A Parallel Implementation of the Block-GTH algorithm*

Yuan-Jye Jason Wu†

September 2, 1994

Abstract

The GTH algorithm is a very accurate direct method for finding the stationary distribution of a finite-state, discrete time, irreducible Markov chain. O'Leary and Wu developed the block-GTH algorithm and successfully demonstrated the efficiency of the algorithm on vector pipeline machines and on workstations with cache memory. In this paper, we discuss the parallel implementation of the block-GTH algorithm and show effective performance on the CM-5.

1 Introduction

To find the stationary distribution of a finite-state, discrete time, irreducible Markov chain is a fundamental task for many probabilistic problems. This is equivalent to seeking the left eigenvector corresponding to the eigenvalue 1 of a transition matrix P of order n:

$$\pi P = \pi \ , \ \sum_{i=1}^{n} \pi_i = 1 \ . \tag{1}$$

Grassmann, Taksar and Heyman [2] introduced a direct algorithm, the *GTH algorithm*, to solve the steady-state vector π. Later Heyman [3] showed

*This work was supported by NSF Grant 91-15568.

†Applied Mathematics Programs, University of Maryland, College Park, MD 20742. yunu@cs.umd.edu

that the GTH algorithm gave very good numerical results and O'Cinneide [4] showed that the computed vector π has low componentwise relative error. Cohen, Heyman, Rabinovitch, and Brown [1] also developed a parallel implementation on the MasPar system. In order to reach high performance over a large class of computers, O'Leary and Wu [5] developed a block form algorithm, the *block-GTH algorithm*, and successfully demonstrated the efficiency of the algorithm on vector pipeline machines and on workstations with cache memory. They also proved that the block-GTH algorithm computes π with low componentwise relative error.

In this paper, we discuss the parallel implementation of the block-GTH algorithm. The machine used to implement the algorithm is the Connection Machine CM-5. In §2, we will briefly introduce the parallel computing environment. The algorithm and data partition will be stated in §3. The experimental results will be given in §4. Finally, we state conclusions in §5.

2 The CM-5 Environment

The CM-5 machines we used contain either 32 or 256 processing nodes, each of which includes a 32-megahertz SPARC processor, 32 megabytes of memory, and four high speed vector-processing units. These processing nodes are supervised by a collection of control processors which are SUN microsystems workstations running a version of the UNIX operating system. The control processors also handle the system's I/O devices, interfaces, and serial user tasks. The processing nodes can execute both single-instruction multiple-data (SIMD) and multiple-instruction multiple-data (MIMD) instructions. The basic architecture of the data network in the CM-5 is a fat-tree.

The CM-5 system comes with the CM Fortran language which is based on standard Fortran 77 supplemented with the array-processing extension of Fortran 90. CM Fortran also supports some additional intrinsic functions and data allocation statements which are useful in data parallel programming.

For example, the *complier directives*, which contain *layout, align,* and *common* statements, can specify information to the CM Fortran compiler for the allocation of arrays in the Connection Machine memory. If we declare an array in the program as

```
      REAL G(100,100)
CMF$  LAYOUT G(:SERIAL,:NEWS)
```

Then each column of the matrix G is saved in one processor only, and a single processor may contain several columns. In this manner, a *subscript* type instruction like

```
G(1,1:n)=G(1,1:n)+G(2,1:n)
```

will be executed within each processor simultaneously without interprocessor communications. This is the fundamental technique in programming SIMD code for the block-GTH algorithm.

3 The Block-GTH Algorithm

Instead of working on the original transition matrix P in (1), we seek the left eigenvector corresponding to the eigenvalue 0 of the *generator* $G = I - P$, i.e.

$$\pi G = 0 \ , \ \sum_{i=1}^{n} \pi_i = 1 \ .$$

Note that G has nonnegative off-diagonal entries and

$$Ge = 0 \ , \tag{2}$$

where e is the column vector of ones.

The GTH algorithm is a variant of Gaussian elimination and consists of two phases: LU factorization and backsubstitution. The key difference between GTH and Gaussian elimination is the way of computing the diagonal entries during the LU decomposition. Using the property of zero row sums, GTH calculates the main diagonal pivot elements as the negative row sum of off-diagonal elements.

