チップマルチプロセッサ上での粗粒度タスク並列処理によるデータローカライゼーション

中野　啓史[†]　　小　高　剛[†]
木村　啓二[†‡]　　笠原　博徳[†††]

近年，次世代のマイクロプロセッサアーキテクチャとして，複数のプロセッサコアを 1 チップ上に集積するチップマルチプロセッサ (CMP) が大きな注目を集め，研究及び実用化されている．これらの CMP アーキテクチャは，共有キャッシュ等のメモリアーキテクチャを採用しているが，依然として従来のマルチプロセッサシステムで大きな課題となっていたキャッシュやローカルメモリ等のプロセッサコア近接メモリの有効利用に関する問題を抱えている．一方，筆者等はマルチグレイン並列処理との協調動作による実効性能が高く価格性能比の良いコンピュータシステムの実現を目指して，OSCAR CMP を提案している．この OSCAR CMP は，全てのプロセッサコアがアクセスできる集中共有メモリ (CSM) の他に，プロセッサコアのプライベートデータを格納するローカルデータメモリ (LDM) とプロセッサコア間の同期やデータ転送に使用する 2 ポートメモリ構成の分散共有メモリ (DSM) を持ち，これらのメモリをコンパイラが適切に使用するデータローカライゼーションを適用することにより，前述のプロセッサコア近接メモリの有効利用に関する問題に対処する．本稿では，FORTRAN プログラムをループ・サブルーチン・基本ブロックの 3 種類の粗粒度タスクに分割し，粗粒度タスク間の制御依存・データ依存を解析して並列性を抽出する粗粒度タスク並列処理において，配列の生死解析情報を用いて粗粒度タスクの並び替えを行うスタティックスケジューリングアルゴリズムについて述べる．さらに，スケジューリング後のタスクに，生死解析情報を用いて CSM–LDM 間のデータ転送を適切に挿入する手法についても説明する．本データローカライゼーション手法を OSCAR FORTRAN マルチグレイン並列化コンパイラ上に実装し OSCAR CMP 上で評価を行った結果，SPEC 95fp の Tomcatv において，CSM のレイテンシを 20 クロックとしたときに約 1.3 倍，40 クロックとしたときに約 1.6 倍の速度向上がそれぞれ得られた．

Data Localization using Coarse Grain Task Parallelization on Chip Multiprocessor

Nakano Hirofumi[†]　, Kodaka Takeshi[†]　, Kimura Keiji[† ‡]
and Kasahara Hironori[† ††]

Recently, Chip Multiprocessor (CMP) architecture has attracted much attention as a next-generation microprocessor architecture, and many kinds of CMP have widely developed. However, these CMP architectures still have the problem of effective use of memory system nearby processor cores such as cache and local memory. On the other hand, the authors have proposed OSCAR CMP, which cooperatively works with multigrain parallel processing, to achieve high effective performance and good cost effectiveness. To overcome the problem of effective use of cache and local memory, OSCAR CMP has local data memory (LDM) for processor private data and distributed shared memory (DSM) having two ports for synchronization and data transfer among processor cores, in addition to centralized shared memory (CSM). The multigrain parallelizing compiler uses such memory architecture of OSCAR CMP with data localization scheme that fully uses compile time information. This paper proposes a coarse grain task static scheduling scheme considering data localization using live variable analysis. Furthermore, data transfer between CSM and LDM insertion scheme using information of live variable analysis is also described. This data localization scheme is implemented on OSCAR FORTRAN multigrain parallelizing compiler and is evaluated on OSCAR CMP using Tomcatv form SPEC fp 95 benchmark suite. As the results, the proposed scheme gives us about 1.3 times speedup using 20 clocks as the access latency of CSM, and about 1.6 times using 40 clocks as the access latency of CSM respectively against without data localization scheme.

