ハイパフォーマンス
コンピューティング 47－5
（1993．6．18）

リスト構造を用いた疎な非対称連立一次方程式の直接解法

藤木健士　　竹生政資

九州工業大学情報科学センター

この研究では，ポインタを用いたリスト構造とGauss消去法を組み合わせることにより，大規模な非対称疎行列を係数にもつ連立一次方程式を比較的短い簡潔なプログラムにより効率よく解くことができることを示した．Fortranで記述した非ゼロ要素の行と列のインデクスとその値を順に配列に格納する従来の方法に比べ，Cのようなポインタを持つ言語でメモリの動的確保機能を利用しながらリスト構造で行列を格納することにより，比較的簡単に効率よく処理できるプログラムを実現できることがわかった．本稿では，この具体的な実現方法や性能評価について紹介する．

A DIRECT METHOD FOR SOLVING LARGE UNSYMMETRIC SPARSE LINEAR SYSTEMS USING LIST STRUCTURES

Takeshi Fujiki　　Masasuke Takefu

Information Science Center, Kyushu Institute of Technology

1-1 Sensuicho, Tobata-ku Kitakyushu-shi, Fukuoka, 804 Japan

We report a method solving a system of linear equations having a large unsymmetric sparse matrix in C language which simplifies the implimentation of the Gauss elimination algorism by introducing the list structures with pointers, not arrays in Fortran, and the dynamic memory allocation. The result is that the programs developed are more simple and straightforward, even efficient on workstations, resulting smaller size of source codes, than the conventional ones written in Fortran.

1 はじめに

大規模な疎行列を係数とする連立一次方程式を直接法で解く際に，係数行列の大部分の要素が0であるという性質を利用して適当なデータ構造に圧縮することにより，記憶領域を節約し高速に演算する手法がいくつか考案されている．これらの中で，連立方程式を表現する行列の形が帯状のものや対称であるものを処理するプログラムについては，メモリを効率的に利用し高速に演算を行なうものが報告されている．しかし非対称な疎行列の連立一次方程式を解く汎用的なプログラムは例が少ない．しかも，これらの例では複雑なデータ構造を用いているため，その取り扱いもかなり複雑なものとなっている場合が多い．[2, 3, 4]

この主な原因のひとつとして，プログラミング言語 Fortran を用いることを前提としていることが考えられる．Fortran はポインタ型やアドレス演算を持たないので，メモリを効率的に用いるためには，配列を工夫して複雑なデータ構造を実現する必要がある．また言語の標準的な機能として動的なメモリ割り付け機能が提供されていないため，計算過程で行列の0でない要素が動的に変化するのに応じて記憶領域を効率良く利用していく上で限界がある．例えば，計算開始前に使用するメモリ領域を予想して静的に割り付けたとしても，もし計算の途中で予想以上の記憶領域が必要になった場合には，それに対応できず実行を中止するしかない．

このような複雑な処理に起因する速度低下をできるだけ抑え，記憶領域の効率的な利用を実現するためには，ポインタとメモリの動的割り付け機能を備えた言語を用いるのが望ましい．本稿ではリスト構造による記憶領域の効率的な管理方法および計算処理方法をプログラミング言語 C で実現した例を説明する．また，これにより作成したプログラムについて実際に処理を行なった結果に基づき実行時間やメモリ利用状況などの評価を行なった．

2 疎行列を格納するデータ構造

今回作成したプログラムでは，非対称疎行列を取り扱うためにメモリの利用効率と処理効率を考慮して，ポインタを用いたリスト構造を採用した．具体的には図1に示すように，各行の0でない要素を昇順につないだリスト構造と，各行の先頭を指すポインタの配列でデータ構造を構成した．このようなデータ構造を用いた理由は次のとおりである．

1. メモリを効率よく利用することができ，それに伴う負荷が軽くなる．
2. 0でない要素の挿入 (fill in) や不要な領域の削除が容易である．
3. ピボット選択の際に，行の入れ換えがポインタのつけ換えだけで済む．

まず第1項目は，メモリの利用効率を向上させ大規模な計算を可能にするために必要である．これを実現するために，必要最小限のデータに対してのみメモリを割り当てる動的なメモリ割り付けを行なうようにした．つまり，計算過程で不要になった領域は不要になった時点で解放し，領域が必要になった時点でそれを再利用するようにした．こうした機能の実現にリスト構造は適しており，このことについては次章でさらに詳しく述べる．

