分散メモリ型並列計算機向け スパース対称行列ソルバの開発と評価

山本有作[†] 猪貝光祥[††] 直野健[†]

分散メモリ型並列計算機上での構造解析向け連立一次方程式ソルバとして，直接法に基づく対称疎行列用ソルバを開発し，並列計算機 SR2201 上で評価を行った．本ソルバは対称正定値でかつ非零構造が 3×3 のブロック単位で現れる行列を対象とし，行列のオーダリングから，プロセッサ間への分割，シンボリック分解，コレスキー分解，前進後退代入までを一貫して行うプログラムである．計算の中心となるコレスキー分解においては，行列の不規則な非零構造に合わせて局所的に最適なループ展開を行うことにより，RISC 向けの最適化を行っている．本プログラムを SR2201 上で評価したところ，比較的小規模な約 32000〜70000 元の 3 次元構造解析の問題で単体性能約 100 MFLOPS（ピークの 33%），16 プロセッサ時に約 10 倍の加速率を達成し，実用的な性能が得られることを確認した．

Development and Evaluation of a Direct Solver for Sparse Symmetric Systems on Distributed-memory Parallel Processors

Yusaku Yamamoto,[†] Mitsuyoshi Igai[††] and Ken Naono[†]

We developed a parallel direct solver for a sparse symmetric positive-definite system of equations arising from structural analysis on distributed-memory parallel processors. Our solver is intended for a symmetric positive-definite matrix whose nonzero elements appear in a 3 by 3 cell, and includes programs for ordering, distribution of the matrix, symbolic factorization, numerical factorization, and solution. The numerical factorization part is optimized for RISC processors by using the optimal loop unrolling pattern for each part of the matrix, according to the local nonzero structure. We evaluated this solver on the Hitachi SR2201 parallel processors, and obtained single processor performance of about 100 MFLOPS and speedup of 10 on 16 processors for relatively small structural analysis problems of dimension approximately from 32000 to 70000.

1. はじめに

有限要素法による構造解析は，機械設計や建物設計をはじめとして，産業界で広く使われているアプリケーションの 1 つである．近年，計算の大規模化・精緻化にともない，構造解析においても，高い演算能力と広いメモリ空間を持つ分散メモリ型並列計算機の利用要求が高まっており，MARC や NASTRAN など代表的なソフトウェアの分散メモリ型並列計算機への移植が行われている．構造解析では，多くの場合，連立一次方程式の求解が計算の中心であり，計算時間の大部分を連立一次方程式ソルバが占める．したがって，全体性能を上げるには効率の良い分散メモリ型並列計算機向け連立一次方程式ソルバの開発が不可欠である．

構造解析で現れる行列の多くは，対称正定値で条件数が大きい大規模疎行列である．このような行列を係数とする連立一次方程式を安定に解く方法として，スパースソルバと呼ばれる直接法（コレスキー分解）に基づく解法がある[1]．スパースソルバではコレスキー分解後に非零になる要素についてのみ格納場所を用意し，演算を行うことにより，スカイライン法に比べて大幅な演算量削減を実現している．

スパースソルバの並列化の研究は数多く行われており，オーダリング[4)〜6)]，シンボリック分解[2]，コレスキー分解[8]，前進後退代入[7]など，ソルバを構成する各部分について，分散メモリ型並列機向けのアルゴリズムが提案・評価されている．しかし，これらすべての処理を一貫して行い，連立一次方程式の求解が行えるプログラムはまだ数が少なく，ScaLAPACK プロジェクトの一環として開発された CAPSS[9]，欧州の

† 株式会社日立製作所中央研究所
Central Research Laboratory, Hitachi Ltd.
†† 株式会社日立超 LSI システムズ
Hitachi ULSI Systems Corp.

共同プロジェクトで開発中の並列疎行列ソルバライブラリ PARASOL の一部である MUMPS [10),12)]，ミネソタ大学の Kumar らにより開発された PSPACES [11)] など，いくつかのシステムが利用可能となっているのみである．

本研究では，有限要素法による構造解析で現れる，対称正定値でかつ非零要素が 3×3 のブロック単位で現れる行列を対象として，行列のオーダリングから前進後退代入までを一貫して行うソルバを開発した．本ソルバの特徴は，(1) シンボリック分解以降の処理を完全に分散メモリ上で行うことで，分散メモリ型並列機の広いメモリ空間を最大限利用したいユーザに対応できるとともに，1 プロセッサのメモリ上に格納した行列を複数プロセッサ間に自動分割する行列分割ユティリティにより，分散メモリによるプログラミングの複雑さを最小限に抑えて並列ソルバを利用したいユーザにも対応できる点，(2) 計算の中心部であるコレスキー分解の消去演算のループにおいて，局所的に最適なループ展開を行うことにより RISC プロセッサ向けの最適化を行った点の 2 つである．

