非対称疎行列の並列LU分解におけるリオーダリング効果

山本富士男[1]・長谷川　聡[2]・荒木智行[3]

[1]情報工学科
[2]情報工学科卒業
[3]情報工学科

Reordering Effects on Parallel LU Decomposition of Unsymmetric Sparse Matrices

Fujio YAMAMOTO[1)], Satoshi HASEGAWA[2)], Tomoyuki ARAKI[3)]

Abstract

Experimental study on the effects of sparse matrices reordering on parallel LU decomposition is presented. Sparse matrices treated here are large unsymmetric unstructured ones such as circuit admittance matrices. Two reordering algorithms, Tewarson's and Stewart's are investigated using forty randomly generated matrices. In most cases, they proved to reduce the number of parallel LU decomposition steps as well as the total number of operations.

Key Words: Sparse Matrix, Reordering, Parallel Computing, LU Decomposition

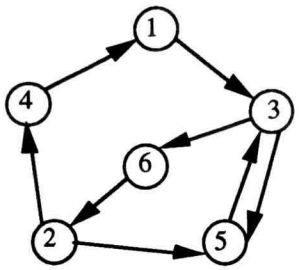
1. まえがき

電子回路解析、電力網解析、線形計画法、化学工学などでは、多数の要素間の接続関係をもとにした大規模連立一次方程式を解く必要がある。特に非線形性や時間依存性を持つ要素を含む場合は、それらを数百～数千回以上繰り返す。これらの連立一次方程式の係数行列は、零要素の割合が非常に高い疎行列となる。さらに、多くの場合、その非零要素は不規則に散在し、それらの位置関係に対称性がない。

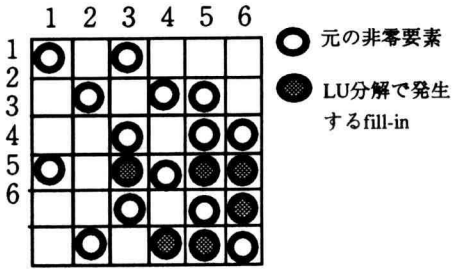
このような特性の連立一次方程式を、ベクトル型スーパーコンピュータやマルチプロセッサを使って高速に解くための研究がなされて来た。これらの中には、反復系の解法を適用し[1)]、ある程度の成功を納めたものもある。しかしながら、一般には、LU分解をベースとした直接解法によっているもの[2),3)]が多い。それは、係数行列の不規則性に起因すると思われる。すなわち、もし、反復系の解法を適用するとしたら、収束速度を高めるための前処理がどうしても必要であるが、その方法として適当なものがあまり無いということである。LU分解をベースにした解法では、その演算どおしの並列性を利用することが一つのキーポイントである。そのような研究[4)~7)]は、超並列コンピュータに代表されるハードウエアの急速な進展に伴って、より一層活発になりつつある。

ところで、疎行列のLU分解では、未知数、あるいは行や列の番号の付け方がその演算量に大きく影響することが知られている。すなわち、特に注意せずに付番した行列のままLU分解を行うと、元々零であった要素が、演算中に非零になり、それがさらに非零要素を次々と発生させる、いわゆるfill-inが多発する。そこで、LU分解を行う前に、適当な方法で行列番号を再順序付けすることが必要とされる。再順序付けの手法[8)~11)]は、もともとこのようなfill-inの発生をできるだけ抑止するために考案されたものである。

一方、不規則疎行列のLU分解での並列性を検出するアルゴリズム[4)]は、このような再順序付けを特に意識していないので、その影響はあまり知られていない。また、近年市販されている著名な数学ソフトウェア[12),13)]などにおいても、これに関する言及は見当たらない。そこで、本論文では、一般の非対称不規則疎行列の並列LU分解アルゴリズムにおいて、再順序付けを行った場合、どのような効果があるかを明らかにする。具体的には、乱数により発生させた多数の非対称不規則疎行列に対して、(1)再順序付けなし、(2)Tewarson法による再順序付け、(3)Stewart法による再順序付け、の各々について並列LU分解をシミュレーションで実施し、その終了時間を実験的に評価する。



