ハイパフォーマンス
コンピューティング 69－8
（1997. 12. 12）

不完全 LU 分解の並列化とその性能

津野直人† 野寺 隆†

偏微分方程式の境界値問題を離散化すると大規模で疎な係数行列をもつ連立1次方程式が生じる．これらの連立1次方程式を解く反復法に対して，不完全 LU 分解前処理は重要な役割を担っている．本稿では，不完全 LU 分解前処理の並列化のためのデータ分散と通信スキームを示す．最後に，64台のプロセッサをもつ分散メモリ型並列計算機 AP1000 上で，非対称問題に対する BiCGstab(ℓ) 法と GMRES 法を用いた性能評価の結果を報告する．

Parallelization and Performance of Incomplete LU Decomposition

NAOTO TSUNO† and TAKASHI NODERA†

Incomplete LU decompostion preconditioner plays an important role in iterative methods to solve linear systems of equations that are applied during the solution of discretized partial differential equations. In this paper, a data distribution and a communication scheme are presented for parallel sparse iterative solvers. At last, performance tests, using BiCGstab(ℓ) and GMRES method for nonsymmetric problems, were carried out on AP1000 with 64 processors.

1. はじめに

偏微分方程式の境界値問題を有限差分法や有限要素法を用いて離散化すると，大規模で疎な係数行列をもつ連立1次方程式

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

が生じる．本稿では，連立1次方程式 (1) の反復解法に対する前処理（preconditioning）の一つである不完全 LU 分解（ILU 分解とも呼ばれる）について考える．

本稿で扱う問題は，正方形領域 Ω 上の偏微分方程式

$$-u_{xx} - u_{yy} + g_1(x,y)u_x + g_2(x,y)u_y = f(x,y)$$

を5点中心差分法で離散化して生じるものとする．ここで離散化のメッシュサイズを MESH × MESH で表すことにする．離散化により位置 (i,j) にあるメッシュ上で作られる方程式は

$$A^S_{i,j}u_{i,j-1} + A^W_{i,j}u_{i-1,j} + A^C_{i,j}u_{i,j} + A^E_{i,j}u_{i+1,j} + A^N_{i,j}u_{i,j+1} = C \tag{2}$$

と書くことができる．ここで，$A^S_{i,j}$ 等や C は実数，$u_{i,j}$ は位置 (i,j) にあるメッシュ上で関数 u を代表する値である．この方程式 (2) を全てのメッシュ上で作り，連立させることで生じる $n \times n$ の係数行列 A を

$$A = \begin{Bmatrix} & A^N_{i,j} & \\ A^W_{i,j} & A^C_{i,j} & A^E_{i,j} \\ & A^S_{i,j} & \end{Bmatrix} \tag{3}$$

と表記することにする[1]．

以下，第2節で不完全 LU 分解の算法について述べる．次に第3節で不完全 LU 分解の並列化について述べる．第4節では，第3節で行った並列化の方法に関し，その並列処理の効率を計算する．第5節では，並列計算機上での数値例から不完全 LU 分解前処理の性能を評価し，第6節でその結論について述べる．

2. 不完全 LU 分解

行列 A の（完全な）LU 分解は，図1の算法により実行できる．ただし，この算法では行列 A, L, U の第 (i,j) 要素をそれぞれ $a_{i,j}, l_{i,j}, u_{i,j}$ で表してある．

不完全 LU 分解では，行列 A の疎構造が行列 L, U にそのまま引き継がれる．ここで，メッシュの順序づけに整合順序（natural ordering とも呼ばれる）を用いることにする．このとき，行列 L, U は (3) と同じ表記を用いて

$$L = \begin{Bmatrix} & 0 & \\ L^W_{i,j} & L^C_{i,j} & 0 \\ & L^S_{i,j} & \end{Bmatrix}$$

$$U = \begin{Bmatrix} & U^N_{i,j} & \\ 0 & U^C_{i,j} & U^E_{i,j} \\ & 0 & \end{Bmatrix}$$

† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University

```
for i = 1, 2, ..., n do
    for j = 1, 2, ..., i do
        l_{i,j} = a_{i,j} - Σ_{k=1}^{i-1} l_{i,k} u_{k,j}
    endfor
    u_{i,i} = 1
    for j = i+1, i+2, ..., n do
        u_{i,j} = (1/l_{i,i}) ( a_{i,j} - Σ_{k=1}^{i-1} l_{i,k} u_{k,j} )
    endfor
endfor
```

図 1　(完全) LU 分解の算法

```
for j = 1, 2, ..., MESH do
    for i = 1, 2, ..., MESH do
        L^S_{i,j} = A^S_{i,j}
        L^W_{i,j} = A^W_{i,j}
        L^C_{i,j} = A^C_{i,j} - L^S_{i,j} U^N_{i,j-1} - L^W_{i,j} U^E_{i-1,j}
        U^C_{i,j} = 1
        U^E_{i,j} = A^E_{i,j} / L^C_{i,j}
        U^N_{i,j} = A^N_{i,j} / L^C_{i,j}
    endfor
endfor
※ただし，U^N_{i,0} と U^E_{0,j} は 0 として扱う．
```

図 2　不完全 LU 分解の算法

と書くことができる．行列 A, L, U がもつこれらの疎構造を図 1 の算法に適用し整理することで，不完全 LU 分解の算法を導くことができる．結果として得られる，行列 A の不完全 LU 分解を行う算法を図 2 に示す．

3. 並　列　化

並列計算機上に不完全 LU 分解前処理を実装するには，3 つの過程を並列化する必要がある．(i) 係数行列 A を行列 L, U に不完全 LU 分解する過程．そして，(ii) 行列 L を用いる前進代入の過程と，(iii) 行列 U を用いる後退代入の過程である．

3.1　不完全 LU 分解の並列化

図 2 の算法を用いて行列 A を行列 L, U に不完全 LU 分解する際，あるメッシュが他のメッシュ上にあるデータを必要とするのは $L^C_{i,j}$ の計算を行うときのみである．詳しく書くと，位置 (i,j) のメッシュは下側の位置 $(i,j-1)$ のメッシュ上にあるデータ $U^N_{i,j-1}$ と，左側の位置 $(i-1,j)$ のメッシュ上にあるデータ $U^E_{i-1,j}$ を知る必要がある (図 3)．この，「下側と左側にあるメッシュへのデータ依存」を利用することで，不完全

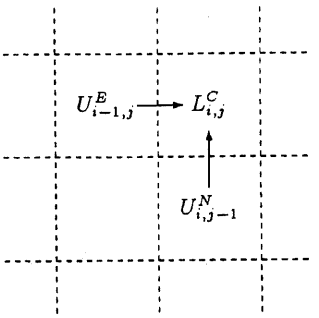


図 3　メッシュ間のデータ依存関係

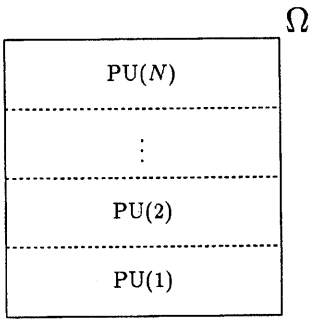


図 4　プロセッサによる領域分割

LU 分解を並列化することができる[1)]．

N 台のプロセッサ PU(1),...,PU(N) による並列処理を行う場合，図 4 のように領域 Ω を分割するとよい．このとき，まず第 1 ステップでは，PU(1) が位置 $(1,y), y = 1, 2, \ldots, \mathrm{MESH}/N$, のメッシュ上で計算を行う．次の第 2 ステップでは，PU(1) が位置 $(2,y), y = 1, 2, \ldots, \mathrm{MESH}/N$, のメッシュ上で計算を行い，同時に PU(2) が位置 $(1,y), y = (\mathrm{MESH}/N)+1, \ldots, 2\times\mathrm{MESH}/N$, のメッシュ上で計算を行う．同様の繰り返しにより，第 k ($\leq N$) ステップで PU(k) が計算を始めることができる．

