オペーレーティング・システム　54－5
（１９９２．　３．　１３）

線型方程式の高速化技術

中西　誠　　三上　次郎

富士通株式会社

線型方程式の数値解法について，ベクトル計算機および汎用計算機での高速化技術を報告する．ベクトル計算機向けの基本的な高速化技術と，各数値解法への適応性を評価・分析した結果，２重アンローリングを伴う外積型のガウスの消去法を，演算パイプラインの能力を最大限活用する観点から，Ｎ重アンローリングに一般化した「ブロッキングＬＵ分解法」を開発した．この解法は，複数組の複合命令（積と和）を同時に処理するベクトル計算機に適用すると，一般にメモリアクセスパイプライン間のバンク競合を起こすが，これを回避する技術についても報告する．更に，この解法は，キャッシュメモリを持つ汎用計算機での高速化技術としても有効であることを報告する

Techniques to exploit high performance of linear equation solver

Makoto Nakanishi　Jiro Mikami

FUJITSU LIMITED

140 Miyamoto, Numazu-shi, Shizuoka, 410-03, JAPAN

This report describes the techniques to exploit the performance of linear equation solver on both vector supercomputer and general purpose computer. To pack instructions in the operation-pipes closely, we developed the N order unrolling version of outer product type Gaussian elimination, that is blocking LU decomposition, which is generalized version of double unrolling one. We also developed the methods to avoid the bank conflict caused by two load/store pipes.

And we report this method is also suitable for general purpose computer with cash mechanism, as the data on the cash is effectively reused.

1．はじめに

線型方程式の解法の性能を引き出す上ためには，ハードウェア特性を考慮することが必要である．

ベクトル計算機での高性能は，十分長いベクトル長で，演算パイプラインが蜜に動作するとき達成され，スーパーベクトル性能と呼ばれる．

このために，ベクトル長をできるだけ長くし，ムダなロードストアを減らし，演算パイプラインをとぎれなく動作させる観点からのアプローチが必要である．

また，ベクトル計算機のメモリは幾つかのバンクにメモリを分けて構成されているため，同一のバンクに対するアクセスは性能の劣化を引き起こす．

これを回避するアクセス方法を考える必要がある．

この２つの観点から，外積型のガウスの消去法を２重アンローリングした版を一般化したブロッキングＬＵ分解法を開発し，上記の観点からチューニングを行った．

一方，汎用計算機で性能を引き出すためには，キャキシュとメモリの構造を意識して，キャッシュ上のデータを有効に利用することと，浮動小数点演算を効率的に行うため，有限の浮動小数点レジスタを有効に利用することが重要である．

ベクトル計算機向きに開発したブロッキングＬＵ分解はこれらの特性を考慮して，汎用機上の性能を引き出す上でも有効な方法であることを報告する．

2．線型方程式の解法

線型方程式の直接解法には以下の３種類があり，これらの方法について，ベクトル計算機向けのチューニングをおこなった場合，外積型のガウスの消去法がもっとも高速である．[1],[2]

①外積型のガウスの消去法
②内積型のガウスの消去法
③クラウト法

2．1　ベクトル計算機向けの基本的な高速化技術

最初に，以上の方法をベクトル計算機でチューニングする上で使われた技術をまとめ，各種解法での適用性を評価・分析する．

2．1．1　ループアンローリング

ベクトル計算機で性能を引き出す上で最も重要な技術はループアンローリングである．

この技術は，ループをアンロールすることで，演算パイプラインを蜜に動作させる作用がある．

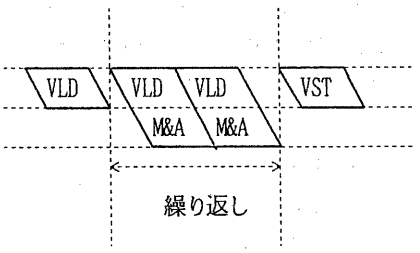
行列の積を計算するとき，列ベクトルに注目して，計算を行う．このときループは３重ループになり，最内ループはベクトルに対応する．２番目のループをアンロールすることを考える（図１）．

更新部分が，ベクトルレジスタに保持されると考えると，ロード命令と演算命令がチェインし更新部分をベクトルレジスタにロード・ストアする部分が無視できるときスーパーベクトル性能を達成する（図２）．

