Mastering Performance Challenges with the New MPI-3 Standard

By Mikhail Brinskiy, Software Development Engineer, Alexander Supalov, Intel Cluster Tools Architect, Michael Chuvelev, Software Manager, and Evgeny Leksikov, Intel Corporation

Introduction

This article demonstrates the performance benefits of the MPI-3 nonblocking collective operations supported by the Intel® MPI Library 5.0 and the Intel® MPI Benchmarks (IMB) 4.0 products. We'll show how to measure the overlap of communication and computation, and demonstrate how an MPI application can benefit from the nonblocking collective communication.

The Message Passing Interface (MPI) standard is a widely used programming interface for distributed memory systems. The latest MPI-3 standard contains major new features such as nonblocking and neighbor collective operations, extensions to the Remote Memory Access (RMA) interface, large count support, and new tool interfaces. The use of large counts lets developers seamlessly operate large amounts of data, the fast, one-sided operations strive to speed up Remote Memory Access (RMA)-based applications, and the nonblocking collectives enable developers to better overlap the computation and communication parts of their applications and exploit potential performance gains.
Here, we concentrate on two major MPI-3 features: nonblocking collective operations (NBC) and the new RMA interface. We evaluate the effect of communication and computation overlap with NBC, and describe potential benefits of the new RMA functionality.

For our measurements, we use Intel MPI Library 5.0 Beta and Intel MPI Benchmarks 4.0 Beta (IMB), both of which support the MPI-3 standard. Starting from version 4.0 Beta, IMB contains two new binaries: IMB–NBC, intended to measure the effectiveness of nonblocking collective operations, and IMB–RMA, covering the most important MPI-3 updates to the RMA interface. The complete IMB download, as well as a detailed description of all IMB benchmark suites, are available at: http://software.intel.com/en-us/articles/intel-mpi-benchmarks.

**Nonblocking Collective Operations (NBC)**

NBC semantics allow an MPI user to overlap communication with computation, like nonblocking point-to-point operations already do. Also, NBC mitigate the effects of pseudo-synchronization that is inherent in many collective algorithms due to data dependencies. A detailed analysis of the nonblocking collective operations potential is available at: http://htor.inf.ethz.ch/publications/img/hoefler-nbc-standard.pdf.

A common assumption about the nonblocking operations (collective, as well as point-to-point) is that the progress happens in the background without user intervention. However, the MPI standard does not define an explicit progression rule and leaves it up to a so-called “high quality” implementation to offer true asynchronous progress. Thus, many early MPI libraries did not offer asynchronous progression and just performed all communication in the intervening MPI calls. With the advent of NBC, a user of MPI is free to use different approaches to overlap a collective operation with computation: either periodically calling a `MPI_Test()` routine to ensure explicit message progression, or using a single `MPI_Wait()` call to complete the operation.

The IMB methodology to measure the effectiveness of communication and computation overlap is designed to match the most popular use of nonblocking communication, specifically in cases when some activity is done between the operation start and its completion by the test or wait calls. This methodology assumes the communication and computation times to be approximately equal, in order to minimize the inaccuracy induced by the network or operating system (OS) noise. IMB makes the CPU busy for the desired amount of time by calculating a fully vectorized 100×100 matrix by vector product. Thus, in a nutshell, the benchmark flow looks as follows:

1. Measure the time needed for a pure communication call (e.g., `MPI_Ibcast()` followed by `MPI_Wait()`)
2. Start communication (e.g., call `MPI_Ibcast()`)
3. Start computation with duration equal to the time measured in step 1. Thus we ensure that communication and computation parts consume approximately the same amount of time.
4. Wait for communication to finish (i.e., call `MPI_Wait()`).
Given the description above, the IMB-NBC benchmarks output four timings:

- \textit{time\_pure} is the time for nonblocking operation, which is executed without any concurrent CPU activity;
- \textit{time\_CPU} is the time of the test CPU activity (which is supposed to be close to the \textit{time\_pure} value);
- \textit{time\_ovrlp} is the time for nonblocking operations concurrent with CPU activity;
- \textit{overlap} – the estimated overlap percentage obtained by the following formula:
  \[
  \text{overlap} = 100 \times \max(0, \min(1, \frac{(\text{time\_pure} + \text{time\_CPU} - \text{time\_ovrlp})}{\max(\text{time\_pure}, \text{time\_CPU})}))
  \]

The Intel MPI Library supports asynchronous message progressing in the multithreaded library only, so we use it in our experiments to leverage the performance potential of the nonblocking collective operations. To enable asynchronous progress in the Intel MPI Library, the environment variable \texttt{MPICH\_ASYNC\_PROGRESS} should be set to 1.

In our experiment, we compare the results of the IMB–NBC benchmarks measuring overlap potential of nonblocking collectives with the regular blocking collective operations results obtained using the IMB–MPI1. The aim is to check which is more efficient: to perform a blocking collective operation and run computation afterward, or to use its nonblocking counterpart and overlap it with the same amount of computation. Thus, the charts reflect two timings: the time taken by the nonblocking operation concurrent with the CPU activity (which is \textit{time\_ovrlp} value in the IMB-NBC output), and the aggregated communication and computation time. The latter is the sum of the blocking collective operation time (measured by the IMB–MPI1 benchmark) and the time of the test CPU activity (which is the \textit{time\_CPU} value in the IMB–NBC output).

The results shown in these charts have been collected using 48 processes on 4 IVT nodes (12 processes per each node). Each node is equipped with two Intel® Xeon® processors E5-2697 and one Mellanox Connectx-3 InfiniBand* adapter.