(a) 有向グラフ



(b) fill-inを含めた疎行列

step1	$a_{41} /= a_{11}$ $a_{62} /= a_{22}$ $a_{53} /= a_{33}$
step2	$a_{43} -= a_{41} * a_{13}$ $a_{64} -= a_{62} * a_{24}$ $a_{65} -= a_{62} * a_{25}$ $a_{55} -= a_{53} * a_{35}$ $a_{56} -= a_{53} * a_{36}$
step3	$a_{43} /= a_{33}$ $a_{64} /= a_{44}$
step4	$a_{45} -= a_{43} * a_{35}$ $a_{46} -= a_{43} * a_{36}$
step5	$a_{65} -= a_{64} * a_{45}$ $a_{66} -= a_{64} * a_{46}$
step6	$a_{65} /= a_{55}$
step7	$a_{66} -= a_{65} * a_{56}$

(c) LU分解における並列性

図1 非対称不規則疎行列とそのLU分解における並列性検出

2. 不規則疎行列と演算の並列性

大規模な密行列のLU分解には、演算の高い並列性がある。一方、sparsity（非零要素の割合）が高い疎行列ではその並列度は極端に低下するものと考えられていた。しかし、並列性検出アルゴリズムMVA[4]によれば、多くの場合高い並列性を利用できることが明らかにされた。まず、そのアルゴリズムを簡単な例で説明する。

図1の(a)は、6個のノード間の、方向を持った接続関係を示している。(b)は、それを行列に表現したものである。この行列を係数とする連立一次方程式を、LU分解で解くとする。その手順は以下の通りである。ただし、ここで行列(これをaとする)は対角優位であり、ピボット選択は必要ないとする。

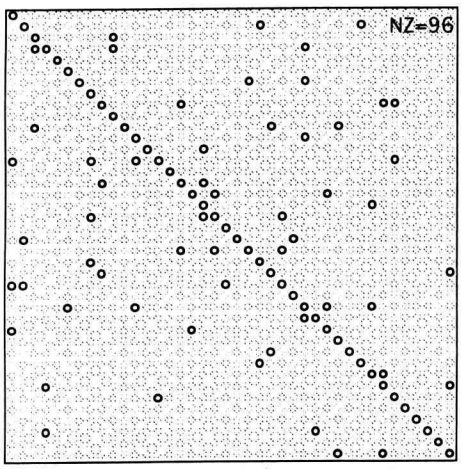
```
for k=1, 5
  for i= k+1, 6
    a_ik = a_ik / a_kk
    for j= k+1, 6
      a_ij =a_ij - a_ik * a_kj
    endfor
  endfor
endfor
```

このプログラムにおいて実際には、(b)の図の中の非零要素についてのみ演算を実施する。ここでは、for iとfor jのループについて2重の並列性がある。しかし、非零要素についてのみ演算を実行するので、実際に並列に実行できる演算は非常に少ないものとなってしまう。

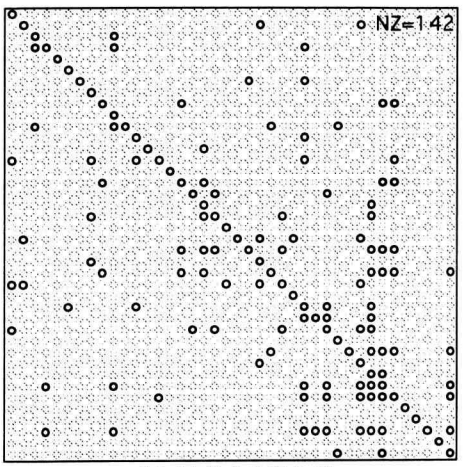
そこで、上記のMVAアルゴリズムを使うことになる。MVAは、除算$a_{ik} = a_{ik} / a_{kk}$と更新演算$a_{ij} = a_{ij} - a_{ik} * a_{kj}$について、代入文の左辺要素をいつ更新してよいかを、「データ参照レベル」という考えから求めるものである。このMVAにより、(b)の行列をLU分解すると(c)のようになり、疎行列の中により高い並列性を見いだすことが可能になる。上記の1回の除算、1回の更新演算をそれぞれ1演算として計算すると、合計16演算が必要である。仮に、5台のCPUからなる並列計算機を使い、MVAを用いてLU分解するならば、論理的には、(c)に示すとおり、これらを7ステップで終了することができる。各ステップ（step1, step2, . . .）毎に並列実行可能な演算を示してある。