1　はじめに

半導体の集積技術の向上にともない，これまでにない量のトランジスタを 1 チップ上に搭載可能となっている．これまでのマイクロプロセッサでは，増加したトランジスタをスーパースカラ度の増加や投機的実行のメカニズムのために用いて性能向上を果たしてきた．しかしながら，プログラムから抽出できる命令レベル並列性の限界や配線遅延の相対的な増大などの問題から，これらの手法で達成できる性能向上は限界に達しつつある．そのため，次世代のマイクロプロセッサアーキテクチャとして，複数のプロセッサコアを 1 チップ上に集積するチップマルチプロセッサ (CMP) が大きな注目を集め，研究が進められている．

これらのアーキテクチャは，CMP により得られる低レイテンシ・高スループットのプロセッサコア間結合網の活用を一つの目的としている．また，メモリシステムとして，Stanford の Hydra[1] や IMB の Power4[2] のような共有キャッシュ型アーキテク

[†]早稲田大学理工学部電気電子情報工学科
〒 169-8555 東京都新宿区大久保 3-4-1 TEL:03-5286-3371

[†]Dept. of Electrical, Electronics and Computer Engineering, Waseda University
3-4-1 Ohkubo Shinjuku-ku, Tokyo 169-8555, Japan Tel: +81-3-5286-3371
[‡]早稲田大学理工学総合研究センター
[††]アドバンスト並列化コンパイラプロジェクト

チャを採用しているものも多い．しかしながら，これらのアーキテクチャは従来のマルチプロセッサシステムで大きな課題となっていたキャッシュやローカルメモリ等のプロセッサコア近接メモリの有効利用に関する問題を依然として抱えている．

マルチプロセッサシステムにおいて，プログラムの持つデータローカリティを利用した最適化手法は古くから研究されてきた．コンパイラによるデータローカリティ最適化の例としては，複数のループリストラクチャリングを統合して行う Affine Partiioning[3)~5)] が，また，データローカリティを利用する手法としてはループ分割後のタスクの垂直実行 [6)] が提案されている．さらに，シングルプロセッサ上での粗粒度タスク間のローカリティを利用したキャッシュ最適化手法 [7)] も提案されている．次世代マイクロプロセッサアーキテクチャにおけるデータローカリティ最適化の例としては，MIT の Raw Architecture 上でバンクコンフリクトを避ける equivalence-calss unification 及び modulo unrolling[8)] が，FlexRAM 上でデータローカリティを考慮してコードを非対称な二つのプロセッサに自動的にマッピングする手法 [9)] が提案されている．

一方，筆者等は，実効性能が高く価格性能比及びプログラムの生産性の良いコンピュータシステムの実現を目指し，命令レベル並列性を利用する近細粒度並列処理に加え，ループイタレーションレベルの並列性を利用する中粒度並列処理，及びループブロックやサブルーチン間の並列性を利用する粗粒度タスク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と協調動作する OSCAR チップマルチプロセッサ (OSCAR CMP) を提案している [10)]．この OSCAR CMP は，全てのプロセッサコアがアクセスできる集中共有メモリ (CSM) の他に，プロセッサコアのプライベートデータを格納するローカルデータメモリ (LDM) とプロセッサコア間の同期やデータ転送に使用する 2 ポートメモリ構成の分散共有メモリ (DSM) を持つ．これらのメモリをコンパイラが適切に使用するデータローカライゼーション手法を適用することにより，プログラムの持つ並列性とデータローカリティの両方を最大限に活用する．

本稿では，粗粒度タスク並列処理 [11)~13)] におけるループ整合分割 [14),15)] を利用した，チップマルチプロセッサ上での粗粒度タスク並列処理におけるデータローカライゼーション手法を提案する．より具体的には，配列の生死解析情報を用いて粗粒度タスクの並替えを行うスタティックスケジューリングアルゴリズム，及び，スケジューリング後のタスクに，生死解析情報を用いて CSM–LDM 間のデータ転送を適切に挿入する手法を提案する．本手法を OSCAR Fortran マルチグレイン並列化コンパイラ [16)] 上に実装し，OSCAR CMP 上で性能評価を行った．

