Applied Mathematics, 2013, 4, 177-182
http://dx.doi.org/10.4236/am.2013.41A028 Published Online January 2013 (http://www.scirp.org/journal/am)

Parallel Implementation of the Gauss-Seidel Algorithm on *k*-Ary *n*-Cube Machine

Mohammad H. Al-Towaiq
Department of Mathematics and Statistics, Jordan University of Science and Technology, Irbid, Jordan
Email: towaiq@just.edu.jo

Received August 5, 2012; revised September 7, 2012; accepted September 14, 2012

ABSTRACT

In this paper, we present parallel implementation of the Gauss-Seidel (GS) iterative algorithm for the solution of linear systems of equations on a *k*-ary *n*-cube parallel machine using Open MPI as a parallel programming environment. The proposed algorithm is of $O(N^3/k^n)$ computational complexity and uses $O(nN)$ communication time to decompose a matrix of order N on the a *k*-ary *n*-cube provided $N \geq k^{n-1}$. The incurred communication time is better than the best known results for hypercube, $O(N \log n!)$, and the mesh, $O(N\, n!)$, each with approximately $n!$ nodes. The numerical results show that speedup improves as number of processors increased and the proposed algorithm has approximately 80% parallel efficiency.

Keywords: Cluster Computing; Parallel Computing; Linear Systems of Equations; Direct Methods; Iterative Methods

1. Introduction

In linear algebra the solution of the system of linear equations

$$Ax = b \tag{1}$$

where A is an n by n dense matrix, b is a known n-vector and x is n-vector to be determined, is probably the most important class of problems. It is needed in many areas of science and engineering which require great computational speed and huge memory.

There are two classes of methods for solving linear systems of equations, direct and iterative methods. A direct method is a fixed number of operations are carried out once, at the end of which the solution is produced. Gauss elimination and related strategies on a linear system is an example of such methods. Direct methods are the primary for solving linear systems. Those methods are often too expensive in either computation time or computer memory requirements, or possible both. As an alternative, such linear systems are usually solved with iterative methods. A method is called iterative when it consists of a basic series of operations which are carried out over and over again, until the answer is produced, or some exceptional error occurs, or the limit on the number of steps is exceeded [1].

There are two different aspects of the iterative solution of linear systems of Equation (1). The first one is the particular acceleration technique for a sequence of iteration vectors, that is the technique used to construct a new approximation for the solution x, with information from previous approximation. The second aspect is the transformation of a given system to one that can be more efficiently solved by a particular iteration method (preconditioning). For the early parallel computer, in existence in the eighties, it was observed that the single iteration steps of most iteration methods offered too little opportunity for effective parallelism, in comparison with, for instance, direct method for dense matrices. In particular, the few inner products that required per iteration for many iteration methods were identified as obstacles because of communication. This had led to attempts to combine iteration steps, or to combine the message passing for different inner products.

Recently, many parallel implementation and the computing architecture have become increasingly parallel attempt to overcome this limitation [2-9]. Some implementations have been developed for regular problems such as Laplace equation [10,11], circuit simulation problems [12], the power load-flow problems were an alternating sequential/parallel multiprocessors is used [13], and for many applications of inter-dependent constraints or as a relaxation step in multi-grid methods [2]. Also, [14] presented several parallelization strategies for the dense Gauss-Seidel method. These strategies are compared and evaluated through performance measurements on a large range of hardware architectures. They found that these new architectures do not offer the same trade-off in term of computation power versus communication and synchronization overheads, as traditional high-performance platforms.

In 1999, Feng Wang and Jinchao Xu [15], present a specific technique of solving convection-dominated problems. Their algorithm uses crosswind thin blocks in a block Gauss-Seidel method. Their method is based on a special partitioning technique for a block iterative method for solving the linear system derived from a monotone discretization scheme for convection-diffusion problems. They conclude that crosswind grouping is essential for the rapid convergence of the method.

In 2005, Jens Grabel *et al*. [16], present two simple techniques for improving the performance of the parallel Gauss-Seidel method for 3D Poisson equation by optimizing cache usage as well as reducing the number of communication steps.

