EXTENDING PERFORMANCE EXPERT SYSTEM TEMPLATES SET TO SUPPORT MPI 3.0

A. LINEY¹, S. FROLOV², A. SIDNEV³, A. VIKHIREV³, O. LOGINOV⁴

¹ Department of Software Engineering, Lobachevsky State University of Nizhny Novgorod, Russia
² Department of numerical and functional analysis, Lobachevsky State University of Nizhny Novgorod, Russia
³ Department of Software, Lobachevsky State University of Nizhny Novgorod, Russia
⁴ Intel Corporation
E-mail: ¹alin@unn.ru, ²sergeyfrolov@hotmail.com

ABSTRACT

Development of high performance applications requires support from various profiling and analysis tools. Advances in technology should be accompanied by rework of existing instruments and continuous revision of their compliance to state of the art. We demonstrate the ability of existing tools, such as Performance Expert, to address performance problems of MPI-3.0 functions. This paper describes the rules required to expand these performance tools. A set of simulations and a real-world application are used to confirm the correctness of our approach.

Keywords: MPI, Performance analysis, Performance assistance, Performance problems detection, Non-blocking communication

1. INTRODUCTION

Message Passing Interface standard is one of the most widely used programming interfaces for high-performance computing. Most MPI applications reach only a small fraction of peak performance. In attempts to reach highest potential, you will have to deal not only with traditional computational bottlenecks, but also with communication overhead. While there are many advanced tools which provide guidance for computational performance problems like PerfExpert [1] or MACPO [2], addressing MPI performance problems becomes more difficult with releases of new standard and appearing of high-performance implementations of these standards. These libraries provide great possibilities for optimization, but software engineers not always could realize all the potential of new functions (i.e. RMA and NBC).

In order to make optimization of MPI functions easier and more accessible, several performance analysis tools and frameworks [3] were developed. There are two general kinds of performance analysis: profiling and tracing. Profiling tools collect statistics at run time resulting in less overhead and less data. They could barely collect enough information to make comprehensive automatic analysis. Tracing tools save event history allowing more detailed analysis and visualize the execution traces.

There are many tools to collect data, needed to detect MPI performance problems. Many initiatives appeared to create unified infrastructure of performance tools for various HPC standards, e.g. OMTP for OpenMP [4], Score-P for MPI [5], but we are going to address this problem in more abstract way and use Intel PIN [6] - general tool with basic binary instrumentation features. Since this tool uses simple techniques for custom analysis [7], approaches discussed in this paper should be usable for any other tool.

There are also many different tools to process and analyze data, but not all of them have performance assistance feature that helps to address performance problems:

- HPCToolkit [8] contains many powerful tools and metrics, which makes detection of performance problems easy, but without automatic recommendations.
- Vampir [9] is a very powerful tool, custom metrics could help to make difficult performance problems visible, but there is no an easy to use assistant.
• TAU [10] is large integrated toolkit for solving performance problems, profile and trace visualization.

Making automatic recommendations, but without support of MPI-3 functions:

• Periscope [11] has analysis of MPI bottlenecks, but does not support MPI-3.

• Intel Trace Analyzer and Collector [12] includes performance assistant. Does not support MPI-3 as well.

• KOJAK [13] is a trace analysis environment, which includes EXPERT tool allowing automatic detection of performance problems.

With support of MPI-3 functions:

• Cube4 [14] is used as performance report explorer for Scalasca and Score-P. It measures time spent on MPI functions, including e.g. on late sending/receiving, computational imbalance, making it easier to locate the problem. It does not make automatic detection, but addresses the problem directly.

MPI-3 functions do not only expand list of existing optimizations, such as RDMA (remote direct memory access) [15] or pipelining [16] with new approaches, such as NBC (non-blocking collectives) [17]. In some cases new functions increase performance, in other could significantly decrease. And it is not obvious for application developers, whether performance degraded because of improper use of MPI-3 functions or because this application could hardly benefit from them. That’s why we are highly motivated to expand existing tools and approaches to support actual version of standard.

This work is based on research of Dergunov [18] and developed by him Performance Expert [19] profiling tool. In this paper we introduce possible extension of this method and corresponding tool for new functions specified in MPI-3.0 standard.

The paper is structured as follows. Section 2 describes performance problems and how to detect them. Section 3 presents results of performance tests.

2. METHODOLOGY

2.1 Model of a performance problem

Performance problem can be defined as a set of actions that inhibit good performance, because the actions are not synchronized [20]. Figure 1 illustrates problem of late sending, which results in suboptimal performance. All the tools with automatic recommendations contain templates of typical performance problems in different representations, but abstract models of these problems are very similar.

