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

2006－ARC－169 (28)
2006／8／2

OSCARマルチコア上でのローカルメモリ管理手法

中野啓史† 仁藤拓実‡ 丸山貴紀§
中川正洋† 鈴木裕貴¶ 内藤陽介‖
宮本孝道† 和田康孝†
木村啓二† 笠原博徳†

半導体集積度向上に伴う消費電力の増大，プロセッサ実質速度向上の鈍化，ハードウェア，ソフトウェア開発期間の増大といった問題を解決すべく，一つのチップ上に複数のプロセッサコアを集積するマルチコアプロセッサが次世代プロセッサアーキテクチャとして注目を集めている．この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伴うメモリウォールは深刻な問題であり，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重要なポイントとなっている．このような事項を考慮して筆者等は自動マルチグレイン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良いコンピュータシステムの実現を目指すOSCARマルチコアプロセッサを提案している．このOSCARマルチコアプロセッサは，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)，そしてデータ転送オーバヘッドの隠蔽を目指し，プロセッサコアと非同期に動作可能なデータ転送ユニット (DTU) を持つ．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LDM/DSM管理手法について述べる．性能評価の結果，逐次実行に比べ8PE時，MP3エンコーダで約7.1倍，MPEG2エンコーダで約6.3倍，JPEG2000エンコーダで約3.8倍の速度向上が得られた．

Local Memory Management on OSCAR Multicore

HIROFUMI NAKANO† , TAKUMI NITO‡ , TAKANORI MARUYAMA§ ,
MASAHIRO NAKAGAWA† , YUKI SUZUKI¶ , YOSUKE NAITO‖ ,
TAKAMICHI MIYAMOTO† , YASUTAKA WADA† , KEIJI KIMURA†
and HIRONORI KASAHARA†

Along with the advancement of integration technology of semiconductor devices, to overcome the increase of power consumption, the slowdown of processor effective performance improvement rate, and the increase of period for hardware/software developing transistors integrated on to a chip, multicore processors have attracted much attention as a next-generation microprocessor architecture. However, the memory wall caused by the gap between memory access speed and processor core speed is getting a serious problem also on the multicore processors. Therefore, the effective use of fast memories like cache and local memory nearby a processor is important. Considering these problems, the authors have proposed the OSCAR multicore processor architecture which cooperates with OSCAR multigrain parallelizing compiler and aims at developing a processor with high effective performance and good cost performance. The OSCAR multicore processor has local data memory (LDM) for processor private data, distributed shared memory (DSM) having two ports for synchronization and data transfer among processor cores, centralized shared memory (CSM) to support dynamic task scheduling, and data transfer unit (DTU) which transfers data asynchronously and aims at overlapping data transfer overhead. This paper describes data localization scheme that aimed at improving the effective use of LDM and DSM using coarse grain task parallel processing and compiler-controlled LDM and DSM management scheme. As the results, the proposed scheme gives us 7.1 times speedup for MP3 encoding program, 6.3 for MPEG2 encoding program and 3.8 for JPEG2000 encoding program against the sequential execution without the proposed scheme on 8 processors automatically.

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は重

†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
‡株式会社 日立製作所
§松下電器産業株式会社
¶株式会社 NTT データ
‖株式会社野村総合研究所

要であり，キャッシュやローカルメモリ等のチップ内の近接メモリの有効利用を行う必要がある．

筆者等は自動マルチグレイン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良いコンピュータシステムの実現を目指す OSCAR マルチコアプロセッサを提案している [6]．この OSCAR マルチコアプロセッサは，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)，そしてデータ転送オーバヘッドの隠蔽を目指し，プロセッサコアと非同期に動作可能なデータ転送ユニット (DTU) を持つ．

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，我々はコンパイラによるデータローカライゼーション手法 [7] を用い，プログラムのデータローカリティの抽出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た．

