An hybrid parallel algorithm for solving tridiagonal linear systems versus the Wang's method in a Cray T3D BSP computer [1]

JOAN-JOSEP CLIMENT, LEANDRO TORTOSA, ANTONIO ZAMORA
Departament de Ciència de la Computació i Intel·ligència Artificial
Universitat d'Alacant
Ap. 99 E-03080 Alacant
SPAIN
jcliment@dccia.ua.es, tortosa@dccia.ua.es, zamora@dccia.ua.es

Abstract: In this paper we describe an hybrid algorithm for an even number of processors based on an algorithm for two processors and the Overlapping Partition Method for tridiagonal systems. Moreover, we compare this hybrid method with the Partition Wang's method in a BSP computer. Finally, we compare the theoretical computation cost of both methods for a Cray T3D computer, using the cost model that BSP model provides.

Key-words: Tridiagonal linear systems, BSP, Wang's method, Overlapping Partition (OPM) method.

1 Introduction

Tridiagonal matrices arise in many practical scientific and engineering problems. For instance, tridiagonal matrices have appeared in the error analysis of numerical solutions of two-point boundary problems associated with ordinary differential equations employing finite difference methods. These systems can be solved efficiently in sequential computers but are difficult to solve efficiently in parallel computers, where the communications take a significant part of the total execution time. This fact, together with the fast progress that parallel computing has experimented in the last decades, has increased the interest and efforts towards the development of fast and efficient algorithms for solving such systems. The *Bulk Synchronous Parallel (BSP) Computing* model, introduced by Valiant [9], is a style of parallel programming developed for general purpose parallelism; that is, parallelism across all the application areas and a wide range of architectures.

We consider the problem of solving a linear system,

$$A\boldsymbol{x} = \boldsymbol{y}, \tag{1}$$

where

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}$$

is a tridiagonal and nonsingular matrix, and $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]^T$ is the right-hand side vector. We also assume that matrix A is unreduced; that is, $b_i \neq 0$ and $c_{i+1} \neq 0$, for $i = 1, 2, \ldots, n-1$. Moreover, A is strictly diagonally dominant; that is, $|a_i| \geq |b_i| + |c_i|$, for $i = 1, 2, \ldots, n$, with $c_1 = b_n = 0$.

2 The BSP model

A BSP computer consists of the following elements: a set of p sequential processor-memory pairs, a global communication network that allows processors to have access to non local data, and a mechanism for the global synchronization of all the processors. The BSP programming model is based on the concept of *superstep*. A BSP program is a sequence of supersteps and each one can be decomposed into three phases: local computation using local data within each processor-memory pair; communication actions amongst the processors; and the last one is a barrier synchronization, which waits for all the communication actions to be completed. At the end of a superstep, after the barrier synchronization is done, the data that were moved to the local memory of destination processors are available. If the maximum number of incoming or outgoing messages per processor is h, then such a communication pattern is called an h-relation.

The cost model summarizes in a very reduced number of parameters the differences among the different parallel architectures. If we define the time step as the time required for a single local operation such as an addition or a multiplication in floating point, then a BSP computer can be characterized by the following four parameters: p, the number of processors; s, the processor speed (the number of time steps per second); l, the synchronization cost, and g, where gh is the number of flops required for the processors to perform an h-relation.

The computational cost of a BSP algorithm is eas-

[1]This work was partially supported by Spanish DGES grants PB97-0334 and PB98-0977

ily obtained as the sum of all individual supersteps. The cost of a single superstep measured in time steps or flops, is upper bounded by $w + hg + l$ where w is the arithmetic cost, and hg is the communication cost. The total cost (in flops) of a BSP algorithm is given by the addition of the individual cost of all the supersteps needed to implement it; that is, is upper bounded by $\Omega + Hg + Kl$ where Ω is the total arithmetic cost of the algorithm, H is the total sum of the words circulating through the network along the execution of the computation, and K is the total number of supersteps.

