Optimization for Hybrid MPI-OpenMP Programs with Thread-to-thread Communication

TA QUOC VIET,† TSUTOMU YOSHINAGA† and MASAHIRO SOWA†

This paper proposes a middle-grain approach to construct hybrid MPI-OpenMP solutions for SMP clusters from an existing MPI algorithm. Experiments with “the High Performance Parallel Benchmark—HPL” on different cluster platforms show that, our solutions exceed the solutions based on a de-facto MPI model in most cases, and sometimes by as much as 34.5% of performance. We also prove an automatic outperformance of a thread-to-thread communication model over a classical process-to-process communication model in hybrid solutions.

1. Introduction

Clusters of Symmetric Multi-processor (SMP) have recently gained more and more popularity. Consequently, a proper choice of a parallel programming model becomes extremely important. The three model-candidates are listed as follows:

1. Pure MPI (MPI): each processing element (PE) is used for one MPI process.

2. Hybrid MPI-OpenMP with process-to-process communication (Hybrid PC): each SMP node is used for one MPI process. OpenMP is applied for computation parallelization inside SMP nodes. MPI communication is executed outside of OpenMP parallel regions.

3. Hybrid MPI-OpenMP with thread-to-thread communication (Hybrid TC): similar to the hybrid PC model. However, MPI communication tasks are done inside OpenMP parallel regions by a single thread. During communication time, non-communicating threads are assigned to do computation tasks. A computation-communication overlap inside a node is the particularity of the model.

So far, hybrid models apply two common ways to parallelize computation works with OpenMP inside a node: (1) fine-grain loop-level parallelization; and (2) coarse-grain SPMD parallelization. This paper proposes and applies another one, a middle-grain approach. Avoiding complexity of the SPMD method, it also denies a poor performance characteristic of the fine-grain parallelization (Section 3.1). Applying the middle-grain method together with an overlapping-oriented task-schedule (Section 3.2), we can create effective hybrid TC solutions that exceed MPI in performance in most cases.

For experiments, on two different platforms: a cluster of Sun Enterprise 3500 SMPs and a cluster of Intel Dual-processor SMPs, we solve a dense linear equation system from HPL (Section 5).

Hybrid PC has been examined in a large number of previous studies. In 1), the authors showed a common path to construct a fine-grain hybrid PC code from an existing MPI one. Based on this path, 2) and 3) built a fine-grain hybrid PC solution for the NAS benchmarks. Authors of 4) compared hybrid PC to pure MPI through solving the Smooth Particle Applied Mechanics (SPAM) problem. Above studies come to a conclusion that hybrid PC loses to pure MPI in most cases. Even on the Earth Simulator with the CG problem, hybrid PC outperforms pure MPI only with a considerably large number of nodes \(n\) (Section 5).

Another work has shown how to construct a coarse-grain SPMD pure OpenMP solution on a shared-memory platform. This solution outperforms a pure MPI one in all experiments. The results proved the ability to get inner-node OpenMP performance better than that of the MPI model. However, it is not clear how to apply the approach in hybrid solutions.

Hybrid TC was then discussed and put into comparison with hybrid PC in 7), 8) and 9). According to the authors, there is no automatic outperformance for hybrid TC. In Subsection 2.4, we can prove that, hybrid TC always has better performance than that of hybrid PC.

Our main contribution in this paper are listed as follows:

1. A detailed comparison among the three models, and a full analysis on hardware and software factors affecting their performance.

2. A proof for automatic outperformance of hybrid TC over hybrid PC in any circumstances.

† The Graduate School of Information Systems, University of Electro-Communications
3. An algorithm to construct a hybrid TC solution consisting of a middle-grain approach and an overlapping-oriented task-schedule, which dominates the de-facto MPI solution in most cases.

2. Parallel Model Comparison

2.1 Methodology

We compare performance of the three models through their execution time $T$, which includes computation time $T_p$ and communication time $T_m$. We have $T_p = V_p/S_p$ and $T_m = V_m/S_m$, where $V_p$ and $S_p$ are computation volume and speed, respectively; $V_m$ and $S_m$ are communication volume and speed, respectively. When work-load is not balanced among nodes, execution time of the heaviest node should be considered. Bracketed strings “M”, “PC” and “TC” are added into variable names to distinguish MPI, hybrid PC and hybrid TC, respectively. For example, $T_p(M)$ represents computation time for the MPI model.

