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

2005－ARC－165（10）
2005／12／1

マルチコアプロセッサ上でのデータローカライゼーション

中野啓史[†]　浅野尚一郎[†]　内藤陽介[†]
仁藤拓実[†]　田川友博[†]　宮本孝道[†]
小高剛[‡]　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データローカライゼーション手法と LDM 管理手法について述べる．提案手法を MPEG2 エンコーダに適用して評価を行った結果，逐次実行に比べ，8PE で約 8.01 倍の速度向上率が得られた．

Data Localization on a Multicore Processor

Hirofumi Nakano[†], Shoichiro Asano[†], Yosuke Naito[†], Takumi Nito[†], Tomohiro Tagawa[†], Takamichi Miyamoto[†], Takeshi Kodaka[‡], Keiji Kimura[†] and Hironori Kasahara[†]

Along with the increase of integration degree of semiconductor devices, to overcome the increase of power consumption, the slowdown of improvement of processor effective performance, and the increase of period for hardware/software developing transistors integrated on to a chip, multicore processors, which integrate multiple processor cores on a single chip, 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 still 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 computer system. 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 using coarse grain parallel processing and compiler-controlled LDM management scheme. As the results, the proposed scheme gives us 8.01 times speedup for MPEG2 encoding program against the sequential execution 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，消費電力の増大等の顕在化に伴い，今後の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よりも粗い粒度の並列性が利用可能となっている．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

[†]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　東芝
[‡]TOSHIBA Corporation

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 [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 [9),10)] 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．

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，分割されたループ中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抑えることが可能になる．分割されたループ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が可能となる．分割されたループ中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付けを行い，配列の生死解析情報に基づき，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割り付け手法について述べる．第 5 章では本手法の性能評価を MPEG2 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．

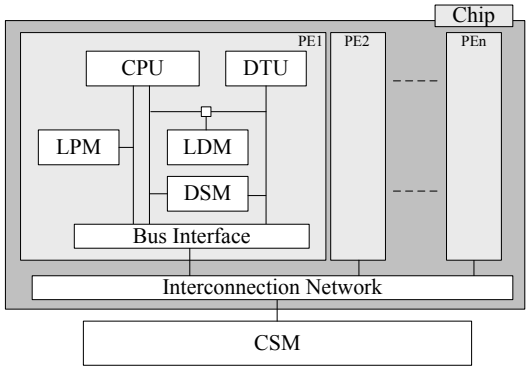

図 1: OSCAR Chip Multiprocessor アーキテクチャ

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．さらに本稿では集中共有メモリがチップ外に接続されている．

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 制御用命令が使用する基本的なパラメータは転送コマンド，転送元アドレス，転送先アドレス，転送領域サイズ，転送終了通知フラグアドレスの 5 つである．これらを指定することで一つの連続領域の転送を行うことが可能となる．さらに転送元ストライド長，転送先ストライド長，転送回数を設定することで，転送元，転送先でストライド長が異なるストライド転送や，SCATTER, GATHER 転送を実現している．これらのパラメータはデータ転送範囲と配列の宣言サイズからコンパイラが自動的に生成する．

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上述の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．上述のパラメータで設定した転送終了通知フラグアドレスに DTU がフラグを立て，転送先の CPU がこのフラグをビジーウェイトしてチェックすることで，割り込み処理を使わずに転送の終了を通知することが可能となっている．

