ハイパフォーマンス
コンピューティング 79−13
（1999. 12. 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　鷲尾 巧　土肥 俊

日本電気株式会社 C&Cメディア研究所

概要

有限要素法による離散化の結果生じる大規模連立１次方程式をベクトル計算機上で解くことを考える。解法として、有限要素法の１節点上の複数の未知数をブロックとするBlock ILU(BILU) 前処理反復法を用いる。一般に行列ベクトル積のベクトル化のためのデータ構造として知られるDJAD 形式をBILU 前処理行列に適用する。これにより、BILU 前処理による前進後退代入計算において、CRS 形式よりも長いベクトル長が得られる。評価例題（３次元構造解析、未知数約３７万）により本手法の効果をNEC SX-4/8 A(1CPU) 上で評価し、前処理演算の計算時間が１３分の１に短縮できるという結果を得た。一般にILU 前処理においては、未知数のオーダリングが反復法の収束性、ベクトル性に大きな影響を与えることが知られている。本稿では、オーダリング方法についても考察し、BILU 前処理においてオーダリングの影響がNEC SX-4/8A 上でどのように現れるか評価した結果を示す。

A vectorization technique of block ILU preconditioning for unstructural problems

Kunihide Maruyama　Takumi Washio　Shun Doi

C& C Media Research Laboratories, NEC Corporation

Abstract

This paper deals with large sparse linear systems on high performance vector computers. Block incomplete LU (BILU) preconditioned iterative methods are adopted, where each block consists of unknowns on each node on a mesh in finite element or finite volume applications. The DJAD (Descending Jagged Diagonal) format is commonly applied to vectorize matrix vector multiplication for random sparse matrices. Proposed here is an extension of DJAD format for the BILU preconditioning. This technique enables to realize the vector length longer than the case implemented with a standard CRS (Compressed Row Storage) format. Numerical experiments using three dimensional structural analysis problems show that the computational speed obtained with a solution method using this DJAD format is 13 times faster than that obtained with the same solution method with the CRS format on an NEC supercomputer SX-4/8A. In general, an ordering of nodes in the ILU preconditioning has substantial influence on the convergence of the preconditioned iterative methods and the parallelism in the preconditioning. The effect of different orderings, i.e., the RCM (Reverse Cuthill- Mckee) and the multicolor orderings, on the total CPU time will also be compared on the SX-4/8A vector parallel supercomputer.

1　はじめに

偏微分方程式系をメッシュ分割し離散化して解く場合、大規模な連立１次方程式系を解くことになる。解くべき連立１次方程式系を

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

とする。連立１次方程式の反復法による求解では、係数行列の条件数が大きい場合、収束までの反復回数が増大したり、収束しない場合がある。そこで通常、連立１次方程式の反復法による求解は、前処理と組み合わせた前処理付き反復法を用いる。これは、方程式を変形し係数行列の条件数をなるべく小さくした上で、反復法を適用して解を求める方法である。

前処理行列の生成方法としてしばしば用いられるのがILU(Incomplete LU) 分解である。この前処理手法を物理空間内の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 分解（以後BILU 分解とする）がある。これは、成分単位でILU 分解を行うのではなく、未知数をブロックに分割し、ブロック単位でILU 分解を行う手法である。通常、ブロックの作り方としては、一つの節点上の変数の集合を一つのブロックとする。

ブロック内にゼロ成分が含まれる場合は、つぎのような理由から、一般にブロック単位でILU分解を行った方がロバストな前処理が実現できる。通常、物理空間内の同一節点上にある変数は互いに強く結びついている。したがって、もしある節点上の変数が他の節点上のある変数と強く結びついている場合は、それらの節点上のすべての変数どうしが強く結びついているといえる。このような場合、あるブロック行列内に非ゼロ成分があれば、そのブロック行列内に生じるfill-in成分はすべて取り込んだ方が妥当である。さらに、後にみるように前処理演算の実装では、対角ブロック行列の逆行列を陽に持つので、前進代入・後退代入演算における同一ブロック内の変数同士の依存性がなくなり、ベクトル処理に有利になる。以上のように、一節点上に多数の変数が存在するようなアプリケーションを念頭におき、効率良くベクトル化されたBILU前処理付き反復法のベクトル化手法を開発した。