富士通のＶＰ２０００シリーズには，スカラとベクトルの積とその中間結果ベクトルとベクトルの加算を１命令で行うＭｕｌｔ＆Ａｄｄ命令がある．

```
      DIMENSION X(N),A(N,N),B(N,N),C(N,N)
        DO 100 I=1,N
        DO 200 K=1,N
        X(K)=0.
  200 CONTINUE
        DO 300 J=1,N,2
        DO 300 K=1,N
        X(K)=X(K)+A(K,J  )*B(J  ,I)
       *          +A(K,J+1)*B(J+1,I)
  300 CONTINUE
        DO 400 K=1,N
        C(K,I)=X(K)
  400 CONTINUE
  100 CONTINUE
```

図１　行列積のプログラム



上段：ロードストアパイプ
下段：演算パイプ

図2　ループのタイムチャート

2．1．2　行方向のロード・ストアの回避

ＦＯＲＴＲＡＮの文法では，２次元行列の列方向のアクセスはメモリ上の連続アクセスとなり，行方向のアクセスは一定間隔の要素をアクセスすることとなる．

行方向のロード・ストアの性能は一般的に列方向のロード・ストアに比べて悪い．

また，一定間隔の要素が同じバンクに格納されている場合，バンクコンフリクトを発生する可能性がある．

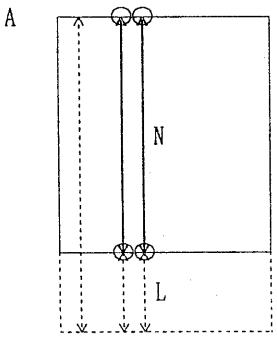
2．1．3　2本のロードストアパイプに起因するバンクコンフリクト

富士通のＶＰ２０００シリーズの上位機種では，ロード・ストアパイプラインと，演算パイプライン（Ｍｕｌｔ＆Ａｄｄパイプ）が2本づつある．

一般に，２本のロード・ストアパイプを同時に動作したとき，同じバンクにあるメモリ要素をアクセスすると，バンクコンフリクトを生じる．この現象は，Ｍｕｌｔ＆Ａｄｄ複合命令を導入した結果，ロード・ストアパイプラインが蜜となり顕在化した．連続な要素をアクセスするとき，このバンクコンフリクトを避けるため，一方のロードもしくはストアの開始をバンクコンフリクトが起こらないようになるまで遅らせる．

つまり2本のロード・ストアパイプに起因するバンクコンフリクトは，連続な領域に対するロード・ストアパイプラインの動作が次の命令で切り替わる部分で生じる．

行列積の計算で，これを回避する手段として，宣言する２次元配列の１次元目（リーディングディメンジョン）を調整することで回避できる．つまり，行列の列ベクトルの先頭要素が違うバンクに入るように，リーディングディメンジョンを調節することで対処できる．



DIMENSION A(N+L,N)

図3　リーディングディメンジョンによるバンクコンフリクトの回避

ただし，おなじ領域で，行列の大きさが変化する行列積の場合は，リーディングディメンジョンを調節しただけでは，この現象をすべて避けることはできない．この現象がブロッキングＬＵ分解法では起こる．

2．2　線型方程式の解法のベクトル計算機への適合度の評価

線型方程式の解法を上記の技術の適用という観点から，クラウト法と外積形のガウスの消去法について考察してみる．

2．2．1　クラウト法

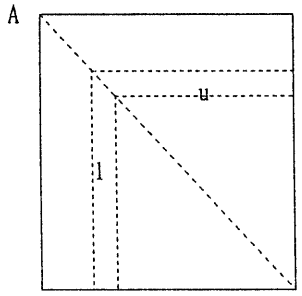
クラウト法は，更新部分で，内積計算を行う．この計算を精度よく行うことで，解の精度を高めることができるため，富士通の科学用サブルーチンライブラリＳＳＬⅡで提供してきた．

