A Tool for Optimizing Runtime Parameters of Open MPI

Mohamad Chaarawi[1,2], Jeffrey M. Squyres[2], Edgar Gabriel[1], Saber Feki[1]

[1] Parallel Software Technologies Laboratory,
Department of Computer Science, University of Houston,
{mschaara, gabriel, sfeki}@cs.uh.edu

[2] Cisco Systems, San Jose, CA USA
jsquyres@cisco.com

Abstract. Clustered computing environments, although becoming the predominant high-performance computing platform of choice, continue to grow in complexity. It is relatively easy to achieve *good* performance with real-world MPI applications on such platforms, but obtaining the best possible MPI performance is still an extremely difficult task, requiring painstaking tuning of all levels of the hardware and software in the system. The Open Tool for Parameter Optimization (OTPO) is a new framework designed to aid in the optimization of one of the key software layers in high performance computing: Open MPI. OTPO systematically tests large numbers of combinations of Open MPI's run-time tunable parameters for common communication patterns and performance metrics to determine the "best" set for a given platform. This paper presents the concept, some implementation details and the current status of the tool, as well as an example optimizing InfiniBand message passing latency by Open MPI.

1 Introduction

In the current top 500 list [10], clustered high performance computing systems clearly dominate from the architectural perspective. Off-the-shelf components make clusters attractive for low-end budgets as well as for large scale installations, since they offer the opportunity to customize the equipment according to their needs and financial constraints. However, the flexibility comes at the price: the performance that end-users experience with real-world applications deviates significantly from the theoretical peak performance of the cluster. This is mainly due to the fact, that each system represents a nearly unique execution environment. Typically, neither software nor hardware components have been hand-tuned to that particular combination of processors, network interconnects and software environment.

In order to optimize the performance of a particular system, research groups have turned to extensive pre-execution tuning. As an example, the ATLAS project [12] evaluates in a configure step a large number of implementation possibilities for the core loops of the BLAS routines. Similarly, the Automatically

Tuned Collective Communication project [8] incorporates an exhaustive search in order to determine the best performing algorithms for a wide range of message lengths for MPI's collective operations. The FFTW library [1] tunes fast fourier transform operations (FFT) in a so-called planner step before executing the FFT operations of the actual application.

One critical piece of software has however not been systematically approached in any of these projects. MPI libraries represent the interface between most parallel applications and the hardware today. Libraries such as MPICH [4] and Open MPI [2] provide flexible and tunable implementations, which can be adapted either at compile or at runtime to a particular environment. In this paper, we introduce OTPO (Open Tool for Parameter Optimization), a new tool developed in partnership between Cisco Systems and the University of Houston. OTPO is an Open MPI specific tool aiming at optimizing parameters of the runtime environment exposed through the MPI library to the end-user application. Since parameters might expose explicit or implicit dependencies among each other, some of them possibly even unknown to the module developers, OTPO aims at systematically discovering those hidden dependencies and the effect they have on overall performance, such as the point-to-point latency or bandwidth. We present the current status of OTPO and the ongoing work.

The rest of the paper is organized as follows: Sec. 2 presents the concept and the architecture of OTPO. Sec. 3 discusses some implementation details of OTPO. In Sec. 4, we show how OTPO is used to explore the parameter space of Open MPI's short message protocol in order to minimize latency on InfiniBand networks. Finally, Sec. 5 summarizes the paper and discusses ongoing work.

2 Concept

Open MPI [2] is an open source implementation of the MPI-1 [6] and MPI-2 [7] specifications. The code is developed and maintained by a consortium consisting of 14 institutions[3] from academia and industry. The Open MPI design is centered around the Modular Component Architecture (MCA), which is the software layer providing management services for Open MPI frameworks. A framework is dedicated to a single task, such as providing collective operations (i.e., the COLL framework) or providing data transfer primitives for a particular network interconnect (i.e., the Byte Transfer Layer framework – BTL). Each framework will typically have multiple implementations available in the form of modules ("plugins") that can be loaded on-demand at run time. For example, BTL modules include support for TCP, InfiniBand, Myrinet, shared memory, and others.

Among the management services provided by the MCA is the ability to accept run-time parameters from higher level abstractions (e.g., `mpirun`) and pass them down to the according frameworks. MCA runtime parameters give system administrators, end-users and developers the possibility to tune the performance of their applications and systems without having to recompile the MPI library.

[3] As of January, 2008.

