OSCAR 上でのスパース行列直接解法の並列処理†

笠原博徳†† ウィチェン プレムチャイサワディ††
田村光雄†† 前川仁孝†† 成田誠之助††

本論文では，従来並列処理が困難であった構造化されていないスパース線形方程式求解に対する細粒度タスクを用いた並列処理手法を提案すると共に，その性能評価をマルチプロセッサシステム OSCAR (Optimally SCheduled Advanced MultiprocessoR) 上で行った結果について述べる．直接解法を用いたスパース線形方程式求解では電子回路シミュレータで使用されるループフリーコードが最も高速であることが知られている．提案する手法では，専用目的コンパイラがこのループフリーコードを自動生成した後，細粒度タスクへ分割し，それらのタスクをプロセッサ間のデータ転送オーバヘッドを考慮してプロセッサにスケジューリングする．次にコンパイラはデータ転送オーバヘッドや同期オーバヘッドが最小となるように並列化マシンコードを生成する．この並列化マシンコードは，OSCAR の各プロセッサ上のローカルメモリにロードされ，実行される．このようなループフリーコードの細粒度レベルでの並列処理が，実マルチプロセッサ上で実現されたのは世界初である．

1. はじめに

構造化されていない (unstructured) スパース係数行列を持つ線形方程式の高速な求解は，電子回路解析や電力システム潮流計算，過渡安定度解析など様々なアプリケーション分野で必要とされている．

このようなスパース行列に対する解法としては，従来よりLU分解を伴った直接解法がよく用いられてきた[1)-3),23)]．

直接解法を使用した回路シミュレーションでは，求解時間を最小化するために，ループフリーコードを生成するコード生成法[1),4)]と，フィルインを少なくするリオーダリング[11),13)]がよく用いられる．ただし，コード生成法では，生成されたコードを格納するために大きなメモリ空間が必要となるという問題があることが知られている[2),3)]．しかし，単純な Fortran による求解プログラムに比べて，ループフリーコードを用いた方式は，処理時間を顕著に短縮することができる[3),12)]．そのため，最近ではベクトルパイプラインマシンやマルチプロセッサマシン上でのループフリーコードの並列実行手法が，多く提案されている[6),7),9),10)]．

ループフリーコード，すなわち算術代入文のみからなるスカラ計算部分を効果的にマルチプロセッサ上で処理するためには，数個の浮動小数点演算から構成される細粒度タスクの並列処理を行わなければならない．

しかし，このループフリーコードの細粒度並列処理は，実際には非常に困難である．細粒度タスクを用いたスパース行列求解の理論的な処理手法として，Wing と Huang が，ループフリーコードから生成される細粒度タスクの並列処理のための方式を提案している[6),7)]．しかし，細粒度タスクをプロセッサに最適に割り当てるというマルチプロセッサスケジューリング問題は“強” NP 困難であるため，彼らはすべてのタスク処理時間が同じであり，プロセッサ間のデータ転送が無視できる場合に限定してスケジューリング問題を解いた．しかしながら，細粒度タスクの並列処理を実際のマルチプロセッサ上で行う場合，直接法におけるデバイド・オペレーションとアップデート・オペレーションが常に1クロックで実行できるという仮定は非現実的であると共に，データ転送オーバヘッドを考慮していないために効率の良い並列処理は期待できない．

以上のことを考慮して，本論文ではヒューリスティックスケジューリングアルゴリズム CP/DT/MISF 法を用いた並列化コンパイレイション手法を提案する．本手法では各々が異なる処理時間を持つタスクをデータ転送オーバヘッドと同期オーバヘッドを考慮に入れプロセッサにスケジューリングすることが可能となる．また，本手法の有効性は，マルチプロセッサシステム OSCAR 上で確認される．

† Parallel Processing of Direct Solution Method for Unstructured Sparse Matrices on OSCAR by HIRONORI KASAHARA, WICHIAN PREMCHAISWADI, MITSUO TAMURA, YOSHITAKA MAEKAWA and SEINOSUKE NARITA (Department of Electrical Engineering, School of Science and Engineering, Waseda University).

†† 早稲田大学理工学部電気工学科

2. OSCAR のアーキテクチャ

OSCAR は Optimally SCheduled Advanced MultiprocessoR の略であり，そのアーキテクチャを**図1**に示す．OSCAR は，共有メモリと分散メモリを持った共有メモリ型のマルチプロセッサであり，最大16台のプロセッサエレメント (PE) と，1台のコントロール & I/O プロセッサ (CIOP) を3本のバスで3つの共有メモリ (CM) に接続している．

各 PE は 5MFLOPS の能力を有する 32 ビット RISC プロセッサであり，プログラムメモリ，データメモリ，分散共有メモリとして使われるデュアルポートメモリ (DPM) を有している．以下では，各プロセッサ内のデータメモリとプログラムメモリをローカルメモリ (LM) と表現する．バスの最大データ転送速度は，毎秒 60 メガバイトであり，PE による各バスへのアクセスは，各 PE に設定された優先度に基づいて制御される．

