ハイパフォーマンス
コンピューティング 82－8
（２０００．８．３）

非構造メッシュ用 Block ILU 前処理付き反復法のベクトル化手法

丸山 訓英* 襲田 勉* 鷲尾 巧* 土肥 俊* 山田 進♭
*日本電気株式会社 ♭日本原子力研究所

概要

本論文では、偏微分方程式系を不規則メッシュ上で離散化することによって生じる大規模連立一次方程式をベクトル計算機上で高速に解くときのアルゴリズムの実装方法について述べる。解法としては、同一節点上に定義された未知数を一つのブロックとしてILU 前処理を行う Block ILU 前処理 (以下 BILU 前処理) 付き反復法を用いる。この BILU 前処理のベクトル化のためのデータ形式として、独立集合 (independent set) ごとに JAD 形式を適用する IDS-JAD 形式を開発した。このデータ形式を用いたベクトル化にあたっては、ブロックサイズ可変な問題に対応可能な汎用性のある手法を開発したが、間接メモリ参照の負荷が大きく演算性能を充分発揮することができなかった。そこで今回新たに、ブロックサイズ一定問題においてはメモリへの負荷を低減する最適化が可能であることに着目し、ブロックサイズ一定問題用に最適化された BILU 前処理演算を実装した。この実装により、例題として 3 次元構造解析問題（自由度約 100 万）を使用し、収束判定条件を相対残差ノルムが 10^{-8}以下とした場合の NEC の SX-5A(1CPU, ピーク性能 8Gflops) 上での数値実験において、Multicolor オーダリングの場合、CPU 時間は 51 秒（反復回数 455)、演算性能は 3Gflops、Reverse Cuthill-Mckee オーダリングの場合、CPU 時間は 30 秒（反復回数 223)、演算性能は 2.5Gflops という結果を得た。

Vectorization techniques of block ILU preconditioning for unstructural problems

Kunihide Maruyama * Tsutomu Osoda * Takumi Washio * Sun Doi * Susumu Yamada ♭
* NEC ♭ JAERI

Abstract

In this report, large sparse linear systems which stem from discretization of a system of partial differential equations on an unstructured mesh are dealt with on vector machines. In the previous authors report of this meeting, the IDS-JAD format was proposed to vectorize block ILU preconditioning, where the well known JAD format is applied to each of independent sets for the substitutions. Here, a block corresponds to unknowns on one node in the mesh. In this implementation, the preconditioner was applicable also to variable block size problems. However, it was not optimal for the constant block size problems, which are more common than the former case. In this report, an optimal vectorization technique of the BILU preconditioning for the constant block size problems is introduced and evaluated in comparison with the former version. With the new optimal version, we achieved the following results on an NEC SX-5 machine (8Gflops peak) for a three dimensional structural problem. With Multicolor ordering the CPU time was 50seconds and the sustained performance was 3Gflops, with Reverse Cuthill-Mckee ordering the CPU time was 30seconds and the sustained performance was 2.5Gflops.

1 はじめに

物理現象のシミュレーションでは、偏微分方程式を離散化し、そこから生じる大規模連立一次方程式を解く。離散化手法には例えば有限要素法があり、そこでは不規則メッシュが用いられる。この離散化の結果生

じる連立一次方程式は、不規則スパース行列を係数行列に持つ。そのような大規模連立一次方程式の求解では、クリロフ部分空間法に前処理を付加し解の収束を加速させる"前処理付き反復法"がよく用いられる。具体的には、解くべき連立1次方程式を

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

とする時、M を行列 A を近似するように構成し、例えば式 (1) の両辺に M の逆行列を左からかけて

$$M^{-1}Ax = M^{-1}B \tag{2}$$

と変形する。そして、式 (2) を $M^{-1}A$ を係数行列に持つ連立1次方程式としてクリロフ部分空間法を適用する手法である。ここで、行列 M を前処理行列と言う。前処理としては、通常 Incomplete LU(以後 ILU) 前処理が用いられる。この前処理手法をメッシュの各節点上に複数の変数が存在する問題に適用する方法として、Block ILU（以下 BILU）前処理がある。これは、成分単位で ILU 分解を行うのではなく、メッシュの一節点上の未知数をひとまとめにした Block という集合を一つの単位として ILU 分解を行うことでより効率良い前処理を実現する手法である。BILU 前処理では、前処理行列 M として以下のような BILU 分解行列を用いる。

$$M = (L + D)D^{-1}(D + U). \tag{3}$$

ここで、L, D, U はそれぞれ、ブロック単位の下三角、対角、上三角行列で次の条件を満たすものとする。なお、非ゼロブロックパターン PB として、通常は行列 A のブロック単位非ゼロパターンを用いる。