3 An hybrid parallel method to solve tridiagonal systems

3.1 A method for two processors

Assume that $n = 2q$, for some $q \geq 1$ and consider matrix A factorized as

$$A = MDV, \tag{2}$$

where A, M, D and V are partitioned into two blocks of size $q \times q$. Matrix D is diagonal, and first block of M (respectively, V) is lower bidiagonal (respectively, upper bidiagonal). Second block of M (respectively, V) is upper bidiagonal (respectively, lower bidiagonal). Elements in the upper diagonal of both matrices M and V (respectively, lower diagonal) are labelled as u_i (respectively, l_i) while elements in the main diagonal of D are labelled as d_i. If we proceed as in the LDU factorization of matrix A (see for example Golub and Van Loan [3]), then the elements of M, D and V are computed as follows:

- Let $d_1 = a_1$ and $d_n = a_n$.
- For $i = 1, 2, \ldots, q-1$, let

$$\left.\begin{array}{rcl} u_i & = & \frac{b_i}{d_i}; \\ l_{i+1} & = & \frac{c_{i+1}}{d_i}; \\ d_{i+1} & = & a_{i+1} - l_{i+1} d_i u_i \\ & = & a_{i+1} - \frac{c_{i+1} b_i}{d_i}. \end{array}\right\} \tag{3}$$

- Let

$$\begin{array}{rcl} u_q & = & \frac{b_q}{d_q}, \\ l_{q+1} & = & \frac{c_{q+1}}{d_q}. \end{array}$$

  Compute

$$\begin{array}{rcl} l_n & = & \frac{c_n}{d_n}, \\ u_{n-1} & = & \frac{b_{n-1}}{d_n}. \end{array}$$

- For $i = n-1, n-2, \ldots, q+2$, compute

$$d_i = a_i - l_{i+1} d_{i+1} u_i = a_i - \frac{c_{i+1} b_i}{d_{i+1}} \tag{4}$$

  and let

$$\begin{array}{rcl} l_i & = & \frac{c_i}{d_i}; \\ u_{i-1} & = & \frac{b_{i-1}}{d_i}. \end{array}$$

- Finally, compute

$$d_{q+1} = a_{q+1} - \frac{c_{q+1} b_q}{d_q} - \frac{c_{q+2} b_{q+1}}{d_{q+2}}. \tag{5}$$

To implement the above computation in a computer with two processors we use a technique similar to those developed by Van der Vorst [10] for symmetric matrices. To solve system (1) using the factorization (2) we need to solve system $MD\boldsymbol{z} = \boldsymbol{y}$, for $\boldsymbol{z}$ and then solve system $V\boldsymbol{x} = \boldsymbol{z}$, for $\boldsymbol{x}$. As a consequence of the structure of matrices M and D the vector $\boldsymbol{z} = [z_1, z_2, \ldots, z_n]^T$ can be obtained directly as

$$\begin{array}{rcll} z_1 & = & \frac{y_1}{d_1}; \quad z_n = \frac{y_n}{d_n}, & (6) \\ z_i & = & \frac{y_i - c_i z_{i-1}}{d_i}, & (7) \\ & & i = 2, 3, \ldots, q, & \\ z_i & = & \frac{y_i - b_i z_{i+1}}{d_i}, & (8) \\ & & i = n-1, n-2, \ldots, q+2, & \\ z_{q+1} & = & \frac{y_{q+1} - c_{q+1} z_q - b_{q+1} z_{q+2}}{d_{q+1}}. & (9) \end{array}$$

Now, as a consequence of the structure of matrix V, when we solve $V\boldsymbol{x} = \boldsymbol{z}$, for $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$ we obtain that $x_{q+1} = z_{q+1}$ and then

$$\begin{array}{rcll} x_i & = & z_i - \frac{b_i x_{i+1}}{d_i}, & i = q, q-1, \ldots, 1, \quad (10) \\ x_i & = & z_i - \frac{c_i x_{i-1}}{d_i}, & i = q+2, \ldots, n. \quad (11) \end{array}$$

The above computation suggest the following BSP algorithm for two processors. We assume that matrix A and vector $\boldsymbol{y}$ are stored in the main processor P_0.

