シングルチップマルチプロセッサ上での近細粒度並列処理

木村啓二[†] 尾形航[†]
岡本雅巳[††] 笠原博徳[†]

1チップ上に集積可能なトランジスタ数の増大に従い，次世代マイクロプロセッサでは，これらのトランジスタをいかに有効に利用し，プロセッサの実効性能を向上させるかが大きな課題になっている．しかし，現在主流のスーパースカラあるいはVLIW，それらの複合形のマイクロプロセッサでは，命令レベル並列性等の限界によりスケーラブルな実効性能の向上が困難と考えられている．これに対して，筆者らは従来のチップ内細粒度並列処理に加え，より並列性の大きいループイタレーションレベルの中粒度並列処理（ループ並列処理），サブルーチン，ループ，基本ブロック間の粗粒度並列性を階層的に組み合わせて使用するマルチグレイン並列処理を実現できるシングルチップマルチプロセッサ（SCM）はスケーラブルな実効性能の向上を可能にすると考えている．本論文では，マルチグレイン並列処理を効果的に実現できるSCM検討の第一歩として，共有キャッシュ，グローバルレジスタ，分散共有メモリ，ローカルメモリの近細粒度並列処理に対する有効性に関する評価を行った結果について述べる．

Near Fine Grain Parallel Processing on Single Chip Multiprocessors

KEIJI KIMURA,[†] WATARU OGATA,[†] MASAMI OKAMOTO[††] and HIRONORI KASAHARA[†]

With the increase of the number of transistors integrated on a chip, how to use transistors efficiently and improve effective performance of a processor is getting an important problem. However, it has been thought that superscalar and VLIW which have been popular architectures would have difficulty to obtain scalable improvement of effective performance because of limitation of instruction level parallelism. To cope with this problem, the authors have been proposing a single chip multiprocessor (SCM) approach to use multi grain parallelism inside a chip, which hierarchically exploits loop parallelism with large parallelism and coarse grain parallelism among subroutines, loops and basic blocks in addition to instruction level parallelism. This paper evaluates effectiveness of single chip multiprocessor architecture with a shared cache, global registers, distributed shared memory and/or local memory for near fine grain parallel processing as the first step of research on SCM architecture for supporting multi grain parallel processing.

1. はじめに

1チップ上で使用可能なトランジスタ数の増大に従い，現在様々な種類の次世代マイクロプロセッサアーキテクチャが提案されている．それらのアーキテクチャの代表的な例として，スーパースカラ，VLIWあるいはそれを組み合わせたアーキテクチャ上で投機的実行を用い，命令レベル並列処理を行おうとするもの[1),2)]，ロジックDRAM混載技術により大量のDRAMを用い，ベクトル演算を行おうとするもの[3)]，DRAMと複数のCPUを混載したシングルチップマルチプロセッサ[4),5)]やそのキャッシュ構成法[6),7)]，投機的実行も考慮し複数のスレッドの並列実行によりデータ転送オーバヘッドを隠すマルチスレッドアーキテクチャ[8)~11)]があげられる．特に文献7)ではチップ内のプロセッサ間でデータ転送を効果的に行うためのキャッシュ構成法に関する研究が行われており，L2キャッシュ共有はL1キャッシュ共有とほぼ同じ効果があるという結果を得ている．

一方，筆者らは細粒度並列処理[12)~14)]に加え，ループイタレーションレベルの中粒度並列処理[15)]，および

† 早稲田大学理工学部電気電子情報工学科
Department of Electrical, Electronics and Computer Engineering, Waseda University
†† 株式会社東芝
Toshiba Corporation

サブルーチンあるいはループ，基本ブロック間の粗粒度並列性[16),17)]を階層的に組み合わせて使用することにより，高い実効性能を達成することを目指すマルチグレイン並列処理[18)]を従来から提案している．このマルチグレイン並列処理をシングルチップマルチプロセッサ（SCM）に適用することにより，実効性能の高い高性能な計算機システムを実現することができると考えている[19)]．

そこで，本論文では，このマルチグレイン並列処理を効率良く実現できるSCMアーキテクチャ研究の第1ステップとして，近細粒度並列処理をSCMに適用し，グローバルレジスタ，共有キャッシュ，ローカルメモリ，分散共有メモリの効果を評価した結果について述べる．

以下，2章でマルチグレイン並列処理について，3章でマルチグレイン並列処理をサポートするアーキテクチャ，および本論文で評価するSCMについて，4章でこれらのアーキテクチャに近細粒度並列処理を適用して評価した結果について述べる．

2. マルチグレイン並列処理

本章では，筆者らが提案するシングルチップマルチプロセッサアーキテクチャが前提とする，マルチグレイン並列処理手法について述べる．

マルチグレイン並列処理手法[18)]とは，ループやサブルーチン等の粗粒度タスク間の並列処理を利用するマクロデータフロー処理[16),17)]，ループレベルの並列処理である中粒度並列処理，基本ブロック内部のステートメントレベルの並列性を利用する近細粒度並列処理[12),13)]を階層的に組み合わせて，並列処理を行う手法である．このマルチグレイン並列処理手法は，OSCARマルチグレインFortran並列化コンパイラに実装されている[19)]．

2.1 マクロデータフロー処理

マクロデータフロー処理では，ソースとなるFortranプログラムを疑似代入文ブロック（BPA），繰返しブロック（RB），サブルーチンブロック（SB）の3種類の粗粒度タスク〔マクロタスク（MT）〕[16)]に分割する．

ここで，BPAは基本的には通常の基本ブロックであるが，並列性抽出のために単一の基本ブロックを複数に分割したり，逆に1つのBPAの処理時間が短く，ダイナミックスケジューリング時のオーバヘッドが無視できない場合には，複数のBPAを融合して1つのBPAを生成する．

RBは，最外側ナチュラルループである．ただし，Doallループは，ループインデクス範囲を分割することにより複数の部分Doallループに分割し，分割後の部分Doallループを新たにRBと定義する．

