume representing the ground area to be modeled is decomposed into 3D sub-grids. Each processor is responsible for performing stress and velocity calculations for its portion of the grid, as well as applying boundary conditions at the external edges of the volume. Ghost cells, comprising a two-cell-thick padding layer, manage the most recently updated wavefield parameters exchanged from the edge of the neighboring sub-grids, see Figure 2 for a 2D schematic of these cells. The Ghost cell update presents a promising scenario to use one-sided communication for accelerated performance. Recent optimization of the 2-sided Ghost cell update from blocking to asynchronous send and receives along with removing some collective calls greatly improved the application's performance on large core counts.

The main loop of the AWM Olsen has been outlined in Figure 1. The velocity values are updated for the interior and the boundary of the volume. Then velocity values are exchanged with processors containing the neighboring sub-grids in the directions of north, south, east, west, up and down. This is followed by stress calculations and updates which are done in a similar manner.

2.2 MPI One-sided Semantics

In the MPI one-sided communication model [12, 9, 8, 16], each process defines a memory region, which is called *window*, in its local address space for other processes to access. Data transfer happens through communication calls: MPI_Put copies data from the caller's (origin's) memory to the target's memory; MPI_Get transfers data from the target's memory to the caller's (origin's) memory; and MPI_Accumulate atomically updates the target's memory with the result of an operation on data from origin's and target's memory. In these operations, all parameters required for the transfer are specified at the origin and thus intervention of the target in not required. However, completion of the operations requires explicit synchronization calls. The MPI one-sided model defines two modes for synchronization: active and passive. Active synchronization involves both the origin and the target processes and has either point-to-point semantics

MAIN LOOP IN AWM-OLSEN

do i = timestep0, timestepN

Compute Velocities

Swap Velocity data with neighboring sub-grids

Compute Stresses

Swap Stress data with neighboring sub-grids

enddo

SWAP VELOCITY DATA

North and South Exchange

s2n(u1,north-mpirank, south-mpirank) *! recv from south, send to north*

n2s(u1, south-mpirank, north-mpirank) *! send to south, recv from north*

. . . repeat for velocity components v1,w1

East and West Exchange

w2e(u1,west-mpirank, east-mpirank) *! recv from west, send to east*

e2w(u1, east-mpirank, west-mpirank) *! send to west, recv from east*

. . . repeat for velocity components v1,w1

Up and Down Exchange

. . .

S2N

Copy 2 planes of data from variable to sendbuffer *!north face excluding ghost cells*

MPI_Isend(sendbuffer, north-mpirank)

MPI_Irecv(recvbuffer, south-mpirank)

MPI_Waitall()

Copy 2 planes of data from recvbuffer to variable *! south ghost cells*

Figure 1: AWM-Olsen Application Pseudo-Code

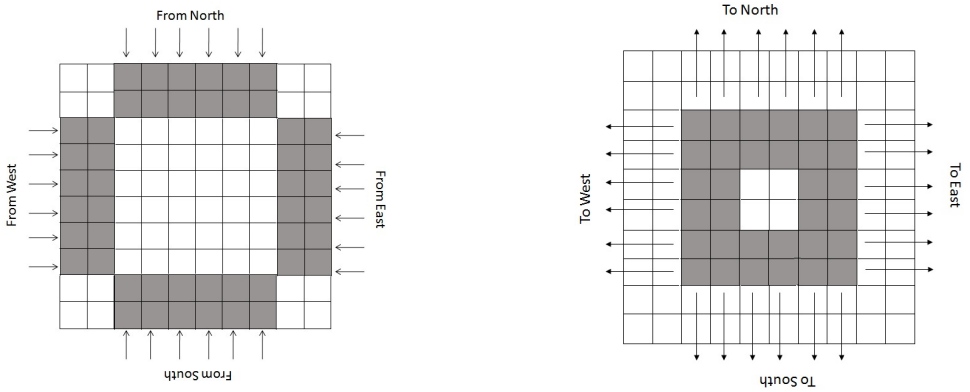

Figure 2: Data transfer patterns in AWM-Olsen

or collective semantics. MPI_Win_fence is an example of active synchronization with collective semantics which requires the participation of all processes of the communicator. MPI_Win_post-MPI_Win_wait/MPI_Win_start-MPI_Win_complete provide a point-to-point mode of active synchronization which can coordinate among a subset of processes in the communicator. Passive synchronization involves locking and unlocking of remote windows requiring participation of only the origin process. Both shared and exclusive lock semantics are provided. Multiple communication operations (to distinct locations in a window) can be issued between processes before they synchronize to amortize the overhead of synchronization. The communication and synchronization operations provided in the MPI one-sided model are summarized in Figure 3.

