MPI Advisor: a Minimal Overhead Tool for MPI Library Performance Tuning

Esthela Gallardo
The University of Texas at El Paso
Department of Computer Science
egallardo5@miners.utep.edu

Jerome Vienne
The University of Texas at Austin
Texas Advanced Computing Center
viennej@tacc.utexas.edu

Leonardo Fialho
The University of Texas at Austin
Texas Advanced Computing Center
fialho@tacc.utexas.edu

Patricia Teller
The University of Texas at El Paso
Department of Computer Science
pteller@utep.edu

James Browne
The University of Texas at Austin
Texas Advanced Computing Center
browne@cs.utexas.edu

ABSTRACT

A majority of parallel applications executed on HPC clusters use MPI for communication between processes. Most users treat MPI as a black box, executing their programs using the cluster's default settings. While the default settings perform adequately for many cases, it is well known that optimizing the MPI environment can significantly improve application performance. Although the existing optimization tools are effective when used by performance experts, they require deep knowledge of MPI library behavior and the underlying hardware architecture in which the application will be executed. Therefore, an easy-to-use tool that provides recommendations for configuring the MPI environment to optimize application performance is highly desirable. This paper addresses this need by presenting an easy-to-use methodology and tool, named MPI Advisor, that requires just a single execution of the input application to characterize its predominant communication behavior and determine the MPI configuration that may enhance its performance on the target combination of MPI library and hardware architecture. Currently, MPI Advisor provides recommendations that address the four most commonly occurring MPI-related performance bottlenecks, which are related to the choice of: *1)* point-to-point protocol (eager *vs.* rendezvous), *2)* collective communication algorithm, *3)* MPI tasks-to-cores mapping, and *4)* Infiniband transport protocol. The performance gains obtained by implementing the recommended optimizations in the case studies presented in this paper range from a few percent to more than 40%. Specifically, using this tool, we were able to improve the performance of HPCG with MVAPICH2 on four nodes of the Stampede cluster from 6.9 GFLOP/s to 10.1 GFLOP/s. Since the tool provides application-specific recommendations, it also informs the user about correct usage of MPI.

EuroMPI '15, September 21-23, 2015, Bordeaux , France

DOI: http://dx.doi.org/10.1145/2802658.2802667

Keywords

Performance tool; performance optimization; MPI; HPC.

1. INTRODUCTION

A majority of parallel applications executed on High Performance Computing (HPC) clusters use an MPI[1] library for communication. Implementations of MPI libraries have parameters that may be set to optimize the performance of a given application for a target architecture or for the characteristics of the application. MPI library developers and cluster system administrators try to set the default MPI parameters to provide good performance for most applications. However, for many applications the default parameters may lead to performance degradation, particularly for large and complex applications that often have long run times[2]. For these cases, choosing different MPI parameters can produce significant application performance enhancements. However, most users treat MPI as a black box and run their applications using one of the available MPI libraries configured using the cluster's default parameters.

There are several well-established tools that target optimization of an application's use of MPI. Often these tools are effective when used by performance experts and in some cases enable comprehensive optimization of MPI library performance. However, most users do not have the deep knowledge of MPI library component behaviors and hardware architectures required for effective use of these tools. And, few, if any, of these tools provide recommendations for optimizing parameters for a given combination of application, MPI library implementation, and hardware architecture. For these reasons most users do not use the existing tools and MPI performance tuning is usually done only by performance experts. Therefore, there is a need for an easy-to-use, low-overhead tool that collects and interprets performance measurements and recommends performance-effective MPI library parameters that enable users to tune their MPI usage.

This paper presents such a tool, MPI Advisor: a simple-to-use tool that requires no instrumentation of the application and is fully executed from a single command line on

[1] http://www.mpi-forum.org/

[2] Recent studies done at the Texas Advanced Computing Center have found that thousands of jobs have resource use patterns, which suggests that different choices of MPI parameters could substantially improve their performance [23].

an HPC cluster access terminal. In designing the methodology and tool, the goal was to automate the tasks that require knowledge of MPI library internals and provide the user with easy-to-implement recommendations that could enhance communication performance by tuning MPI library parameters. In general, performance optimization has four steps: measurement, analysis of performance bottlenecks, generation of recommendations for optimization, and implementation of the recommended optimizations. For MPI-related performance, MPI Advisor automates the first three steps of this process, *i.e.*, through generation of recommendations for optimization. Accordingly, MPI Advisor incorporates: *1)* data collection using the existing MPI profiling interface (`PMPI`) and MPI Tools Information (`MPI_T`) interface, *2)* analysis that translates the collected data into performance metrics which identify specific performance degradation factors, and *3)* recommendations for optimization, which are presented in a way that users with a minimum knowledge of MPI will be able to implement them. We believe that informing users about correct usage of MPI libraries will enable them to make efficient use of MPI in the future. Currently, MPI Advisor provides tuning strategies to address the four most commonly occurring MPI-related performance bottlenecks, which are related to the choice of: *1)* point-to point protocol (eager *vs.* rendezvous), *2)* collective communication algorithm, *3)* MPI tasks-to-cores mapping, and *4)* Infiniband transport protocol. Support for other MPI-related performance issues can and will be added in the future. The current version of the tool supports both MVAPICH2[3] and Intel MPI[4]. Support for other MPI library implementations will be included in future versions.

The innovations and contributions implemented in MPI Advisor are: *1)* specification of the measurements that are required to perform the four MPI optimizations, including the means to acquire this information using standard interfaces such as `PMPI` and `MPI-T`, *2)* a low-overhead implementation of these measurements that gathers the required data in a single execution of the input application and requires no application instrumentation by the user, and *3)* a set of rules that uses the acquired measurements for recommending any supported optimizations that are needed. As the paper demonstrates, MPI Advisor has been used in a real environment with benchmarks and real applications and currently available high-performance MPI libraries. The performance gains obtained in the case studies presented in this paper range from a few percent to more than 40%.

This paper is organized as follows: Section 2 provides a brief background about the MPI standard features and the `hwloc` tool, which is used by MPI Advisor. Section 3 describes the MPI Advisor methodology and tool, while Section 4 discusses the four tuning strategies currently supported by MPI Advisor and indicates the performance tradeoffs associated with each, in particular, for the MPI libraries available on the Texas Advanced Computing Center's Stampede cluster. Section 5 demonstrates the efficacy of MPI Advisor recommendations by presenting the recommendations provided by each tuning strategy and the results of implementing them. In Section 6 we review related research and, finally, in Section 7 we conclude the paper and discuss future research directions.