The overlap obtained by the \texttt{ialltoall} and \texttt{iallgather} benchmarks is close to being ideal. Figure 1 and Figure 2 reflect only big message sizes as the most representative ones. Even though the nonblocking operations with asynchronous progress enabled cause significant overhead, especially noticeable at small message sizes, such an overlap is more efficient than the regular blocking collective followed by the computation. However, this is not always the case. Figure 3 depicts the results for the broadcast benchmark and small messages sizes. While the obtained overlap is quite good (about 60–70%), the regular blocking broadcast performs much faster, which makes it more performance efficient on message sizes of up to 2KB.
1. **MPI_Ialltoall** overlapped with computation vs. **MPI_Alltoall** followed by computation.

2. **MPI_Iallgather** overlapped with computation vs. **MPI_Allgather** followed by computation.

3. **MPI_IBcast** overlapped with computation vs. **MPI_Bcast** followed by computation.
Combining these results, we reach the following conclusions:

> Nonblocking collective operation may provide significant performance benefits in comparison with its blocking counterparts when overlapped with computation, mostly on medium and large message sizes.

> In some cases, the overhead introduced by the nonblocking operation itself and asynchronous message progressing may negate the benefit of the overlap. This is mostly relevant for small message sizes, where latency is usually an important factor.

### New RMA interface

MPI-3 standard significantly extends the RMA interface by introducing several major features, including different memory models, several new communication and synchronization calls, and new ways of creating RMA windows. Support for different memory models is one of the most important novelties intended to distinguish situations when cache coherency is supported (unified model) and when it is not supported (separate model, typical of the MPI-2 standard). This distinction enables high-performance MPI implementations and easier programming. The major update of the synchronization semantics is intended to align the passive target communication mode with the current HPC needs. The main problems of the previous RMA interface that MPI-3 standard tries to address are described at: [http://upc.lbl.gov/publications/bonachea-duell-mpi.pdf](http://upc.lbl.gov/publications/bonachea-duell-mpi.pdf).

Prior to version 4.0 Beta, IMB provided a set of RMA benchmarks that only utilize active communication mode. These benchmarks are available in the IMB-EXT module. IMB 4.0 Beta concentrates on the new MPI-3 RMA functionality and passive target communication mode. The new IMB-RMA module contains 19 benchmarks measuring new atomic functions, halo exchanges using the `MPI_Put()` and `MPI_Get()` operations, and concurrent RMA calls to different targets. All of these benchmarks utilize the passive target communication mode.

One of the new RMA benchmarks demonstrates whether MPI implementation supports the natural passive mode of the communication, in which the target process should not necessarily call MPI for the origin process to complete its access epoch. This may be supported via the asynchronous message progressing described above, or by utilizing underlying hardware capabilities that support communication offload in some way. In Intel MPI Library 5.0 Beta, RMA operations are implemented using regular point-to-point messages; therefore the operation needs to be progressed on the target, as well as on the origin, to ensure its completion. But as stated above, asynchronous message progressing is supported by the multithreaded version of the Intel MPI Library. The single-threaded version of the library does not support the real passive target mode. Therefore, the target must call MPI to ensure operation progressing. In our test, we compare the results obtained with the enabled asynchronous progress and the results collected with the single-threaded library to evaluate the benefits of the truly passive mode support in the Intel MPI Library.
The benchmark described above measures two timings:

1. The time consumed by the origin process to complete the `MPI_Put()` operation, while the target process is sitting in `MPI_Barrier()` call (therefore ensuring progress).

2. The time consumed by the origin process to complete the `MPI_Put()` operation, while the target process performs some computations outside the MPI stack, and then invokes `MPI_Barrier()`. The computation lasts for about the same time as is needed for the origin process to complete step 1. To achieve this, the origin process sends the timing value needed for the `MPI_Put()` completion to the target process.

The target and the origin processes are synchronized prior to each `MPI_Put()` operation. Thus, in case of absence of direct communication offload capabilities or a separate thread for message progressing, the time obtained in step 2 should be noticeably greater than the time obtained in step 1 (about 2x). But if the MPI implementation provides truly passive mode for RMA, these timings should be about the same.

4. Influence of asynchronous progress on the results of the truly passive put benchmark (small to medium message sizes).

5. Influence of asynchronous progress to the results of the truly passive put benchmark (large message sizes).
The charts in Figure 4 and Figure 5 show that enabling asynchronous message progressing brings very significant overhead, which is clearly seen on message sizes of up to 128 KB. However, on big message sizes the overhead is not really noticeable, while the computation performed on the target does not delay the completion of the MPI_Put() operation on the origin.

We analyzed the behavior of the passive mode communication model with the Intel MPI Library 5.0 Beta and observed that while the truly passive mode is supported by the library with asynchronous progress enabled, there are still significant improvement opportunities. These include native support for RDMA progress and various performance optimizations, such as improvement of asynchronous message progressing that currently introduces significant overhead and is available only in the multithreaded version of the library. The updated RMA interface in MPI-3 provides MPI users with a very flexible interface for one-sided operations. The potential benefits of this new interface are described at: http://htor.inf.ethz.ch/publications/img/mpi_mpi_hybrid_programming.pdf. Now, it's time for MPI implementations to be performance-efficient and competitive with various PGAS languages.

Conclusion

MPI-3 standard provides MPI users with a more flexible interface for writing performance-efficient programs. We have examined two major novelties—nonblocking collective operations and the new RMA interface with the help of the Intel MPI Library 5.0 Beta. We conclude that each of these new features may contribute to performance improvement of MPI applications. Though for existing MPI programs the use of new MPI-3 features may lead to application modification, in some cases, as we have demonstrated, these benefits far outweigh the efforts.

References