次に第2項目に関しては，一般に疎行列に対して Gauss 消去を施すと，その行列は元の行列よりも密になることが知られている．これは計算過程で0であった要素が0でなくなり，データ構造への追加 (fill in) が生じることを意味している．ポインタでつながれたリスト構造を用いると，この追加処理は領域を確保して2つのポインタを付け換えてリストに挿入するだけでよく，非常に効率が良い．

またピボット選択については，今回作成したプログラムでは行の交換のみを行なう部分ピボット選択を採用した．このデータ構造を用いると，ピボット選択の際に要素をコピーする必要なく，行の先頭を指すポインタを付け換えるだけよい．したがって，コピーのために必要なバッファ領域が必要でなく，負荷も軽い．この方法は密行列の場合に対しても適用することができる．

3 記憶領域の利用方針およびその管理方法

3.1 主記憶に関して

連立一次方程式を解く際の元数の上限は記憶領域の制限による場合が多い．可能な限り大規模な計算を行なうためには，できる限りメモリを節約しなければならない．連立一次方程式を解く直接法のアルゴリズムとしては，通常 Gauss 消去法またはクラウト法による LU 分解の方法が用いられている．しかし，本研究では極力メモリを節約するために，LU 分解をせず部分ピボット選択だけを行なう Gauss 消去法を用いた．係数行列をいったん LU 分解し，続いて前進代入と後退代入により解を求める方法は，連立方程式の定数ベクトル部分だけを変えて何度も同じ係数行列の方程式を解く場合には効率がよい．我々の方法は容易にこのような LU 分解を含む汎用的なプログラムに拡張することができる．

$$\begin{bmatrix} 3.1 & 0 & 2.4 & 1.2 \\ 0 & 4.8 & 0 & 7.2 \\ 0 & 2.1 & 0 & 0 \\ 5.3 & 0 & 0 & 1.5 \end{bmatrix}$$

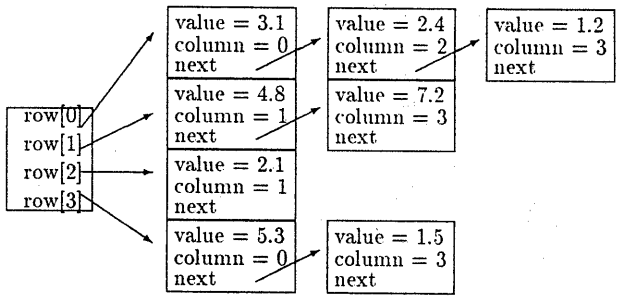

図 1: リストを用いた行列の表現

Gauss 消去法では，計算が進行するにつれて下三角行列に不要な要素が発生し，一方上三角行列では fill in が発生して新たに領域が必要となるので，メモリをできるだけ有効に利用するためには動的にメモリを管理する必要がある．今回作成したプログラムでは，再利用可能なメモリ領域をポインタでつないだリスト構造により管理した．ある領域が不要になり解放されたら，それを再利用のためのリストに追加し，新たな領域が必要になった時点でそのリストから割り付けるようにした．再利用可能な領域をすべて使い切ったら，新たに領域を外部†から確保するようにした．

以上の方法を用いるとガベージコレクションの必要がなく，比較的軽い負荷で動的に記憶領域を管理し，効率良く利用することができる．

3.2 補助記憶に関して

Gauss 消去法を用いて連立一次方程式を解く場合，前方消去における N 行目の処理が終了すると，次に N 行目が参照されるのは最下行から始まる後方代入処理が N 行目に到達する時である．この性質を利用すると，かなり大規模な連立一次方程式でも補助記憶装置を併用することにより比較的高速に処理することができる．以下にメモリ管理の指針を示す．

1. 計算開始時点ではすべての要素を主記憶上に置く．
2. 不要になった下三角行列の領域を再利用しながら計算を進め，領域が不足したらその時点の枢軸行の直前までの行のデータを補助記憶装置に書き出し，これにより不要になった領域を再利用して計算を進める．
3. 上の処理を行なっても領域が不足する場合には外部†から領域を確保する．

この方針で補助記憶装置を併用した場合，主記憶装置のみを用いて計算する場合の約 1.5 倍程度の時間で処理できることがわかった．

†C 言語を用いる場合には malloc 関数を用いて OS に記憶領域の動的に割りつけを要求する．

4 リストで表現された行列での Gauss 消去

行列データを配列に格納した場合，各要素へのアクセスはどのような順序で行なってもコストは同じである．しかし，本研究で用いた単方向のリスト構造では，その構造上，各要素へのアクセスが逆順やランダムな場合非常にコストが高くなる．したがって処理を高速に行なうためには，できるだけリストを先頭から順にスキャンして処理を行なうようにプログラムを設計する必要がある．