[3]http://mvapich.cse.ohio-state.edu/

[4]https://software.intel.com/en-us/intel-mpi-library

2. BACKGROUND

MPI is one of the most popular programming models for writing parallel applications for HPC clusters. The MPI standard defines a large set of primitives that programmers can employ to implement communication and coordination among processes. This set of primitives includes point-to-point, collective, and one-sided communication, which employ standard and user-defined data types; logical management of communication peers using task groupings, communicators, and topology; and dynamic process creation and management. MPI also provides interfaces for inserting tools for profiling the execution behavior of programs that use MPI. The standard defines only the functionality and behavior of communication, not the implementation. Libraries based on the MPI standard may vary considerably in their implementations of the functions defined by the standard.

Implementations of MPI libraries are closely tied to the underlying network architectures and characteristics of the target HPC systems. MPI library developers and the system administrators for the clusters on which such libraries are installed try to provide applications with the best communication performance on a given interconnection network, mostly through selection of default MPI parameters. Given the diversity of application communication patterns and the complexity of modern HPC system architectures, it is impossible for default parameters to be optimum for all circumstances. And, many users lack the knowledge to choose the best parameters for their applications.

To optimize the MPI performance of an application, MPI libraries provide different ways to profile its execution, the most common being the `PMPI` interface. This interface encapsulates each MPI routine and provides hooks for tools to execute any task before and after the execution of the routine. For example, it enables tools to identify the arguments passed by the application to any MPI routine. In addition, the `MPI_T` interface provides query functions to detect variables and their values inside the MPI library. These variables are split into two classes: control variables and performance variables. `MPI_T` control variables are used to report the current settings of the employed MPI library. Similar to hardware performance counters in modern processors, `MPI_T` performance variables are incremented each time an event occurs during the execution of an MPI program. For example, they count the number of times buffer memory allocation, message sent, and message received occur. The `MPI_T` variables are not defined by the MPI standard; each library provides its own set of variables. Tables 1 and 2 present examples of the variables available in MVAPICH2.

To facilitate the understanding of complex HPC system

Name	Value
MPIR_CVAR_MEMDUMP	1
MPIR_CVAR_ASYNC_PROGRESS	0
MPIR_CVAR_DEFAULT_THREAD_LEVEL	"z"
MPIR_CVAR_DEBUG_HOLD	0
MPIR_CVAR_SUPPRESS_ABORT_MESSAGE	0
MPIR_CVAR_PROCTABLE_SIZE	64
MPIR_CVAR_SCATTER_INTER_SHORT_MSG_SIZE	2048

Table 1: Example of MVAPICH2 control variables.

Name	Value
mem_allocated	level
mem_allocated	high watermark
num_malloc_calls	counter
num_calloc_calls	counter
num_memalign_calls	counter
num_strdup_calls	counter
num_realloc_calls	counter
num_free_calls	counter

Table 2: Example of MVAPICH2 performance variables.

architectures, Hardware Locality (`hwloc`) [1] gathers a detailed profile of the characteristics and properties of the hardware execution environment, including threads, cores, shared caches, sockets, and NUMA nodes, and exposes it in a generic and portable manner. It abstracts the machine architecture and its characteristics as a hierarchical tree of resources that applications and runtime systems can traverse to retrieve information.

3. MPI ADVISOR

As shown in Figure 1, MPI Advisor's execution comprises three phases: data collection (measurement), analysis, and recommendations. During these phases MPI Advisor identifies: *1)* the MPI environment in which the input application is executed, *2)* the application's characteristics that are pertinent to the supported tuning strategies, and *3)* the guidance to provide to the user.

Data collection is performed when MPI Advisor is installed on a target system and when it is used to profile an application. During installation, a custom collective evaluation script is executed to collect architecture-specific MPI-performance data that are utilized to build a table that is stored in MPI Advisor's configuration files for future use. Currently, this table is used in the analysis phase for one purpose: to identify, for every message size of interest, the best algorithm to employ for each MPI collective operation. Thus, the table is built by executing the Intel MPI benchmarks (IMB)[5] and OSU Micro-Benchmarks (OMB)[6] with each MPI library available on the system, and during their execution collecting performance data for every collective algorithm.

Application-specific MPI-performance data are collected by MPI Advisor when it is used to profile an application. Data collection is enabled by: *1)* identification of the MPI library version (`MPI_Get_library_version()`), which was introduced in the MPI 3.0 standard, *2)* mpiP [21], which is a lightweight profiling tool for MPI applications that uses the `PMPI` interface, *3)* `MPI_T`, which is a tools interface that was also introduced in the MPI 3.0 standard, and *4)* `hwloc`, which is the Portable Hardware Locality software package of the Open MPI group.

If the library follows the MPI 3.0 standard, MPI Advisor uses `MPI_Get_library_version()` to identify the MPI library employed by the application and the specific `MPI_T` variables to observe during application execution. Next,

[5]http://software.intel.com/en-us/articles/intel-mpi-benchmarks

[6]http://mvapich.cse.ohio-state.edu/benchmarks/

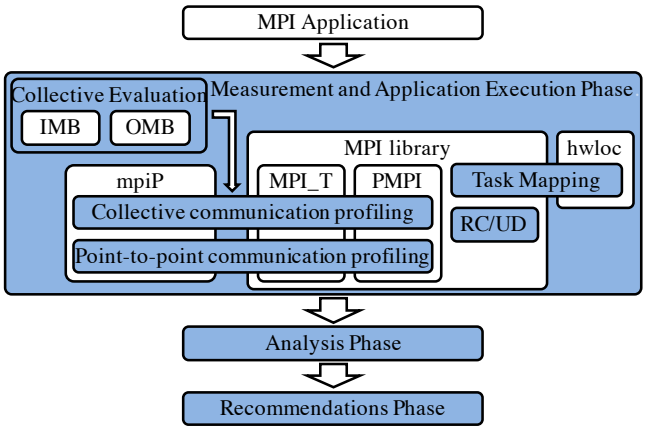


Figure 1: MPI Advisor framework and execution flow.

MPI Advisor launches the execution of the application to collect application-specific statistical performance data. Data regarding MPI functions are collected using mpiP; pertinent MPI environment characteristics are collected via `MPI_T`; and the mapping of MPI tasks to the cores of a node, as well as information about the location of communication devices, are detected through the use of `hwloc`.

