スタティックスケジューリングを用いた電子回路シミュレーションの粗粒度/近細粒度階層型並列処理手法

前川仁孝[†] 高井峰生[†] 伊藤泰樹[††]
西川健[†,☆] 笠原博徳[†]

本論文では，回路分割を用いた分割回路間粗粒度並列処理手法と分割回路内のステートメント間近細粒度並列処理手法を階層的に適用する，直接法を用いた電子回路シミュレーションの並列処理手法を提案する．従来より，マルチプロセッサ上での電子回路シミュレーションでは，タスク粒度を比較的粗くできる回路分割手法を用いた並列化がよく行われてきた．しかし，この回路分割法では効率良い並列処理が可能な回路の分割数は必ずしもプロセッサ数と同一でないという問題点がある．そこで本論文で提案する粗粒度/近細粒度階層型並列処理手法では，回路分割により生成された各分割回路をマクロタスクとしてプロセッサクラスタに割り当て，各プロセッサクラスタ内では割り当てられた分割回路内の解析計算をステートメントレベルの近細粒度タスクに分割し，データ転送を考慮したスタティックスケジューリングアルゴリズムを用いて並列処理することにより，従来の回路分割法のみでは効率良い並列処理が行えなかったような回路の処理を高速化することが可能となる．本手法をマルチプロセッサシステムOSCAR上で性能評価した結果，従来の回路分割による粗粒度並列処理のみの場合と比べ，プロセッサを16台用いた場合平均して約40%程度の速度向上が得られることが確かめられた．

A Coarse Grain/Near Fine Grain Hierarchical Parallel Processing Scheme of Circuit Simulation Using Static Scheduling

YOSHITAKA MAEKAWA,[†] MINEO TAKAI,[†] TAIKI ITO,[††]
TAKESHI NISIKAWA[†,☆] and HIRONORI KASAHARA[†]

This paper proposes a hierarchical parallel processing scheme of circuit simulation using direct method, which hierarchially combines coarse grain parallel processing with circuit tearing and near fine grain parallel processing. Parallel processing schemes of a circuit simulation using the circuit tearing have been often employed for distributed memory multiprocessor systems. However, in the circuit tearing, it is difficult to tear a circuit to a optimal number of processors by which parallel processing time can be minimum. Taking this fact into consideration, this paper proposes a hierarchical parallel processing scheme, which combines coarse grain parallel processing using circuit tearing and near fine grain parallel processing. Torn circuits scheduled to processor clusters. Next, analysis of a torn circuit is performed by processors inside a processor clustor by near fine grain parallel processing technique. In the near fine grain parallel processing, statement level tasks are generated from loop free code for the solution of unstructured sparse matrix using Crout method inside a torn circuit. Then the generated tasks are scheduled to processors inside the processor cluster by static scheduling algolithm. Also, performance of the proposed scheme is evaluated on a multiprocessor system OSCAR. It has been confirmed that a hierarchical scheme allows us to obtain about 40% speedup compared with ordinary circuit tearing parallel processing method.

1. はじめに

半導体技術の進歩とともに，VLSIの集積度は年々上昇している．これにともない，回路の設計と検証に多くの時間とコストが必要となっている．特に，電子回路シミュレーションの処理時間の短縮は，このようなVLSI開発を短時間で行うための重要な課題のひと

† 早稲田大学理工学部
School of Science and Engineering, Waseda University
†† 大蔵省造幣局
Mint Bureau Japan
☆ 現在，興銀システム開発株式会社
Presently with IBJ SYSTEMS, LIMITED

つである．そのため，従来よりベクトルスーパーコンピュータを用いた電子回路シミュレーションの高速化が行われているが，この電子回路シミュレーションはランダムスパース行列を係数に持つ連立一次方程式の求解をともなうため，科学技術計算の中で最も並列化しにくい計算のひとつである[1]．そこで，現在ではマルチプロセッサシステム上での並列処理による高速化の研究が多く行われている[2)~13)]．ここで問題となるのは，ランダムスパース行列を係数に持つ連立一次方程式の並列求解や非線形素子のパラメータ計算などをどのように各プロセッサ上に分割配置し，並列処理の効率を上げるかということである．従来より電子回路シミュレーションの並列処理においては，Waveform 法[9)] や Piecewise 法[10)] などの緩和法を用いた並列化手法の研究が多く用いられている．しかし，この緩和法を用いた手法は，並列性が高く容易に並列処理を行えるが，計算の収束性のために適用できる回路が限られるという問題がある．これに対し，直接法を用いた電子回路シミュレーション手法[2)~8),15),16)] は任意の回路に対し適用できるが，ランダムスパース行列を係数に持つ連立一次方程式をクラウト法などを用いて解く必要があるため，並列化が難しいという問題点がある．また，この直接法を用いた電子回路シミュレーションにおいても，回路分割[2)~5),11)~15)]を適用し，分割回路間の並列性を用いる粗粒度並列処理手法の研究[2)~5),11)~13)]も行われているが，分割数を増やすと分割回路間の計算負荷が増大し，効率良い並列処理が行えないことが知られている[11)]．