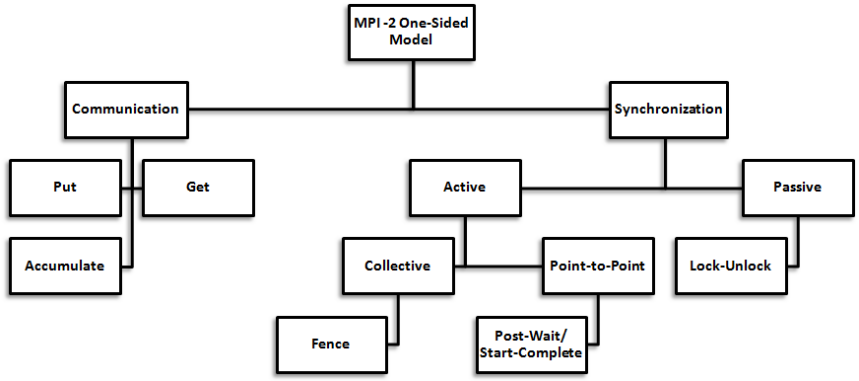

Figure 3: An Overview of MPI-2 Communication Model

3 Proposed Designs

3.1 Design and Implementation using ISend/IRecv

Blocking and non-blocking two-sided communication semantics have been part of the original MPI-1 standard [11] and are quite popular with application scientists. With the option to initiate multiple transfers concurrently, non-blocking semantics provide a better potential for reduced synchronization and communication-computation overlap when compared to blocking semantics, and are less prone to deadlock. The AWM-Olsen application was originally [10, 15] written using blocking semantics. Recently, Cui et. al. [5, 4] have enhanced the application by optimizing TeraShake initialization and I/O handling. This has helped the application to scale to several thousands of cores. This version uses non-blocking semantics for communication.

However, it does not take complete advantage of the non-blocking semantics to reduce synchronization. It naively replaces every MPI_Send and MPI_Recv with MPI_Isend and MPI_Irecv, respectively, and waits for completion by calling an MPI_Wait after each transfer. As discussed in Section 2, in the AWM-Olsen application, every process exchanges data with its nieghbours in all directions and each of these exchanges involved multiple data transfers. In our design, we initiate all of the communication in one direction simultaneously and then wait for the completions. As shown in Figure 4 all of the MPI_Isend and MPI_Irecv are first issued, then only one MPI_Waitall is issued to check all of the completions. It is to be noted that the send or receiver buffers must be defined globally, as the buffers should not be freed until the call to MPI_Waitall

completes. This approach is expected to provide better asynchronous progress. We refer to it as "Async-reduce-sync".

Generally, computation and communication overlap is closely related to the asynchronous progress. The velocity and stress information exchanged in the AWM-Olsen application has several sub-components. For example, velocity has three components, u, v and w. Each of these components corresponds to a data grid. The computation of one sub-component is independent of others which provides an opportunity for overlap. To exploit this characteristic, we reorganize the communication pattern and the main loop is restructured as shown in Figure 5. Instead of issuing all of the transfers in one direction one after another, we group the transfers in all directions related to one component. At the end, the waitall function is called. By doing this, we can overlap the computation of one component with the exchange of another component. We believe this implementation can further hide the communication latency and improve the overall performance.

VELOCITY EXCHANGE
North and South Exchange
    s2n(u1,north-mpirank, south-mpirank) *! recv from south, send to north*
    n2s(u1, south-mpirank, north-mpirank) *! send to south, recv from north*
    . . . repeat for velocity components v1,w1
    wait_onedirection()
    s2nfill(u1, recvbuffer, south-mpirank)
    n2sfill(u1, recvbuffer, north-mpirank)
    . . . repeat for velocity components v1,w1
East and West, Up and Down Exchanges
    . . .