3. リオーダリング（再順序付け）

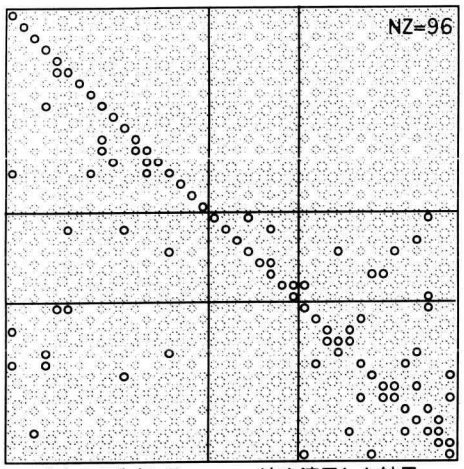
図1(b)の行列の非零要素とfill-inの位置は、図1(a)の有向グラフでのノードの番号付けに依存する。したがって、それらの番号を別のやりかたで付与するならば、非零要素の出現位置が変わり、さらにその影響でfill-inの様子も異なるものになるであろう。ここでは、最初に、このような番号のリオーダリング･アルゴリズムを簡単に説明する。



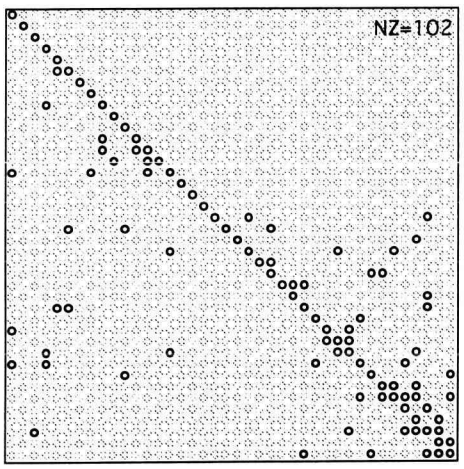
(a) 乱数により生成された疎行列



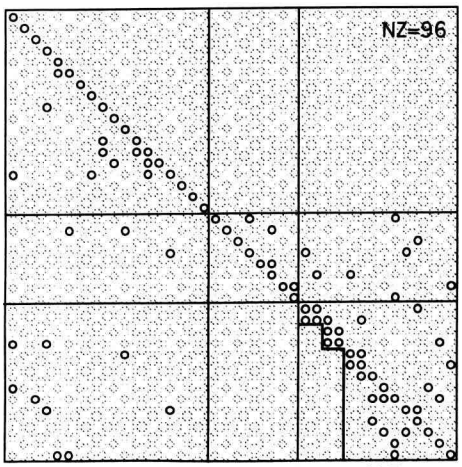
(b) 左図(a) のfill-in 後



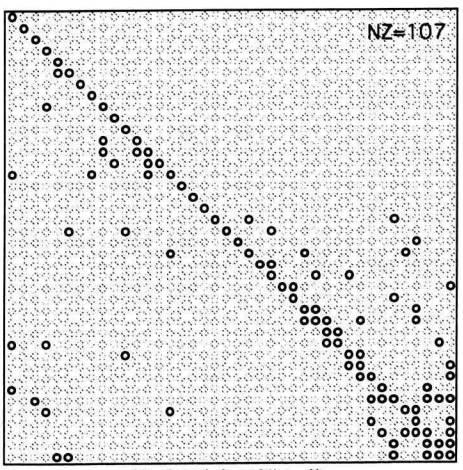
(c) 上図(a)にTewarson法を適用した結果



(d) 左図(c) のfill-in 後



(e) 上図(a)にStewart法を適用した結果



(f) 左図(e) のfill-in 後

図2　疎行列非零要素の分布例（行列サイズ＝40 x 40、丸印は比例要素位置、NZは非零要素数）

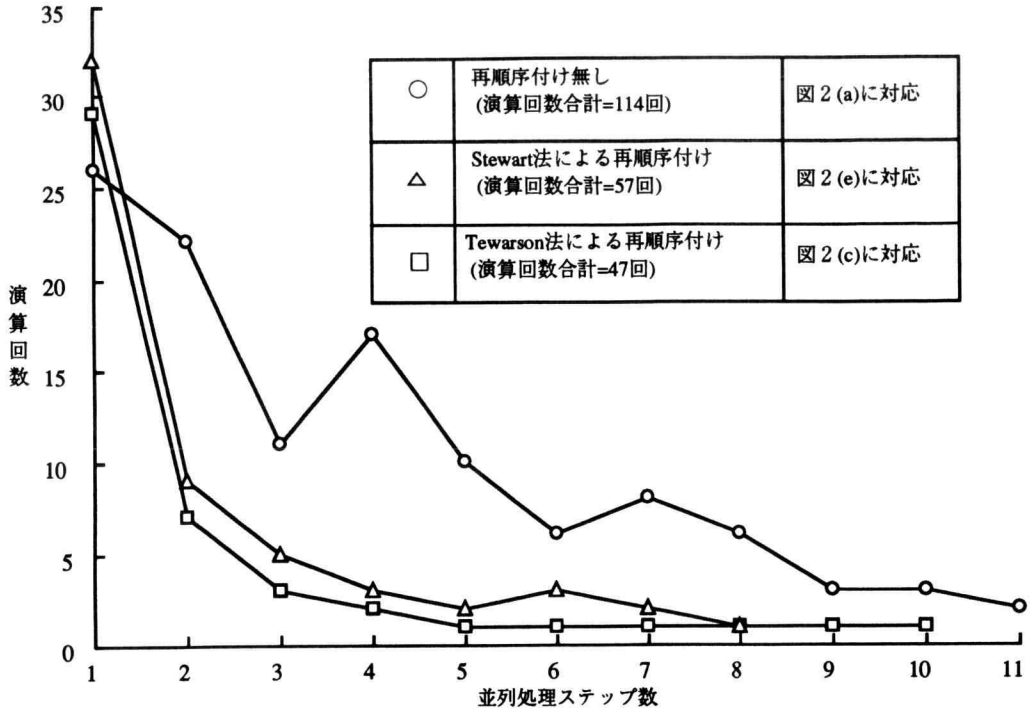


図3 LU分解における並列度解析結果(図2に対するもの)

3.1 リオーダリング・アルゴリズム

(1) Tewarsonの方法[8),9)]

一般の疎行列向きの再順序付けアルゴリズムである。前処理としてまず、非零要素数が1個の行（これを行シングルトンと呼ぶ）を行列上部に集めて上三角行列部を作る。次に、非零要素数が1個の列（これを列シングルトンと呼ぶ）を行列左方向に集めて下三角行列部を作る。なお、行／列シングルトンは、すでに見つかったものを除いた小行列について次々と探してゆく。

