社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (11)
2007－HPC－109 (11)
2007／3／1

マルチグレイン並列化コンパイラにおけるローカルメモリ管理手法

三浦 剛[†] 田川友博[†] 村松裕介[†]
池見明紀[†] 中川正洋[†] 中野啓史[†]
白子 準[†] 木村啓二[†] 笠原博徳[†]

半導体集積度向上に伴うスケーラブルな性能向上，低消費電力，価格性能を達成するためにマルチコアプロセッサが大きな注目を集めている．消費電力を抑えつつマルチコアプロセッサの実効性能を向上させ，アプリケーションソフトウェアの開発期間を短縮するためには自動並列化コンパイラが重要な役目を果たす．この実効性能の向上のためには，マルチコアプロセッサにおいても，プロセッサとメモリ動作速度のギャップに起因するメモリウォール問題への対処が必要となる．具体的には，プロセッサに近接したキャッシュやローカルメモリ等の高速メモリの有効利用が実効性能向上のために必須である．本稿では，OSCAR マルチグレイン自動並列化コンパイラを用いた粗粒度タスク並列処理において，プログラム全域のデータローカリティを有効利用した，マルチコア上のローカルメモリ管理手法について提案する．SPEC 95fp の tomcatv を用いた OSCAR マルチコアシミュレータ上の性能評価において，本手法未適用時の共有メモリを用いた逐次処理に対して，本手法適用によるローカルメモリ利用最適化により，8 プロセッサで 19.6 倍の性能向上が得られた．

A Local Memory Management Scheme in Multigrain Parallelizing Compiler

Tsuyoshi Miura[†], Tomohiro Tagawa[†], Yusuke Muramatsu[†], Akinori Ikemi[†], Masahiro Nakagawa[†], Hirofumi Nakano[†], Jun Shirako[†], Keiji Kimura[†] and Hironori Kasahara[†]

Multicore systems have been attracting much attention for performance, low power consumption and short hardware/software development period. To take the full advantage of multiprocessor systems, parallelizing compilers serve important roles. On multicore processor, a memory wall caused by the speed gap between processor core and memory is also serious problem. Therefore, it is important for performance improvement to use fast memolies like cache and local memory nearby a processor effectively. This paper proposes a local memory management scheme for coarse grain task parallel processing. In the evaluation using SPEC 95fp tomcatv, the proposed scheme using 8 processors achieved 19.6 times speedup against the sequantial execution without the proposed scheme on the OSCAR multicore processor by the effective use of local memories.

1 はじめに

従来，マイクロプロセッサの性能向上の牽引力になっていた命令レベル並列性の利用と周波数の向上は半導体集積度の向上と共に，並列性抽出の限界，消費電力の増大のため，進展が難しくなっている．これらを解決する技術としてマルチコアプロセッサが注目を集めている[1)~5)]．マルチコアプロセッサは複数のプロセッサコアを一つのチップ上に集積するため，プロセッサコア間で命令レベル並列性以外の粗い粒度の並列性も利用可能となる．また，各プロセッサコアを低周波数で動作させ，適切に並列処理することで，より高性能，低消費電力が実現可能なアーキテクチャとなっている．一方で，マルチコアでも従来より問題となっていたメモリウォールの問題への対応は重要であり，キャッシュやローカルメモリ等のチップ内プロセッサの近接メモリの有効利用を行う必要がある．

ローカルメモリは全てのメモリアクセスをユーザあるいはコンパイラが制御するなため，キャッシュと比較すると，組み込み向けアプリケーションで必要となるリアルタイム制約への対応が容易となる．しかし，ローカルメモリを持つアーキテクチャではプログラムの挙動に応じ，ローカルメモリに配置するデータの選択およびメモリ配置，そしてオフチップメモリとローカルメモリ間のデータ転送をプログラマあるいはコンパイラが適切に制御する必要がある．プログラマがローカルメモリ管理やデータ転送命令挿入を行っていては生産性も上がらず，エラーの温床ともなりうる．そこで，筆者等はコンパイラによるデータローカライゼーション手法[6)]を用い，プログラムの複数ループ間でのデータローカリティの最適化およびローカルメモリ管理とデータ転送命令挿入の自動化を行う手法を提案し，OSCAR コンパイラに実装している．

コンパイラによる初期のローカルメモリ管理に関する研究としてはデータのローカルメモリへの静的割り付け[7),8)]により実現したものがある．静的割り付けではプログラムの開始から終了までプログラム中で頻繁に参照されるデータについてのみローカルメモリに配置し，それ以外のデータについてはオフチップのメモリ上に配置される．そのため，プログラムの挙動に応じた効率的なローカルメモリの利用ができない．静的割り付けの問題点を解決する手法としては，プログラム中のループの挙動をコンパイル時に解析し，ローカルメモリとオフチップメモリ間で適切にデータ転送を行い，ローカルメモリ上の同じ領域を異なる用途で使い回す動的なローカルメモリ管理手法[6),9),10)]が提案されている．また，ループをタイリングすることで，あるループ中のネストにおけるデータのアクセス量をローカルメモリサイズ以下に抑え，そのネストにおけるデータをローカルメモリへ配置する手法[11)]が提案されている．これらの手法はいずれもプログラム