![Figure 1: Problem of late sending](image)

According to Dergunov’s specification, formal definition of performance problem is as follows:

\[ \text{pb} = \langle \text{pd, dur, TRRULES, ANRULES, REC(AINFO)} \rangle \]

where

- \( \text{pd} \) – textual description of the problem;
- \( \text{dur} \) – duration of the problem;
- \( \text{TRRULES} \) – trace rules for actions that introduce the problem;
- \( \text{ANRULES} \) – analysis rules to recognize the problem in sequence of events in trace file;
- \( \text{REC} \) – recommendations to fix the problem;
- \( AINFO = \langle f_i, t_i, d_i, pr_i, cs_i \rangle \) – description of actions that introduced the problem (where \( f_i \) – the function that was called, \( t_i \) – time when the function was called, \( d_i \) – duration of the function execution, \( pr_i \) – process that initiated the call, \( cs_i \) – the call site represented, for example, by source file name and line number in MPI application).

2.2 Performance Expert

Analysis of MPI applications using Performance Expert consists of sequence of steps:

1) Instrumentation (modification of application in order to collect statistics).
2) Run of modified program, which produces trace.
3) Automatic processing of collected data.
4) Presentation of analysis including performance problems and recommendations how to deal with them.

Dynamic instrumentation in Performance Expert is performed by Intel PIN. Based on collected trace, basic events are assembled into composed events, which could be performance problem. Found performance problem is reported with unique recommendation for every kind of problem and location of cause in source code.

Knowledge base of performance problems in Performance Expert includes 10 typical patterns of performance problems of point-to-point, collective and RMA operations. This set does not correspond to latest version of MPI standard, that's why we expand it with classification suggested by Scalasca’s [21] developers which is expanded by MPI-3.0 performance problems and contains Dergunov’s classification as subset.

Performance Expert detects performance problems based on rules of three classes:
- Tracing Rules filters tracing events which must be logged at run time;
- Composite Events Construction Rules unite several low level events like function calls into logical events like data block transmitting;
- Performance Problem Detection Rules are templates for detecting possible performance losses.

Performance Expert had full support of MPI-1.0 and partial support of MPI-2.x. MPI-3.0 introduces next communication operations:

- One Sided – Remote Memory Access was introduced in MPI-2.0 and significantly expanded in subsequent updates.
- Neighborhood Collective Operations.
- Non-blocking Collective Operations.
- Accumulate Ordering and Memory Semantics.
- New Completion/Synchronization Semantics.

It also introduces other features which are not relevant to this paper, such as scalability improvements and fault tolerance.

Tracing Rules and Composite Events Construction Rules was simply extended with new MPI 3.0 functions. Also we added new Performance Problem Detection Rules for uncovered situation from Scalasca’s classification.

3. RESEARCH RESULTS

We designed a set of tests which show usage of existing and added rules, which includes synthetic and real applications. For all synthetic tests performance problems was successfully diagnosed, with detection of total time spent in function calls.

Testing environment:
- 4 nodes
- CPU: 2x Intel Xeon L5630 (2.13GHz, 4 Cores)
- Memory: 24.0 GB
- Network: Infiniband QDR
- OS: Microsoft Windows Server 2008 HPC Edition x64

3.1. Simulation of “Late Sender” performance problem.

This simple app illustrates our approach of simulation of performance problems:

```c
if (g_processId == 0) {
    Sleep(sleepTime);
    // Simulating work
    MPI_Send((void*)(msg),MsgLength, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
}
if (g_processId == 1) {
    MPI_Recv(buffer,MessageLength, t, 0, 0, MPI_COMM_WORLD, &status);
}
```

Rule which ensures detection of this problem is described as follows:
```
declare problem for point_to_point
    when
        send_start_time > recv_wait_start_time
    parameters(
        name = "Late sending",
        description = "Sending message is initiated long after receive is initiated. As a result, blocking receive must wait.",
        advice = "Make changes in source code location» + recv_wait_call_site + ", so that receive happens after send is done",
        duration = send_start_time - recv_wait_start_time);
```


We simulated this problem by using MPI_Exchange. Part of source code:

```c
if (processId == lateProcessId) {
```