ベクトル計算向きにチューニングを行うのは，この

内積部分であるが，更新は，列ベクトルに対する更新と行ベクトルに対する更新が交互にかかる．（図4）

ただし，行ベクトルの更新で，行ベクトル参照を行うため，性能は落ちる．

また，更新が進むにつれて，参照するベクトルの本数は増すが，ベクトル長は短くなる．



$A=(a_{ij})$, $L=(l_{ij})$, $U=(u_{ij})$

$$u_{ij}=(a_{ij}-\sum_{k=1}^{i-1} l_{ik}u_{kj})/l_{ii},\quad i=1,..,j-1$$

$$l_{ij}=a_{ij}-\sum_{k=1}^{i-1} l_{ik}u_{kj}\quad i=j,..,n$$

図4　クラウト法

このため，ＳＳＬⅡ／ＶＰ（ベクトル計算機向けＳＳＬⅡ版）の従来のクラウト法では，終盤の更新は，内積形式に変換することで，ベクトル長を確保する方法を採用している．[3]

2．2．2　外積型のガウスの消去法

外積型のガウスの消去法は，ピボットをとり，更新を列ベクトルと行ベクトルの外積で計算を行う方法である．このままだと，ロード・ストアと演算量のバランスが悪く，性能が十分でない問題があった．

ベクトル計算機向きの外積型のガウスの消去法は，ピボットを2本，更新に先立ち決定し，2本の列ベクトルと行ベクトルによる外積更新を行うことにより演算量を増やすことで，このロード・ストアと演算量をバランスさせる．

つまり，外積計算による更新部分に，ループアンローリングの技法を適用し，また共通に参照される部分をベクトルレジスタに保持する．

```
      DO 100 L=3,N,2
      DO 200 I=L,N,2
      DO 300 K=L,N
      A(K,I  )=A(K,I  )-A(K,L-2)*A(L-2,I)
     *                 -A(K,L-1)*A(L-1,I)
  300 CONTINUE
  200 CONTINUE
  100 CONTINUE
```

下線の付いたベクトルをレジスタに保持

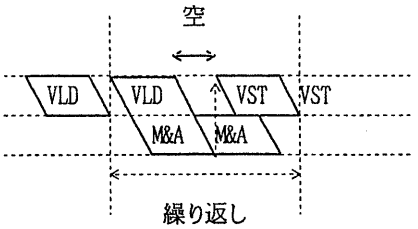


図5　2重アンローリングした更新部分のプログラムとタイムチャート

図5の下線の付いたベクトルをレジスタに保持しない場合

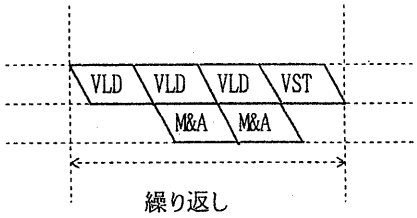


図6　行列積として計算したタイムチャート

しかし，このタイムチャート（図5）から分かるように，繰り返し部分で，ロードとストアが，演算パイプをはさんでチェインニングしているため，みかけ上は，3段のチェインニングに見え，2段のチェインニ

ングに比べ性能が劣化する.

一方，最外ループにアンローリングを適用しなければ，2本の演算パイプは並列動作するが，ロード・ストアが，1チャイムチェイニングしないため，2倍の処理時間となってしまう.

2重アンローリングした外積型のガウスの消去法は，ベクトル計算機に十分適合した良い方法であるが，まだ改良の余地がある.

2．3　n重アンローリングに一般化した外積型のガウスの消去法

前節で，クラウト法に比べ外積型のガウスの消去法が改良の余地があることが分かった.

この改良方法を述べる．2重アンローリングした外積型のガウスの消去法の更新部分は，幅が2の行列積の計算になっている.

行列積に関する，ベクトル計算機での高速技法はよく知られた，外積型の方法があり，ほぼベクトル計算機のピーク性能を達成することができる.

この技法は，更新するべき列ベクトルをベクトルレジスタに保持して，3重ループの中間のループをアンローリングする技術である.

この結果，ロード・ストアパイプト演算パイプの2段がチェイニングして，スーパーベクトル性能を達成することができる.

固定大きさの行列に対しては，配列のリーディングディメンジョンを調整することで，バンクコンフリクトを回避できる.

つまり，2重アンローリングした外積型のガウスの消去法を一般化し，n重アンローリングした外積型のガウスの消去法を考えると，更新部分は，幅nの行列の積となり，上記の技術を適用できる.