また，サブルーチンは，可能な限りインライン展開するが，コード長を考慮し効果的にインライン展開ができないサブルーチンはSBとして定義する．

さらに，SBやDoall不可能なRBの場合，これらの内部の並列性に対し，階層的マクロデータフロー処理を適用する[21)]．

MT生成後，コンパイラはBPA，RB，SB等のMT間のコントロールフローとデータ依存を解析し，それらを表したマクロフローグラフ（MFG）[20),22)]を生成する．さらにMFGからMT間の並列性を最早実行可能条件解析[20),22)]により引きだし，その結果をマクロタスクグラフ（MTG）[20),22)]として出力する．その後MTは，条件分岐等の実行時不確定性が存在する場合にはダイナミックスケジューリングで，それ以外の場合にはスタティックスケジューリングにより各プロセッサクラスタ（PC）に割り当てられ実行される．

2.2 中粒度並列処理（ループ並列処理）

PCに割り当てられたMTがDoall可能なRBである場合，このRBはPC内のプロセッシングエレメント（PE）に対して，イタレーションレベルで分割され並列実行される．

2.3 近細粒度並列処理[12),13)]

PCに割り当てられたMTが，BPAやシーケンシャルループで構成される場合，それらはステートメントレベルのタスクに分割され，PC内のPEにより並列処理される．

図1はクラウト法によるスパース行列の求解を，シンボリックジェネレーション法を用いてループフリーコードに展開して行うプログラムである．OSCARコンパイラでは，このような基本ブロック内のステートメントをタスクとして定義し，タスク間のデータ依存を解析する．その後，**図2**のような各タスク間のデータ依存，すなわち先行制約を表したタスクグラフと呼ばれる無サイクル有向グラフを生成する．図中，各タスクは各ノードに対応している．図2において，ノード内の数字はタスク番号 i を表し，ノードの脇の数字はPE上でのタスク処理時間 t_i を表す．また，ノード N_i から N_j に向けて引かれたエッジは，タスク T_i が T_j に先行するという半順序制約を表している．タスク間のデータ転送時間も考慮する場合，各々のエッジは一般に可変な重みを持つ．タスク T_i と T_j が異なるPEへ割り当てられた場合，この重み t_{ij} がデータ転送時間となる．図2では，データ転送および同期

<< LU Decomposition >>

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

<< 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 1 近細粒度タスクの例
Fig. 1 Near fine grain tasks.

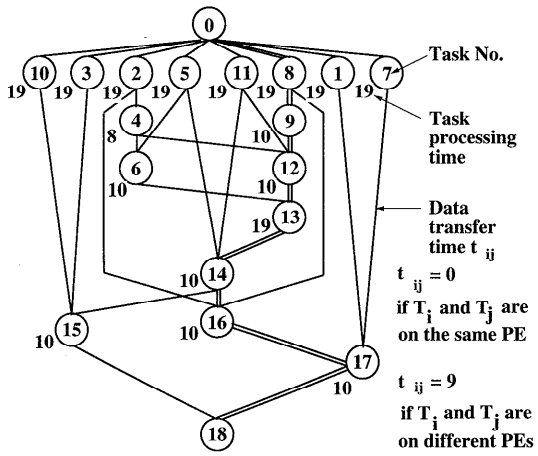


図 2 近細粒度タスクのタスクグラフ
Fig. 2 A task graph for near fine grain tasks.

に要する時間を 9 clock と仮定している．逆にこれらのタスクが同一 PE に割り当てられた場合，重み t_{ij} は 0 となる．

このようにして生成されたタスクグラフを各プロセッサにスタティックにスケジュールする．この際，OSCAR コンパイラでは，スケジューリングアルゴリズムとして，データ転送オーバヘッドを考慮し実行時間を最小化するヒューリスティックアルゴリズムである CP/DT/MISF 法，CP/ETF/MISF 法，ETF/CP 法，および DT/CP 法[22] の 4 手法を適用し最良のスケジュールを自動的に選んでいる．また，タスクをスタティックにプロセッサに割り当てることにより，BPA 内で用いられるデータをタスクの実行前にローカルメモリもしくは分散共有メモリに配置することができる[23]．

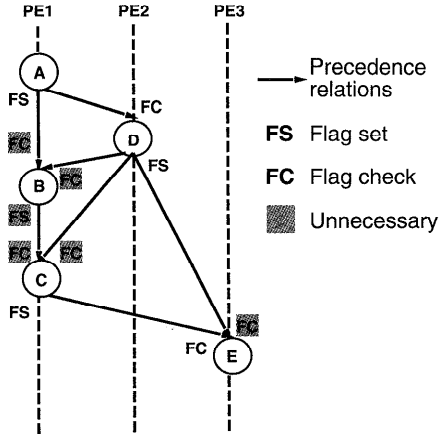


図 3 冗長な同期の削除
Fig. 3 Elimination of redundant synchronizations.

スケジューリング後，コンパイラは PE に割り当てられたタスクの命令列を順番に並べ，データ転送命令や同期命令を必要な箇所に挿入することにより，各 PE のマシンコードを生成する．近細粒度タスク間の同期にはバージョンナンバー法を用い，同期フラグの受信は受信側 PE のビジーウェイトによって行われる．

マシンコード生成時，コンパイラはスタティックスケジューリングの情報を用いたコード最適化を行うことができる．たとえば，同一データを使用する異なるタスクが同一 PE に割り当てられたとき，そのデータをレジスタを介して受渡しすることができる．また，同期のオーバヘッドを最小化するため，タスクの割当て状況や実行順序から，冗長な同期を除去することもできる[12]．この様子を**図 3** を用いて説明する．図中，タスク A，B，C は PE1 に，タスク D は PE2 に，タスク E は PE3 に割り当てられ，タスク間のエッジはデータ依存を表すものとする．つまり，PE 間のエッジはデータ転送および同期を表している．同期フラグを集中共有メモリ上に配置した場合，タスク B，C の実行前にタスク D の実行は終了しているので，タスク E はタスク D 終了の確認を行う必要がない．つまり，タスク D とタスク E の間の同期を除去することができる．