Rule which ensures detection of this problem is described as follows:
declare problem for collective when
(function_name eq "MPI_Allgather" or function_name eq "MPI_Allgatherv" or function_name eq "MPI_Allreduce" or function_name eq "MPI_Alltoall" or function_name eq "MPI_Alltoallv" or function_name eq "MPI_Reduce_scatter" or function_name eq "MPI_Scan" or function_name eq "MPI_Exscan") and
minv(start_times) < maxv(start_times) parameters(
  name = " Wait at "many-to-many"",
  description = " Calls of "many-to-many" functions are not synchronized. As a result, blocked processes waste time waiting for others.",
  advice = "You should synchronize the calls of collective operation.",
  duration = sum_of_earlier(start_times, maxv(start_times)));

3.3. Simulation of “Wait at window creation” performance problem.

We simulated this problem by the same method using MPI_Get_Accumulate.
Rule which ensures detection of this problem is described as follows:
declare problem for collective when
function_name eq "MPI_Win_create" and
minv(start_times) < maxv(start_times) parameters(
  name = "Wait at window creation for RMA",
  description = " Calls of "MPI_Win_create" functions are not synchronized. As a result, blocked processes waste time waiting for others.",
  advice = "You should synchronize the calls of MPI_Win_create",
  duration = sum_of_earlier(start_times, maxv(start_times)));

3.4. IMB Benchmarks

Intel MPI Benchmarks is a set of benchmarks which measures performance of various MPI operations. This package consists of 5 components, covering functionality of MPI-1, one-sided, I/O, NBC and RMA functions [22].
Next functions were used to illustrate performance problems:
- IMB-MPI1: Bcast, Alltoall
- IMB-NBC: lbcast_pure, lalltoall_pure
In all cases expanded Performance Expert showed corresponding performance problem.

3.5. Poisson equation

We used implementation of Poisson equation (2-D, size = 10 000 x 10 000). Data transmission between ranks during general calculations is performed by point-to-point asynchronous functions – MPI_Isend/MPI_Irecv. Data transmission during estimation of current deviation is performed by global reduction by using different MPI functions (in different implementations of program):
1) – MPI_Allreduce
2) Asynchronous variant – MPI_Iallreduce

Table 1: Testing results of different application runtime (64 ranks).

<table>
<thead>
<tr>
<th>Operation</th>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall clock time</td>
<td>100.685</td>
<td>100.506</td>
</tr>
<tr>
<td>Jacobi time</td>
<td>36.270</td>
<td>36.993</td>
</tr>
<tr>
<td>Calculating change time</td>
<td>63.015</td>
<td>63.471</td>
</tr>
<tr>
<td>Allreduce/Iallreduce time</td>
<td>1.398</td>
<td>0.0055</td>
</tr>
<tr>
<td>MPI_Waitall time</td>
<td>-</td>
<td>0.0352</td>
</tr>
</tbody>
</table>

We got following results by using Performance Expert:
Point-to-point performance problems:
1. Late sender
   Actions:
   - MPI_Isend call from jacobi() function.
   - MPI_Waitall call from jacobi() function
   - MPI_Irecv call from jacobi() function

2. Late receiver
   Actions:
   - MPI_Isend call from jacobi() function.
   - MPI_Waitall call from jacobi() function
   - MPI_Irecv call from jacobi() function

Collective performance problems:
3. Wait at “many-to-many
   Duration: 1.389 s (Version 1), 0.041 (Version 2)
Actions:
- MPI_Allreduce call from main() function. (Version 1)
- MPI_Iallreduce call from main() function. (Version 2)
- MPI_Waitall call from main() function. (Version 2)

In this case, duration of blocking call determines the maximal potential benefit from using non-blocking collective, instead of common version. This benefit is not guaranteed and depends on specificity of algorithm implementation. Generally, to reach better performance you should effectively overlap communications and calculations.

4. CONCLUSIONS

Trace analysis of MPI applications provides opportunity to find and fix all performance problems caused by suboptimal usage of MPI calls. We expanded set of rules of Performance Expert analysis system, based on classification of Scalasca to support latest versions of MPI-3.0 standard. Subset of appeared performance problems were already covered by existing rules and needed minimal extension.

We demonstrated that detection of performance problems of asynchronous calls could be done by existing means. We showed effectiveness of MPI-3.0 functions and made one more important step towards spreading of its usage.

This work does not include the research of possibility to detect automatically the situations where one could benefit from using MPI-3.0 functions instead of MPI-1, 2. This is a very promising direction of future research, which would let developers reach higher performance easier.

5. ACKNOWLEDGEMENT

This research was partially supported by grant from Intel Corporation.

REFERENCES