今回作成したプログラムでは，第 N 行の処理を行なう時点で各行のリストの先頭を指すポインタは第 N 列以降の最初の 0 でない要素を指す．これによりピボット選択時にリンクをたどる手間をなくすことができる．また Gauss 消去法の前進消去では

$$a_{ij} = a_{ij} - a_{ik} * a_{kj}/a_{kk}$$

の計算を行なうので，K 行目を枢軸行とすると，値の変化が生じるのは K 列の要素が 0 でない行の要素のうち，枢軸行の要素が 0 でない列のものである．計算を要する各行をスキャンして，枢軸行の要素が 0 でない列の要素に関して値の更新または挿入を行なう．このような工夫を行なうことにより，連立一次方程式を解く処理では行列要素に対してランダムにアクセスする必要はなく，一方向の順次アクセスだけで十分である．このことからリスト構造は疎行列の計算処理に適していることがわかる．ただし，間接アドレスを用いる方法はベクトル計算機などのベクトル処理ではむしろ速度低下の要因となる場合があるが，ワークステーションなどのスカラー計算機では十分の性能を発揮するものと思われる．

5 Fortran ではなく C 言語を用いた理由

数値計算プログラムは Fortran 言語を用いて記述するのが慣習となっている．しかし我々は今回，大規模な疎行列連立一次方程式を解くプログラムを開発するために Fortran ではなく敢えてポインタ型を持つ C 言語を用いた．理由のひとつは，最近 C 言語が規格化され比較的高性能のコンパイラが身近に利用できるようになったことである．また C 言語で開発した関数を Fortran の中から非常に簡単に (あたかも Fortran で開発したサブルーチンと全く同じ形式で) 利用できるようになり，もはやプログラム全体を「ひとつの」言語で統一することにこだわる時代ではなくなってきた．しかし最も大きな理由は，リストのようにポインタを用いたデータ構造では，アドレス演算が多くなるにつれて Fortran による記述はかなり困難となり，しかも実行効率が悪くなることである．この原因について少し詳しく考えてみる．

Fortran にはポインタ型やポインタ演算の機能がないため，通常整数の配列にデータの添字を入れてポインタの代用としている．従来の疎行列連立一次方程式を解くプログラムでも，そのようなデータ構造が用いられている．例えば，図 2に示すリスト構造を Fortran で実現するためには，図 3に示すように，値の入っている配列と次のリストの値が入っている配列を用いて実現する．これに対してポインタを持つ言語では構造体とポインタを用いてデータ構造をつくるのが一般的である．図 4に C 言語による例を示す．

さて，配列でリスト構造を実現した場合とポインタで実現した場合のそれぞれについて必要な処理のコストを比較してみる．リスト構造を用いて処理する場合，リンクをたどりながら処理を行なうという動作を繰り返す．そこでいま，リストをたどり次のノードにアクセスするという基本的な 1 サイクルの動作コストを比較してみる．以下にそれぞれの動作を行なう SPARC のコードの例を示す．

1. Fortran77 による配列処理の SPARC コード

```
i = ptr(i)
```

レジスタ o0 に添字が入っている状態から，リストの次のデータの添字が o0 に入るまでの命令列

```
sethi   %hi(BaseAddress),%o1
or      %o2,%lo(BaseAddress),%o1
sll     %o0,2,%o0
ld      [%o0+%o1],%o0
```

2. C によるポインタ処理の SPARC コード

```
p = p->next;
```

レジスタ o0 にアドレスが入っている状態から，リストの次のデータのアドレスを得るのに必要な命令列

```
ld      [%o0+8],%o0
```

上の例で示したように Fortran で配列を用いた場合，まず配列の先頭アドレスをレジスタにロードし，1 つの整数につき 4 バイトの領域が確保されるので，添字を 4 倍してオフセットを得て，それら 2 つを加算したアドレスから次の添字を得る．一方 C でポインタを使った場合，ポインタのアドレスに次のノードのアドレスを格納する変数 next の構造体内のオフセット 8 を加算して，そのアドレスからデータをロードすればよい．配列を利用した場合には，最適化によりベースアドレスをレジスタに割り付けて前半の処理が必要なくなったとしても，添字からオフ