OSCAR のアーキテクチャは，スタティックスケジューリングを使用し細粒度タスクの並列処理が効果的に行えるように設計されている．例えば，OSCAR の RISC プロセッサは，浮動小数点の加算，減算，乗算を含めたすべての命令を1クロックで実行する．また，それらを組み合わせた浮動小数点演算も一定のクロック数で実行することができる．実際のマルチプロセッサシステム上でスタティックスケジューリングを行う場合，タスクの実行時間の正確な推定は一般的には困難であるが，OSCAR では1命令の処理を確実に1クロックで行えるため簡単かつ正確にコンパイル中に行うことができる．さらに，各 PE の DPM すなわち分散共有メモリは，1つの PE から他の1つの PE への転送に使う直接データ転送と，1つの PE から全 PE におくるブロードキャストデータ転送という2種類のデータ転送方式のために使用できる．また，この2種類に加え OSCAR は，CM を介した PE 間の間接データ転送もサポートしている．DPM を使った直接データ転送は，DPM 上にデータを1回書き込むだけで自分の PE から他の PE へ32ビットデータを転送することができる．しかし，CM を使った間接データ転送は，CM へのデータの書き込みと CM からのデータの読み込みが必要であり，最低計2回のデータ転送が必要である．またデータブロードキャスト転送は，CM を使った複数回のデータ転送を行うのに比べて，データ転送時間を顕著に短縮することができる．したがって，データ転送のオーバヘッドを減らすためには，スタティックスケジューリングにおいて3種類のデータ転送を使い分けることが重要である．特に本手法ではスタティックスケジューリングを用いて PE 間の直接データ転送を効果的に使用することが可能である．

3. スパース線形方程式の直接解法の並列処理

本章では，OSCAR 上でスパース線形方程式を効率良く解くための並列処理手法について述べる．

3.1 スパース行列の直接解法

本論文では，$Ax=b$ (ただし，A 行列はスパース行列であり，必ずしも帯行列あるいはブロック対角行列である必要はない) という線形方程式をガウス消去法

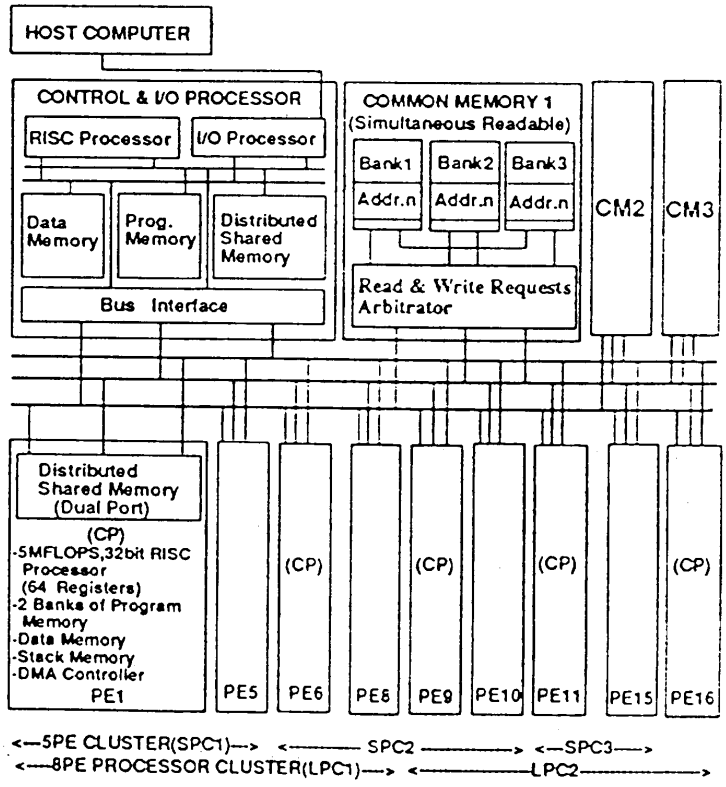


図 1 OSCAR のアーキテクチャ
Fig. 1 OSCAR's architecture.

やクラウト法などのような直接法を用いて解く問題を取り扱う．直接法を用いてスパース行列を解く時には第1章で述べたようにリオーダリングやコード生成手法が，処理時間を最小化するために使われる．

例えば，電子回路の過度解析においては，リオーダリングすなわち1回のみのプレオーダリングが，LU分解中のピボッティングの代わりとして使われている．このようなリオーダリングは，ロバスト性を失うことなく計算時間を大きく減少させることが従来より知られている[9]．リオーダリングにおいては，フィルインを減らすことにより計算量を少なくし，シーケンシャル処理時間を減少させるリオーダリング[11),13)]や，LU分解時の並列性を高めるリオーダリング[7)]が提案されており，並列処理においてどちらを利用すべきかは難しい問題である．しかし後者の並列性を高めるリオーダリングは，並列性を高めるメリットに比べて全体の計算時間が増えるデメリットが大きいため，前者のフィルイン最小化リオーダリングの方がより並列処理に効果的であることが知られている[9]．そこで，本論文ではフィルイン最小化リオーダリングを使用する．

また，ループフリーコードを生成するコード生成法は，長年にわたり電子回路のシミュレーションで使われてきた[1),3)]．コード生成法は，第1章でも述べたように，大きなメモリ容量を必要とするが直接法による通常の Fortran のプログラムと比べ，大幅に計算時間を短縮する[1),3)-5),10)]．以上のことを考慮して，提案する手法では，回路分割[8)]により生成された小回路の解析を目標として，コード生成法により生成されたループフリーコードの並列処理を行う．

3.2 並列化コンパイレイション手法

提案する並列化コンパイレイション手法は，タスク生成，タスクグラフ生成，タスクスケジューリング，マシンコード生成の4つのステップから成る．以下ではこれらのステップについて詳述する．