[†]早稲田大学理工学部コンピュータ・ネットワーク工学科
〒 169-8555 東京都新宿区大久保 3-4-1 Tel: 03-5286-3371

[†]Department of Computer Science,
School of Science and Engineering, Waseda University 3-4-1 Ohkubo, Shinjuku-ku, Tokyo, Japan 169-8555 Tel: +81-3-5286-3371

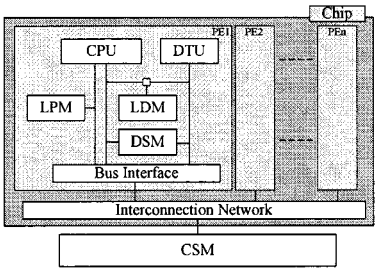

図 1: OSCAR マルチコアアーキテクチャ

中の特定のデータあるいは特定のループを対象としたものである．

本稿ではプログラムを大域的に解析し，プログラム全域のデータローカリティを有効利用した，マルチコア上のローカルメモリを管理する手法を提案する．具体的には以下の通りである．粗粒度タスク並列処理において，コンパイラは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．次に異なるプロシージャを含むプログラム全域から，同じ配列にアクセスし，データ依存関係にあるループを集め，それらをグループ化し，ループ整合分割する．これにより，プログラム全域に渡り，アクセスされるデータ量をローカルメモリサイズ以下に抑えることが可能となり，プログラムの各部でデータのローカルメモリへの割り当てが可能になる．分割されたマクロタスクを並列性とデータローカリティを考慮して，粗粒度タスクスケジューリングすることで，粗粒度タスク間のデータローカリティの有効利用を行う．その後，マクロタスクでアクセスされる配列についてローカルメモリへの割り当てを行い，データ転送を挿入する．

本稿の構成を以下に示す．第 2 章では対象とするマルチコアアーキテクチャである OSCAR マルチコアアーキテクチャについて述べる．第 3 章では粗粒度並列処理手法について述べる．第 4 章ではローカルメモリ管理手法について述べる．第 5 章では本手法の性能評価を簡単なサンプルコードおよび SPEC 95fp より tomcatv を用いて行う．第 6 章で本稿のまとめを述べる．

2 OSCAR マルチコアアーキテクチャ

OSCAR マルチコアアーキテクチャはマルチグレイン自動並列化コンパイラとの協調動作により，実効性能が高く価格性能比のよいコンピュータシステムの実現を目指したコンパイラ協調型アーキテクチャである．

OSCAR マルチコアアーキテクチャを図 1 に示す．OSCAR マルチコアは 1 つのチップ上に複数のプロセッサエレメント (PE) を持つ．各 PE は単純な一命令発行の in-order プロセッサコア，プロセッサプライベートなデータを保持する 1 ポートのローカルデータメモリ (LDM)，共有データや同期変数を保持する 2 ポートの分散共有メモリ (DSM)，プログラムコードを保持するローカルプログラムメモリ (LPM)，そして CPU と非同期にバースト転送が可能なデータ転送ユニット (DTU) を持つ．チップ上の全ての PE はバスやクロスバといった Interconnection Network によって接続されている．さらに本稿の評価では集中共有 (CSM) メモリがチップ外に接続されていると仮定している．

2.1 データ転送ユニット (DTU)

OSCAR マルチコア上のデータ転送ユニット (DTU) について説明する．DTU は連続転送とストライド転送を行う機能を持つ．このための DTU 制御命令およびパラメータはコンパイラが自動生成する．

DTU の起動には二種類の方法がある．一つはコンパイラが生成したパラメータをローカルメモリ上に設定し，

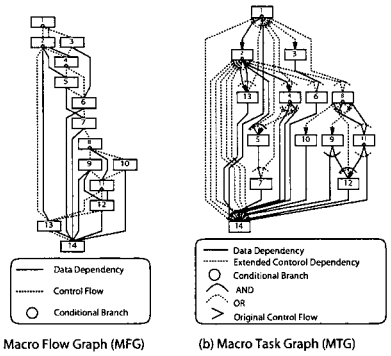

図 2: マクロフローグラフとマクロタスクグラフ

実行時に転送パラメータの先頭アドレスを DTU に通知し，DTU を駆動する方法である．このとき，複数のパラメータをローカルメモリ上の連続する領域に設定しておけば，パラメータチェインが形成され，CPU による一度の駆動で複数の領域を転送することが可能となっている．もう一つは CPU が転送パラメータ値を直接 DTU のレジスタに設定し，駆動する方法である．

