CRAY-2에서 멀티/마이크로 태스킹 라이브러리를 이용한 선형시스템의 병렬해법

마 상 백[†]

요 약

CRAY 에서 멀티/마이크로 태스킹은 다수의 CPU를 이용하여 계산속도를 증가시키는 하나의 방법이다. CRAY-2 에는 4개의 CPU 가 있으므로 적절히 설계된 알고리즘을 가지고 최대 4배의 speedup을 실현할 수 있다. 저자는 이 논문에서 CRAY-2에서 멀티태스킹/마이크로태스킹 라이브러리를 이용한 2가지의 선형시스템의 해의 병렬화를 제시한다. 하나는 조밀행렬에 대한 가우스 소거법이고 다른 하나는 Radicati di Brozolo가 제안한 준비행렬을 이용한 대형이산 행렬의 반복적 해법이다. 첫째 경우에 크기가 600인 행렬에서 2개의 CPU에 멀티태스킹을 이용하여 1.3의 speedup 을 얻었으며 두 번째 경우에서는 크기가 8192인 행렬에서 4개의 CPU에 마이크로 태스킹을 사용하여 3이상의 speedup을 얻었다. 첫째 경우에서는 비균일한 벡터길이 때문에 speedup 이 제한되었다. 두 번째 경우에서는 Radicati 의 테크닉을 혼합한 ILU(0) 준비행렬은 4개의 프로세서에서 상당히 높은 speedup을 얻었다.

Parallel solution of linear systems on the CRAY-2 using multi/micro tasking library

Sang Back Ma[†]

ABSTRACT

Multitasking and microtasking on the CRAY machine provides still another way to improve computational power. Since CRAY-2 has 4 processors we can achieve speedup up to 4 with properly designed algorithms. In this paper we present two parallelizations of linear system solution on the CRAY-2 with multitasking and microtasking library. One is the LU decomposition on the dense matrices and the other is the iterative solution of large sparse linear systems with the preconditioner proposed by Radicati di Brozolo. In the first case we realized a speedup of 1.3 with 2 processors for a matrix of dimension 600 with the multitasking and in the second case a speedup of around 3 with 4 processors for a matrix of dimension 8192 with the microtasking. In the first case the speedup is limited because of the nonuniform vector lengths. In the second case the ILU(0) preconditioner with Radicati's technique seem to realize a reasonably high speedup with 4 processors.

† 정 회 원 : 한양대학교 공과대학 전자계산학과

논문접수 : 1997년 1월 7일, 심사완료 : 1997년 9월 24일

1. Introduction

In this paper we address the problem of the vectorizing and parallelizing the solution of the dense and sparse linear systems on a CRAY-2. The sparse linear systems arise in the discretization of the partial differential equations, and due to their special structure they need special attentions, otherwise the cost of the sparse linear systems with the LU-type methods becomes so prohibitive. Iterative methods offer good alternatives since they require minimal memory and are inherently amenable to parallel execution. For the dense linear systems we use the classical LU decomposition algorithm and we parallelized on the 2 processors. For sparse linear systems we combined the vectorizing on a processor and parallelizing on 4 processors.

The CRAY-2 is a four-processor machine. Each processor can execute independent tasks concurrently. All processors have equal access to the large central memory. The CRAY-2 at Minnesota Supercomputer Center has 512 Megawords of central memory. Each CRAY-2 processor has 8 vector registers(each 64 words long) and has data access through a single path between its vector registers and main memory. Each processor has 16 K words of local memory with no direct path to central memory but with a separate data path between local memory and its vector registers. Also there are six parallel pipelines: common memory to vector register, load/store vector register to local memory, load/store floating addition/subtraction, floating multiplication/division, integer addition/subtraction and logical pipelines. The central memory is divided into four quadrants, and assignment of four quadrants to four processors takes place and changes at each memory cycle. Hence if more than one processor request an access to the same quadrant, then a memory conflict will happen.

The CRAY-2 offers the microtasking and multitasking for the parallel execution involving more than one processors. Microtasking is suitable for parallel execution of tasks with small granular size, especially the inner do-loop, and multitasking for the tasks with large grain size. Microtasking has less overhead, hence more efficient than the multitasking.