本論文の構成は以下の通りである．第 2 章では OSCAR Fortran マルチグレイン並列化コンパイラ上での粗粒度タスク並列処理手法，第 3 章ではデータローカライゼーション手法，第 4 章では性能評価についてそれぞれ述べる．

2 粗粒度タスク並列処理

粗粒度タスク並列処理とは，ソースプログラムを疑似代入文ブロック (BPA)，繰り返しブロック (RB)，サブルーチンブロック (SB) の 3 種類のマクロタスク (MT) に分割し，そのマクロタスクを複数のプロセッサエレメント (PE) から構成されるプロセッサグループ (PG) に割り当てて実行することにより，マクロタスク間の並列性を利用する並列処理手法である．

OSCAR マルチグレイン並列化コンパイラにおける粗粒度タスク並列処理の手順は次のようになる．

1. ソースプログラムを階層的に 3 種類のマクロタスクに分割
2. 各階層のマクロタスク間のコントロールフロー，データ依存を解析しマクロフローグラフ (MFG) を生成
3. マクロフローグラフ上の制御依存とデータ依存を考慮してマクロタスク間の並列性を抽出する最早実行可能条件解析を行いマクロタスクグラフ (MTG) を生成
4. MTG がデータ依存エッジしか持たない場合は，マクロタスクはスタティックスケジューリングによって PG または PE にコンパイル時に割り当てられる．一方，MTG が条件分岐などの実行時不確定性持つ場合は，コンパイラがユーザコード中に生成したダイナミックスケジューリングルーチンによって，マクロタスクを PG または PE に実行時に割り当てる．

2.1 マクロタスクの生成

粗粒度タスク並列処理では，まずソースプログラムを BPA，RB，SB の 3 種類のマクロタスクに分割する．

次に 3 章で述べるように，生成された RB がループイタレーションレベルの並列処理が可能な場合，その RB を PG 数やローカルメモリサイズを考慮して異なる複数のマクロタスクに分割し，ループイタレーション間の並列性およびマクロタスク間でのデータローカリティを利用する．

ループ並列処理不可能な実行時間の大きい RB やインライン展開を効果的に適用できない SB に対しては，その内部を階層的に粗粒度タスクに分割して並列処理を行う．

2.2 マクロフローグラフ (MFG) の生成

マクロタスクの生成後，マクロタスク間のコントロールフローとデータ依存を解析し，その結果を表す図 1 に示すようなマクロフローグラフ (MFG) を生成する．

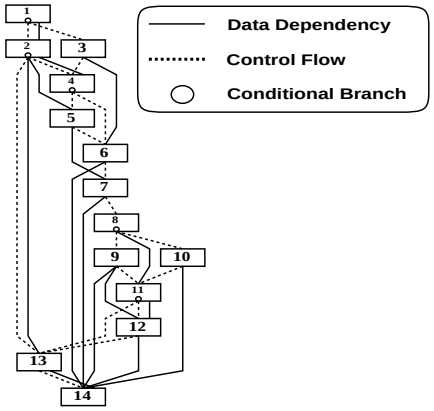

図 1: マクロフローグラフの例

図 1 の各ノードはマクロタスクを表し，実線エッジはデータ依存を，点線エッジはコントロールフローを表す．また，ノード内の小円は条件分岐を表す．MFG ではエッジの矢印は省略されているが，エッジの方向は下向を仮定している．

2.3 マクロタスクグラフ (MTG) の生成

MFG はマクロタスク間のコントロールフローとデータ依存は表すが，並列性は表していない．並列性を抽出するためには，コントロールフローとデータ依存の両方を考慮した最早実行可能条件解析をマクロフローグラフに対して行う．マクロタスクの最早実行可能条件とは，そのマクロタスクが最も早い時点で実行可能になる条件である．