3.2　前進代入，後退代入の並列化

行列 L を用いる前進代入 $w = L^{-1}v$ は，位置 (i,j) のメッシュ上において

$$w_{i,j} = (v_{i,j} - L^S_{i,j} w_{i,j-1} - L^W_{i,j} w_{i-1,j}) / L^C_{i,j}$$

を計算することで実行できる．このとき，位置 (i,j) のメッシュは，位置 $(i,j-1)$ のメッシュ上にあるデータ $w_{i,j-1}$ と，位置 $(i-1,j)$ のメッシュ上にあるデータ $w_{i-1,j}$ を知る必要がある．このメッシュ間のデータ依存関係は不完全 LU 分解の過程と全く同じである．したがって，前進代入の過程も不完全 LU 分解と同様の方法で並列化が可能である．

行列 U を用いる後退代入 $w = U^{-1}v$ も，位置 (i,j) のメッシュ上において

$$w_{i,j} = v_{i,j} - U^N_{i,j} w_{i,j+1} - U^E_{i,j} w_{i+1,j}$$

を計算することで実行できる．したがって，メッシュ間のデータ依存関係が「上側と右側にあるメッシュへのデータ依存」に変わることを除けば，不完全LU分解の過程，前進代入の過程と同様に並列化が可能である．ただし，この場合の計算はPU(N)から始まり，PU(1)が一番最後に計算を始めることになる．

4. 並列処理の効率

第3節で行った並列化の方法に関し，その並列処理の理論的な効率を考える．ただし，議論を簡単にするため，プロセッサ間通信から生じるオーバヘッドは無視する．

4.1 定　　義

ある処理Aを1台のプロセッサで処理したときの実行時間がt_1，N台のプロセッサで処理したときの実行時間がt_Nだとする．このときの並列処理効率Eを

$$E := \frac{t_1}{t_N}\frac{1}{N}$$

と定義する．

一般に，1台のプロセッサによる実行時間が等しいk個の処理があり，それらをN台のプロセッサで実行するときの効率をそれぞれ$e_1, e_2, \ldots, e_k$とすると全体の効率Eは

$$\frac{1}{E} = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{e_i} \tag{4}$$

によって定まる．

4.2 不完全LU分解，前進代入，後退代入の並列処理効率

第3節で行った並列化の方法では，メッシュ間のデータ依存関係に違いこそあるものの，不完全LU分解，前進代入，後退代入の3つの過程とも同じ方法で並列化を行った．したがって，どの過程も同じ並列処理効率をもつことになる．

まず，第1ステップでは1台のプロセッサによりMESH/N個のメッシュ上で計算が行われる．したがって，これらのメッシュに関する並列処理効率は$1/N$である．第2ステップでは2台のプロセッサにより$2\times\mathrm{MESH}/N$個のメッシュ上で計算が行われる．したがって，これらのメッシュに関する並列処理効率は$2/N$である．以下同様に，第i ($<N$)ステップで計算される$i\times\mathrm{MESH}/N$個のメッシュに関して，その並列処理効率はi/Nである．以上は計算開始直後のステップに関する並列処理効率で，同じことが計算終了直前のステップに関してもいえる．つまり，全体としては，並列処理効率i/Nのメッシュが$2\times i\times\mathrm{MESH}/N$個あることになる．また，残り$\mathrm{MESH}\times(\mathrm{MESH}-N+1)$個のメッシュに関する並列処理の効率は$N/N=1$である．図5にメッシュごとの並列処理効率の例を示す．この図は，色の濃いメッシュほど並列処理効率が低いことを表している．