S2N
    Copy 2 planes of data from variable to sendbuffer *!north face excluding ghost cells*
    MPI_Isend(sendbuffer, north-mpirank)
    MPI_Irecv(recvbuffer, south-mpirank)

WAIT_ONEDIRECTION
    MPI_Waitall(list of receive requests)

S2NFILL
    Copy 2 planes of data from recvbuffer to variable *! south ghost cells*

Figure 4: Velocity Exchange with Reduced Synchronization using Non-blocking Two-Sided Semantics

```
MAIN LOOP
do i = timestep0, timestepN
   VELOCITY COMPUTATION AND EXCHANGE
    Compute velocity component u
    Start exchange of velocity component u with neighboring sub-grids
    Compute velocity component v
    Complete exchange of velocity component u
    Start exchange of velocity component v with neighboring sub-grids
    Compute velocity component w
    Complete exchange of velocity component v
    Start exchange of velocity component w with neighboring sub-grids
    Complete exchange of velocity component w
   STRESS COMPUTATION AND EXCHANGE
    ...
enddo
```

Figure 5: Overlapped Velocity Exchange in Main Loop of AWM-Olsen

3.2 Design and Implementation using One-sided Semantics

3.2.1 Window Design for One-sided Implementations

In the MPI-2 one-sided model the window is a contiguous region of memory each process exposes for other processes to read and write data. Organization of these windows not only impacts their creation overhead but also effects the efficiency of the associated communication. In the AWM-Olsen application, data exchange between processes happen in multiple dimensions including north-south, east-west and up-down. While the source and destination array sections in the east-west dimension are contiguous, the array sections for ghost cells in the other dimensions are non-contiguous, considering the column-major allocation in FORTRAN. To include all the target ghost cells, the window will have to span the complete 3D data grid, and there are multiple such grids involved, one for each sub-component exchanged. In MPI implementations that exploit the RDMA feature of InfiniBand, window creation involves pinning the memory and registering it with the NIC. The registration costs increase as the window size increases and there is a limit on the total amount of memory that can be registered. Also the larger the memory pinned the lesser is the memory available for the computation part of the application.

To overcome these limitations, we use intermediate staging buffers as windows. After each synchronization operation, the data is copied from the staging buffers to the actual location in the 3D data grid. Though this involves an extra data copy, it reduces the cost of creating larger windows. This also gives us the flexibility to issue multiple data transfers between two processes though the final destination memory regions are overlapping, thus amortizing the synchronization overhead.

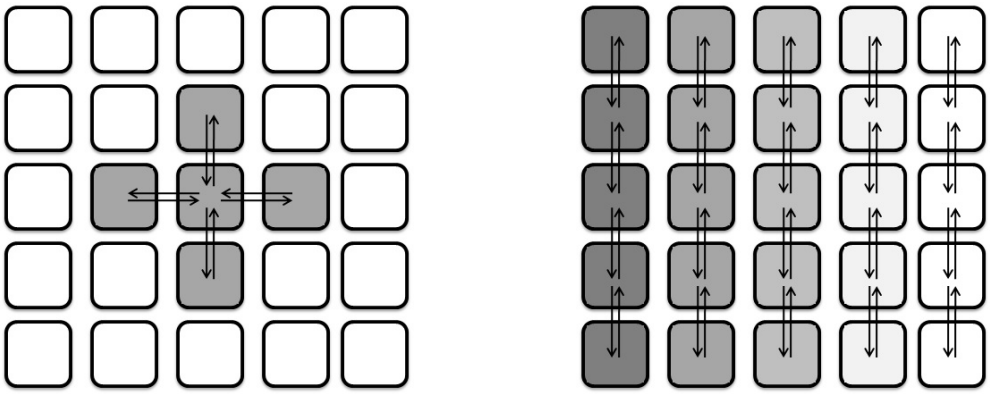

Figure 6: Grouping of process based on the communication dependencies in AWM-Olsen a) Nearest-neighbor pattern b) Row-dependency pattern