$$M = A \quad \text{on} \quad PB, \tag{4}$$

$$L = U = 0 \quad \text{outside} \quad PB. \tag{5}$$

さて、BILU 前処理付き反復法のベクトル計算機上での実装では、行列ベクトル積、BILU 前処理演算のベクトル化が重要となる。行列ベクトル積については Jagged Diagonal(以後 JAD) 形式 [2] による行列データ格納方法を用いた効率良いベクトル化手法がある。一方、BILU 前処理演算は、前進後退代入演算により構成され、そこではデータの参照関係があるため行列ベクトル積と同様のベクトル化手法が適用できない。特に不規則構造問題では効率良いベクトル化が難しいとされてきた。

そこで我々は BILU 前処理の前進後退代入演算において、独立集合 (Independent set)[2] ごとに JAD 形式を適用し、前進後退代入演算の効率良いベクトル化を実現させるため IDS-JAD 形式を開発した。ここで、独立集合とは、前進後退代入において、同時に計算可能なブロックにより構成される集合である。この IDS-JAD 形式を用いたベクトル化手法については、第 79 回 HPC 研究会で発表した [1]。その際、ソルバの汎用性を重視し、1節点あたりの未知数の数が変化する問題にも対応可能な汎用性のあるベクトル化手法を開発し発表した。しかし、後に述べるように充分な演算性能を発揮することができなかった。

そこで、今回新たに多くのアプリケーションにおいては、未知数の数が節点により変化しないことを考慮し、ブロックサイズ一定の場合の最適化版 BILU 前処理演算を実装し、ソルバの演算性能を向上させた。

本論文では、これら二つのベクトル化手法について説明し、その性能比較をベクトル型スーパーコンピュータ SX-5A-model 上で行った結果を示す（第 4節)。

2 BILU 前処理の並列性

本節では、BILU 前処理の並列性について述べる。

BILU 前処理をクリロフ部分空間法に適用した場合、与えられたベクトル r に対して

$$Mp = r \tag{6}$$

を p について解く計算が毎回の反復で必要となる。式 (6) は、前処理行列 M の定義式 (3) より

$$(L + D)D^{-1}(D + U)p = r$$

となる。これを解くには、まず、$q = D^{-1}(D+U)p$ とおき

$$(L+D)q = r \tag{7}$$

を q について解く。これを前進代入と呼ぶ。そして、$D^{-1}(D+U)p = q$ を p について解く、これを後退代入と呼ぶ。

以後前進代入について述べるが、後退代入についても同様である。式 (7) を変形すると $q = D^{-1}(r - Lq)$ となる。ここで、L は下三角ブロック行列なので q の k 番目のブロックを計算するためには、右辺の q は $k-1$ 番目のブロックまでが求められていればよい。このように番号の若いブロックから順次 q を求めていくことができる。

以上述べたように前進代入計算は、若い番号から順に計算するので、基本的には並列に計算することができない。しかし、L が疎行列の場合、L のブロック非ゼロパターンよりデータの参照関係を解析し、同時に計算可能なブロックを抽出し、前進代入演算を並列に実行できる。この同時に計算可能なブロックにより構成される集合を独立集合と呼ぶ [2]。そして、独立集合を構成した後、1番目の独立集合、2番目の独立集合、… 最後の独立集合の順にブロックを並べ替える。このようにブロックを並べ替えても非ゼロブロック小行列でつながっているブロック間の前後関係は保たれるので前進代入演算の計算結果は変わらない。図 2

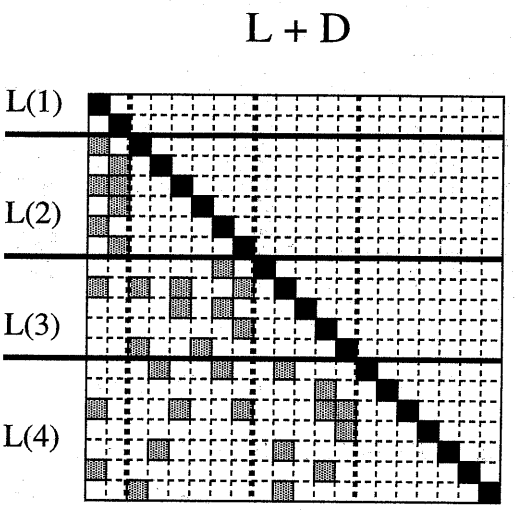

図 1: 独立集合順にブロックを並べた後の $L+D$ の非ゼロブロックパターン

は、独立集合を構成しブロックを並べ替えた後の行列の非ゼロブロックパターンの例を示している。太い実線は、行方向の独立集合の区切り、太い点線は列方向の独立集合の区切りを示す。図 2のようにブロックを並べ替えた後の前進代入演算のアルゴリズムは次のようになる。ここで、$L(il), D(il)$ はそれぞれブロック下三角、ブロック対角行列を il 番目の独立集合の行方向に制限したものである。また、$nids$ は独立集合の総数である。

