Repeatable Hybrid Parallel Implementation of an Inverse Matrix Computation Using the SMW Formula for a Time-Series Simulation

Yuta MATSUI†, Nonmember, Shinji FUKUMA†, and Shin-ichiro MORI†, Members

SUMMARY In this paper, the repeatable hybrid parallel implementation of inverse matrix computation using SMW formula is proposed. The authors’ had previously proposed a hybrid parallel algorithm for inverse matrix computation. It is reasonably fast for a one time computation of an inverse matrix, but it is hard to apply this algorithm repeatedly for consecutive computations since the relocation of the large matrix is required at the beginning of each iterations. In order to eliminate the relocation of the large input matrix which is the output of the inverse matrix computation from the previous time step, the computation algorithm has been redesigned so that the required portion of the input matrix becomes the same as the output portion of the previously computed matrix in each node. This makes it possible to repeatedly and efficiently apply the SMW formula to compute inverse matrix in a time-series simulation.

key words: interactive simulation, linear equation solver, SMW formula, parallel processing, real-time processing

1. Introduction

In this paper, we have focused on the linear equation solver for $Ax = b$ which appears in many simulation codes. In particular, our interest is in the interactive simulation environment where a part of the matrix $A$ and right hand side vector $b$ are gradually changed due to either the progress of time or real-time interaction with the operator running the simulation. Figure 1 shows a demonstrative scenario of interactive simulation where we have to solve a series of linear equations with matrices $A_i (i = 0, \ldots)$ in such a way that most of the element of $A_i$ are the same as that of $A_{i+1}$ ($A_i \sim A_{i+1}$).

In general, it is not recommended to compute the inverse matrix of $A$ to solve a linear equation $Ax = b$ even if $A$ is known to be a sparse matrix. However, if the inverse matrix of a matrix approximate to $A$ is known, there exists some situation where the computation of an inverse matrix becomes feasible, especially when it can be reused multiple times to solve the linear equation for multiple right-hand side vectors. For this purpose, the Sherman-Morrison-Woodbury formula [1] (SMW formula, in short) can be used to compute the inverse matrix as follows.

If we define a matrix $A'$ as $A' = A + AA$ and let $AA$ as $AA = AA_c \times E_c$, then, the SMW formula can be re-written as in Eq. (1) which leads the $(A')^{-1}$.

$$(A + AA)^{-1} = A^{-1} - A^{-1}AA_c(I + E_cA^{-1}AA_c)^{-1}E_cA^{-1}$$

(1)

For the sake of simplicity, in the following discussion, we assume that only $s \times s$ sub-matrix of $AA$ is non-zero based on the similarity of $A$ in a time-series simulation. Then, we can make $AA_c$ and $E_c$ as the matrices of size $n \times n$ and $s \times s$, respectively. We also assume $n \gg s$.

2. Previous Work

Since most of the computational steps in Eq. (1), shown below in Fig. 2, are simple matrix-matrix multiplication, one can expect a benefit from parallel processing to accelerate the computation. In our previous work [2], we had proposed a hybrid parallel implementation of this computation. Assuming that a whole part of the initial input inverse matrix is given in each node, step 1 through 6 are computed redundantly in all nodes while the computation of the steps 7 to 9 are equally distributed among nodes in 1D (Column) block distribution fashion. And the computation steps in each node are further parallelized using OpenMP. This implementation is nearly optimal for a one-time computation.

Fig. 1 A time-line scenario of interactive simulation

Fig. 2 Computational steps
of an inverse matrix since it does not require inter-node communication during computation and the memory bottleneck can be resolved by increasing the number of nodes.

However, if \( A \) changes repeatedly, then the computation of the inverse matrix for each new \( A \) requires the relocation of the previously computed inverse matrix which is partially stored in each node. This relocation is all-to-all gather of \( A \) which significantly slows down the startup time for the new inverse matrix computation.

3. **Repeatable Hybrid Parallel Implementation**

3.1 **Fundamental Strategy**

In order to eliminate the relocation step required to repeatedly apply the SMW formula to a series of inverse matrix computations, we have redesigned our previous hybrid parallel implementation by accepting, to a certain extent, the slowdown due to the inter-node communication during the computation. To realize the relocation free implementation, only the sub-matrix of the result of previous inverse matrix computation held in each node can be used as the input for each node in the next computation. It means that the input sub-matrix for each node is the same as the result of the 1D block distribution used in step 8 of the previous implementation. In this situation, we have to rethink of the parallel execution strategy for steps 1 to 6. Considering the amount of communication, we choose a strategy in which steps 1, 2, and 5 are performed in 1D (Row) block distribution fashion whereas steps 3, 4, and 6 are performed redundantly in each node. In this strategy, only steps 2 and 5 require inter-node communication. In order to hide this communication overhead as much as possible, both steps 2 and 5 are further divided into three phases: a) local sub-matrix multiplication, b) all to all sub-matrix exchange, and c) accumulation of sub-matrix, as shown in Fig. 3. Note that, b) and c) phases are realized by the “Allreduce” function of MPI in our current implementation. Then, considering the inter-node data dependency, the reordering of the execution order of these steps is performed to increase the dependency distance. The proposed implementation also performs the overlapping of computation and communication by introducing a dedicated helper thread for communication in case of multiple node executions [3]. Figure 4 shows an overview of the execution flow of this implementation in the case of two nodes.