3.2.2 Design and Implementation using Fence

From an application programmer's perspective, FENCE [12, 6] is the simplest form of synchronization that the MPI-2 standard provides. FENCE is a collective call and is intended for applications which alternate between global computation phases and global communication phases. All the processes call a FENCE to start an access epoch and each process can issue multiple Put/Get/Accumulate operations to other processes in its communicator group as long as the target buffers are non-overlapping. Finally all the processes call a second FENCE to ensure the completion of one-sided operations issued during the access epoch. The second FENCE marks the completion of the current access epoch and also signals the start of the next access epoch. This ability to issue multiple communication operations in each access epoch provides the opportunity for exploit reduced synchronization overhead. The fence form of synchronization also helps the user escape the complexity of handling constantly changing communication patterns.

Though AWM-Olsen application goes through alternating computation and communication phases, each process is involved in data exchange with only its six neighbors in the 3-dimensional grid. The collective nature of fence synchronization causes an unnecessary overhead in the case of such a localized communication pattern. Also this overhead increases as the process count increases. We have explored the use of sub-communicators to reduce this overhead. A straightforward implementation is to create a sub-communicator for each process including its nearest-neighbors as shown in Figure 6 a. However, this leads to a complex set of dependencies where each process is involved in fences over six overlapping sub-communicators.

Each communication phase of the AWM-Olsen application can be divided into three sub-phases based on the direction in which the transfers happen. First, all the processes are involved in a north-south exchange. Each process sends data to its northern neighbor and then, to its southern neighbor. A similar exchange happens in the east-west and north-south directions. As we show in Figure 6 b, the dependencies in each of these sub-phases are localized to the row/column of processes in one dimension. We use this localization to create sub-communicators. Each process is part of three sub-communicators, one in each direction. After communication in a given direction, the processes call a fence on their sub-communicators in that dimension. This reduces the amount of synchronization to a minimum. For example, in a 20x20x20 process grid, localized fence synchronization involves only 20 processes instead of 8000 processes as in the

naive case. Figure 7 shows the velocity exchange implemented using fence.

Through this study we try to show the need for localized fence semantics in the MPI standard. Though such an effect can be achieved using sub-communicators, at the application level, as we have done here, in scenarios where the communication patterns are more asymmetrical and dynamic in phases, this will become a complex task for the programmer. Also this can be handled more efficiently inside the MPI implementation.

VELOCITY EXCHANGE
North and South Exchange
s2n(u1,north-mpirank, south-mpirank) *! recv from south, send to north*
n2s(u1, south-mpirank, north-mpirank) *! send to south, recv from north*
. . . repeat for velocity components v1,w1
MPI_Win_fence(north-south sub-communicator)
s2nfill(u1, window buffer, south-mpirank)
n2sfill(u1, window buffer, north-mpirank)
. . . repeat for velocity components v1,w1
East and West, Up and Down Exchanges
. . .

S2N
Copy 2 planes of data from variable to sendbuffer *!north face excluding ghost cells*
MPI_Put(sendbuffer, north-mpirank)

S2NFILL
Copy 2 planes of data from window buffer to variable *! south ghost cells*

Figure 7: Velocity Exchange with Reduced Synchronization using Fence

3.2.3 Design and Implementation using Post-Wait/Start-Complete

The Post-Wait/Start-Complete model provided by MPI-2 aims at reducing synchronization between processes to the minimum by requiring only pairs of processes to synchronize. This model is aimed at applications where processes communicate with only a few logical neighbors and the communication pattern is fixed or changes infrequently. The asynchronous nature of these semantics also provide a good opportunity for communication and computation overlap.