3. マルチグレイン並列処理をサポートするアーキテクチャ

本章では，2 章で述べたマルチグレイン並列処理を

SCM 上で効率良く実現するために必要なアーキテクチャサポートについて述べる．

3.1 マクロデータフロー処理をサポートするアーキテクチャ

2.1 節で述べたように，マクロデータフロー処理では，プログラム内に実行時不確定性が存在する場合，各 MT を実行時に PC に割り当てる．そのため，スケジューリング情報や MT 間で共有されるデータを格納できる集中共有メモリがあることが望ましい．さらに，スケジューリング情報のような小規模データの PE 間通信を行うための低レイテンシな PE 間ネットワークおよび同期機構の付加によって，効率的なダイナミックスケジューリングを行うことができる．

また，PE ローカルメモリの付加によって，MT 間データ転送オーバヘッドの最小化を可能とするデータローカライゼーション手法[24] を用いることができる．この際，CPU でのタスク処理とは独立にデータ転送を行うことができるデータ転送ユニット（DTU），およびデュアルポートメモリで構成された分散共有メモリ（DSM）を用いることによって，残存するデータ転送のタスク処理とのオーバラップ[23] が可能となる．

3.2 中粒度並列処理をサポートするアーキテクチャ

中粒度並列処理においては，ループ内でのテンポラリ配列を格納する PE ローカルメモリの使用が有効であると考えられる．また，リダクションループおよび Doacross における同期およびデータ転送のオーバヘッドを最小化するため，リモート PE の命令実行を妨げることなく直接データ転送を行える，デュアルポートメモリ構成の DSM，もしくは共有キャッシュ[7]，共有グローバルレジスタの利用が考えられる．

3.3 近細粒度並列処理をサポートするアーキテクチャ

2.3 節で述べたように，近細粒度並列処理は基本的に基本ブロック内に適用されるため，コンパイル時のスタティックスケジューリングにより実行時のスケジューリングオーバヘッドがなく，データ転送および同期オーバヘッドを最小化できる．このため，近細粒度並列処理の性能を最大限に引き出すためには，プログラム実行がコンパイル時のスケジューリングどおりのタイミングで行われることが好ましく，そのためには，コンパイラがタイミングを正確に決められるように，すべての命令が固定クロックで実行できることが望ましい．

また，プロセッサ間データ転送オーバヘッドを最小化するために，低レイテンシのデータ転送，および低オーバヘッドの同期機構が重要となる．このため，3.2 節で述べたようなデュアルポートメモリ構成の DSM，もしくは共有キャッシュ，共有グローバルレジスタの利用が近細粒度並列処理でも有効であると考えられる．

さらに，スタティックスケジューリングの結果により，プロセッサローカルに使用することが決定できる変数を割り当てることができ，また DSM より大容量を実現できるローカルメモリも有効と考えられる．

3.4 評価対象アーキテクチャ

本節では，上記のアーキテクチャサポートを考慮して，今回評価を行ったシングルチップマルチプロセッサアーキテクチャについて述べる．

評価対象アーキテクチャとして，まずプロセッシングエレメント（PE）間のメモリアーキテクチャの違いにより，データキャッシュ共有型と OSCAR 型（分散共有メモリ + ローカルメモリ）の 2 例を用意した．さらに，これらに対して PE 間グローバルレジスタを付加したものをそれぞれ用意した．

これらのアーキテクチャを，クロックレベルの精密なシミュレータを用いて評価を行う．

3.5 共通仕様

本論文で評価する SCM アーキテクチャは，32 bit 固定命令長，ロード/ストアアーキテクチャのシンプルなシングルイシュー RISC アーキテクチャの CPU を，1 チップ上に 1 基から 8 基まで搭載するものとした．この CPU は整数演算および浮動小数点演算の両方に使用することができる汎用レジスタを 64 本持ち，また，FMUL，FADD を含む全命令の実行を 1 クロックで処理することができるものとする．このようなシンプルな CPU を用いる理由は 3.3 節でも述べたように，近細粒度並列化におけるスタティックスケジューリングの結果をプログラム実行時に正確に再現することを可能とするためである．

プロセッサの内部には各々の CPU で実行するプログラムが格納されるローカルプログラムメモリ（LPM）があり，LPM には 1 クロックでアクセスできるものとする．LPM を想定しているのは，今回評価で用いるプログラムのサイズがあまり大きくないため，命令キャッシュを用いる場合との性能差は少ないと考えているためである．

また，プロセッサの外部には共有データを格納する集中共有メモリ（CSM）が接続される．この CSM のアクセスレイテンシは 20 クロックとする．

3.6 データキャッシュ共有型アーキテクチャ

今回評価に用いるデータキャッシュ共有型アーキテクチャは，図 4 に示すように，CPU とローカルプログラムメモリ（LPM）を持つプロセッシングエレメ

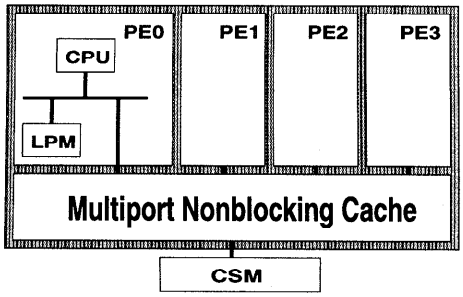


図4 データキャッシュ共有型アーキテクチャ
Fig. 4 Shared data cache architecture.