3 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．

3.1 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 [12] に分割する．

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 [13]．

3.2 粗粒度並列性抽出

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，マクロフローグラフ (MFG)[12),14)] を生成する．

次に，階層的に生成されたマクロフローグラフに対し最早実行可能条件解析 [12),14)] を適用し，階層的なマクロタスクグラフ (MTG)[12),14)] を生成する．最早実行可能条件とは，制御依存とデータ依存を考慮したマクロタスクの最も早く実行を開始してよい条件であり，マクロタスクグラフは粗粒度タスク並列性を表す．マクロフローグラフ及びマクロタスクグラフの例を図 2 に示す．

次に，4.1 節で述べるように，マクロタスクグラフ中の 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．分割が行われた後のマクロタスクグラフにおいて，4.2 節で述べる配列依存範囲解析を行い，マクロタスク間の配列フロー依存・入力依存を解析する．

3.3 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．

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．

本手法では，スタティックスケジューリングを行うことができるマクロタスクグラフを対象として 4.4 節で述べるローカルメモリ割り当てを行う．

4 ローカルメモリ管理

OSCAR コンパイラで実現していたデータローカライゼーション手法では，大量の配列データを共有する直接先行接続または直接後続接続されたループを集めてターゲットループグループ (Target Loop Group: TLG) と呼ぶループ群を作り，並列性とキャッシュメモリやローカルメモリサイズを考慮して，小さな部分ループにループ整合分割 (Loop Aligned Decomposition: LAD)[6),15),16)] する．次に，並列性とデータローカリティを考慮しながら，一度高速なプロセッサ近接メモリに配置したデータを複数の部分ループ間で連続的にアクセスするようにスタティックスケジューリングを行う．

キャッシュメモリアーキテクチャと異なり，ローカルメモリを持つアーキテクチャにおいて効率的なプログラムの実行を行うには，プログラム中でアクセスされるデータをローカルメモリに明示的に割り当て，適切にデータ転送を挿入する必要がある．その際，各マクロタスクでアクセスされるデータサイズがローカルメモリサイズと比べて大きい場合にはグローバルループ整合分割を行い，各マクロタスクでアクセスされるデータサイズを縮小する必要がある．

本手法では，フラグメンテーションを避けるために，ローカルメモリを Block と呼ぶ固定長の領域に分割して管理を行う．プログラム中に出現する分割された配列のサイズに応じ，Block 内部は整数分の一の大きさの Sub Block に分割して管理する．本論分では一例として，2 のべき乗分の 1 の大きさの Sub Block に分割して管理する．

本ローカルメモリ管理手法は，グローバルループ整合分割によるマクロタスクの分割と，分割された配列を割り当てる Block の決定から構成される．従来のループ整合分割手法では複数の Doall ループを中心とした TLG 単位で分割が行われるため，配列ごとにある一つの次元しかアクセス範囲を分割縮小することができず，マクロタスクでアクセス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分割できない場合も生じた．これに対し，グローバルループ整合分割ではローカルメモリサイズに応じ，ネストしたループの各レベルで分割を行うことで，配列の複数次元に渡り，そのアクセス範囲の縮小を行い，マクロタスクでアクセスされるデータ量をローカルメモリサイズ以下に分割することができる．なお，前処理としてループディストリビューションやループフュージョンを行い，完全ネストループを増やし，効率よくグローバルループ整合分割を行うことができるようにしておく．

分割された各マクロタスクに対して，配列依存範囲解析を行い，解析結果を基にデータ転送コストを見積もりながら，マクロタスクの実行順序を決定するスタティックスケジューリングを行う．複数のヒューリスティックアルゴリズムを用いてスケジューリングを行い，スケジューリング長が最も短くなるものを採用する．この段階ではタスク間のデータ転送を最小化できるようにマクロタスクのプロセッサへの割り当て及び実行順序を決定するだけで，メモリ管理は行わない．

最後にスケジューリング結果とマクロタスク間の配列依存範囲解析結果を基にマクロタスク内でアクセスされるデータのローカルメモリへの割り当てを行い，配列依存範囲解析結果を用いてコンパイラによって自動的に生成される DTU 転送命令を使い，DTU によってタスク処理とオーバーラップして効率的に転送される．

4.1 グローバルループ整合分割

グローバルループ整合分割では，従来のループ整合分割手法で用いられていたローカリティ利用の単位であるターゲットループグループ (Target Loop Group: TLG) を基に，4.1.1 項で述べる TLG 集合を生成する．複数のネストレベルによる分割を行うことを可能とするために，4.1.2 項で述べる整合可能ループグループ集合 (Alignable Loop Group set: ALG set) を生成し，分割後の ALG set 内のマクロタスク内で扱うデータがローカルメモリサイズ以下となるように複数のネストレベルによる分割を行う．