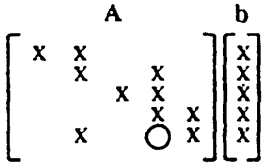
3.2.1 タスク生成

本論文では，まず最初に，スパース行列求解用専用目的コンパイラが，与えられたスパース線形方程式に対してリオーダリングとクラウト法を用いたコード生成法を適用して，図2に示すようなループフリーコードを自動的に生成する．図2のコードは A 行列の求解プログラムを，コード生成法を用いて生成したものである．しかし，これは，説明用に Fortran ライクな表現で書いたサンプルプログラムとなっているが，実際のインプリメントでは，専用の中間言語を用いて生成される．次に，コンパイラは，コード（プログラム）を，プロセッサエレメント（PE）に割り当てるための基本的な単位であるタスクに分割する．このタスク分割においては，1つの浮動小数点演算，1つの代入文，複数代入文というようないくつかの異なったタスク粒度から最適な粒度を選択しなければならない．最適なタスク粒度決定はプロセッサの演算処理能力，プロセッサ間データ転送能力，同期オーバヘッド，タスクスケジューラの能力など，使用するマルチプロセッサシステムに関係した様々な要因を考慮に入れて注意深く選択しなければならない難しい問題である．

本論文では，代入文レベルの粒度，すなわち近細粒度タスクが，OSCAR の処理能力およびデータ転送能力を考慮に入れて選択している．図2の例では，複数の浮動小数点演算を含むステートメントタスクとして定義している．したがってこの例では17個のタスクが LU 分解，前進代入，後退代入の処理のために生成されている．

3.2.2 タスクグラフ生成

生成されたタスク間にはフロー依存や出力依存，逆



X = Nonzero
◯ = Fill-in

<< LU Decomposition >>

1) $u_{12} = a_{12} / l_{11}$
2) $u_{24} = a_{24} / l_{22}$
3) $u_{34} = a_{34} / l_{33}$
4) $l_{64} = -l_{52} * u_{24}$
5) $u_{45} = a_{45} / l_{44}$
6) $l_{55} = a_{56} - l_{54} * u_{45}$

<< Forward Substitution >>

7) $y_1 = b_1 / l_{11}$
8) $y_2 = b_2 / l_{22}$
9) $b_5 = b_5 - l_{52} * y_2$
10) $y_3 = b_3 / l_{33}$
11) $y_4 = b_4 / l_{44}$
12) $b_5 = b_5 - l_{54} * y_4$
13) $y_5 = b_5 / l_{55}$

<< Backward Substitution >>

14) $x_4 = y_4 - u_{45} * y_5$
15) $x_3 = y_3 - u_{34} * x_4$
16) $x_2 = y_2 - u_{24} * x_4$
17) $x_1 = y_1 - u_{12} * x_2$

図2 スパース行列の解法の細粒度タスクによるコード生成の例
Fig. 2 An example of generated fine grain tasks for sparse matrix solution.

依存のようなデータ依存が存在する[14)~16)]．データ依存すなわち先行制約は，**図3**に示されるような“タスクグラフ”[17),19)]によって表すことができる．図中，各ノードはタスクに対応しており，ノード内側の数字はタスク番号 i を，ノード横の数字はタスク処理時間 t_i を表している．ノード N_i から N_j に向かうエッジはタスク T_i がタスク T_j に先行するという部分的順序制約を表している．また図3において入口ノードのノード N_0 と出口ノードのノード N_{18} はプログラム開始終了を表すためのダミーノード（処理を含まないノード）である．またこのタスクグラフ上で N_i からタスクグラフの出口ノードまでの最長パス長を各タスクのレベル l_i と表現することとする．図3のタスク T_{14} の場合，出口ノードまでのパスは T_{14}-T_{15}-T_{18} および T_{14}-T_{16}-T_{17}-T_{18} の2つでパス長は20および30であるからレベル l_{14} は30になる．また，タスク間のデータ転送を考慮に入れる場合は各々のエッジには可変の重みが付けられる．その重み t_{ij} は，もしタスク T_i と T_j が異なる PE に割り当てられるなら T_i と T_j の間のデータ転送時間になり，同じ PE に割り当てられているなら，ゼロあるいはレジスタやローカルメモリへのアクセス時間となる．

3.2.3 スタティック・スケジューリング・アルゴリズム

マルチプロセッサシステム上で一連のタスクを効率良く処理するためには，タスクのプロセッサへの割り当て法と，同一プロセッサに割り当てられたタスクの実行順序の決定が最適になされなければならない．この最適な割り当てと実行順序を決定する問題は，並列処理時間最小すなわちスケジュール長最小を目的とする伝統的なマルチプロセッサスケジューリング問題として扱うことができる[17),19),21),22)]．より厳密に定義するとこのスケジューリング問題は，n 個のタスクとその計算に必要な処理時間，タスク間の先行関係が与えられた時，それらのタスクを同処理能力の m 台のプロセッサで最小時間で並列処理するスケジュールを決定するという問題である．しかし，このスケジューリング問題は，強 NP 困難な問題として知られている[18)]．換言すると，もし P=NP でないならば，疑似多項式時間最適化アルゴリズムだけでなく，両完全多項式時間近似スキームすらも構築できないということを表している．したがってこの問題を解くために，様々なヒューリスティックアルゴリズムと，実用的な最適化アルゴリズムが対抗策として提案されている[17),19),21)]．