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 [8],[9] 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 [7],[10],[11] が提案されている．一方でこれらの手法はローカルメモリサイズ以上の大きさのデータのローカルメモリへの配置に関する問題について考慮していない．

一般的なアプリケーションにおいて，アクセスされるデータサイズがすべてローカルメモリサイズ以下となることは考えづらい．そこで，ループを変換し，その後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領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 [12] が提案されている．

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，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 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．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，実行時のある瞬間において，アクセスされるデータ領をローカルメモリサイズ以下に抑えることが可能になり，どのようにスケジューリングを行ってもローカルメモリへの割り当てが可能になる．分割された MT 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および並列性の抽出を行う．その後，MT 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，配列の依存範囲情報に基づき，データ転送を挿入する．

本稿の構成を以下に示す．第 2 章では我々が提案する 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 MP3 エンコーダ，MPEG2 エンコーダそして JPEG2000 エンコーダを用いて行う．第 6 章で本稿のまとめを述べる．

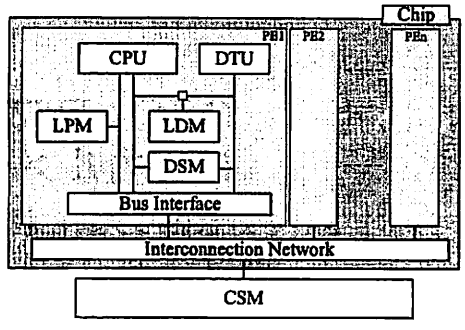

図 1: OSCAR マルチコアアーキテクチャ

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自動マルチグレイン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，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に分割する．

次に 4 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．

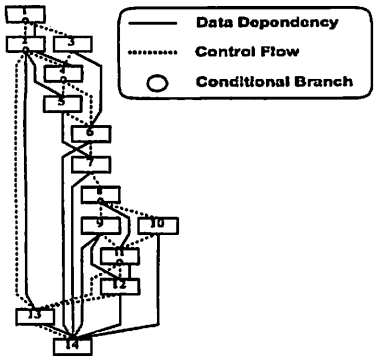

図 2: マクロフローグラフの例

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．

3.2 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 2 に示すようなマクロフローグラフ (MFG) を生成する．

図 2 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．

3.3 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 3 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．

3.4 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．

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．

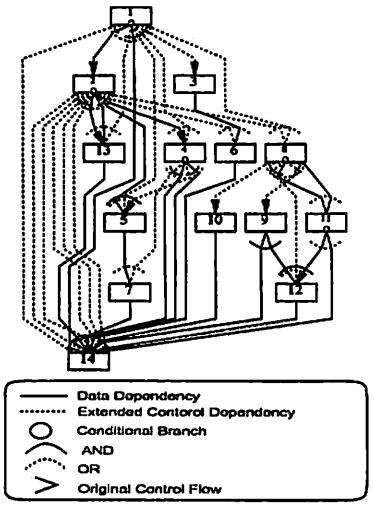

図 3: マクロタスクグラフの例

4 ローカルメモリ管理

近年，プロセッサ速度とメモリアクセス速度の差が増大し，メモリウォールがますます深刻となっている．これを解決するために時間的局所性，空間的局所性を有効に利用し，プロセッサ近傍の高速なキャッシュメモリやローカルメモリ上に一度ロードしたデータを複数のタスク間で長期間に渡り利用する技術がデータローカライゼーションである．

従来提案してきたデータローカライゼーション手法では，異なるプロシージャを含むプログラム全域から大量の配列データを共有するループを選択し，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)[7),13),14)] 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なるべく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，ローカルメモリをも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，すべての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．そのために，従来対象としていなかった MT についてもローカルメモリに載るように分割を行い，すべての MT でアクセスされる配列データをローカルメモリへ割り当てなければならない．そこで，すべての MT を対象とするグローバルループ整合分割を行う．本手法では，フラグメンテーションを避けるために，ローカルメモリはすべて同一サイズのスロットと呼ばれる固定長ブロックに分割し管理している．グローバル整合分割において，すべての MT がローカルメモリに載るようにスロットサイズと分割数を決定する．分割された各 MT 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，なるべくデータ転送が発生しないように，MT 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．この段階ではあくまでも MT の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 MT 間の配列依存範囲解析結果を基に MT 内でアクセスされる配列データのローカルメモリへの割り当てを行う．データローカライゼーションによって残存したデータ転送は，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 によって効率的に転送される．