4.1.1 TLG 集合生成

大量の配列データを共有するマクロタスクグラフ上で直接先行接続または直接後続接続された整合可能なループを集めてターゲットループグループ (Target Loop Group: TLG) と呼ぶループ群を作る．ここで，2 つのループが整合可能であるとは，以下の整合条件 1〜4 を満たすことである [17)]．

1. 各ループが Doall ループ，Reduction ループ，ループキャリッドデータ依存 (リカレンス) による Sequential ループのいずれかである
2. ループ間に配列変数のデータ依存が存在する
3. それぞれのループのループ制御変数が同一配列の同じ次元の添え字式で使用されており，次元の配列添字がループ制御変数の一次式で表されている
4. ループ間にデータ依存を生じる各配列に対して，配列添字中のループ制御変数係数のループ間での比が一定

従来のループ整合分割手法では，同一の階層内に存在するマクロタスク間でのローカリティの利用を考えており，TLG として選ばれたループの内側にループが存在していても，内側ループに対して TLG 生成が行われなかった．これに対し本手法では，いずれかの TLG に選ばれたループの内側にもループが存在していた場合，内側のループに対しても TLG を生成する．また，他のループと整合可能でないループは，そのループ単独で TLG を成すものとする．

図 3(a) から成るサンプルコードに対し，TLG 生成を行った様子を図 3(b) に示す．従来のループ整合分割手法では，図 3(b) 中の最外側ネストである i ループが存在する階層でのローカリティ利用のみを考えていたため，TLG 1 のみが生成されていた．それに対し，本手法ではネストされた j, k の各ループに対しても TLG 生成を行う．

次に，プログラム全域から整合可能な TLG を集めて TLG 集合を生成する．ここで，2 つの TLG が整合可能とは，TLG 中の全てのループ間で整合条件 1〜4 を満たすことを言う．ただし，互いに異なるプロシージャに存在する 2 つの TLG が同一の TLG 集合に選ばれるために

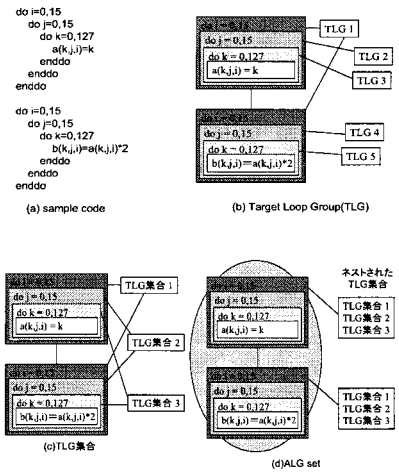

図 3: TLG，TLG 集合および ALG set 生成

は，共有配列の形状が一致している必要がある．Fotranにおいては整合配列や擬寸法配列を用いることにより，サブルーチンの呼び出し元と呼び出し先とで配列の次元数や各次元の宣言サイズが異なっている場合がある．異なるTLG内で共有配列の次元数や各次元の宣言サイズが異なっている場合，整合条件1〜4を満たしていた場合でもそれらのTLGは同一のTLG集合には選ばない．また，他のTLGと整合可能でないTLGは，そのTLG単独でTLG集合を成すものとする．

図3(b)で生成されているTLGを基にTLG集合を生成した様子を図3(c)に示す．図3(c)においてまずTLG 1に着目すると，TLG 1に属しているループのループ制御変数は配列a及びbの3次元目の添字で使用されており，他のTLGに属しているループのループ制御変数では配列aまたはbの3次元目の添字で使用されることはない．よってTLG 1単独でTLG集合を成す．TLG 2に着目すると，TLG 2に属しているjループのループ制御変数は配列aの2次元目の添字で使用されており，ループ制御変数の係数は1である．これはTLG 4に属しているjループも同様である．よってTLG 2とTLG 4は同一のTLG集合に選ばれる．TLG 3とTLG 5についても同様に同一のTLG集合に選ばれる．

TLG集合中のループについて，Inter Loop Dependence(ILD)[18),19)]を解析する．TLG集合中で処理コストのもっとも大きなマクロタスクを標準ループとして選択する．標準ループ以外のループの何番目のイタレーションが標準ループのk番目のイタレーションに依存されるか，あるいは標準ループのk番目のイタレーションに依存するかを解析するのがILD解析である．さらに，TLG集合のグループ標準インデックス範囲[18)]を計算する．グループ標準インデックス範囲とは，TLG集合内の全てのループにおけるデータのアクセス範囲を，標準ループのインデックス範囲に換算したものである．

4.1.2 整合可能ループグループ集合生成

次に，ネストされたループにおいて，各ループがどのTLG集合に属しているかを調べる．ネストされたループで，いずれかのTLGに選ばれている最も外側のループを基準とし(以降，最外側ループと表記)，そのループに完全ネストされているループまでを対象として，それぞれのループがどのTLG集合に属しているかを調べる(以降，ネストされたTLG集合と表記)．最外側ループによって完全ネストされるループが無い場合は，最外側ループが属しているTLG集合のみをネストされたTLG集合として考える．