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

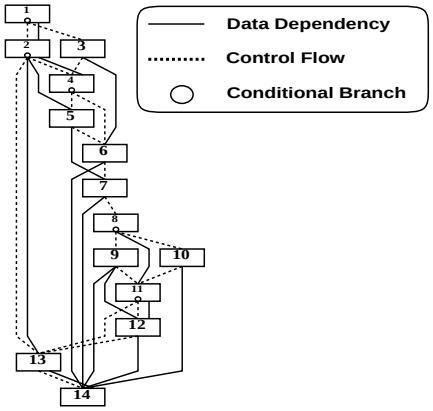

図 2: マクロフローグラフの例

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．

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．

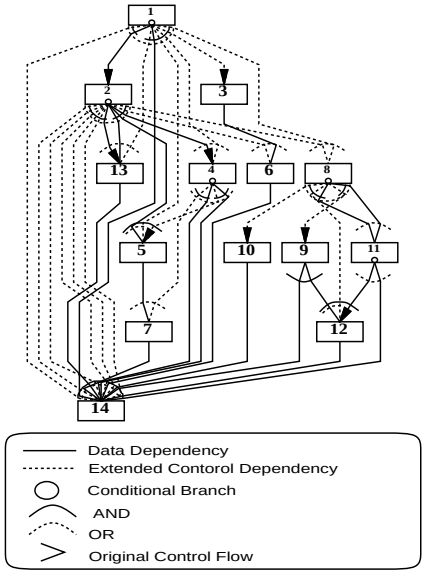

図 3: マクロタスクグラフの例

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．

4 データローカライゼーション

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．

OSCAR コンパイラ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，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)[12),13)] を適用し，並列性を考慮しながら，一度近接メモリ上に配置したデータをなるべく連続的にアクセスするようにタスクをスタティックスケジューリングし，データローカライゼーション [12),14)] を行う．データローカライゼーションによって削減できなかった MT 間に残存するデータ転送は，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 CPU によるタスク実行と並列して，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

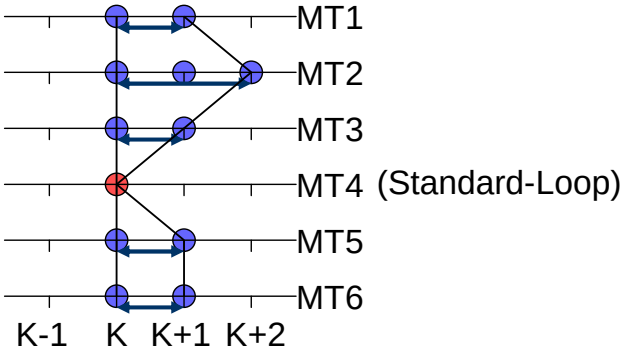

図 4: Inter Loop Dependence 解析の結果

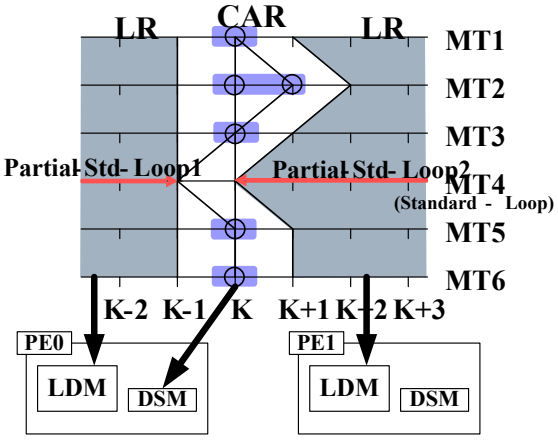

図 5: ループ整合分割例およびメモリへの割り当て

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．ILD の様子を図 4 に示す．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．たとえば，図 4 において，MT4 が標準ループとして選択されている．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．

ILD の結果，OSCAR コンパイラは標準ループを N 個の部分ループに分割する．次に複数の部分ループにアクセスされるイタレーションを Commonly Accessed Region(CAR) として定義する．単一のプロセッサからアクセスされるイタレーションを Localizable Region(LR) として定義する．一つの LR 中でアクセスされる配列サイズの最大値が LDM 上のローカル配列用の領域サイズ以下となるように分割数 N を決定する．この分割の様子を図 5 に示す．また，分割後の各部分ループは図 5 に示した通り，LR は PE 上の LDM に CAR は DSM にそれぞれ割り当てられる．

ループ整合分割後，同一の分割された配列範囲を共有するループは Data Localizable Group(DLG) としてグループ化される．同一の DLG 中に含まれる全ての MT は共有データを LDM を介して授受するために，同一のプロセッサに連続的に割り当てられる．

ループ整合分割前の MTG の例と各 MT の配列解析の結果を図 6 に示す．ループ整合分割後の MTG の例を図 7 に示す．図 6 中の MT1, MT2, MT3 は図 7 中の MT1_1, MT2_1 そして MT3_1，MT1_2, MT2_2 そして MT3_2, MT1_3, MT2_3 そして MT3_3, MT1_4, MT2_4 そして MT3_4 にそれぞれ 4 つに分割されている．また，MT1_1, MT2_1 そして MT3_1 は DLG1 に，MT1_2, MT2_2 そして MT3_2 は DLG2 に，MT1_3, MT2_3 そして MT3_3 は DLG3 に，MT1_4, MT2_4 そして MT3_4 は DLG4 にそれぞれ属する．DLG 中の MT は後述するスケジューリングするフェーズにおいて，同一のプロセッサ上に連続的に割り当てられる．