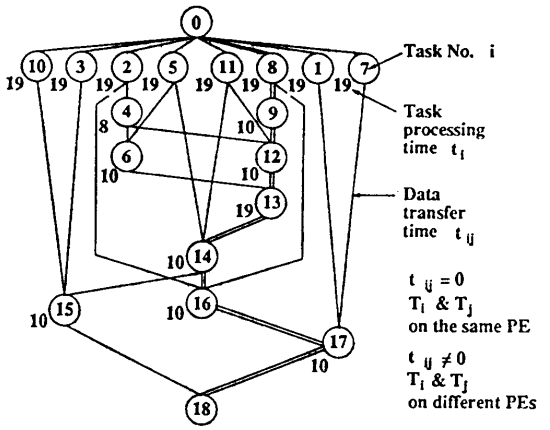


図3 図2のタスクグラフ
Fig. 3 Task graph of Fig. 2.

しかし，本論文で扱う細粒度タスクの並列処理では，計算処理に比べてデータ転送の割合が大きいため，データ転送によるオーバヘッドを無視できない．

従来，この種のスケジューリング問題において，プロセッサ間データ転送時間が無視できる場合に対する実用的なアルゴリズムとしては，CP/MISF[19),20)] と呼ばれる手法が使用されていた．そこで本コンパイレイション手法では CP/MISF の改良バージョンであり，データ転送を考慮したヒューリスティックスケジューリングアルゴリズム CP/DT/MISF をスケジューリングに要する時間と生成されたスケジュールの質を考慮に入れて採用している．

ここで提案する CP/DT/MISF はリストスケジューリングアルゴリズムの一種であり各タスクの優先順位（プライオリティ）をグラフ形状などから決定し，アイドルプロセッサ（その時点で使用可能なプロセッサ）に，レディタスク（その時点で実行可能なタスク）のプライオリティの高いものから順に割り当てていく方式である．

この時，プライオリティ決定にクリティカルパス長 (CP)，データ転送に要する時間 (DT)，直接後続タスク数 (MISF) と呼ぶ3種のプライオリティ決定法を用いるリストスケジューリングアルゴリズムを CP/DT/MISF と呼んでいる．

具体的には CP/DT/MISF は，以下の手順から成る．

Step 1 各タスクのレベル l_i を決定する．

Step 2 データ転送を考慮したリストスケジューリングを実行する．

Step 2.1 スケジューリング時刻 $t_{current}$ を0に

する．$t_{current}$ におけるレディータスク，アイドルプロセッサを見つける．

Step 2.2 最も高いレベルを持つレディタスク（n_{ready} 個）をアイドルプロセッサ（m_{idle} 個）に割り当てる可能な組み合わせ（$n_{ready}*m_{idle}$ 個の組み合わせ）に対してデータ転送時間の計算を行う．このデータ転送時間の計算においては $t_{current}$ 以前に割り当てられたタスクのスケジュールを考慮に入れて，データが同一 PE 内のローカルメモリ上あるいは分散共有メモリ上にある場合にはそこからロードし，他PE 上のローカルメモリ上にあれば分散共有メモリを用いてデータ転送を行うものとしてデータ転送時間を算出する．

Step 2.3 最小のデータ転送時間を得るようなレディタスクのアイドルプロセッサへの割り当てを選ぶ．そのような割り当てがいくつもあれば直接の後続タスクが最も多いタスクを優先する．アイドルプロセッサがなくなれば Step 2.4 に進み，そうでなければ Step 2.3 を繰り返す．

Step 2.4 次のスケジューリング時刻 $t_{current}$ を見つける．これは，少なくとも1つのプロセッサが，すでに割り当てられたタスクの実行を終了して，アイドル状態になる最も早い時刻である．その $t_{current}$ におけるレディタスクとアイドルプロセッサを見つける．1つ以上のレディタスクがあれば Step 2.2 に進み，そうでなければすべてのプロセッサの実行が終了するまで Step 2.4 を繰り返す．すべてのプロセッサが実行を終了した時点でスケジューリングを終了する．

図4は Step 2.3 によるレディタスクのアイドルプロセッサへの割り当ての例を示している．図4（a）は，$t_{current}$ における4つのレディタスク T_A, T_B, T_C, T_D でありそれぞれのレベルが 100, 80, 80, 80, 直接の後続タスク数が 2, 4, 3, 2 であることを示している．まず最初に最も高いレベルを持つ T_A が割り当てられる．図4（b）は T_A をアイドルプロセッサ PE1, PE2, PE3 に割り当てた場合のデータ転送時間を示している．ここでは T_A は PE2 に割り当てられる．これは，T_A の PE2 への割り当てが最小のデータ転送時間2クロックしか要さないからである．図4（c）は同じレベルを持つ残りのレディタスクを残りのアイドルプロセッサに割り当てた場合のデータ転送時間を示している．この場合にもデータ転送時間が最小になるように T_D の PE1 への割り当てが行われる．最後のアイドルプロセッサ PE3 へは2つのレディタスク T_B, T_C のうち T_B が割り当てられる．どちらの場合でもデータ転送時間は同じであるが，T_B の方が T_C よりも直接後続タスクが多いからである．T_C の割り当てはこの $t_{current}$ では行われない．

また，この CP/DT/MISF 法の性能を，乱数を用いて生成したタスクグラフ 1,512 例に対して適用し評価した結果を**表1**に示す．

ここでタスクグラフはタスク数 100, 200, 400 に対して以下のような乱数を用いて生成した．

処理時間

平均 50　標準偏差 25 または 10 の正規乱数

先行タスク数

平均1　標準偏差 0.5 または 1.0 の正規乱数

平均2　標準偏差 1.0 または 2.0 の正規乱数

平均3　標準偏差 1.5 または 3.0 の正規乱数

レディタスク	A	B	C	D
レ　ベ　ル	100	80	80	80
後続タスク数	2	4	3	2

（a）レディタスクとプライオリティ
（a）Ready tasks and their priorities.