Examples for MCA runtime parameters include the values of cross-over points between different algorithms in a collective module, or modifying some network parameters such as internal buffer sizes in a BTL module. Due to its great flexibility, Open MPI developers made extensively use of MCA runtime parameters. The current development version of Open MPI has multiple hundred MCA runtime parameters, depending on the set of modules compiled for a given platform. While average end-users clearly depend on developers setting reasonable default values for each parameter, some end-users and system administrators might explore the parameter space in order to find values leading to higher performance for a given application or machine.

OTPO is a tool aiming at optimizing parameters of the runtime environment exposed through the MPI library to the end-user application. OTPO is based on a highly modular concept, giving end-user the possibility to provide or implement their own benchmark for exploring the parameter space. Depending on the purpose of the tuning procedure, most often only a subset of the runtime parameters of Open MPI will be tuned at a given time. As an example, users might choose to tune the networking parameters for a cluster, optimizing the collective operations in a subsequent run etc. Therefore, one of the goals of OTPO is to provide a flexible and user friendly possibility to input the set of parameters to be tuned. OTPO supports testing two general types of MCA parameters:

1. Single-value parameters: these parameters represent an individual value, such as an integer.
2. Multiple-value parameters: these parameters are composed of one or more sub-parameters, each of which can vary.

From a higher level perspective, the process of tuning runtime parameters is an empirical search in a given parameter space. Depending on the number of parameters, the number of possible values for each parameter, and dependencies among the parameters themselves, the tuning process can in fact be very time consuming and complex. Thus, OTPO is based on a library incorporating various search algorithms, namely the Abstract Data and Communication Library (ADCL) [3]. ADCL is a runtime optimization library giving applications the possibility to register alternative implementations for a given functionality. Using various search algorithms, the library will evaluate the performance of some/each implementation(s) provided, and choose the version leading to the lowest execution time. Furthermore, the application has the possibility to characterize each implementation using a set of attributes. These attributes are the basis of some advanced search operations within ADCL in order to speed up the tuning process. This mechanism has been used by OTPO for registering and maintaining the status of different MCA parameters.

3 Implementation

Upon start of an optimization run, OTPO parses an input file and creates a global structure that holds all the parameters and their options. OTPO then registers a function for each possible combination of MCA parameters which satisfies the Reverse Polish Notations (RPNs) conditions specified in the parameter file with ADCL. The function registered by OTPO first forks a child process. The child process sets the parameters in the environment that need to be provided to the `mpirun` command, such as the number of processes, the MCA parameters and values, and the application/benchmark to run. Finally, the child launches `mpirun` with the argument set. The parent process waits for the child to complete, and checks if the test was successful. If the child succeeded, the parent will update the current request with the value (e.g., latency) that the child measured for the current parameter values. If the child does not complete within a user specified timeout, the parent process kills it, and updates the request with an invalid value.

When measurements for all combinations of parameter values have been updated by ADCL, OTPO gathers the results and saves them to a file. Each run of OTPO produces a file with a time stamp that contains the best attribute combinations. The result file contains the set of best measured values, the number of combinations that produced these values, and the parameter value combinations themselves. The result file might be large, having thousands of different parameter combinations.

These results files produced by the first version of OTPO are intented to be intermediate results. Currently ongoing work focuses on presenting the results in an intuitive and visual manner.

3.1 OTPO Parameter File

The OTPO parameter file describes the MCA parameters and potential values to be tested. In order to provide a maximum flexibility to the end-user, the parameters can be described by various options, e.g. depending on whether a parameter can have continues values, certain discrete values, or whether the value consists of different strings. Each line in the parameter file specifies a single parameter by giving the name of the parameter and some options, the options being one of the following:

– `d default_value`: specifies a default value for this parameter.
– `p <list of possible values>`: explicitly specify the list of possible values for the parameter.
– `r start_value end_value`: specify the start and end value for the parameter.
– `t traversal_method <arguments>`: specifies the method to traverse the range of variables for the parameter. The first version of OTPO only includes one method: "increment," which takes the operator and the operand as arguments.
– `i rpn_expression`: RPN condition that the parameter combinations must satisfy.
– `v`: specifies if the parameter is virtual, which means that it will not be set as an environment variable, but will be part of another parameter.

– a `format_string`: specifies that the parameter is an aggregate of other parameters in a certain format. Each sub parameter is surrounded by dollar signs ($) in the format string.

4 Performance Evaluation

This section presents an example using OTPO to optimize some of the InfiniBand parameters of Open MPI on a given platform. We therefore first describe Open MPI's InfiniBand support and some of its run-time tunable parameters, then present the results of the optimization using OTPO.