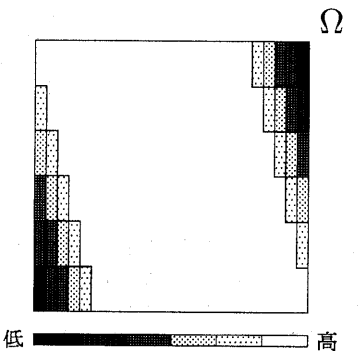


図5　メッシュの位置による並列処理効率の違い

以上より，第3節で行った並列化の並列処理効率Eは，式(4)から

$$\begin{aligned}\frac{1}{E} &= \frac{1}{(\mathrm{MESH})^2}\left\{\sum_{i=1}^{N-1}\frac{2\times i\times\mathrm{MESH}}{N}\times\frac{1}{i/N}\right.\\ &\qquad\left.+\mathrm{MESH}\times(\mathrm{MESH}-N+1)\times\frac{1}{N/N}\right\}\\ &= \frac{\mathrm{MESH}+N-1}{\mathrm{MESH}}\end{aligned}$$

故に

$$E = \frac{\mathrm{MESH}}{\mathrm{MESH}+N-1} \tag{5}$$

となる．

式(5)を利用して，$\mathrm{MESH}=64,128,256$の各場合に対してプロセッサの台数Nを$2^i, i=1,2,\ldots,6$としたときの並列処理効率を表1に示す．

5. 数　値　例

不完全LU分解前処理を，連立1次方程式(1)の反復解法に組み込み，実験を行う．反復解法はGMRES(m)法[2]とBiCGstab(ℓ)法[3]を採用し，前処理は右側から行う．実験結果は収束条件を満たすまでに要した実行時間で示す．反復回数は紙面の都合上省略する．

各実験に共通する条件は，特に断りのない限り次のとおりである．

- 収束条件：$\|r_i\|/\|r_0\| < 1.0\times10^{-12}$
- 初期近似解：$x_0 = (0,0,\ldots,0)^T$
- 計算機：分散メモリ型並列計算機AP1000
- 計算精度：倍精度
- 時間計測：1つの問題に対し3回の実験を行い，その実行時間の平均値

例1．正方形領域$\Omega=[0,1]\times[0,1]$における楕円型偏微分方程式の境界値問題

$$\begin{aligned}-u_{xx}-u_{yy}+\alpha u_x &= f(x,y)\\ u(x,y)|_{\partial\Omega} &= 1+xy\end{aligned}$$

表1　不完全LU分解の並列処理効率（%）

メッシュサイズ	プロセッサ台数					
	2	4	8	16	32	64
64×64	98.46	95.52	90.14	81.01	67.37	50.39
128×128	99.22	97.71	94.81	89.51	80.50	67.02
256×256	99.61	98.84	97.34	94.46	89.20	80.25

を考える[4]．ただし，右辺の関数 $f(x,y)$ は厳密解が $u(x,y) = 1 + xy$ となるように定めるものとする．この問題を5点中心差分法を使ってメッシュサイズ 256×256 で離散化し，連立1次方程式 (1) を作る．結果として生じる係数行列 A は，適切なスケーリングを施すことで

$$A = \begin{Bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{Bmatrix} + \alpha h \times \begin{Bmatrix} & 0 & \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ & 0 & \end{Bmatrix}$$

となる．ただし，$h = 1/257$ である．

上の問題を64台のプロセッサを用いて解いた．その実験結果を表2に示す．不完全LU分解の計算等から生じる時間的なオーバヘッドを十分に埋め合わせる反復回数の減少があり，実行時間は大幅に短縮された．

次に，使用するプロセッサ台数を変えて，1反復にかかる平均実行時間を調べた．その実験結果を図6に示す．不完全LU分解を組み込んだ場合でも，ほぼ理想的な台数効果が得られた．