The block-GTH algorithm is similar to block LU factorization. Suppose the block size is $l, 1 \leq l \leq n$. Let G_k be a $(n-(k-1)l) \times (n-(k-1)l)$ matrix and partition G_k as

$$G_k = \begin{bmatrix} A_k & B_k \\ C_k & D_k \end{bmatrix} , \tag{3}$$

where A_k and D_k are square matrices of order l and $(n-kl)$ respectively. Then $G_1 = G$ and we have a block LU factorization for G_k in the form

$$G_k = \begin{bmatrix} A_k & 0 \\ C_k & D_k - C_k A_k^{-1} B_k \end{bmatrix} \begin{bmatrix} 1 & A_k^{-1} B_k \\ 0 & I \end{bmatrix} .$$

Now, define G_{k+1} as

$$G_{k+1} = D_k - C_k A_k^{-1} B_k \; .$$

To check the inheritance of the zero row sums property, assume that G_k satisfies (2), i.e.,

$$[A_k \; B_k] \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = [C_k \; D_k] \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = 0 \; ,$$

where e_1 and e_2 are l and $(n - kl)$ column vectors of ones respectively. Since $A_k^{-1} B_k e_2 = -e_1$, we have

$$\begin{aligned} G_{k+1} e_2 &= D_k e_2 - C_k A_k^{-1} B_k e_2 \\ &= D_k e_2 + C_k e_1 \\ &= 0 \; . \end{aligned}$$

In addition, all off-diagonal entries of G_{k+1} are nonnegative, so G_{k+1} is a generator.

As for the backsubstitution phase, suppose we have the row vector p_{k+1} such that $p_{k+1} G_{k+1} = 0$. It is easy to verify that

$$p_k = [-p_{k+1} C_k A_k^{-1} \;\; p_{k+1}] \tag{4}$$

satisfies $p_k G_k = 0$. Finally, we compute the stationary distribution by normalizing the vector p_1.

The only task left is how to represent A_k^{-1}. Since G_k is a generator for every k, it is natural to compute a LU decomposition for A_k by the GTH algorithm. To reduce the data access time and to maintain the block structure, we can patch the column vector $B_k e_2$, which is part of the row sums, to A_k. It has been shown in [5] that a LU decomposition for A_k can be obtained by applying the GTH algorithm to the matrix

$$H_k = \begin{bmatrix} A_k & B_k e_2 \\ 0 & 0 \end{bmatrix} \; . \tag{5}$$

Suppose the matrix A_k has a LU factorization as $A_k = L_k \, U_k$, where U_k has unit diagonal entries. Then after performing the LU factorization phase in block-GTH, the generator G can be expressed as

$$G = \begin{bmatrix} L_1U_1 & 0 & & \\ & L_2U_2 & 0 & \\ C_1 & C_2 & \ddots & \\ & & & L_nU_n \end{bmatrix} \begin{bmatrix} I & (L_1U_1)^{-1}B_1 & & \\ & I & (L_2U_2)^{-1}B_2 & \\ 0 & 0 & \ddots & \\ & & & I \end{bmatrix}.$$

We now state the block-GTH algorithm formally. The notation $\lceil a \rceil$ means the smallest integer which is larger than or equal to the number a. The algorithm is excerpted and modified from [5].

Algorithm block-gth
factorization phase:

Given: Generator G, Block size l.

1. Let $G_1 = G$. Let $\hat{n} = \lceil n/l \rceil$.

2. For $k = 1, 2, ..., \hat{n}$

- **2.1.** Partition G_k as in (3), where A_k is an $l \times l$ matrix.
- **2.2.** Compute the row sums of B_k and define H_k by (5).
- **2.3.** Compute matrices L_k and U_k by applying the factorization phase of the GTH algorithm to the matrix H_k.
- **2.4.** If $k < \hat{n}$
  - **2.4.1.** Replace B_k by $(L_kU_k)^{-1}B_k$.
  - **2.4.2.** Compute $G_{k+1} = D_k - C_kB_k$.