4.1 ループ整合分割

プログラムの広域に渡り，同一の配列にアクセスし，データ依存関係にあるループを探す．その際，ループ同

士が異なるプロシージャに存在した場合は，必要に応じインライン展開を行い，それらのループ同士を同一の階層に持ってくると共に，最早実行可能条件解析に基づくコードモーションにより，集められたループ群をターゲットループグループ (Target Loop Group: TLG) と呼ぶ．

TLG 中のループについて，Inter Loop Dependence(ILD) を解析する．TLG 中で処理コストのもっとも大きな MT 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 である．

従来のループ整合分割が適用されなかった MT もローカルメモリサイズ以下に分割することで，どのようにスケジューリングを行っても，ある瞬間に実行中の MT でアクセスされる配列はすべてローカルメモリに配置することができ，ローカルメモリをもつアーキテクチャにおいてもプロセッサ近傍の高速なメモリをプログラム全域に渡って，有効利用することができ，プログラムの効率的な実行が可能となる．

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 MT 間の配列依存範囲解析が重要となる．

本節で述べる配列依存範囲解析はローカルメモリをもつアーキテクチャを想定しており，MT 内で前方露出参照されるデータに加え，定義され得る範囲も MT 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ものである．

ここで，配列範囲は配列のアクセス範囲の上限値と下限値で表され，配列範囲間の演算は，差を '$-$'，積を '$\cap$'，和を '$\cup$' でそれぞれ表すものとする．

まず，MT 毎に確実に定義される配列範囲を Kill，定義される可能性のある配列範囲を MayMod，参照される可能性のある配列範囲を MayUse，前方露出される配列範囲を ExposedUse としてそれぞれ解析する．

次に以下の手順に従い，MT 間の配列依存範囲解析を行う．

1. MT_i の配列依存範囲解析を行う．MT_i について，以下のように定義された $Consumption_i$，$Production_i$ を求める．

$$\begin{aligned} Consumption_i &= In_i \cap \left(ExposedUse_i \right. \\ &\quad \left. \cup (MayMod_i - Kill_i)\right) \\ Production_i &= Out_i \cap MayMod_i \end{aligned}$$

   ここで，In_i は MT_i に生きて入る配列，Out_i は MT_i から生きて出る配列を表す．

2. $Production_i$ を $valid_i$ とする．
3. MT_i の各後続タスク MT_j に対し，コントロールフロー順に $valid_i$ と $Consumption_j$ を比較し，重なる配列範囲を MT_i から MT_j への配列フロー依存範囲 $FlowDep_{i-j}$ とする．$valid_i$ から MT_j の $Production_j$ と重なる範囲を引き，新たな $valid_i$ とする．

$$FlowDep_{i-j} = valid_i \cap Consumption_j$$

4. $valid_i$ の範囲が空になるか階層の最後のマクロタスクまで到達したら，解析を終了する．
5. MT_i から MT_j，MT_k に対してフロー依存 $FlowDep_{i-j}$，$FlowDep_{i-k}$ が得られた場合，$FlowDep_{i-j}$ と $FlowDep_{i-k}$ の重なる配列範囲を MT_j から MT_k に対する入力依存範囲 $InputDep_{j-k}$ とする．

$$InputDep_{j-k} = FlowDep_{i-j} \cap FlowDep_{i-k}$$

6. 解析対象の階層の外側ブロックが繰り返しブロックであるとき，階層の最後まで到達した $valid_i$ のうち，次のイタレーションで参照される範囲と重なる部分はイタレーション間でのデータ依存範囲であり，これを $LoopCarriedDep_i$ とする．