残りの右下行列部分を核と呼ぶ。この核の下三角部分の非零要素をできるだけ少なくするように行と列を入れ替える。具体的には、行sの非零要素数と列tの非零要素数の積が最小となるような行sと列tが軸位置に来るようにする。この時、LU分解を記号的に実施（シミュレート）して、fill-inを含めた非零要素数でこの積を評価する。

(2) Stewartの方法[9)]

この方法も一般の疎行列に適用できる。Tewarsonの方法と同様に、最初に行シングルトンと列シングルトンを見つける操作を行って、核部分を分離する。次に核の部分をブロック型の下三角行列になるよう、行と列を入れ替える。すなわち、核の下三角部分を、対角に近い要素だけを含む小ブロックに分割し、LU分解での消去演算量を削減しようとするものである。

3.2 リオーダリングの適用例

上記2つのリオーダリング・アルゴリズムが実際にどのように働くかを、図2の例を使って説明する。この例は40x40の疎行列を乱数により発生させたものである。その非零パタンが(a)である。

(1) 非零要素数

図2において、(b)は(a)の行列をLU分解した後の非零要素のパタンを示している。すなわち、fill-in後の状況である。(c)は(a)に対してTewarson法を適用してリオーダリングした結果である。上から18行の行シングルトンと、つづいて8列の列シングルトンにより、核部が分離されている。fill-in後の非零要素数は、(b)に比べ著しく減少していることが分かる。

また、(e)は(a)に対し、Stewart法を適用してリオーダリングした結果である。Tewarson法の場合と同じく、行シングルトンと列シングルトンが得られ、核部が分離された。さらに核部は図に示すように、3個のブロックで構成される下三角ブロック行列となった。これにLU分解を施した結果が(f)である。(d)の場合より若干非零要素数が増えているが、元の行列のLU分解結果(b)と比較すると大幅に非零要素数は減少している。