最も多くネストされたTLG集合を持つ最外側ループを基準として，そのネストされたTLG集合の部分集合となるようなネストされたTLG集合を持つ最外側ループを整合可能ループグループ集合(Alignable Loop Group set: ALG set)としてグルーピングする．ALG setの基準となったループ以外一つも最外側ループが選ばれなかった場合は，基準となったループ単独でALG setを構成する．まだいずれのALG setに含まれていない最外側ループに対して同様の処理を行い，全ての最外側ループがいずれかのALG setに選択されるまでこれを繰り返す．

図3(b)において，二つのiループはTLG 1に選ばれているため，それぞれ最外側ループとなる．それぞれのiループに対して，内側に存在するj,kの各ループはiループに完全ネストされているため，各最外側ループは内側に存在するj,kの各ループが図3(c)においてそれぞれどのTLG集合に属しているかを調べる．その結果，各最外側ループが持つネストされたTLG集合は，最外側ループ自身の情報も含めて図3(d)のようにTLG集合1,2および3となる．ここで，二つの最外側ループが持つネストされたTLG集合が一致しているので，これら二つのjループは同一のALG setに選ばれる．このALG setが持つネストされたTLG集合はTLG集合1,2及び3となる．

4.1.3 Blockサイズ決定

次に，生成されたALG setのうち，それに属するループの総コストが最も大きいものを基準としてBlockサイズを決定する．ネストされたTLG集合を分割し，ALG set内の分割後の最外側ループでアクセスされる全てのデータがローカルメモリに配置できるように分割を行う．ローカルメモリや分散共有メモリにはユーザプログラム中のデータ以外にも同期用変数やデータ転送用パラメータなどを配置する必要があり，分割を行う前の段階ではそれらがどの程度のサイズを占めるかが不明なため，実際には分割後の最外側ループの総データサイズがローカルメモリサイズよりも小さくなるように分割を行う．

具体的なBlockサイズ決定手法について述べる．最外側ループが属しているTLG集合の分割数から考える．この時，解析された並列性を損なわないようにするために，各最外側ループに割り当てられたPG数を調べ，その最小公倍数を仮分割数とする．対象TLG集合のグループ標準インデックス範囲を仮分割数で割り，それを基に分割後の各ループがアクセスするデータサイズを計算する．この時，各配列において次元ごとに，アクセス範囲よりも大きな2のべき乗の数を求め，それを用いてデータサイズを計算する．計算された配列データサイズのうち最も大きなものを仮Blockサイズとし，ローカルメモリ上に配置できる仮Block数を計算する．

計算された仮Blockサイズおよび仮Block数を用いて，ALG set中でアクセスされるデータ全てをローカルメモリ上に配置可能ならば，仮分割数で対象TLG集合内の各ループを分割することを決定し，同時にBlockのサイズ，使用できるBlockの個数も決定する．この時，どの配列がどのようなアクセスパターンをするかを保持しておき，ローカルメモリ割り当てを行うときに，どのデータがどの大きさのBlockまたはSub Blockを使用すべきかわかるようにしておく．

配置できない場合は仮分割数を大きくして同様の計算を行う．着目しているTLG集合について，それ以上仮分割数を大きくすることができなくなってもまだローカルメモリ上にデータを配置できない場合，そのTLG集合の分割数を可能な限り大きな分割数で分割することとし，以降は内側のTLG集合に着目して同様の処理を繰り返し，Blockサイズを決定する．

4.1.4 分割数決定

まだ分割数が決まっていないALG setが残っている場合，4.1.3と同様にして分割数を決めていく．この時，既にBlockサイズが決まっているため，分割後の配列の最大データサイズがBlockサイズ以下になるようにしなければならない．

4.1.5 マクロタスク分割

4.1.3項および4.1.4で決定された分割数でマクロタスクを分割する．最外側だけでなく内側のTLG集合でも分割を行った場合，ループディストリビューションを行うことで，各最外側ループにおいて，必要なデータを全てローカルメモリ上に配置できるサイズに縮小する．

4.2 配列依存範囲解析

ローカルメモリをもつアーキテクチャにおいて，明示的にデータの転送範囲を指示したり，データ転送量をなるべく削減するためにはマクロタスク間の配列依存範囲解析が重要となる．

配列依存範囲解析はマクロタスク間の配列のフロー依存・入力依存範囲を解析する．その際，マクロタスク内で前方露出参照されるデータに加え，あいまいな定義範囲もマクロタスク開始時点までにローカルメモリに配置することで，書き戻し時のデータの正しさを保証する．