なお，本研究で開発したソルバと，上で述べた既存のソルバとの関係は次のとおりである．MUMPS [10),12)] では，プロセッサのうち 1 台をホストプロセッサとし，入力行列全体をホストプロセッサのメモリ上に置いてソルバをコールする仕様になっている [12)]．これは単一プロセッサ用の有限要素法プログラムと同じ行列格納形式のため，ユーザに馴染みやすい反面，1 プロセッサのメモリサイズで解ける問題のサイズが制約されるという問題がある．これに対して本ソルバでは分散形式の行列入力も可能とし，大規模問題を解きたいユーザに対応しやすいようにする方針をとった．一方 CAPSS [9)] は，オーダリングの部分も含めすべての処理を分散メモリ上で行っているという特徴を持つ．しかし，演算量削減効果の大きいオーダリング手法は並列化が困難であるため，CAPSS では Cartesian Nested Dissection 法 [9)] と呼ぶ比較的単純で演算量削減効果の小さい手法を利用している．これに対して本ソルバでは，演算量削減効果の大きいオーダリング手法を利用するため，オーダリングの部分のみは単一プロセッサで行う方式を採用した．PSPACES [11)] はすべての処理を分散メモリ上で行うソルバであり，かつオーダリングの部分では本研究で採用した Multilevel Nested Dissection（MND）法 [4),5)] と同じ原理に基づく演算量削減効果の大きいアルゴリズムを分散メモリ上で並列化するなど，先進的な試みを行っている．本ソルバではすでに実績のあるアルゴリズムである逐次型の MND 法を採用したが，今後はより多いプロセッサ台数に対応するため，PSPACES のような分散メモリ型のオーダリング手法についても検討を行う予定である．

以下では，まず 2 章でスパースソルバの基本的な事項を述べた後，3 章で並列化の詳細，4 章で RISC 向けの最適化について述べる．5 章では分散メモリ型並列機 SR2201 上での評価結果を示す．最後に 6 章でまとめと今後の課題を述べる．

2. スパースソルバ

2.1 処理の流れ

スパースソルバでは，オーダリング，シンボリック分解，コレスキー分解，前進後退代入の処理を順次行うことにより，N 元の対称正定値行列 A を係数とする連立一次方程式 $Ax = b$ の解を求める．以下，これらの各処理について簡単に説明する [1),2)]．

（1）オーダリング

適当な置換行列 P により係数行列 A の行と列に対して同時置換を行い，$A' = PAP^t$ に変換する．置換行列 P は，コレスキー分解 $A' = LL^t$ を行ったときのフィルイン（分解前に零であって分解後に非零になる要素）の数ができるだけ少なく，かつコレスキー分解における並列性ができるだけ高くなるように選ぶ．オーダリングの主な手法としては，消去の各ステップで生じるフィルインの数を最小にするというアイディアに基づく Minimum Degree（MD）法系統の手法 [3)] と，再帰的な領域分割に基づく並列機向きの Nested Dissection（ND）法系統の手法 [4)~6)] がある．

ND 法では，有限要素法のメッシュをセパレータと呼ばれる節点集合により 2 つの部分領域 A と B に分割する．ここでセパレータは，A に属する節点と B に属する節点とを結ぶ辺がないようにとる．分割後，A に属する節点，B に属する節点，セパレータに属する節点の順に番号を付け直す．セパレータのとり方より，A に属する節点と B に属する節点とを結ぶ行列要素はないため，このオーダリングによって行列は縁付きブロック対角行列（$N \times N$ の正方行列であって，左上の $K \times K$ 小行列（ただし $2 \leq K \leq N-1$）がブロック対角行列であるような行列）に変形される．さらに，この分割を各部分領域に対して再帰的に繰り返すことにより，行列は，各対角ブロックの内部が再び縁付きブロック対角行列であるような，再帰的縁付きブロック対角行列に変形される．

ND 法によるオーダリングの例として，簡単のため 2 次元の有限要素法の場合を**図 1** と**図 2** に示す．図 1 に示す 2 次元の規則的メッシュ上でポアソン方程式を

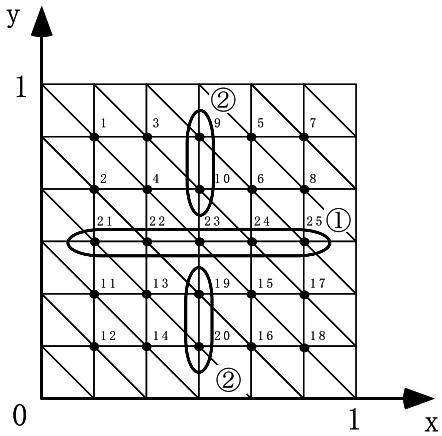


図 1 有限要素法のメッシュとセパレータの例
Fig. 1 An example of Finite Element mesh and the separators.

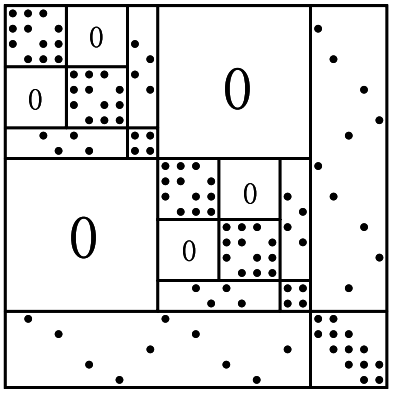


図 2 ND 法によりオーダリングを行った有限要素法の行列
Fig. 2 A matrix generated by the FEM and reordered by the ND method.