The nearest neighbor communication pattern of the AWM-Olsen application suites this model well. Considering the north-south communication phase, a naive implementation has each process post the window to its southern neighbor, issue the data transfer to its northern neighbor and wait for it to complete and then post the window at its northern neighbor. The MPI-2 one-sided semantics allow a process to post its window to a group of processes with one Post call. Similarly a process can start accessing the windows of a group of processes with one Start call. Also, as discussed earlier, multiple put operations can be issued concurrently with the constraint that the target buffers and the source buffers of these operations at a process are non-overlapping. As shown in Figure 8, we post windows to the northern and southern neighbors and wait for completions after issuing all transfers in both the directions, thus reducing the synchronization

VELOCITY EXCHANGE

North and South Exchange

MPI_Win_post(window, north-south group)
MPI_Win_start(window, north-south group)
s2n(u1,north-mpirank, south-mpirank) *! recv from south, send to north*
n2s(u1, south-mpirank, north-mpirank) *! send to south, recv from north*
. . . repeat for velocity components v1,w1
MPI_Win_wait(window, north-south group)
MPI_Win_complete(window, north-south group)
s2nfill(u1, window buffer, south-mpirank)
n2sfill(u1, window buffer, north-mpirank)
. . . repeat for velocity components v1,w1

East and West, Up and Down Exchanges

. . .

S2N

Copy 2 planes of data from variable to sendbuffer *!north face excluding ghost cells*
MPI_Put(sendbuffer, north-mpirank)

S2NFILL

Copy 2 planes of data from window buffer to variable *! south ghost cells*

Figure 8: Velocity Exchange with Reduced Synchronization using Post-Wait/Start-Complete

overhead. We follow a similar pattern in other directions as well. The synchronization overhead can be further reduced by increasing the size of the staging buffers and by issuing transfers in all directions before synchronizing. But this comes at the cost of larger pinning and registration overhead.

Similar to the code restructuring described in Section 3.2, in this implementation, we also interleave the computation and communication for more overlap with post-wait/start-complete synchronization.

4 Experimental Results

4.1 Experimental Setup

We have run all of our experiments on the TACC Ranger system. Ranger is a blade-based system. Each node is a SunBlade x6420 running a 2.6.18.8 Linux kernel. Each node contains four AMD Opteron Quad-Core 64-bit processors (16 cores in all) on a single board, as an SMP unit. Each node has 32 GB of memory. The nodes are connected with InfiniBand SDR adapter from Mellanox [1]. In our experiments, we use MVAPICH2 1.4rc2 [14] as the underlying MPI model. We use a weak scaling model for our experiments to simulate the real world application use. We increase the size of the data set as the process count increases such that data grid size per process remains at 256x256x256 elements.

4.2 Performance of ISend/IRecv-based Designs

To study the effect of reduced synchronization at the application level, we modified the velocity and stress exchanges, which form part of the main loop in AWM-Olsen. In this section we compare the performance of the original asynchronous two-sided version with the asynchronous two-sided version with reduced synchronization. Part (a) in Figure 9 shows the times of the velocity exchange loop for one iteration. We see an improvement of 11% for 512 processes and 9% in the case of 4,096 processes. We believe the variation in performance improvement across problem sizes is because of the weak scaling we have used. Part (b) shows the improvement we observe due to this optimization in the overall velocity and stress computation and exchange times. Figure 10 shows gains in velocity computation and exchange times due to overlap. We see a 2% improvement for 4,096 processes.

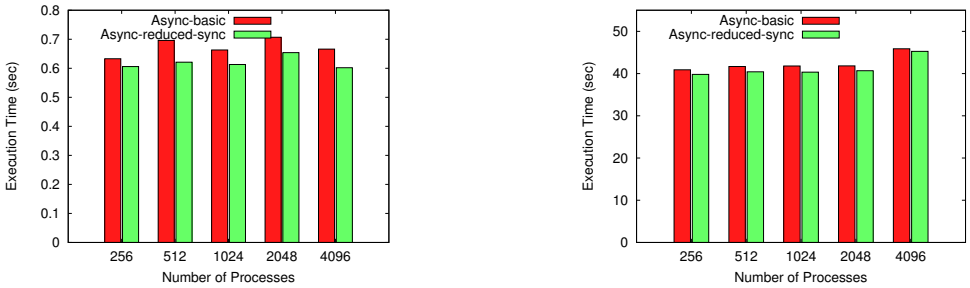

Figure 9: Asynchronous Two-sided Versions with Reduced Synchronization: (a) Velocity Exchange Time and (b) Velocity and Stress Computation and Exchange Time