In 2006, O. Nobuhiko *et al*. [17], present a novel parallel algorithm for block Gauss-Seidel method. The algorithm is devised by focusing on Reitzinger's coarsening scheme for the linear systems derived from the finite element discritizations with first order tetrahedral elements.

Most of the results show that the time consuming on communication between processors limit the parallel computation speed. Motivated by this fact and to conquer this problem we use the *k*-ary *n*-cube machine in order to change the interconnection network topology of parallel/ computing, and develop a cluster-based Gauss-Seidel algorithm, which is suitable for the parallel computing. A generic approach for the method will be developed and implemented. Also execution time prediction models will also be presented and verified.

The rest of the paper is structured as follows: in Section 2 we introduce and analyze the Gauss-Seidel sequential algorithm. We describe the proposed parallel algorithms in Section 3. We evaluate the performance of the parallel algorithm in Section 4. We conclude the paper in Section 5.

2. Gauss-Seidel Sequential Algorithm

To evaluate the performance of parallel Gauss-Seidel (GS) algorithm one must first look at the sequential performance. GS is an improvement of the Jacobi algorithm. That is, GS corrects the *i*th component of the vector $x_i^{(k)}$, in the order $i = 1, 2, \cdots, n$. However, the approximation solution is updated immediately after the new component is determined. The newly computed component $x_i^{(k+1)}$, $i = 1, 2, \cdots, n$ can be changed within a working vector which redefined at each relaxation step, and this results in the following iterative formula

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \Big/ a_{ii} \qquad (2)$$

In matrix notation, Equation (2) becomes

$$(D-L)x^{(k+1)} = Ux^{(k)} + b, k \geq 0 \qquad (3)$$

where L, D, and U are the lower, diagonal, and upper-triangular parts of matrix A respectively.

Figure 1 present GS sequential algorithm.

Performance Analysis

In this section we evaluate the behavior of the sequential GS algorithm. It is well known that the algorithm will always converges if the matrix A is strictly or irreducible diagonally dominant. So, we used different problems with different matrix sizes $n = 32, 64, 128, 256, \cdots, 16368$.

Figure 2 shows the relationship between different sizes of matrix A and the real execution time (in seconds) needed for the solution of the problem. The figure indicates that the convergence of GS algorithm is dependent of the problem size and the number of iterations increases as the problem size becomes large (it is of order N^2, $O(N^2)$, for k iterations we need $k(N^2)$ operations). So, the algorithm is not scalable.

3. Parallel Implementation of the Proposed Algorithm

In this section we propose a parallel algorithm for Gauss-Seidel using *k*-ary *n*-cub machine which well-suited for cluster computing. First, we overview the parallel computing, present the data distribution methodology, then we present the proposed parallel algorithm.

3.1. Overview

The computing tasks from the scientific and engineering fields are more and more complex which require huge memory and great computational speed. One way of increasing the computational speed is by designing a parallel computer systems to match this need. Where, the

```
Sequential_GS
    Input A, b, x^(0), tolerance
    for k = 0 to k_max do the following:
        for i = 1,..., n
            sum = 0
                for j = 1, 2,...., i − 1
                    sum = sum + a_ij x_j^(k+1)
                end j
                for j = i + 1,..., n
                    sum = sum + a_ij x_j^(k)
                end j
            x_i^(k+1) = (b_i − sum)/a_ii
        end i
    if ||x^(k+1) − x^(k)|| < tolerance , then output the solution, stop
end k
end Sequential_GS
```

Figure 1. Sequential Gauss-Seidel algorithm.