Algorithm 1 *Parallel BSP algorithm for two processors.*

Superstep 1

Processor P_0 sends to processor P_1 the elements a_i, b_i, c_i, y_i, for $i = q+1, q+2, \ldots, n$.

Superstep 2 *Compute the elements d_i and z_i.*

- *In processor P_0,*
  - *Let $d_1 = a_1$ and compute z_1 using (6).*
  - *For $i = 2, 3, \ldots, q$, compute d_{i-1} and z_i using (3) and (7).*
  - *Processor P_0 sends to processor P_1 the elements a_q, b_q, d_q and z_q.*
- *In processor P_1,*
  - *Let $d_n = a_n$ and compute z_n using (6).*
  - *For $i = n-1, n-2, \ldots, q+2$, compute d_i and z_i using (4) and (8).*
  - *Processor P_1 sends to processor P_0 the elements d_{q+2} and z_{q+2}.*

Superstep 3 *Compute the elements d_{q+1}, z_{q+1} and the solution.*

- *Both processors compute d_{q+1} and z_{q+1} according to (5) and (9), respectively. Let $x_{q+1} = z_{q+1}$. Then,*
  - *In processor P_0 compute x_i, for $i = q, q-1, \ldots, 1$, using (10).*
  - *In processor P_1, compute x_i, for $i = q+2, \ldots, n$, using (11).*
- *Processor P_1 sends to processor P_0 the components of the solution x_i, for $i = q+2, \ldots, n$.*

The cost of Algorithm 1 is $\left(\frac{9}{2}n + 6\right) + \left(\frac{5}{2}n + 5\right) g + 3l$ flops (see [2]).

3.2 Generalization for p processors

In this paper we propose an hybrid parallel algorithm based on Algorithm 1 and the OPM method (see [6]). Consider system (1) partitioned into $\frac{p}{2}$ blocks as figure 1. The OPM method consider a new partition based on the above partition, adding $2m$ equations (respectively, components) to each general central block of coefficient matrix (respectively, right-hand side vector), and m equations (respectively, components) to the first and last block of coefficient matrix (respectively, right-hand side vector). The new blocks are now overlapped one each other as a consequence of the new partition. Therefore, we can rewrite the new subsystems as $\hat{A}_i \hat{\boldsymbol{x}}_i = \hat{\boldsymbol{y}}_i$, for $i = 1, 2, \ldots, \frac{p}{2}$. We slightly modify this partition adding $2m$ equations (respectively, components) to the first and last blocks of the coefficient (respectively, right-hand side vector). So, each of the subsystems has the same size. The OPM method proposes to solve each of these intermediate systems in the processors using the LU factorization. Nevertheless, instead of this technique we propose to apply Algorithm 1 to solve modifyed systems in each pair of processors (P_{2i-2}, P_{2i-1}) for $i = 1, 2, \ldots, \frac{p}{2}$.

We can describe the method in three phases.

Phase 1. Each pair of processors (P_{2i-2}, P_{2i-1}) for $i = 1, 2, \ldots, \frac{p}{2}$ receive $\hat{A}_i$ and $\hat{\boldsymbol{y}}_i$. Each processor can receive in a unique communication step the needed data to run Algorithm 1 in next phase. The total number of data received by each one of the p processors are $4t$, with

$$t = \frac{n}{p} + m = k + m.$$

In order to perform the communication in one step, we introduce a new vector of p components that we call $\boldsymbol{row}$, which represents the number of the first row that each processor must receive from the main one. That is, the ith component of this vector represents the number of the first row that must be communicated from the main processor to the processor P_i. Observe that this vector is computed in the main processor. We can set the following algorithm to compute the components of vector $\boldsymbol{row}$.