以下、添字の表現は、行列の成分単位の表現は小文字、ブロック単位の表現は大文字とする。前処理行列をMとする時、BILU分解行列は次のように表される。

$$M = (L + D)D^{-1}(D + U). \tag{2}$$

上の式を展開すると

$$M = L + D + U + LD^{-1}U \tag{3}$$

となる。以下、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、次のようにブロック単位で表現されているものとする。$nbks$はブロックの総数とする。

$$L = (L_{I,J})_{1 \le I,J \le nbks},$$
$$U = (U_{I,J})_{1 \le I,J \le nbks},$$
$$D = \mathrm{diag}(D_I)_{1 \le I \le nbks}.$$

このような条件下で、非ゼロパターンPBのもとでのBILU分解は次のように定式化される。通常は、行列Aのブロック単位非ゼロパターンをPBとする。

$$M_{I,J} = A_{I,J} \quad \text{for} \quad (I,J) \in PB, \tag{4}$$
$$L_{I,J} = 0 \quad \text{for} \quad (I,J) \notin PB, \tag{5}$$
$$U_{I,J} = 0 \quad \text{for} \quad (I,J) \notin PB. \tag{6}$$

式(3)より、上記の条件は、次の式が満たされることと等価であることがわかる。

$$D_I = A_{I,I} - \sum_{K<I} L_{I,K} {D_K}^{-1} U_{K,I} \tag{7}$$

$$L_{I,J} = \begin{cases} A_{I,J} - \sum_{K<\min(I,J)} L_{I,K} {D_K}^{-1} U_{K,J} & \text{for} \quad (I,J) \in PB \\ 0 & \text{for} \quad (I,J) \notin PB \end{cases} \tag{8}$$

$$U_{I,J} = \begin{cases} A_{I,J} - \sum_{K<\min(I,J)} L_{I,K} {D_K}^{-1} U_{K,J} & \text{for} \quad (I,J) \in PB \\ 0 & \text{for} \quad (I,J) \notin PB \end{cases} \tag{9}$$

一般に、BILU前処理を行う場合の記憶方法は、非ゼロブロック行列内の成分を総て密に記憶する方式が採用されることが多い。しかし、この方式ではブロック行列内にゼロ成分が多く存在する場合、ゼロ成分を記憶し演算してしまう。例えば、化学反応を伴う系のシミュレーションでは、未知数単位でグラフを構成すると、物理空間の同一節点内の未知数としか繋がっていない未知数が多く存在する。このとき非対角非ゼロブロック内には、総てゼロ成分となる行や列が現れる。この場合、BILU分解実行後も非対角非ゼロブロック行列$L_{I,J}$, $U_{I,J}$の中に多くのゼロ成分が現れる。その理由は次の通りである。式(8)、(9)から、非対角非ゼロブロック行列は$L_{I,K} D_K^{-1} U_{K,J}$を引いていくことによって計算される。下三角非ゼロブロック$L_{I,K}$の第i行の成分が総てゼロとすると、行列積の項の第i行の成分も総てゼロとなる。よって$\sum_{K<\min(I,J)} L_{I,K} D_K^{-1} U_{K,J}$の第i行の総ての成分もゼロになる。上三角非ゼロブロック行列$U_{I,J}$については第j列が総てゼロ成分とすると同様ことが言える。これよりBILU分解後も非対角非ゼロブロック行列にはゼロ成分が多く現れる。そこで、前処

理行列の生成時には、行列データの格納方法として、D^{-1} の対角ブロック部分は密に記憶し、L, U は非ゼロ成分のみを CRS (compressed row storadge) [2] 形式で記憶する方法を用いた。しかし、この方法では前処理演算実行時に L, U の各行の非ゼロ成分の個数の長さしかベクトル長が取れない。

そこで、前処理演算のベクトル化を考慮した格納方式として、独立集合ごとにベクトル演算高速化のためのデータ構造として知られる DJAD (Descending jagged diagonal) [2] 形式で行列データを格納する方法を提案する。ここで独立集合とは、前処理演算における前進・後退代入時に依存関係のないブロックによって構成される集合である。(第2節参照)。この手法を採用することで、独立集合内のブロックの数が少ない場合や、ブロックの大きさが異なる場合でも、効率良い前処理演算のベクトル化を実現できる。

次に、未知数のオーダリング方法について述べる。前処理付き反復法のベクトル性、収束性を向上させるため、ILU 分解における未知数のオーダリングが既にいくつか考えられている。ベクトル長を長く確保しベクトル性能を向上させるオーダリングとして、Multicolor オーダリングがある。しかし、Multicolor オーダリングは収束性が悪い。一方で、収束性を向上させるオーダリングとして、Reverse Cuthill-Mckee オーダリングがある。ただし、これはベクトル性能があまり良くない。一般に、規則構造の問題の場合についての ILU 前処理におけるオーダリング方法の変更による収束性向上とベクトル性向上の間にはトレードオフの関係があることが知られている [1]。本稿では、非構造の問題を仮定し、BILU 前処理におけるオーダリング変更によるトレードオフの関係がどのように現れるか NEC SX-4/8A の上で評価した結果を示す (4節参照)。