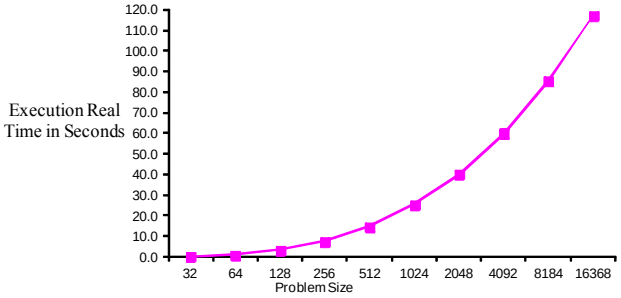

Figure 2. The relationship between the problem size and the real time needed for GS solver.

single problem split to small pieces and each processor operates on one or more pieces and the whole problem can be computed more quickly than a single processor. Different processors communicate with each other in most cases, so data message passing are not avoidable. These data and message passing is the most important factor that limits the speed of parallel computers speed. Recent development in microprocessor technology has been focused on simultaneous multi-threading [18], in the algorithm, a basic operation is recursively applied to a sequence of array data structures. Gauss-Seidel has the same type of internal data dependencies, and it is traditionally parallelized using multicolor orderings of the grid points. Unfortunately, the algorithms arising from these orderings are difficult to parallelize if cache-blocking techniques are added to increase data reuse. As an alternative, we propose to use a much more synchronization-intensive data flow parallelization technique for the standard, natural ordering of the unknowns. This type of scheme enables us to exploit recently cached data in a better way than the standard multicolor technique.

3.2. Mapping the Matrix Elements onto the Processors

Solving a linear system $Ax = b$ requires to distribute the n-by-n matrix A over a set of processors.

The effective mapping of matrix elements to processors is the key factor to efficient implementation of the algorithm in parallel. In this paper, we employ the general distribution functions suggested by Al-Towaiq [19] because it embrace a wide range of matrix distribution functions, namely column/row blocked, column/row cyclic, column/row block cyclic, and column and row (or 2D) block cyclic layouts [20-22]. The main issues of choosing a mapping are the load balancing and communication time. It was confirmed in previous studies [23] that cyclic layout offers reasonable performance and load balancing, but not the best. The better layout is 2-D block cyclic. The 2-D partitioning induces less amount of communication than the 1-D partitioning [17].

3.3. Parallel Implementation of Gauss-Seidel Algorithm

In this section, we implement and analyze the performance of the proposed algorithm via simulation, on the k-ary n-cube $\left(Q_n^k\right)$ machine.

Definition [24]: The k-ary n-cube Q_n^k has $N = k^n$ nodes each of the form $P = x_{n-1}x_{n-2}\cdots x_0$, where $0 \le x_i < k$, for all $0 \le i < n$. Two nodes $X = x_{n-1}x_{n-2}\cdots x_0$ and $Y = y_{n-1}y_{n-2}\cdots y_0$ in Q_n^k are connected if and only if there exists i, $0 \le i < n$, such that $x_i = y_i \pm 1 \pmod k$ and $x_j = y_j$, for $i \ne j$. It is shown in [25] that Q_n^k has degree $2n$ and diameter $n\lfloor k/2 \rfloor$.

Figure 3 illustrates an example of Q_2^3 network grouping.

This is apparent in the algorithm which illustrate that each processor requires information from each previous node after its computational loop is complete. That is, the first node complete its computation before the second node can begin, and similarly for each successive node (called pipe lining). Our technique does not have this problem since the algorithm requires message passing between neighboring nodes.

The proposed algorithm uses an array of size q by N, where q is the number of groups and N is the problem size. These groups chosen to reduce the communication required by the computation operations. Each group partition its nodes into interior nodes and boundary nodes. The interior nodes can operate on GS iteration loop without communication. Only boundary nodes requiring massage passing for the updates values $x^{(s+1)}$ at each step(s) of the GS iteration as follows:

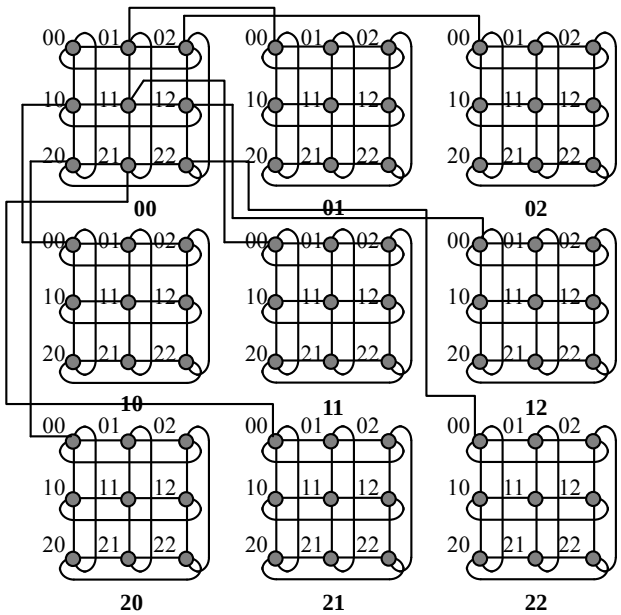

Figure 3. The OTIS-Q_2^3 network.