The middle-grain approach can avoid poor inner-node computation performance of the fine-grain approach applied so far for hybrid PC solutions. The fine-grain approach do the computation tasks slower than MPI mainly because of its bad cache usage. Meanwhile, the middle-grain approach supplies hybrid models with the same computation speed as MPI. Detailed explanation of the middle-grain approach is shown in section 3.1. Thus, we can perform the model comparison under following bases:

**Basis 1.** $V_p(M) = V_p(PC) = V_p(TC)$

**Basis 2.** $S_p(M) = S_p(PC)$

**Basis 3.** $V_m = f(nprocs)$

**Basis 4.** $S_m(PC) = S_m(TC) = S_m(hybrid)$; and $S_{inter,m}(PC) \geq S_m(hybrid)$

where $T_{intra,m}(M)$ and $T_{inter,m}(M)$ are time spent for intra-node and inter-node communication, respectively.

Consequently, a list of unfavourable factors for MPI is shown as follows:

**Factor 1.** Small $S_{intra,m}(M)$: Slow intra-node communication speed causes more costs to the MPI model.

**Factor 2.** Large $nppn$: In general, MPI intra-node communication volume increases together with the number of processors per node $nppn$.

**Factor 3.** Large $S_m(hybrid)$: If a single channel can saturate (or almost occupy) the inter-node communication bandwidth, the communication speed advantage of MPI will be eliminated.

**Factor 4.** An increase of $V_m$ together with $nprocs$: If $V_m$ is an increasing function of $nprocs$, MPI suffers more communication because of larger $nprocs$.

2.2 Execution Time Calculation

Figure 1 illustrates a breakdown of execution-time for the three models.

**MPI Execution Time $T(M)$:**

$$T(M) = T_p(M) + T_{intra,m}(M) + T_{inter,m}(M) \quad (1)$$

**Hybrid PC Execution Time $T(PC)$:**

$$T(PC) = T_p(PC) + T_m(PC) \quad (2)$$

**Hybrid TC Execution Time $T(TC)$:**

With

- $T_{p,only}(TC)$: computation-only time,
- $T_{m,only}(TC)$: communication-only time,
- $T_{overlap}(TC)$: overlap time,

we have:

$$T_m(TC) = T_{m,only}(TC) + T_{overlap}(TC)$$

and $T(TC)$ can be found by:

$$T(TC) = T_{p,only}(TC) + T_m(TC) \quad (3)$$

2.3 MPI versus Hybrid PC

Because of bases 1 and 2, we have:

$$T_p(M) = T_p(PC) \quad (4)$$

From (1), (2) and (4), we can calculate difference in execution time between MPI and hybrid PC as:

$$T(M) - T(PC) = T_{intra,m}(M) + \frac{V_{inter,m}(M)}{S_{inter,m}(M)} - \frac{V_m(PC)}{S_m(PC)}$$

If the result of the subtraction is a positive number, hybrid PC is better (and vice versa). In comparison to MPI, hybrid PC does not suffer intra-node communication. On exchange, it has a slower inter-node communication speed (basis 4). Therefore, between MPI and hybrid PC, which one is better depends on the four factors shown in section 2.1.

2.4 Hybrid PC versus Hybrid TC

We compare (2) and (3). According to bases 3 and 4:

$$T_m(PC) = T_m(TC) \quad (5)$$

Since during the computation-only stage in hybrid TC, a node uses all $nppn$ PEs for computation, as in hybrid PC, we have:

$$S_{p,only}(TC) = S_p(PC) \quad (6)$$

From (2), (3), (5) and (6):

$$T(PC) - T(TC) = \frac{V_p(PC) - V_{p,only}(TC)}{S_p(PC)}$$

On the other hand:

$$V_{p,only}(TC) = V_p(TC) - V_{p,overlap}(TC)$$

and according to basis 1:

$$V_p(TC) = V_p(PC)$$

Consequently,
or “hybrid TC is faster than hybrid PC in any circumstances”. The difference in execution time is in direct proportion to the computation volume to be overlapped. When there is no overlap at all, the two models have the same execution time.

2.5 MPI versus Hybrid TC
The two models can be put into comparison via the hybrid PC model. Since hybrid TC has large advantages over hybrid PC, and hybrid PC is not so poor in comparison to MPI, we expect an advantage for hybrid TC in most cases.

3. Hybrid Solution
This section presents a new method to create hybrid solutions from original MPI algorithms. A hybrid TC solution consists of a middle-grain approach and an overlapping-oriented task-schedule. A hybrid PC solution applies only the middle-grain approach together with an original MPI task-schedule.

3.1 The Middle-grain Approach
A middle-grain approach we use in this study has the following features:

*MPI-Algorithm basement:* The solution is based on the original MPI algorithm rather than its code.

*Middle-grain parallelization:* an OpenMP parallel construct is applied for a “block” that may include both computation and communication tasks. Computation tasks are partitioned into indexed grains so that each grain can be effectively executed by a single PE.