マクロタスクの最早実行可能条件は図 2 に示すようなマクロタスクグラフ (MTG) で表される．

MFG と同様に，MTG におけるノードはマクロタスクを表し，ノード内の小円はマクロタスク内の条件分岐を表している．実線のエッジはデータ依存を表し，点線のエッジは拡張されたコントロール依存を表す．拡張されたコントロール依存とは，通常のコントロール依存だけでなく，データ依存とコントロールフローを複合的に満足させるため先行ノードが実行されないことを確定する条件分岐を含んでいる．

また，エッジを束ねるアークには 2 つの種類がある．実線アークはアークによって束ねられたエッジが AND 関係にあることを，点線アークは束ねられたエッジが OR 関係にあることを示している．

MTG においてはエッジの矢印は省略されているが，下向きが想定されている．また，矢印を持つエッジはオリジナルのコントロールフローを表す．

2.4 スケジューリングコードの生成

粗粒度タスク並列処理では，生成されたマクロタスクはプロセッサグループ (PG) に割り当てられて実行される．PG にマクロタスクを割り当てるスケジューリング手法として，コンパイル時に割り当てを決めるスタティックスケジューリングと実行時に割り当てを決めるダイナミックスケジューリングがあり，マクロタスクグラフの形状，実行時不確定性などを元に選択される．

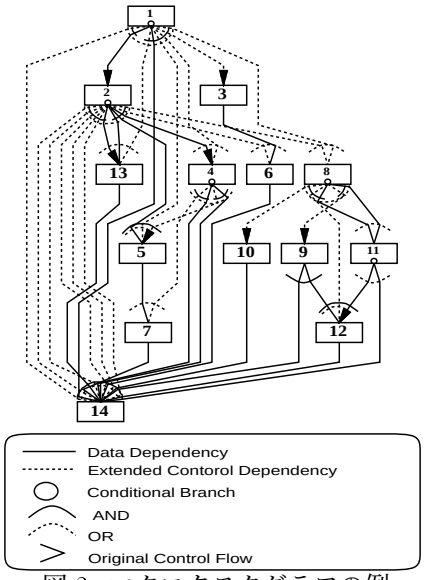

図 2: マクロタスクグラフの例

スタティックスケジューリングは，マクロタスクグラフがデータ依存エッジのみを持つ場合に適用され，コンパイラがコンパイル時にマクロタスクの PG への割り当てを決定する方式である．スタティックスケジューリングでは，実行時スケジューリングオーバーヘッドを無くし，データ転送と同期のオーバーヘッドを最小化することが可能である．

スタティックスケジューリングアルゴリズムの詳細については第 3 章で説明する．

3 データローカライゼーション

この章では配列の生死解析情報を利用した粗粒度タスクスタティックスケジューリングアルゴリズムおよびローカルメモリ管理，データ転送を行うデータローカライゼーション手法について述べる．

本手法ではまず，同一のプロセッサグループへ割り当てられるデータローカライゼーショングループ (DLG) 内のマクロタスクで定義 · 参照されるデータ量がローカルメモリサイズ以下となるようにマクロタスクを分割する．次に配列の生死解析情報を用いたローカルメモリ上のデータのシミュレーションを使い，粗粒度タスクスタティックスケジューリングを行う．最後に，スケジューリング結果をもとに必要となるデータ転送の計算，データのメモリ配置を行い，データローカライゼーションを実現している．

3.1 マクロタスク分割

プログラム全域に渡り，ローカルメモリの有効利用を考える場合，定義 · 参照するデータ量がローカルメモリサイズを越えてしまうと，どのようにマクロタスクを割り当てたとしてもデータをローカルメモリから共有メモリへと書き戻す必要が起こり，ローカルメモリを用いた効率的なデータの授受が行えなくなる．そこで，このような場合マクロタスク