For $i = 2$ *to* q *do the following*
- *Group* i *sends* $x^{(s+1)}$ *back to group* $i-1$, *receives* $x^{(s)}$ *from group* $i+1$, *then receives* $x_i^{(s+1)}$ *from group* $i-1$
- *Call GS subroutine*
- *Send* $x^{(s+1)}$ *to group* $i+1$

Continue

The execution time consists of two parts; the computation time and the communication time. We utalize the following standared communication formula to estimate the communication time:

$$\text{Communication time} = (\lambda + \alpha * \beta)$$

where λ is the message latency which equal to 32×10^{-6} sec, α is the message transmision which equale to 3×10^{-3} sec and β is the number of the transition values between the boundary nodes of the groups. Using the Q_n^k communication paradigm machine, significant improvements in the performance of the algorithm were observed compared to more traditional communication paradigms that use the standard send and receive functions in conjunction with packing data into communications buffers. The GS process requires $O\left(\frac{N^3}{k^n}\right)$ computation complexity.

4. Experimental Results

In this section, we evaluate experimentally the performance of the proposed algorithm on a cluster of sixteen Linux workstations; each of which has a single Pentium IV with 128 MB of memory and 20 GB disk space. These hosts are connected together using 16-port Myrinet switch providing full-duplex 2 + 2 Gbps data rate links. The workstations use Mandrake Linux 7.2 operating system running a kernel version 2.2. 17 - 12 mk. In the implementation process we used cyclic distribution functions. In the experiments we used the matrix orders of 32, 64, ···, 16,368.

Figure 4 present the real execution time (in seconds) of the parallel GS algorithm for different problem sizes run with 4, 8, and 16 processors. All the execution times are normalized relative to the sequential GS. The expermental results show that the proposed parallel algorithm perform better for large problems.

Figure 5 present the speedup curves on each processors. The curves indicate that the speedup improves as number of processors increases.

Figure 6 shows the efficiency curves for different problem sizes. The curves indicate that our proposed algorithm is perfectly parallel it gives approximately 80% efficiency.

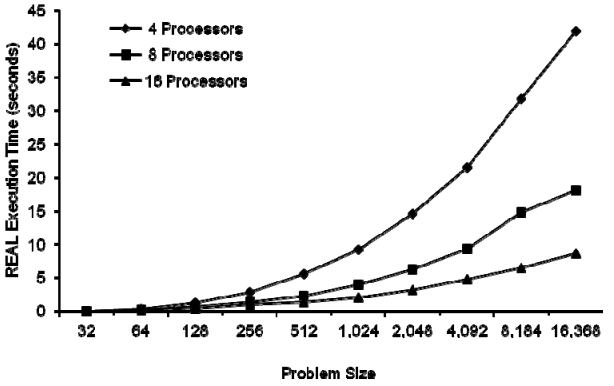

Figure 4. Cluster Gauss-Seidel algorithm real execution time.

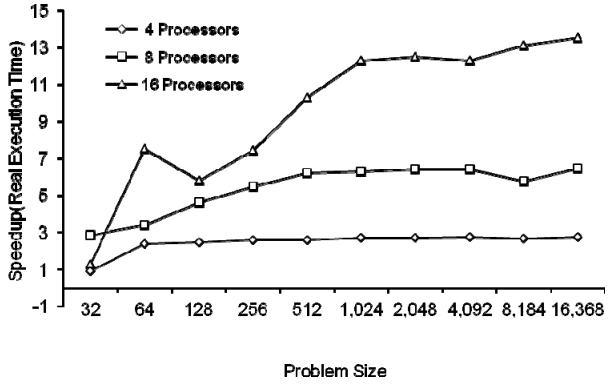

Figure 5. The real speed up ratio for 4, 8, and 16 processors of G-S algorithm.