2重アンローリングした外積型のガウスの消去法の評価部分で問題となった，みかけ上の3段パイプとなり，空が生じる問題は，m×nとn×mの行列の積でnが大きくなると，一般の行列積の計算の方が有利となる.

われわれは，この方法を，n重アンローリングして処理することが，幅nのブロックに分割してＬＵ分解を行うことと同一であることより，ブロッキングＬＵ分解法と呼ぶ.

2．3．1　数学的フレーム

n重アンローリングした，外積型のガウスの消去法は，幅nの部分をＬＵ分解した後，行列積でのもう一つの行列を，行列に関する三角方程式を解き決定する．これにより求まった，行列の積で，残りの部分を更新する．これは，Ｄｕｆｆ等によって提案されている方法と数学的には同一のフレームである．[4]

Ｄｕｆｆ等は，この方法で共有メモリを結合した並列計算機での性能引き出しに有効であることも示している.

つまり，k番目の処理で，ブロッキングＬＵ分解法は以下のように計算する.

ブロック化した外積型のガウスの消去法を行うときマトリックスを$L_2^{(k)}$，行マトリックスを$U_2^{(k)}$，更新部$M^{(k)}$とする．配置は図7のとおり.

更新部分はを次の式で行う.

$$M^{(k)} = M^{(k)} - L_2^{(k)} \cdot U_2^{(k)}$$

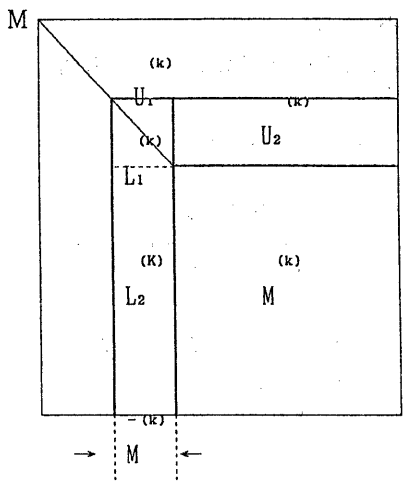


図7　ブロック化した配列Mの各要素の配置関係

(k) (k)

これに先立ち，L_2，U_2 は以下の式で更新する．

$\overline{\quad}^{(k)}$ $(k)T$ $(k)T$ T (k)

まず，M を (L_1, L_2) と U_1 に分解して，更新する．

(k) $(k)-1$ (k)

$U_2 = (L_1) \cdot U_2$

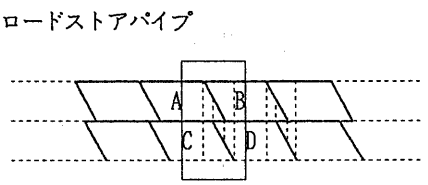
2．3．2 ブロッキングＬＵ分解法で特徴的な問題とその解決方法

2．3の説明で，ブロッキングＬＵ分解法がベクトル計算機に対して，ロード・ストアパイプラインと演算パイプラインのバランス上非常に適合していることが分かった．

次に，ブロッキングＬＵ分解法で問題となる，2本のロード・ストアパイプラインに起因するバンクコンフリクトとその回避方法を述べる．

2本のロード・ストアパイプラインに起因するバンクコンフリクトは，行列積で使われるロード・ストアパイプラインが蜜に詰まっているとき，連続なアクセスの切り替わり分，つまりロードの切り替わり部分で発生することを前述した．大きさ固定の配列の場合は，リーディングディメンジョンを調整することで，回避出来たが，ブロッキングＬＵ分解法の場合は，完全には回避できない．