Algorithm 2 *Computation of vector $\boldsymbol{row}$.*
$\boldsymbol{row}(0) = 1$.
IF $p > 4$ *THEN*
  FOR $i = 1, 2, \ldots, \frac{p}{2} - 2$,
    $\boldsymbol{row}(2i) = \dfrac{2in}{p} - m + 1$,
  END
ENDIF
$\boldsymbol{row}(p-2) = n - \dfrac{2n}{p} - 2m + 1$
FOR $i = 1, 2, \ldots, \frac{p}{2}$,
    $\boldsymbol{row}(2i-1) = \boldsymbol{row}(2i-2) + t$,
END.

In the same communication step, we must send from main processor to the remaining ones the values of parameters m and t because they will be required in further computations.

Phase 2. In this phase each pair of processors (P_{2i-2}, P_{2i-1}), for $i = 1, 2, \ldots, \frac{p}{2}$, execute Algorithm 1 for the elements that have been received in phase 1, except the initial and last communication of the algorithm.

$$\begin{bmatrix} A_1 & B_1 & & & & \\ C_2 & A_2 & B_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & C_{\frac{p}{2}-1} & A_{\frac{p}{2}-1} & B_{\frac{p}{2}-1} & \\ & & & C_{\frac{p}{2}} & A_{\frac{p}{2}} & \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_{\frac{p}{2}-1} \\ \boldsymbol{x}_{\frac{p}{2}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_{\frac{p}{2}-1} \\ \boldsymbol{y}_{\frac{p}{2}} \end{bmatrix}$$

Figure 1:

Phase 3. After executing Algorithm 1 in phase 2, each processor has obtained a partial solution vector of t components. Now, the objective of this phase is to send from each processor to the main one the desired components of the partial solution vector, accordingly with the OPM method. We take all the components from the first and the last processor; the last $t-m$ components from the even processors, except for the last even processor, from which we take the last $t-2m$ components. Finally, we take the first $t-m$ components from the odd processors, except for the first even processor, from which we take the first $t-2m$ components.

Now, our objective is to derive a method that allow us to build in the main processor the final solution vector of (1), which contains the desired components of each partial solution vector that have been computed in each processor in Phase 2. To accomplish this, we define, in each processor, three new variables called as *row2*, *numeqs* and *jump*. The first variable *row2* takes, in each processor, the initial row number of the first component in its partial solution vector that become as part of the final solution in the main processor. The second variable, called *numeqs*, represents the number of components of the partial solution vector, in each processor, which must be transmitted to the main one as part of the final solution. The third variable, called *jump*, give us the position, in the solution vector of main processor, where the first choose component of the partial solution vector of each processor must be placed. The following algorithm computes the values of these variables.

Algorithm 3 *Computations required to obtain in each processor the variables row2, numeqs and jump.*

- *Processor* P_0, $row2 = 1$, $numeqs = t$, $jump = 0$.
- *Processor* P_1, $row2 = 1$, $numeqs = t - 2m$, $jump = t$.
- *Processor* P_{2i}, *for* $i = 1, 2, \ldots, \frac{p}{2} - 2$, $row2 = m + 1$, $numeqs = t - m$, $jump = \frac{in}{p}$.
- *Processor* P_{2i-1}, *for* $i = 2, \ldots, \frac{p}{2} - 1$, $row2 = 1$, $numeqs = t - m$, $jump = \frac{(i-1)n}{p} + t - m$.
- *Processor* P_{p-2}, $row2 = 2m + 1$, $numeqs = t - 2m$, $jump = \frac{(p-2)n}{p}$.
- *Processor* P_{p-1}, $row2 = 1$, $numeqs = t$, $jump = n - t$.

The following BSP algorithm implements the new parallel method for solving tridiagonal systems, based on the three phases we have described above.