の分割を行い，一つのデータローカライゼーショングループ内のマクロタスクが定義・参照するデータ量をローカルメモリサイズ以下となるようにする．

マクロタスク分割の際，異なるプロセッサグループに割り当てられたマクロタスク間でのデータ転送を低減し，同一プロセッサグループに割り当てられたマクロタスク間でローカルメモリを介したデータ授受を行うためには，各マクロタスクにおけるデータの使用範囲がなるべく等しくなるように，複数のループを整合して分割する必要がある．そこで，マクロタスク間のデータ共有量と並列性の両方を考慮する分割手法であるループ整合分割 [14),15)] を利用してマクロタスク分割を行う．

ループ整合分割は任意形状のマクロタスクグラフに対して適用される．ループ整合分割の適用単位となるターゲットループグループ (TLG) は以下のように定義される． ターゲットループグループはマクロタスクグラフのクリティカルパス上の RB，及びその RB に直接先行接続あるいは直接後続接続された RB で構成される．これらの対象 RB は Doall ループ，リダクションループ，シーケンシャルループを仮定している．

次に TLG 毎にループ間におけるイタレーションに関するデータ依存を解析し，データの使用範囲がなるべく等しくなるように各ループを整合分割する．ターゲットループグループに選択されたマクロタスクの内，グラフのクリティカルパス上のマクロタスクをループ整合分割してできたマクロタスクをデータローカライゼーショングループ (DLG) として定義する．同一データローカライゼーショングループ内のマクロタスクは同一プロセッサグループへ割り当てられる．

3.2 スケジューリングアルゴリズム

データを共有している二つのマクロタスクを異なるプロセッサグループへ割り当てた場合，それらのプロセッサグループ間でデータ転送が必要になる．データ転送を最小化するためには，あるプロセッサグループ PG_i に既に割り当てられたマクロタスク MT_j とデータ共有量の多いマクロタスク MT_k は並列性を損なわない範囲で同一のプロセッサグループ PG_i に割り当てる必要がある．そこで，あるプロセッサグループとマクロタスクの組合わせ毎に次のようにデータ転送ゲインを定義し，プロセッサグループ間データ転送の最小化を行う．ここで，プロセッサグループ PG_i とマクロタスク MT_j のデータ転送ゲイン $Gain_{ij}$ は， PG_i のローカルメモリ上のデータと MT_j とのデータ共有量として定義する．ただし，ローカルメモリ上のデータは次のように推定する．マクロタスク MT_j を PG_i に割り当てる場合，まず MT_j 内で定義・参照される配列分の領域をローカルメモリ上に確保する．このとき，容量不足でローカルメモリ上に配列が確保できない場合は LRU アルゴリズムを用い書き戻す配列を決定し，必要な容量を確保する．次に配列毎の生死解析情報に基づき，MT_j で死ぬ配列の領域を解放し，後続 MT 用のローカルメモリ上の配列領域確保に備える．

以上の前提をもとに本手法で実装したスケジューリングアルゴリズムである DLG を考慮したデータ転送ゲイン/CP/MISF スケジューリング法 [15)] は以下の通りとなる．ここで，DLG_MT とはデータローカライゼーショングループに属するマクロタスク，NOT_DLG_MT とは DLG_MT 以外のマクロタスクとする．

手順 1 レディタスク中に DLG_MT があれば手順 2 を，なければ手順 5 を選択する．

手順 2 PG_j に既に割り当てられたマクロタスクと同一の DLG に属するマクロタスクがレディタスク中にあれば，手順 3 を，なければ手順 4 を選択する．

手順 3 レディタスク中に存在する DLG の内，PG_j に最後に割り当てられた DLG_MT と同一の DLG_i に属する $DLG_i\_MT_k$ を割り当てる．手順 6 に行く．