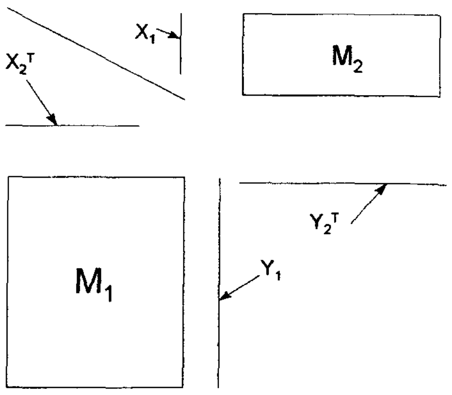
2. LU factorization with partial pivoting

2.1. Algorithm

The algorithm can be described as having basically three distinct parts within a loop. Let n be the dimension of the general square matrix, $A = LU$.

```
do j = 1, n
  Perform the j-th matrix vector product(forms part of L).
  Search for a pivot and exchange
  Perform the j-th vector matrix product(forms part of U)
end
```

For parallel execution, first let the matrix A look like as follows.[4], [5]



(Fig. 1) Parallel execution of LU

Then, LU factorization needs

```
Do j = 1, n

1) Search for a pivot and interchange
```

2) $y_1 = y_1 + M_1 X_1$: j−th matrix-vector product
3) $y_2 = y_2 + X_2 M_2$: j−th vector-matrix product

Let us denote the step 2) of above by C_j and 3) by R_j. For the sake of simplicity we assume that no pivoting is necessary. Then we could rearrange the above operations as follows.

```
$C_1$
  DO j = 1, N − 1
    $R_j$
    $C_{j+1}$
  End
  $R_n$
```

In other words we want to take advantage of the fact that R_j and C_{j+1} can be executed concurrently. Except for A(j, j + 1) in R_j, which is computed in R_j and subsequently needed in C_{j+1}, the remaining entries of y_1 and y_2^T are independent of each other for parallel processing. In CRAY-2 this is suitable for the multitasking. So for the A(j, j + 1), we need to synchronize between two tasks by event-related subroutines, such as Evwait, Evpost, Evclear, and Evasgn in the multitasking library.

2.2. Synchronization

1. We assign $C_2, C_3, \ldots, C_{N-1}$ to task 2 and the remaining to task 1.
2. Before calling LU subroutine, task 2 is created and waiting for the execution.
3. At the beginning of each iteration, R_j is started first, and as A(j, j + 1), it calls the C_{j+1} to start execution.
4. At the end of each execution, R_j is finished and wait for C_{j+1} to finish.
5. After N − 1 iterations, R_N is executed and LU decomposition is achieved.

2.3. Experiments

〈Table 1〉 shows the execution time in seconds using 1 and 2 processors. T1 and T2 denote the CPU time spent with 1 and 2 processors, respectively. For the N tested the speedups stay around 1.3, with the efficiency Ep = 65%. The lengths N seem to not affect the speedups much. The test runs were not made in exclusive mode and the CPU time was taken to be the minimum of several runs, so the exact speedup could be higher than reported here. Even so, the speedup cannot be said to be satisfactory. The main reason could be that as j increases in the LU decomposition loop the vector length becomes smaller relative to the vector size 64 of CRAY-2. Actually, the vector length in $M_1 x_1$ or $x_2^T M_2$ is n − j. This might be the factor limiting the speedup for this algorithm. This kind of low speedup is somewhat expected in LU decomposition routines, since they are inherently serial. In the iterative methods we might see an improvement.

〈Table 1〉 Timing of multitasked algorithm(T2) vs regular one(T1) in seconds.

Matrix size	T1	T2	T1/T2
100	0.0575	0.0416	1.38
300	0.988	0.779	1.23
400	2.21	1.70	1.3
500	4.14	3.27	1.27
600	6.95	5.34	1.3

3. Preconditioned iterative method

3.1 Introduction

The two most common ways to approximate the solution of partial differential equations by discretization of the original problems are FDM(Finite Difference Method) and FEM(Finite Element Method). They both lead to sparse banded matrices. For many applications, a matrix dimension N greater than 10,000 is not uncommon. For this kind of problems a direct method, such as, Gaussian elimination cannot be used because of its prohibitive cost. The use of iterative methods allows a solution at the reasonable cost. For the symmetric matrices, the Conjugate Gradient Method with proper preconditioning can be successfully used. For nonsymmetric matrices, the Conjugate Gradient Method does not apply. This difficulty can be overcome in several ways. We can solve Ax=f by solving the normal equations: $A^T A x = A^T f$. However, the matrix $A^T A$ generally squares the condition number of the matrix A, which could lead to slower convergence. Vinsome[17] proposed an algorithm, Orthomin[k], which requires minimum storage and achieves the convergence for matrices with the positive real part of the eigenvalue. The optimal k depends on the nonsymmetry of the problem. For symmetric problems k=1 yields the Conjugate Residual method.