This application-specific MPI performance data, which is collected during just a single execution of the input application, and the architecture-specific data provided by the collective evaluation script are used in the analysis phase. This phase: *1)* compares the collected data to a set of predefined heuristics that are based on expert knowledge and *2)* provides information to the recommendations phase that enables MPI Advisor to recommend the appropriate eager threshold, collective algorithm(s), tasks-to-cores mapping, and Infiniband transport protocol that should be employed by the application. Note, however, that if the employed library does not follow the MPI 3.0 standard, MPI Advisor only collects information that is pertinent to the tasks-to-cores mapping and transport protocol tuning strategies, both of which are agnostic of the employed MPI library and are supported by the MPI 2.0 standard.

If the employed library follows the MPI 3.0 standard, MPI Advisor provides recommendations that are specific to that library. Recommendations for the eager threshold and collective algorithms require the use of `MPI_T`. Two sets of reports are generated by the recommendations phase: *1)* one generated by mpiP, which contains an execution profile of the application, and *2)* one generated by MPI Advisor, which explains how the application performed in terms of each supported tuning strategy, suggestions on how the application's MPI performance could be improved, and examples of how to implement the recommendations. If the employed MPI library does not follow the MPI 3.0 standard, MPI Advisor recommends that the user to consult the documentation associated with the employed library for information regarding the MPI variables that need to be modified to implement the recommendations.

Since MPI Advisor was built upon mpiP, it acts as a link-time library and, thus, is simple to use. It provides the user with application-specific MPI-performance data, which is collected with a single execution of the input application, analyses that detect targeted MPI-performance bottlenecks,

Default	with mpiP	with MPI Advisor
29.01	28.99	28.63

Table 3: Impact of MPI Advisor on the performance of HPCG with 64 MPI tasks (values in GFLOP/s).

and recommendations to improve the application's MPI performance. Moreover, MPI Advisor has minimal overhead. For example, Table 3 shows the performance impact of mpiP and MPI Advisor on the GFLOP/s attained by HPCG (pure MPI), which was executed on Texas Advanced Computing Center's Stampede cluster using MVAPICH2 and 64 cores. As can be seen, MPI Advisor's performance impact is very low, *i.e.*, it reduces the achieved GFLOP/s by 1.3%, while mpiP reduces it by less than 0.1%.

4. SUPPORTED TUNING STRATEGIES

Currently MPI Advisor supports four classical and useful MPI performance-tuning strategies. Although they may seem commonplace to experts, they are likely not familiar to non-experts. Below, for each strategy, we provide background information, motivation for its use, and implementation details. As mentioned previously, only a single execution of the input application is required for MPI Advisor to collect all the data required to detect any performance bottlenecks that are pertinent to the tuning strategies, and to provide recommendations for enhancing the application's performance.

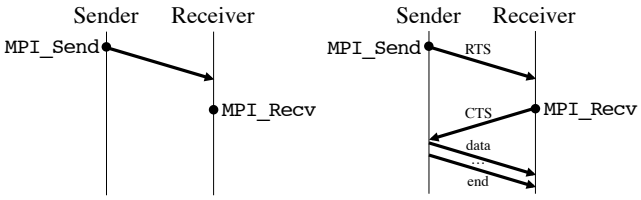
In addition, MPI Advisor can be easily expanded: each strategy is implemented as an independent component, which defines its own data collection and analyses.

4.1 Point-to-Point Protocol Threshold

In general, MPI libraries employ either the eager or rendezvous protocol for point-to-point communication. As illustrated in Figure 2, the eager protocol is asynchronous, while the rendezvous protocol is synchronous. Consequently, they differ in terms of their relative memory requirements. The eager protocol, which is used for small messages, stores each received message in an MPI-defined buffer whether or not a matching receive has been posted. Although this reduces synchronization delays and, thus, provides lower message latencies, significant memory may be required to provide buffer space for messages. In contrast, the synchronous rendezvous protocol employs a form of handshaking to initiate a data payload transfer; a pending message is sent only when there is adequate user-defined buffer space at the receiver.

Each MPI library defines the largest message size that it will send using the eager protocol; messages with sizes above this threshold are sent using the rendezvous protocol. A user-defined parameter can be used to change this threshold, which we generically call *eager threshold*. The MVAPICH2 default value of eager threshold is dependent on the transport media (shared memory, TCP/IP, Infiniband SDR, QDR, FDR, etc.); it is 17 KB for intra-node communication on Stampede, while Intel MPI sets the default eager threshold to 256 KB independent of the architecture.

Note that we focus only on increasing eager threshold since decreasing it could lead to more messages using the rendezvous protocol and, thus, degradation of application



(a) eager protocol (b) rendezvous protocol

Figure 2: Protocols used for point-to-point operations.

performance. Nonetheless, increasing eager threshold also increases the size of the buffer associated with the eager protocol, which may cause program termination due to node memory exhaustion. However, a study conducted at TACC shows that, as depicted in Figure 3, most of the MPI jobs do not use the entire memory available on a typical Stampede node (32 GB). That being said, to identify a value of eager threshold that is appropriate for an application requires knowledge that is not commonplace among many of the users of an HPC facility.

To tune the eager threshold parameter, MPI Advisor detects the predominant message sizes transmitted by the input application and infers the point-to-point protocol in use. When applicable, to increase application performance, MPI Advisor recommends the reduction of the number of messages being transmitted using the rendezvous protocol by increasing the eager threshold value. This recommendation is specific to each MPI library, and warns the user that making the change may increase the memory footprint of the MPI library. To determine whether or not to increase the eager threshold, during the data collection and analysis phases, MPI Advisor: *1)* uses `MPI_T` to identify the value of eager threshold, *2)* uses mpiP performance data to determine the number and size of messages transmitted via send and receive operations, and *3)* computes the median message size per call site, determines the maximum of these, compares the maximum to the value of eager threshold, and determines the appropriate value for eager threshold. If the computed value is larger than the default value, then MPI Advisor outputs its recommendation, instructions, and warnings. This strategy is based on several experiments and may be improved in the future.

