Solving Laplace Equations on the Connection Machine*

Min-You Wu†

Abstract. In this paper, we present a domain decomposition method of a modified Gauss-Seidel algorithm for the Connection Machine. This algorithm converges as fast as the Gauss-Seidel algorithm, and its dependency pattern between processors is the same as the Jacobi algorithm. It proves that the medium-grained approach can achieve better performance compared to the virtual processor scheme. We also show that problems of larger sizes can run on the Connection Machine with this medium-grained approach.

1. INTRODUCTION

Most domain decomposition methods of Laplace equations are designed for the Jacobi algorithm [Jenn77] [ReFu88]. The Jacobi algorithm is not as efficient as the Gauss-Seidel algorithm since the former converges more slowly than the latter [WuGa88a]. However, the Gauss-Seidel algorithm has a much heavier dependency than the Jacobi algorithm [FJLO88]. This dependency may cause processor suspension on a concurrent implemen-

*This work was supported by the Office of Naval Research under Contract N00014-86-K-0310.

†Department of Computer Science, Yale University, New Haven, CT 06520.

tation. We suggest a modified Gauss-Seidel algorithm for the Connection Machine. This algorithm converges as fast as the Gauss-Seidel algorithm, and its dependency pattern between processors is the same as the Jacobi algorithm.

The Connection Machine is a massively parallel, single instruction multiple data (SIMD) machine [SaSw88] [TuRo88] [Thin87]. Known as a fine-grained parallel architecture, it is usually used in the style of allocating a single data item to one processor. If the number of data items exceeds the number of physical processors, virtual processors may be used. The Connection Machine uses fine granularity in partitioning to explore maximum parallelism. However, when a problem is partitioned in this fine-grained style, communication could dominate the total execution time and result in low efficiency. We will show in this paper that by using proper granularity, better performance can be obtained. Moreover, with medium granularity of partitioning we are able to run larger problems on the Connection Machine compared to that with the virtual processor scheme.

2. ALGORITHMS

There are many algorithms designed for solving Laplace equations [Jenn77]. In the Jacobi algorithm, the value $A_{i,j}$ at the kth iteration is obtained by using the values of the previous iteration from its neighboring points. The update procedure for step k may be presented as:

$$A_{i,j}^{(k)} = \frac{1}{4}\,[A_{(i-1),j}^{(k-1)} + A_{i,(j-1)}^{(k-1)} + A_{(i+1),j}^{(k-1)} + A_{i,(j+1)}^{(k-1)}]$$

All values at the same iteration may be updated simultaneously. It is easy to reach a balanced load distribution. However, this algorithm converges very slowly, especially for large problem sizes.

In the Gauss-Seidel algorithm, the values $A_{i,j}$ are updated in sequence. Its update procedure for step k may be presented as:

$$A_{i,j}^{(k)} = \frac{1}{4}\,[A_{(i-1),j}^{(k)} + A_{i,(j-1)}^{(k)} + A_{(i+1),j}^{(k-1)} + A_{i,(j+1)}^{(k-1)}]$$

Each value of $A_{i,j}$ at iteration k is obtained from two newly generated values for iteration k and two old values of iteration k-1. This algorithm assumes a particular updating sequence. Since $A_{i,j}$ depends on the values of the same iteration, not all values of an iteration can be obtained simultaneously. Although heavy dependency slows down execution speed and causes processor suspension, it converges much faster than the Jacobi algorithm.

The Jacobi algorithm can be implemented on the Connection Machine in a straightforward manner. The Gauss-Seidel algorithm converges fast, but it is not good for concurrent implementation because of its heavy dependency. The number of iterations to converge for the Jacobi and Gauss-Seidel algorithms is shown in Table 1. The matrices are randomly generated, and *Delta* is 0.01. When the matrix sizes get larger, the number of iterations for the Jacobi algorithm increases fast, whereas that for the Gauss-Seidel algorithm is almost invariant.

Table 1: Comparison for Convergency (Iterations)

	Matrix size				
	16	32	64	128	256
Jacobi	137	242	529	1029	2211
Gauss-Seidel	9	10	11	10	12

The Connection Machine is usually used in the style of allocating a single data item to one processor. When the Jacobi or Gauss-Seidel algorithm is implemented in this style, a processor must obtain four values from its neighbors before it updates its own value. That is, four communications must be carried out for one updating step. Although the NEWS (north, east, west, and south) grid is used for this particular problem, communication takes longer than the computation itself. Also, a particular sequence must be preserved for the Gauss-Seidel algorithm so that a processor receives two newly generated values before updating its own value. Although the computation amount is equal for each processor, different processors may suspend at different times because they must wait for others' values [WuGa88a].