The Conjugate Gradient Squared method (CGS) [14] was derived from the Biconjugate Gradients (BI-CG)[9] method by simply squaring the residual and direction matrix polynomials. CGS does not need multiplication by the transpose of a matrix. The residual and the directions in CGS are not bi-orthogonal or bi-conjugate respectively. However it can be viewed as the result of polynomial preconditioning with the polynomial varying from iteration to iteration. Thus it turns out that CGS is in practice faster than BI-CG. CGS computes exactly the same parameters as BI-CG and so it has exactly the same breakdown conditions as BI-CG. In fact along the iteration of CGS one can superimpose a BI-CG iteration with additional cost of one matrix vector multiplication but without the need for multiplication by the transpose.

For an iterative method, preconditioning the given linear system reduces the number of iterations substantially. We used Incomplete LU factorization(ILU (0))([2], [10], [12]) in this paper. However ILU-factorization has to solve $Ly=z$, $Ux=y$, which is a serial recurrence. For a vector machine, this causes serious slowdown. In this paper we adopted the von Neumann series approach by van der Vorst([15]). Also, the above-mentioned recurrences become a bottleneck in parallel execution. Radicati di Brozolo ([1]) proposed a technique to handle this problem, and achieved a high speedup on IBM 3090 with six processors. Our results on CRAY-2 with four processors on CRAY-2 also seem to confirm this.

3.2 A Model Problem

Let us consider the second order elliptic PDE problem in two dimensions in a rectangular domain Ω in R^2 with homogeneous boundary conditions:

$$-(au_x)_x-(bu_y)_y+(cu)_x+(du)_y+fu=g,$$

where u=0 on $\partial\Omega$ and a(x, y), b(x, y), c(x, y), d(x, y) and g(x, y) are sufficiently smooth functions defined on Ω and a, b > 0, c, d, f > 0 on Ω.

Discretization of the above equation by FDM or FEM leads to a linear system of equations, where the matrix is sparse. Let Ω be (0, 1) x (0, 1), with n grid points and h=1/(n+1) as the mesh size both in x and y directions, and the unknowns be ordered in the natural ordering. For the finite difference solution we used central difference for the first order terms and the five-point difference for the second order terms. Hence the whole discretization has a truncation error of $O(h^2)$, where h is mesh size. This leads to a Block-Tridiagonal matrix with five diagonals. The FDM discretization gives a linear system of equations:

$$Ax=f$$

of order $N=n^2$. If c(x, y) or d(x, y) is nonzero, then resulting matrix A is a nonsymmetric, block tridiagonal matrix. On the other hand, let L(u) be the partial differential operator. The finite element method seeks an approximate solution of the form

$$\mu=\sum_{k=1}^{N} \mu_k \phi_k,$$

are the basis functions. If we use Lagrangian basis functions with square elements, then the resulting matrix is also Block-Tridiagonal(See [11]). In this case the matrix has nine-diagonals, rather than five for FDM.

3.3 Iterative methods

We next describe two CG-like iterative methods to solve Ax = f, where A is a nonsymmetric matrix of order N. They are the Orthomin(k) which works for the symmetric part of A being positive definite, and the Conjugate Gradient Squared([12]), which converges for general matrices provided that the iteration does not break down. Here, we used right preconditioning, because in this form we can compute the same residual as in the non-preconditioned case.