以上のことから、この例題では、Tewarson法とStewart法は共に非零要素数を削減させ、演算量の低減におおき

な効果があると言える。

(2) LU分解の並列性

次に、上記リオーダリングを施したことによって、LU分解での並列性がどのように変わるかを調べた結果が図3である。この図の横軸は、LU分解を2. で述べたMVA法を使って実施した場合に必要なステップ数を表す。縦軸は、その各ステップで並列実行可能な演算回数を示している。ここでの演算回数とは、2. で述べたとおりである。すなわち、除算、および更新演算の回数のことである。

この図から、仮に32台のCPUからなる並列計算機を使うとしたら、論理的には、リオーダリング無しの場合、11ステップでLU分解が終了する。一方、Tewarson法でのリオーダリングを行えば、演算回数の合計が大幅に減少するのに加えて、並列処理ステップ数も10ステップで済む。また、Stewart法の場合には、演算回数合計はTewason法の場合より若干増えるが、並列処理ステップ数は逆に8ステップに減少することが分かる。

4. 演算の並列性とリオーダリングの関係

上記3. の例題で考察した、リオーダリングとLU分解の並列性との関係をもっと一般的に調べるため、以下のような実験を行った。

(1)実験の方法

実験コストの制約の元で、できるだけ現実の問題を反映し得るよう考え、行列のサイズは1000x1000に固定した。非零要素は乱数により配置した。そして、比零要素数が2300個生成される20ケースと、2500個生成される20ケースの合計40ケースの疎行列を対象とした。この非零要素数は、一つの要素に平均2.3～2.5個の要素が何らかの関係を持つことを意味しており、現実問題としても妥当な(つまり、かなりあり得る)設定と考えた。用いた乱数生成では、非零要素の位置は不規則になり、対称性も生じない。また、ここでは、常に対角要素が存在し、しかも対角優位性があると仮定した。したがって、LU分解中にピボット選択は必要としない。

リオーダリングとLU分解の並列性との関係を示すデータを採取するため、以下の手順を実施した。

a)LU分解での並列性を検出するMVA法によるプログラムと、各種データ採取プログラムを作成した。C言語で約560行となった。

b)二つのリオーダリングアルゴリズム、すなわち、Tewarson法とStewart法のプログラムは、文献[9]のものを使った。ただし、原板はFortranで書かれていたので、Unixワークステーションでのハンドリングの都合上、これをF2CプログラムによりC言語に変換し、若干の修正を加えて使った。

c)上記に述べた最初の20ケースと、次の20ケースのそれぞれについて以下のようにデータを採取した。

- LU分解をシミュレートしてfill-in個数と総演算回数を計測する。
- MVA法によりLU分解の並列度を解析し、プロセッサ台数無制限の並列計算機を用いた場合、何ステップでLU分解が完了するかを評価する。

これを、以下の3通りの状態についてそれぞれ実行した。

- 非零要素配置の初期状態。
- 非零要素配置の初期状態に対し、Tewarson法でリオーダリングした後の状態。
- 非零要素配置の初期状態に対し、Stewart法でリオーダリングした後の状態。

(2)実験の結果

上記の手順で実験した結果をまとめたものが図4である。左側の(a)(c)(e)は非零要素数NZ=2500の場合であり、右側の(b)(d)(f)は非零要素数NZ=2300の場合である。(a)(b)は、リオーダリングを行わない場合に対する、Tewarson法とStewart法のfill-in発生抑止の効果を示している。両方法とも大きな効果があることが確認された。(c)(d)は、LU分解での演算回数の相対的比較である。これは、(a)(b)で示したfill-in発生状況を反映したものであり、やはり両リオーダリング方法の効果が著しい。演算回数に関しては、Tewarson法が、リオーダリングを行わない場合の1～4割程度となっており、全般的にStewart法よりやや有利である。

並列性に関しては、(e)(f)にその結果を示した。(e)は比零要素数NZ=2500の場合であるが、並列処理ステップ数は、ほとんどのケースで、リオーダリングを施した場合の方がより少なくなっている。具体的には、20ケースのうち19ケースにおいて、いずれかのリオーダリング法により並列処理ステップ数の削減がなされた。その削減率は最大で50%程度である。一方、NZ=2300の場合は、リオーダリングを施さない場合より並列処理ステップ数が若干増加する場合がある。具体的には、20ケースのうち11ケースでは、いずれかのリオーダリング法により並列処理ステップ数が削減し、逆に9ケースでは、いずれのリオーダリング法でも増加した。しかし、その増加の度合いは1ケースを除けば1.1～1.7倍程度に過ぎない。