三角形一次要素を用いて有限要素法により離散化し，得られた行列に ND 法による置換を行った結果が図 2 である．なお，図 1 中の各節点に対し，図 2 の行列での対応する列の番号を右上に 1 から 25 までの数字で示してある．この例では，図 1 のメッシュをまず①のセパレータにより 2 分割し，各部分領域をさらに②のセパレータにより 2 分割することにより，行列を再帰的縁付きブロック対角形に変形している．

（2）シンボリック分解

コレスキー分解に先立って分解後の非零要素の位置をあらかじめ計算し，フィルインのための記憶領域を確保するとともに，コレスキー分解において非零要素をアクセスするためのインデックスリストを作成する．

（3）コレスキー分解

シンボリック分解で作成したインデックスリストを用いてコレスキー分解 $A' = LL^t$ を行う．コレスキー分解では，ガウス消去法の中心演算

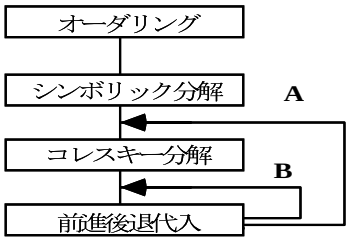
$$a_{ij} := a_{ij} - a_{ik}a_{kj}/a_{kk} \qquad (1)$$

において，行列の対称性を利用して下三角部分についてのみ演算を行い，かつ分解後に非零となる要素のみを計算する．主な計算方法として，更新に使うピボット列 k に関するループを最外側に持ってくる外積形式，更新される列 j に関するループを最外側に持ってくる内積形式，外積形式において更新を行列に対してすぐに行わずにフロンタル行列と呼ばれる密行列にためておき，後でまとめて更新を行うマルチフロンタル法などがある [2]．

（4）前進後退代入

方程式 $Ly = b$ と $L^t x = y$ とを順次解くことにより，方程式 $Ax = b$ に対する解を求める．

スパースソルバ全体の処理のフローチャートを**図 3** に示す．シンボリック分解をコレスキー分解から分離したことにより，行列の非零構造が変わらず行列要素の値のみが変化する方程式を繰り返し解く場合には，コレスキー分解以降の処理のみを反復すればよい．このことは，たとえば係数行列が解 x の関数となるような非線型連立方程式 $A(x)x = b$ をニュートン法で解く場合，反復計算の各ステップで連立一次方程式を解くときに利用できる．一方，線形の時間発展問題の計算のように，行列が変わらず右辺ベクトルのみが変化する場合には，前進後退代入のみを反復すればよい．



A: 非零要素の値のみが変わる場合
B: 右辺ベクトルのみが変わる場合

図 3 スパースソルバのフローチャート
Fig. 3 Flow chart of the sparse solver.

2.2 消去木と消去演算の並列性

次に，スパースソルバの並列化を行うにあたって便利な消去木について述べる [2]．消去木はコレスキー分解後の行列 L の非零構造を用いて定義される節点数 N のグラフであり，各節点が L の各列に対応する．2 個の節点 i と $j\,(i > j)$ に対し，$i = \min\{k > j | L_{kj} \neq 0\}$ であるとき，i を j の親であると定義し，$i = p(j)$ と書く．行列が既約な（すなわち行と列の置換によってブロック対角行列に変換されない）場合，このグラ

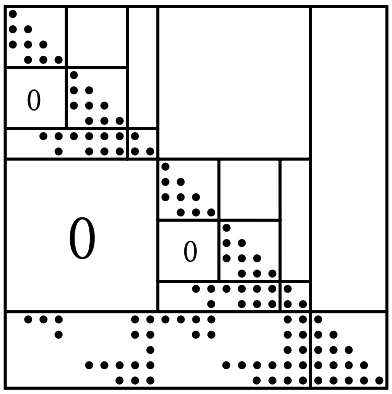


図 4 図 2 の行列の分解後の非零構造
Fig. 4 The Nonzero structure of the matrix in Fig. 2 after decomposition.

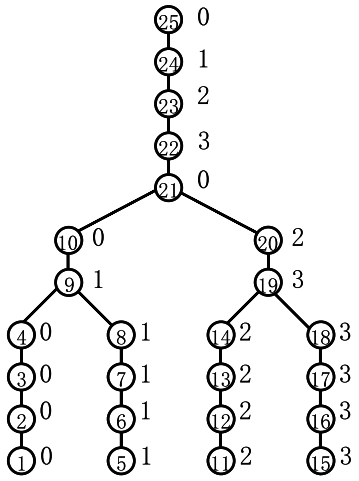


図 5 図 4 の行列に対する消去木
Fig. 5 The elimination tree corresponding to the matrix in Fig. 4.

フは行列の第 N 列に対応する節点を根とする木になる[2]．図 2 の行列に対するコレスキー分解結果の非零構造（下三角部分のみ）を**図 4** に，また対応する消去木を**図 5** に示す．図 5 において，丸の中の数字は各節点に対応する列の番号を示す．行列の再帰的ブロック対角構造と消去木の再帰的な部分木構造とが対応し，対角ブロックが部分木に，セパレータが分岐と分岐の間の鎖の部分に対応する．

消去木の性質として，「行列の第 j 列による更新が第 i 列に影響を及ぼすのは，消去木において節点 i が節点 j の祖先であるときに限る」ということが成り立つ．したがって，コレスキー分解に内積形式を用いる場合，ある列 i の更新に使う列はすべて消去木上での節点 i の子孫であり，節点を共有しない複数の部分木の更新演算は独立に行える．

このことを用いて subtree-to-subcube 割当て[2]という行列データの分割法が提案され，広く使われている．subtree-to-subcube 割当てでは，消去木の根から始めて各節点（すなわち行列の各列）にプロセッサをサイクリックに割り当ててゆき，木の分岐点にきたらプロセッサ群を 2 分割して各々を 2 個の部分木に割り当て，各部分木内で再びサイクリック割当てを繰り返す．図 5 の消去木に対して subtree-to-subcube 割当てにより 4 台のプロセッサを割り当てたときの担当プロセッサ番号を，各節点の右に示してある．

subtree-to-subcube 割当てでは，1 台のプロセッサに割り当てられた部分木の更新演算を通信なしに行える．したがって，演算の並列性を高めるには，なるべく多くの節点がこれらの部分木に含まれるように（すなわちセパレータがなるべく小さくなるように）し，かつ各部分木の大きさがなるべく均等になるように（ただし，各部分木に割り当てられたプロセッサ数に差があるときは，1 プロセッサあたりの負荷が均等になるように）すればよい．