3.3.1 Orthomin(k)

The Generalized Conjugate Residual method(GCR) [7] is a direct generalization of CR for symmetric and positive definite linear systems. In the absence of round-off error, GCR gives the exact solution in at most N iterations. The main difference between GCR for nonsymmetric matrix and CR for a symmetric matrix is that in i-th iteration of GCR, we have to keep in storage all of previous i-1 direction vectors, and compute p_i based on $A^T A$ orthogonality of p_i and p_j, $j<i$. Thus, as i gets larger, the cost and storage become prohibitive. The GCR method converges for A nonsymmetric with symmetric part positive definite. Vinsome([17]) proposed the Orthomin(k), as a practical version of the Generalized Conjugate Residual, with storage requirement for k directional vectors. Here, P_r is the right preconditioner. We used right preconditioning, since it minimizes the residual norm rather than minimizing the norm of $P_r\ r_i$, where r_i is the i-th residual vector.

```
Algorithm : Orthomin(k)
1. Choose $x_0$.
2. Compute $r_0 = f - Ax_0$.
3. $p_0 = r_0$.
For i = 0 step 1 Until Convergence Do
    4. $a_i = (r_i, Ap_i)/(Ap_i, Ap_i)$.
    5. $x_{i+1} = x_i + a_i\ p_i$
    6. $r_{i+1} = r_i - a_i\ A\ p_i$.
    7. Compute $A\ P_r\ r_{i+1}$
    8. $p_{i+1} = P_r\ r_{i+1} + \sum_{j=j_i}^{i} b_j^i p_j$. where
    9. $b_j^i = -\frac{(A\ P_r\ r_{i+1},\ A\ p_j)}{(A\ p_j,\ A\ p_j)}$ , $j \le i$.
    10. $A\ p_{i+1} = A\ P_{r+1} + \sum_{j=j_i}^{i} b_j^i\ A\ P_j$.
Endfor
```

In this algorithm $J_i = \max(0, i-k+1)$.

3.3.2 CGS

BI-CG was proposed to solve indefinite linear system, by Fletcher([9]). CGS results from the squaring of the BI-CG matrix polynomials for r_i. CGS breaks down whenever BI-CG does. In the absence of break down the CGS method converges in less than $N/2$ iterations. In the following P_r is the right preconditining operator.

```
Algorithm CGS
1. $r_0 = f - A\ x_0$, $P_r\ r_0$, $A\ r_0$.
2. $q_0 = p_{-1} = 0$.
3. $\rho_{-1} = 1.0$
For i = 0 step 1 Until Convergence Do
    4. $\rho_i = \tilde{r}_0^T\ r_i$, $b_i = \frac{\rho_i}{\rho_{i-1}}$
    5. $u_i = r_i + b_i\ q_i$
    6. $p_i = u_i + b_i(q_i + b_i\ p_{i-1})$
    7. $v_i = A\ P_r\ p_i$.
    8. $\sigma_i = \tilde{r}_0^T\ v_i$.
    9. $a_i = \frac{\rho_i}{\sigma_i}$
    10. $q_{i+1} = u_i - a_i\ v_i$
    11. $x_{i+1} = x_i + a_i\ P_r(u_i + q_{i+1})$
    12. $r_{i+1} = r_i - a_i\ A\ P_r(u_i + q_{i+1})$
Endfor
```

3.4 Preconditioning

For the preconditioning matrix P_r, we look for matrices such that

$$P_r A \approx I$$

or $P_r\ A$ has the clustered eigenvalues, and the linear system $P_r\ x = y$ is easy to solve. One natural choice is the Incomplete LU factorization([7], [10], [12]), where $A = L\ U + E$, where $L_{i,j} = U_{i,j} = 0$, if $A_{i,j} = 0$, and $E_{i,j} = 0$ if $A_{i,j} \neq 0$. In other words, L and U have the same sparsity patterns as A.

Let NZ(A) denote the set of pairs of [i, j] for which the entries $A_{i,j}$ of the matrix A are nonzero, the nonzero pattern of A.

Algorithm : The Incomplete LU Factorization.

```
For i = 1 step 1 Until N Do
For j = 1 step 1 Until N Do
If((i, j) belongs to NZ(A)) Then
    $s_{ij} = A_{ij} - \sum_{t=1}^{\min(i,j)-1} L_{it} U_{ij}$
    If(i ≥ j) Then $L_{ij} = s_{ij}$
    If(i < j) $U_{ij} = \frac{S_{ij}}{L_{ii}}$
  Endif
Endfor
Endfor
```

Here, we set $U_{ii} = 1$, for $1 \leq i \leq N$. Then, the finite difference matrix, is generated as follows.

```
For i = 1 step 1 Until N Do
$L_{i,i-1} = A_{i,i-1}$
  $L_{i,i} = A_{ii} - A_{i,i-1} L_{i,i-1} - U_{i-n,i} L_{i,i-n}$
$U_{i,i+1} = \frac{U_{i,i+1}}{L_{ii}}$
Endfor
```