(3)実験結果についての考察

図1(e)の結果は、リオーダリングアルゴリズムを適用すると、fill-inと演算量が大幅に削減できると同時に、並列処理ステップ数もほとんどの場合低減することを示している。この(e)の場合、すなわち、非零要素数NZ=2500では、Tewarson法がStwart法よりも効果がある。しかしながら、これより若干sparsity(スパース率)が高いケース、すなわち、(f)に示したNZ=2300の場合は、Tewarson法と

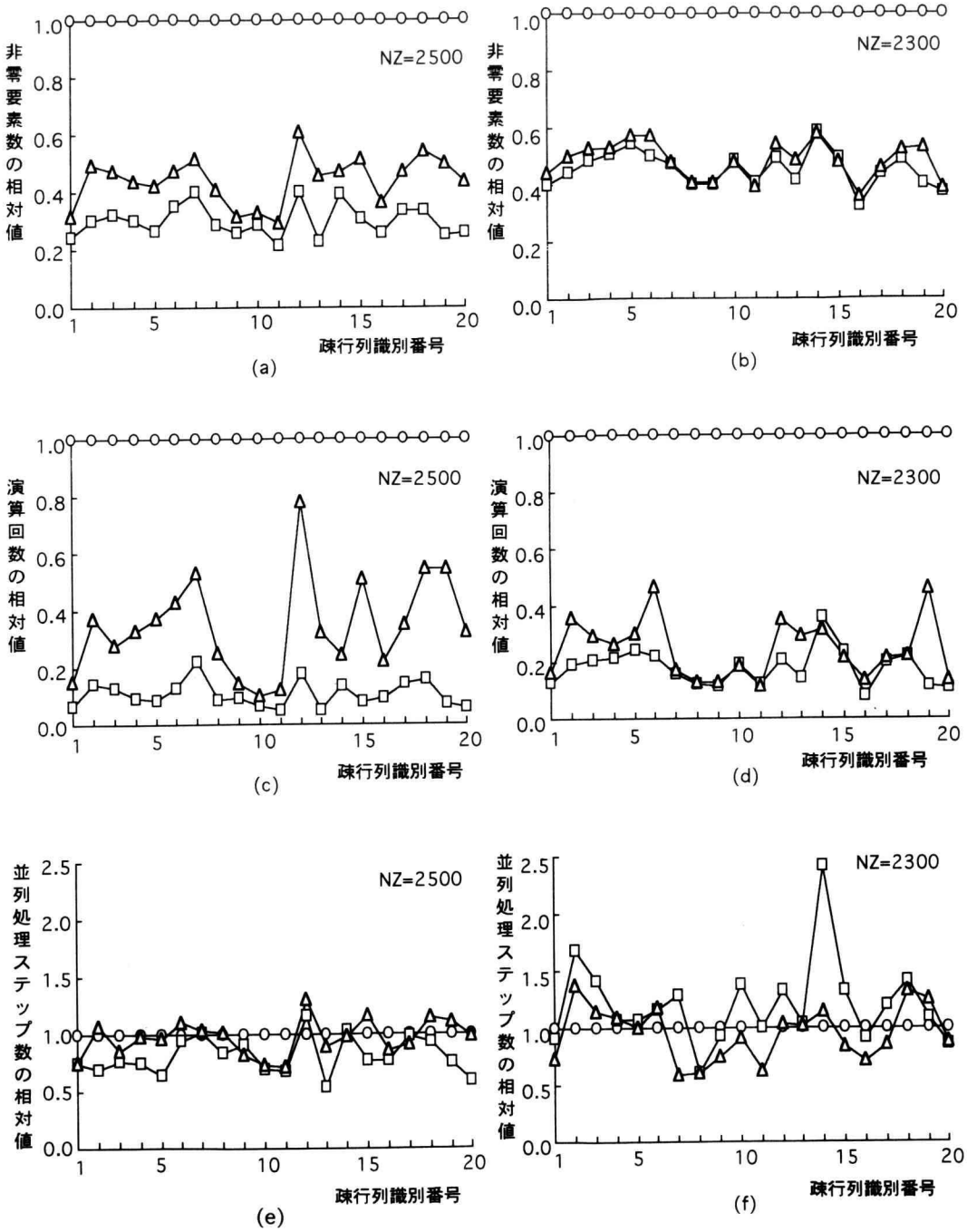


図4 LU分解における並列度解析結果（1000x1000非対称不規則疎行列40例について）

Stewart法の効果の度合いが逆転し、Stewart法の方が並列処理ステップ数をより削減できることを示している。

Stewart法について、20ケースのうち特に効果があった9ケースを調べた結果、それらは、リオーダリング時に分離した核部のブロック三角化がうまくいっている場合であると考えられる。実際、これら9ケースの場合、核部の下三角ブロック部のブロックの個数は、4～85個となっていた。このようにブロックの個数が多いことが並列処理ステップの削減に貢献していると考えられる。一方、残りの11ケースでは、この部分のブロックの個数は1個のみ、すなわち、核部がブロック三角化されていないことを意味する。

以上の結論として、以下のことが言える。

1) 一般の疎行列に対して、リオーダリングは、多くの場合、演算回数削減と同時に並列処理ステップ数の削減にも効果がある。
2)並列処理ステップ数削減効果はスパース率の変動に敏感なところがある。
3)並列処理ステップ数削減に関しては、スパース率が高いほど、Stewart法の方がTewarson法より効果的である。

5. むすび

工学の多くの分野に出現する一般疎行列のLU分解の並列処理において、行列番号のリオーダリングがどのような影響を与えるかを実験的に調べた。具体的には、1000x1000の疎行列40ケースを乱数により生成し、リオーダリングアルゴリズムとして、Stewart法とTewarson法の2つを適用した。40ケースのうち30ケースでは、何れかのリオーダリング法を適用することによって、並列処理ステップ数を削減（リオーダリングしない場合に対し）できた。その削減率は最大で50%であった。すなわち、これらのリオーダリングによって、多くの場合、演算回数が削減すると同時に、並列計算機を使った場合の処理ステップ数も削減することが分かった。

しかし、その効果の程度は、対象とする疎行列のスパース率にかなり敏感であることも判明した。この点に関しては、今後対象とする行列をさらに慎重に選択して研究を続ける必要がある。また、今回の並列処理における効果の検討は、プロセッサ台数無制限を仮定し、同期やデータ通信のオーバヘッドなど現実的な要因を入れていない論理レベルのものである。今後、実マシンを対象とした評価を行う。

参考文献