2 前処理演算をベクトル化するための行列データ格納形式

本節では、前処理演算をベクトル化するための行列データの格納方法について述べる。

まず、ブロック単位の非ゼロパターン PB を基に前進後退代入時に同時に計算可能なブロックを集め独立集合を構成する。そして、D^{-1}, L, U を独立集合ごとに行列ベクトル積のベクトル化を行うためのデータ構造として用いられる DJAD 形式で格納する。このデータ構造を用いることで、ベクトル長は独立集合内に含まれる未知数の総数に近くなるので、独立集合内のブロック数が少ない場合でも長いベクトル長が期待できる。また、この方式はブロックのサイズが可変な問題にも適用できる。しかし、この方式では、列番号を参照するためのメモリアクセスによる付加が大きくなるということにも注意が必要である。

独立集合 $IDS(1), IDS(2), ...$ は、行列 $L+U$ の非ゼロパターンを PB とすると以下のようにして帰納的に生成される。

$$IDS(1) := \{I | I \leq J \text{ for } \forall (I,J) \text{ or } (J,I) \in PB\} \tag{10}$$

$$IDS(k) := \{I | I \leq J \text{ for } \forall (I,J) \text{ or } (J,I) \in PB, J \notin IDS(1) \cup ... \cup IDS(k-1)\} \tag{11}$$

nids をこのようにして構成された独立集合の総数とする。このとき、前進代入 $x := D^{-1}(y - Lx)$ は $IDS(1) \rightarrow IDS(2)...$ の順で、後退代入 $x := x - D^{-1}Ux$ は $IDS(nids) \rightarrow IDS(nids-1)....$ の順で計算でき、各独立集合内の演算は並列に行える。具体的には、次のアルゴリズムにおいて、各独立集合 $IDS(il)$ の中で実行される行列ベクトル積をベクトル演算高速化のためのデータ構造である DJAD 形式のもとで演算することでベクトル化することができる。

独立集合にもとずく前処理アルゴリズム

前進代入によりで $(L+D)x = y$ を解く
for $il = 1, ..., nids$
  for $I \in IDS(il)$
    $w_I := y_I - (Lx)_I$
  end
  for I $\in IDS(il)$
    $x_I := (D^{-1}w)_I$
  end
end

後退代入により $(I+D^{-1}U)x_{NEW}=x_{OLD}$ を解く
for $il=nids,...,1$
  for $I \in IDS(il)$
    $w_I := (Ux)_I$
  end
  for $I \in IDS(il)$
    $x_I := x_i - (D^{-1}w)_I$
  end
end

3 ブロックオーダリング

本節では、Multicolor（以下MCとする）オーダリングとReverse Cuthill-Mckee（以下RCMとする）オーダリングについて述べる。以下では、行列のブロック単位の非ゼロ構造をグラフと同一視し、グラフ理論の用語を使ってオーダリングについて議論する。ここで、ノードはブロックに対応する。$G=(V,E)$ をブロック単位の非ゼロ構造に対応する向き付きグラフ、$\bar{G}=(V,\bar{E})$ を対応する向き無しグラフとする。

Cuthill-Mckee(CM) オーダリングでは、全ノードを互いに交わらないレベル集合と呼ばれる部分集合 $LS(0)$, $LS(1)$,... に分け、

$$LS(0) \rightarrow LS(1) \rightarrow LS(2) \rightarrow \cdots$$

の順に並べる。ここで、$LS(l)$ 内では、通常 degree の小さなノードから先に並べていく。与えられたノードの degree とは、そのノードにつながっている辺の個数のことである。向き無しグラフ $\bar{G}=(V,\bar{E})$ のもとで、レベル集合 $LS(l)$ は適当に選ばれたノード v_0 に対して

$$LS(l) := \{v \in V \mid d(v,v_0)=l\}, \tag{12}$$

により定義される。ここで $d(v,u)$ はノード v と u の間のグラフ $\bar{G}$ 上の距離である。

RCM オーダリングでは、まず任意に選んだ出発ノードから CM オーダリングを構成し、その最後尾のノードを出発ノード v_0 として再び CM オーダリングを適用する。v_0 はグラフの疑似的な境界上の点とみなすことができ、境界点から出発することにより、オーダリング後の行列のバンド幅をなるべく小さくすることができる。