For the finite element case, the ILU(0) preconditioner matrix is generated as follows.

```
For I = 1 step 1 Until N do
  $L_{i,i-n} = A_{i,i-n-1}$
  $L_{i,i-n} = A_{i,i-nx} - L_{i,i-n-1} L_{i-n-1,i-n-1}$
$L_{i,i-n+1} = L_{i,i-n+1} - L_{i,i-n} L_{i-n,i-n+1}$
  $L_{i,i-1} = A_{i,i-1} - L_{i,i-n-1} U_{i-n-1,i-1} - L_{i,i-n} U_{i-n,i-1}$
    $L_{ii} = A_{ii} - U_{i-1,i} L_{i,i-1} - U_{i-n-1,i} L_{i,i-n-1}$
        $- U_{i-n,i} L_{i,i-n} - U_{i-n+1,i} L_{i,i-n+1}$
$U_{i,i+1} = \frac{A_{i,i+1} - L_{i,i-n} U_{i-n,i+1} - U_{i-n+1,i+1} L_{i,i-nx+1}}{L_{ii}}$
$U_{i,i+n-1} = \frac{A_{i,i+n-1} - L_{i,i-1} U_{i-1,i+n-1}}{L_{ii}}$
$u_{i,i+n} = \frac{A_{[i,i+n]} - L_{i,i-1} U_{i-1,i+n}}{L_{ii}}$
$u_{i+n+1} = \frac{A_{i,i+n+1}}{L_{ii}}$
```

4. Experiments

4.1 Vectorization

Implementation on a vector computer requires proper vectorization of the computations to take advantage of the full single processor capacity. Vectorization for these methods is relatively straightforward. The matrix is stored in diagonals, so that Matrix-Vector multiplication is fully vectorized. Also the inner products and linear combinations are vector operations. The only computation which needs further attention is that of the preconditioning step,

$$L\ U\ x = y$$

which consists of solving the two triangular systems, $L\ z = y$ and $U\ x = y$, where L and U are the incomplete LU factors ,consisting of five (nine in FEM) diagonals. Solution of these systems requires back-solving, which is a serial operation. We can solve this problem by using a von Neumann series expansion, proposed by van der Vorst([15]), or by using the block preconditioning approach by G. Meurant([13]).

In this paper we adopted the first method. Assume the given matrix A is block-tridiagonal. Then, we can write A = L U +err. Assume further that the diagonal entries of L and U are 1, by scaling of the original problem. Then we can write L = I + E + F, where E is the matrix consisting of sub-diagonal $L_{i,\,i-1}$, F is the matrix consisting of the rest subdiagonals L_{ij}, $j < i-1$. Solving

$$(I+E+F)\,z_i = y_i$$

amounts to

$$(I+E)\,z_i = y_i - F\,z_{i-1}$$

Assuming E is small relative to I in norm, we expand in von Neumann series,

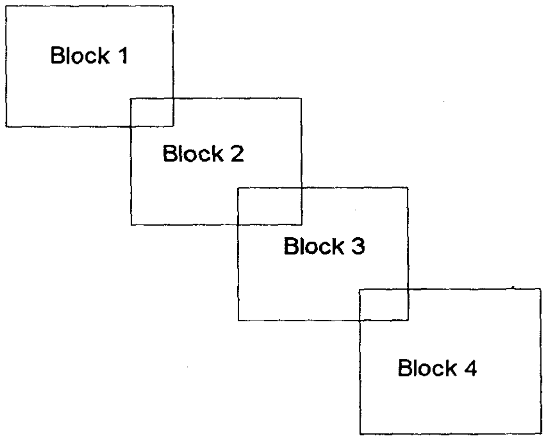
$$z_i = (I+E)^{-1}\,(y_i - F\,z_{i-1})$$

$$= (I - E + E^2 - E^3 + \dots)\,(y_i - F\,z_{i-1}).$$

and we truncate the power series at the m-th term. In [15] m = 2 was chosen. We also chose m = 2. The assumption about the norm of E relative to I is satisfied if the matrix is diagonally dominant. Most of the problems in elliptic PDE satisfy this assumption, or can be made so by increasing the number of meshes. The backward solution for U is computed similarly. This scheme vectorizes the computation, with vector length = n, where n is the number of grid points in the x-direction using the natural ordering.