なお，subtree-to-subcube 割当てにより各プロセッサに分割された行列データは，各プロセッサごとに次の 3 個の配列を用いて格納する．

(1) 行列の非零要素のうちで自プロセッサが担当するもののみを，第 1 列から第 N 列まで 1 次元に並べた配列，
(2) 配列 (1) 中の各非零要素の行番号を同様に 1 次元に並べた配列，
(3) 配列 (1)，(2) 中で各列に属する最初の要素を指すポインタの配列．

さらに，全プロセッサで共通な配列として，各列を担当するプロセッサの番号を格納する配列，消去木の各節点に対して親の節点番号を格納する配列も用意する．

3. スパースソルバの並列化

3.1 行列の入力方式

本章では，スパースソルバの各部分の並列化手法について述べる．

図 3 に示したスパースソルバの各処理のうち，本研究で開発したプログラムでは，オーダリング部分のみを 1 プロセッサで行い，その後のシンボリック分解から前進後退代入までの処理は完全に分散メモリ上で行う方式を採用した．オーダリングを 1 プロセッサで行うことにした理由は，分散メモリ上でオーダリングを行う手法が研究途上で[9]まだ確立されておらず，またオーダリングは分解前の行列の非零構造に対して行う

ので，分解後の行列の非零構造を扱うシンボリック分解以降のステップに比べて所要メモリが少ないためである．また本ソルバでは，図3に示した4つの処理に加えて，1プロセッサのメモリ上に置かれた行列をソルバの要求する形式（2.2節参照）で複数のプロセッサに分割するユティリティプログラムも用意した．

以上に述べた構成より，分散メモリ型並列機の広いメモリ空間を最大限利用して大きい問題を解きたいユーザは，分散メモリ上で行列生成を行う有限要素法プログラムと本ソルバとを接続することで，1プロセッサに置ける行列のサイズに制約されずに，大きい問題を解くことが可能である（ただしこの場合，行列のオーダリングは事前に行っておくか，あるいは分散メモリ上でオーダリングができるプログラムを別途用意する必要がある）．一方，分散メモリに付随するプログラミングの複雑さを最小限に抑えつつ，負荷の重い連立一次方程式求解部分のみを並列化したいユーザは，1プロセッサ用の有限要素法プログラムと本ソルバとを行列分割ユティリティを用いて接続することで，容易に本ソルバを利用することが可能である．

3.2 各部分の並列化

（1）オーダリング

オーダリングにはND法の一種であるMultilevel ND（MND）法[4),5)]を採用した．ただし本ソルバでは有限要素法のメッシュを入力とせず，行列と右辺ベクトルのみを入力とするため，メッシュを分割する代わりに，入力行列の非零構造により定義されるグラフに対して分割を行う．MND法は，入力グラフを粗いグラフで近似し，粗いグラフ上でのセパレータを求め，それを再び元のグラフ上に引き戻す，という処理を再帰的に行うことによりセパレータを求める手法である．MND法は，セパレータが小さく部分木の大きさが均等な分割が得られるだけでなく，セパレータを求めるための演算量もSpectral ND法[6)]など他のND法系統のアルゴリズムに比べて小さいという特長を持つ．

本ソルバでは，MND法により入力行列に対応するグラフをプロセッサ台数分の部分グラフへ分割する．これにより，行列はプロセッサ台数分の対角ブロックを持つ再帰的縁付きブロック対角行列へ変形される．消去木上で見ると，各対角ブロックは異なる部分木に含まれるため，subtree-to-subcube割当てにより，各対角ブロックの消去を並列に行うことができる．

なお，オーダリングでは行列を再帰的縁付きブロック対角行列に変形した後，演算量削減のため，各対角ブロック内部でさらにオーダリングを行う必要があるが，本ソルバではこの部分に演算量削減効果の大きい近似MD法[3)]を用いた．これにより，5章で示す$N = 70032$の例題に対しては，各対角ブロック内部もMND法でオーダリングを行った場合に比べ，コレスキー分解のための全演算量を10.5 GFLOPから9.1 GFLOPへ14%削減することができた．

（2）行列の分割

オーダリング後の行列に対して消去木を作成し，subtree-to-subcube割当てにより行列の各列をプロセッサに割り当てる．なおサイクリックに割当てを行う部分では，実際にはブロックサイクリック分割を採用した．ここで，ブロックサイズLの決定に関しては，Lを大きくすると通信単位が大きくなって通信起動のオーバヘッドが削減できる反面，小規模問題でのプロセッサ間の負荷不均等性が大きくなるというトレードオフがある．ここでは，予備的な実験の結果から，$L = 12$というブロックサイズを採用した．

（3）シンボリック分解

シンボリック分解は，原理的にはコレスキー分解において，通常の実数演算の代わりに，要素の値が零か非零かのみに着目して演算を行うことによって実現できる．しかし，この方法ではコレスキー分解と同程度の演算量が必要となるため，本ソルバでは，より効率的な次のアルゴリズム[2)]を利用する．