4.3 データローカリティを考慮したスケジューリング

ループ整合分割後，本稿ではスタティックスケジューリングを用い，マクロタスクをPEに静的に割り当てる．ただし，この段階では配列データのローカルメモリ割り当てまでは考慮せず，スケジューリングを行う．ただし，同一のPEに割り当てられたマクロタスク間ではデータ転送は発生せず，異なるPEに割り当てられたマクロタスク間では配列依存範囲解析結果に従い，データ転送が発生すると仮定して行う．実際は同一のPEに割り当てられた場合でも，ローカルメモリ上の異なるBlockに割り当てられるとデータ転送が発生することになる．複数種類のヒューリスティックアルゴリズムを用いてスケジューリングを行い，スケジューリング長が最も短くなるものを採用する．現在OSCARコンパイラではETF/CP/MISF[20)]

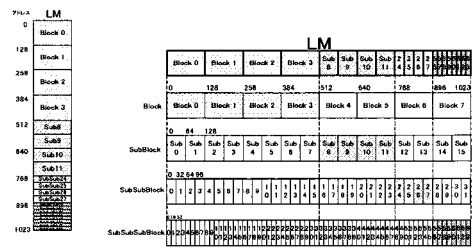

図 4: Block・Sub Block

および CP/ETF/MISF[20] を用いてスケジューリングを行っている.

4.4 ローカルメモリ割り当て

スケジューリング後, OSCAR コンパイラは分割されたデータをローカルメモリに配置する. スケジューリングされた時刻が早いマクロタスクから順に, ローカルメモリ割り当てを決定する. ここで, 想定するローカルメモリとして, 自プロセッサからしかアクセスできないローカルメモリと他プロセッサからもアクセス可能な分散共有メモリの二つを考えるものとする.

フラグメンテーションを可能な限り抑えるために, ローカルメモリ, 分散共有メモリとも Block および Sub Block と呼ぶ固定長の領域に分割して管理する. Block サイズは分割後のマクロタスクの最大アクセス配列範囲を基に決定するので, 分割後のマクロタスク内でアクセスされる各配列は全て 1 Block に収まる. アクセス範囲が小さな配列に関しては, 適切な大きさの Sub Block に配置されるため, ローカルメモリを効率よく使用することができる. 各 Block, Sub Block はメモリマップ上同じ空間に設定し, 随時適切なサイズの Block,Sub block として利用するようにメモリ管理を行うことも可能である. 図 4 がローカルメモリ上に Block,SubBlock を設定したイメージ図であり, ローカルメモリのメモリアドレスと同じ空間に様々な大きさの Block がマッピングされている. 例えば Block0 と SubBlock0,1 と SubSubBlock0,1,2,3 と SubSubSubBlock0〜7 は同じメモリ空間アドレス 0〜127 番地を指している.

各データをどの Block に配置するかを決定する方法について 4.4.1 項で述べる. データを配置することができる Block が無い場合, ローカルメモリ上に配置されているデータを共有メモリに書き戻して Block を使用できるようにする. この書き戻しは 4.4.2 項で述べる掃き出しプライオリティに基づいて行う.

4.4.1 割り当て Block 決定方法

以下の手順により割り当て Block を決定する.

1. 割り当て対象データが自プロセッサからのみアクセスされる場合はローカルメモリ上の Block に, プロセッサ間で共有される場合は分散共有メモリ上の Block に優先的に配置する
2. 割り当て対象データと全く同じ範囲のデータが既にいずれかの Block に割り当てられていれば, その Block をそのまま使用し続ける
3. 空いている Block があればその Block を使用する
4. 掃き出しプライオリティに基づいて Block を掃き出して空の Block を作り, それを使用する

簡単な例を図 5 に示す. 図 5 において, 横方向がアドレス空間上の連続領域を表し, 縦方向が Block を表す.

4.4.2 掃き出しプライオリティ

あるマクロタスクにおいて必要となるデータをローカルメモリに載せようとした時, 空いている Block が足りない場合 Block 内のデータを掃き出してから使用する. 掃き出し先は集中共有メモリもしくは, 分散共有メモリとなる. 出来る限りローカルメモリに使用するデータを載せつつ, また無駄なデータ転送を少なくするためにはローカルメモリ上に残しておきたいデータと, 掃き出してもあまり影響が出ないデータの選別が必要となる. そ

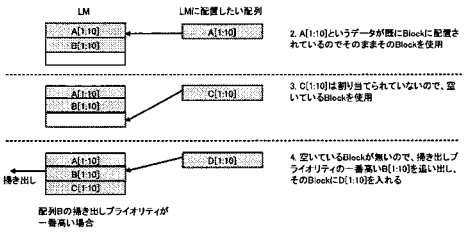

図 5: 割り当てブロック

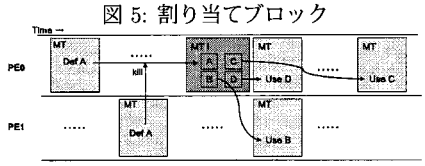

図 6: 掃き出しプライオリティ

こで次のような掃き出しプライオリティを計算し, このプライオリティを基に Block を必要な分だけ掃き出すという処理を行う.