4.2 Parallelization

In the codes of Orthomin, CGS, and CRS with/without preconditioning, the iteration loop cannot be executed in parallel, since the residual vectors and the directional vectors need the previous ones to be updated. Note that in these codes, most of the execution time is spent in the computational kernels, such as, the Matrix-Vector Product, the Dot product, the SAXPY, or the double SAXPY. So we could realize reasonable speedup from parallelization of these kernels. For the preconditioning block, the recurrence in (4.1) poses difficulties to the parallelization. We adopted the technique by Radicati di Brozolo[1]. It solves the preconditioned matrix system by dividing into four submatrices, ignoring the original relation between those submatrices. However we can make up for this loss of connection, by introducing an overlapping region between consecutive submatrices. Then we take the average of the computed values by the subsystem in the overlapped regions. According to [1], this overlapping strategy gives better performance than the non-overlapping one.



(Fig. 2) Parallel execution of ILU matrix-vector product with overlapping

4.3 Test Problem

Problem. [14] Convection-Diffusion equation

$$-\varepsilon(u_{xx}+u_{yy}) + v_y u_x + v_y\, u_y = 0, \Omega = (0,1)\,\mathrm{x}(0,1)$$

$$v_x = 2y(1-x^2),\ v_y = 2x(1-y^2),$$

$$u = \begin{cases} 0 & x = -1 \\ 0, & x = 1 \\ 0, & y = 1 \\ 1+\tanh(10(2x+1)), & y = 0,\ -1 \le x \le 9 \\ \dfrac{\partial u}{\partial n} = 0, & y = 0,\ 0 \le x \le 1 \end{cases}$$

4.4 Results

We ran experiments with Nx = 64, 128, 256, and Ny = Nx/2, where Nx is the number of nodes in X-direction, and Ny in Y-direction. We set $\varepsilon = 0.01$. For both FDM and for FEM we used the vectorizable von Neumann series ILU(0) Preconditioning. For Radicati's technique, we let the two consecutive submatrices overlap in region of two Nx nodes. For both FDM and FEM, we used natural ordering. We used the microtasking library for the parallelization of the inner do-loops. ⟨Table 2⟩ shows the speedups with 4 processors for various Nx and Ny, iterative methods and FDM and FEM. Orth(4) and CGS denote the iterative methods without the preconditioning. Orth(4)-ILU(0) and CGS-ILU(0) denote the Oth(4) and CGS methods with ILU(0) preconditioning with Radicati's technique. As in the LU case the test runs were not done in exclusive mode but taken to be the minimum of several runs. The real speedup will be higher than that reported here. As we see in the 3rd and 4th rows Radicati's technique seem to realize a reasonable speedup of 2.5-3.0 on the average, which makes the technique quite useful in the case of moderate number of processors. In the case of Orth(4) and CGS without preconditioning we see an appreciable increase in the speedup as we increase Nx and Ny both in FDM and FEM. This confirms that even though the Radicati di Brozolo preconditioning might be efficient the algorithm parallelizes better without the preconditioning part. With the preconditioning increase of Nx and Nx does not necessarily imply corresponding increase in the speedup. Also, any of the two iterative methods seem to not outperform the other method for all Nx and Ny. One noticeable experiment is the speedup of 3.66 for CGS-ILU(0) for Nx = 128, Ny = 64. It is even higher than that of CGS without the preconditioning. The reason could be that by using Radicati's technique we are actually employing some algorithm different from that without the preconditioning. It is reported by [1] that the technique sometimes produces unusually fast iteration. One final thing to note is that in every cases FDM produces higher speedup than the FEM for same Nx and Ny.

5. Conclusion

We have parallelized 2 linear algebra problems with multiple processors of CRAY-2. In the first case of LU decomposition we have achieved speedup of 1.3 with 2 processors on a matrix of dimension 200. For the second case of preconditioned iterative methods we were able to obtain a speedup of 2.5-3.2 with 4 processors. In the first case we used multitasking and microtasking in the second case. Microtasking is efficient since it parallelizes the inner DO loops and hence has minimum overhead and minimum load balancing problem. For a parallel execution of general tasks multitasking needs to be adopted, which has more overhead and load balancing problem. The parallel execution of LU decomposition algorithm for general dense matrices with 2 processors produces a