  End if.

End for.

backsubstitution phase

3. Let $p_{\hat{n}}$ be computed from the back substitution phase of the gth algorithm applied to $H_{\hat{n}}$.

4. For $k = \hat{n}-1, \hat{n}-2, ..., 1$,

> **4.1.** Compute p_k by (4), using L_k, U_k.
>
> End for.

5. Renormalize $\pi = p_1/(p_1 e)$.

There are two steps that can be implemented by using intrinsic functions supported by the CM Fortran in the block-GTH algorithm. At step 2.2, we use the function `SUM` to calculate the row sums of the matrix B_k. It can be written as a single instruction:

```
ROWSUM=SUM(B_k(:,:), DIM=2)
```

The `DIM=2` means the summation carried out along the column dimension, and `ROWSUM` will be an l vector saving the row sums. The function `MATMUL`, matrix multiplication, can be used at step 2.4.2.

As we mentioned before, the proper use of compiler directives combined with subscript instructions will significantly improve the performance of a CM Fortran program. We now have to decide in which way the generator G is allocated.

There are three possibilities:

`G(:NEWS,:NEWS)`, `G(:NEWS,:SERIAL)`, and `G(:SERIAL,:NEWS)`.

(`G(:SERIAL, :SERIAL)` produces a serial implementation.) Based on our experiments, we find `G(:SERIAL,:NEWS)` has better performance than the other two. The `G(:NEWS,:NEWS)` case makes the function `MATMUL` faster but costs too much interprocessor communication in the other steps. As for `G(:NEWS,:SERIAL)`, it does not fully utilize the parallel processors since the optimal block size usually is much smaller than the order n of the generator. Thus, the column dimension of the matrix B_k is usually larger than the row dimension. To declare the second axis to be serial does not achieve parallel performance at step 2.2 and 2.4.1.

4 Experimental Results

In this section, we will present the experimental results on the CM-5. To make a comparison, we also show the serial performances with the same generators on a SUN SPARCstation IPX which has a 64k byte cache memory.

Two technical remarks have to be noted when we compile the code:

1. The CM-5 run time system uses a *block layout* which divides the array into contiguous *subgrid* blocks of array elements and distributes one subgrid block per vector unit. Since the length of vector unit is eight, the natural subgrid block size is a multiple of 8. This is referred to as *vector length padding*. To make the array layout flexible, we compile the code with the option *-nopadding*.

2. The standard Fortran 90 function `MATMUL` does not utilize the vector units, but `MATMUL` is specially optimized in the CM Scientific Software Library (CMSSL). One can link in the CMSSL library by compiling code with option *-lcmsslcm5vu*.

First, we tested four different sizes of generator, i.e., $n = 512, 800, 1024,$ and 2048. The block size was varied as $l = 4, 8, 12, \ldots, n$ on the CM-5 with 32 nodes. For the SUN machine, we varied the block size from 1 to some predicted optimal size (see [5]) with increment 1. Once it passed the predicted number, we changed the increment to 20. Table 1 shows the timings, optimal block size, megaflops rate and speedup, defined by

$$\text{speedup} = \frac{\text{the times on CM-5}}{\text{the times on SUN SPARC IPX}} .$$

We can see the speedup increases from 8.53 to 48.78 when the problem size become larger. Next, we tested larger problem size $n = 4096$ and 8192 on the CM-5 with 256 nodes. Table 2 gives the timings, block size, and megaflops rate.

Figure 1 shows timing data for the block-GTH algorithm on the CM-5 for problem size 1024. The backsubstitution phase cost only 0.10-0.16 second, so we do not plot it in the figure. For small block size, the most expensive steps are matrix multiplication and row sums, while LU decomposition and triangular solve are expensive when the block size is large. Initially, the cost of matrix multiplication and row sums drops faster than LU decomposition and triangular solve grow, and the optimal block size is about 100. The jaggedness of the time curve for triangular solving for matrix B_k (step 2.4.1) is caused by the loops at step 2.4. When $\lceil n/l \rceil$ drops by one as the block size l increases, we will save one triangular solve. Therefore, the time drops when the block size l evenly divides n.