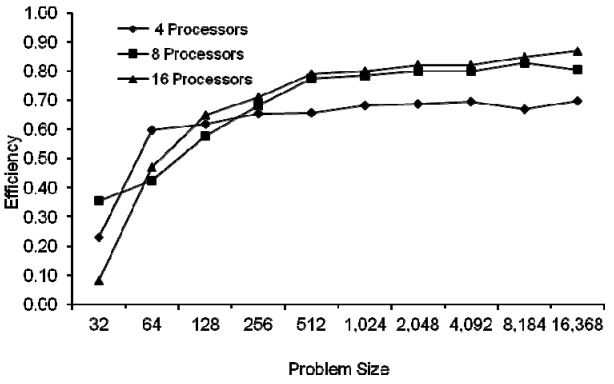

Figure 6. The efficiency curves.

5. Conclusions

In this paper, we have designed and analyzed the complexity of a parallel implementation of the Gauss-Seidel algorithm for solving a system of linear equations on a k-ary n-cube parallel machine. The proposed parallel algorithm is of $O\left(N^3/k^n\right)$ computational complexity and uses $O(nN)$ communication time to decompose a matrix of order N on the a k-ary n-cube provided $N \geq k^{n-1}$. The incurred communication time is better than the best

known results for hypercube, $O(N \log n!)$, and the mesh, $O(Nn!)$, each with approximately $n!$ nodes. The parallel algorithm takes advantage of the attractive topological properties of the k-ary n-cube in order to reduce the inter-node communication time involved during the various tasks of the parallel computation.

The numerical results show the following interesting observations:

1) Reduced time gain in the parallel algorithm.

2) Good processor load balancing.

3) Almost zero communication time done on the interior nodes, but more communication done at the boundary nodes.

4) Speedup improves using the 16 processors over the 8 and 4 processors.

5) The parallel algorithm has approximately 80% parallel efficiency.

REFERENCES

[1] L. Adams and D. Xie, "New Parallel SOR Method by Domain Partitioning," *SIAM Journal on Scientific Computing*, Vol. 20, No. 22, 1999, pp. 2261-2281.

[2] M. F. Adams. "A Distributed Memory Unstructured Gauss-Seidel Algorithm for Multigrid Smoothers," *Proceedings of 2001 ACM/IEEE Conference on Supercomputing*, Donver, 10-16 November 2001, p. 4.

[3] C. J. Hu, J. L. Zang, J. Wang, J. J. Li and L. Ding, "A New Parallel Gauss-Seidel Method by Iterative Space Alternate Tiling," 16*th International Conference on Parallel Architecture and Compilation Techniques*, Brasov, 15-19 September 2007, p. 410.

[4] M. Murugan, S. Sridhar and Sunil Arvindam "A Parallel Implementation of the Gauss-Seidel Method on the Flosolver," Technical Report, National Aeronautical Labaratory, Bangalor, 24 July 2006.

[5] L. Olszewski. "A Timing Comparison of the Conjugate Gradient and Gauss-Siedel Parallel Algorithms in a One-Dimensional Flow Equation Using PVM," *Proceedings of the 33rd Annual on Southeast Regional Conference*, Clemson, March 1995, pp. 205-212.

[6] U. Thongkrajay and T. Kulworawanichpong. "Convergence Improvement of Gauss-Seidel Power Flow Solution Using Load Transfer Technique," *Proceedings of Modelling, Identification, and Control*, Innsbruck, 11-13 February 2008,

[7] D. Wallin, H. Lof, E. Hagersten and S. Holmgren, "Multigrid and Gauss-Seidel Smoothers Revisited: Parallelization on Chip Multiprocessors," *Proceedings of ICS06 Conference*, Cairns, 28-30 June 2006.