これは，ＬＵ分解を幅ｎのブロックに対応して進める



① 同一パイプでの終了部と開始部
　AとB，CとD
② 異パイプ間での終了部と開始部
　AとD，BとC
③ 異パイプ間での開始部同士

図8　2本のパイプ間のバンクコンフリクト

る過程で，更新するべき行列部分の大きさが変化することにより，切り替わり部分でバンクコンフリクトが発生するためである．

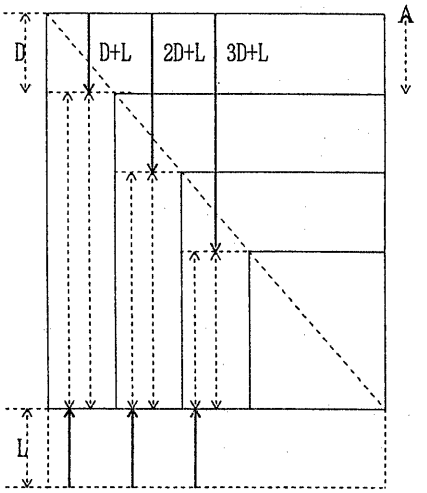


図9　更新部分の大きさの変化によるバンクアクセスの変化

これを回避するためには，行列積の計算部分を複数列ベクトルを同時に更新するようにして，参照する列ベクトルを共通に利用するようなコーディングを行う．この結果，ロード時に，バンクコンフリクトが生じたとき，それを避けるために遅れて動作したロードが待たされるが，この遅れの影響は演算パイプラインが1本のロードに対して複数個詰まっているため，これを参照している演算パイプラインの遅れにならない．

2．3．3 ブロック幅とベクトル長

ベクトル計算機では，性能を引き出すもう一つの観点は，できるだけベクトル長を長くすることである．

ブロッキングＬＵ分解法は，ロード・ストアパイプラインと演算パイプラインのバランスと，パイプラインを蜜に詰める上で適した解法である．ただしブロック幅を大きくしていくと，更新部分での行列計算でのベクトル長は小さくなる．

```
      DIMENSION A(N,N),B(N,N),C(N,N)
      DIMENSION W1(N),W2(N),W3(N),W4(N)
      ....
      DO 100 K=1,N,4
      DO 200 I=1,N
      W1(I)=A(I,K  )
      W2(I)=A(I,K+1)
      W3(I)=A(I,K+2)
      W4(I)=A(I,K+3)
  200 CONTINUE
      DO 300 J=1,N,2
      DO 400 I=1,N
      W1(I)=W1(I)-B(I,J  )*C(J  ,K  )
                 -B(I,J+1)*C(J+1,K  )
      W2(I)=W2(I)-B(I,J  )*C(J  ,K+1)
                 -B(I,J+1)*C(J+1,K+1)
      W3(I)=W3(I)-B(I,J  )*C(J  ,K+2)
                 -B(I,J+1)*C(J+1,K+2)
      W4(I)=W4(I)-B(I,J  )*C(J  ,K+3)
                 -B(I,J+1)*C(J+1,K+3)
  400 CONTINUE
  300 CONTINUE
      DO 500 I=1,N
      W1(I)=A(I,K  )
      W2(I)=A(I,K+1)
      W3(I)=A(I,K+2)
      W4(I)=A(I,K+3)
  500 CONTINUE
  100 CONTINUE
      ....
```

図１０複数列同時更新の行列積計算プログラム

ロードバランスの観点からブロック幅を大きくする分ベクトル長が短くなってしまう．この２つの性能因子は拮抗する因子であり，適当なブロック幅で，性能はピークを達成する．

３．汎用計算機での線型方程式の高性能化

汎用計算機で線型方程式の性能を引き出す上で必要な考慮点は，キャッシュを備えたメモリ系の構造から，できるだけキャッシュに保持されたデータを再利用することでメモリとキャッシュの間の転送を減らすことである．

また，浮動小数点レジスタに保持したデータをなるべく有効に利用することである．

３．１　メモリーとキャッシュ

ここでは，富士通の大型計算機Ｍ７８０のキャッシュの機構を例にキャッシュの効率的な使用法について述べる．[5]

Ｍ７８０では，命令用とデータ用にそれぞれ６４キロバイトのキャッシュがある．

各々は，６４バイトのブロックと呼ばれる単位から構成されていて，６４ライン×１６ウエイ，つまり１０２４個のブロックからなっている．

同一ラインのメモリはアドレスの適当なオフセットを持つもの，つまり下位ビットを無視すると同じアドレスとなるもので，最大１６個まで，キャッシュとして保持できるという意味である．この構造から次のような使い方が，性能を引き出す上で留意するべき点である．