Idle PE \ Task	A
1	5
2	2
3	3

（b）T_A のデータ転送時間
（b）Data transfer time of T_A.

Idle PE \ Task	B	C	D
1	10	6	4
3	5	5	8

（c）T_B, T_C, T_D のデータ転送時間
（c）Data transfer time of T_B, T_C and T_D.

T_A→PE2　T_D→PE1　T_B→PE3

（d）レディタスクのアイドルプロセッサへの割り当て
（d）Assignment of ready tasks to idle processors.

図4 CP/DT/MISF におけるタスク割り当て過程
Fig. 4 Task assignment process in CP/DT/MISF.

表 1 CP/DT/MISF の性能 α (FIFO との比較 [%])
Table 1 Performance of CP/DT/MISF (compared with FIFO).

プロセッサ数	CP/MISF	CP/DT/MISF
2	0.455	1.266
3	1.456	3.246
4	3.302	6.001
5	6.077	9.458
6	9.233	12.919
7	11.475	15.671
8	13.588	17.910
平　均	6.513	9.497

テストケース数 1,512 例
$\alpha=(T_{FIFO}-T_{SCHE})/T_{FIFO}\times 100$ [%]
T_{FIFO}: FIFO の平均スケジュール長
T_{SCHE}: 各アルゴリズムの平均スケジュール長

データ転送時間
平均 5　標準偏差 2.5 の正規乱数
5 または 10 の定数

また，スケジューリングにおいてプロセッサ数は 2 台から 8 台で行った．

評価の結果 CP/DT/MISF の平均スケジュール長は FIFO タイプのスケジューリング，すなわちタスクに優先順位を割り当てないデータ駆動型のスケジューリングより約 10% 程度処理時間を短縮できること，および，データ転送時間がタスク処理時間の 5%～10% 程度でも，CP/DT/MISF は CP/MISF より平均で 3%程度よい性能を示すことが確認された．また，CP/DT/MISF の時間複雑さは $O(n^3m)$ であり，約 1,000 個のタスクをスケジュールするのに通常のワークステーション上で数秒程度しか要しない．

3.2.4 並列マシンコード生成

実マルチプロセッサシステム上で効率良い並列処理を行うためには，スケジューリング結果を用いて，同期オーバヘッドおよびデータ転送オーバヘッドを最小とするように最適化した並列マシンコードを生成しなければならない．スタティックスケジューリングにより与えられる情報は，各 PE 上で実行すべきタスクの集合，同一 PE 上で実行されるべきタスクの実行順序，プロセッサ間データ転送を必要とするタスクの組とそのデータ転送が行われるタイミング，同期の必要な箇所などである．以上の情報を十分に活用することにより，様々な実行時オーバヘッドを最小化し，最小並列処理時間を与えるマシンコードを生成することが可能となる．

例えば，タスクの割り当てと実行順序についての情報を使えば，PE 間でデータ転送を行う場合に，分散共有メモリを使用した 1PE 対 1PE の直接データ転送モード，1PE 対全 PE のブロードキャストデータ転送モードを正しく使い分けることができる．これらのデータ転送モードはタスク間の同期のためにも使用することができ，従来の集中型共有メモリを使用する場合に比べてオーバヘッドを大幅に減少させることができる．分散共有メモリ上に同期フラグ領域を割り当てた場合，各 PE はその PE 内の分散共有メモリ上にある同期フラグをチェックすることができるのでビジーウェイトのために外部バスを使用する必要がなくなる．したがって従来のように集中型共有メモリを使用する場合に問題となった同期のためのビジーウェイトによるバスバンド幅の低下を回避できる．さらに，スケジューリングによって得られた情報を利用することによって冗長なタスクの同期を削除することができる[20]．以上のような有効性は次章で評価する．

また，本並列処理手法では，各プロセッサに対して異なるマシンコードを生成し，それを各 PE 上のプログラムメモリ上にダウンロードし，実行する．

4. OSCAR 上で実行した評価

本章では，提案するスパース行列求解のための並列化手法の性能を，OSCAR 上で評価した結果について述べる．

図 5 は，OSCAR 上でスパース線形方程式を解いた時の並列処理時間をプロセッサ台数に対してプロットしたものである．実線は OSCAR 上で計測した処理時間を，点線は生成されたマシンコードの実行に必要とする時間のシミュレーションによる推定値を示している．

図中の LM モードは

1) スタティックスケジューリングの結果プロセッサ内部だけで使用されることが分かった変数はローカルメモリに割り当てる
2) 同一プロセッサに割り当てられたタスク間のデータ授受はローカルメモリを介して行う
3) プロセッサ間のデータ転送は分散共有メモリを用いて 1PE 対 1PE 直接データ転送あるいは 1PE 対全 PE ブロードキャストを用いて行う

ということを意味する．

また，CM モードは

1) タスク間で共有される変数すべてを集中型共

有メモリに割り当てる

2) すなわち同一プロセッサに割り当てられたタスク間および異なるプロセッサに割り当てられたタスク間のデータの授受をすべて共有メモリを介した間接データ転送により行う

ということを意味する.