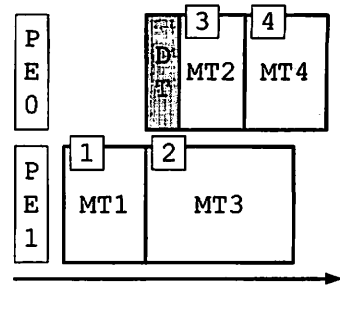

図 4: スケジューリング結果とローカルメモリ割り当て順番

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，MT 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 MT 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 MT 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スロットに割り当てられるとデータ転送が発生することになる．ここで，評価するスタティックスケジューラのプライオリティはヒューリスティックアルゴリズムである ETF/CP/MISF[15] をベースとし，第二プライオリティとして MT 間のデータ共有量を追加した．

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配列データをローカルメモリに配置する．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 LDM と他プロセッサからもアクセス可能な DSM の二つを考えるものとする．このとき，フラグメンテーションを可能な限り抑えるために，LDM/DSM とも同一サイズのスロットと呼ぶ固定長の領域に分割して管理する．スロットサイズは分割後の MT の最大アクセス配列範囲として決定するので，分割後の MT 内でアクセスされる各配列はすべて 1 スロットに収まる．アクセス範囲がスロットサイズよりも小さな配列に関しては，領域節約のために，同一のスロットに複数配置される．スケジューリングされた時刻が早い MT から順に，ローカルメモリ割り当てを決定する．たとえば，図 4 に示すようにスケジューリングされていたら，MT1, MT3, MT2, MT4 の順でローカルメモリマッピングを決定する．

今ローカルメモリへの割り当てを考えている MT を MT_j，MT_j の生産者 MT を MT_i とする．MT_i と MT_j 間の配列 A に関するフロー依存範囲を $FlowDep_{i-j}(A)$ とする．MT_i が割り当てられたプロセッサを PE_p，MT_j が割り当てられたプロセッサを PE_q とそれぞれする．MT_i の配列 A が割り当てられているスロットを $Slot_m$，MT_j の配列 A の割り当てを考えているスロットを $Slot_n$ とそれぞれする．PE_q に MT_j よりも先に割り当てられていて，配列 A が $Slot_n$ に割り当てられている MT を MT_k とする．このとき，MT_k と MT_j 間の配列 A に関する入力依存範囲を $InputDep_{k-j}(A)$ とする．各 MT と依存範囲の関係を図 5 に示す．なお，本来入力依存はマクロタスクグラフ中には現れないが説明上図示する．

以上の仮定をもとに，MT_j の配列 A を $Slot_n$ に割り当てたときのデータ転送削減量 $Gain_{An}$ は以下のように求まる．

PE_p と PE_q が等しくかつ $Slot_m$ と $Slot_n$ が等しい場合，$FlowDep_{i-j}(A)$ はデータ転送する必要がなく，データ転送削減量に削減されるデータ転送時間を加算し，

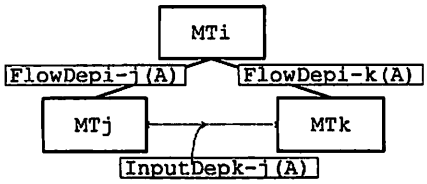

図 5: 配列依存範囲を記述したマクロタスクグラフ

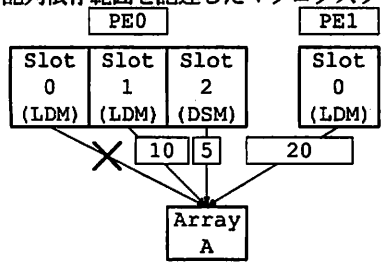

図 6: Slot 割り当てプライオリティ

$$Gain_{An} \quad += \quad (FlowDep_{i-j}(A) \times (Slot_m\text{から } Slot_n\text{へのストアレイテンシ})$$

となる.