そこで，本論文では，任意台数のプロセッサを持つマルチプロセッサ上で効率良い並列処理を可能とするために，電子回路をつねにプロセッサ数に分割するのではなく，効率的な並列処理が可能な範囲で回路を分割し，分割した回路内の解析を，ステートメントレベルの近細粒度タスク[7)] に分割して並列処理する粗粒度/近細粒度階層型並列処理手法を提案する．生成されたマクロタスク，近細粒度タスクを効率良く並列処理するために，タスクをプロセッサに割り当てるスケジューリング手法が必要となるが，従来分割回路間の並列性を有効に利用するために提案されていたスケジューリング手法[5)] では，タスク間のデータ転送について考慮していないため，データ転送オーバヘッドが大きいという問題があった．これに対し，本論文で提案する手法は，分割回路間の粗粒度タスクと分割回路内のステートメントレベルの近細粒度タスクをデータ転送時間を考慮したスタティックスケジューリングアルゴリズムを用いてプロセッサにスケジューリングすることにより，データ転送オーバヘッドおよび同期オーバヘッドを軽減することを可能としている．

また，本論文では，提案手法に基づき作成された専用目的並列化コンパイラを用いてマルチプロセッサシステム OSCAR (Optimally Scheduled Advanced Multiprocessor)[17)] 上で性能評価を行った結果について述べる．

2. 直接法を用いた電子回路シミュレーションの粗粒度/近細粒度階層型並列処理手法

本章では，電子回路シミュレーション専用目的コンパイラ開発のための粗粒度/近細粒度階層型並列処理手法について述べる．

一般に，電子回路の過渡特性は，非線形連立微分方程式

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, t) = 0$$

とモデル化することができる．ここで，$\mathbf{x}$ は接点電圧などを表す N 次元の解ベクトルである．この方程式求解のために，本論文では回路の特性によらず適用可能な直接法を用いる．また，微分方程式を解くためのインプリシット積分法として BDF 法[18)]を用い，非線形方程式を解くためにニュートンラフソン法を用い，ニュートンラフソン法中の線形方程式を解くためにクラウト法を用いる．

直接法を用いた電子回路シミュレーションの並列処理を行う場合には，回路分割による粗粒度並列性[2)~5),11)~13)]と，分割した回路内の計算のステートメントレベル分割による近細粒度並列性[7)] が存在する．本論文では，粗粒度並列性と近細粒度並列性を階層的に用い，従来よりよく使用されている回路分割を用いた効率的な並列処理が困難な回路も効率良く並列処理できる新しい粗粒度/近細粒度階層型並列処理手法を提案する．以下の節では，回路分割手法，マクロタスク生成手法，近細粒度タスク生成手法，マクロタスクと近細粒度タスクを階層的にスケジューリングする手法，並列化マシンコードを生成する手法について述べる．

2.1 回路の自動分割およびマクロタスクの生成

本節では，回路の自動分割手法およびプロセッサクラスタへの割当て単位となるマクロタスクの生成法について述べる．マクロタスクとは，複数のプロセッサからなるプロセッサクラスタ（PC）に割り当てられる基本単位であり，このマクロタスクはさらに近細粒度タスクに分割され PC 内のプロセッサエレメント（PE）で階層的に並列処理される．

電子回路シミュレーションの計算は，BDF 法にお

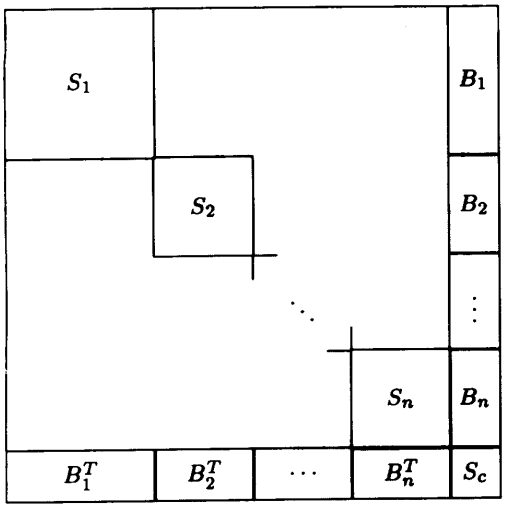

図1 縁付きブロック対角行列
Fig. 1 Bordered block diagonal matrix.

ける予測子の計算，微分値の計算，ニュートンラフソン法を用いた非線形連立方程式の求解，BDF 法における次のステップや次数の決定を行う．提案手法におけるマクロタスクの定義手法は，線形方程式求解の前に行う予測子の計算と微分値の計算を 1 つのマクロタスクと定義する．線形方程式の求解部については，回路分割法と合わせて後に述べる．線形方程式求解の後に行う収束判定は，回路分割により分割された各回路内の収束判定と分割回路間の収束判定に分け，それぞれの計算をマクロタスクと定義する．最後に，次のステップや次数の決定の計算を 1 つのマクロタスクと定義する．

次に，回路分割を適用した際の回路シミュレーション計算と，線形方程式求解部のマクロタスクの定義手法について述べる．電子回路を 2 分割する際の代表的な手法に繰返し改善法[19),20)] があり，特に Fiduccia と Mattheyses のアルゴリズム[20)] が回路解析などで使われている[5),14)]．本論文では，この Fiduccia と Mattheyses のアルゴリズムを用いて回路の 2 分割法を再帰的に適用することによって回路を多分割する手法を用いる．

回路分割を用いた際の行列計算では，与えられた回路に対して Fiduccia と Mattheyses のアルゴリズムを用いて回路の分割点を決定し，分割点ノードに対応する要素を行列の縁側に配置し，**図 1** のような縁付きブロック対角構造を持った行列にリオーダリングする．図 1 中の $S_k\,(k=1,\ldots,n)$ は，分割された各行列より生成される小行列で，$S_c, B_k, B_k^T\,(k=1,\ldots,n)$ は，分割点の接点電圧のノードの集合からなるブロックである．また，線形方程式の求解を行う部分では，図 1 のような行列構造を持つ方程式をクラウト法を用い，LU 分解，前進代入，後退代入を行う．