図5(a)は，サイズが 50×50 で4% の非零要素を持った行列の並列処理時間を示したものである．CM モードの OSCAR 上での処理時間は，PE が1台では 0.99 ms であったのが，3台で 0.38 ms，6台で 0.25 ms，8台で 0.27 ms と減少している．また，LM モードの OSCAR 上での処理時間は，PE 1台では 0.65

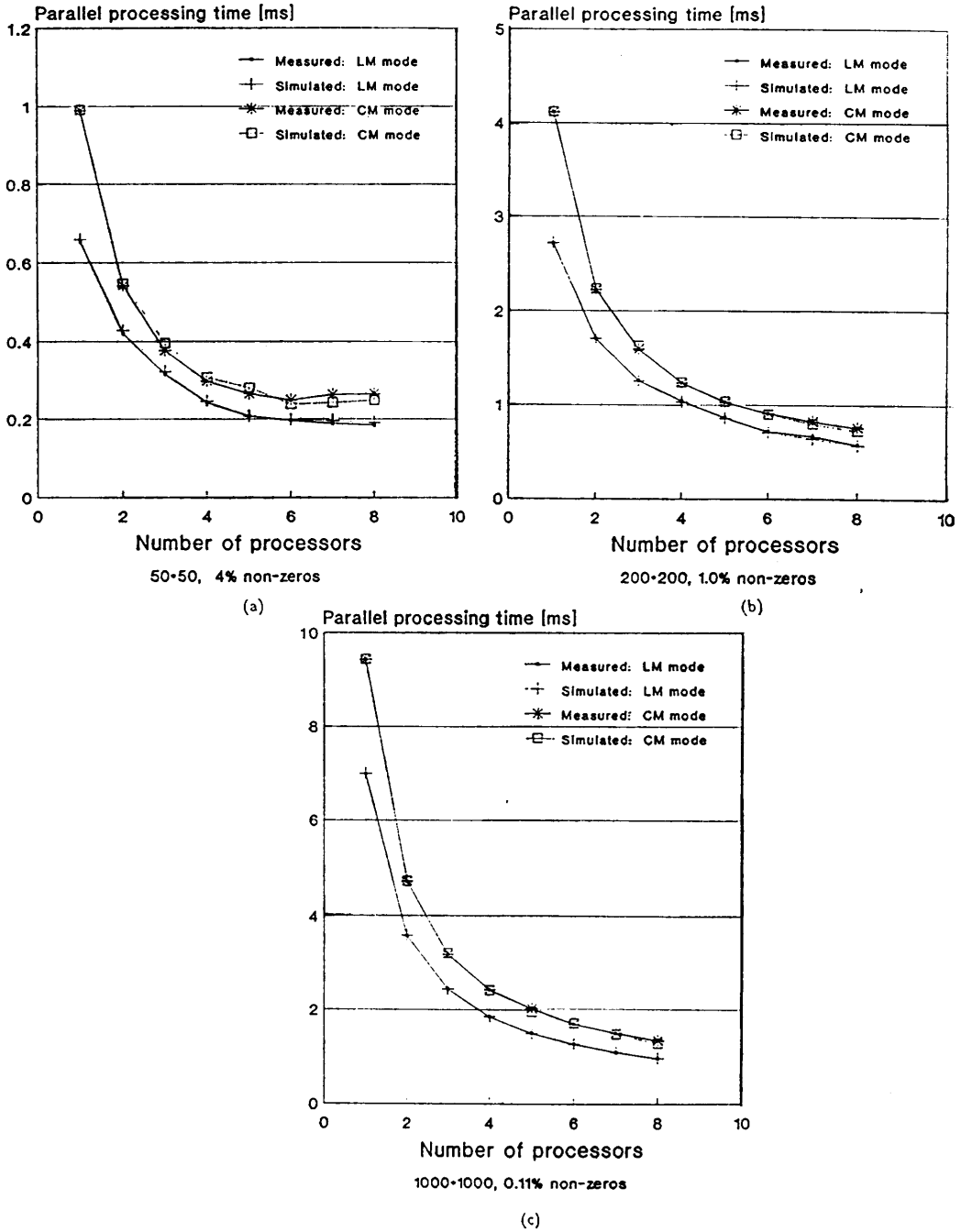


図 5 OSCAR 上での並列処理時間
Fig. 5 Parallel processing time on OSCAR.

ms であったのが，3台で 0.32ms，6台で 0.20ms，8台で 0.195 ms と減少している．ここで CM モードの処理時間を見てみると，より多くのプロセッサを使用することにより，逆に処理時間が増加してしまう場合があり，このことから，並列処理においては，プロセッサ数の増加が常に処理時間を減少させるとは限らず，適切なプロセッサ数の選択が重要であることが分かる．

図5(b)は，サイズが 200×200 で1% の非零要素を持った行列の並列処理時間を示したものである．CM モードの OSCAR 上での処理時間は，PE 1台で 4.12 ms であったのが，4台で 1.24 ms，8台で 0.76 ms と減少している．また，LM モードの OSCAR 上での処理時間は，PE 1台では 2.71 ms であるのに対して，4台で 1.04 ms，8台で 0.58 ms と減少している．