A modified Gauss-Seidel algorithm, in the medium-grained approach, is proposed for the Connection Machine. In this algorithm, a matrix is partitioned into many blocks and each processor takes responsibility for one data block. If we apply the Gauss-Seidel algorithm restrictly in the block, some data items at the boundary will require new values of the same iteration from other processors. Now, we modify the Gauss-Seidel algorithm so that the dependency between data blocks during each iteration can be eliminated. If any data item at the boundary needs a new value from other blocks, we substitute the new value with

Table 2: Convergency for the Modified Gauss-Seidel Algorithm (Iterations)

	Matrix size					
Grain size	16	32	64	128	256	512
2 * 2	12	14	13	14	15	14
4 * 4	11	12	12	12	13	12
8 * 8	9	10	11	12	13	12

the old value of the previous iteration. In this manner, the data exchange between blocks can be performed simultaneously at the end of each iteration. Thus, the communication behavior of this modified Gauss-Seidel algorithm is similar to the Jacobi algorithm, and the updating sequence inside of the data block follows the Gauss-Seidel algorithm. The major steps of the modified Gauss-Seidel algorithm can be described below:

```
1. store one data block to each processor;
2. for each iteration k,
 2.1. exchange boundaries of data blocks simultaneously;
 2.2. update values inside of the block in sequence.
```

This algorithm is shown in Fig. 1. The load for each processor is well balanced, and this algorithm converges fast. Its convergency speed is shown in Table 2. The program with larger granules converges slightly faster than that with the smaller granule.

3. IMPLEMENTATION AND RESULTS

These three algorithms are implemented in C* and run on a CM-2 with 4K processors. Performance data is shown in Table 3. The Jacobi and the Gauss-Seidel algorithms can run up to the matrix size of 256. For the matrix size of 128 or more, the virtual processor scheme is used since the number of physical processors is not large enough. The modified Gauss-Seidel algorithm can run up to the matrix size of 2048 in 32.36 seconds, with the grain size of 32*32.

To obtain the best performance, the grain size of partitions is 2*2 for matrix sizes no greater than 128. The grain size should be increased for larger matrix sizes. That is, the grain size is 4*4 for a matrix size of 256 and 8*8 for a matrix size of 512, and so on. When the matrix size increases, the grain size must be large enough to avoid using

```
Modified_Gauss_Seidel_Algorithm
     /*
     in each processor, e is a two-dimensional array
       to hold an n*n data block;
     a two-dimensional NEWS grid is defined;
     */

   poly int rowId = row index of NEWS grid;
   poly int colId = column index of NEWS grid;

   for each iteration {
       /* set up boundaries */
     for (i=1; i<=n; i++)
       e[i][0] = fetch the value of e[i][n] from the west;
     for (i=1; i<=n; i++)
       e[i][n+1] = fetch the value of e[i][1] from the east;
     for (j=1; j<=n; j++)
       e[0][j] = fetch the value of e[n][j] from the north;
     for (j=1; j<=n; j++)
       e[n+1][j] = fetch the value of e[1][j] from the south;

       /* update */
     for (i=1; i<=n; i++)
       for (j=1; j<=n; j++)
         e[i][j] = (e[i-1][j]+e[i+1][j]+e[i][j-1]+e[i][j+1])*0.25;
   }
```

Figure 1: The modified Gauss-Seidel algorithm for Laplace equations.

Table 3: Performance Comparison of Three Algorithms (secs)

Matrix size	Jacobi	Gauss-Seidel	Modified Gauss-Seidel		
			2*2	4*4	8*8
16	3.08	1.28	0.59	1.13	2.44
32	5.32	2.21	0.69	1.25	2.73
64	12.17	3.76	0.63	1.25	3.00
128	61.74*	14.07*	0.70	1.27	3.37
256	430.04*	92.58*	2.01*	1.38	3.60
512	—	—	6.32*	4.31*	3.72

* Use virtual processors

virtual processors. Thus, more data items are grouped into one process to reduce the communication demand. Figure 2 shows the comparison of the three algorithms. Here, the best grain sizes are chosen for different matrix sizes in the modified Gauss-Seidel algorithm.