例2．正方形領域 $\Omega = [0,1] \times [0,1]$ における楕円型偏微分方程式の境界値問題

$$-u_{xx} - u_{yy} + \alpha\{g_1(x,y)u_x + g_2(x,y)u_y\} = f(x,y)$$

$$g_1(x,y) = y - \frac{1}{2}$$

$$g_2(x,y) = \left(x - \frac{1}{3}\right)\left(x - \frac{2}{3}\right)$$

$$u(x,y)|_{\partial\Omega} = 1 + xy$$

を考える[4]．ただし，右辺の関数 $f(x,y)$ は厳密解が $u(x,y) = 1 + xy$ となるように定めるものとする．この問題を5点中心差分法を使ってメッシュサイズ 128×128 で離散化し，連立1次方程式 (1) を作る．結果として生じる係数行列 A は，適切なスケーリングを施すことで

$$A = \begin{Bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{Bmatrix} + \alpha h \times \begin{Bmatrix} & \frac{1}{2}g_2 & \\ -\frac{1}{2}g_1 & 0 & \frac{1}{2}g_1 \\ & -\frac{1}{2}g_2 & \end{Bmatrix}$$

となる．ただし，h はメッシュ幅を表し，$h = 1/129$ である．

上の問題を64台のプロセッサを用いて解いた．その実験結果を表3に示す．この問題に対し，GMRES(m) 法は多くの場合，最大反復回数3000回内で収束条件を満たさなかった．しかし，不完全LU分解を前処理として用いることで収束条件を満たすようになった．一方，BiCGstab(ℓ) 法に不完全LU分解前処理を行った場合，問題のパラメータ αh の値が小さいときに反復回数の減少はあったものの，実行時間は増加してしまった．しかし，前処理なしの BiCGstab(ℓ) 法が収束条件を満たすまでに多くの反復を必要としている場合，不完全LU分解を前処理として用いることで大幅な実行時間の短縮が観測された．

次に，使用するプロセッサ台数を変えて1反復にかかる平均実行時間を調べた．その実験結果を図7に示す．例2のメッシュサイズは例1よりも小さい．したがって，不完全LU分解を組み込んだ場合，プロセッサが64台のときの速度向上がやや鈍っている．しかし，全体的には例1と同様にほぼ理想的な台数効果が得られた．

6. ま と め

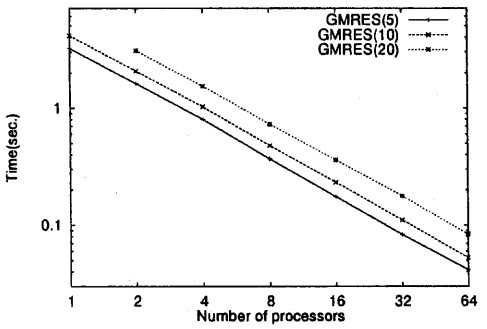
反復回数に関する不完全LU分解前処理の性能には，よく知られている通りの高いものがある．しかし，不完全LU分解前処理は前進代入や後退代入の演算を必要とし，本質的に逐次処理である．

本稿では，有限差分法による離散化から得られるメッシュ間のデータ依存関係を利用して，不完全LU分解前処理の並列化を行った．分散メモリ型並列計算機 AP1000 による数値実験の結果からは，本稿で行った並列化の方法が高いスケーラビリティをもち，不完全LU分解前処理が並列計算機上でも前処理としての高い性能を維持することが確認された．