図5(c)は，サイズが 1,000×1,000 で 0.11% の非零要素を持った行列の並列処理時間を示したものである．CM モードの OSCAR 上での処理時間は，PE 1台で 9.44 ms であったのが，4台で 2.43 ms，8台で 1.34 ms と減少している．また，LM モードの OSCAR 上での処理時間は，PE 1台では 6.99 ms であるのに対し，4台で 1.85 ms，8台で 0.97 ms と減少している．

しかしながら，OSCAR はまだオペレーションテストの段階なので，図5に示した処理時間は，OSCAR の能力を落とした状態で計測されたものである．OSCAR は通常運転モードで 5 MHz で動作し2ワードのデータを5クロックで転送するが，ここでは 3 MHz で動作し2ワードのデータを9クロックで転送する転送モードで評価を行った．したがって，通常の状態で動作させれば，処理時間とデータ転送のオーバヘッドは現在より減るものと思われる．

以上の結果に共通して，図5より以下の3つのことが結論できる．

まず第1は，LM モードのほうが CM モードよりも OSCAR 上での処理時間が短いということである．このような処理時間の差が生じる原因を以下に述べる．

まず，プロセッサ台数1台の時の処理時間の差は，LM モードではすべてのタスク間データ授受をローカルメモリを用いて行うのに対して，CM モードではすべて共有メモリを介してタスク間データ授受を行うために処理時間の差が生じる．

またプロセッサ台数が複数の時の処理時間の差は主にプロセッサ間でのデータ転送時間の差により生じる．すなわち単一データをあるプロセッサから他の一台のプロセッサに送る場合には

1) LM モードでは，分散共有メモリを使用して，送信プロセッサが受信プロセッサの分散共有メモリ上に 1PE 対 1PE 直接データ転送によってデータを書き込む
2) CM モードでは，送信プロセッサが共有メモリにデータを書き込み，それを受信プロセッサが読み込む

という方法となり，1データの転送のために LM モードではライト1回を行うのに対し，CM モードでは共有メモリにライトとリードの二回のアクセスを行わなければならない．

また，あるプロセッサが単一データを他のすべてのプロセッサに送る場合には

1) LM モードでは送信プロセッサが，受信プロセッサの分散共有メモリ上にブロードキャスト転送によって同時にデータを書き込む
2) CM モードでは送信プロセッサが，共有メモリにデータを書き込み，それを他のすべてのプロセッサが次々に読み込む

という方法となり，1データの転送のために LM モードでは同じくライト一回を行うのに対し，CM モードでは共有メモリにプロセッサ台数回のアクセスを行わなければならない．

このような LM モードの使用はスタティックスケジューリング使用時のみに可能となるものであり，以上の結果は，スタティックスケジューリングに基づくローカルメモリと分散共有メモリの使用が，データ転送のオーバヘッドを減少させるのに有効であることを示している．

第2に，OSCAR 上での実並列処理時間とシミュレーションによる推定処理時間との間の差がほとんどないということである．これは，コンパイラが正確に実行のスケジューリングをしていることを意味する．いいかえると，コンパイラによる最適化は，OSCAR 上での処理オーバヘッドの軽減のために有効であることを示している．

第3に，テストしたすべてのケースにおいて，使用するプロセッサ数の増加と共に計算時間が顕著に減少しているということである．この結果が意味することは，提案するスパース行列の求解のためのコンパイレ

イション手法は有効かつ実用的であることを示している．

5. おわりに

本論文では，従来並列処理が困難であった線形スパース方程式直接解法の，マルチプロセッサシステム用並列化コンパイレイション手法を提案した．提案する手法ではコード生成法により生成されたループフリーコードを

1) ループフリーコードの細粒度タスクへ分割
2) タスクをプロセッサへ CP/DT/MISF を用いてスケジューリング
3) 最適化された並列マシンコードの生成

の手順で並列化する．

また，本論文では，提案する手法の有効性を示すため実プロセッサ上で性能評価を行った．評価の結果，提案する手法によりランダムスパース行列の求解時間をプロセッサ台数の増加と共に短縮できることが確かめられると共に，ローカルメモリと分散共有メモリの使用により，データ転送のオーバヘッドを共有メモリを使用した場合に比べて顕著に軽減できることも確かめられた．