手順 4 CP/MISF [16)] のプライオリティに従い，DLG_MT を割り当てる．手順 6 に行く．

手順 5 最大のデータ転送ゲインを与えるマクロタスクのプロセッサグループへの割り当てを一つ選択する．同一のデータ転送ゲインを与える割り当てが二つ以上あれば CP/MISF のプライオリティに従いマクロタスクを割り当てる．手順 6 に行く．

手順 6 以上の手順のいずれかにより割り当てられたマクロタスクが割り当てられたプロセッサグループとレディマクロタスクとのデータ転送ゲインを計算する．もしレディタスクがあれば手順 1 へ行き，なければスケジューリングを終了する．

3.3 データ転送の挿入及びメモリ配置

スタティックスケジューリングの後，データ転送命令の挿入，メモリ配置の決定を行うが，その概要は以下の通りとなる．まず，DLG 内の MT からたかだか一回しか定義・参照されない配列をローカル化対象配列から除外する．次に各マクロタスク毎に必要となるデータ転送を計算する．マクロタスク MT_i 実行時にロードする必要のある配列は MT_i に生きて入り，なおかつ MT_i 内で参照される配列となる．次にスケジューリング結果に従い，先行マクロタスクを順に辿り，この配列の生産者を探す．生産者のマクロタスクにストア命令を付加する．ある配列の生産者が同一のマクロタスクであるとき後続のロード命令を削除する．また，生産者が直接ローカルメモリにストアしている場合も後続のロード命令を削除する．最後に共有メモリ上の配列名をローカルメモリ上の配列名に書き換えてデータローカライゼーションを実現している．

4 性能評価

本章では本論文で提案した手法の性能評価について述べる．

4.1 OSCAR Fortran マルチグレイン並列化コンパイラ

本手法を実装した OSCAR Fortran マルチグレイン並列化コンパイラについて述べる． OSCAR Fortran マルチグレイン並列化コンパイラはフロントエンド，ミドルパスおよび複数のバックエンドから構成される．

フロントエンドは Fortran77 および OpenMP Fortran を読み込み，中間言語を生成する．

ミドルパスは，コントロールフロー解析，データ依存解析を行い，プログラムのリストラクチャリング，マクロタスクの生成および，並列性の自動抽出を行なう．さらに，データローカライゼーションを考慮した粗粒度タスクのスタティックスケジューリング，データ転送の挿入及びメモリ配置を行う．これらの解析結果に基づき並列化された中間言語を生成する．

OSCAR Fortran マルチグレイン並列化コンパイラは，OSCAR マルチプロセッサシステム， UltraSparc，富士通 VPP， STAMPI，OpenMP といった様々なターゲット用のバックエンドを持ち，ミドルパスが出力した並列化中間言語から，各ターゲット用のアセンブリコード，もしくは並列化 Fortran を生成する．今回は UltraSparc バックエンドを使用し，粗粒度タスク並列処理によるデータローカライゼーションを実現した．

4.2 評価環境

性能評価に用いた OSCAR チップマルチプロセッサ (CMP) およびベンチマークアプリケーションについて述べる．

OSCAR CMP のネットワークおよびメモリアーキテクチャは，図 3 のように CPU，データ転送ユニット (DTU)，ローカルプログラムメモリ (LPM)，ローカルデータメモリ (LDM) および分散共有メモリ (DSM) を持つ複数のプロセッサエレメント (PE) を相互接続網 (バス結合，クロスバ結合など) で接続し 1 チップ上に搭載したアーキテクチャである．今回の評価では，データ転送を CPU の処理とオーバーラップして行なえる DTU についてはオーバーラップデータ転送スケジューリングアルゴリズムの開発が終っていないため利用してない．また，PE 間相互結合網は 3 本バスを利用している．