ント（PE）が，データキャッシュを共有するアーキテクチャである．

データキャッシュは複数のバンクを持つものを想定する．各々のバンクは各ポートとスイッチを介して接続される構成とし，同一バンクへのアクセスが生じた場合はいずれかのアクセスのみが優先されるが，それ以外の場合は各ポートが独立にキャッシュにアクセスできる．キャッシュの連想方式は 4-way set associative とし，Write Back もしくは Write Through 方式のいずれかを用いるものとする．

このようにデータキャッシュを共有することにより，キャッシュのコヒーレンスを気にすることなく各 PE 間のデータ転送，とりわけ近細粒度タスク間のデータ転送を効率良く行うことができる．

この共有データキャッシュは，ヒット時のアクセスタイムは 1 もしくは将来の GHz クロックを考慮して 3 クロック，容量は PE 数が 4 以下のときには 4 Mbyte，PE 数が 6 以上のときには 8 Mbyte とした．また，異なる 2 つのプロセッサからの要求が同一バンクでコンフリクトした場合は，一方が 1 クロック待たされるものとする．

このとき，キャッシュヒット時のアクセスタイムが 1 クロックの場合はデータのロード，ストア，および同期フラグのセットに各 1 クロック，同期フラグのチェックに最小で 3 クロック，また，ヒット時のアクセスタイムが 3 クロックの場合には，データのロード，ストア，および同期フラグのセットに各 3 クロック，同期フラグのチェックに最小 5 クロック要する．

3.7 OSCAR 型アーキテクチャ

OSCAR 型アーキテクチャとは，マルチプロセッサシステム OSCAR[25] を基に構成されたアーキテクチャである．OSCAR 型アーキテクチャの全体図を図 5 に示す．

OSCAR 型アーキテクチャは，CPU，LPM，データ転送ユニット（DTU），ローカルデータメモリ（LDM），

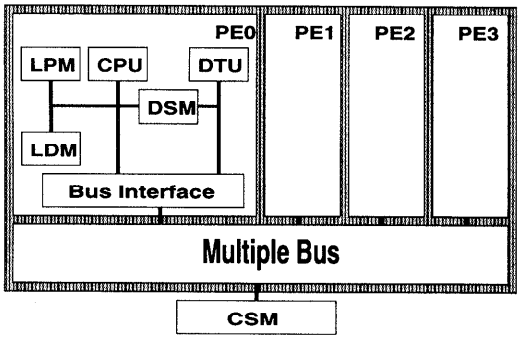


図5 OSCAR 型アーキテクチャ
Fig. 5 OSCAR type architecture.

そしてデュアルポートメモリで構成された分散共有メモリ（DSM）を持つプロセッシングエレメント（PE）を 3 本のバスを介して接続したアーキテクチャである．

LDM は自 PE からのみアクセスできるメモリであり，PE に割り当てられたタスク間で使用されるローカルデータを保持するために使用する．また，DSM は他 PE からも直接リード/ライトできるメモリであり，近細粒度タスク間のデータ転送に使用する．LDM のアクセスにかかるクロック数は 1 クロックと 3 クロックの 2 種類を用意，容量は 1 PE あたり 1 Mbyte とし，DSM のアクセスにかかるクロック数は，自 PE 上の DSM にはそれぞれ 1 あるいは 3 クロック，他 PE 上の DSM へのリモートアクセスにはそれぞれ 4 あるいは 6 クロックかかり，容量は 1 PE あたり 16 Kbyte とした．また，異なる 2 つのプロセッサからバスへのアクセスコンフリクトが生じた場合には，一方が 1 クロック待たされるものとする．同様に，同一 PE 上の DSM へのアクセスコンフリクトが生じた場合には一方が 2 クロック待たされるものとする．

このとき，自 PE の DSM へのアクセスタイムが 1 クロックの場合はデータのストアおよび同期フラグのセットに各 4 クロック，データのロードに 1 クロック，同期フラグのチェックに最小で 3 クロック，また，自 PE の DSM へのアクセスタイムが 3 クロックの場合はデータのストアおよび同期フラグのセットに 6 クロック，データのロードに 3 クロック，同期フラグのチェックに最小 5 クロック要する．

3.8 グローバルレジスタ

本論文では，3.6 節および 3.7 節で述べたデータキャッシュ共有型アーキテクチャ，OSCAR 型アーキテクチャの各々に，グローバルレジスタ（GR）を付加したアーキテクチャついても評価を行う．

GR はマルチポートレジスタで，各 PE 内の CPU が同時にアクセスすることができるものとする．また，

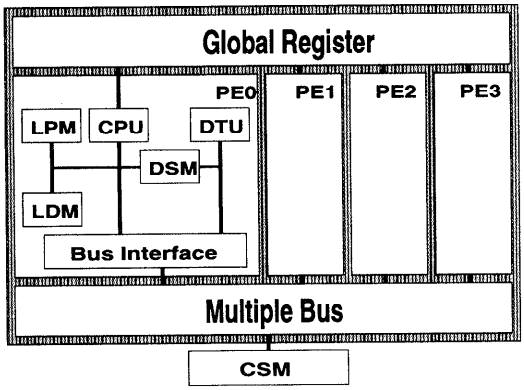


図 6 OSCAR 型アーキテクチャ+グローバルレジスタ
Fig. 6 OSCAR type architecture with global register.

このときのアクセスタイムは 1 クロックとする.

GR の本数は 16 本とし，これらは近細粒度タスクのデータ転送，および同期に使用する．この GR を使用することにより，データのロード，ストア，および同期フラグのセットを 1 クロック，同期フラグのチェックを最小 3 クロックで行うことができる．また，GR がスピルしたときには，OSCAR 型アーキテクチャでは DSM，データキャッシュ共有型アーキテクチャではキャッシュを使用した同期およびデータ転送を行う．

OSCAR 型アーキテクチャに GR を付加したときの全体図を図 **6** に示す.

3.9 トランジスタ数の比較

OSCAR 型アーキテクチャとデータキャッシュ共有型アーキテクチャのハードウェア量の比較を行うため，LDM，DSM およびキャッシュメモリが要求するトランジスタ数を概算[26] した結果，**表 1** のように，4 PE 時においては，OSCAR 型アーキテクチャにおいて LDM に 203.6 M トランジスタ，DSM に 6.4 M トランジスタ，合計 210.0 M トランジスタ程度，データキャッシュ共有型アーキテクチャでは 238.2 M トランジスタ程度と推定された.

本表のように，今回の評価では OSCAR 型アーキテクチャがデータキャッシュ共有型アーキテクチャよりも少ないトランジスタを用いたメモリシステムを仮定している.

4. 評　　価

本章では，データキャッシュ共有型アーキテクチャと OSCAR 型アーキテクチャに対して近細粒度並列処理を適用して評価した結果について述べる.

評価に用いたプログラムは，電子回路シミュレーションにおけるランダムスパースマトリクスを係数に持つ線形方程式の求解プログラム，および航空宇宙技術研究所の CFD プログラム「NS3D」のサブルーチン「SUB4」の内側ループで最も実行時間の大きいループの 1 イタレーション分のプログラムである.