Figure 2 gives the total time as a function of block size on both the CM-5 and the SUN. Note that the program performs the GTH algorithm when the block size is equal to the problem size. The parallel GTH has better performance than the serial block-GTH, and the parallel block-GTH is even better.

5 Conclusion

We have produced an efficient parallel implementation of the block-GTH algorithm, distributing the matrix G by columns, and demonstrated that it is a very effective algorithm for parallel computation.

6 Acknowledgements

I thank Dianne P. O'Leary and Jerry Sobieski for very helpful comments.

References

[1] David M. Cohen, Daniel P. Heyman, Asya Rabinovitch, and Danit Brown. A parallel implementation of the GTH algorithm. Bellcore, Morristown, New Jersy, unpublished notes, 1994.

[2] W. K. Grassmann, M. I. Taksar, and D. P. Heyman. Regenerative analysis and steady state distributions. *Operations Research*, 33:1107–1116, 1985.

[3] Daniel P. Heyman. Further comparisons of direct methods for computing stationary distributions of Markov chains. *SIAM J. Alg. Disc. Meth.*, 8 No.2:226–232, 1987.

[4] C. A. O'Cinneide. Entrywise pertrubation theory and error analysis for Markov chains. *Numerische Mathematik*, to appear.

[5] Dianne P. O'Leary and Yuan-Jye J. Wu. A block-GTH algorithm for finding the stationary vector of a Markov chain. Technical report, Inst. for Advanced Computer Studies Report TR-93-123, Computer Science Department Report TR-3182, University of Maryland, College Park, 1993.

Problem Size n	SUN Actual Optimal block	SUN Time (sec)	SUN Mflop Rate	CM-5 (32 nodes) Actual Optimal block	CM-5 (32 nodes) Time (sec)	CM-5 (32 nodes) Mflop Rate	Speedup
512	21	22.30	4.03	32	2.61	34.36	8.53
800	16	79.00	4.33	48	6.58	52.01	12.00
1024	14	193.02	3.72	64	9.85	72.83	19.60
2048	140	1981.18	2.85	128	40.61	139.10	48.78

Table 1: Timings, optimal block size, megaflops rate, and speedup for generators of order 512, 800, 1024, and 2048

CM-5 (256 nodes)			
Problem Size	Block Size	Time (sec)	Mflop Rate
4096	128	38.20	1199.82
8192	256	153.13	2394.11

Table 2: Timings, block size, and megaflops rate for generators of order 4096 and 8192

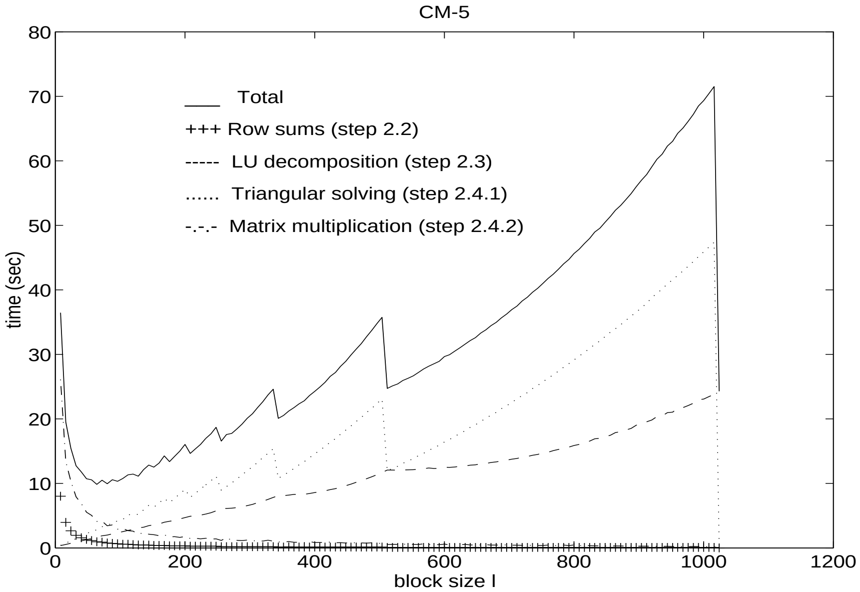

Figure 1: Detailed block-GTH time as a function of block size for generator of order 1024