3.2 **Experimental Results and Discussions**

In the following discussion, we are going to explain the implementation results. Experimental environment is an 8 node PC cluster interconnected by a Gigabit Ethernet switch. Each node consists of CPU:Core2Quad Q9550 2.83[GHz] with L2 Cache of 12[MB] and 2GB of Memory, operating on OS:Linux 2.6.191.2895.fc6 with Intel compiler icc 12.0 (with -O3 option and MKL10.0.3.020). The size of matrix \( A \) is 3759 \((n = 3759)\) and its non-zero elements are 1.00\% of a whole elements. The size of \( s \) is fixed to 32, similar to the number of non-zero elements in each column of the original matrix \( A \). Table 1 shows the execution time of the previous and proposed implementations using two nodes without any overlapping of communication and computation. From this table, we could confirm that we could eliminate the data relocation time of more than 700ms with roughly 10ms, or 10\%, of slowdown in the inverse matrix computation itself. Table 2 shows how the computation time decreases with the increase of the number of nodes. Due to the memory bandwidth limitation [2], the differences between two- and four-threads executions are small in this environment, but the scalability to the number of nodes is

Table 1  Comparison of execution time[ms]

<table>
<thead>
<tr>
<th>Num. Threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>Relocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>121.8</td>
<td>94.8</td>
<td>95.2</td>
<td>723.7</td>
</tr>
<tr>
<td>Proposed</td>
<td>131.7</td>
<td>105.9</td>
<td>106.6</td>
<td>0</td>
</tr>
</tbody>
</table>

![Fig. 3 Subdivision of step 2 in the case of a 2 nodes execution](image)

![Fig. 4 Time chart of repeatable hybrid parallel implementation](image)
Concerning the effect of communication and computation overlapping, we could confirm that the time for step 1 is almost hidden in the total execution time, though its effect on the inverse matrix computation time is not that large.

4. Conclusion

In this paper, a repeatable hybrid parallel implementation of inverse matrix computation using SMW formula is proposed. In order to eliminate the relocation of the previously computed inverse matrix at the startup of the next consecutive inverse matrix computation, the computation algorithm has been redesigned. Due to intra-node data exchange which is not required in the original algorithm, the computation time of the inverse matrix gets slower, but the increase in time is far smaller than the time reduced by eliminating the data relocation as long as we choose a reasonable number of nodes. This makes it possible to repeatedly and efficiently apply the SMW formula to compute an inverse matrix in a time-series simulation. For future work, we plan to improve the efficiency of computation and communication by introducing the tiling technique and fine-grained dependency analysis.

Acknowledgments

This research is supported in part by JSPS Grants-in-Aid for Scientific Research #25280042 and #25540043, and JST A-Step #AS242Z00401H.

References


Table 2  Inverse matrix computation time with comm. and comp. overlapping[ms]

<table>
<thead>
<tr>
<th>Num. Nodes</th>
<th>Num. Threads in Each Node</th>
<th>Previous</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>121.8</td>
<td>130.1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>49.1</td>
<td>65.5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>95.2</td>
<td>106.5</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>48.7</td>
<td>65.6</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>24.0</td>
<td>44.6</td>
</tr>
</tbody>
</table>

Table 3  Breakdown of the inverse matrix computation time

<table>
<thead>
<tr>
<th>Nr. of Threads</th>
<th>Elapsed Time[ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Concerning the effect of communication and computation overlapping, we could confirm that the time for step 1 is almost hidden in the total execution time, though its effect on the inverse matrix computation time is not that large.

4. Conclusion

In this paper, a repeatable hybrid parallel implementation of inverse matrix computation using SMW formula is proposed. In order to eliminate the relocation of the previously computed inverse matrix at the startup of the next consecutive inverse matrix computation, the computation algorithm has been redesigned. Due to intra-node data exchange which is not required in the original algorithm, the computation time of the inverse matrix gets slower, but the increase in time is far smaller than the time reduced by eliminating the data relocation as long as we choose a reasonable number of nodes. This makes it possible to repeatedly and efficiently apply the SMW formula to compute an inverse matrix in a time-series simulation. For future work, we plan to improve the efficiency of computation and communication by introducing the tiling technique and fine-grained dependency analysis.

Acknowledgments

This research is supported in part by JSPS Grants-in-Aid for Scientific Research #25280042 and #25540043, and JST A-Step #AS242Z00401H.

References