いま，行列Aの第j列の非零要素の行番号の集合を$\mathrm{Struct}\,(A_{*j})$と表す．また，消去木における節点$i$の親の節点を$p(i)$と表す．このとき，分解後の行列$L$の非零構造$\mathrm{Struct}\,(L_{*j})$は，次の式により計算できることが知られている[2)]．

$$\mathrm{Struct}\,(L_{*j}) = \mathrm{Struct}\,(A_{*j}) \cup (\cup_{i<j}\{\mathrm{Struct}\,(L_{*i}) \mid p(i) = j\}) - \{j\} \quad (2)$$

消去木の各節点に対して，親の節点はただ1つだから，式(2)においてjが1からNまで動くとき，各iに対する$\mathrm{Struct}\,(L_{*i})$は式の右辺に1回しか現れない．したがって，集合の和演算が和をとる集合の要素数の合計に比例する計算量で行えることを考慮すると，式(2)を用いたLの非零構造の計算は，Lの非零要素数に比例する計算量で行えることが分かる．これにより，コレスキー分解に基づく方法に比べ，計算量は大きく削減できる．

式(2)より，列jの非零構造$\mathrm{Struct}\,(L_{*j})$を計算するには，消去木上での$j$のすべての子供に対する非零構造が分かっていればよい．したがって，subtree-to-subcube割当てでは，各プロセッサに割り当てられた部分木上での式(2)の計算は通信なしに行える．一方，ブロックサイクリック分割を行っている部分では，式(2)の右辺に出てくる各子供iがどのプロセッサに割

り当てられているかを判定し，j と別のプロセッサに割り当てられている場合は，そのプロセッサから通信により Struct (L_{*i}) を受け取って計算を行う．

なお，本ステップで作成した L の非零構造は，L の各列の非零要素の行番号を格納したインデックスリストとして保持する．

（4）コレスキー分解

コレスキー分解には，通信量が比較的少ない内積形式を採用した．いま，行列 L の第 k 行の非零要素の列番号の集合を Struct (L_{k*}) と表すと，内積形式における第 k 列の更新演算は次のように書ける[2]．

$$\begin{aligned}&\text{for}\quad j \in \text{Struct}\,(L_{k*})\,\text{do}\\&\qquad \text{cmod}\,(k,j)\\&\text{第 } j \text{ 列全体を } (L_{kk})^{1/2} \text{で除算}\end{aligned}\qquad (3)$$

ここで，for 文中の演算 cmod (k,j) は，L の第 j 列に L_{kj} を掛け，L の第 k 列から差し引く計算である．ただし，実際の計算は結果が非零となる要素のみに対して行う．

いま，節点 k が 1 個のプロセッサに担当される部分木（たとえば図 5 の節点 1〜4）中にあるときは，2.2 節で述べた消去木の性質より，k の更新に使われるすべての列 $j \in$ Struct (L_{k*}) もその部分木中にある．したがって，式 (3) の演算に必要なデータはすべて節点 k を担当するプロセッサが持っている．このため，部分木中での更新演算は，通信なしに各プロセッサが並列に行うことができる．

一方，節点 k が複数個のプロセッサに担当される部分木（たとえば図 5 の節点 9，10）中にあるときは，式 (3) の Struct (L_{k*}) の要素は，その複数のプロセッサ間に分割されている．この場合は，各プロセッサが Struct (L_{k*}) のうち自分の持つ列からの式 (3) への寄与をそれぞれ独立に計算してローカルなワーク配列にためておき，最後にそれを節点 k を持つプロセッサに集計することにより，L の第 k 列に対する更新を行う．なお，この部分木を担当しないプロセッサは処理に関与しないため，自分の担当する部分木について独立に処理を進めることができる．

このようにして，消去木の分岐を下から上に 1 つたどるごとに式 (3) を協調して計算するプロセッサの数は増えてゆき，一番上の分岐より上では，すべてのプロセッサが協調して式 (3) の計算を行う．

例として図 5 の消去木の場合を取り上げると，節点 1〜4，5〜8，11〜14，15〜18 の更新演算はそれぞれプロセッサ 0，1，2，3 が独立に行うが，節点 9，10 の更新および節点 19，20 の更新は，それぞれプロセッサ群 $\{0,1\}$，$\{2,3\}$ が協調して行う．さらに，節点 21〜25 の更新は，4 台のプロセッサすべてが協調して行う．このように，ここで採用したコレスキー分解の並列化手法においては，消去木の葉に近い部分では主として部分木単位の並列性を用い，根の近くでは，主として式 (3) における j に関する並列性を用いて計算を行う．

（5）前進後退代入

消去木を用いて考えると，前進消去は消去木の葉の部分から根に向かって順に解を求めてゆく計算，後退代入は逆に根から葉に向かって順に解を求めてゆく計算となる．

前進消去で方程式 $Ly = b$ を解く際には，すでに求めた解の要素 y_j を用いて別の解の要素 $y_i\,(i > j)$ を更新する演算 $y_i := y_i - L_{ij}y_j$ が中心演算となる（ここで，右辺ベクトル b は解 y によって上書きされると仮定する）．このための主な計算方法としては，j に関するループを内側に持ってくる内積形式と，i に関するループを内側に持ってくる外積形式とがあり，本ソルバでは，行列 L の非零要素を各列 j の非零要素の行番号 i のリストにより管理していることから，内側ループが i の外積形式が適している．しかし外積形式による計算をそのまま行うと，各ステップで求めた解 y_j を用いて，消去木上で祖先にあたる解をすべて更新することになる．これらの解は一般に他のプロセッサが担当しているため，各ステップ j ごとに通信が必要になってしまう．