[8] T. Kim and C.-O. Lee. "A Parallel Gauss-Seidel Method Using NR Data Flow Ordering," *Journal of Applied Mathematics and Computation*, Vol. 99, No. 2-3, 1999, pp. 209-220. doi:10.1016/S0096-3003(98)00008-3

[9] M. Adams, M. Brezina, J. Hu and R. Tuminara, "Parallel Multigrid Smoothing: Polynomial versus Gauass-Seidel," *Journal of Computational Physics*, Vol. 188, No. 2, 2003, pp. 593-610.

[10] G. Fox, M. Johnson, G. Lyzanga, S. Otto, J. Salmon and D. Walker, "Solving Problems on Concurrent Processors," Printice Hall, Upper Saddle River, 1988.

[11] G. Golub and J. M. Ortega, "Scintific Computing with an Introduction to Parallel Computing," Academic Press, Boston, 1993.

[12] R. A. Saleh, K. A. Gallivan, M. Chang, I. N. Hajj, D. Smart and T. N. Trich, "Parallel Circuit Simulation on Supercomputers," *Proceedings of the IEEE*, Vol. 77, No. 12, 1989, pp. 1915-1930. doi:10.1109/5.48832

[13] Y. Wallch, "Calculations and Programs for Power System Networks," Printice Hall, Upper Saddle River, 1986.

[14] H. Courtecuisse and J. Allard, "Parallel Dense Gauss-Seidel Algorithm on Many-Core Processors," *High Performance Computation Conference* (*HPCC*), Seoul, 25-27 June 2009, pp. 139-147.

[15] F. Wang and J. Xu, "A Crosswind Block Iterative Method for Convection-Dominated Problems," *SIAM Journal on Scientific Computing*, Vol. 21, No. 2, 1999, pp. 620-645. doi:10.1137/S106482759631192X

[16] J. Grabel, B. Land and P. Uebertholz, "Performance Optimization for the Parallel Gauss-Seidel Smoother," *Proceedings in Applied Mathematics and Mechanics*, Vol. 5, No. 1, 2005, pp. 831-832.

[17] O. Nobuhiko, M. Takeshi, I. Takeshi and S. Masaaki, "A Parallel Block Gauss-Seidel Smoother for Algebraic Multigrid Method in Edge-Element Analysis," *IEE Japan Papers of Technical Meeting on Static Apparatus*, Vol. 6, No. 58-61, 63-75, 2006, pp. 55-60.

[18] P. Kongetira, K. Aingaran and K. Olukotun. "Niagra: A 32-Way Multithreaded SPARC Processor," *IEEE Micro*, Vol. 25, No. 2, 2005, pp. 21-29. doi:10.1109/MM.2005.35

[19] M. Al-Towaiq, "Clustered Gauss-Huard Algorithm for the Solution of A*x* = *b*," *Journal of Applied Mathematics and Computation*, Vol. 184, No. 2, 2007, pp. 485-595.

[20] James, "Demmel Home Page, Applications of Parallel Computers," 1999. www.cs.brekeley.edu/~demmel/

[21] W. Lichtenstein and S. L. Johnsson, "Block Cyclic Dense Linear Algebra,"*SIAM Journal on Scientific Computing*, Vol. 14, No. 6, 1993, pp. 1257-1286. doi:10.1137/0914075

[22] M. Al-Towaiq, F. Masoud, A. B. Mnaour and K. Day, "An Implementation of a Parallel Iterative Algorithm for the Solution of Large Banded Systems on a Cluster of Workstations," *International Journal of Modeling and Simulation*, Vol. 28, No. 4, 2008, pp. 378-386.

[23] M. Al-Towaiq and H. Al-Aamri, "A Parallel Implementation of GESPP on a Cluster of Silicon Graphics Workstations," *Proceedings of the 9th IEEE International Conference on Parallel and Distributed Systems*, Hsinchu, 17-20 December 2002, pp. 226-230.

[24] K. Day, "Optical Transpose *k*-Ary *n*-Cube Networks," *Journal of Systems Architecture*, Vol. 50, No. 11, 2004, pp. 697-705. doi:10.1016/j.sysarc.2004.05.002

[25] K. Day and A. E. Al-Ayyoub, "Fault Diameter of *k*-Ary *n*-Cube Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 9, 1997, pp. 903-907. doi:10.1109/71.615436