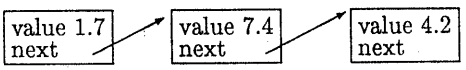

図 2: リスト構造の例

```
real*8   value(MAX)
integer  ptr(MAX)
```

$value(i) = 1.7$	$value(j) = 7.4$	$value(k) = 4.2$
$ptr(i) = j$	$ptr(j) = k$	$ptr(k) = 0$

図 3: Fortran77 によるリスト構造の実現の例

```
struct node{
   double value;
   struct node *next;
};
```

	(address P)	(address Q)
$value = 1.7$	$value = 7.4$	value = 4.2
$next = P$	$next = Q$	$next = 0$

図 4: C によるリスト構造の実現の例

セットを得るためのシフト演算またはかけ算の処理が余計にかかり，ポインタを利用する場合よりもコストが高い．また配列の添字あるいは構造体へのアドレスが変わっている時点から値をアクセスする場合も同様に，ポインタを用いた方がコストが低い．このように頻繁に行なう処理の中に差が生じると，ポインタが利用できる言語とできない言語では全体としての負荷にかなりの差が生じる．

この他に Fortran は動的な記憶領域確保ができないという欠点がある．今回作成したプログラムのように，実行中に必要な記憶領域が不規則的に増加していくような場合，Fortran では実行を開始する前に予め必要なメモリを経験的に予想し静的に確保しておかなければならない．このため，しばしば予想以上に領域が必要になりプログラムが計算途中で中断したり，試行錯誤により必要領域を知るためにテスト的に何度も実行を繰り返さなければならない．また実行開始時に静的に記憶領域を確保するので，計算が長時間にわたる場合でも最初から最後までずっとその記憶領域を確保し続けなければならない．最初は少量の領域しか必要なく，処理の進行に伴い必要な領域が増加するような処理では動的に記憶領域を利用した方が計算機全体としてのスループットは向上する．

上で述べたように，今回作成したプログラムの実現においては，ポインタ型や動的なメモリ割り付け機能を利用できる言語の方がいろいろな面で有利である．これまで数値計算用のプログラムにおいては専ら Fortran が用いられてきたが，場合によっては別の言語でプログラムを作成した方がプログラミングが容易になり，実行効率も向上する場合がしばしばある．最近，いろいろなプログラミング技術を取り入れた新しい Fortran90 の国際規格が定められたが，まだ普及するまでにはすこし時間がかかると思われる．

今後はそれぞれの目的に最も適したデータ構造をもつ言語を積極的に活用し，効率の良い優れた数値計算プログラムを開発していくべきであろう．従来のソフトウェア資産を用いたり，メインプログラムに Fortran を用いるエンドユーザへのサービスに関しては，最近実用的に用いられている他言語とのリンク技術が有効である．

6 性能評価

上で述べた方法の性能を評価するために，集団遺伝学において突然変異や自然淘汰，世代交代時のランダムサンプリングなどの効果による遺伝子頻度の固定に要する時間を記述する 2 次元拡散方程式 (遺伝子組換え効果は無視) の解を求める計算を行なった[5]．この問題では大部分の境界条件が方程式自体の中に組み込まれているため，差分化された連立一次方程式の係数行列は対称にならない．表 1 は，主記憶 48MB，スワップ領域 117MB の Sparc Station 2(サンマイクロシステムズ社製，以下 SS2 と呼ぶ) を用いて，次元数をいろいろ変えながら，主記憶のみを用いた場合と補助記憶装置を併用した場合について，それぞれの計算に要した時間を測定したものである．表に示された CPU 時間は UNIX の time コマンドを用いて測定した．なお，表中の分割数は x 座標 ($1 \geq x \geq 0$) と y 座標 ($1 \geq y \geq 0$) の分割数を表し，元数は (分割数 +1) の 2 乗である．また，非零要素率は係数行列の非零要素数を全要素数で割り 100 をかけて百分率で表したものである．左側の CPU 時間はメモリだけを用いて計算したときの計算時間 (カッコ中の数値は実際に使用されたメモリ量)，右側の CPU 時間は補助記憶装置を併用した場合の計算時間 (カッコ内の数値は実際に使用された主記憶量と補助記憶量) を表したものである．表 1 から以下のことがわかる．