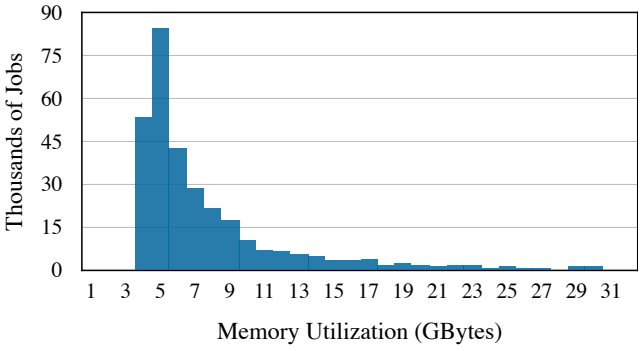


Figure 3: Average memory utilization distribution of multi-node jobs using MPI on Stampede.

4.2 Algorithms for Collective Operations

Often an MPI library includes more than one algorithm to implement each collective operation. Since algorithm performance may depend on system size, message size, and the communication-initiator task, MPI libraries provide a default algorithm for each combination of target system, collective operation, and message size. Note, however, that Intel MPI's default settings are based on the performance of the Intel MPI Micro-Benchmarks (IMB), while MVAPICH2's default settings are based on the performance of the OSU Micro-Benchmarks (OMB). In addition, MPI libraries often provide user-definable parameters for selection of an algorithm for either all message sizes or specific ones.

As the communication pattern of a given application may be different from that of the benchmark used to tune a given MPI library, using an algorithm that is not the library's default can result in surprisingly good performance. For example, as shown in Figure 4, for small size messages (128 bytes or less), a 256-core execution of OMB's `MPI_Bcast` (`osu_bcast`) with Intel MPI 4.1 and a user-defined (tuned) algorithm resulted in an execution time that is 35 times faster than with Intel MPI 4.1 and its default algorithm, and similar to MVAPICH2 1.9a2 and its default algorithm. Another example (shown in Figure 5) results from running OMB's `MPI_Gather` (`osu_gather`) four times with Intel MPI 4.1, each time using another one of Intel MPI's three algorithms (i.e., user-defined fixed value) and one time using the library's default algorithm. Even though Intel MPI's auto-selected algorithm is the best choice for small message sizes (it selects algorithm 2), for large message sizes Intel MPI's `MPI_Gather` auto-selected algorithm is far from optimal (it selects algorithm 3). As is indicated by these results, expert recommendations regarding the algorithms to use for collective operations could result in better application performance as compared to that achieved using the MPI library's auto-selection strategy.

To tune an MPI library for a specific application, MPI Advisor detects the algorithm used for each collective operation employed by the application and determines if it is the best choice. For each collective operation for which an alternate algorithm will provide improved performance, MPI Advisor recommends the alternate algorithm and provides the user with instructions on how to select this algorithm. To accomplish this, during the data collection phase, MPI Advisor records, via mpiP, the execution time and message size of each collective operation employed by the application and, using `MPI_T`, identifies the algorithm that was employed. Next, for each of these operations, using the table that was built at installation time using the CE script, which includes the execution times of every collective operation algorithm in each MPI library installed on the system, it identifies the best algorithm to use on the target architecture for each message size of interest. If there are collective operations for which the application should use different algorithms, MPI Advisor outputs related recommendations.

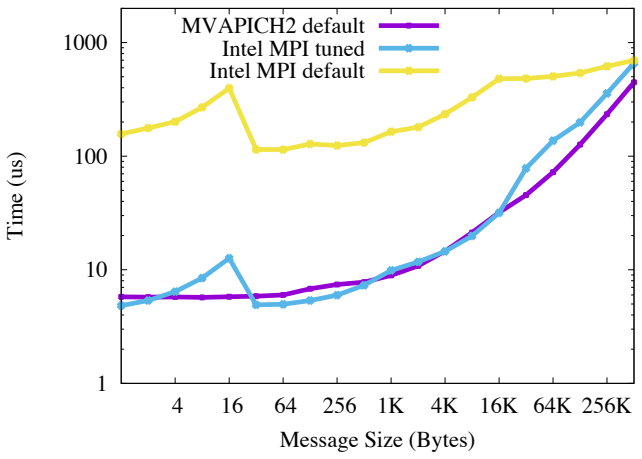


Figure 4: 256-core `MPI_Bcast` performance of OMB (`osu_bcast`) with default and tuned algorithms in Intel MPI *vs.* default algorithm in MVAPICH2.

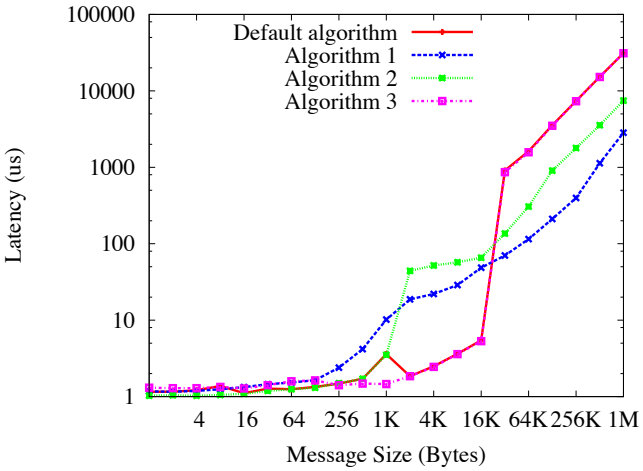


Figure 5: 256-core `MPI_Gather` latency of OMB (`osu_gather`) with each algorithm in Intel MPI (user-defined) *vs.* default algorithm (auto-selected).

4.3 Mapping of MPI Tasks to Cores

Each MPI library provides its own default strategy for mapping tasks to sockets and cores. There is no single best strategy since the optimal mapping is fairly dependent on application properties. Since the mapping determines the proximity of the root process (in general, task 0) to the host channel adapter (HCA) card, it can impact the efficiency with which task 0 communicates with other MPI tasks.