Algorithm 4 *A BSP parallel algorithm for solving tridiagonal systems for* p *processors.*

Superstep 1 *Start and initial communication.*

- *Compute* m *and* t *in* P_0 *(see [6]).*
- *Each of the processors receive from main processor the corresponding elements from* A *and* $\boldsymbol{y}$*, using the vector* $\boldsymbol{row}$*, given by Algorithm 2.*
- *Main processor sends* m *and* t *to the remaining ones.*

Superstep 2 *Execution of Algorithm 1 in each pair of processors.*

- *For* $i = 1, \ldots, \frac{p}{2}$, *processors* (P_{2i-2}, P_{2i-1}) *execute supersteps* 2 *and* 3 *of Algorithm 1, except for the final communication of the solutions at the end of superstep* 3*.*
- *Each processor computes row2, numeqs and jump, according with Algorithm 3.*

- *Main processor obtains the solution vector from the partial solution vector in each processor, using the variables row2, numeqs and jump.*

The computational cost of the Algorithm 4, (see [2]), is

$$\begin{aligned}&(3n + 9k + 9m + 3p + 30)\\ &+ \quad (5n - 5k - 5m + 4mp + 5p - 2)]g \quad (12)\\ &+ \quad 3l \quad \text{flops.}\end{aligned}$$

4 Wang's Method to solve tridiagonal systems

For a detailed description of the Wang's method, see [11]. We can briefly describe the method saying that we proceed simultaneously to eliminate the elements located up and below the main diagonal of coefficient matrix, carrying out the necessary elementary operations until finally A is diagonalized. The nonzero elements of the subdiagonal blocks appearing in the first stage of the Wang's method are labelled as f_i, for $i = 1, 2, \ldots, n$.

We carried out a modification of the original method consistent in updating, at the same time, in all the processors the nonzero elements g_{ki}, for $i = 1, 2, \ldots, p-1$, which appear throughout the process in the superdiagonal blocks . The updating of the elements in the right-hand side vector is achieved in processors $P_1, P_2, \ldots, P_p$ as follows

$$\begin{aligned}r_p &= d_n,\\ r_t &= d_{kt} - \frac{g_{kt}}{a_{k(t+1)}} r_{t+1}, \quad t = p-1, p-2, \ldots, 2.\end{aligned}$$

The processor P_i, for $i = 1, 2, \ldots, p-1$, updates the element d_{ki} from the received elements from processors P_j, for $j = i+1, i+2, \ldots, p$, by means of the following elementary operation

$$d_{ki} = d_{ki} - \frac{g_{ki}}{a_{k(i+1)}} r_{i+1}. \qquad (13)$$

Once these elements have been updated, we proceed to eliminate the off diagonal elements in each of the diagonal blocks and we modify the components $d_{k(i-1)+j}$, for $j = 1, 2, \ldots, p-1$. In this way, we avoid the loosing of parallelism when we develop the final computations, which lead us to obtain a diagonal matrix. So we save a step of communication to the main processor. The following algorithm resumes the characteristics exposed above.

Algorithm 5 *A BSP Wang's partition algorithm for tridiagonal systems.*

Superstep 1

Processor P_i, for $i = 2, 3, \ldots, p$, receives block A_i, the elements b_{ki}, c_{ki+1} and the vector $[d_{k(i-1)+1}, d_{k(i-1)+2}, \ldots, d_{ki}]^T$.

Superstep 2

- *For $i = 1, 2, \ldots, p$,*
  - *Processor P_i vanishes the elements $c_{k(i-1)+j}$, for $j = 2, 3, \ldots, k$.*
  - *Processor P_i vanishes the elements $b_{k(i-1)+j}$, for $j = k-2, k-3, \ldots, 1$.*
- *For $i = 2, 3, \ldots, p$, processor P_i receives the elements a_{kj}, d_{kj} and b_{kj}, for $j = 1, 2, \ldots, p$ with $j \neq i$ from the remaining ones.*

Superstep 3

For $i = 1, 2, \ldots, p-1$, processor P_i vanishes the elements b_{ki} and receives the updated elements a_{ki} and d_{ki} from the remaining processors.