電子回路シミュレーションを行う際に，非線形素子が存在するときは，LU 分解を行う前にヤコビ行列の再計算を行う必要がある．ここでは以下で定義するマクロタスク $LU_k\,(k=1,\ldots,n), LU_c$ に含まれる要素の再計算をそれぞれマクロタスクと定義し $P_k\,(k=1,\ldots,n), P_c$ と表す．

LU 分解では，各部分行列 $S_k, B_k, B_k^T\,(k=1,\ldots,n)$ に関する LU 分解計算をそれぞれマクロタスクと定義し，$LU_k\,(k=1,\ldots,n)$ と表す．

前進代入では，$S_k\,(k=1,\ldots,n)$ に含まれる要素を使用して計算できる部分をそれぞれマクロタスクと定義し，$FS_k\,(k=1,\ldots,n)$ と表す．また，残った S_c に含まれる要素を使って行う計算をマクロタスクと定義し，FS_c と表す．

後退代入でも同様に，$S_k\,(k=1,\ldots,n)$ に含まれる要素を使って行う計算をそれぞれマクロタスクと定義し，$BS_k\,(k=1,\ldots,n)$ と表す．また，残った S_c に含まれる要素を使って行う計算をマクロタスクと定義し，BS_c と表す．

生成されたマクロタスク間には，フロー依存，出力依存，逆依存のようなデータ依存が存在する．並列処理においては，これらのデータ依存から生じる実行順序制約を満たしながら処理を行う必要がある．これらのデータ依存は，マクロタスクグラフと呼ぶ無サイクル有向グラフで表される．ここで定義された各マクロタスク間のデータ依存解析を行い，冗長なデータ依存エッジの除去，データ転送オーバヘッドを軽減するためのマクロタスク融合[21)]を行うと，線形方程式求解部の各マクロタスク間のデータ依存関係を表すマクロタスクグラフ[22),23)] は**図 2** のようになる．これらのマクロタスクは，各プロセッサクラスタへの割当て単位（粗粒度並列処理の単位）となる．

また，電子回路シミュレーションでは，図 2 中央部の $LU_c \cdot FS_c \cdot BS_c$ からなるマクロタスクのように，他のマクロタスクとの間に並列性がない支配節となるマクロタスクが生成される．このため本手法では，マクロタスクグラフをスケジューリングする際にこのようなマクロタスクが現れた場合には，まずこのマクロタスクの上下でマクロタスクグラフを分割し，上下それぞれの部分の部分グラフに対して提案する階層型スケジューリングを適用し，残った $LU_c \cdot FS_c \cdot BS_c$ のマクロタスクは，プロセッサクラスタの構成をすべてのプロセッサからなる 1 プロセッサクラスタ構成を変更して近細粒度レベル並列処理することにより，プロ

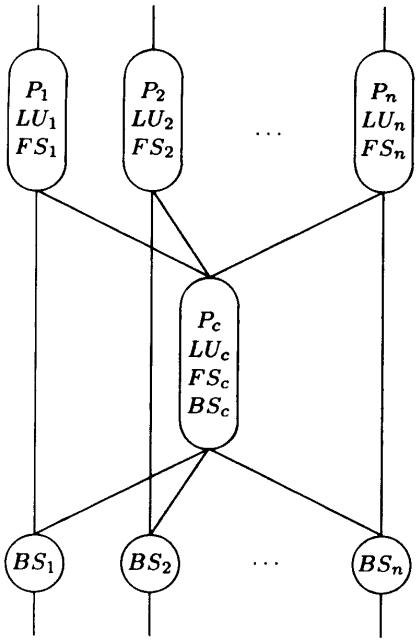

図2 回路分割時におけるクラウト法の粗粒度タスク間のマクロタスクグラフ
Fig. 2 Macrotask graph for crout method with circuit tearing.

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ & a_{22} & & a_{24} & \\ & & a_{33} & a_{34} & \\ & & & a_{44} & a_{45} \\ & a_{52} & & & a_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

LU分解部

$$\begin{aligned} u_{12} &= a_{12}/a_{11} \\ u_{24} &= a_{24}/a_{22} \\ u_{34} &= a_{34}/a_{33} \\ l_{54} &= -a_{52} \times a_{22} \\ u_{45} &= a_{45}/a_{44} \\ l_{55} &= a_{55} - l_{54} \times u_{45} \end{aligned}$$

前進代入部

$$\begin{aligned} y_1 &= b_1/a_{11} \\ y_2 &= b_2/a_{22} \\ b_5 &= b_5 - a_{52}/y_2 \\ y_3 &= b_3/a_{33} \\ y_4 &= b_4/a_{44} \\ b_5 &= b_5 - l_{54} \times y_4 \\ x_5 &= b_5/l_{55} \end{aligned}$$

後退代入部

$$\begin{aligned} x_4 &= y_4 - u_{45}/x_5 \\ x_3 &= y_3 - u_{34}/x_4 \\ x_2 &= y_2 - u_{24}/x_4 \\ x_1 &= y_1 - u_{12}/x_2 \end{aligned}$$

図3 ループフリーコード
Fig. 3 Loop free code.

