Personalized MPI library for Exascale Applications and Environments

Shweta Jha, Edgar Gabriel
Department of Computer Science, University of Houston, Houston, USA.
Email: {sjha2, egabriel}@uh.edu

Saber Feki
KAUST Supercomputing Laboratory, King Abdullah University of Science and Technology, Saudi Arabia.
Email: Saber.Feki@kaust.edu.sa

Abstract—Minimizing the communication costs associated with a parallel application is a key challenge for the scalability of petascale and future exascale application. This paper introduces the notion of a personalized MPI library that is customized for a particular application and platform. The work is based on the Open MPI communication library, which has a large number of runtime parameters that can be modified without having to recompile the MPI library or the application. The approach described in this paper leads to a parameter set that minimizes the execution time of a given application. The paper presents a search algorithm adapted from experimental design theory to deal with large parameter spaces and the integration of the Open MPI mpiexec tool with a database to retrieve parameter sets based on application and platform specific settings. We demonstrate the benefits of this approach by tuning point-to-point and collective communication operations, and discuss advantages and disadvantages of various optimization strategies from an application developers point of view.

I. INTRODUCTION

Communication operations often represent sections with limited scalability in parallel applications. The main challenge for optimizing communication operations stem from the fact that certain parameters influencing the performance of communication operations are dependent on application and platform specific characteristics and can not be easily generalized. For example, the cross over point between various algorithms of a collective operation, or the switching point from eager to rendezvous protocol for point-to-point operations have typically been determined using very limited testing, and by no means exploring all feasible combinations of (message length, number of processes). Ultimately, a one-size-fits-all solutions for MPI libraries will be difficult to maintain for exascale environments: MPI libraries will have to be customized for specific applications and platforms to maximize their performance.

In this paper, we target the development of a personalized MPI library, i.e. an MPI library which is customized for a particular application and platform. The work is based on the Open MPI communication library [2] which has a large number of runtime parameters influencing its performance. These parameters can be modified at runtime without having to recompile the MPI library or the application. One approach to provide a personalized communication library is therefore to determine a parameter set which minimizes the execution time of a given application. To achieve this goal, three fundamental problems have to be solved: (i) how to optimize a very large parameter set, since an exhaustive evaluation of all possible parameter combinations is not feasible; (ii) how to manage, store and retrieve the optimized parameter sets; (iii) developing a time efficient optimization strategy for a given application scenario. This paper addresses all three challenges mentioned above.

The remainder of the paper is organized as follows: In section II we describe briefly the $2^k$ factorial design algorithm and its integration within our framework, while section III describes the integration of a data base system with the Open MPI execution environment. Section IV evaluates various search algorithms for tuning several Open MPI runtime parameters and highlights the effectiveness of the $2^k$ factorial design algorithm. Finally, section V summarizes the work.

II. PERFORMANCE TUNING IN LARGE PARAMETER SPACES

Tuning a problem with a large number of parameters can be mapped to an empirical search problem trying to identify the optimal values for the corresponding parameters. The challenge lies in the fact that the search space increases combinatorially with the number of parameters, necessitating advanced algorithms to avoid the cost of an exhaustive search.

Generally speaking, the goal of designing an experiment is to find the optimum design with a minimum number of experiments. In case of code tuning, this translates to obtaining the lowest execution time of the benchmark by evaluating the minimum number of parameter combinations. The first step therefore is to reduce the number of factors and to choose those that have significant impact on the performance of the application. For this, each parameter needs to be reduced initially to have exactly two values. For unidirectional parameters, i.e. parameters which lead to a continuous increase/decrease in the performance of the benchmark with increasing/decreasing parameter values, one could choose the outermost values.

The $2^k$ factorial design algorithm determines the effect of $k$ parameters or factors, requiring a total of $2^k$ experiments in the first stage. Using a non-linear regression model, this algorithm gives an indication on the proportion of the total variation in the performance that is explained by each factor/parameter. Thus, the algorithm helps classify the factors according to their performance impact and allows to select the parameters
which have the greatest influence on the performance. These parameters will be explored in the second step in more details, by deploying a brute-force search over all parameters which have not yet been removed after the first stage.