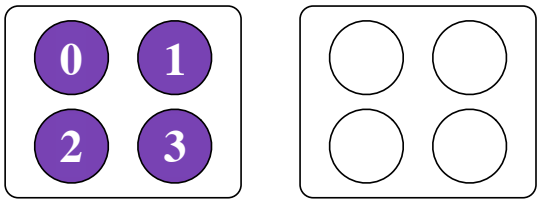
For hybrid applications (MPI+OpenMP), the default mappings provided by MVAPICH2 and Open MPI often do not deliver the best performance because all of the threads associated with a MPI task are mapped to the same core. Therefore, hybrid application performance may be drastically affected by this mapping. For example, assume that a hybrid code with four MPI tasks, each with two OpenMP threads, is executed on a cluster with nodes that are each comprised of two 4-core processors on sockets S1 and S2, and one HCA on S2. In this case, by default, MVAPICH2, Intel MPI, and Open MPI define different tasks-to-cores mappings. As shown in Figure 6: *a)* MVAPICH2 maps the four tasks to S1, *b)* Open MPI (version 1.7.4 and higher) maps each pair of tasks to the first two cores of S1 and S2, and *c)* Intel MPI maps each task to a pair of cores (one for each OpenMP thread) on S1 and S2. Since MVAPICH2 and Open MPI

utilize only four of the eight cores, each core will be over-subscribed by the OpenMP threads. In contrast, because the latter mapping is the only one that utilizes all cores on the node and assigns task 0 to a core that is close to the HCA card, it is likely that Intel MPI will provide the best performance.

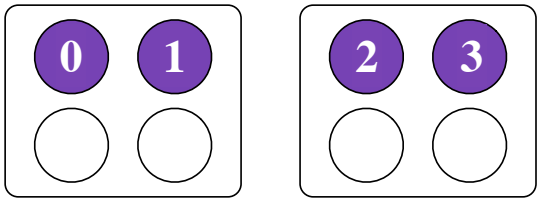
Although, MPI libraries provide parameters that can be used to change the tasks-to-cores mapping, identifying and effecting a suitable mapping requires knowledge regarding the node architecture and the related parameter settings. This is not a trivial task for the average user.

To recommend a better tasks-to-cores binding, MPI Advisor detects the mapping being used by the input application and determines if it is the most suitable. If it is not, MPI Advisor provides the user with the details of the current mapping and recommends related parameter settings that can be utilized to affect an alternate mapping, which may improve performance. To do this, during MPI Advisor's data collection phase: *1)* using the POSIX function `getenv()`, the `OMP_NUM_THREADS` variable is read to identify the number of MPI tasks and OpenMP threads executing on each node, *2)* by calling `MPI_Get_processor_name()`, the hostname associated with each task is identified to determine the number of MPI tasks on each node (the number with the same hostname), and *3)* using `hwloc`, the mapping used by the application is identified.

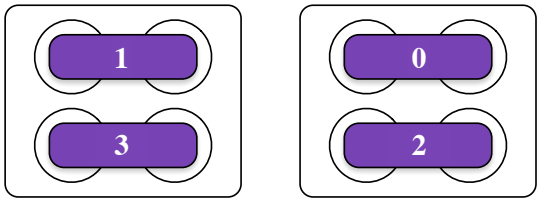
During the analysis phase, this information is used to determine if one of the following situations exist: *1)* the node is under- or over-utilized, *i.e.*, the number of resident OpenMP threads is smaller or larger than the number of resident cores, *2)* the cores are under- or over-subscribed, *i.e.*, the number of cores assigned to an MPI task is greater or less than the number of OpenMP threads executing on a node, and *3)* the root process is not close to the HCA card. Any one of these situations triggers MPI Advisor to output a recommendation.



(a) MVAPICH2



(b) Open MPI



(c) Intel MPI

Figure 6: Default mapping of 4 multi-threaded MPI tasks inside a dual-socket node equipped with two 4-core processors according to each MPI library.

4.4 Infiniband Transport: RC and UD

To provide high-throughput communication InfiniBand is the interconnection network of choice of many HPC systems. Of the four transports defined by the InfiniBand specification, reliable connection (RC) and unreliable datagram (UD) [11] are the most commonly used, with RC being the primary (default) transport over InfiniBand for MVAPICH2, Intel MPI, and Open MPI. As shown in Table 4, RC and UD differ in several ways. But, one of the most important is in terms of their relative memory footprints. In general, the default MPI library settings usually cause RC memory requirements per node to grow linearly with the number of MPI tasks with which it communicates. This is despite the fact that InfiniBand provides mechanisms to reduce the footprint of RC [10] and MPI implementations provide a way to make use of them. In contrast, UD's memory usage is near constant.

To select the Infiniband transport method, MPI Advisor assumes that the input application is using RC by default. To determine which transport is more suitable, during the data collection phase, MPI Advisor calls `MPI_Comm_size()` to identify the number of tasks employed by the application. If this number exceeds 4,096, MPI Advisor recommends that UD be employed (in accordance with [11]) and provides instructions on how to make this change.

5. PERFORMANCE EVALUATION

To demonstrate the efficacy of MPI Advisor recommendations for optimization, we conducted experiments on TACC's Stampede and Maverick clusters. Stampede has 6,400 dual-socket 8-core Sandy-Bridge E5-2680 compute nodes each with 32 GB of memory. Maverick contains 132 dual-socket 10-core Ivy-Bridge E5-2680v2 compute nodes each with 256 GB of memory. On both clusters, the nodes are interconnected by InfiniBand HCAs in FDR mode [22] and the operating system is Linux (kernel version 2.6.32-358.el6, CentOS distribution version 6.4). We used MVAPICH2 version 1.9a2 and Intel MPI version 4.1.0.030 for this evaluation since these are the two MPI libraries most commonly used on Stampede and Maverick. The case study applications were compiled using the Intel Compiler version 13.0.2.146.

Characteristic	RC	UD
Scalability	n^2	n
Corrupt data detected	yes	yes
Delivery guarantee	yes	no
Ordering guarantee	yes	no
Data loss detection	yes	no
Error recovery	yes	no
Send / RDMA write	yes	no
Receive / RDMA read	yes	no
Max message size	1 GB	MTU

Table 4: Infiniband RC and UD transport characteristics.

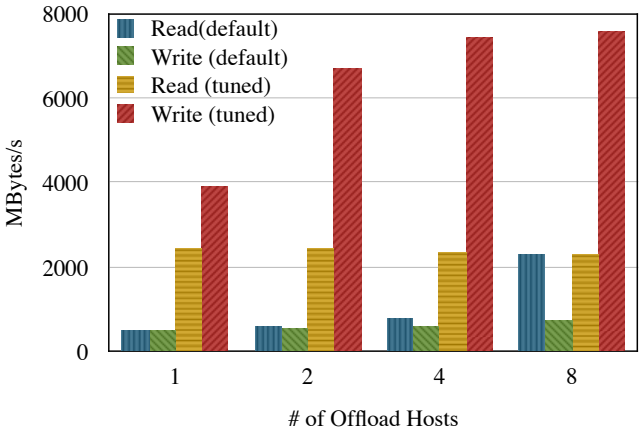


Figure 7: Read and write performance of default and tuned MVAPICH2's eager threshold settings.