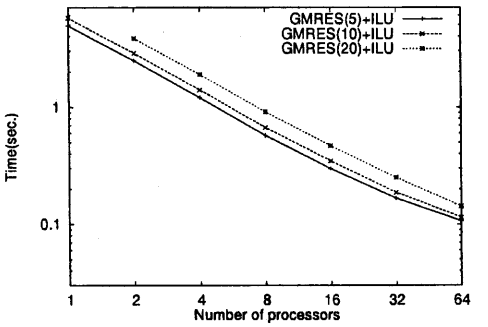
表 2　例 1 に対する収束状況（秒）

算法	αh の値									
	0	2^{-3}	2^{-2}	2^{-1}	2^{0}	2^{1}	2^{2}	2^{3}	2^{4}	2^{5}
GMRES(5)	*	*	63.25	31.74	33.51	32.15	32.98	32.82	36.68	43.91
GMRES(5)+ILU	*	48.21	25.74	27.29	22.91	18.10	12.24	9.23	7.07	6.13
GMRES(10)	*	117.04	50.52	47.75	50.41	50.89	50.75	47.82	44.42	43.54
GMRES(10)+ILU	296.18	36.25	39.58	40.95	37.02	25.45	16.08	11.94	8.31	6.63
GMRES(20)	*	109.68	89.45	88.79	94.16	92.41	92.00	90.39	84.65	79.01
GMRES(20)+ILU	197.47	60.54	76.01	70.29	54.82	36.51	29.32	15.55	9.83	5.89
BiCGstab(1)	30.11	23.83	20.34	20.66	23.18	23.04	48.42	101.04	*	*
BiCGstab(1)+ILU	30.34	21.28	17.82	17.25	15.09	10.91	8.46	6.73	4.86	3.13
BiCGstab(2)	35.16	26.78	24.14	24.67	26.38	29.65	30.07	25.64	24.69	24.90
BiCGstab(2)+ILU	31.86	22.87	19.67	21.27	16.47	10.37	7.47	6.51	5.23	3.63
BiCGstab(4)	47.28	33.40	30.93	32.86	39.23	39.51	36.47	36.75	32.33	30.96
BiCGstab(4)+ILU	34.38	24.56	22.47	23.17	18.99	12.00	8.49	7.79	5.69	4.30

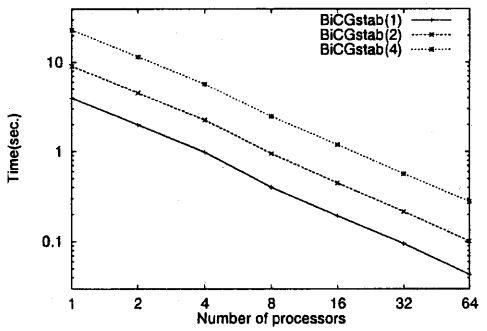
* 最大反復回数 3000 回内で収束せず



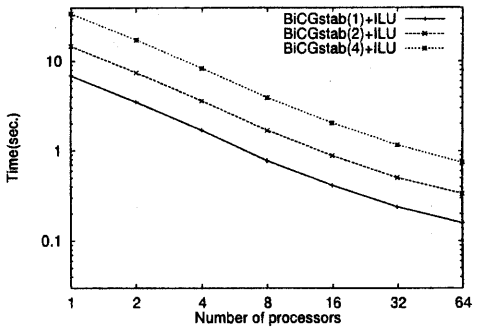
(a) GMRES(m) 法



(b) ILU 前処理付き GMRES(m) 法



(c) BiCGstab(ℓ) 法



(d) ILU 前処理付き BiCGstab(ℓ) 法

図 6　例 1（256 × 256 メッシュ）に対する 1 反復当たりの平均実行時間

参 考 文 献

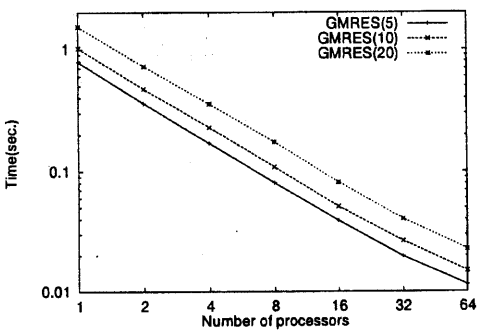
1) Bastian, P. and Horton, G.: *Parallelization of Robust Multigrid Methods: ILU Factorization and Frequency Decomposition Method*, SIAM J. Sci. Stat. Comput., Vol. 12, No. 6, pp. 1457–1470 (1991).

2) Saad, Y. and Schultz, M.H.: *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, SIAM J. Sci. Stat. Comput., Vol. 7, No. 3, pp. 856–869 (1986).