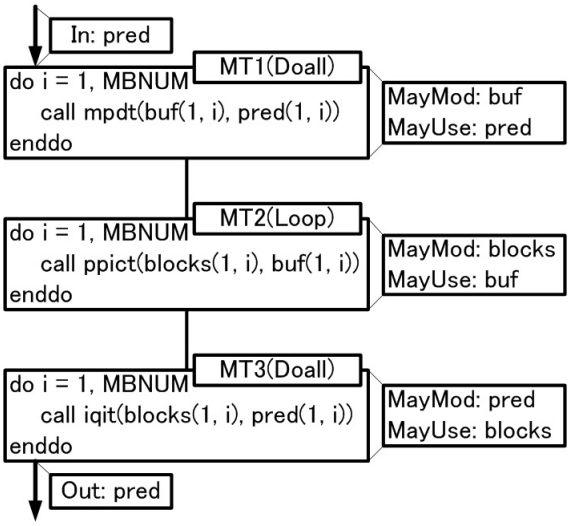

図 6: マクロタスクグラフおよび配列解析例

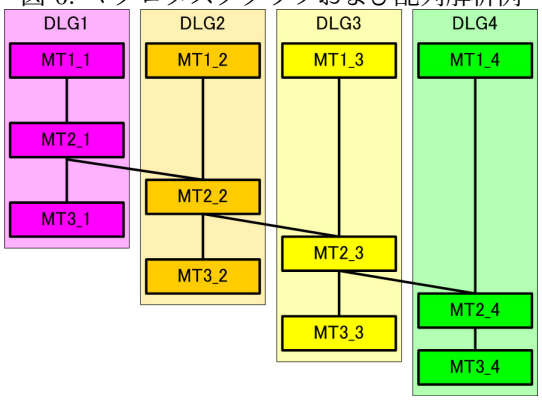

図 7: LAD で 4 分割された MTG

4.2 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．ループ整合分割は DLG 中の全ての MT が同一プロセッサに連続的に割り当てられるように分割を行う．ここで評価するスタティックスケジューラは ETF/CP/MISF[15] をベースとし，タスク割り当ての際に DLG 中の MT を連続的に割り当てるようにプライオリティを変更してある．図 7 に示した MTG をスケジューリングした結果を図 8 に示す．CPU0 には DLG1 と 3 が，CPU1 には DLG2 と 4 がそれぞれ連続的に割り当てられているのが分かる．

4.3 ローカルメモリ管理

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 LDM に配置する．第 4.1 節で説明したように，DLG 中でアクセスされる配列データ量は LDM 上のローカル配列用領域サイズ以下となるように分割数が決定されているため，DLG を連続的にスケジューリングすることで，DLG 中でアクセスされるデータは LDM 上のローカル配列用領域に載ることになる．そこで，DLG 中の各配列のアクセス範囲に応じ，ローカルメモリ上に部分配列領域を確保し，各配列を割り付ける．