1. 死んでいるデータ (それ以降自プロセッサからも他プロセッサからも一切アクセスされないデータ)
2. 他プロセッサでアクセスされるが, 今後自プロセッサでアクセスされないデータ
3. 再び自プロセッサでアクセスされる
4. すぐに自プロセッサで使用するデータ

図 6 のように各マクロタスクがスケジューリングされ, MT i のメモリ管理を行う際に, データ A,B,C,D がローカルメモリ上の Block に配置されているものとして, データの掃き出しが必要となった場合について考える. この時掃き出しプライオリティを考えると, データ A が条件 1 に, データ B が条件 2 に, データ C が条件 3 に, データ D が条件 4 にそれぞれ該当するため, データ A,B,C,D の順に掃き出しプライオリティが高くなる.

5 性能評価

本章では OSCAR マルチコア上でのローカルメモリ管理の性能評価結果について述べる. 図 3(a) で示したプログラムおよび SPEC 95fp ベンチマークの tomcatv を用いて評価を行った.

5.1 評価環境

本評価はクロックレベルの詳細なシミュレータを用いて行った. 評価プログラムには, 図 3 に示したサンプルコードと SPEC 95fp より tomcatv を用いた. 今回, プロセッサコアの周波数は組込み用途を想定して 400MHz とし, 各メモリのレイテンシを CSM は 24 クロック, LDM は 1 クロック, ローカル DSM は 1 クロック, リモート DSM は 4 クロック, そして LPM は 1 クロックと設定した. LDM および DSM のサイズについては後述する. チップ内のメモリレイテンシの算出には ITRS 2003[21] および CACTI[22] を, チップ外のレイテンシには Elpida Memory 社のデータシート [23),24)] をそれぞれ用いた. また DTU のバースト幅は 64byte とした.

5.2 サンプルプログラムにおける評価

本節では, 図 3(a) で示したプログラムに対する本手法の性能評価について述べる.

このプログラムで扱う総配列データサイズは 256kB である. 従って, ローカルメモリ上の管理対象領域が 16kB 以上であれば, 最外側ループのみを分割することでデータワーキングセット全てをローカルメモリ上に配置できるが, 16kB よりも小さな領域しかない場合はネストされた内側ループも分割を行う必要がある.

本評価では, 逐次処理において本手法適用時と従来手法適用時の比較を行った. LDM サイズを変化させて, 本手

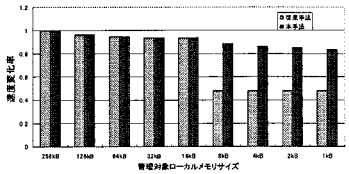

図 7: サンプルプログラムにおける性能評価結果

法適用時と従来手法適用時の性能を評価した．なお，DSM サイズは 1kB とした．

図 7 に評価結果を示す．横軸が管理対象とする LDM のサイズ，縦軸が管理対象の LDM サイズが 256kB の時の従来手法適用時の性能に対する速度変化率を表す．

管理対象の LDM サイズが 16kB までは本手法，従来手法ともに同じ性能であるが，管理対象の LDM サイズが 8kB 以下になると，従来手法ではデータワーキングセットを管理対象の LDM サイズ以下にすることができず，全てのデータが CSM に配置され，CPU で直接それらのデータをロード，ストアして実行するため，約 52.0%という大きな性能低下を招く．

これに対して本手法では，管理対象の LDM サイズが 8kB 以下になった場合はネストされたループに対して分割を行うことによって，データワーキングセット全てを LDM 上に配置することができ，管理対象の LDM サイズが 1kB の場合においても約 16.7%の性能低下に抑えることができた．

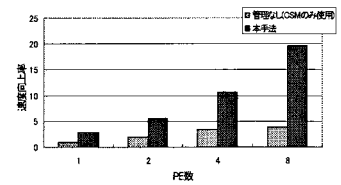

図 8: tomcatv における性能評価結果

5.3 tomcatv における評価

本節では，SPEC 95fp ベンチマークの tomcatv に対する本手法の性能評価を述べる．tomcatv は，Vectorized Mesh 生成プログラムであり，入出力部分と収束計算ループから構成されている．本評価では収束計算部における実行時間のみを評価対象とした．また，本手法ではスタティックスケジューリングを行うため，ループ回転数を手動で定数化したプログラムに対して評価を行った．総データサイズは約 14MB であり，LDM サイズは 4MB，DSM サイズは 256kB とした．

本評価では，本手法適用時の実行時間と，未適用時の実行時間の比較を行った．

図 8 に評価結果を示す．横軸がプロセッサ数，縦軸が本手法未適用時の逐次処理性能に対する速度向上率を表す本手法の適用により，8 プロセッサにおいて，本手法未適用時の逐次処理に対して約 19.6 倍の性能向上が得られた．