- 偏微分方程式の差分化によって得られる連立一次方程式の係数行列の非零要素率は一般に非常に小さい．
- 我々の開発したプログラムはメモリだけを用いた場合，約 4000 元の連立方程式を約 1 分，約 30000 元を約 1 時間程度で解くことができる．
- 補助記憶装置を併用した場合，実行速度はメモリだけを用いる場合に比べて約 1.5 倍になるが，40000 元以上の連立一次方程式でも解くことができる．
- 補助記憶装置を用いない場合 40000 元以上の計算はできない (主記憶とスワップ領域のサイズにも依存する)．

表 2 はアルゴリズム，係数行列，機種の違いによる処理性能を比較したものである．最初の CPU 時間は SS2 における我々のプログラムの性能で，表 1 に示したものと同じである．第 2 番目は同じ連立一次方程式を IMSL の dlslxg サブルーチンによって解いたものである．分割数が 128 以上ではパラメータをいろいろ調整しても “Bus error” が発生して計算が異常終了した．第 3 番目は係数行列の非零要素数は同じだがその場所と値を一様乱数 (対

角要素だけは非零に固定) によって作成し計算したものである．最後の CPU 時間は最初の 2 つと同じ連立一次方程式を機種を Sparc Station 10 (以下 SS10 と呼ぶ) に替えて実行したものである．表 2 から次のような傾向が読みとれる．

- 我々の開発したプログラムは IMSL の dlslxg より若干性能がよい．しかし dlslxg は汎用性をもたせるため LU 分解を行なっているが我々のものは行なっていない．したがって，この差が純粋にアルゴリズムの差に帰着できるかどうか明らかではない．特に dlslxg では 128 分割以上を解くことができないが，これは LU 分解のために下三角領域を保持するために余計なメモリが必要となることが原因かもしれない．
- 係数行列の非零要素の配置がランダムな場合，その配置によって計算時間にある程度バラツキが生じる．
- SS2 と SS10 の性能を比較した場合，元数が小さいときは SS10 の方が性能がよいが元数が大きくなるにつれて SS2 よりも性能が悪くなる．これは SS10 が新しいアーキテクチャであるためコンパイラがまだ SS10 の性能を十分に引き出していないことが考えられる．

最後に，参考のため，九州大学大型計算機センターに設置してある富士通製汎用大型計算機 (主記憶:256MB) およびベクトル計算機 (主記憶:512MB, ただし一般ジョブは 200MB まで) との比較性能を表 3 に示す．使用した大型計算機には一般的な疎行列係数の連立一次方程式を解く標準ライブラリが用意されていないので，「違う土俵」での比較にはなるが，上と同じく拡散方程式の問題を一般的な密行列用のサブルーチンを用いて解いてみた．汎用大型計算機およびベクトル計算機用のサブルーチンとしてはクラウト法による dlax および dvlax(富士通 SSL-II および SSL-II/VP)，ワークステーション (SS2; 主記憶:48MB) 用のサブルーチンとしては IMSL の dlslrg (Gauss 消去法) を用いた．表 3 の検討から次のことがわかる．

- 汎用大型計算機は SS2 の約 3 倍程度の性能である．ただし，使用したサブルーチンが全く同じものではないので正確な性能比較はできない．
- 汎用大型計算機とワークステーションでは密行列係数の連立一次方程式については 4000 元以上は困難である．ベクトル計算機でも 4000 元程度が限界である．
- ベクトル計算機はワークステーション (SS2) に比べて約 400 倍から約 600 倍程度速い．

7 終りに

本研究を通して，我々は特に次のことを強調したい．従来は，主にソフトウェア資産の豊富さから数値計算言語として専ら Fortran が用いられてきたが，近年ワークステーションやパソコンの性能が著しく向上し，Fortran コンパイラに加えて高性能の C コンパイラが広く利用できるようになった．このような環境下では，たとえメインプログラムを Fortran 言語で開発する場合であっても，部品としてのライブラリ開発には積極的に C 言語を活用すべきである．これによって，しばしばアルゴリズムの大幅な簡略化と，ライブラリの使いやすさ (特に作業配列の動的確保など)，性能の向上などが比較的容易に実現できることがある．

今後の課題としてはリオーダリングを行なって fill-in を減らし，さらにメモリを節約する方法を検討したい．またページフォルトを減らす方法や並列処理におけるマッピングの研究を行なう予定である．

参考文献

[1] J.J.Dongarra, I.S.Duff, D.C.Sorensen, H.A.van der Vorst, Solving Linear Systems on Vector and Shared Memory Computers, SIAM 1991.

[2] I.S. Duff, MA28-A set of Fortran subroutines for sparse unsymmetric linear equations. AERE-R8730 (Revis.),ii+104, 1980.