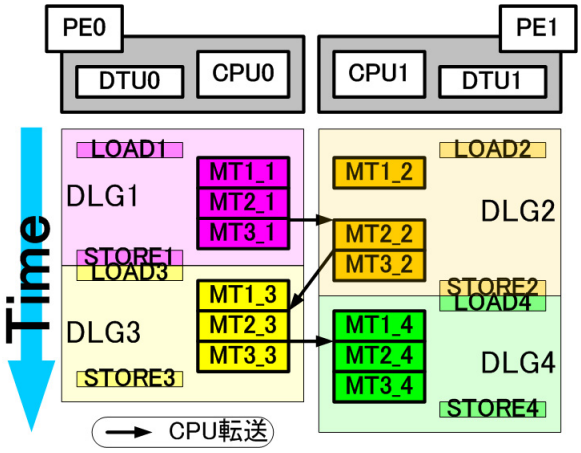

図 8: スケジューリング結果とデータ転送タイミング

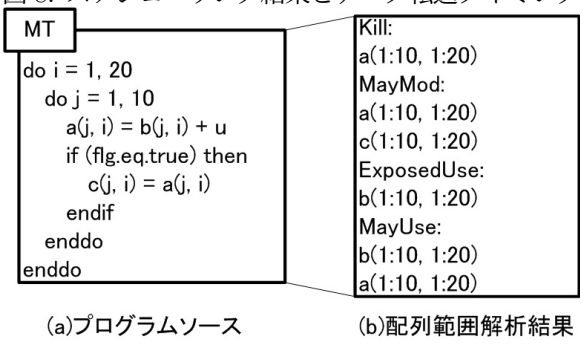

図 9: 配列範囲解析

4.4 データ転送計算

次に，コンパイラはデータ転送範囲の計算を行う．本手法ではデータ転送は必ず MT 実行の先頭あるいは末尾のタイミングで駆動そして転送終了確認されるものとする．

まず，コンパイラ内部での配列範囲解析について述べる．各 MT 毎に以下の四つの配列領域を解析する．すなわち，MT 中で必ず定義される配列領域 (Kill)，定義される可能性のある配列領域 (MayMod)，前方露出参照される配列領域 (ExposedUse)，参照される可能性のある配列領域 (MayUse) の四つを解析する．配列範囲は各次元毎に上下限値で表される．図 9a のプログラムソースにおいて，配列 c は条件分岐により，配列の定義が決まるので，Kill には含まれず，Kill は配列 a となる．MayMod は MT 中で定義される可能性のある全ての配列を表すので，配列 a, c となる．ExposedUse は MT 中で定義よりも先に参照される前方露出参照を表すので，配列 b となる．MayUse は MT 中で参照される可能性のある全ての配列を表すので，配列 b, a となる．配列範囲解析結果を図 9b に示す．配列範囲間の演算として，差を'−'，積を'∩'，和を'∪' でそれぞれ表す．

ローカライズを行う階層の全ての MT について配列範囲解析を行う．配列範囲解析結果をもとにデータ転送のタイミングとその範囲を求める．まず，DLG 中でアクセスされる配列 A のデータ転送タイミングについて考える．DLG 中で最も早く A にアクセス (MayMod または MayUse) する MT_j を探す．MT_j の実行開始を A の CSM から LDM へデータ転送 (Load) するタイミングとする．次に DLG 中で最も遅く A を定義 (MayMod) する MT_k を探す．MT_k の実行終了を A の LDM から CSM へデータ転送 (Store) するタイミングとする．たとえば，図 6 において，配列 pred に注目すると，pred が最初にアクセス (MayUse) されるのは MT1 なので，MT1 の実行開始が配列 pred を Load するタイミングとなる．また，配列 pred が最も遅く定義 (MayMod) されるのは MT3 なので，MT3 の実行終了が配列 pred を Store するタイミングとなる．

次にデータ転送範囲を計算する．DLG_l の配列 A について Load する範囲 $Load_l^A$ は

$$Load_l^A = In_j \cap (\cup_{DLG_l\text{中の }MT_m}(ExposedUse_m^A \cap (MayMod_m^A - Kill_m^A)))$$

となる．ただし，In_j は MT_j に生きて入る配列を表す．$ExposedUse_m^A$，$MayMod_m^A$ そして $Kill_m^A$ は MT_m の配列 A に関する ExposedUse, MayMod そして Kill をそれぞれ表す．

次に DLG_l の配列 A について Store する範囲 $Store_l^A$ は

$$Store_l^A = Out_k \cap (\cup_{DLG_l\text{中の }MT_m}(MayMod_m^A))$$

となる．ただし，Out_k は MT_k から生きて出る配列を表す．$MayMod_m^A$ は MT_m の配列 A に関する MayMod を表す．

以上をもとに図 7 の MTG についてデータ転送を考えると，図 8 に示すように，MT1_[1234] の先頭と MT3_[1234] の末尾にそれぞれ DTU による Load と Store がスケジューリングされる．

5 性能評価

本章では OSCAR マルチコア上でのデータローカライゼーションおよび 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