4.1 InfiniBand Parameters in Open MPI

Open MPI supports InfiniBand networks through a Byte Transfer Layer (BTL) plugin module named `openib`. BTL plugins are the lowest layer in the Open MPI point-to-point communication stack and are responsible for actually moving bytes from one MPI process to another. The `openib` BTL has both single- and multiple-value parameters that can be adjusted at run-time.

There are more than 50 MCA parameters that are related to the `openib` BTL module, all of which can be modified at runtime. Open MPI attempts to provide reasonable default values for these parameters, but every application and every platform is different: maximum performance can only be achieved through tuning for a specific platform and application behavior.

MPI processes communicate on InfiniBand networks by setting up a pair of queues to pass messages: one queue for sending and one queue for receiving. InfiniBand queues have a large number of attributes and options that can be used to tailor the behavior of how messages are passed. Starting with version v1.3, Open MPI exposes the receive queue parameters for short messages through the multiple-value parameter `btl_openib_receive_queues` (long messages use a different protocol and are governed by a different set of MCA parameters). Specifically, this MCA parameter is used to specify one or more receive queues that will be setup in each MPI process for InfiniBand communication. There are two types of receive queues, each of which have multiple sub-parameters. It is however outside of the scope of this paper to give detailed and precise descriptions of the MCA parameters used. The parameters are:

1. "Per-peer" receive queues are dedicated to receiving messages from a single peer MPI process. Per-peer queues have two mandatory sub-parameters (*size* and *num_buffers*) and three optional sub-parameters (*low_watermark*, *window_size*, and *reserve*).
2. "Shared" receive queues are shared between all MPI sending processes. Shared receive queues have the same mandatory sub-parameters as per-peer receive queues, but have only two optional sub-parameters (*low_watermark* and *max_pending_sends*).

The `btl_openib_receive_queues` value is a colon-delimited listed of queue specifications specifying the queue type ("P" or "S") and a comma-delimited list of the mandatory and optional sub-parameters. For example:

```
P,128,256,192,128:S,65535,256,128,32
```

will instantiate one per-peer receive queue for each inbound MPI connection for messages that are ≤ 128 bytes, and will setup a single shared receive queue for all messages that are > 128 bytes and $\leq 65,535$ bytes (messages longer than 65,535 bytes will be handled by the long message protocol).

Another good example for how to explore the parameter space by OTPO are the tunable values controlling Open MPI's use of RDMA for short messages. Short message RDMA is a resource-intensive, non-scalable optimization for minimizing point-to-point short message latency. Finding a good balance between the desired level of optimization and the resources consumed by this optimization is exactly the kind of task that OTPO was designed for. Among the most relevant parameters with regard to RDMA operations are `btl_openib_ib_max_-rdma_dst_opts`, which limits the maximum number of outstanding RDMA operations to a specific destination; `btl_openib_use_eager_rdma`, a logical value specifying whether to use the RDMA protocol for eager messages; and `btl_-openib_eager_rdma_threshold`, only use RDMA for short messages to a given peer after this number of messages has been received from that peer. Due to space limitations, we will not detail all RDMA parameters or present RDMA results of the according OTPO runs.

4.2 Results

Tests were run on the shark cluster at the University of Houston. Shark consists of 24 dual-core 2.2GHz AMD Opteron nodes connected by 4x InfiniBand and Gigabit Ethernet network interconnects. The InfiniBand switch is connected to a single HCAs on every node, with an `active_mtu` of 2048 and an `active_speed` of 2.5 Gbps. OFED 1.1 is installed on the nodes. A pre-release version of Open MPI v1.3 was used to generate these results, subversion trunk revision 17198. A nightly snapshot of the trunk was used, and configured with debug disabled. All the tests were run with `mpi_leave_pinned` MCA parameter set to one. The benchmark used for tuning the parameters was NetPIPE [11].

OTPO was used to explore the parameter space of `btl_openib_receive_-queues` to find a set of values that yield the lowest half round trip short message latency. Since `receive_queues` is a multiple-value parameter, each sub-parameter must be described to OTPO. The individual sub-parameters become "virtual" parameters, each with a designated range to explore. OTPO was configured to test both a per-peer and a shared receive queue with the ranges listed in Table 1. Each sub-parameter spanned its range by doubling its value from the minimum to the maximum (e.g., 1, 2, 4, 8, 16, ...).