III. INTEGRATION WITH A DATABASE SYSTEM

Once an optimal parameter set has been identified, the main challenge lies in storing and retrieving the parameter sets in a scalable and user friendly manner. For this, we deploy a POSTGRESQL data base system. A parameter set can be uploaded into the database using a tool developed as part of the OTPO [1] framework. The tool connects to the database systems, and uploads a parameter file and additional metadata which will be used to query for a parameter set subsequently. The metadata consists of: (i) hostkey: a unique identifier for a host or cluster; (ii) application key: a key uniquely identifying a parallel application; and (iii) application characteristics: list of characteristics necessary to identify a particular use case, and could include the no. of processes, message length or the application problem size.

To retrieve a particular parameter set, the mplexec command of the Open MPI library has been extended by parameters indicating the address of the database system, as well as the hostkey and the application key to be used. When parsing the parameter files, Open MPI will try to contact the database and retrieve the parameter set for this particular scenario. The resulting parameter set is dumped into a local file, and will be processed by all MPI processes similarly to the default parameter file.

IV. EVALUATION

The evaluation of the work described in this paper is organized in two subsections: (i) evaluate the benefits of using $2^k$ based search to create a personalized configuration of Open MPI; (ii) a use-case scenario for determining the performance sensitivity of Open MPI to a set of network parameters as a function of the message length.

A. Evaluating the $2^k$ factorial design search algorithm

In this subsection, we demonstrate that the $2^k$ factorial design search algorithm significantly reduces the time to find a (nearly) optimal parameter set. The platform used for the subsequent experiments is the shark cluster at the University of Houston, which consists of 24 single processor dual-core AMD Opteron nodes and 5 dual processor quad-core AMD Opteron nodes. The nodes are interconnected by an InfiniBand network interconnect. Open MPI trunk revision 22580 was used for the tuning. Using the NetPipe [4] benchmark for a message length of 10 Kbytes, we compare the time taken by OTPO to tune various parameter sets of the InfiniBand module of Open MPI (openib) for all three search algorithms. Table I summarizes the results.

First, it is important to note that all three search algorithms were able to identify parameter combinations which lead to the same (minimal) execution time for a point-to-point data exchange between two processes for a 10 KByte message. The results shown in table I for tuning four parameters reinforce that the brute force search takes significantly longer than the $2^k$ factorial design based search (over 5 hours vs. 32 sec), with the attribute based search being in between the other two algorithms. This is due to the fact, that the number of parameter combinations evaluated is significantly smaller for the $2^k$ factorial design and attribute based search, which is shown in table I in the third row of each test case (# Combinations).

<table>
<thead>
<tr>
<th>Time</th>
<th># Solutions</th>
<th># Combinations</th>
<th>2^k Fact.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>5h 20min</td>
<td>2136</td>
<td>24</td>
</tr>
<tr>
<td>Attribute based</td>
<td>1 min 52 sec</td>
<td>109</td>
<td>24</td>
</tr>
</tbody>
</table>

Results obtained by tuning 8, 10 and 11 parameters of the openib module reinforce the previous findings. To keep the time spent within reasonable limits, the brute force search algorithm was stopped when 5% of all parameter combinations have been evaluated, and the tuning time for this algorithm extrapolated based on this value. The results are ranging from one day to more than 12 days using the brute force search. For the $2^k$ factorial design search algorithm results were in the range of a few minutes, with a maximum of one and a half hours for the largest configuration, which makes tuning a relatively large number of parameters feasible.

To expand on the analysis of the last paragraph, we evaluated the impact of the threshold value of the $2^k$ factorial design search algorithm which determines whether a parameter is removed from the search space or not, on the tuning time and the quality of the solution found. Table II presents the results obtained for various threshold values for the $2^k$ factorial design search using the SKaMPI [3] benchmark for tuning a broadcast operation for a message length of 1 MByte with 32 processes. Five different MCA parameters were tuned in these tests, namely two parameters of the hierarch collective communication module and three InfiniBand parameters. The results indicate that the time for tuning increases with lower values of the threshold with the solutions quality being closer to the solution identified by the brute force search, since few parameters are excluded by the $2^k$ factorial analysis. Note that setting the threshold to 0% is equivalent to a brute force search since none of the parameters will be ignored. On the other hand, a high threshold value leads to fewer parameters being actually explored, which consequently takes less time but potentially impacts the quality of the solution found. Thus, the threshold value provides a mechanism to balance search time with the quality of the search result.