ブロックを独立集合ごとに並べ替えた後の前進代入アルゴリズム

for $il = 1, ..., nids$
  $q := D(il)^{-1}(r - L(il)q)$
end

上のアルゴリズムから分かるように一つの独立集合内の前進代入演算は行列 $L(il)$ とベクトル q の積と行列 $D(il)^{-1}$ とベクトル $r - L(il)q$ の積の二つの行列ベクトル積演算より構成される。これより、行列ベクトル積の効率良いベクトル化手法として知られるデータ構造である JAD 形式を各 $L(il), D(il)^{-1}$ に適用することが可能となる。

このように、独立集合ごとに JAD 形式を適用する格納形式を IDS-JAD 形式と呼ぶことにする。

3 BILU 前処理のベクトル化手法

本節では前節で述べた BILU 前処理の並列性を使った前進後退代入演算のベクトル化手法について述べる。

前節では、独立集合内の演算は行列ベクトル積と同様になるため、行列ベクトル積の効率良いベクトル化手法として知られる JAD 形式を独立集合ごとに適用することが可能であると述べた。これより、BILU 前処理の効率良いベクトル化を実現するためには、JAD 形式による行列ベクトル積の効率良いベクトル化手法を開発すればよいことになる。本節では、ブロック構造を持つ不規則疎行列に関して JAD 形式による行列ベクトル積のベクトル化手法について二つの方法を述べる。まずその前に、JAD 形式について説明する。

JAD 形式とは、行列の各行の非ゼロ成分を左に詰め、各行の非ゼロ成分の数の大い順に行を並べ替え、縦方向に行列の非ゼロ要素と対応する列番号をそれぞれ 1 次元配列に格納する方式である。例として図 2に JAD 形

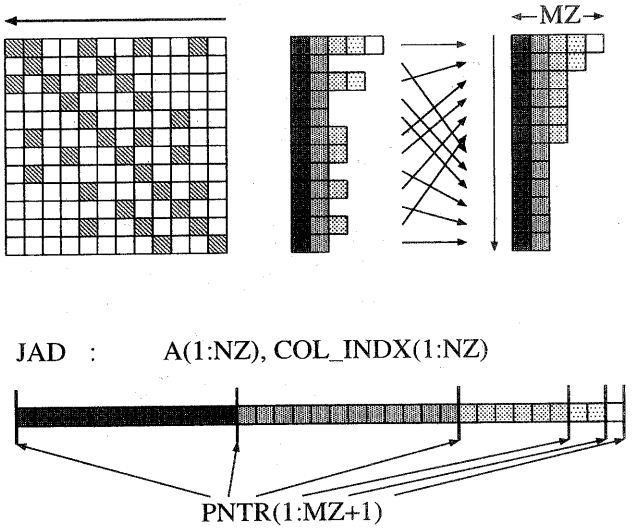

図 2: JAD 形式

式で格納する時に用いる配列を示す。NZ は、行列 A の非ゼロ成分の総数、MZ は各行の非ゼロ成分の個数の最大数である。A(1:NZ) に非ゼロ要素、COL_INDX(1:NZ) に対応する列番号を記憶する。PNTR(1:MZ+1) は図 2に示すようにデータを縦方向に記憶していく時の列の切れ目を示すポインタである。例えば、図 2の上段右端の一番左の縦の列に含まれる非ゼロ成分の列番号は、COL_INDX(PNTR(1))···COL_INDX(PNTR(2))-1 に格納される。

行列が、ブロック構造を持ちブロックサイズが一定な場合には、ブロック単位で JAD 形式を適用すればよい。つまり、図 2の小正方形をブロック小行列に対応させる方法である。すなわちブロックサイズを bsize とするとき次のように非ゼロ要素とベクトルを格納する配列を用意する。行列の非ゼロ要素は、各ブロック小行列内の行と列のインデックスを加え 3 次元配列 A(1:NZ,1:bsize,1:bsize) に格納する。ここで NZ は非ゼ

ロブロック小行列の総数である。ベクトルvは、第2インデックスにブロック内のインデックスを加え2次元配列v(1:N,1:bsize)に格納する。ここでNはベクトル内のブロックの総数である。