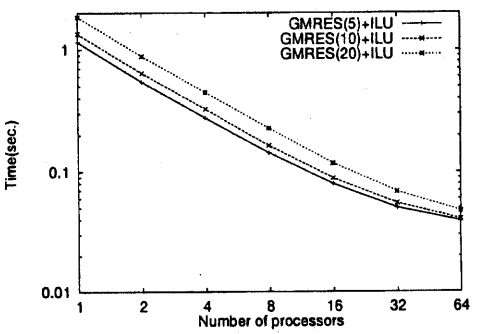
表 3　例 2 に対する収束状況（秒）

算法	αh の値									
	0	2^{-3}	2^{-2}	2^{-1}	2^{0}	2^{1}	2^{2}	2^{3}	2^{4}	2^{5}
GMRES(5)	*	*	*	*	*	*	*	*	*	*
GMRES(5)+ILU	57.68	47.70	44.18	29.12	25.92	29.55	34.19	35.17	34.71	*
GMRES(10)	*	*	*	*	*	*	*	*	*	*
GMRES(10)+ILU	32.34	20.26	21.24	23.36	21.43	25.44	30.63	32.74	33.28	40.21
GMRES(20)	*	50.89	66.23	50.32	54.08	71.57	*	*	*	*
GMRES(20)+ILU	22.05	19.69	19.09	20.46	23.69	29.07	34.09	35.94	28.56	31.53
BiCGstab(1)	4.72	5.13	6.10	6.23	6.60	8.29	9.64	14.90	*	*
BiCGstab(1)+ILU	6.39	7.43	8.23	8.29	8.63	8.95	8.17	7.25	6.28	5.85
BiCGstab(2)	4.91	6.24	6.60	6.83	7.27	8.66	10.56	12.51	19.07	29.05
BiCGstab(2)+ILU	6.17	7.02	7.74	8.34	8.35	9.19	8.58	7.26	6.30	5.58
BiCGstab(4)	6.24	7.05	7.76	8.40	9.28	10.85	13.31	15.87	23.27	32.66
BiCGstab(4)+ILU	6.30	7.43	8.45	8.70	8.96	10.24	9.21	7.68	6.92	6.15

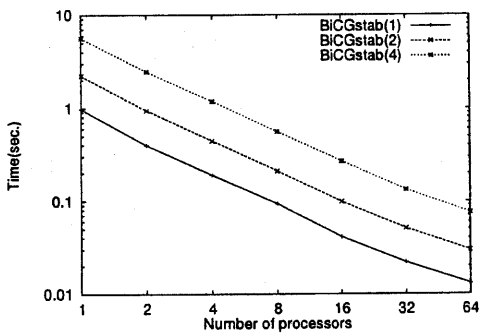
* 最大反復回数 3000 回内で収束せず



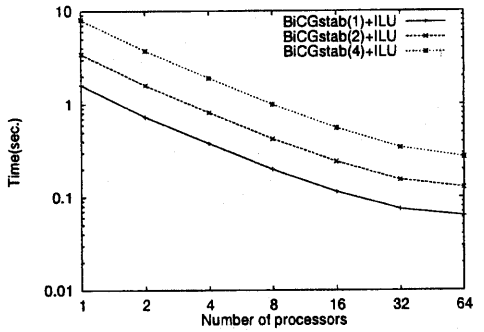
(a) GMRES(m) 法



(b) ILU 前処理付き GMRES(m) 法



(c) BiCGstab(ℓ) 法



(d) ILU 前処理付き BiCGstab(ℓ) 法

図 7　例 2（128 × 128 メッシュ）に対する 1 反復当たりの平均実行時間

3) Sleijpen, G. L. G. and Fokkema, D. R.: *BiCGstab(ℓ) for Linear Equations Involving Unsymmetric Matrices with Complex Spectrum*, ETNA, Vol. 1, pp. 11–32 (1993).

4) Joubert, W.: *Lanczos Methods for the Solution of Nonsymmetric Systems of Linear Equations*, SIAM J. Matrix Anal. Appl., Vol. 13, No. 3, pp. 926–943 (1992).

5) 野寺隆, 野口雄一郎: AP1000 における BiCGStab(ℓ) 法の有効性について, 情報処理学会論文誌, Vol. 38, No. 11, pp. 1–13 (1997).