表 1 OSCAR 型アーキテクチャとデータキャッシュ共有型アーキテクチャの 4 PE 時で各々のアーキテクチャのメモリが要するトランジスタ数の概算
Table 1 The number of transistors for memory of OSCAR type architecture and shared cache type architecture.

メモリモジュール	推定トランジスタ数 (M)
OSCAR/LDM	203.6
OSCAR/DSM	6.4
OSCAR/LDM+DSM	210.0
CACHE	238.2

4.1 ランダムスパースマトリクスの求解

このプログラムは算術代入文のみからなる Fortran ループフリーコードであり，94 個の近細粒度タスク（ステートメント）を持つ．ただし，本論文ではキャッシュにデータがロードされた状態での比較を行うため，基本ブロックをループで囲んで評価を行った.

このプログラムに近細粒度並列処理を施し，OSCAR 型アーキテクチャ（OSCAR），データキャッシュ共有型の Write Through（CACHE-WT）と Write Back（CACHE-WB），および，これらに対して共有グローバルレジスタを付加したアーキテクチャ（GR）のそれぞれで，PE 数 1，2，4，6，8 で実行した．ただし，データキャッシュ共有型では，PE 数が 4 までは，キャッシュのバンク数を 4，容量を 4 Mbyte，それより多い PE 数のときはキャッシュのバンク数を 8，容量を 8 Mbyte として評価を行った.

この結果として，OSCAR アーキテクチャにおける 1 プロセッサでの実行時間を基準としたときの速度向上率を図 **7**（ローカルメモリおよびキャッシュアクセス 1 クロック），および図 **8**（ローカルメモリおよびキャッシュアクセス 3 クロック）に示す．なお，ローカルメモリアクセスが 1 クロック時の 1 プロセッサでの実行時間は，OSCAR アーキテクチャで 126,145 クロック，CACHE-WB アーキテクチャで 128,985 クロック，また 3 クロックのときは OSCAR アーキテクチャで 200,235 クロック，CACHE-WB アーキテクチャで 203,781 クロックである.

図 7 より，まず CACHE-WT アーキテクチャの性能が著しく低いことが分かる．この CACHE-WT アーキテクチャのキャッシュと CSM との間に Write Buffer を接続しても，たかだか数%の性能向上しか得られなかった.

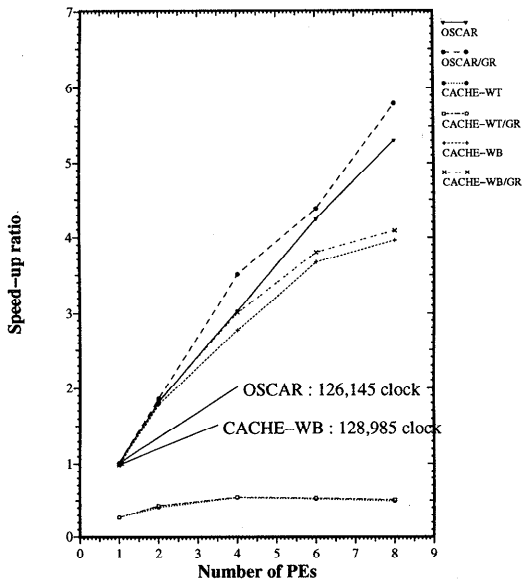


図 7 ローカルメモリアクセス 1 クロック時のランダムスパースマトリクスの求解における速度向上率
Fig. 7 Speed up ratio of random sparse matrix solution with 1clock local memory access.

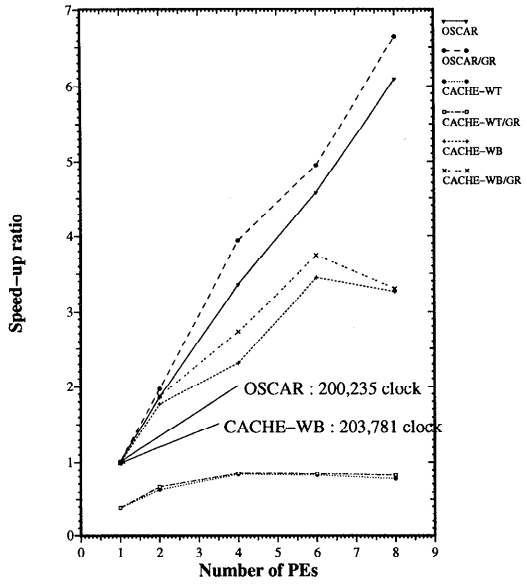


図 8 ローカルメモリアクセス 3 クロック時のランダムスパースマトリクスの求解における速度向上率
Fig. 8 Speed up ratio of random sparse matrix solution with 3clock local memory access.

また，この例の場合，4PE で OSCAR が 3.02 倍，CACHE-WB で 2.76 倍のスピードアップであり，8PE で OSCAR が 5.29 倍，CACHE-WB で 3.95 倍のスピードアップとなった．特に 8PE の場合では OSCAR アーキテクチャの方が CACHE-WB の 1.34 倍のスピードアップを達成しており，OSCAR の LDM，DSM，CSM の各メモリを効率的に用いることにより顕著な速度向上を得られることが確かめられた．

ここで，OSCAR および CACHE-WB アーキテクチャ（ローカルメモリおよびキャッシュアクセス 1 クロック）において，4PE 実行時の全 PE の実行クロック数の合計に対するメモリアクセスの割合を**表 2** に示す．表 2 より，転送データおよび同期フラグのセット（すなわちリモート DSM ストア）に 4 クロック要する OSCAR では，同期/データ転送領域に対するアクセスに 16.1%と大きいクロック数を費やしているが，同様の処理（すなわち共有キャッシュへのストア）を 1 クロックで行える CACHE-WB では同領域に対するアクセスが 6.5%で行えていることが分かる．一方，コンパイル時に変数をローカルメモリに割り当てることができる OSCAR では変数/定数/一時作業用領域に対するアクセスに全クロック数の 17.2%しか要しないのに対し，ラインの置換やバンクコンフリクトが生じる CACHE-WB では，今回の評価では多くのデータをキャッシュに残したホットスタート時での計測であるにもかかわらず，同領域に対するアクセスに 30.1%費やされている．これより図 7 における 4PE 時の性能差は，OSCAR ではコンパイラによる LDM，DSM の有効利用により，平均メモリアクセス時間を短縮できているためと考えられる．

表 2 4PE 時におけるランダムスパースマトリクス求解の実行クロックに対するメモリアクセスの割合
Table 2 The ratio of memory access clocks to executed instruction clocks for random sparse matrix solution with 4 PEs.