次に MC オーダリングについて述べる。MC オーダリングでは、グラフ上で互いにつながっているノード同士が異なるカラーをもつように色分けする。また、ベクトル長を長く確保するため、この際用いるカラーの数がなるべく少なくなるようにする。その後、ノードをカラー毎に並べる。MC オーダリングでは、ノードを1番目のカラーに属するものから順に並べていく。このオーダリング下で、fill-in なしの BILU 分解を行った場合、レベルスケジューリングでの各 color 集合が一つのカラーに対応し、ベクトル効率の良い前処理が実現できる。しかし、このようなオーダリング下での fill-in なしの BILU 前処理は、RCM オーダリングなどに比べ例えば異方性問題に弱いことが知られている。まず、次の手順により color 集合 $C(ic)$ を $ic=1$ に対して作成する。ここで、V 上のリスト *connect* は最初はゼロに初期化されているものとする。

color 集合 $C(ic)$ の作成

for $v_i \in V$ (from smaller i to larger i)
  if ($connect(v_i) \neq ic$) then
    $C(ic) := C(ic) \cup \{v_i\}$
    for $v_j \mid (v_i,v_j) \in E$ or $(v_j,v_i) \in E$
      $connect(v_j) := ic$
    end
  endif
end

このようにして、color集合 $C(1)$ を選択した後、これらのノードとそれに繋がる辺をグラフ $G=(V,E)$ から消去する。このようにして、更新されたグラフ $G=(V,E)$ から、先のアルゴリズムにより次のcolor集合 $C(2)$ を選ぶ。この操作を全てのノードが選ばれるまで繰り返し、ノードのオーダリングを $C(1)$ から順に付けていく。このようなオーダリング下で、BILU分解を実施した場合、各 $C(ic)$ 内では、前進後退代入を並列に実行できる。

4 数値実験の結果

本節では、これまで述べてきたBILU前処理付き反復法のベクトル化手法について、例題を用いて評価した結果を示す。例題は、3次元構造解析の問題である。問題作成では、(財) 高度情報科学技術研究機構 (RIST) で開発、公開されている並列有限要素法コード「GeoFEM/Tiger」を使用した。ブロック構造は、1節点上の3個の未知数を一つのブロックとして定めている。問題の連立1次方程式の係数行列は対称行列であるため反復法としてCG法を用いている。反復計算は、相対残差ノルムが 10^{-8} 以下になるまで実行した。使用マシンは、NEC SX-4/8Aの1 CPUである。CPUのピーク性能は、2Gflopsである。メモリは、6GBのSDRAMである。テスト作業は、他のユーザとの共有時間内で1CPUを使って行った。

4.1 行列データ格納形式によるベクトル性能の変化

この節では、同じオーダリング指定方法のもとで行列データの格納形式を換えることによる前処理演算のベクトル性能についての評価結果を示す。なお、問題規模は1辺あたりの節点数が50の場合で未知数の数は375,000である。表の縦の項目はオーダリング指定方法を示している。"MC" はMulticolorオーダリング、"RCM" はReverse Cuthill-Mckeeオーダリングを示している。以下同様に略記する。表の横軸には、"psolve" で1反復で費やす前処理演算の時間、"Mflops of psolve" で前処理演算のMflops値、( ) 内に前進後退代入時の最内側の平均ループ長を示している。"solve" では1反復全体で費やした時間を示している。"nids" は独立集合の総数を表している。上記の表から、前処理演算を実行する際データの格納方法をCRS形式から今回提案

表 1: データ構造を換えることによる前処理演算のベクトル性能比較

	CRS			DJAD			
	psolve	Mflops of psolve	solve	psolve	Mflops of psolve	solve	nids
MC	2.45	24(20)	3.02	0.17	348(45091)	0.33	8
RCMK	2.46	25(20)	3.33	0.19	323(450)	0.34	624

した方式に換えることで1回の反復における前処理演算の時間が約1 3分の1から約1 4分の1に短縮できることが解る。これは、独立集合ごとにDJAD形式でデータを格納することで、ベクトル長が独立集合内に含まれる未知数の総数に近い数になるからである。