⟨Table 2⟩ Speedup with 4 processors with error tolerance = 0.01

	Finite Difference			Finite Element		
	64 x 32	128 x 64	256 x 128	64 x 32	128 x 64	256 x 128
Orth(4)	2.29	3.01	3.15	3.14	3.15	3.05
CGS	2.11	3.19	3.26	2.10	3.19	3.05
Orth(4)-ILU(0)	3.05	3.17	2.57	2.30	3.05	2.54
CGS-ILU(0)	2.47	3.66	2.85	3.09	2.41	2.71

speedup of poor 1.3 regardless of the matrix size. We believe that this is due to the ununiform vector lengths. The Radicati's technique seem to realize a reasonable speedup with 4 processors. Since it is very easy to implement it could be a useful technique combined with ILU(0) for moderate number of the processors. In our experiments the iterative methods parallelizes better than the LU decomposition methods. Also, it is to be noted that since the CRAY-2 has a shared memory, memory contention might cause the limitations on the maximum speedup possible.

References

[1] G. Radicati di Brozolo and Y. Robert, "Parallel and Vector Conjugate Gradient-like Algorithms for Sparse Nonsymmetric Systems", IBM ECSEC, Italy, 1988.

[2]. J. Cosgrove, J. Diaz, and A. Griewank, "Approximate inverse preconditionings for sparse linear systems", Intern. J. Computer Math., Vol. 44, 1992, pp. 91-110.

[3]. J. J. Dongarra and S. C. Eisenstat, "Squeezing the most out of an algorithm in CRAY Fortran", ACM Transactions on Mathematical Software, Vol. 10, No. 3, September 1984, pp. 219-230.

[4]. J. J. Dongarra and C. C. Hsiung, "Multiprocessing linear algebra algorithms on the CRAY X-MP-2: Experiences with small granularity", Journal of Parallel and Distributed Computing, Vol. 1, No. 1, Aug. 1984.

[5]. J. J. Dongarra and D. C. Sorensen, "Linear Algebra on High Performance Computers", Parallel Computing 85, 1986, pp. 3-32.

[6]. S. Eisenstat, H. Elman, and M. Schultz, "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations", SIAM J. Numer Anal, Vol. 20, 1983, pp. 345-357.

[7]. H. Elman, "Iterative Methods for Large,Sparse, Nonsymmetric Systems of \Linear Equations", Ph. D Thesis, Department of Computer Science, Yale, Univ., 1982,

[8]. S. Filippone and G. Radicati di Brozolo, "Vectorized ILU Preconditioners for General Sparsity Patterns", IBM ECSEC, Italy, 1988.

[9]. R. Fletcher, "Conjugate Gradient Methods for Indefinite Systems", Lecture Notes in Mathematics 506, Springer-Verlag, 1976.

[10]. L. Kolotilina and A. Yeremin, "Factorized sparse approximate inverse preconditionings", SIAM. J. Matrix And Appl., Vol. 14, No. 1, 1993, pp. 43-58.

[11]. L. Lapidus and G. Pinder, Numerical Solution of Partial Differential Equations in Engineering and science, John Wiley & Sons, 1981.

[12]. J. Meijerink and H. Van Der Vorst, "An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix", Math. Comp. 31, pp. 148-162, 1977.

[13]. G. Meurant, "The Block Preconditioned Conjugate Gradient Method on Vector Computers", BIT Vol. 24, 1984, pp. 623-633.

[14]. P. Sonneveld, "CGS, a Fast Lanczos-Type Solver for Nonsymmetric Systems", SIAM Sci. Stat Comp, Vol. 10, 1989, pp. 36-52.

[15]. H. Van Der Vorst, "A Vectorizable Variant of Some ICCG Methods", SIAM Sci. Stat Comp, Vol. 3, 1982, pp. 350-356.

[16]. H. Van Der Vorst, "High Performance Preconditioning", SIAM J. Sci. Stat Computing, 10(6), 1989, pp. 1174-1185.

[17]. P. Vinsome, "An Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations", Society of Petroleum Engineers of AIME,paper SPE 5729, 1976.

[18]. "An Overview of CRAY-2 Microtasking", MSCTB115, Minnesota Supercomputer Center, 1988 July.

마 상 백

1974년~1978년 서울대학교 자연대학 계산통계학과

1978년~1983년 국방과학연구소 연구원

1983년~1987년 미국 미네소타 주립대학 수학과 석사

1987년~1993년 8월 미국 미네소타 주립대학 전자계산학과 박사

1994년 3월~1997년 2월 한양대학교 공학대학 전자계산학과 전임강사

1997년 3월~현재 한양대학교 공과대학 전자계산학과 조교수