1) 山本富士男、梅谷征雄、高橋栄：大規模回路シミュレーションに対する完全LU分解付き共役残差法とその適用性評価、情報処理学会論文誌、第27巻第8号、pp. 774-782 (1986).
2) I. S. Duff: MA28 - A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations, Report R8730, HMSO, AERE Harwell (1977).
3) Z. Zlatev, J. Wasniewski, and K. Schaumburg: Y12M - Solution of Large and Sparse Systems of Linear Algebraic Equations, volume 121, Springer-Verlag (1981).
4) F. Yamamoto and S. Takahashi: Vectorized LU Decomposition Algorithms for Large-Scale Circuit Simulation, IEEE Transactions on Computer-Aided Design, Vol.CAD-4, No.3, PP.232-239(1985).
5) O. Wing and J. W. Huang: A Computational Model of Parallel Solution of Linear Equations, IEEE Trans. Computers, vol. C-29, pp. 632-638 (1980).
6) A. George, et al: Sparse Cholesky Factorization on a Local-Memory Multiprocessor, SIAM J. Sci. Statist. Comput., vol. 9, pp.327-340 (1988).
7) 山本富士男：超並列計算機向けデータ分割の自動評価方式、並列処理シンポジウムJSPP'92論文集、pp. 407-414 (1992).
8) R. P. Tewarson: Sparse Matrices, Mathematics in Science and Engineering, Volume 99, Academic Press (1973).
9) 小国力 編著：行列計算ソフトウェアーWS、スーパーコン、並列計算機ー、丸善（平成7年）.
10) 小国力 訳、J. J. Dongarra他著：コンピュータによる連立一次方程式の解法ーベクトル計算機と並列計算機ー、丸善（平成5年）.
11) 村田健郎、小国　力、唐木幸比古：スーパーコンピューター科学技術計算への適用、丸善（昭和60年）.
12) M. L. Abell and J. P. Braselton: The Mathematica Handbook, Academic Press (1992).
13) 小国　力：MATLABと利用の実際ー現代の応用数学とCGー、サイエンス社(1995).