*Intra-node flag-communication:* All data-dependencies are followed by status-flags.

3.2 Overlapping-oriented Task-schedule
Overlap volume is the key-factor to hybrid TC. An overlap is available if there is no data-dependency between some computation and communication tasks. Hereafter we propose a 4-step algorithm to build a new task-schedule from an original MPI algorithm that allows a large overlap volume.

**Step 1.** Select blocks for parallelization. For each block, do steps 2 to 4 described below:

**Step 2.** Build a task-dependency graph for the block. If the block is a loop-iteration, the graph should include also dependencies concerning the previous and the next iterations.

**Step 3.** Try one or more of the following techniques to enlarge the overlap-available part:

(a) When the block is a loop-iteration, reconstruct the loop so that larger computation and communication tasks with no data-dependency appear.

(b) If a communication task \( M \) depends on only a part of a computation task \( P \), split \( P \) into \( P_1 \) and \( P_2 \) so that \( M \) depends on \( P_1 \) only. Now we can overlap \( P_2 \) with \( M \).

(c) If a large communication task \( M \) depends on a large computation task \( P \), split \( M \) and \( P \) respectively into \( M_1, M_2, M_3 \ldots \) and \( P_1, P_2, P_3 \ldots \) so that \( M_i \) depends on \( P_i \) only. Now \( M_i \) and \( P_j \) \((i \neq j)\) become independent and can be overlapped. However, size of \( M_i \) should be large enough to avoid a decrease in communication speed\(^4\).

**Step 4.** Rebuild the task-dependency graph with modifications caused by steps 2 and 3. Based on the newly created graph, build the hybrid TC task-schedule.

4. Platform Specification
Table 1 shows configurations for Sun and Intel clusters applied in this study. Exchanging bandwidth is measured as the speed at which a node exchanges data. Exchanging bandwidth for hybrid models and intra-node bandwidth
Table 1 Sun and Intel cluster specification.

<table>
<thead>
<tr>
<th></th>
<th>Sun</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnodes</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>nppn</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Processor type</td>
<td>Ultra Sparc-II</td>
<td>Xeon</td>
</tr>
<tr>
<td>Frequency (MHzs)</td>
<td>336</td>
<td>2800</td>
</tr>
<tr>
<td>IPC</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Peak perf. (GFlops)</td>
<td>5.376</td>
<td>100.80</td>
</tr>
<tr>
<td>Cache per PE (MB)</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>Network (Mbps)</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

| OS       | Solaris 9   | Red Hat 9   |
| MPI library | HPC Cluster 4 | MPICH 1.2.5 |
| BLAS library | Forte 6     | goto blas 0.94 |

| MPI BW (Mbps) | 89.15 | 607.12 |
| Hybrid BW    | 75.27% | 58.91% |
| Intra. BW    | 10.63 times | 5.24 times |

IPC: instructions per cycle
MPI BW: MPI inter-node exchanging bandwidth
Hybrid BW: in comparison to MPI BW
Intra. BW: intra-node BW, in comparison to MPI BW

are shown as their ratio to one of MPI.

Checking the MPI-unfavourable factor list discussed in Subsection 2.3, the Sun cluster is more suitable for hybrid models than the Intel cluster in factors 2 and 3 and less suitable in factor 1.

5. Hybrid Solutions for HPL

5.1 Problem Description

HPL solves a random dense linear equation system using the right-looking variant of a block LU decomposition algorithm. Its major task is to factorize an \( n \times n \) random dense square coefficient matrix \( A \) into “upper” and “lower” triangulars \( U \) and \( L \) so that \( A = U \cdot L \). When \( n \) is large enough, the factorization occupies more than 99% of overall execution time. HPL accepts any \( n \) processes, which are organized into a \( P \times Q \) process-grid. Data in HPL are stored in \( nb \times nb \) square blocks distributed by a “block-cyclic” scheme onto the \( P \times Q \) process-grid\(^{10}\).

According to the right-looking variant, LU factorization is done by a loop with \( \lceil n/nb \rceil \) iterations. Related data for an \( i^{th} \) iteration is shown in Fig. 2(a). \( D \) is the \( i^{th} \) block of the main diagonal, \( L \) and \( T \) are the current parts of the lower, upper and trailing matrices, respectively. Table 2 shows a task-list for such an iteration. Tasks 2, 4 and 6 are communication tasks. Task 1 depends on task 7 of the previous iteration. Task 7 is the major computation task. Meanwhile, tasks 4 and 6 occupy almost all the communication volume.

5.2 Hybrid TC Task-schedule

Step 1: Block Selection

A loop-iteration described above is chosen as the middle-grain block.