次に生産者と消費者で割り当てられたスロットが異なった場合，スロット間で必要となるデータ転送範囲 $Range$ は

$$Range \quad = \quad FlowDep_{i-j}(A) - \cup InputDep_{k-j}(A)$$

となる.

PE_p と PE_q が等しく，$Slot_m$ と $Slot_n$ が異なる場合，プロセッサ内部ではメモリ種別に関わらず，スロット間の直接転送が可能なため，データ転送削減量からデータ転送時間を引き，

$$Gain_{An} \quad -= \quad Range \times (Slot_m\text{から } Slot_n\text{へのストアレイテンシ})$$

となる.

PE_p と PE_q が異なる場合，$Slot_n$ のメモリ種別によって，データ転送時間が異なる．$Slot_n$ が DSM 上のスロットで，直接転送が可能な場合，そのデータ転送時間をデータ転送削減量から引き，

$$Gain_{An} \quad -= \quad Range \times (\text{リモート } DSM \text{ ストアレイテンシ})$$

となる.

直接転送できない場合は，CSM を介した転送となり，そのデータ転送時間をデータ転送削減量から引き，

$$Gain_{An} \quad -= \quad Range \times (CSM \text{ ストアレイテンシ} + CSM \text{ ロードレイテンシ})$$

となる.

配列とスロットのすべての組み合わせについてデータ転送削減量を計算し，削減量が大きなものから順に割り当てる．どのスロットにも割り当てきれない配列が存在したら，スケジューリング結果と配列依存範囲解析結果をもとに最も利用頻度の低いスロットを掃き出す．各 MT 毎にすべての配列がスロットに割り当てられるまで繰り返す．このように割り当てスロットの決定，掃き出しスロットの決定を行うことで，限られたローカルメモリに効率よく配列データを割り当てることが可能となる．図 6 を使って Array A の割り当ての様子を説明する．各スロットの状態は以下の通りとする．PE0 の Slot 0 は容量不足で割り当て不可，Slot 1，Slot 2 からは Array A に対してそれぞれ 10, 5 のデータ量を持つフロー依存範囲があるものとする．また，PE1 の Slot 0 から 20 のデータ量を持つフロー依存範囲があるものとする．このとき，Array A の Slot 1, Slot 2 への割り当てプライオリティ $Gain_{A1}$, $Gain_{A2}$ は，それぞれ

$$Gain_{A1} = 10 \times (LDM \text{ ストアレイテンシ}) - 5 \times (LDM \text{ ストアレイテンシ}) - 20 \times (CSM \text{ ストアレイテンシ} + CSM \text{ ロードレイテンシ})$$

$$Gain_{A2} = 5 \times DSM \text{ ストアレイテンシ} - 10 \times DSM \text{ ストアレイテンシ} - 20 \times \text{リモート } DSM \text{ ストアレイテンシ}$$

となる．一般にリモート DSM レイテンシが CSM レイテンシよりも小さいので，$Gain_{A1} < Gain_{A2}$ となり，Array A は Slot 2 に割り当てられることになる．このように自プロセッサ内でのデータ共有量と他プロセッサからのデータ受信量を考慮して，配列を Slot へと割り当てていくことで効率のよい割り当てを実現している.

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および DTU によるバースト転送の性能評価結果について述べる.

5.1 評価環境

MP3 エンコーダ，MPEG2 エンコーダ，JPEG2000 エンコーダの 3 つのメディアアプリケーションを用いて，本手法の性能評価を行った．MP3 エンコーダは UZURA3: MPEG1/LayerIII Encoder in FORTRAN90[16] を FORTRAN で参照実装したプログラムを用いた．入力の PCM データが全て CSM 上に格納されている状態からエンコード後のデータが CSM に書き込まれるまでを評価の対象とした．入力データはサンプリングレート 44.1kHz のステレオの PCM データ 32 フレーム，出力データのビットレートは 128kbps とし，圧縮方式は不可逆変換とし，その他のエンコードオプションはすべて参照した UZURA のデフォルトパラメータと同一とした.