B. Sensitivity of the network parameters to the message length

In this section, we present a use-case scenario in which we analyze the sensitivity of the Open MPI point-to-point performance to a set of runtime parameters depending on
the message length. For this, OTPO is being used to tune a set of seven parameters of the openib module using the NetPiPe benchmark for various message lengths, and calculate the relative performance improvement compared to the default performance of Open MPI if these parameters are set to their default values. Tests in this subsection have been executed on the crill cluster at the University of Houston using Open MPI 1.8.1. The crill cluster consists of 16 nodes with four 12-core AMD Opteron (Magny Cours) processor cores each (48 cores per node, 768 cores total) and 64 GB of main memory per node. Each node further has two 4X DDR InfiniBand HCAs, although only one HCA has been used in the subsequent tests.

Fig. 1. Sensitivity of the Open MPI point-to-point performance to openib parameters.

The results shown in the left part of fig. 1 indicate, that the performance of Open MPI over this InfiniBand network interconnect can be improved by up to 10% using optimized values for these parameters, but only for messages in the range of 12 KByte to 36 KByte. The performance of Open MPI can not be improved significantly using these parameters for very short and very long messages.

To demonstrate the implications of improving the performance of a point-to-point operation by a small percentage, consider the results shown in table III. The table documents the execution time of three different all-to-all communication algorithms (linear, pairwise exchange and Bruck’s algorithm) for a message length of 16 KByte, the message length which showed the largest sensitivity for the openib component according to fig. 1. Results are shown for broadcast operations using 32 and 128 processes and the relative performance improvement when using the optimized parameter sets. Each data point shown in table III is the average of multiple runs (typically between two and four runs), with minimal variations seen between individual executions of SkaMPI. The results indicate significant performance improvements when using the optimized openib parameter set for the linear and the pairwise algorithm, with performance benefits ranging from 20% to over 60%. However, Bruck’s algorithm does not show any improvement when using the optimized openib parameters. While our analysis into this behavior is not yet finalized, indications are that certain properties of the implementation such as early-sender/late receiver or late-sender/early-receiver [5] can either amplify a small performance improvement for point-to-point operations - or make them completely irrelevant.

TABLE II

<table>
<thead>
<tr>
<th>Threshold</th>
<th># params.</th>
<th>Time</th>
<th>Perf. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>5</td>
<td>5h 6min</td>
<td>0%</td>
</tr>
<tr>
<td>30%</td>
<td>3</td>
<td>13min 32sec</td>
<td>3.71%</td>
</tr>
<tr>
<td>50%</td>
<td>2</td>
<td>7min 2sec</td>
<td>4.97%</td>
</tr>
<tr>
<td>70%</td>
<td>1</td>
<td>5min 14sec</td>
<td>14.13%</td>
</tr>
</tbody>
</table>

V. Conclusion

In this paper we introduced the notion of a personalized MPI library by creating a custom set of runtime parameters for a particular application and platform. We gave a brief overview of an advanced search algorithm from the experimental design theory. We evaluated its effectiveness by comparing the time required to tune multiple parameters for different benchmarks with three different search algorithms. The results indicate that the tuning time is significantly reduced in all scenarios compared to the other search algorithms, while the quality of the solution found when using the $2^n$ factorial design search was either identical or very close to the solution derived from an exhaustive search. We also described a framework developed for storing application and platform specific parameter sets in a database system and retrieving them from within the mpiexec tool of the MPI library.

Acknowledgments. Partial support for this work was provided by the National Science Foundation under award no. CNS-0846002, CNS-1148052 and CRI-0958464. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES


