数値解析 20－3
（1987. 3. 14）

問題の性質を利用した
大規模スパース行列の高速解法

福井　義成
（株）東芝　総合情報システム部

本報告では，回路解析のコード生成による高速化の結果を問題の性質を利用した高速解法の一つの例という面から述べる．

大規模な問題を解く場合に，その問題の持つ性質を利用して，効率的良く計算する手法の研究が行われている．利用可能な問題の性質としては，（1）非線形問題の構造，（2）問題の性質を反映したデータ構造，などがある．SPICE－GT（CRAY X－MP版）のコード生成も，データの構造を利用して，大規模スパース行列を効率良く解く例となっている．ここでは，gater／scatterの機能を持たない場合のコード生成の例について報告する．この場合でも，FORTRANに比べて重要な問題で約26倍速くなり，通常のベクトル化以上の高速化を実現している．

A CODE-GENERATION METHOD
FOR LARGE RANDOM SPARSE MATRICES

Yoshinari Fukui

Total Information & Systems Division, TOSHIBA Corporation
72, Horikawa-cho, Saiwai-ku, Kawasaki-shi 210, Japan

We can generate an efficient code if we utilize the characteristic property of a given problem. This paper describes automatic generation of such fast codes for large random sparse matrices. The codes generated are linear equation solvers written in a machine language. Compared a code generated for a large random sparse matrix with a conventional FORTRAN subroutine on a CRAY X-MP, we found out code was about 26 times faster than the latter.

１．はじめに

我々は，回路解析プログラムの高速化を行ってきた．マルチ・タスクの手法を用いた結果については，既に報告した[1]．ここでは，コード生成による高速化の結果を問題の性質を利用した高速解法の一つの例という面から述べる．

大規模な問題を解く場合に，その問題の持つ性質を利用して，効率的良く計算する手法の研究が行われている[2],[3]．利用可能な問題の性質としては，

（１）非線形問題の構造

（２）問題の性質を反映したデータ構造

などがある．ＳＰＩＣＥ－ＧＴ（ＣＲＡＹ　Ｘ－ＭＰ版）のコード生成も，データの構造を利用して，大規模スパース行列を効率良く解く典型的な例である．ここでは，ｇａｔｅｒ／ｓｃａｔｔｅｒの機能を持たない場合のコード生成の例について報告する．この場合でも，ＦＯＲＴＲＡＮに比べて重要な問題で約２６倍速くなり，通常のベクトル化以上の高速化を実現している．ＳＰＩＣＥ－ＧＴは，ＳＰＩＣＥ２Ｇ．６をベースに東芝で改良・機能強化したものである．

２．解法の適用範囲

計算機の出現以来，数値計算の分野では，汎用の数値解法に力が注がれてきたものと思われる．ところが，非常に大規模な計算を行う場合，汎用の解法では，十分な速度が得られないものがある．例えば，ランダムなスパース行列は，汎用の解法ルーチンでは十分な性能が得られないことがある．密行列の消去法の場合には，データの規則性が良いため，問題のもつデータ構造の特徴を利用して速くする余地は小さいが，ランダムなスパース行列の場合には，問題にごとのデータ構造の特徴を利用して，高速化する余地がある．ＦＯＲＴＲＡＮで書かれた汎用のランダム・スパース行列の消去法ルーチンは，汎用化のため，データの配置の情報を間接番地で持たなければならない[4]．しかし，個々の問題が決まれば，その問題の持つデータ構造の特徴を利用して，専用の解法ルーチンを作ることが可能となる．

密行列を扱う場合は図１の例のようになっており，比較的簡単なデータ構造を連続的に扱うことができるため，スーパ・コンピュータではベクトル化がうまく行うことができ，このままで十分高速化が実現できる．そのため，データ構造の面から，一般化された解法ルーチンを改善する余地はあまりない．しかし，ランダム・スパース行列の場合は図２の例のようになる．メモリへの間接番地による参照をベクトル化する機能のないスーパ・コンピュータではベクトル化がうまく行われず，このプログラムは高速に処理されない．メモリへの間接番地による参

```
      ＤＯ　１００　Ｊ＝１，１００
      ＤＯ　１００　Ｉ＝１，１００
          Ａ（Ｉ，Ｊ）＝Ｂ（Ｉ，Ｊ）＊＊２
１００　ＣＯＮＴＩＮＵＥ
```

図１．　密行列を扱う例

```
      DO  200  I=1,1000
         A(L1(I))=B(L2(I))**2
200   CONTINUE
```

図２．　ランダム・スパース行列を扱う例

照をベクトル化する機能があっても，メモリへの参照がほぼ２倍になるため，図１の例ほどには速くならない．ランダム・スパース行列の場合，非ゼロ・データの処理方法によって，効率が大きく異なる．

メモリへの間接番地による参照をベクトル化する機能のない場合，最も効果的なのが，プログラム・ループを展開し，プログラムを書き下すことである．例として，ある配列の不規則な要素の和をとる場合，ＦＯＲＴＲＡＮでは普通，図３の例のようにするであろうが，これを図４の例のように展開した形で計算すれば，メモリへの間接番地による参照がなくなり，速く計算できる．しかも，図３の例ではループを制御するための手間が必要であるが，図４の例では，これが不必要であり，その効果によっても速くなる．一方，ループを展開するので，プログラムを格納するための領域（メモリ）が大きくなってしまう．プログラムを格納するためのメモリは，行列の非零要素の数に比例するので，密行列にこの方法を適用するのは，ほとんど無意味であるが，非零要素が１％程度のランダム・スパース行列に対しては，効果が大きい．

```
      DATA  L/1,3,6,7,15,16,21,63/
      S=0.0
      DO  100  I=1,8
         S=S+A(L(I))
100   CONTINUE
```

図３．　ループ構造で不規則な要素の和をとる例

```
      S=0.0
      S=S+A(  1)
      S=S+A(  3)
      S=S+A(  6)
      S=S+A(  7)
      S=S+A(15)
      S=S+A(16)
      S=S+A(21)
      S=S+A(63)
```

図４．　展開形で不規則な要素の和をとる例

図４に示したように，この方法は原理的には可能であるが，ＦＯＲＴＲＡＮのようなコンパイラ言語で，実現するのは非現実的である．ＦＯＲＴＲＡＮでは，

個々のデータ構造によって，別々のプログラムとする以外には方法がない．したがって，これをＦＯＲＴＲＡＮで行うには，ある固定のデータ構造しか扱うことができず，汎用性がなくなってしまう．汎用性を残し，自由なデータ構造を扱えるようにする手段としては，

（１）数式処理システム等を利用し，個々のデータ構造ごとに自動的にＦＯＲＴＲＡＮソースを生成させる．

（２）実行時にメモリ内に機械語でプログラムを生成する．（コード生成）

がある．

数式処理を利用する方法は，筆者らも，ある固定のデータ構造ですむ場合に，電車のシミュレーションやロボットの解析において既に利用している[5],[6]．しかし，現在の数式システムでは，ＦＯＲＴＲＡＮプログラムを一旦停止し，ＲＥＤＵＣＥ等を使いＦＯＲＴＲＡＮソースをファイルに書き，コンパイル・再リンクしなければならない[7]．コード生成も面倒なようであるが，生成するコードは非常に種類が少なくて済む場合は，コードを生成するサブルーチンを用意しておけば簡単である．（ただし，普通にＦＯＲＴＲＡＮを使うのとは異なるため，オペレーティング・システムやコンパイラの変更があっても，再コンパイルするだけで済むように考慮する必要がある．）

また，回路解析などの場合，生成しなければならないコードが非常に長くなることがある．このような場合が実用上最も重要であが，１０，０００ステップ以上のサブルーチンをコンパイルしてくれるＦＯＲＴＲＡＮは少ないので，コード生成が必要となる．

３．回路解析におけるコード生成

回路解析において，回路の過渡応答を解析する場合，ニュートン法のプロセスで係数が毎回異なる連立一次方程式を多数回解く必要がある．この解法ルーチンをＦＯＲＴＲＡＮで記述した場合は，使用ＣＰＵ時間の大半を消費することが普通である．したがって，この部分を高速化し，大規模問題を速く計算できるようにすることが，製品開発において重要になっている．この解法ルーチンを高速化する手段の１つとしてコード生成法がある[8]．以下において，回路解析にコード生成法を用いた結果について述べる．

３．１　生成するコード

ＳＰＩＣＥ－ＧＴにおけるコード生成は，通常はＦＯＲＴＲＡＮで書かれた連立一次方程式を解くルーチンと同じ機能をもつコードを，プログラムの実行中に機械語レベルでメモリ中に生成し，それを実行するものである．必要となる機能は，下記の６つである．

（１）サブルーチンの前処理

（２）ピボッティングの処理

（３）除算

（４）内積

（５）内積の変形（除算との組み合わせで使う）

（６）サブルーチンの後処理

ｆｉｌｌ－ｉｎ　等を考慮し，これらの機能を用いてコードを構成するアルゴリズムについては文献を参照されたい[9]．この部分については各計算機の特徴に無関係になつており，どの計算機でも共通である．コード生成項目（１）～（６）では，以下の機能を果たさなければならない．

（１）サブルーチンの前処理

・このサブルーチンを呼んだルーチンを覚えておく
・フラグ等を覚えておく
・引数の結合をする

（２）ピボッティングの処理

・連立一次方程式の対角項の大きさをチェツクする
・対角項が小さい場合は，あるεで置き換える
・置き換えの回数を数える

（３）除算

・対角項でその行の要素を割る

（４）内積

・連立一次方程式の消去を行う

（５）内積の変形

・除算との組み合わせで消去を行う

（６）サブルーチンの後処理

・対角項の置き換えの回数をこのコードを呼んだルーチンへ返す
・フラグ等を元に戻す
・このコードを呼んだルーチンへ戻る

上記項目のうち（１）と（６）は，他のルーチンとのインターフェス部分であるためオペレーティング・システムやコンパイラのバージョンに依存する可能性がある．さらにＳＰＩＣＥにおいては，生成されたコードの領域が他のＳＰＩＣＥのデータと同様に扱われるため，計算途中で，生成されたコードの位置が移動する点に特徴がある．このため，生成されたコードが，再配置可能なものでなければならない．

３．２　オペレーティング・システムのバージョンに対する独立性

生成されたコードも，メインのＦＯＲＴＲＡＮから見た場合，独立したサブルーチンとして動作しなければならない．そのため，生成されたコードの先頭に前処理，最後に後処理をするものが必要である．しかし，このサブルーチン間の呼び出し手順は，オペレーティング・システムのバージョンによって変更されることがある．現に，ＣＲＡＹ　Ｘ－ＭＰ用のコード生成を開発している時点で，ＣＯＳ１．１１とＣＯＳ１．１３では，サブルーチンの呼び出し手順が大きく変更になり，機械語レベルでは互換性がまったくなかった．また，将来のオペレーティング・システムやコンパイラの変更を考慮すると，サブルーチン呼び出し手順とは独立なコードを生成する必要がある．汎用性を高めるため，サブルーチン呼び出し手順は，ＦＯＲＴＲＡＮとアセンブラが自動的に処理してくれるリンクマ

クロを利用することとした．ＦＯＲＴＲＡＮとアセンブラのリンクマクロで，サブルーチン呼び出しのバージョン依存性を吸収した．このようにすれば，オペレーティング・システムやコンパイラが変更になった場合も，ＦＯＲＴＲＡＮとアセンブラを再コンパイル・再アセンブルするだけで良く，生成するコードについてはまったく変更する必要がない．ＣＲＡＹでは，ＣＯＳ１．１１，ＣＯＳ１．１３，ＣＯＳ１．１４，ＣＯＳ１．１５において，再コンパイル・再アセンブルだけで動いている．

３．３　再配置可能性

再配置可能性とは，生成したコードがメモリ内で場所を移動しても，何ら変更なく，このコードが実行可能であることである．上記項目の中で，（２）以外は，ほとんどの計算機で問題がないが，（２）のＩＦ文の処理が再配置可能かどうかは，計算機に依存する．上記の条件をそのまま作成すると，条件付きジャンプを行わなければならず，その飛び先番地が固定であると再配置可能とはならない．例えば，ＡＣＯＳ－６においては，プログラム・カウンタ（ＩＣ）の内容に定数を加えた番地へ条件分岐可能なため，上記の機能を満足することは，容易である．しかし，ＣＲＡＹ　Ｘ－ＭＰでは，現在実行中の番地から，ある決まった大きさを加えた番地へ条件分岐する機能（相対番地分岐）がないため，これと同じ効果をレジスタ間のシフトで実現している．（相対番地分岐は，パイプラインを乱すため，ＣＲＡＹ　Ｘ－ＭＰでは用意されていないものと思われる．）

３．４　高速化

ＣＲＡＹのようなスーパー・コンピュータでは，独立して動作可能な演算ユニットが複数あり，これらを同時に動作させると計算の高速化が図れる．そのため，順番を入れ換えても結果が変わらない演算は，レジスタや演算ユニットの競合をなるべく少なくするようなコードを生成すように工夫した．

生成されるコードは，上記の６種類である．この中で，（１）と（６）は，各々一回しか使用されない．残りは，連立一次方程式を解くルーチンを作成することを考えると高速化の可能性がある．（２）は，ピボットのチェツクを行うものであるため連続して使われることはないが，（３）～（５）は連立一次方程式の解法のアルゴリズムを考えると，同じような計算が連続しておこなわれることが多いと思われる．同じような計算が連続していれば，高速化ができるはずである．今回は出現頻度調査の結果から効果が大きいことが分かった（３）と（４）についコードの最適化を行った．

（３）においては，同じ値による除算が続く場合，逆数を予め計算しておいて，その値を掛けることにより，除算を乗算で置き換え高速化した．この変更は計算機によっては結果が異なることがあるが，ＣＲＡＹは，もともと，除算器をもたず，逆数計算器しかもつていないため，上記の変更を行っても，まったく結果は変わらない．

連立一次方程式の解法で，（４）は内積型の計算であり，

```
x = x - y * z
```

において，ｙとｚは毎回異なるが，ｘは同じものである．この場合，ｘをレジスタに置くと，中間のロード，ストアは不要となり，最初と最後だけに行えばよい．中間のロード，ストアをやめることにより，メモリへのアクセスを半分にすることが可能になる．スーパー・コンピュータのように演算器とメモリの速度差が大きい計算機ではその効果が大である．さらに，中間のロード・ストアを行っていた命令が必要でなくなるため，命令を解釈する時間がいらなくなり，命令を入れておくメモリも不要となり，一石二鳥である．

以上の結果，実行速度は，表１のようになつた．

項目	ＦＯＲＴＲＡＮ版	コード生成版
行列分解	１００６．１０６(1)	１４．７０７(3)
求解	２９．３４０(2)	０．０２２(4)
コードの実行時間	１０２０．７１７(5)	３８．７６７
比率	２６．３倍	１．０倍

表１．　重要な問題のおける実行時間の比較（単位：秒）

(5)=(1)+(2)-(3)-(4)

このように，ＣＲＡＹ　Ｘ－ＭＰのスカラー版のコード生成でも，ＦＯＲＴＲＡＮ版に比べて，約２６倍高速化されている．この理由は，次のように考えられる．

［１］生成されたコードには，ＦＯＲＴＲＡＮのようなループを制御する命令がなく，その処理の必要性がない分だけ速い．

［２］ＣＲＡＹのようなスーパー・コンピュータでは，分岐することが不得意であるが，生成されたコードには分岐がなく，命令の先取りがうまく働き，速くなる．

［３］ＦＯＲＴＲＡＮ版は，解くべきデータの構造がわからない段階でコーディングされているため，どのようなデータにも対応するように，すべてのデータへのアクセスが間接番地とならざるをえない．これに対して，コード生成においては解くべき問題（モデル）が確定し，データの構造が分かってから，そのデータ向きの解法ルーチンを作っているため，データを直接アクセスできることにより速くなっている．

４．まとめ

ここで示した例のように，問題の性質をうまく利用した解法が，大規模問題においては大切である．３．４節の［１］と［２］の効果がそれぞれどの程度かは分離できなかったが，その性質から，［１］と［２］の効果は２～３倍程度と考えられる．したがって，最も効果が大きいのは［３］のデータの構造が分かって

から解法ルーチンを作っていることであると考えられる．したがって，現在の回路解析ソフトでは実現困難であるが，生成されたコードとして機械語ではなくＦＯＲＴＲＡＮを作り，それをコンパイル・リンクし，実行することができれば，それでも大幅な効果があると思われる．このようなことが簡単に行えることが望ましいであろう．

この他の回路解析コードの高速化の手段としては，我々は，回路行列の作成部分にマルチタスク化も実現している．経過時間で１．６倍程度の高速化を行っている．今後は，ｇａｔｅｒ／ｓｃａｔｔｅｒの機能を利用して，回路解析のベクトル化を図っていく予定である．さらに，その後，ベクトル版コード生成やベクトル化版のマルチ・タスクも考慮している．

参考文献

［１］福井，大吉，加藤，渡辺：マルチプロセッサ・システムにおける回路解析コードの並列処理，情報処理学会数値解析研究会資料１７－５，（１９８６）

［２］室田一雄：組合せ理論と大規模数値計算－方程式系のブロック三角化，日本ＯＲ学会第１５回シンポジウム，（１９８６）

［３］大沢，後藤：回路シミュレーション・コードの自動生成，ｂｉｔ別冊「計算機による数式処理のすすめ」ｐｐ．１３２－１３９，（１９８６）

［４］George,A & Liu,J:Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, (1981)

［５］福井，隅田：数式処理を利用した多関節ロボットの手先位置の計算，ＲＥＤＵＣＥプログラミング資料第三集ｐｐ．２２０－２２３，東京大学大型計算機センター，（１９８６）

［６］福井，隈本：数式処理を利用した回生車のシミュレーション，ＲＥＤＵＣＥプログラミング資料第三集ｐｐ．２２４－２３２，東京大学大型計算機センター，（１９８６）

［７］福井，佐々木，鈴木，佐藤：数値・数式融合計算のためのＦＯＲＴＲＡＮとＲＥＤＵＣＥの一結合方式，日本ソフトウェア科学会第３回大会論文集，Ｄ－６－２，（１９８６）

［８］村田，小国，唐木：スーパーコンピュータ，丸善，ｐｐ．１８０－１８５，（１９８５）

［９］池田，大月：大規模ランダムスパース行列演算プログラムライブラリー開発の試み，京都大学数理科学講究緑３１０，ｐｐ．１－１８，（１９７７）