そこで本ソルバでは，消去木上で祖先にあたる解への更新を直接には行わず，更新分を各プロセッサのワーク配列にためこんでおき，自分が担当する部分木中の解の計算がすべて終了した時点で，他のプロセッサに送る．これにより，各部分木での計算は通信なしに実行できる．

一方，後退代入で方程式 $L^t x = y$ を解く際には，演算 $x_i := x_i - L_{ji}x_j$ が中心演算となるため，内側ループが j の内積形式が適している．内積形式では，各ステップ i での解の計算のため，消去木上で節点 i の祖先にあたる解が必要となるが，各プロセッサが担当する部分木中の解の計算を始める前に，それまでに求めた解をすべてそのプロセッサに転送しておくことにより，各部分木での計算は通信なしに実行できる．

4. RISC 向け最適化

前章で述べたスパースソルバの処理のうち，最も計算量の多いのはコレスキー分解である．コレスキー分解の中心演算式 (3) は，**図 6** のように小行列 A_k とベクトル $\boldsymbol{b}_k$ との積を計算してベクトル $\boldsymbol{c}_k$ から引く演算 $\boldsymbol{c}_k = \boldsymbol{c}_k - A_k\boldsymbol{b}_k$ となる．ただし実際には，分割のブ

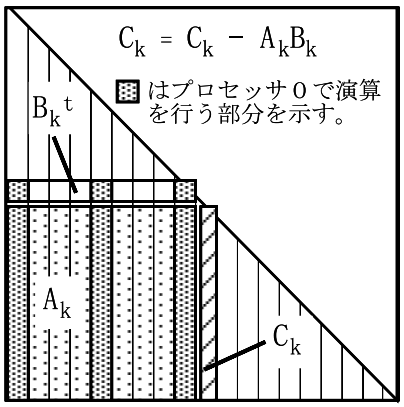


図 6 コレスキー分解の中心演算
Fig. 6 Kernel operation of the Cholesky decomposition.

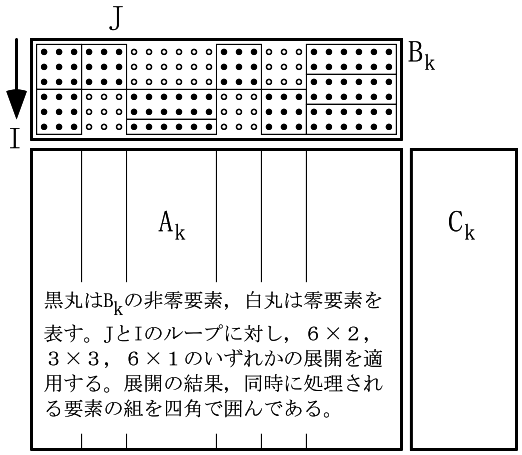


図 7 局所的な非零構造に基づくループ展開
Fig. 7 Optimized loop unrolling for the local nonzero structure of the matrix.

ロック化にともなってコレスキー分解の多段化を行っているため，$\boldsymbol{b}_k$，$\boldsymbol{c}_k$ は幅がブロックサイズと等しい行列 B_k，C_k となり，演算は行列乗算 $C_k = C_k - A_k B_k$ となる．

この計算において，A_k，B_k，C_k はすべて疎行列である．そのため，インデックスリストを介してアクセスを行わなくてはならず，密行列演算の場合に比べて一般に計算機の実効性能は低下する．さらに RISC プロセッサの場合には，ロード/ストアを削減して性能を引き出すために不可欠なループ展開に関しても困難が生じる．疎行列の乗算では，ループの各繰返しごとに非零要素の位置や数が異なるため，通常のループ展開を機械的に適用すると，本来零である要素に対しても余分な演算を行ってしまうことになり，かえって演算効率が落ちてしまうからである．

そこで本ソルバでは，行列の不規則な非零構造に合わせて局所的に最適なループ展開を行うことで，RISC プロセッサ上での性能向上を図った．3 次元構造解析では行列の非零構造が 3×3 のブロック単位で現れる．そこで，**図 7** に示すように行列 B_k の非零構造を 6×2，3×3，6×1 の 3 種類のブロックの直和に分割し，それぞれに応じた局所的なループ展開を適用すれば，演算量を増やさずにループ展開が行える．この 3 種類の展開はこの順に性能が高いため，6×2 のブロックをできるだけ多くとり，残りを 3×3 と 6×1 に分割した．なお，この分割処理はシンボリック分解のフェーズを利用して行うため，同じ非零構造を持つ行列を複数回解く場合には，分割のためのオーバヘッドは無視できる．

5. 性能評価

本研究で開発したスパースソルバについて，分散メモリ型並列計算機 Hitachi SR2201 上で次元数 $N = 70032$ の 3 次元構造解析の実例題を用いて評価を行った．プロセッサ台数 p は 4，8，16 の 3 通りについて実行した．なお，$p = 1, 2$ については，分解後の行列を格納するためのメモリが不足し，同じ例題が実行できなかったため，1 回り小さな $N = 32274$ の例題を用いてコレスキー分解部分の評価を行った．

表 1 ソルバの各部分の実行時間（秒）
Table 1 Execution time of each part of the sparse solver (in seconds).