The parameters that are used are explained as follows:

– The size of the receive buffers to be posted.

Table 1. InfiniBand receive queue search parameter ranges.The "max pending sends" sub-parameter is only relevant for shared receive queues.

Sub-parameter	Range	Per-peer	Shared
Buffer size (bytes)	65,536 → 1,048,576	√	√
Number of buffers	1 → 1024	√	√
Low watermark (buffers)	32 → 512	√	√
Max pending sends	1 → 32		√

Table 2. OTPO results of the best parameter combinations.

Per Peer Queue		Shared Receive Queue	
Latency	Number of Combinations	Latency	Number of Combinations
$3.78\mu s$	3	$3.77\mu s$	1
$3.79\mu s$	3	$3.78\mu s$	4
$3.80\mu s$	15	$3.79\mu s$	18
$3.81\mu s$	21	$3.80\mu s$	32
$3.82\mu s$	31	$3.81\mu s$	69
$3.83\mu s$	34	$3.82\mu s$	69

– The maximum number of buffers posted for incoming message fragments.
– The number of available buffers left on the queue before Open MPI reposts buffers up to the maximum (previous parameter).
– The maximum number of outstanding sends that are allowed at a given time (SRQ only).

The parameter space from Table 1 yields, 275 for per peer queue and 825 for shared queue valid combinations (after removing unnecessary combinations that would lead to incorrect results). These combinations stressed buffer management and flow control issues in the Open MPI short message protocol when sending 1 byte messages. It took OTPO 3 minutes to evaluate the first case by invoking NetPIPE for each parameter combination and 9 minutes for the second case. Note that NetPIPE runs several ping-pong tests and reports half the average round-trip time. OTPO sought parameter sets that minimized this value.

The results are summarized in Table 2, and reveal a small number of parameter sets that resulted in the lowest latency ($3.78\mu s$ and $3.77\mu s$). However, there were more parameter combinations leading to results within $0.05\mu s$ of the best latency. These results highlight, that typically, the optimization process using OTPO will not deliver a single set of parameters leading to the best performance, but will result in groups of parameter sets leading to similar performance.

5 Summary

In this paper we presented OTPO, a tool for optimizing Open MPI runtime parameters. The tool gives interested end-users and system administrators the possibility to "personalize" their Open MPI installation. OTPO has been successfully used to optimize the network parameters of the `openib` InfiniBand communication module of Open MPI in order to minimize the communication latency.

The currently ongoing work on OTPO includes multiple areas. As of today, OTPO only supports NetPIPE as the application benchmark. However, we plan to add more benchmarks to be used with OTPO such as the IMB [5]and SKaMPI [9] benchmarks in order to optimize collective modules. Some of the benchmarks will also require OTPO to support additional optimization metrics, such as bandwidth or memory usage. The foremost goal however is to develop a result gathering tool that takes the results file produced by OTPO and presents it to the user in a more readable and interpretable manner.

Acknowledgments. This research was funded in part by a gift from the Silicon Valley Community Foundation, on behalf of the Cisco Collaborative Research Initiative of Cisco Systems.

References

1. Matteo Frigo and Steven G. Johnson. The Design and Implementation of FFTW3. *Proceedings of IEEE*, 93(2):216–231, 2005.
2. E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.
3. Edgar Gabriel and Shuo Huang. Runtime optimization of application level communication patterns. In *12th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, Long Beach, CA, USA, March 2007.
4. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.
5. Intel MPI Benchmark. http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm.
6. Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, June 1995. http://www.mpi-forum.org/.
7. Message Passing Interface Forum. *MPI-2: Extensions to the Message Passing Interface*, July 1997. http://www.mpi-forum.org/.
8. Jelena Pjesivac-Grbovic, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, and Jack J. Dongarra. Performance Analysis of MPI Collective Operations. *Cluster Computing*, 10(2):127–143, 2007.
9. Ralf Reussner, Peter Sanders, Lutz Prechelt, and Matthias Muller. SKaMPI: A Detailed, Accurate MPI Benchmark. In *PVM/MPI*, pages 52–59, 1998.
10. TOP 500 webpage. http://www.top500.org/, 2007.
11. D. Turner and Xuehua Chen. Protocol-dependent message-passing performance on linux clusters. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on Linux Clusters*, pages 187–194. IEEE Computer Society, 2002.
12. R. Clint Whaley and Antoine Petite. Minimizing development and maintenance costs in supporting persistently optimized blas. *Software: Practice and Experience*, 35(2):101–121, 2005.