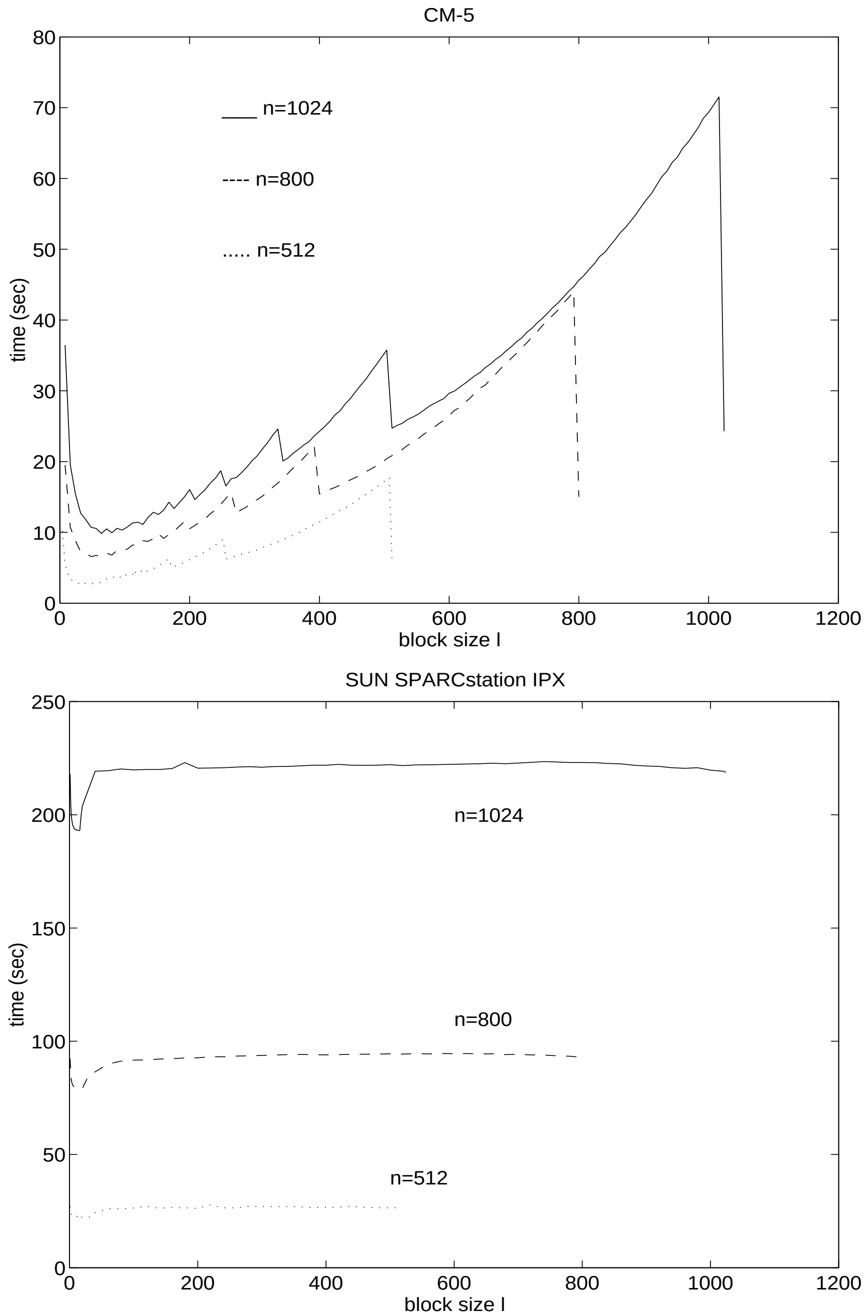

Figure 2: Block-GTH time as a function of block size for generator of order 512, 800, and 1024