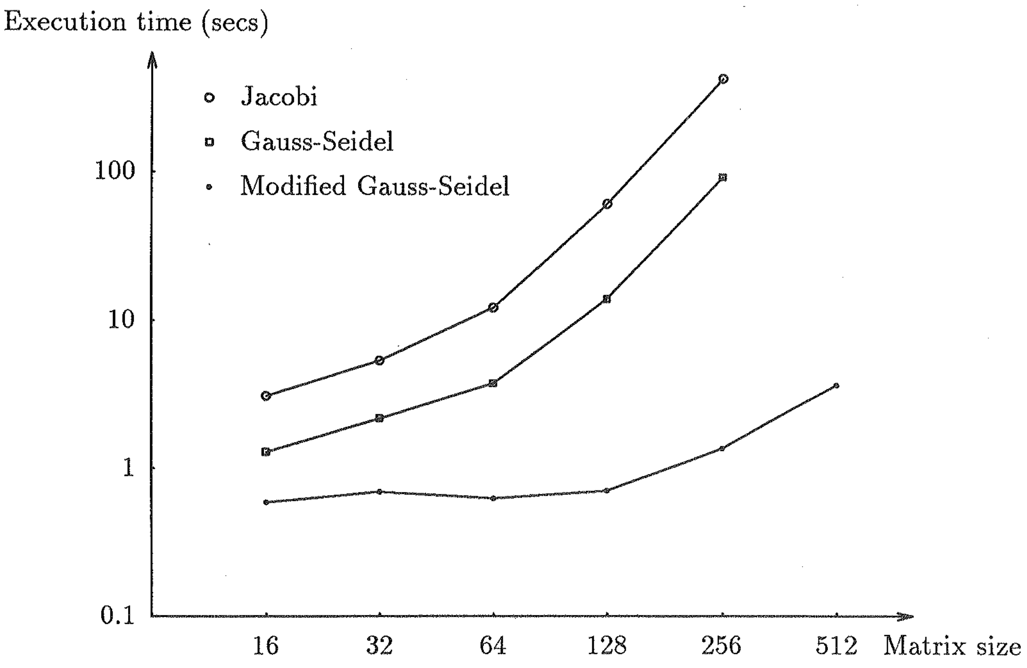

Figure 2: Performance Comparison of Three Algorithms.

4. CONCLUSION

The main problem in multiprocessing is not only how to build a system, but also how to use it. That requires development of parallel algorithms and programs that can be executed efficiently. The Connection Machine is a massively parallel system, which potentially delivers high performance. However, it should be used in a prudent way to obtain good performance. To solve Laplace equations, a modified Gauss-Seidel algorithm has been introduced. This algorithm converges fast and all blocks can be updated simultaneously to minimize processor suspension. The grain size is selected to fully utilize the computation resource.

The style of allocating a single data item to one processor requires heavy communication. Furthermore, as the number of data items increases, the virtual processor scheme must be used by this style. It often leads to more communication traffic and low efficiency. As an alternative, an application problem can be partitioned into several processes, each of which has many data items. Each process is allocated to one processor. When the number of data items increases, this medium-grained approach is especially favorable compared to the virtual processor scheme. By using proper granularity, many interprocessor communications are eliminated. Communication overhead and communication delay caused by network contention is reduced too.

Acknowledgments

The author wishes to thank Eileen Daily for her editorial efforts. The generous support of the Office of Naval Research under Contract N00014-86-K-0310 is gratefully acknowledged.

References

[FJLO88] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*, Vol. I, Prentice-Hall, 1988.

[Jenn77] A. Jennings, *Matrix Computation for Engineers and Scientists*, John Wiley & Sons, New York, 1977.

[ReFu88] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, 1988.

[SaSw88] R. K. Sato and P. N. Swarztrauber, "Benchmarking the Connection Machine 2," *Proceedings Supercomputing '88*, pp. 304-309, Nov. 1988.

[Thin87] *C* Reference Manual*, Version 4.0, Thinking Machines Corp., Aug. 1987.

[TuRo88] L. W. Tucker and G. G. Robertson, "Architecture and Applications of the Connection Machine," *IEEE Computer*, Vol. 21, No. 8, pp. 26-38, Aug. 1988.

[WuGa88] M. Y. Wu and D. D. Gajski, "A Programming Aid for Hypercube Architectures," *Journal of Supercomputing*, 2, pp. 349-372, 1988.

[WuGa88a] M. Y. Wu and D. D. Gajski, "Computer-Aided Programming for Multiprocessing Systems," *Tech. Report 88-19*, Dept. Information and Computer Science, UC Irvine, June, 1988.