アーキテクチャ	各領域に対するメモリアクセスの割合（%）	
	同期/データ転送	変数/定数/一時作業用
OSCAR	16.1	17.2
OSCAR/GR	6.5	19.0
CACHE-WB	6.7	30.1
CACHE-WB/GR	2.3	31.2

次にグローバルレジスタの有無について述べる．OSCAR アーキテクチャでは，データ転送および同期フラグのセットを行うために送信先の PE のリモート DSM に 4，もしくは 6 クロックかけてデータを書き込まなければならないところを，グローバルレジスタを使用することによって 1 クロックでデータをセットすることができる．このグローバルレジスタの付加によって，OSCAR アーキテクチャでは 4PE 時に 3.02 倍の速度

向上率であったものが3.50倍となり，15.9%性能を向上させることができる．同様に，CACHE-WBアーキテクチャでも4PE時の速度向上率が2.76倍であったものがグローバルレジスタの付加により2.99倍となり，8.3%性能が向上している．これより，OSCAR，CACHE-WB双方においてグローバルレジスタの付加が性能向上に顕著に貢献することが分かる．

図8のローカルメモリおよびキャッシュメモリのアクセスに3クロックとした場合も，図7とほぼ同様にOSCARアーキテクチャが良い性能を示している．この場合，近細粒度タスク間のデータ転送や同期のオーバヘッドが増大しているため，グローバルレジスタの付加による性能向上が図7のときよりも顕著で，OSCARアーキテクチャの4PE時に17.3%，CACHE-WBの4PE時に18.1%の性能向上を得ている．

4.2 NS3D/SUB4

このプログラムは，航空宇宙技術研究所のCFDプログラム「NS3D」のサブルーチン「SUB4」の最外側ループに内包される4つのDo-loopのうち，最も実行時間の大きい2番目のDo-loopのループボディを取り出したもので，429個の近細粒度タスク（ステートメント）を持つ．SUB4の最外側ループはDoall可能なループであり，今回は外側ループの並列性はシングルチップマルチプロセッサ（SCM）を複数接続しているマルチプロセッサシステムのSCM間で使用し，SCMに割り当てられたイタレーションをさらにSCM内のPE間で近細粒度並列処理を行うことを想定して評価を行っている．

このプログラムに，4.1節と同様に近細粒度並列処理を施し，OSCAR型アーキテクチャ（OSCAR），データキャッシュ共有型のライトスルー（CACHE-WT）とライトバック（CACHE-WB），および，これらに対してグローバルレジスタを付加したアーキテクチャ（GR）のそれぞれで，PE数1，2，4，6，8で実行した．

この結果として，OSCAR型アーキテクチャにおける1プロセッサでの実行時間を基準としたときの速度向上率を**図9**（ローカルメモリおよびキャッシュアクセス1クロック），および**図10**（ローカルメモリおよびキャッシュアクセス3クロック）に示す．なお，ローカルメモリアクセスが1クロックのときの1プロセッサでの実行時間は，OSCARで881,204クロック，CACHE-WBで884,908クロック，また3クロックのときはOSCARで1,405,364クロック，CACHE-WBで1,409,966クロックである．

図9より，4.1節と同様にCACHE-WTアーキテク

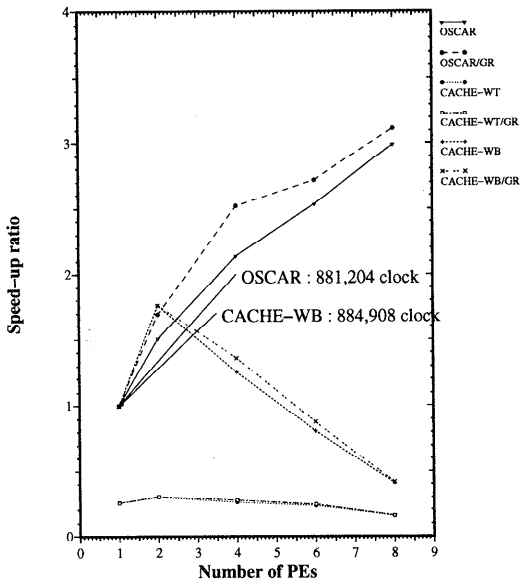


図9 ローカルメモリアクセス1クロック時のNS3D/SUB4における速度向上率

Fig. 9 Speed up ratio of NS3D/SUB4 with 1clock local memory access.

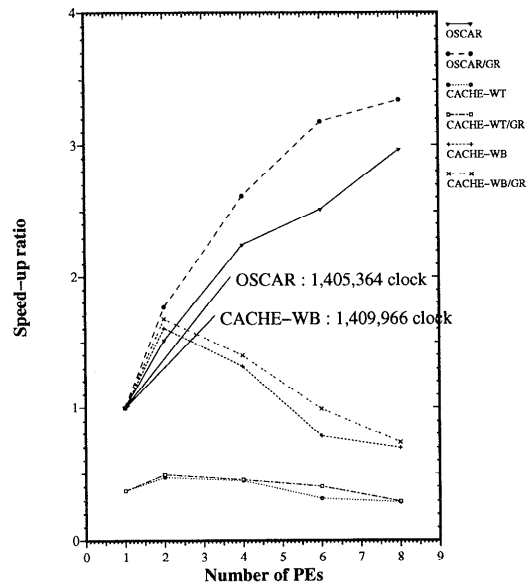


図10 ローカルメモリアクセス3クロック時のNS3D/SUB4における速度向上率

Fig. 10 Speed up ratio of NS3D/SUB4 with 3clock local memory access.

表3 4PE時におけるNS3Dの実行クロックに対するメモリアクセスの割合
Table 3 The ratio of memory access clocks to executed instruction clocks for NS3D with 4 PEs.

アーキテクチャ	各領域に対するメモリアクセスの割合（%）	
	同期/データ転送	変数/定数/一時作業用
OSCAR	27.7	15.2
OSCAR/GR	20.7	18.4
CACHE-WB	17.8	38.6
CACHE-WB/GR	16.0	38.3

チャの性能が著しく低いことが分かる．さらに，OSCARアーキテクチャがPE数4のときには2.14倍，8で2.99倍の速度向上率を得ているのに対しCACHE-WBアーキテクチャはPE数4で1.25倍であるが，その後はPE数の増加とともに性能が落ち，PE数8のときには1台のときの0.42倍となっている．

表3に，OSCARおよびCACHE-WBアーキテクチャ（ローカルメモリおよびキャッシュアクセス1クロック）各4PEで実行したときの，全PEの実行クロック数の合計に対するメモリアクセスの割合を示す．表3より，同期/データ転送領域に対するアクセスがリモートDSMストアに4クロックを要するOSCARで27.7%，共有キャッシュストアを1クロックで行えるCACHE-WBで17.8%を要していることが分かり，キャッシュアクセスのバンクコンフリクトがかなり大きいことが分かる．また，本アプリケーションのPE間データ転送および同期に費やされる割合が表2のランダムスパースマトリクスの求解と比較して大きいことが分かる．このとき，変数/定数/一時作業用領域に対するアクセスはOSCARで15.2%，CACHE-WBでは38.6%，メモリアクセスに要するクロック数の平均は，OSCARで1.4クロック，CACHEで2.6クロックであり，OSCARではLDM，DSMの有効利用により平均メモリアクセス時間を小さく抑えることで速度向上を可能としていることが分かる．

CACHE-WBアーキテクチャでPE数の増加とともに性能が向上しない原因は，本アプリケーションの場合，PE間データ転送がスパースマトリクスの求解のときよりも多く，共有キャッシュへの同期フラグ確認のためのビジーウェイトにより，データ転送および同期時のバンクコンフリクトが非常に大きくなるためと考えられる．

グローバルレジスタの付加に関しては，特にOSCARアーキテクチャでの性能向上が著しく，PE数が4のときにグローバルレジスタを付加したものは，付加しなかったものと比較して17.5%の性能向上を得ている．本アプリケーションのように，PE間のデータ転送と同期の回数が非常に多いとき，特にグローバルレジスタの使用が有効であることが分かる．

図10の3クロックアクセスの場合も，図9と同様な傾向を示しているが，近細粒度タスク間のデータ転送と同期のオーバヘッドが増大した分，グローバルレジスタの使用が有効であった．特に，OSCARアーキテクチャでPE数6のときには，グローバルレジスタを使用したときは使用しないときと比較して，26.5%の性能向上を得ている．

5. まとめ

本論文では，マルチグレイン並列処理に適したシングルチップマルチプロセッサアーキテクチャの検討を行うための第一歩としてデータキャッシュ共有型アーキテクチャ，OSCAR型アーキテクチャ，およびこれらのアーキテクチャにグローバルレジスタを付加したアーキテクチャ上で近細粒度並列処理を適用し，性能評価を行った．

その結果OSCAR型アーキテクチャでは，コンパイル時のスケジューリングにより決定できるプロセッサローカル変数のローカルメモリへの配置，プロセッサ間データ転送や同期のためのDSMあるいはグローバルレジスタの有効利用等のコンパイラ最適化を活用でき，これらの最適化によりランダムスパースマトリクス求解では，グローバルレジスタなし，ローカルメモリアクセス1クロック時に共有キャッシュアーキテクチャに対して，4PEで9.6%，8PEでは34.1%速度向上が可能であり，OSCAR型アーキテクチャの有効性が確かめられた．さらにCFD計算においても，OSCAR型アーキテクチャは共有キャッシュアーキテクチャに対しては4PEで70.9%，8PEで733.9%と著しい速度向上を得ることができた．また，グローバルレジスタの付加により，近細粒度タスクのデータ転送および同期を効果的に行うことができ，OSCARアーキテクチャでは最大26.5%，共有キャッシュアーキテクチャでは最大26.6%の速度向上を得ることができ，グローバルレジスタの付加が近細粒度並列処理で有効であることが分かった．

今後は，この近細粒度並列処理と中粒度並列処理および粗粒度並列処理を階層的に組み合わせて利用するマルチグレイン並列処理をシングルチップマルチプロセッサに対して適用し，マルチグレイン並列処理を有効に適用することができるシングルチップマルチプロセッサアーキテクチャについて検討を重ねていく予定

である．また，今後の課題として，SPARC，PowerPC等の商用スーパースカラRISCプロセッサをCPUコアにしたシングルチップマルチプロセッサ，L2キャッシュやスヌープキャッシュ[7]等本論文で評価したアーキテクチャとは異なるメモリ構成を持ったシングルチップマルチプロセッサアーキテクチャ等との比較，および今後のVLSI実装において問題となるハードウェア量，バスドライブ能力に関する検討等があげられる．

謝辞 本研究の一部は，通産相次世代情報処理基盤技術開発事業並列処理分散分野マルチプロセッサコンピューティング領域研究の一環として行われた．

最後に，CFDプログラムNS3Dをご提供いただいた航空宇宙技術研究所の皆様に感謝いたします．

参考文献

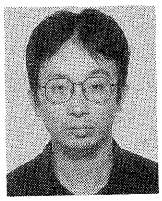
1) Patt, Y., et al.: One Billion Transistors, One Uniprocessor, One Chip, *Computer*, Vol.30, No.9, pp.51–57 (1997).
2) Lipasti, M. and Sben, J.: Superspeculative Microarchitecture for Beyond AD 2000, *Computer*, Vol.30, No.9, pp.59–66 (1997).
3) Kozyrakis, C., et al.: Scalabel Processors in the Billion-Transisteor Era: IRAM, *Computer*, Vol.30, No.9, pp.75–78 (1997).
4) Hammond, L., Nayfeb, B. and Olukotun, K.: A Single-Chip Multiprocessor, *Computer*, Vol.30, No.9, pp.79–85 (1997).
5) 岩下，宮嶋，村上：リファレンス PPRAM「$PPRAM^R$」に基づく「$PPRAM^R_{mf}$」アーキテクチャの概要，情報処理学会研究報告，Vol.96, No.80, pp.161–166 (1996).
6) 井上，若林，木村，天野：シングルチップマルチプロセッサ用半共有型スヌープキャッシュ，信学技報 CPSY, Vol.98, No.233, pp.75–81 (1998).
7) Nayfeh, B., Hammond, L. and Olukotun, K.: Evaluation of Design Alternatives for a Multiprocessor Microprocessor, *Proc. 23rd Anual International Symposium on Computer Architecture* (1996).
8) Shoi, G., Breach, S. and Vijaykumar, T.: Multiscalar Processors, *Proc. 22nd Anual International Symposium on Computer Architecture* (1995).
9) Tsai, J.-Y. and Yew, P.-C.: The Superthreaded Architecture: Thread Pipelining with Run-time Data Dependence Cehecking and Control Speculation, *Proc. Int'l Conf. on PACT'96* (1996).
10) 鳥居，近藤ほか：オンチップ制御並列プロセッサ MUSCAT の提案，情報処理学会論文誌，Vol.39, No.6, pp.1622–1631 (1998).
11) 玉造，松本，平木：実行時再構成方式テストベット Ocha-Pro の性能評価，情報処理学会研究報告 ARC, Vol.98, No.70, pp.127–132 (1998).
12) Kasahara, H., Honda, H. and Narita, S.: Parallel processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR (Optimally Scheduled Advanced Multiprocessor), *Proc. Supercomputing '90* (1990).
13) 笠原：マルチプロセッサシステム上での近細粒度並列処理，情報処理，Vol.37, No.7, pp.651–661 (1996).
14) 尾形，吉田，合田，岡本，笠原：スタティックスケジューリングを用いたマルチプロセッサシステム上での無同期近細粒度並列処理，情報処理学会論文誌，Vol.35, No.4, pp.522–531 (1994).
15) Padua, D. and Wolfe, M.: Advanced Compiler Optimization for Super Computers, *Comm. ACM*, Vol.29, No.12, pp.1184–1201 (1996).
16) 笠原，合田，吉田，岡本，本多：Fortran マクロデータフロー処理のマクロタスク生成手法，信学論（D-I），Vol.J75-D-I, No.8, pp.511–525 (1992).
17) 本田，合田，岡本，笠原：Fortran プログラム粗粒度タスクの OSCAR における並列実行方式，信学論（D-I），Vol.J75-D-I, No.8, pp.526–535 (1992).
18) Kasahara, H., Honda, H. and Narita, S.: A Multigrain Parallelizing Compilation Scheme for OSCAR, *Proc. 4th Workshop on Lang. And Compilers for Parallel Computing* (1991).
19) 笠原，尾形ほか：マルチグレイン並列化コンパイラとそのアーキテクチャ支援，信学技報，IDC98-10, CPSY98-10, FTS98-10, pp.71–76 (1998).
20) 本田，岩田，笠原：Fortran プログラム粗粒度タスク間の並列性検出法，信学論（D-I），Vol.J73-D-I, No.12, pp.951–960 (1990).
21) 岡本，合田，宮沢，本田，笠原：OSCAR マルチグレインコンパイラにおける階層型マクロデータフロー処理，情報処理学会論文誌，Vol.35, No.4, pp.513–521 (1994).
22) 笠原：並列処理技術，コロナ社 (1991).
23) 藤原，白鳥，鈴木，笠原：データプレロードおよびポストストアを考慮したマルチプロセッサスケジューリングアルゴリズム，電子情報通信学会論文誌（D-1），Vol.75, No.8, pp.495–503 (1992).
24) Kasahara, H. and Yoshida, A.: A Data-Localization Compilation Scheme Using Partial Static Task Assignment for Fortran Coarse Grain Parallel Processing, *Journal of Parallel Computing Special Issue on Languages and Compilers for Parallel Computers* (1998).
25) 笠原，成田，橋本：OSCAR（Optimally SCheduled Advanced multiprocessoR）のアーキテクチャ，信学論（D），Vol.J71-D, No.8 (1988).
26) Geuskens, B. and Rose, K.: *MODELING MI-*

CROPROCESSOR PERFORMANCE, Kluwer Academic Pub. (1998).

(平成 10 年 9 月 7 日受付)
(平成 11 年 3 月 5 日採録)

木村　啓二

昭和 47 年生．平成 8 年早稲田大学理工学部電気工学科卒業．平成 10 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．同年同大学院理工学研究科電気工学専攻博士後期課程に入学，現在に至る．マルチグレイン並列処理用プロセッサアーキテクチャに関する研究に従事．

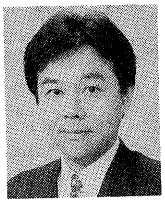


尾形　　航（正会員）

昭和 42 年生．平成 3 年早稲田大学理工学部電気工学科卒業．平成 5 年同大学大学院修士課程修了．現在，同大学理工学部電気電子情報工学科助手．近細粒度並列処理手法，計算機アーキテクチャの研究に従事．

岡本　雅巳（正会員）

平成 2 年早稲田大学電気工学科卒業．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 9 年同大学院理工学研究科電気工学専攻博士後期課程単位取得退学．在学中は並列化コンパイラ，並列実行方式に関する研究を行う．同年（株）東芝入社．現在東芝府中工場勤務．発電監視制御システムの開発に従事．電子情報通信学会会員．



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 年同大学同学部助手．昭和 60 年学術振興会特別研究員．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 年同助教授．平成 9 年同大学電気電子情報工学科教授，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．平成 9 年度情報処理学会坂井記念特別賞受賞．著書「並列処理技術」（コロナ社）．電子情報通信学会，IEEE 等会員．