Superstep 4

For $i = 1, 2, \ldots, p-1$,

- *Processor P_{i+1} vanishes the elements f_{ki+j}, for $j = 1, 2, \ldots, k$.*
- *Processor P_i receives the elements g_{ki}, $a_{k(i+1)}$, $d_{k(i+1)}$.*

Superstep 5

- *For $i = 1, 2, \ldots, p-1$,*
  - *Processor P_i updates the element d_{ki} according with (13) and then vanishes g_{ki}.*
  - *Processor P_i vanishes the elements $g_{k(i-1)+j}$, for $j = 1, 2, \ldots, k-1$.*
- *For $i = 1, 2, \ldots, p$, processor P_i computes*

$$x_i = \frac{d_{k(i-1)+j}}{a_{k(i-1)+j}}, \quad \text{for} \quad j = 1, 2, \ldots, p.$$

- *Each processor sends its partial solution to the main one.*

Table 1: BSP parameters for a CRAY T3D.

	s	p	l	g	$N_{1/2}$
CRAY T3D	12	32	201	1.1	28
		64	148	1.0	27
		128	301	1.1	20
		256	387	1.2	15

The computational cost of algorithm 5 is, (see [1]),

$$\begin{aligned}&(21k+3p-18)\\ &+\quad (5n-5k+8p^2-16p+8)g\\ &+\quad 5l \quad \text{flops.}\end{aligned} \tag{14}$$

5 Numerical results

In this section, we compare theoretical predicted times for Algorithm 4 and Wang's method (a fast and classical method to solve tridiagonal linear systems) on a Cray T3D using the cost model provided by the BSP model. The BSP parameters, for this machine, are resumed in Table 1.

To compute the parameter g, we follow the model proposed by Hockney [5], where g is defined as a function of the message size r as

$$g = \left(1 + \frac{N_{1/2}}{r}\right) g_\infty$$

with g_∞ the asymptotic communication cost for very large messages and $N_{1/2}$ is the size of message that produces half the optimal bandwidth of the machine.

In Table 2 theoretical cost measured in seconds is presented for 32, 64, 128, and 256 processors. These times are computed using expressions (12) and (14), for different sizes of the coefficient matrix n in a range that varies from 4096 to 4194304.

As we can see in Table 2, when the number of processors is high the method proposed in Section 3 improves the execution times given by the Wang's method, for some sizes. Observe that for $p = 128$, Wang's method is slower than the other one for sizes less than 32768; when the number of processors increase to 256, the size increase to 131072, as we show in Figures 2 and 3.

References:

[1] Climent, J.J., Tortosa, L., Zamora, A.: Comparing the BSP cost of different algorithms for tridiagonal systems. Report DTIC-97/06 Universitat d'Alacant (1997)

[2] Climent, J.J., Tortosa, L., Zamora, A.: A new BSP algorithm for tridiagonal systems. Actas de las IX Jornadas de Paralelismo San Sebastian (1998) 183–190

[3] Golub,G.H., Van Loan,C.: Matrix Computations. Johns Hopkins University Press Baltimore (1989)

[4] Hill, J.M., Crumpton, P.I., Burgess, D.A.: Theory, practice, and a tool for BSP performance prediction. EuroPar'96 **1124** in Lecture Notes in Computer Science Springer-Verlag (1996) 697–705

[5] Hockney, R.: Performance Parameters and benchmarking of supercomputers. Parallel Computing **17** (1991) 1111–1130

[6] Larriba, J.L., Jorba, A., Navarro, J.J.: Solution of strictly diagonal dominant tridiagonal systems on vector computers. Report CEPBA Universitat Politècnica de Catalunya 93/09 (1993)

[7] Miller, R.: A library for bulk-synchronous parallel programming. Proceedings of the British Computer Society Parallel Processing Specialist Group Workshoop on General Purpose Parallel Computing (1993)