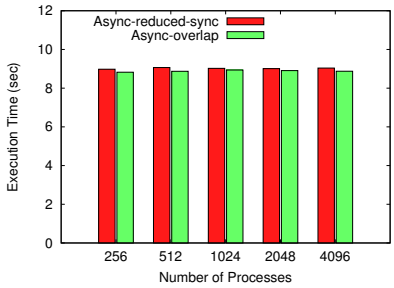

Figure 10: Asynchronous Two-sided Version with Overlap - Velocity computation and exchange time

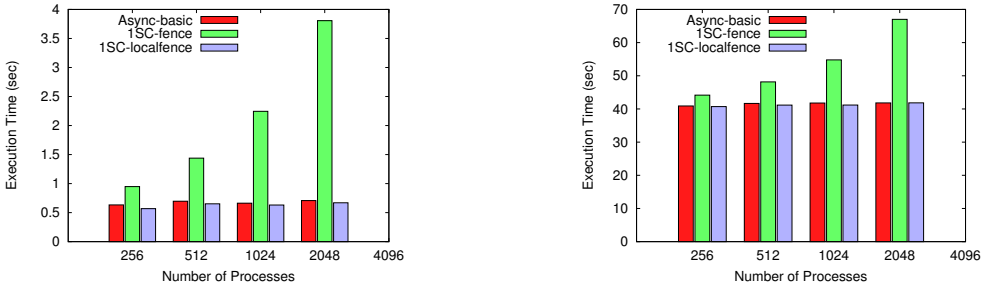

Figure 11: Fence Versions with Reduced Synchronization: (a) Velocity Exchange Time and (b) Velocity and Stress Computation and Exchange Time

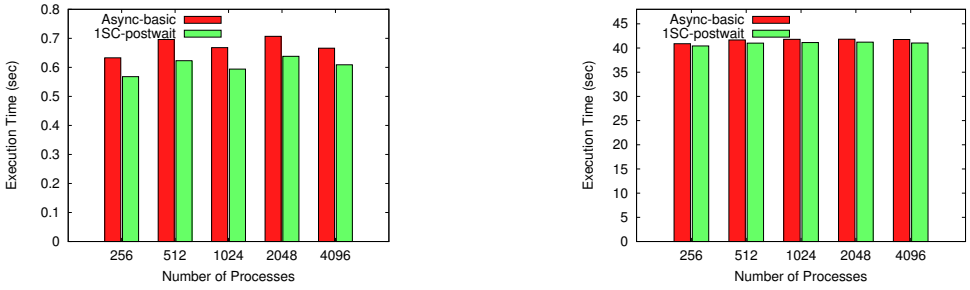

Figure 12: Post-wait/Start-Complete Versions with Reduced Synchronization: (a) Velocity Exchange Time and (b) Velocity and Stress Computation and Exchange Time

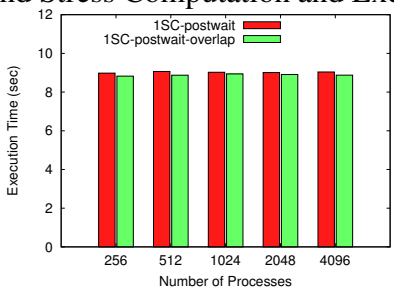

Figure 13: Post-wait/Start-Complete Version with Overlap - Velocity computation and exchange time

4.3 Performance of Fence-based Designs

Here, we compare the two new designs of AWM-Olsen application that use fence synchronization with the original asynchronous version. Results in figure 11 clearly show that the overhead of global fence operation increases drastically with the number of processes while the localized fence design avoids this overhead upto 2048 processes and performs better than the original asynchronous design due to the effect of reduced synchronization. We see a 5% improvement in the velocity exchange times for 2048 processes.