	$p = 4$	$p = 8$	$p = 16$
シンボリック分解	0.55	0.48	0.29
コレスキー分解	23.86	16.17	9.68
前進後退代入	0.65	0.57	0.35
合計	25.06	17.22	10.32

（1）プロセッサ台数を変えた場合の性能

本ソルバにおいて分散メモリ型並列計算機向けの並列化を行ったシンボリック分解，コレスキー分解，前進後退代入の部分について，プロセッサ台数を変えたときの実行時間を**表 1** に示す．なお，コレスキー分解の実行時間は，前章で述べた RISC 向け最適化を行った場合の時間である．

表より，p を 4 から 16 に増やした場合，コレスキー分解部分では約 2.5 倍の加速が得られるが，シンボリック分解，前進後退代入では 2 倍弱の加速しか得られていないことが分かる．これは，シンボリック分解と前進後退代入ではコレスキー分解に比べて計算量は大幅に少ない（分解後の行列の非零要素の個数のオーダー）が，必要な通信回数のオーダーはコレスキー分解と同程度であることによると考えられる．ただし，

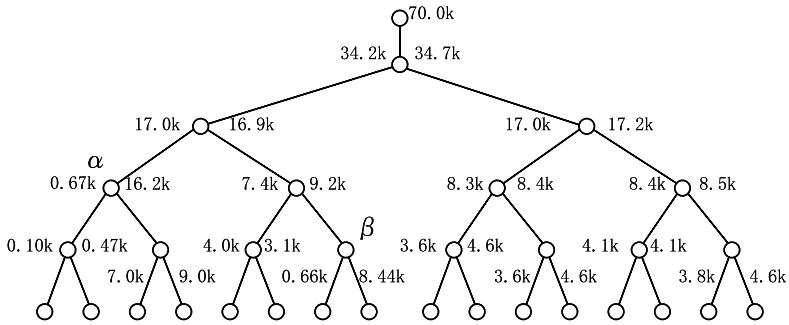


図 8 3 次元構造解析の実例題に対する消去木

Fig. 8 The elimination tree to the real 3-dimensional structural analysis problem.

計算量が少ないため，これらのステップが全実行時間に及ぼす影響は小さい．

本例題では p を 4 から 16 へと 4 倍に増やした場合，コレスキー分解部分の性能は約 2.5 倍となっているが，ここで性能向上を抑えている要因を調べるため，消去木を用いた分析を行った．本例題に対する消去木を**図 8** に示す．ただし図には分岐節点のみを記し，また消去木が 16 個の部分木に分かれた以降の部分は省略した．図中の数字は，各分岐節点において，右側の部分木と左側の部分木とに含まれる節点の個数（単位は 1000 個）を表す．図より，最初の 2 段の分岐（プロセッサ台数が 4 までの分割に対応）では 70032 個の全節点がきわめて均等に分割されているが，第 3 段目で，α と記した分岐に大きな不均等が生じている．これにより，$p = 8$ では負荷最大のプロセッサの担当する消去木の節点数は 16194 個と他のプロセッサの 2 倍近くになり，$p = 4$ から 8 にかけて並列化効率の大きな劣化が生じると考えられる．第 4 段目でも β と記した分岐に大きな不均等が生じているが，全体として見ると，$p = 16$ のときの負荷最大のプロセッサの担当する消去木の節点数は 9043 個であり，$p = 8$ のときの約半分になっている．したがって，$p = 8$ から 16 にかけての並列化効率の劣化はそれほど大きくないと考えられる．

以上の考察と表 1 のコレスキー分解の実行時間を照らし合わせてみると，表 1 で $p = 4$ から 8 にかけての性能向上は 1.47 倍，$p = 8$ から 16 にかけては 1.67 倍であり，各部分木の演算量に基づく並列化効率の推定と定性的に一致している．一方，通信のオーバヘッドのみを考えると，$p = 4$ から 8 にかけての性能向上の割合は $p = 8$ から 16 にかけての性能向上の割合よりも大きいはずである．したがって，本例題で $p = 4$ から 8 にかけての性能向上を抑える主な要因は，通信オーバヘッドよりもむしろ部分木間の演算量の不均等さにあると推定できる．

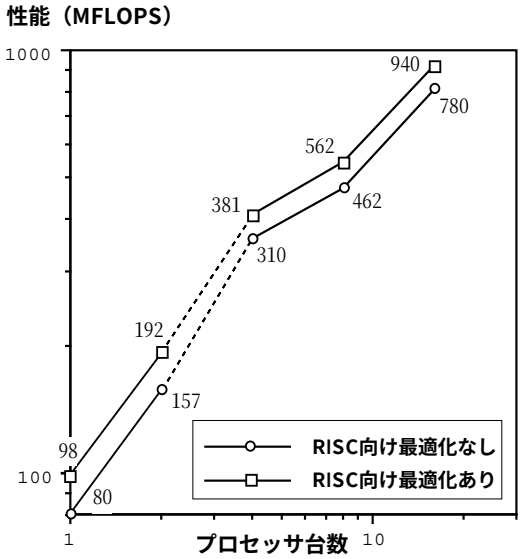


図 9 3 次元構造解析での評価

Fig. 9 Evaluation by the 3-dimensional structural analysis problem.