セッサ利用効率を上げることができる．

2.2 分割回路内での近細粒度タスク生成

本節では，マクロタスク内の近細粒度タスクの生成手法について述べる．前述のように各マクロタスク内では，部分行列のLU分解等の計算が行われる．本手法では，このLU分解等の計算をコード生成法を用いて**図3**のような非零要素の計算を列挙したループフリーコードに変換し[24),25)]，ステートメントレベルの近細粒度並列処理を行う[7)]．このループフリーコードは通常のFortranプログラムと比べ逐次処理時間を50～100分の1に短縮できることが知られている[24)]．

この生成された近細粒度タスクは，マクロタスクグラフ生成のときと同様に，近細粒度タスク間のフロー依存，出力依存，逆依存のようなデータ依存を解析することにより，**図4**のような近細粒度タスクグラフと呼ぶ無サイクル有向グラフで表される．図4は図3中のループフリーコードのデータ依存関係を表したものである．図中，タスクはノードに対応しており，ノードの内側の数字はタスク番号を，ノード横の数字はPEにおけるタスクの推定処理時間を表している．また，タスクグラフ上のエッジにはデータ転送を表す可変の重みが付けがされており，エッジのヘッドとテールのタスクが，異なるPEに割り当てられるならばプロセッサ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ば0あるいはローカルメモリへのアクセス時間となる[21)]．

また，ループフリーコード生成時には，部分行列に対しクラウト法におけるフィルインを減少させるリオーダリング手法であるMarkowitz法[26)]を用い逐次計算の量を減らしている．

2.3 階層的タスクスケジューリング

マクロタスクおよび近細粒度タスクをプロセッサに割り当てるために階層的スケジューリングを行う．この階層的スケジューリングで求解が必要となる実行時間最小化を目的としたマクロタスクおよび近細粒度タスクのプロセッサへのスケジューリング問題[23)]は，強NP困難な問題であることが知られている[21),27)]．このため本並列化手法では，スケジューリングに要する時間とスケジュール結果の質を考慮にいれて，データ転送オーバヘッド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であるCP/DT/MISF法[28)]を用いる．CP/DT/MISF法におけるタスク割当ての手順は，あるスケジューリング時点において

(1) レディタスクの中から，出口ノードまでのパス長が最も長いタスクを選ぶ（CP），
(2) 同じレベルのタスクが複数あるときは，各レディタスクをアイドルプロセッサに割り当てた際に

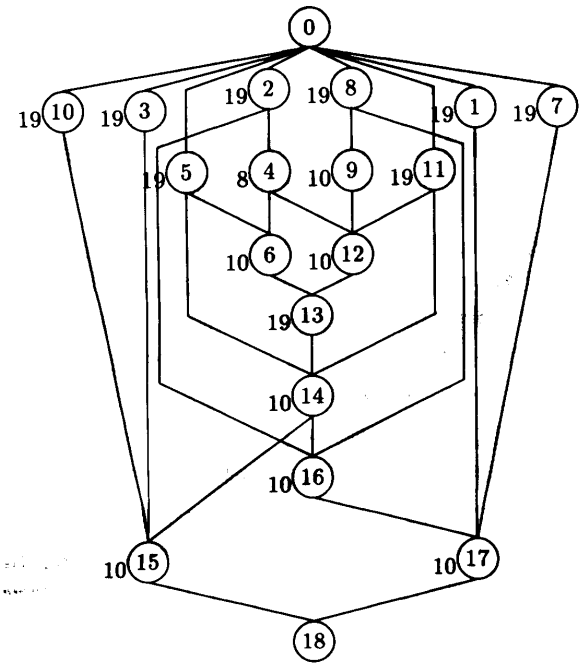

図4 近細粒度タスクグラフ
Fig. 4 Task graph.

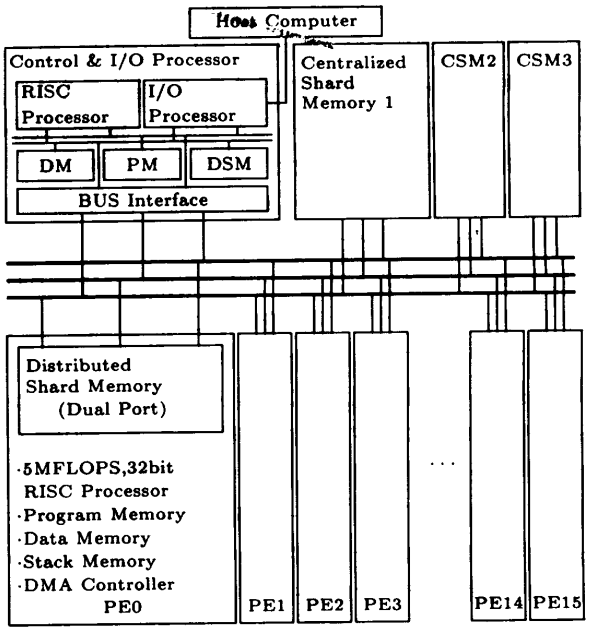

図5 OSCARのアーキテクチャ
Fig. 5 OSCAR's architecture.