参考文献

1) Duff, I. S., Erisman, A. M. and Reid, J. K.: *Direct Method for Sparse Matrices*, Oxford Univ. Press (1986).
2) Newton, A. R.: The Simulation of Large Scale Integrated Circuits, *IEEE Trans. Circuits & Syst.*, Vol. CAS-26, pp. 741-749 (Sep. 1979).
3) Cohen, E.: Program Reference for SPICE2, Electronics Res. Lab., Mem. No. ERL-M592, Univ. of California, Berkeley (June 1976).
4) Gustavson, F. G., Liniger, W. and Willoughby, R.: Symbolic Generation of an Optimal Crout Algorithm for Sparse Systems of Linear Equations, *J. ACM*, Vol. 17, No. 1, pp. 87-109 (1970).
5) Yamamoto, F. and Takahashi, S.: Vectorized LU Decomposition Algorithm for Large-scale Circuit Simulation, *IEEE Trans. Computer-Aided Design*, Vol. CAD-4, pp. 232-238 (July 1985).
6) Wing, O. and Huang, J. W.: A Computation Model of Parallel Solution of Linear Equations, *IEEE Trans. Comput.*, Vol. C-29, No. 7, pp. 632-638 (July 1980).
7) Huang, J. W. and Wing, O.: Optimal Parallel Triangulation of a Sparse Matrix, *IEEE Trans. Circuits & Syst.*, Vol. CAS-26, pp. 726-732 (1979).
8) Yeh, D. C. and Rao, V. B.: Partitioning Issues in Circuit Simulation on Multiprocessors, *Proc. IEEE Int. Symp. Circuits & Syst.*, pp. 300-303 (1988).
9) Sadayappan, P. and Visvanathan, V.: Circuit Simulation on Shared-Memory Multiprocessors, *IEEE Trans. Comput.*, Vol. 37, No. 12, pp. 1634-1642 (1988).
10) Fukui, Y., Yoshida, H. and Higono, S.: Supercomputing of Circuit Simulation, *Conf. Supercomputing '89*, pp. 81-85 (1989).
11) Markowitz, H. M.: The Elimination Form of Inverse and Its Application to Linear Programming, *Manage. Sci.*, Vol. 3, pp. 255-269 (Apr. 1957).
12) Smart, D. and White, J.: Reducing the Parallel Solution Time of Sparse Circuit Matrices Using Reordered Gaussian Elimination and Relaxation, *Proc. IEEE Int. Symp. Circuits & Syst.*, pp. 627-630 (1988).
13) Berry, R. D.: An Optimal Ordering of Electronic Circuit Equations for Sparse Matrix Solution, *IEEE Trans. Circuit Theory*, Vol. CT-18, pp. 40-50 (Jan. 1971).
14) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic Pub. (1988).
15) Padua, D. A., Kuck, D. J. and Lawrie, D. H.: High-speed Multiprocessor and Compilation Techniques, *IEEE Trans. Comput.*, Vol. C-29, No. 9, pp. 763-776 (Sep. 1980).
16) Padua, D. A. and Wolfe, M. J.: Advanced Compiler Optimizations for Supercomputers, *Comm. ACM*, Vol. 29, No. 12, pp. 1184-1201 (1986).
17) Coffman, E. G., Jr. (ed.): *Computer and Job-shop Scheduling Theory*, Wiley (1976).
18) Garey, M. R. and Johnson, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman (1979).
19) Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, Vol. C-33, pp. 1023-1029 (Nov. 1984).
20) Kasahara, H. and Narita, S.: An Approach to Supercomputing Using Multiprocessor Scheduling Algorithms, *Proc. IEEE Int. Conf. Supercomputing*, pp. 139-148 (Dec. 1985).
21) Polychronopoulos, C. D.: *Parallel Programming and Compilers*, Kluwer Academic Pub. (1988).
22) Sarkar, V.: *Partitioning and Scheduling Parallel Programs for Multiprocessors*, MIT Press (1989).

23) 中田ほか：並列回路シミュレーションマシン Cenju，情報処理，Vol. 31, No. 5, pp. 593-601 (1990).

(平成3年4月8日受付)
(平成4年1月17日採録)

笠原 博徳（正会員）

昭和32年生．昭和55年早稲田大学理工学部電気工学科卒業．昭和60年同大学院博士課程修了．工学博士．昭和58年～60年早稲田大学理工学部助手．昭和61年早稲田大学理工学部電気工学科専任講師．昭和63年早稲田大学理工学部電気工学科助教授，平成3年情報学科助教授，現在に至る．平成元年～2年イリノイ大学 Center for Supercomputing Research & Development 客員研究員．昭和62年 IFAC World Congress 第1回 Young Author Prize 受賞．著書「並列処理技術」(コロナ社)．電子情報通信学会，電気学会，シミュレーション学会，ロボット学会，IEEE, ACM 等の会員．

Wichian Premchaiswadi（正会員）

昭和31年生．昭和53年（タイ王国）Khon Kaen 大学工学部電気工学科卒業．昭和58年 Chulalongkorn 大学大学院 Computer Science 専攻修士課程修了．平成元年早稲田大学大学院理工学部電気工学科修士課程修了．平成4年同大学大学院理工学研究科博士課程修了．工学博士．同年4月より Khon Kaen 大学助教授，現在に至る．各種アプリケーション並列処理の研究に従事．

田村 光雄

昭和42年生．平成2年早稲田大学理工学部電気工学科卒業．平成4年同大学大学院理工学研究科電気工学専攻修士課程修了．同年4月松下電器産業株式会社入社．並列化コンパイラ，マクロデータフロー処理の研究に従事．

前川 仁孝（学生会員）

昭和42年生．平成2年早稲田大学理工学部電気工学科卒業．平成4年同大学大学院理工学研究科電気工学専攻修士課程修了．同年同大学院博士課程在学中．各種アプリケーション並列処理の研究に従事．

成田誠之助

昭和35年早稲田大学理工学部電気工学科卒業．昭和37年同大大学院修士修了．同年米国パデュー大学大学院留学（フルブライト留学生），昭和38年早稲田大学理工学部助手以後講師・助教授を経て昭和48年教授，現在に至る．41, 45年度電気学会論文賞受賞．分散計算機制御システム，並列処理，産業用ロボット制御，ディジタル制御理論，CIM 等に関する研究に従事．電気学会，計測自動制御学会，ロボット学会，IEEE 各会員．工学博士．