6 まとめ

本稿では，OSCAR コンパイラを用いた粗粒度タスク並列処理におけるローカルメモリ管理手法について述べた．グローバルループ整合分割手法，データ転送命令自動生成およびローカルメモリ管理手法を OSCAR マルチグレイン並列化コンパイラ上に実装し，OSCAR マルチコア上で性能評価を行った結果，256kB のデータを扱う簡単なサンプルプログラムにおいて，256kB のローカルメモリ上の領域を用いてデータ管理を行った場合と比較して，1kB のローカルメモリ上の領域を用いてデータ管理を行った場合，約 16.7%の性能低下に抑えることができた．また，SPEC95fp tomcatv において，本手法未適用時の逐次処理に対し，8 プロセッサにおいて約 19.6 倍の性能向上が得られ，本手法の有効性が示された．

7 謝辞

本研究の一部は NEDO“リアルタイム情報家電用マルチコア技術”，の支援により行われた．

参考文献

[1] Suga, A. and Matsunami, K.: Introducing the FR 500 embedded microprocessor, *IEEE MICRO*, Vol. 20, pp. 21–27 (2000).

[2] ARM: *ARM11 MPCore Processor Technical Reference Manual* (2005).

[3] Pham, D., Asano, S. and et al., M. B.: The Design and Implementation of a First-Generation CELL Processor (2005).

[4] Sinharoy, B., Kalla, R. N., Tendler, J. M., Eickemeyer, R. J. and Joyner, J. B.: POWER5 system microarchitecture, *IBM journal of research and development*, Vol. 49 (2005).

[5] Kongetira, P., Aingaran, K. and Olukotun, K.: Niagara: a 32-way multithreaded Sparc processor, *IEEE MICRO*, Vol. 25, pp. 21–29 (2005).

[6] Kasahara, H. and Yoshida, A.: A Data-Localization Compilation Scheme Using Partial Staticc Task Assignment for Fortran Coarse Grain Parallel Processing, *Journal of Parallel Computing*, Vol. Special Issue on Languages and Compilers for Parallel Computers (1998).

[7] Avissar, O., Barua, R. and Stewart, D.: An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems , *ACM Transactions on Embedded Computing Systems*, Vol. 1, No. 1, pp. 6–26 (2002).

[8] Panda, P. R., Dutt, N. and Nicolau, A.: *Memory issues in embedded systems-on-chip*, Kluwer Academic Publishers (1999).

[9] Verma, M., Wehmeyer, L. and Marwedel, P.: Dynamic Overlay of Scratchpad Memory for Energy Minimization, *Proc. of Intl. Symposium on System Synthesis* (2004).

[10] Li, L., Gao, L. and Xue, J.: Memory coloring: a compiler approach for automatic scratchpad memory management, *PACT'05* (2005).

[11] Kandemir, M., Ramanujam, J., Irwin, M. J., Vijaykrishnan, N., Kadayif, I. and Parikh, A.: A compiler based approach for dynamically managing scratch-pad memories in embedded systems, *IEEE Trans. on CAD*, Vol. 23, No. 2, pp. 243–260 (2004).

[12] 笠原博徳: 最先端の自動並列化コンパイラ技術, 情報処理, Vol. 44, No. 4, pp. 384–392.

[13] 白子, 長澤, 石坂, 小幡, 笠原: マルチグレイン並列性向上のための選択的インライン展開手法, 情報処理学会論文誌, Vol.45 No. 5, pp.1345-1356, Vol. 45, No. 5, pp. 1345–1356.

[14] 本多弘樹, 岩田雅彦, 笠原博徳: Fortran プログラム粗粒度タスク間の並列性検出手法, 電子情報通信学会論文誌, Vol. J73-D-1, No. 12, pp. 951–960 (1990).

[15] 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).

[16] 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).

[17] 八木, 板垣, 中野, 石坂, 小幡, 吉田, 笠原: 共有メモリマルチプロセッサ上でのデータローカライゼーション対象マクロタスク決定手法, 情報処理学会研究報告 ARC (2002).

[18] 吉田, 前田, 尾形, 笠原: Fortran マクロデータフロー処理における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. 35, No. 9, pp. 1848–1860 (1994).

[19] 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).

[20] 笠原: 並列処理技術, コロナ社 (1991).

[21] : International Technology Roadmap for Semiconductors 2003 Executive Summary (2003).

[22] Wilton, S. and Jouppi, N.: CACTI: An enhanced cache access and cycle time model, *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 5, pp. 677–688 (1996).

[23] ELPIDA MEMORY, INC.: *PRELIMINARY DATA SHEET 512bits DDR SDRAM EDD 5104 ABTA, EDD 5108 ABTA* (2003).

[24] ELPIDA MEMORY, INC.: *PRELIMINARY DATA SHEET 256bits DDR2 SDRAM EDE 2504 AASE, EDE 2508 AASE, E DE 2516 AASE* (2003).