データ転送時間が最小になるような割当てを選ぶ（DT），

(3) そのような割当てが複数あれば，直接後続タスク数が最も多いタスクを選ぶ（MISF），

のような優先順位でマクロタスクをプロセッサクラスタへ，近細粒度タスクをプロセッサへ割り当てる．またCP/DT/MISF法の時間複雑さは，タスク数を n プロセッサ台数を m とすると $O(n^3m)$ となり，タスク数が数千程度のタスクグラフのスケジューリングに要する時間は，通常のワークステーション上で，数秒から数十秒しか要しない．

階層的タスクスケジューリングを行う際には，この単階層のCP/DT/MISF法を階層的に適用することよりスケジューリングを行う．階層的タスクスケジューリング手法は，まず，近細粒度タスクグラフ上の近細粒度タスクをCP/DT/MISF法を用いてプロセッサクラスタ内プロセッサへスケジューリングする．次に，この近細粒度タスクをスケジューリングした際に得られるスケジュール長を各マクロタスクの推定実行時間として，マクロタスクグラフ上のマクロタスクをCP/DT/MISF法を用いてプロセッサクラスタへスケジューリングする．最後に，このマクロタスクのスケジューリング結果の中に前述の近細粒度タスクのスケジューリング結果を組み込むことにより，階層的スケジューリング結果を得る．ただし，$LU_c \cdot FS_c \cdot BS_c$ のような支配節となるマクロタスクをスケジューリングする際には，プロセッサクラスタの構成をすべてのプロセッサからなる1プロセッサクラスタ構成に変更し近細粒度レベルで並列処理を行うため，支配節中に含まれる近細粒度タスクを単階層のCP/DT/MISF法を用いてスケジューリングした結果を用いる．

以上のようなスケジューリング手法を用いることにより，任意数のプロセッサクラスタおよびプロセッサに対して粗粒度タスク近細粒度タスクをスケジューリングすることができる．

2.4 並列化マシンコード生成

本節では，並列化マシンコード生成について述べる．実際の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，最適化された並列化マシンコードを生成しなくてはならない．このマシンコード生成は，使用するマルチプロセッサシステムのアーキテクチャに依存するので，本手法のインプリメントを実際に行ったマルチプロセッサシステムOSCARを例に説明する．OSCARのアーキテクチャを図5に示す．OSCARは，集中型共有メモリ（CSM）と各プロセッサ上に分散されている分散共有メモリ（DSM）を持った共有メモリ型マルチプロセッサシステムであり，最大16台のプロセッサエレメント（PE），1台のコントロール&I/Oプロセッサ（CIOP），3モジュールのCSMが3本のバスに接続されている．また，OSCARの各PEには，5MFLOPSの能力を持つ32ビットのRISC型プロセッサ，ローカルデータメモリ，ローカルプログラムメモリ，DSM，スタックメモリ，データ

のプレロードやポストストアに使用するDMAコントローラが実装されている．このプロセッサは，加算，減算，乗算の各浮動小数点命令を含めた全命令を1クロック（200 ns）で実行することができ，他の浮動小数点演算も固定クロック数で終了することができる．また，各CSMモジュールへは，3本のバスを介して3台のPEから同時アクセスが可能となっている．さらに3本のバスは各々高速バリア同期機構を持っており，同時に3つのバリア同期をとることが可能である．

OSCARのプロセッサ間データ転送モードとして，DSMを用いた1PE対1PE直接データ転送，1PEから全PEへのブロードキャストデータ転送，CSMを介した間接データ転送モードという3種類のデータ転送モードを持っている．データ転送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を用いてこの3種類のデータ転送モードを使い分けることが重要である．さらに，同期フラグの授受はDSMを用いて行うことにより，同期のオーバヘッドを大幅に減少させることができる．これは，DSM上に同期フラグを割り当てた場合，各PEは自分のPE内のDSM上にある同期フラグをチェックすることができるので，従来の主記憶共有メモリマルチプロセッサのように，同期のためのビジーウェイトによる外部バスアクセスの必要がなくなる．これにより同期オーバヘッドを最小化できるとともに，ビジーウェイトにともなうプロセッサ間バスバンド幅の低下という問題を避けることができる．

また，本並列処理手法の特徴のひとつに，階層的スタティックスケジューリングの結果を用いて各PC，各PEに粗粒度および近細粒度タスクを割り当てるために，各PCおよび各PEに対して異なるマシンコードを生成するという点がある．各プロセッサのマシンコードは，各プロセッサに割り当てられたタスクコードと他のプロセッサとのデータ転送コード，同期コードからなり，プロセッサは各自のローカルプログラムメモリ上にロードされたマシンコードを，他のプロセッサとデータ転送や同期を行いながら実行する．

並列化マシンコードの最適化を行う際には，スケジューリング結果より，各プロセッサ上で実行すべきタスクの集合と実行順序，タスク間で同期をとる場所などの情報が利用できる．これらのスケジューリング情報を利用することによって冗長な同期コードを削除することができる[7]．

また，OSCARは複数のプロセッサごとにグループ化してプロセッサクラスタとすることにより，平等型マルチプロセッサシステムをマルチクラスタシステムとして使用することができ，プロセッサクラスタ数やプロセッサクラスタ内プロセッサ台数を自由に構成することができる．

3. 階層型並列処理手法を用いた電子回路シミュレーションの性能評価

本章では，提案する粗粒度/近細粒度階層型並列処理手法の性能評価について述べる．具体的には，加算器回路，乗算器回路の過渡解析を例に，解析に要する処理時間を，提案する粗粒度/近細粒度階層型並列処理を適用した場合と，従来の粗粒度並列処理のみ，あるいは近細粒度並列処理のみを行った場合と比較を行った．この評価における粗粒度並列処理では，支配節となるマクロタスクに対して，一般的な回路分割並列処理手法で使われるループ並列化手法を用いており，階層型並列処理ではこの部分に対して近細粒度並列処理を適用している．