① ６４キロバイトにできるだけ近い６４キロバイトより小さいメモリを繰り返し使用する．
② 同じオフセットを持つものの多重度を１６以内に抑える．
③ アクセスは連続方向に行う．

一般に，６４Ｋに一杯に載せようとすると，オフセットが重なってしまい，ミスヒットを起す確率が高まる．但し，連続アクセスを行っている限り，８バイトの１回のアクセスで６４バイト分読み込むため，残りの５６バイトはキャッシュ上にあり性能劣化はそれほど目立たない．ただし行方向にアクセスすると，最悪毎回メモリをアクセスすることとなり，性能が著しく劣化する場合がある．

３．２　線型方程式の解法とメモリアクセスパターン

キャッシュを効率的に利用する以上の観点から，線

型方程式のメモリアクセスパターンを比較してみる.

3．2．1 内積形のアクセスパターン

内積形のガウスの消去法では，列ベクトルのアクセスは共通になるが，行ベクトルのアクセスは，ある矩形の領域に渡る．一つの列ベクトルを決定するために，この領域を1回アクセスする．このため，データはキャッシュに保持されることはなく，有効に利用できない．このことは，クラウト法についても同様である.

3．2．2 外積型のガウスの消去法のアクセスパターン

外積形のガウスの消去法では，行ベクトルと列ベクトルの外積で更新を行うため，この2つのベクトルをキャッシュに保持し，更新を列ベクトルに対して行う．ただし，更新ごとに，1回更新部分全体をアクセスする必要がある.

3．2．3 ブロッキングＬＵ分解法のアクセスパターン

ブロッキングＬＵ分解法のアクセスパターンは，行列積のアクセスパターンである.

これは，更新部分の列ベクトルの計算を，行列と列ベクトルの積の繰り返しで行うとすれば，更新する列ベクトルに対して参照する行列は共通となる.

この部分を，キャッシュに保持すれば，効率のよいアクセスができる．また，参照する行列を，行方向にに分割することで，行列の大きさが大きくなっても，参照部分がキャッシュに保持されるようにする.

3．2．3．1 ブロッキングＬＵ分解法の行列積の計算方法

行列積の計算は，大きく内積型のものと外積型の2種類に分かれる.

内積型の計算を前述の行列と列ベクトルの計算でおこなうと，行列の参照パターンが行方向となり，連続アクセスではなくなり性能劣化を引き起こすことがある.

これに比較し，外積型の計算では，行列と列ベクトルの積の計算は，連続アクセスでキャッシュ有効利用の3つの指針を満たし，安定である.

次に，汎用浮動小数点レジスタになるべく効率的に保持して，計算することを考える．一般に，行列の大きさが大きくなるので，列ベクトルに対して共通な要素をレジスタに保持した方が効率がよい．各列ベクトルの更新を行うとき，共通に参照される要素があるため，共通に参照できるように，複数列ベクトル更新することとした.

これにより効率のよいプログラムとなっている.

4．おわりに

外積型のガウスの消去法をn重アンローリングしたブロッキングＬＵ分解法は，ベクトル計算機だけでなく，汎用計算機で性能を引き出す上でも有効である.

参考文献

[1] Dongarra, J.J., Gustavson, F.G., Karp, A., "Implementing Linear Algebra Algorithm for Dense Matrces on a Vector Pipeline Machine", SIAM Review, Vol.26, 1984, pp.91-112

[2] 島崎眞昭，スーパーコンピュータとプログラミング，共立出版株式会社，1989

[3] 伊奈博，三上次郎，秋田典伸，山下真一郎，"ベクトル計算機における連立1次方程式の解法"，情処全大 第27回

[4] P.AMESTOY, M.DAYDE and I.DUFF, "Use of computational kernels in the solution of full and sparse linear equations", M.COSNARD, Y.ROBERT, QUINTON and M.RAYNAL, PARALLEL & DISTRIBUTED ALGORITHMS, North-Holland, 1989

[5] 槌本 隆光，清水 和之，高村 守幸，篠原 誠宮沢 達士，利根 廣貞，"ＣＰＵを1ボードに実装した大型コンピュータＭ７８０ 高速２５６ＫスタチックＲＡＭを主記憶に用い2階層記を採用"，日経エレクトロニクス，１９８６年6月2日号 No.396, pp.179-209