5.1 Point-to-Point Protocol Threshold

To demonstrate the performance improvement achieved by tuning the point-to-point protocol threshold we selected a benchmark that mimics a modified version of the CFOUR Quantum Chemistry Application[7] [7]. This version, provided by Simmons and Schulz [20] augments CFOUR by using the open-source GRVY toolkit library[8] to convert disk transactions into distributed memory transactions using MPI.

The benchmark reads and writes fixed records in random order as part of an out-of-core solve procedure. For this case study the offload hosts are placed on different nodes to maximize the available memory. By using MPI Advisor, we discovered that the messages were mainly point-to-point with sizes around 256 KB or less. Following the advice provided by the tool and shown in Listing 1, we changed the value of the eager threshold of MVAPICH2 from 17 KB to 256 KB by setting the value of `MV2_IBA_EAGER_THRESHOLD` to 262144. Running the micro-benchmark with the 256 KB threshold yielded a significant improvement for write and read operations. The results, presented in Figure 7, report the aggregate write and read speeds for the default and tuned MVAPICH2 settings.

[7]http://www.cfour.de/

[8]https://red.ices.utexas.edu/projects/software/wiki/GRVY

```
Eager vs. rendezvous program details:
- Number of call sites that used MPI_Send: 1
- Maximum median size (bytes) of messages sent
  through MPI_Send: 131072
- Eager threshold of MPI library (bytes): 17408
- For more details on the messages sent,
  consult the mpiP report: ./cfour.88089.1.mpiP

Eager vs. rendezvous suggestions:
- POSSIBLE OPTIMIZATION: The maximum of the
  median messages sent is 131072 bytes, but
  the eager threshold of the MPI Library is
  17408. Consider increasing the eager thres-
  hold to a value higher than 131072 bytes.
- WARNING: Increasing the eager threshold will
  also increase MPI library memory footprint.

MVAPICH2 command that can be used to change the
  eager threshold:
- MV2_IBA_EAGER_THRESHOLD=<nbytes>
- Related documentation can be found in:
  http://mvapich.cse.ohio-state.edu/support/
```

Listing 1: MPI Advisor recommendation for tuning the point-to-point eager *vs.* rendezvous protocol threshold for a benchmark that mimics CFOUR.

5.2 Algorithms for Collective Operations

To illustrate the performance benefit obtainable from tuning collective operations, we use the ASP [9] application, which is a parallel implementation of the Floyd-Warshall algorithm used to solve the all-pairs shortest-path problem. ASP mainly uses **MPI_Bcast** and changes the root of the broadcast operation for each iteration. On Maverick Intel MPI outperforms MVAPICH2 for ASP. The default configuration of MVAPICH2 is tuned based on OMB, which always uses the same root for collective operations. Following the tool's recommendation, which is shown in Listing 2, we were able to improve the performance of ASP by 8.3%. Table 5 provides the results obtained by MVAPICH2 with tuned and default settings, and Intel MPI on the Maverick cluster using 80 MPI tasks. MPI Advisor does not provide any recommendation for Intel MPI because its default value is already tuned.

MVAPICH2 Default	MVAPICH2 Tuned	Intel MPI Default
24.45	22.41	22.38

Table 5: ASP execution time (seconds) on 80 cores.

5.3 Mapping of MPI Tasks to Cores

To illustrate the benefits of using MPI Advisor to tune the MPI tasks-to-cores mapping we use HPCG [3]. HPCG is an application that is used as an alternative ranking of the TOP500 list[9] and can be used only with MPI or with

[9]http://www.top500.org/

```
Collective program details:
- Number of call sites that used MPI_Bcast: 1
- Average MPI_Bcast message sizes:
  * Callsite ID: 2, size: 2097152
- MPI_Bcast algorithm employed: 5
- Root is changing
- For more details on the messages sent,
  consult the mpiP report: ./asp.8.22585.1.mpiP

Collective suggestions:
- POSSIBLE OPTIMIZATION: The algorithm being
  employed for MPI BCAST may not provide the
  best performance for the messages being sent.
  * Consider changing to algorithm 2

MVAPICH2 command that can be used to change the
  MPI_Bcast algorithm:
- MV2_INTER_BCAST_TUNING=<1-9>
```

Listing 2: MPI Advisor recommendation for selecting the appropriate collective operation algorithm for ASP.

MPI+OpenMP. The hybrid version is the default build and using the correct mapping can improve its performance. MPI Advisor detected suboptimal mappings and, as shown in Listing 3, recommended how to select a near-optimal mapping. For HPCG with MVAPICH2, MPI Advisor detected that task 0, which does the communication, was mapped to socket one. Table 6 gives the performance of HPCG using MVAPICH2 on 16 nodes of Stampede (256 cores). The default mapping for the hybrid version is 2 MPI tasks and 8 OpenMP threads per node. MPI Advisor's recommendation is to: *1)* map MPI tasks at the socket level instead of at the core level, and *2)* map task 0 to the socket closest to the Infiniband card. In this case, the performance of the hybrid version increased from 26.05 GFLOP/s to 38.85 GFLOP/s.

5.4 Infiniband Transport: RC and UD

To illustrate the benefits of MPI Advisor's recommendation on selecting the Infiniband transport method, we used the SMG2000[10] benchmark, a parallel semi-coarsening multi-grid solver. SMG2000 may be run with different node counts depending on the size of the matrix being solved. As shown in Listing 4, MPI Advisor recommends using UD when the application uses Intel MPI and has 4,096 or more MPI tasks. Implementing this recommendation, we were able to improve the global performance by 29%. Figure 8 shows the execution times of the three phases of SMG2000 for RC and UD. The setup phase is a communication-intensive phase mainly composed of small point-to-point operations. The mpiP profile shows that there are more than 1,060 million `MPI_Isend` calls during this phase. The execution time with RC is over a minute, while with UD it is approximately 24 seconds, yielding an improvement in the setup phase of 61%. There is no improvement in the performance of the two other phases, mainly because they are both computation intensive.