評価に用いた回路は，BJTトランジスタを用いた回路で，リップルキャリー方式の2ビット，8ビットの整数加算器回路と4ビット整数乗算器回路である．それぞれの回路の総素子数，トランジスタ数，ノード数，行列サイズは，2ビット加算器回路の場合 $203, 18, 113, 190 \times 190$，8ビット加算器回路の場合 $822, 72, 462, 767 \times 767$，4ビット乗算器回路の場合 $1068, 208, 584, 972 \times 972$ となる．

表1は，2ビット整数加算器回路に対して，プロセッサ数を変化させた場合の回路の過渡解析に要した並列処理時間を，近細粒度並列処理のみ，粗粒度並列処理のみ，および階層型並列処理を行った場合について表している．評価の結果，PEを1台のみ用いたときの実行時間が237.4 msであるのに対して，近細粒度並列処理のみを行った場合には，PE2台で並列処理したと

表1 2bit加算器回路の並列処理時間
Table 1 Parallel processing time of 2bits adder circuit.

並列化手法	分割数	PC数	PE数	実行時間 [ms]
	1	1	1	237.4
近細粒度	1	1	2	164.1
	1	1	4	100.9
	1	1	8	63.1
	1	1	16	44.5
粗粒度	2		2	162.7
	4		4	121.8
	8		8	99.3
	16		16	92.7
階層	2	2	1	147.4
	2	2	2	96.7
	2	2	4	60.3
	2	2	8	43.3

きは 164.1 ms（逐次処理を行った場合の 1/1.45 倍），PE4 台のときは 100.9 ms（1/2.35 倍），PE16 台のときは 44.5 ms（1/5.33 倍）となる．また，粗粒度並列処理のみを行った場合には，回路を 2 分割し PE2 台で並列処理したときは 162.7 ms（1/1.46 倍），回路を 4 分割し PE4 台のときは 121.8 ms（1/1.95 倍），回路を 8 分割し PE8 台のときは 99.3 ms（1/2.39 倍），回路を 16 分割し PE16 台のときは 92.7 ms（1/2.56 倍）となり，プロセッサ数が 2 までは，粗粒度並列処理の処理時間の方が近細粒度並列処理のみより速いが，プロセッサ数が 4 以上では，近細粒度並列処理の方が高い性能を示していることが分かる．特に，プロセッサ数が 8 から 16 に増えたときの粗粒度並列処理時間にはほとんど変化がなく，プロセッサ数の増加とともに無闇に分割数を増やしても分割によるオーバヘッドが大きくなることが分かる．

次に，階層型並列処理との比較について述べる．階層型並列処理において，PC 内 PE 台数が 1 のときは，プロセッサクラスタ内では近細粒度並列処理を行わないが，支配節となるマクロタスクでは，前述のように全プロセッサを用いて近細粒度並列処理を行っている．評価の結果，回路を 2 分割し PC 数 2，PC 内 PE 台数 1（計 2PE）の構成としたときは 147.4 ms（1/1.61 倍），同じく 2 分割して PC 数 2，PC 内 PE 台数 8（計 16PE）の構成としたときは 43.3 ms（1/5.48 倍）となり，階層型並列処理の適用により，従来の粗粒度のみ，あるいは近細粒度のみの場合より処理を高速化できることが確認された．

また，支配節のマクロタスクに対してプロセッサクラスタの構成を一時的に変更して全プロセッサを用いた近細粒度レベルの並列処理を行う効果を調べるため，評価に用いた各回路を 16 分割した場合における図 2 中の $LU_c \cdot FS_c \cdot BS_c$ で表されるマクロタスクの 1 回の実行時間を，シーケンシャル処理をした場合と，16 プロセッサを用いて粗粒度並列処理時に用いられるループ並列化をした場合と，階層型並列処理に用いられる近細粒度並列処理をした場合を比較した．その結果，シーケンシャル処理を行った場合 7581 クロックを必要とするのに対して，ループ並列化を行った場合は 3277 クロック，近細粒度並列処理を行った場合は 2571 クロックとなり，支配節となるマクロタスクに対して，全プロセッサを用いた近細粒度並列処理が最も効率的な並列処理を行えることが確認された．

表 2 は，8 ビット整数加算器の回路に対して，表 1 と同様な評価を行った場合を表している．評価の結果，PE を 1 台のみ用いたときが 1295.4 ms であるのに対して，近細粒度並列処理のみを行った場合には，PE2 台のときは 849.6 ms（1/1.52 倍），PE16 台のときは 215.4 ms（1/6.01 倍）となり，また粗粒度並列処理のみを行った場合には，回路を 2 分割し PE2 台のときは 786.1 ms（1/1.65 倍），回路を 16 分割し PE16 台のときは 297.4 ms（1/4.36 倍）となる．また，階層型並列処理を行った場合には，回路を 8 分割し PC 数 8，PC 内 PE 台数 2（計 16PE）の構成としたときは 134.8 ms（1/9.61 倍）となり，回路を 8 分割して階層型並列処理を行った場合が最も高速化されたことが確認できた．また，回路を 8 分割し PC 数を 8，PC 内 PE 台数を 2 の構成としたときの階層型並列処理の処理時間が 8 台のプロセッサを用いた粗粒度並列処理 2 分の 1 以下になっているが，これは前述の支配節部分の並列処理手法の違いのためである．

表 2 8bit 加算器回路の並列処理時間
Table 2 Parallel processing time of 8bits adder circuit.