今回の評価では実行開始時には配列は全て CSM に配置し，ローカライゼーション適用プログラムは必要に応じ LDM へのロードとストアを行う．また，1 プロセッサエレメントを 1 プロセッサグループとして評価を行う．今回はプログラムの並列性のみの比較を行うために LDM の容量を無制限として評価を行う．そのためプロセッサ台数で TLG 中の各ループをループ整合分割している．評価に用いた各メモリのアクセスレイテンシを表 1 に示す．

計測にはクロックレベルの精密なシミュレータを用いる．なお，評価ベンチマークにはシミュレーション時間短縮のため，データセットを 513 から 65 に，収束ループの回転数を 750 から 200 に縮小した SPEC 95fp の Tomcatv を用いる．

表 1: OSCAR CMP のメモリアクセスレイテンシ

メモリ	アクセスレイテンシ (clock)	
LPM	1	2
LDM	1	2
DSM	1(内部)4(リモート)	2(内部)6(リモート)
CSM	20	40

4.3 性能評価結果

CSM アクセスレイテンシを 20 クロックとしたときの性能評価結果を図 4 に，40 クロックとしたときの性能評価結果を図 5 にそれぞれ示す．図中の縦軸はローカライゼーション未適用プログラムの 1PE 時の実行クロック数を 1 としたときの速度向上率を，横軸は評価を行ったプロセッサ数を表す．また，図中の local がローカライズ適用プログラムの，no local が未適用プログラムの速度向上率をそれぞれ表す．

図 4，図 5 において，ローカライゼーション適用未適用に関わらず，4PE まではプロセッサ台数に対しスケーラブルな性能向上を示している．8PE ではスケーラブルな性能向上が得られていない．これは PE 間相互結合網に使われている 3 本バスまた集中共有メモリが 3 ポートメモリであるために，データのロード・ストアがプロセッサ間で衝突し，CSM のデータ供給能力がプロセッサ数に追い付かないことが原因と考えられる．同じプロセッサ数同士で比較するとローカライゼーション適用時は未適用時に比べ，集中共有メモリのアクセスレイテンシが 20 クロックの時におよそ 1.3 倍，40 クロックの時におよそ 1.6 倍の速度向上を示している．このことから，集中共有メモリのアクセスレイテンシが大きいときによりデータローカライゼーションの効果が大きいことが分かる．

5 まとめ

本稿では，チップマルチプロセッサ上での粗粒度タスク並列処理によるデータローカライゼーションについて述べた．本手法を OSCAR Fortran マルチグレイン並列化コンパイラ上に実装し，OSCAR チップマルチプロセッサ上で性能評価を行った．その結果 SPEC 95fp の Tomcatv において，データローカライゼーション未適用時と比較し，集中共有メモリのレイテンシを 20 クロックとした時約 1.3 倍，40 クロックとした時約 1.6 倍の速度向上が得られた．

本研究の一部は NEDO アドバンスト並列化コンパイラプロジェクト，STARC “コンパイラ協調型シングルチップマルチプロセッサの研究開発” 及び 早稲田大学理工総研プロジェクト研究「アドバンスト並列化コンパイラ」により行われた．

参考文献

[1] Hammond, L., Hubbert, B., Siu, M., Prabhu, M. K., Chen, M. and Olukotun, K.: The Stanford HYDRA CMP, *IEEE MICRO Magazine*, Vol. 20, No. 2, pp. 71–84 (2000).

[2] Tendler, J. M., Dodson, S., Fields, S., Le, H. and Sinharoy, B.: POWER4 System Microarchitecture, *Technical White Paper* (2001).

[3] Lim, A. W., Cheong, G. I. and Lam, M. S.: An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication, *Proc. 13th ACM SIGARCH International Conference on Supercomputing* (1999).

[4] Lim, A. W., Liao, S. and Lam, M. S.: Blocking and Array Contraction Across Arbitrarily Nested Loops Using Affine Partitioning, *Proc. of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2001).