4.4 Performance of Post-Wait/Start-Complete-based Designs

Figure 12 shows the improvement due to reduced synchronization using Post-Wait/Start-Complete over the original asynchronous two-sided version. The velocity exchange times improved by 10.4% for 512 processes and by 8.6% for 4,096 processes. The improvements we observe here are similar to those achieved by using Waitall calls in the improved two-sided version. Figure 13 shows the improvement due to overlap in the Post-Wait/Start-Complete version. We observe a 2.5% improvement in velocity exchange latency for 1,024 processes. Again the results show that one-sided and non-blocking two-sided implementations show similar performance improvements. From a high level, one-sided semantics appear to have an advantage over two-sided semantics on modern networks where large message transfers happen through RDMA [16]. This is because address information exchange and memory registration happen during the window creation phase unlike two-sided communication where these happen in the communication critical path. However, techniques like registration cache used in the underlying implementations help remove this overhead in the two-sided communication helping it achieve similar gains as in the one-sided case.

4.5 Application performance

In this section, we present the impact of our optimizations on the overall performance of the AWM-Olsen application. The times shown Figure 14 are for a 6 timestep execution. We see close to 7% improvement in performance on 4,096 processes for the Post-Wait/Start-Complete version with overlap. Assuming that we get similar gains on 32,000 cores, this improvement would correspond to savings of 45875 core-hours in the real world run described in section 2. We also observed that localized fence minimizes the overhead due to global synchronization. However, the size of local fence groups increases as size of the process grid increases. This results in a noticeable overhead even in the case of the localized fence design as we go to 4,096 processes.

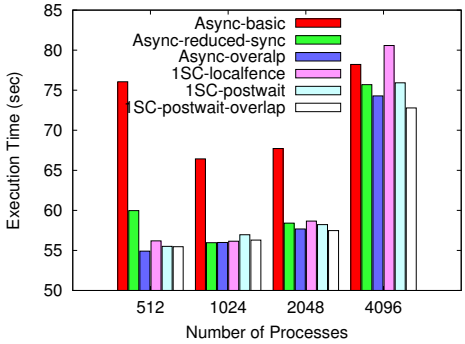

Figure 14: Effect of optimizations on the overall performance of the AWM-Olsen application

5 Related Work

In this section, we give a brief introduction on the related research.

MPI-2 standard [12] expanded MPI to include Remote Memory Access (RMA) or one-sided communication. This mode of communication facilitates the coding of some applications with dynamically changing data access patterns where the data distribution is fixed or slowly changing. There are proposals in the MPI forum to extend the current RMA interfaces [2]. Initial topics include exploiting different memory models, extending accumulate to support user-defined operations, extending accumulate to support read-modify-write operations, active message, etc. Our work in this paper targets at analyzing the potential of existing one-sided and non-blocking two-sided semantics to exploit reduced synchronization and communication-computation overlap in the AWM-Olsen application.

The Anelastic Wave Model (AWM) was originally developed by Kim Olsen et al. [10, 15, 3] and was picked as the primary model for the SCEC TeraShake simulation. As part of their work in [5, 4], Cui et al. enhanced the application through single-processor optimizations, optimization of I/O handling and optimization of TeraShake initialization. In this work, we improve the application performance further by optimizing communication through the afore mentioned latency hiding techniques.

The study in [13] explores the use of MPI one-sided semantics coupled with multi-threading to optimize the Community Atmosphere Model. Hermmans et al. investigate the performance of one-sided communication alternatives in the Nas Parallel Benchmark BT application running on 256 cores of a Blue Gene/P [7].

6 Conclusion and Future work

AWM-Olsen is a widely used earthquake-induced ground wave-propagation simulation code which is run on tens of thousands of cores of TeraGrid HPC machines and consumes 10's of millions of CPU hours every year. Efficient communication design is paramount in such large scale, heavily used applications to make best use of available system resources. In this paper, we optimize the communication in AWM-Olsen using two common latency-hiding techniques - reduced synchronization and communication-computation overlap. We achieve a 7 percent improvement in overall application performance at 4096 cores. Traditionally, non-blocking two-sided semantics from MPI-1 have been used to acheive the afore mentioned benefits in many HEC applications. In this work, we explore the use of the more recently proposed MPI-2 one-sided model to acheive these effects. We observe that, similar benefits can be achieved using either models because the underlying MPI implementation uses similar RDMA operations in both cases.