並列化手法	分割数	PC 数	PE 数	実行時間 [ms]
	1	1	1	1295.4
近細粒度	1	1	2	849.6
	1	1	4	514.7
	1	1	8	313.1
	1	1	16	215.4
粗粒度	2		2	786.1
	4		4	537.1
	8		8	318.4
	16		16	297.4
階層	2	2	1	757.4
	2	2	2	470.1
	2	2	4	295.1
	2	2	8	207.7
	4	4	1	421.7
	4	4	2	264.7
	4	4	4	199.6
	8	8	1	184.5
	8	8	2	134.8

表 3 は，4 ビット整数乗算器の回路に対して，同様な評価を行ったものである．評価の結果，PE を 1 台のみ用いたときが 1488.9 ms であるのに対して，近細粒度並列処理のみを行った場合には，PE2 台のときは 1063.1 ms（1/1.40 倍），PE16 台のときは 318.0 ms（1/4.68 倍）となる．また粗粒度並列処理のみを行った場合には，回路を 2 分割し PE2 台のときは 958.6 ms（1/1.55 倍），回路を 16 分割し PE16 台のときは 444.7 ms（1/3.35 倍）となる．次に，階層型並列処理を行った場合には，回路を 8 分割し PC 数 8，PC 内 PE 台数を 2（計 16PE）の構成としたときは 276.2 ms（1/5.39 倍）となり，この例でも階層型並列処理を行った場合が最も高速化されたことが確認できた．

これらの結果より，同じプロセッサ数を用いた場合，

表3 4bit 乗算器回路の並列処理時間
Table 3 Parallel processing time of 4bits multipler circuit.

並列化手法	分割数	PC 数	PE 数	実行時間 [ms]
	1	1	1	1488.9
近細粒度	1	1	2	1063.1
	1	1	4	669.5
	1	1	8	443.5
	1	1	16	318.0
粗粒度	2		2	958.6
	4		4	675.0
	8		8	569.7
	16		16	444.7
階層	2	2	1	875.3
	2	2	2	578.9
	2	2	4	386.0
	2	2	8	303.9
	4	4	1	526.8
	4	4	2	353.0
	4	4	4	281.6
	8	8	1	345.6
	8	8	2	276.2

どの例でも階層型並列処理を行った場合が最高速であることが分かる.

この要因は，粗粒度並列処理のみを行った場合は，回路を多く分割したことにより負荷バランスの悪化および回路分割における計算のオーバヘッドの増加が生じてしまうのに対し，階層型並列処理を用い分割回路をクラスタに割り当てて，クラスタの内部で近細粒度並列処理することにより，プロセッサ間負荷バランスが良くなり，データ転送転送オーバヘッドも小さくなっているためと思われる.

以上より提案する粗粒度/近細粒度階層型並列処理が，従来の回路分割を用いた粗粒度並列処理手法より任意のプロセッサ数に対して短い処理時間を与えることが確かめられた.

4. おわりに

本論文では，回路の自動分割手法によって分割された回路をプロセッサクラスタに割り当てることにより並列処理する粗粒度並列処理と，分割回路内のスパースマトリクス求解部をステートメントレベルで並列処理する近細粒度並列処理を階層的に組み合わせる直接法を用いた電子回路シミュレーションの粗粒度/近細粒度階層型並列処理手法を提案した．また，提案手法の性能をマルチプロセッサシステム OSCAR 上で評価した結果，提案手法により，プロセッサを 16 台使用した場合で，近細粒度並列処理のみの場合より平均して約 10%，粗粒度並列処理の場合に対して平均して約 40% 程度の処理の高速化が得られていることが確認できた.

今後の課題として，最良の並列効果を得られる回路の分割数を自動的に決定する手法の開発があげられる．また，提案した粗粒度/近細粒度階層型並列処理手法は，将来のワンチップマルチプロセッサをプロセッサエレメントとし，それを複数接続したマルチプロセッサ上で，プロセッサ間粗粒度並列処理，およびプロセッサ内細粒度並列処理のための手法として使用できると考えられる.

参考文献

1) Pointer, L.: Perfect Report: 1, *CSRD Rpt.* No.896 (1989).
2) 中田登志之，田辺記生，梶原信樹，松下 智，小野塚裕美，浅野由裕，小池誠彦：並列回路シミュレーションマシン Cenju，情報処理学会誌，Vol.31, No.5, pp.593–601 (1990).
3) 西垣正勝，田中伸幸，浅井秀樹：直接法に基づく回路解析における階層的な分割と潜在性，第4回回路とシステム軽井沢ワークショップ論文集，pp.139–144 (1991).
4) 下郡慎太郎，大村由紀子，寺前久美子，鹿毛哲郎：ブロック分割による直接法回路シミュレーションの並列化，電子情報通信学会技術研究報告，VLD90–28, pp.1–6 (1990).
5) Cox, P.F., Burch, R.G., Hocevar, D.E., Yang, P. and Epler, B.D.: Direct Circuit Simulation Algorithms for Parallel Processing, *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, Vol.10, No.6, pp.714–725 (1991).
6) Ho, C.W., Ruehli, A.E. and Brennan, P.A.: The Modified Nodal Approach to Network Analysis, *IEEE Trans. Circuits and Systems*, Vol.CAS-22, No.6, pp.504–509 (1975).
7) 前川仁孝，吉成泰彦，中山 功，田村光雄，笠原博徳：直接法を用いた電子回路シミュレーションの近細粒度並列処理，電気学会論文誌 C 分冊，Vol.114-C, No.5, pp.579–587 (1994).
8) Sadayappan, P. and Visvanatan, V.: Circuit Simulation on Shared Memory Multiprocessors, *IEEE Trans. Comp.*, Vol.37, No.12, pp.1634–1642 (1988).
9) Lelarsmee, E., Ruehli, A.E. and Vincentelli, A.L.: The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits, *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, Vol.CAD-1, No.3, pp.131–145 (1982).
10) Ruehli, A.E.: *Circuit Analysis, Simulation and Design 2*, North Holland (1987).
11) Yuan, C.P., Lucas, R., Chan, P. and Dutton,