[10]https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/smg/

```
Affinity-related program details:
- Number of MPI tasks launched: 8
- Number of MPI tasks running on each node: 2
- Number of cores on each node: 16
- Number of OpenMP threads per MPI task: 8
- Number of cores available to each MPI task: 1
- Rank 0: binding restricted to HCA socket
- HCA is located on socket: 1
- 8 OpenMP thread(s) is (are) on the same core.

Affinity-related suggestions:
- POSSIBLE OPTIMIZATION: The number of OpenMP
    threads exceeds the number of cores
  available to the MPI tasks.
- Consider reducing the number of parallel
  threads launched or change the affinity
  settings.

MVAPICH2 variables that can be used to modify
  the mapping:
- MV2_CPU_BINDING_POLICY
- MV2_CPU_BINDING_LEVEL
- MV2_CPU_MAPPING
- Related documentation can be found in:
  http://mvapich.cse.ohio-state.edu/support/
```

Listing 3: MPI Advisor recommendation for improving tasks-to-cores mapping for HPCG.

Default Mapping	Recommended Mapping
26.05	38.85

Table 6: Performance (GFLOP/s) of HPCG Hybrid on 256 processes with different mapping.

6. RELATED WORK

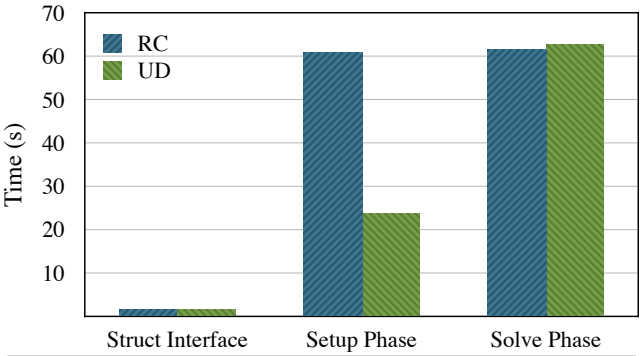
There are several tools and frameworks to help users achieve better application performance by selecting a suitable set of parameters to configure the MPI runtime environment. However, because of the widespread use of multiple different implementations of MPI, developing a tool that supports many implementations is complex. The goal of the MPI Advisor tool, which is introduced in this paper, is to enable automated implementation of an important subset of optimizations across as many different versions of MPI as

```
Infiniband transport selection details:
- Number of MPI Tasks launched: 4096

Infiniband transport suggestions:
- POSSIBLE OPTIMIZATION: 4K MPI tasks are being
  employed
- Consider using UD instead of RC

Intel MPI variables that can be used to modify
  the Infiniband transport:
- I_MPI_DAPL_UD_PROVIDER=ofa-v2-mlx4_0-1u
- I_MPI_DAPL_UD=enable
- Related documentation can be found in:
  https://software.intel.com/en-us/articles/
  intel-mpi-library-documentation
```

Listing 4: MPI Advisor recommendation for selecting Infiniband transport method for SMG2000.



Transport Method	Struct Interface	Setup Phase	Solve Phase	Total Runtime
RC	1.42	61.04	61.63	124.09
UD	1.44	23.63	62.71	87.78

Figure 8: Execution time (seconds) breakdown of SMG2000 running on 4,096 cores with Intel MPI and either the RC or UD Infiniband transport method.

possible. The current implementation automates profiling, analyses, and generation of recommendations for optimization of MPI communication. It also presents the recommendations to the user in a way that enhances his/her knowledge of the efficient use of MPI. The tool works with two MPI implementations, gathers data from both `PMPI` and `MPI_T`, and requires only a single execution of the program to gather the measurements necessary to generate recommendations. Accordingly, we restrict this review of related research to those tools that carry the MPI-performance optimization process through the analysis phase.

The tool most similar in capabilities to MPI Advisor is Periscope in conjunction with the Periscope's MPI Tools Interface [6]. Using the patches in `MPI_T` in the MPICH implementation of MPI, a plugin for Periscope was introduced to gather information on point-to-point calls and collective operations. From the data gathered and analyses performed, the plugin indicates the best eager threshold values for inter- and intra-node MPI communication in the given application. The Periscope Tuning Framework [12] provides guidance on manually applying optimizations. Its implementation depends on Periscope tool plugins [13] that recommend a variety of optimizations such as compiler flags, MPI runtime environment configuration (buffer sizes, collective protocol, number of tasks, process affinity, etc.), and application- and library-specific parameters. In addition, the framework generates a communication matrix that defines a host file with a recommended topology for the target system. Unlike MPI Advisor, Periscope does not provide optimization for collective algorithms or for tasks-to-cores mapping. More importantly, unlike MPI Advisor, Periscope requires multiple executions of an application to collect performance data – this makes it expensive to apply to production-scale codes. And, it appears that on most systems, execution of Periscope requires system-level access privilege.

ATune [15] is an automatic performance-tuning tool that applies machine-learning techniques to specifically identify a select number of optimal MPI settings for an application. ATune was trained on the NAS benchmarks. Like MPI Advisor, it requires only one run of the target application. From experimental evaluations using its Open MPI implementation, ATune's most significant improvements resulted from selecting appropriate affinity assignments for MPI tasks with respect to cores on compute nodes. The breadth of applicability of training with the NAS benchmarks is an open question and ATune is restricted to Open MPI. In more recent work, Pellegrini, et al. [16] apply autotuning to find runtime parameters suitable for optimizing the performance of an application. The approach consists of finding parameters that improve the performance of a set of kernels on the target cluster. The kernels selected apply methods common in many HPC applications (*e.g.*, stencil operations, Fourier transforms, matrix operations, etc.). The kernels are initially executed with varying runtime parameters on the target architecture. Then analysis of variance (ANOVA) is used to determine the set of parameters with the most significant impact on performance. To ensure that the achieved performance is close to the maximum that can be obtained with a subset of the runtime parameters, each of the selected computational kernels was tested with 1,000 distinct parameter combinations. Autotuning requires many executions of the code, which makes its application to production-scale codes very expensive.

OPTO [2] is an optimization tool that works specifically with Open MPI to optimize the MPI runtime environment. It supports optimization of Infiniband parameters among other Open MPI-specific parameters. OPTO operates by running benchmarks to characterize the execution environment and generating a configuration file that should be used on subsequent executions in the given environment.

Intel provides a performance-tuning utility named VTune [17], which can be used to optimize the library parameters of the Intel implementation of MPI either at the cluster or application level. Cray provides CrayPat, which includes a comprehensive set of performance reports and automatic MPI rank order analysis[11].