[3] Z. Zlatev, J. Wasniewski, and K. Schaumburg. Y12M - Solution of Large and Sparse Systems of Linear Algebraic Equations, Volume 121. Springer-Verlag,New York,1981.

[4] User's manual; MATH/LIBRARY Ver.2.0 (1.1.13 Real Sparse Linear Equation Solvers, lslxg, P.237), IMSL, Inc., 1991.

[5] 竹生 政資, 飯塚 勝：拡散過程に付随した拡散方程式の数値計算法について，計算機科学研究報告，九州大学大型計算機センター， Vol.10， pp. 7 - 16 ， 1993.

分割数	元数	非零要素率 (前方消去後)	CPU 時間 (メモリ)	CPU 時間 (メモリ, ディスク)
32	1089	0.45880(3.20063)	3.5(1788)	5.0(1808, 0)
48	2401	0.20818(2.08333)	18.4(3568)	25.6(2044, 1512)
64	4225	0.11832(1.56250)	59.5(5424)	87.2(2364, 4037)
80	6561	0.07620(1.25000)	157.6(10488)	217.4(2776, 7972)
128	16641	0.03004(0.78125)	1028.5(37128)	1368.8(6112, 33289)
160	25921	0.01929(0.62500)	2480.9(71460)	3408.8(9308, 66203)
184	34225	0.01461(0.54348)	4272.8(105632)	5719.2(10764, 99941)
200	40401	0.01238(0.50000)	—(—)	7033.0(13412, 127836)
208	43681	0.01145(0.48077)	—(—)	9627.4(13992, 145062)

注 1. CPU 時間の単位は sec, メモリおよびディスクの単位は KB. 非零要素率の単位は百分率.

注 2. 使用機種は Sun Mycrosystems Inc. 製 Sparc Station 2 (Memory:48MB, SWAP:117MB).

注 3. 使用コンパイラ : Sun Fortran V1.4(最適化オプション 4), SunOS C Compiler V4.1(最適化オプション 4).

表 1: SS2 による 2 次元拡散方程式境界値問題の処理性能 (汎用疎行列ルーチン)

分割数	元数	非零要素率	CPU(SS2)	CPU(SS2;IMSL)	CPU(ランダム;SS2)	CPU(SS10)
32	1089	0.55063	3.5	5.0	7.3	3.0
48	2401	0.24983	18.4	28.5	23.6	12.1
64	4225	0.14199	59.5	60.4	45.0	36.1
80	6561	0.09144	157.6	175.9	96.2	89.9
128	16641	0.03605	1028.5	—	493.9	697.8
160	25921	0.02315	2480.9	—	1201.7	1796.2

注 1. CPU 時間の単位は sec. 第 4,5,7 列は拡散方程式, 第 6 列はランダム係数行列の連立一次方程式を解いたもの.

注 2. 第 5 列は IMSL の dlslxg ルーチンを使用. 第 7 列は SS10 (Memory:48MB, SWAP:126MB) を使用.

注 3. 使用コンパイラ : Sun Fortran V1.4(最適化オプション 4), SunOS C Compiler V4.1(最適化オプション 4).

表 2: アルゴリズム, 係数行列, 機種の違いによる処理性能の比較 (汎用疎行列ルーチン)

分割数	元数	非零要素率	CPU(dlslrg) SS2	CPU(dlax) M-1800/20	CPU(dlax) VP-2600/10	CPU(dvlax) VP-2600/10
32	1089	0.55063	247.8	69.5	0.6	0.5
48	2401	0.24983	2965.0	1244.1	5.1	4.4
64	4225	0.14199	—	—	26.0	23.0

q 注 1. CPU 時間の単位は sec. 使用した Fortran コンパイラは以下のとおり.

注 2. dlax は富士通 (株) 製科学用サブルーチンライブラリ SSL-II の中のサブルーチンの 1 つである.

注 3. dlslrg は IMSL の汎用密行列係数の連立一次方程式を解くサブルーチンの 1 つである.

注 4. Sun Mycrosystems Inc. 製 Sparc Station 10: Sun Fortran V1.4(最適化オプション 4).

富士通 (株) 製汎用大型計算機 M-1800/20: FORTRAN77 EX V12L10.

富士通 (株) 製ベクトル計算機 VP-2600/10(理論最大性能 5GFLOPS): FORTRAN77 EX/VP V12L10.

表 3: SS2, 汎用大型計算機, ベクトル計算機による処理性能の比較 (密行列ルーチン)