R.: Parallel Electronic Circuit Simulation on the iPSC System, *Proc. IEEE 1988 Custom Integrated Circuit Conf.*, pp.6.5.1–6.5.4 (1988).
12) Onozuka, H., Kanoh, M., Mizuta, C., Nakata, T. and Tanabe, N.: Development of Parallelism for Circuit Simulation by Tearing, *Proc. European Conf. on Design Automation with the European inASIC Design*, pp.12–17 (1993).
13) 田辺記生，斎藤敏幸，蜂屋孝太郎：回路シミュレーションにおける並列処理技術の動向と展望，1993 第 7 回回路とシステム軽井沢ワークショップ，pp.19–24 (1993).
14) 黒岩 実，山内 宗，中田登志之，小池誠彦：並列計算機 Cenju2 上での並列グラフ分割アルゴリズムの実装と評価，1992 第 6 回回路とシステム軽井沢ワークショップ，pp.543–548 (1992).
15) Hajj, I.N.: Sparcity Considerations in Network Solution by Tearing, *IEEE Trans. Circuits and Systems*, Vol.CAS–27, No.5, pp.357–366 (1980).
16) Antognetti, P. and Massobrio, G.: *Semiconductor Device Modeling with SPICE*, McGrow-Hill Book Company (1988).
17) 笠原博徳，本多弘樹，橋本 親：OSCAR のアーキテクチャ，電子通信学会論文誌 D，Vol.J71–D, No.8, pp.1440–1445 (1988).
18) Brayton, B.K., Gustavson F.G. and Hachtel G.D.: A New Efficient Algorithm for Solving Differential Algebraic Systems Using Implicit Backward Differential Formulas, *Proc. IEEE*, Vol.60, No.1, pp.98–108 (1978).
19) Kernighan, B.W. and Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs, *Bell System Technical J.*, Vol.49, pp.291–307 (1970).
20) Fiduccia, C.M. and Mattheyses, R.M.: A Linear-time Heuristic for Improving Network Partitions, *Proc. 19th Design Automat. Conf.*, pp.175–181 (1982).
21) 笠原博徳：並列処理技術，コロナ社 (1991).
22) Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, Vol.C–33, No.11, pp.1023–1029 (1984).
23) Coffman, E.G. Jr. (Ed.): *Computer and Jobshop Scheduling Theory*, Wiley (1976).
24) Fukui, Y., Yoshida, H. and Higono, S.: Supercomputing of Circuit Simulation, *Proc. Supercomputing'89*, pp.81–85 (1989).
25) 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).
26) Markowitz, H.M.: The Elimination Form of Inverse and Its Application to Linear Programming, *Management Science*, Vol.3, pp.255–269 (1957).
27) Garey, M.R. and Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman (1979).
28) Kasahara, H., Honda, H. and Narita, S.: Parallel Processing of Near Fine Grain Tasks Using Scheduling on OSCAR (Optimally Scheduled Advanced Multiprocessor), *Proc. Supercomputing'90*, pp.856–864 (1990).

(平成 7 年 8 月 21 日受付)
(平成 8 年 7 月 4 日採録)

前川 仁孝（正会員）

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．平成 5 年日本学術振興会特別研究員 DC．平成 6 年早稲田大学理工学部助手．平成 8 年同大学メディアネットワークセンター特別研究員．各種アプリケーションの並列処理に関する研究に従事．

高井 峰生（学生会員）

昭和 45 年生．平成 4 年早稲田大学理工学部電気工学科卒業．平成 6 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．現在同大学大学院博士課程在学中．平成 7 年同大学情報科学研究教育センター助手．平成 8 年同大学メディアネットワークセンター助手．各種アプリケーションの並列処理に関する研究に従事．電子情報通信学会，日本オペレーションズ・リサーチ学会，SCS 各会員．

伊藤 泰樹

昭和 44 年生．平成 5 年早稲田大学理工学部電気工学科卒業．平成 7 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，大蔵省造幣局に入局．

西川　健（正会員）

昭和46年生．平成6年早稲田大学理工学部電気工学科卒業．平成8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，興銀システム開発株式会社に入社．各種バンキングシステムの開発に従事．

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年同大学大学院博士課程修了．工学博士．昭和58年同大学理工学部助手．昭和60年カリフォルニア大バークレー短期客員研究員，日本学術振興会第1回特別研究員．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年同大学助教授．情報学科助教授を経て，現在電気電子情報工学科助教授．平成1～2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 Auther Prize 受賞．主な著書「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．