MPEG エンコーダは MediaBench[17] に収録されている MPEG2 エンコードプログラムである “mpeg2encode” を FORTRAN で参照実装したプログラムを用いた．本性能評価では MPEG2 エンコーディング処理自体の性能評価を行うため，その処理を行う 7 つのステージ (動き推定，動き予測，DCT モード選択，データ変換，ビットストリーム出力，逆量子化，逆データ変換) を性能評価の対象とした．シミュレーション時間短縮のために，入力画像は MediaBench で用いられる入力画像を 256x256 ピクセルに縮小した画像を 4 フレーム分用い，エンコードを行った．それ以外のエンコードオプションは MediaBench のデフォルトパラメータと同一とした.

JPEG2000 エンコーダは JJ2000[18] を FORTRAN で参照実装したプログラムを用いた．入力画像は 800×600 ピクセル (SVGA) の画像を用いた．入力のデータが CSM 上に読み込まれてから，エンコード結果が CSM に書き込まれるまでを評価の対象とした．Wavelet レベルを 3，圧縮方式は不可逆変換とし，その他のパラメータはすべて参照した JJ2000 のデフォルトパラメータと同一とした.

LDM のうち 200kB を，DSM のうち 20kB をそれぞれ本手法により管理するものとした．また，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．評価した PE 数は 1, 2, 4 そして 8，また DTU のバース ト幅は 64byte とした．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．同一関数を複数個所から異なるデータサイズの引数を渡して実行する際の，高精度タスクコスト推定ルーチンが実装されていないため，JPEG2000 に関しては，実行プロファイルから得られるタスクコストを用いて，タスクスケジューリングを行っている.

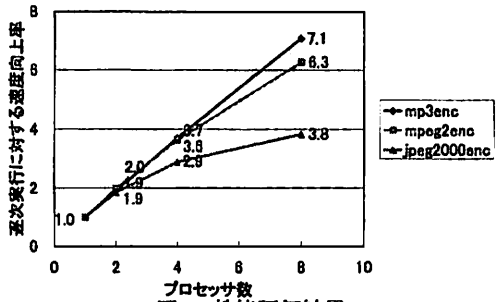

図 7: 性能評価結果

5.2 性能評価結果

MP3 エンコーダ，MPEG2 エンコーダ，JPEG2000 エンコーダの性能評価結果を図 7 に示す．図中の横軸はプロセッサ数を，縦軸は逐次実行に対する速度向上率をそれぞれ表す．図中の mp3enc は MP3 エンコーダの，mpeg2enc は MPEG2 エンコーダの，jpeg2000enc は JPEG2000 エンコーダの評価結果をそれぞれ表す．まず，本手法の適用により，8 プロセッサ時，本手法を適用した逐次実行と比べ，MP3 エンコーダで約 7.1 倍，MPEG2 エンコーダで約 6.3 倍，JPEG2000 エンコーダで約 3.8 倍の速度向上がそれぞれ得られた．

次に，本手法適用時，未適用時の比較を行う．本手法を適用しない場合，全てのデータは CSM に配置され，CPU で直接それらのデータをロード，ストアして実行を行う．本手法適用時は未適用時に比べ，逐次実行において，MP3 エンコーダで約 2.0 倍，MPEG2 エンコーダで約 3.1 倍，JPEG2000 エンコーダで約 4.1 倍の速度向上がそれぞれ得られた．MP3 エンコーダ，MPEG2 エンコーダ，JPEG2000 エンコーダの各アプリケーション毎に速度向上率に差があるのは，本手法未適用時，実行時間に占める CSM アクセス時間がそれぞれ 51%,70%,78%と差があり，それが本手法を適用すると 1%,5%,23%にそれぞれ低下しているためだと考えられる．また，本手法適用時の CSM アクセス時間自体を比較すると，本手法未適用時に比べ，それぞれ 1%,2%,9%に削減されている．これは本手法未適用時 CPU により，直接 CSM にアクセスしていたのが，本手法適用により，DTU を使って，効率的に転送されるようになったためだと考えられる．