4.2 オーダリング方法を換えることによるベクトル性、収束性への影響

本説では、fill-inなしの場合にオーダリングを換えることによるベクトル性能、計算時間、収束性能などへの影響を示す。以下の表では、横軸に1辺あたりの節点数と () 内に未知数の総数を示している。縦軸には、MCオーダリングとRCMオーダリングについて以下の項目を示す。"Mflops of solve" は、反復法計算実行時のMflopsである。"time of solve" は、反復法計算に費やした時間を示している。"length of loop" は、前処理演算実行時の前進・後退代入計算を実行する時の最内側のループ長の平均を示している。() 内の数字は独立集合の総数である。"itr" は、収束まで反復回数を示している。"time of fac" は、BILU分解を実行する部分で費やした時間と () 内でMflops値を示している。

上記の結果を見ると、MCオーダリングは、1辺当たりの節点数が少なくてもベクトル性能が良い。これは、独立集合の数が少ないため、未知数の数が3,000でもベクトル長が300程度確保できるためである。これに対して、RCMオーダリングでは、問題規模を大きくしないとベクトル性が向上しない。これは、MCオー

表 2: オーダリング方法を換えることによるベクトル性・収束性への影響

		1辺当たりの節点数、() 内は未知数の総数				
		10(3,000)	20(24,000)	30(81,000)	40(192,000)	50(375,000)
MC	Mflops of solve	342	449	487	410	457
	time of solve	0.15	2.25	11.21	43.46	97.52
	length of loop	307(8)	2720(8)	9488(8)	22862(8)	45091(8)
	itr	51	113	173	237	297
	time of fac	0.19 (55)	1.69(59)	5.67(63)	14.34(61)	27.11(64)
RCM	Mflops of solve	95	271	348	390	434
	time of solve	0.35	2.20	8.79	25.17	55.36
	length of loop	20(106)	76(234)	166(364)	290(494)	450(624)
	itr	33	66	97	129	161
	time of fac	0.19(40)	1.66(42)	5.77(43)	13.84(43)	26.32(45)

ダリングに比べ独立集合の数が多く、一つの独立集合内の未知数の数が少なくなり、ベクトル長が短くなってしまうためである。

Mflops 値を見ると、MC オーダリングでは 4 万以上の長いベクトル長を確保できるが、ある程度のベクトル長（500 程度）を越えればベクトル性能は飽和していることが解る。一方で、MC オーダリングは、反復回数が RCM オーダリングの約 2 倍程度になり反復法の計算時間を見ると 1 辺当たりの節点数が 20 以上の場合で、RCM オーダリングの方が計算時間が短い。MC オーダリングは反復回数の増大によりベクトル性能が相殺されてしまっていることが解る。

BILU 分解の Mflops 値を見ると、あまりベクトル性能が良くない。これは、我々の実装方式では、最内側のループ長が U の 1 行当たりの非ゼロ成分の数しか確保できないためである。この部分の演算においても独立集合内の各行の演算は並列に実行可能だが、ベクトル化のための適切なデータ構造が知られておらず、今後の検討課題である。特に上記例題では、以下に述べるように BILU 分解に必要な演算量の占めるウエイトが大きいため、この部分のベクトル化は必須である。未知数の数を n とし L, U の 1 行当たりの非ゼロ成分の平均個数をそれぞれ nzl, nzu とする。すると、1 行当たりの計算量は、$nzl \times nzu$ に比例するので、分解全体では、$n \times nzl \times nzu$ に比例する。よって、前処理行列生成部に必要な演算量は n に比例する。一方、反復計算実行部分では、前処理演算、行列ベクトル積、内積の計算は、n に比例し、反復回数は、$n^{1/3}$ に比例する。よって、反復計算全体では、演算量は $n^{4/3}$ に比例する。一般に、このような理由から n が大きい場合では、前処理行列生成部の演算量は反復法実行部の演算量と比べれば少ない。ところが、上記の例題では、$nzl = nzu = 13 \times 3 = 39$ と L, U の 1 行当たりの非ゼロ成分の数が多いため、n が大きい場合でも前処理行列生成部の演算量の全体に占めるウエイトは大きい。

5 まとめ

評価の結果、今後の課題として次のことが必要と言える。MC オーダリングでは、ベクトル長を長く取るために色数をできるだけ少なくしている。しかし、収束性が RCM オーダリングに比べて良くない。また、ベクトル性能を発揮させるにはベクトル長は 500 程度あれば充分である。そこで、ベクトル長がある程度の長さ確保できる程度まで色数を増やし、RCM オーダリングに近い収束性を実現する必要がある。

また、BILU 分解実行部の時間を短縮することが必要であり、今後検討していく予定である。

参考文献

[1] I. Duff and G. Meurant, The Effect of Ordering on Preconditioned Conjugate Gra dients, BIT 29 (1989), pp.635–657.

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