例えばbsize=3の場合、上記のようなデータ構造のもと行列ベクトル積$y = Ax$のプログラムは次のように記述できる。

JAD形式をブロック単位で適用した場合の行列ベクトル積（bsize=3）

```
do i=1,MZ+1
   do j=PNTR(i),PNTR(i+1)-1
      col=COL_INDX(i)
      row=j-PNTR(i)+1
      y(row,1)=y(row,1)+A(i,1,1)*x(col,1)+A(i,1,2)*x(col,2)+A(i,1,3)*x(col,3)
      y(row,2)=y(row,2)+A(i,2,1)*x(col,1)+A(i,2,2)*x(col,2)+A(i,2,3)*x(col,3)
      y(row,3)=y(row,3)+A(i,3,1)*x(col,1)+A(i,3,2)*x(col,2)+A(i,3,3)*x(col,3)
   enddo
enddo
```

上のように各ブロック小行列内の演算をループ内で陽に記述することで、xの間接メモリ参照に関しては、一度のロードで得たデータを三度用いることができ、yの各成分のロードとストアも各ブロック小行列の行列ベクトル積につき一度で済むようにコンパイルされる。このようにデータのロードとストアを極力抑え、高い演算性能を発揮することができる。

アプリケーションによっては異なる支配方程式系を持つ複数の領域を取り扱うものもあり、その場合一般に節点当たりの未知数は領域ごとに異なる。このようにブロックサイズが可変な問題では、明らかに上のような方法は使えない。このような場合は、ベクトル化のためにブロック構造を無視して未知数単位でJAD形式を適用する方法を用いる[1]。この方法では、ブロックサイズ一定の場合の方法に比べ、最内側のループ長がbsize倍になる一方、各非ゼロ成分ごとにyのロードとストアおよびxのロードを行なわなければならずメモリへの負荷が増加する。実際、ブロックサイズが一定の場合に両者のメモリへのアクセス回数を比べてみると表1のようになる。

表1: 一ブロック小行列当たりのメモリアクセス回数

	COL_INDX(ロード)	y(ロード)	y(ストア)	x(間接アドレスによるロード)
サイズ一定	1	bsize	bsize	bsize
サイズ可変	$bsize^2$	$bsize^2$	$bsize^2$	$bsize^2$

4 数値実験の結果

本節では、これまで述べてきた二つのベクトル化手法の演算性能を数値実験の結果を用いて示す。
数値実験は以下の条件で行った。

1. 使用マシン：SX-5A-model 、1CPU(最大ピーク性能=8Gflops)
2. 使用例題：3次元構造解析、ブロックサイズ3

表 2: BILU 前処理付き反復法の計算速度 (Mflops) と CPU 時間

	Mflops 値		CPU 時間 (s)		反復回数	
	MC	RCM	MC	RCM	MC	RCM
ブロックサイズ可変	844	777	181.8	96.8	455	223
ブロックサイズ一定	3039	2483	50.5	30.3	455	223

表 2 は、二つのベクトル化手法を用いた場合のソルバ全体での Mflops 値と CPU 時間を示している。未知数の数は、3 × 343,000=1,029,000 である。BILU 前処理における節点の順序付け（オーダリング）は、Multicolor(MC) オーダリングと Reverse Cuthill-Mckee(RCM) オーダリングの二つを用いた。表 2から、ブロックサイズ一定時の最適化を行うことでブロックサイズ可変な問題に対応した汎用性のあるベクトル化手法に比べ Mflops 値が MC オーダリングで 3.6 倍、RCM オーダリングで 3.2 倍に向上することが分かる。これは、表 1に示されているようにメモリへの負荷がおよそ 3 分の 1 程度に低減されたためである。

5 まとめ

我々は、[1] において、BILU 前処理のベクトル化手法として、ブロックサイズ可変な問題に対応可能にするため未知数単位で前処理行列データを IDS-JAD 形式により記憶する方法を昨年開発した。しかし、汎用性を重視した手法のため、ベクトル長は長く確保できたが、演算性能は 844Mflops に留まった。

そこで、今回新たに、多くの問題ではブロックサイズが一定であることを考慮し、ブロックサイズ一定問題に最適化された BILU 前処理演算を実装した。この方法は、ロード、ストア、間接メモリ参照の負荷を極力削減したものである。この手法により、ブロックサイズが 3 で一定の問題を解いた場合、演算性能が約 3 倍向上し、3Gflops を達成した。これによりソルバの CPU 時間は約 3 分の 1 に短縮できた。この手法は、ブロックサイズがパラメータとして与えられる場合でも、アンローリング技法を用いて適用することができる。

参考文献

[1] ,K. Maruyama,T. Washio,S. Doi, 非構造メシュ用 Block ILU 前処理付き反復法のベクトル化手法, 第 79 回 HPC 研究会 (1999.12)

[2] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Company (1996).