[5] Lim, A. W. and Lam, M. S.: Cache Optimizations With Affine Partitioning, *Proc. of the Tenth SIAM Conference on Parallel Processing for Scientific Computing* (2001).

[6] Vajracharya, S., Karmesin, S., Beckman, P., Crotinger, J., Malony, A., Shende, S., Oldehoeft, R. and Smith, S.: SMARTS: exploiting temporal locality and parallelism through vertical execution, *Proc. of the 1999 international conference on Supercomputing* (1999).

[7] 稲石, 木村, 藤本, 尾形, 岡本, 笠原: 最早実行可能条件解析を用いたキャッシュ利用の最適 化, 情報処理学会研究報告 ARC (1998).

[8] Barua, R., Amarasinghe, S. and Agarwal, A.: Compiler Support for Scalable and Efficient Memory Systems, *IEEE Transactions on Computers* (2001).

[9] Lee, J., Solihin, Y. and Torrellas, J.: Automatic Code Mapping on an Intelligent Memory Architecture, *IEEE Transactions on Computers, Special Issue on High Performance Memory Systems* (2001).

[10] 木村, 尾形, 岡本, 笠原: 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, 情報処理学会論文誌, Vol. 40, No. 5, pp. 1924–1934 (1999).

[11] 岡本, 合田, 宮沢, 本多, 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).

[12] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. 12th Workshop on Languages and Compilers for Parallel Computing* (2000).

[13] 石坂, 八木, 小幡, 吉田, 笠原: 共有メモリマルチプロセッサシステム上での粗粒度タスク並列実現手法の評価, 情報処理学会研究報告 ARC (2001).

[14] 吉田, 越塚, 岡本, 笠原: 階層型粗粒度並列処理における同一階層内ループ間データローカライゼーション手法, 情報処理学会論文誌, Vol. 40, No. 5, pp. 2054–2063 (1999).

[15] 吉田, 八木, 笠原: SMP 上でのデータ依存マクロタスクグラフのデータローカライゼーション手法, 情報処理学会研究報告 2001-ARC-141 (2001).

[16] 笠原: 並列処理技術, コロナ社 (1991).

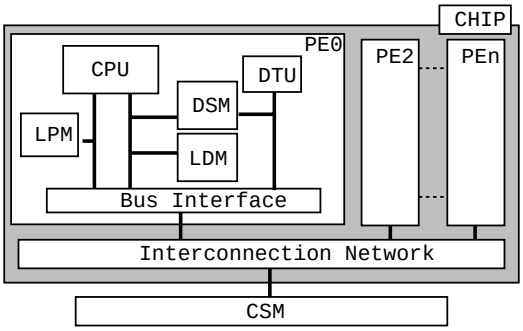

図 3: OSCAR CMP アーキテクチャ

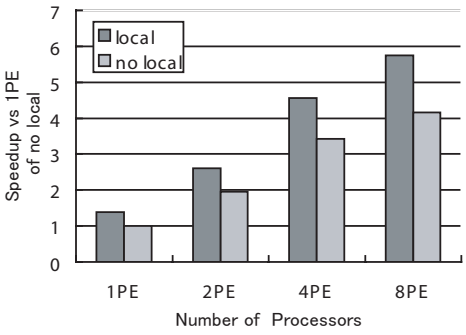

図 4: CSM アクセスレイテンシが 20 クロックの時の速度向上率

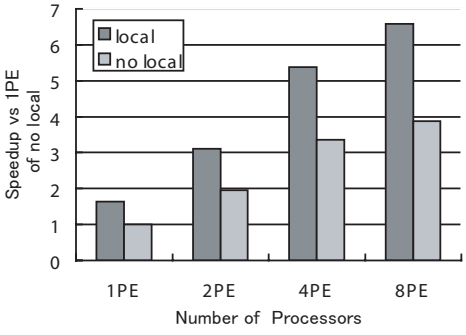

図 5: CSM アクセスレイテンシが 40 クロックの時の速度向上率