Step 2: Original Task-dependency Graph

Based on Table 2, a task-dependency graph is created and shown in Fig. 2(b) where communication tasks are represented by shaded circles. From the graph, there are only two independent pairs of tasks available for an overlap: 3 with 6 or 5 with 4. However, computation tasks 3 and 5 are too small, so we should go to step 3.

Step 3: Overlap-Generation Techniques

As shown in Fig. 2(b), all tasks of the current iteration depend on the previous iteration’s task 7 that updates the “previous” trailing matrix including the “current” \( D, U, L \) and \( T \). So we apply technique (b) to split task 7 into \( 7_1, 7_3, 7_5 \) and \( 7_7 \) that update \( D, U, L, T \), respectively. We have a new task-dependency graph shown in Fig. 3(a). Tasks 1, 3, 5, 7 now depend on previous \( 7_1, 7_3, 7_5 \) and \( 7_7 \), respectively. Since tasks 4 and 6 (major communication) and previous task \( 7_7 \) (major computation) are independent, we apply technique (a) to reconstruct the loop so that they lie in the same iteration and become available for overlap. In Fig. 3(a), tasks of a new loop iteration are bounded by a shaded polygon. It includes tasks belonging to three original iterations. Then, technique (b) is applied once more: \( 7_7 \) is split into \( 7_{7_1}, 7_{7_3}, 7_{7_5} \) and \( 7_{7_7} \) to break a “big” dependency into smaller ones. Table 3 and Fig. 3(b) show the new task-list and corresponding data blocks. Figure 3(c) shows the the final version of the hybrid TC task-dependency graph. Tasks \( 7_{7_1}, 7_{7_3}, 7_{7_5} \) and \( 7_{7_7} \) come from the previous iteration; tasks 1, 2, 3 and 5 come from the next...
iteration. Tasks in the left of the vertical line are assigned to the master thread. The remaining are partitioned into grains and should be parallelized by "SOMP do nowait".

5.3 Task-partitioning & Pseudo-code

We assign an $nb \times nb$ block as a data unit. Updating a data unit forms a grain. All grains in HPL are performed by calling the well-tuned BLAS functions, which supply each grain with a good cache-hit ratio. By this way, hybrid models gain the same computation speed as MPI.

A C-style hybrid TC pseudo-code is shown in Fig. 4. All computation tasks are done by "#pragma omp for schedule(dynamic) nowait". All data-dependencies are followed through flags. If the master thread finishes communication tasks early, it can join the computation team.

5.4 Experimental Results

Using the maximum resources, hybrid TC exceeds MPI with any problem sizes, as shown in Fig. 5 (a) and (b). Its best results outperform MPI by 34.5% and 16.2% on Sun and Intel clusters, respectively. MPI also loses to hybrid PC.

As shown in Fig. 5 (c), Hybrid TC still dominates on the Sun cluster where we fix the problem size to $n=12000$ then change $npp$. When $npp=8$, it exceeds MPI by approximately 38%. MPI is worse than hybrid PC.

However, on the Intel cluster, hybrid TC can not overcome MPI at some numbers of nodes as shown in Fig. 5 (d). Hybrid models have poor performance where a well-balanced $P \times Q$ process-grid is not available (in particular, when $nnode$ is a prime number, e.g. 5 or 7). At these points, the whole lower sub-matrix $L$ of a certain iteration is stored on a single node that has to broadcast it to all the remaining, which makes communication extremely heavy and emphasizes the one-channel weak-point of hybrid models. Moreover, MPI has more options for the process grid, which may decrease internode communication volume.

6. Conclusions

In this paper, we proposed and applied an algorithm to build effective hybrid MPI-OpenMP solutions for SMP clusters. The algorithm includes a middle-grain approach and an overlapping-oriented task-schedule. Our hybrid TC solution dominates MPI in a lot of experimental variations on different cluster platforms. With the HPL problem, hybrid TC outperforms MPI by 34.5% and 16.2% on Sun and Intel clusters, respectively.

Good results for hybrid TC are also noticed with the NAS-CG benchmark. It exceeds MPI by approximately 40% and 31% of performance on the Sun and the Intel clusters, respectively. However, because of the space limitation, we can not include CG experimental details in this paper.

Showing good performance through different experimental problems on different platforms, hybrid TC should be considered as a strong candidate for a parallel programming model on SMP clusters.
We have a plan to do experiments on larger clusters to confirm scalability of the hybrid TC model.

Acknowledgement

Dr. Kenji Kise of UEC has given us really valuable comments on our work. He also supply us with access permission to a larger Intel SMP cluster for further hybrid programming experiments.

This research is supported in part by the Grants-in-Aid for Scientific Research of Japan Society for the Promotion of Science (JSPS) No.15500033.

References