We plan to extend this work by running the MPI-2 one-sided implementation of the AWM-Olsen at full core count (60,000 cores) on Ranger and comparing performances with the two-sided implementations of the AWM-Olsen at that high core count. As future work, we plan to explore hybrid programming techniques such as MPI+OpenMP to further optimize both computation and communication in the AWM-Olsen application. We also plan to analyze the limitations of the existing one-sided model in delivering performance for stensil based applications and propose extensions to overcome these limitations.

7 Acknowledgements

The authors wish to acknowledge Gopal Santhanaraman for developing the initial one-sided fence design and the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources on the Ranger that have contributed to the research results reported within this paper. This paper is based upon work supported by the National Science Foundation under Grants #0833139, #0833155 and #0833169.

References

[1] Mellanox Technologies. http://www.mellanox.com.

[2] MPI-3 RMA. http://meetings.mpi-forum.org/secretary/2008/03/slides/mpi3-rma-summary-3-10.pdf.

[3] Y. Cui, K. B. Olsen, Y. Hu, S. Day, L. A. Dalguer, B. Minster, R. Moore, J. Zhu, P. Maechling, and T Jordan. Optimization and Scalability of an Large-scale Earthquake Simulation Application. In *American Geophysical Union, Fall Meeting*, 2006.

[4] Yifeng Cui, Reagan Moore, Kim Olsen, Amit Chourasia, Philip Maechling, Bernard Minster, Steven Day, Yuanfang Hu, Jing Zhu, and Thomas Jordan. Toward petascale earthquake simulations. In *Acta Geotechnica*, pages 79–93, 2009.

[5] Yifeng Cui, Reagan Moore, Kim Olsen, Amit Chourasia, Philip Maechling, Bernard Minster, Steven Day, Yuanfang Hu, Jing Zhu, Amitava Majumdar, and Thomas Jordan. Enabling very-large scale earthquake simulations on parallel machines. In *ICCS '07: Proceedings of the 7th international conference on Computational Science, Part I*, pages 46–53, Berlin, Heidelberg, 2007. Springer-Verlag.

[6] S. Narravula A. Mamidala G. Santhanaraman, T. Gangadharappa and D. K. Panda. Design Alternatives for Implementing Fence Synchronization in MPI-2 One-sided Communication for InfiniBand Clusters. In *IEEE Cluster*, 2009.

[7] Marc-André Hermanns, Markus Geimer, Bernd Mohr, and Felix Wolf. Scalable detection of MPI-2 remote memory access inefficiency patterns. In *16th Euro PVM/MPI, Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[8] Wei Huang, Gopalakrishnan Santhanaraman, Hyun-Wook Jin, and Dhabaleswar K. Panda. Scheduling of mpi-2 one sided operations over infiniband. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 9*, page 215.1, Washington, DC, USA, 2005. IEEE Computer Society.

[9] Weihang Jiang, Jiuxing Liu, Hyun-Wook Jin, D. K. Panda, W. Gropp, and R. Thakur. High performance mpi-2 one-sided communication over infiniband. In *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 531–538, Washington, DC, USA, 2004. IEEE Computer Society.

[10] Olsen KB. Simulation of three-dimensional wave propagation in the Salt Lake Basin. Technical report, University of Utah, Salt Lake City, Utah, 1994.

[11] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[12] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.

[13] Arthur A. Mirin and William B. Sawyer. A scalable implementation of a finite-volume dynamical core in the community atmosphere model. *Int. J. High Perform. Comput. Appl.*, 19(3):203–212, 2005.

[14] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[15] K. Olsen, S.M. Day, J.B. Minster, Y. Cui, A. Chourasia, M. Faerman, R. Moore, P. Maechling, and T Jordan. Strong Shaking in Los Angeles Expected from Southern San Andreas Earthquake. In *Geophysical Research Letters, Vol 3*, pages 1–4, 2006.

[16] G. Santhanaraman, P. Balaji, K. Gopalakrishnan, R. Thakur, W. Gropp, and D. K. Panda. Natively supporting true one-sided communication in mpi on multi-core systems with infiniband. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 380–387, Washington, DC, USA, 2009. IEEE Computer Society.