次に各アプリケーション毎に結果を見てみる．MP3 エンコーダにおいて，本ローカルメモリ最適化手法は未適用時の逐次実行 (CSM に全データを配置した場合) に対し，4 プロセッサで約 7.5 倍，8 プロセッサで約 14.3 倍とスケーラブルな速度向上が得られた．MP3 エンコーダでは，フレーム単位での並列性を利用し，各ステージで処理されるデータ量がローカルメモリサイズ以下になるように MT を分割し，一度ローカルメモリに配置したフレームデータをエンコーディングステージ間でそのまま受け渡すことで，効率的な並列処理とローカルメモリ管理を実現した．また，前処理といったエンコーディングステージ以外の処理でアクセスされるデータもローカルメモリに割り当てることで，CPU に近接した高速なローカルメモリの有効利用を図り，効率化を行った．

MPEG2 エンコーダにおいて，本手法は未適用時の逐次実行に対し，4 プロセッサで約 11.2 倍，8 プロセッサで 19.3 倍とスケーラブルな速度向上が得られた．MPEG2 エンコーダでは，マクロブロック間の並列性を利用し，各ステージで処理されるマクロブロックデータがローカルメモリに載るサイズ以下になるように分割し，ローカルメモリに配置することで，効率的な並列処理とローカルメモリ管理を実現した．

JPEG2000 エンコーダにおいて，本手法は未適用時の逐次実行に対し，4 プロセッサで約 11.9 倍，8 プロセッサで約 15.7 倍の速度向上が得られた．JPEG2000 エンコーダでは，前 2 つのアプリケーションと異なり，画像データに対し，まず Wavelet 変換部で，縦横異なる方向に処理を行う．次に画像をコードブロック単位で分け，処理を行う．このように処理されるデータの方向やサイズが異なっていても，本手法は並列性とデータサイズを考慮しながら，分割，スケジューリング，ローカルメモリ割り当てを行うことで，効率的な実行を実現している．

6 まとめ

本稿では，OSCAR マルチコア上での OSCAR コンパイラを用いた粗粒度タスク並列処理における LDM/DSM 管理手法について述べた．LDM/DSM 管理手法，DTU 制御命令自動生成および LDM/DSM 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，逐次実行に対し，8PE 時，MP3 エンコーダで約 7.1 倍，MPEG2 エンコーダで約 6.3 倍，JPEG2000 エンコーダで約 3.8 倍の速度向上が得られ，本手法の有効性が示された．

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”，STARC“並列化コンパイラ協調型チップマルチプロセッサ技術” 及び NEDO“先進ヘテロジニアスマルチプロセッサ技術” によって行われた．

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] Kimura, K., Wada, Y., Nakano, H., Kodaka, T., Shirako, J., Ishizaka, K. and Kasahara, H.: Multigrain Parallel Processing on Compiler Cooperative Chip Multiprocessor, *Proc. of 9th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9)* (2005).

[7] 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).

[8] 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).

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

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

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

[12] 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).

[13] 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).

[14] 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).

[15] 笠原: 並列処理技術, コロナ社 (1991).

[16] : UZURA3:MPEG1/LayerIII Encoder in FORTRAN90 (2002). http://members.at.infoseek.co.jp/kitaurawa/cgi-bin/wiki.cgi.

[17] Lee, C., Potkonjak, M. and Mangione-Smith, W. H.: MediaBench: a tool for evaluating and synthesizing multimedia andcommunications systems, *In 30th Annual IEEE/ACM International Symposium on Microarchitecture* (1997).

[18] : JJ2000 A JAVA™implementation of JPEG 2000. http://jj2000.epfl.ch/.