There are several tools that provide profiles of MPI communication operations and present visual displays and data analyses to the user. These tools are mostly oriented towards helping users identify hotspots and load imbalance caused by the use of MPI. Recommendations to alleviate these hotspots include modification to the application source code. The most widely used of these tools include Vampir [14], TAU [19], Open|SpeedShop [18], and Scalasca [5]. Scalasca also has been applied to several MPI performance-optimization issues by its developers. The most recent of these deals with detecting wait states [5]; a complete list is on the Scalasca website[12]. Note that, unlike MPI Advisor, most of these profilers use only `PMPI` to obtain performance data, which limits the range of analyses that can be provided to users. And, most do not overlap significantly with MPI Advisor in terms of the targeted optimizations.

7. CONCLUSION AND FUTURE WORK

This paper presented the first version of a methodology and tool called MPI Advisor, which aims to automate MPI library performance optimization, while, at the same time, informing users about correct usage of MPI libraries. The effectiveness of the methodology and tool was demonstrated on benchmark applications.

MPI Advisor is an ongoing project. The current version supports optimization for only four MPI functionalities: point-to-point tuning, collective tuning, MPI task mapping, and the selection of transport mode for Infiniband clusters. However, we plan to add capabilities for optimization of additional functionalities, including topology-aware mapping by using the Treematch algorithm [8], and for generation of a detailed report to explain the analyses and recommendations made by the tool. This initial release works mainly with the MVAPICH2 and Intel MPI libraries but we are planning to extend it to other popular MPI libraries such as Open MPI and MPICH. A long-term project is to automate implementation of the optimization recommendations as much as is possible integrating it into the PerfExpert framework [4].

8. ACKNOWLEDGEMENTS

This work was partially funded by the National Science Foundation Stampede grant through The University of Texas at Austin grant #UTA13-000072. The authors acknowledge the Texas Advanced Computing Center at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper

[11] http://docs.cray.com/books/S-2376-610/

[12] http://www.scalasca.org

and are especially grateful to Todd Evans for providing the Stampede monitoring data. Also, the authors are thankful to Brice Goglin for his guidance on using `hwloc`.

9. REFERENCES

[1] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In *Proceedings of the Euromicro Conference on Parallel, Distributed and Network-Based Computing*, 2010.

[2] M. Chaarawi, J. Squyres, E. Gabriel, and S. Feki. A Tool for Optimizing Runtime Parameters of Open MPI. In *Proceedings of the European PVM/MPI Users' Group Meeting*, pages 210–217, 2008.

[3] J. Dongarra and M. A. Heroux. Toward a New Metric for Ranking High Performance Computing Systems. Technical Report SAND2013-4744, Sandia National Laboratories, 2013.

[4] L. Fialho and J. Browne. Framework and Modular Infrastructure for Automation of Architectural Adaptation and Performance Optimization for HPC Systems. In *Proceedings of the International Conference on Supercomputing*, pages 261–277, 2014.

[5] M. Geimer, P. Saviankou, A. Strube, Z. Szebenyi, F. Wolf, and B. J. N. Wylie. Further Improving the Scalability of the Scalasca Toolset. In *Proceedings of the PARA 2010: State of the Art in Scientific and Parallel Computing*, pages 463–473, 2012.

[6] M. Gerndt and M. Ott. Automatic performance analysis with Periscope. *Concurrency and Computation: Practice and Experience*, 22(6):736–748, 2009.

[7] M. E. Harding, T. Metzroth, J. Gauss, and A. A. Auer. Parallel Calculation of CCSD and CCSD(T) Analytic First and Second Derivatives. *Journal of Chemical Theory and Computation*, 4(1):64–74, 2008.

[8] E. Jeannot, G. Mercier, and F. Tessier. Process placement in multicore clusters: Algorithmic issues and practical techniques. *IEEE Transactions on Parallel and Distributed Systems*, 25(4):993–1002, 2014.

[9] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. *SIGPLAN Notices*, 34(8):131–140, 1999.

[10] M. J. Koop, J. K. Sridhar, and D. K. Panda. Scalable MPI design over InfiniBand using eXtended Reliable Connection. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 203–212, 2008.

[11] M. J. Koop, S. Sur, Q. Gao, and D. K. Panda. High Performance MPI Design Using Unreliable Datagram for Ultra-scale InfiniBand Clusters. In *Proceedings of the International Conference on Supercomputing*, 2007.

[12] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin. AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications. In *Proceedings of the PARA 2012: State of the Art in Scientific and Parallel Computing*, pages 328–342, 2013.

[13] R. Mijaković, A. P. Soto, I. A. C. Ureña, M. Gerndt, A. Sikora, and E. César. Specification of Periscope Tuning Framework Plugins. In *Proceedings of the International Conference on Parallel Computing*, pages 123–132, 2013.

[14] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and Analysis of MPI Resources. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 69–80, 1996.

[15] S. Pellegrini, T. Fahringer, H. Jordan, and H. Moritsch. Automatic Tuning of MPI Runtime Parameter Settings by Using Machine Learning. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 115–116, 2010.

[16] S. Pellegrini, J. Wang, T. Fahringer, and H. Moritsch. Optimizing MPI Runtime Parameter Settings by Using Machine Learning. In *Proceedings of the European PVM/MPI Users' Group Meeting*, pages 196–206, 2009.

[17] J. Reinders. *VTune Performance Analyzer Essentials*. Intel Press, Hillsboro, 1st edition, 2005.

[18] M. Schulz, J. Galarowicz, D. Maghrak, and W. Hachfeld. Open | SpeedShop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2–3):105–121, 2008.

[19] S. Shende and A. D. Malony. The Tau Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[20] C. S. Simmons and K. W. Schulz. A Distributed Memory Out-of-core Method on HPC Clusters and Its Application to Quantum Chemistry Applications. In *Proceedings of the Conference of the Extreme Science and Engineering Discovery Environment*, 2012.

[21] J. S. Vetter and M. O. McCracken. Statistical Scalability Analysis of Communication Operations in Distributed Applications. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, 2001.

[22] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni, and D. K. Panda. Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems. In *Hot Interconnects*, pages 48–55, 2012.

[23] J. West, T. Evans, W. L. Barth, and J. Browne. Multilevel Workload Characterization With Applications in High Performance Computer Systems Management. *Submitted to the 2015 IEEE Int. Symposium on Workload Characterization*, 2015.