[8] Miller, R., Reed, J.L.: The Oxford BSP library users' guide. Technical report Programming Research Group University of Oxford (1993)

[9] Valiant, L.G.: A Bridging Model for Parallel Computation. Communications of the ACM **33** (1990) 103–111

[10] Van der Vorst, H.A.: Analysis of a parallel solution method for tridiagonal linear systems. Parallel Computing **5** (1987) 303–311

[11] Wang, H.H.: A parallel method for tridiagonal equations. ACM Transactions on Mathematical Software **7** (1981) 170–183

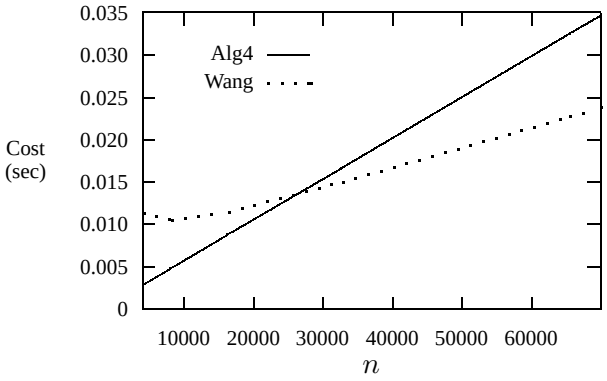

Figure 2: Theoretical times measured in a CRAY T3D for $p = 128$ with $4096 \leq n \leq 65536$.

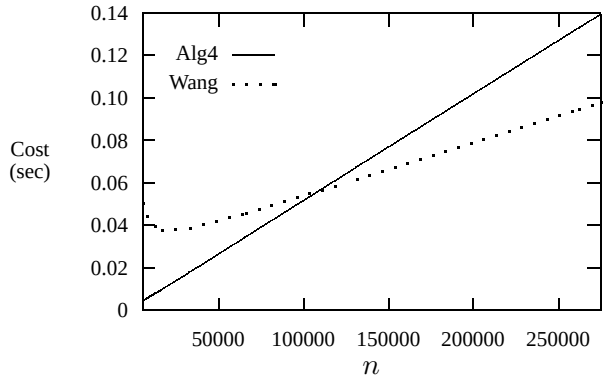

Figure 3: Theoretical times measured in a CRAY T3D for $p = 256$ with $4096 \leq n \leq 262144$.

Table 2: Theoretical times measured in a CRAY T3D for 32, 64, 128 and 256 processors.

	$p = 32$		$p = 64$		$p = 128$		$p = 256$	
n	Alg. 4	WANG	Alg. 4	WANG	Alg. 4	WANG	Alg. 4	WANG
4096	0.0023	0.0019	0.0024	0.0033	0.0029	0.0113	0.0038	0.0526
8192	0.0044	0.0029	0.0043	0.0039	0.0048	0.0105	0.0057	0.0415
16384	0.0084	0.0052	0.0081	0.0057	0.0088	0.0115	0.0097	0.0375
32768	0.0165	0.0097	0.0158	0.0094	0.0167	0.0150	0.0179	0.0386
65536	0.0328	0.0188	0.0311	0.0170	0.0325	0.0227	0.0344	0.0455
131072	0.0652	0.0369	0.0617	0.0322	0.0642	0.0384	0.0672	0.0615
262144	0.1302	0.0732	0.1228	0.0627	0.1275	0.0699	0.1330	0.0946
524288	0.2601	0.1457	0.2452	0.1236	0.2542	0.1331	0.2646	0.1615
1048576	0.5198	0.2908	0.4900	0.2455	0.5076	0.2594	0.5278	0.2956
2097152	1.0393	0.5809	0.9794	0.4892	1.0143	0.5122	1.0541	0.5638
4194304	2.0783	1.1612	1.9584	0.9766	2.0277	1.0177	2.1068	1.1004