なお，図 8 の分岐節点 α におけるこの不均等さは，本例題でのメッシュ形状が不規則で均等分割が難しいために生じた問題であり，現在の MND 法では解消することが困難であった．このような場合にも自動で均等な分割ができるように MND 法を改良すること，あるいは subtree-to-subcube 割当てを一部変更し，図 8 に示したよりもさらに下位の部分木構造を用いて各プロセッサへの割当て節点数を均等化することは，今後の課題である．

（2）RISC 向け最適化の効果

次に，RISC 向け最適化を行った場合と行わなかった場合について，コレスキー分解部分の性能評価結果を**図 9** に示す．ただし，問題サイズは $p = 1$，2 の場合が $N = 32274$，$p = 4$，8，16 の場合が $N = 70032$

である．最適化を行った場合，1プロセッサでの性能は約100 MFLOPSと，最適化なしの場合に比べて約20%向上した．また，MFLOPS値で比較すると，16プロセッサでの実行時には，1プロセッサに比べ，約10倍の加速が得られた．

なお，本ソルバの中心演算である行列乗算部分で採用した3種類のループ展開パターンのうち，最も性能の高い 6×2 の展開の1プロセッサでの性能は166 MFLOPSであった．これは，SR2201でインデックスリストを用いた参照が存在する場合の，行列乗算に対するハードウェアの限界性能と考えられる．ここで提案したRISC向け最適化手法では，中心演算以外の部分も含めたコレスキー分解全体の性能において，この限界性能の60%を達成することができた．

6. おわりに

本研究では，分散メモリ型並列機向けのスパースソルバを開発し，評価を行った．本ソルバは対称正定値でかつ非零構造が 3×3 のブロック単位で現れる行列を対象として，行列のオーダリングから，プロセッサ間への分割，シンボリック分解，コレスキー分解，前進後退代入までを一貫して行うプログラムである．計算の中心となるコレスキー分解においては，局所的なループ展開により，RISC向けの最適化を行った．SR2201上の評価では，比較的小規模な約32000〜70000元の3次元構造解析の問題で1プロセッサでの性能約100 MFLOPS，16プロセッサ時に約10倍の加速率を達成し，実用的な性能が得られることを確認した．

今後の課題としては，シンボリック分解や前進後退代入を含めたスパースソルバ全体の最適化，並列計算機SR8000向けの最適化と性能評価，CAPSS[9]，PARASOL[10),12)]，PSPASES[11)]など他のソルバとの性能比較があげられる．

謝辞 日頃からご指導いただいている（株）日立製作所中央研究所の稲上泰弘博士，伊藤智博士，および同ソフトウェア開発本部の後藤志津雄HPC推進部長，五百木伸洋主任技師に感謝申し上げます．また，多くの貴重なご助言をくださった査読者の皆様に感謝いたします．

参考文献

1) Duff, I.S., Erisman, A.M. and Reid, J.K.: *Direct Methods for Sparse Matrices*, Oxford University Press (1986).
2) Gallivan, K.A., et al.: *Parallel algorithms for Matrix Computations*, SIAM (1990).
3) Davis, T.A., Amestoy, P. and Duff, I.S.: An Approximate Minimum Degree Ordering Algorithm, *SIAM Journal on Matrix Analysis and Applications*, Vol.17, No.4, pp.886–905 (1996).
4) Boman, E.G. and Hendrickson, B.: A Multilevel Algorithm for Reducing the Envelope of Sparse Matrices, Technical Report SCCM-96-14, Stanford University (1996).
5) Karypis, G. and Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, Technical Report TR95-035, Department of Computer Science, University of Minnesota (1995).
6) Pothen, A., Simon, H. and Liou, K.: Partitioning Sparse Matrices with Eigenvectors of Graphs, *SIAM Journal on Matrix Analysis and Applications*, Vol.11, pp.430–452 (1990).
7) Gupta, A. and Kumar, V.: Parallel Algorithms for Forward and Backward Substitution in Direct Solution of Sparse Linear Systems, *Proc. Supercomputing '95* (1995).
8) Gupta, A., Karypis, G. and Kumar, V.: Highly Scalable Parallel Algorithms for Sparse Matrix Factorization, *IEEE Trans. Parallel and Distributed Systems*, Vol.8, No.5, pp.502–520 (1997).
9) Heath, M.T. and Raghavan, P.: Performance of a Fully Parallel Sparse Solver, *The International Journal of Supercomputer Applications and High Performance Computing*, Vol.11, No.1, pp.49–64 (1997).
10) Amestoy, P. and Duff, I.S.: The PARASOL Project and the Multifrontal Parallel Solver for Sparse Systems, *Proc. 9th SIAM Conference on Parallel Processing for Scientific Computing*, SIAM (1999).
11) http://www-users.cs.umn.edu/~mjoshi/pspases/
12) Amestoy, P., Duff, I.S. and L'Excellent, J.-Y.: MUMPS MUltifrontal Massively Parallel Solver Version 2.0, Technical Report TR/PA/98/02, CERFACS (1998).

(平成11年9月2日受付)
(平成12年2月4日採録)

山本　有作（正会員）

1966 年生．1990 年東京大学工学部計数工学科（数理工学コース）卒業．1992 年同大学院工学系研究科物理工学専攻修士課程修了．同年（株）日立製作所中央研究所入所．以来，並列計算機 SR2001，SR2201，SR8000 向け行列計算ライブラリの研究開発に従事．大規模疎行列に対する固有値解法，連立一次方程式解法，およびその応用に興味を持つ．

猪貝　光祥

1963 年生．1987 年横浜市立大学文理学部物理課程卒業．同年現（株）日立超 LSI システムズ入社．以来，科学技術計算用ソフトウェアおよびその並列化手法に関する研究開発に従事．

直野　　健（正会員）

1968 年生．1992 年京都大学理学部数学科卒業．1994 年同大学院理学研究科数理解析専攻修士課程修了．同年（株）日立製作所中央研究所入所．以来，並列計算機 SR2201，SR8000 向け行列計算ライブラリの研究開発に従事．特に並列固有値計算に興味を持つ．日本応用数理学会員．