MediaBench[16] 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て本手法の性能評価を行う．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 で用いられるものと同一とする．本手法はスタック上の配列を対象とし，配列用の領域として LDM のサイズを 256kB 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，DSM は 1 クロック，そして LPM は 1 クロックと設定した．チップ内のメモリレイテンシの算出には ITRS 20003[17] および CACTI[18] を，チップ外のレイテンシには Elipida Memory 社のデータシート [19),20)] をそれぞれ用いた．評価した PE 数は 1, 2, 4, 8 そして 16 である．プログラム実行の初期状態では，全ての配列データは CSM 上に配置されている．DTU のバースト幅は 64byte とした．バースト幅 64byte 転送時の CSM メモリアクセスクロック数は CPU 転送では 192 クロック，DTU 転送では 45 クロックとなる．また，クロックレベルの詳細なシミュレータを用い，評価を行った．

5.2 性能評価結果

MPEG2 エンコーディングの性能評価結果を図 10 に示す．図中の横軸はプロセッサ数を，縦軸は逐次実行時間に対する速度向上率をそれぞれ表す．本手法の有効性を

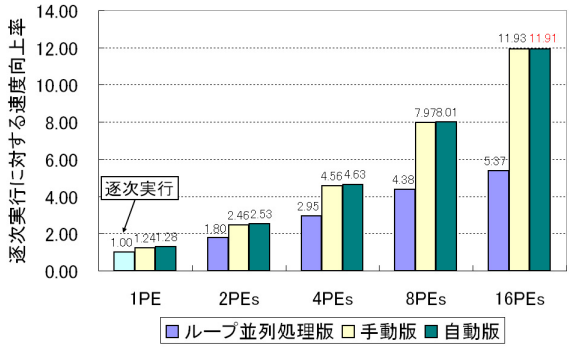

図 10: MPEG2 エンコーディングの性能評価結果

示すために，従来のマルチプロセッサシステム用並列化手法であるループ並列処理を適用した場合の性能を左側のバーで，手動でデータローカライゼーション手法およびオーバラップ転送を適用した場合 [21] の性能を中央のバーで，コンパイラにより自動でデータローカライゼーション手法を適用した場合の性能を右側のバーでをそれぞれ示す．ここで，オーバラップ転送とは CPU の処理中に，DTU によるデータ転送を行い，データ転送時間を隠蔽する技術である．

性能評価結果より，ループ並列処理適用版，手動版，自動版それぞれについて，4 プロセッサで 2.95 倍，4.56 倍，4.63 倍，8 プロセッサで 4.38 倍，7.97 倍，8.01 倍，16 プロセッサで 5.37 倍，11.93 倍，11.91 倍の速度向上が得られ，自動版により従来の手動版と同等以上の性能向上が得られることが確かめられた．本手法をループ並列処理版と比較すると，自動化によりループ並列処理版では選択されていなかった配列変数がローカルメモリ上に配置されたこと，CPU の代わりに DTU により LDM–CSM 間のデータを転送したこと，そしてループ中でアクセスされる変数をループの前後でデータ転送した場合，全体で約 106MB のデータ転送が必要となるのに対し，ループ間のデータローカリティを考慮すると，データ転送量は約 83MB に削減できることの以上 3 点が速度向上の理由である．次に本手法と手動版 [21] の比較では，自動化により手動版では選択されていなかった配列変数がローカルメモリ上に配置されたため，本手法ではオーバラップ転送を適用していないにも関わらず，手動版とほぼ同様のスケーラブルな速度向上が得られた．本性能評価で用いた MPEG2 エンコーディングプログラム中でアクセスされる配列の総サイズは約 76MB である．各プロセッサ上に搭載された LDM のサイズが本自動化手法では 256kB とはるかに小さいメモリでも，効率よく並列処理が行え，データローカライゼーション自動ローカルメモリ管理の有効性が確かめられた．

6 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おける CPU と非同期に動作するデータ転送ユニット (DTU) を用いたデータ転送によるデータローカライゼーションについて述べた．データローカライゼーション手法，DTU 制御命令自動生成および LDM 管理手法を 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，MPEG2 エンコーディングにおいて 8PE でデータローカライゼーション手法を適用し，DTU 転送を行った場合，8.01 倍の速度向上が得られ，以前行った手動版とほぼ同等の結果が得られ，本手法の有効性が示された．

本研究の一部は 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”，日本学術振興会特別研究員奨励費 (#1501202)，文部科学省科学研究費補助金若手研究 (B)(#15700074)，NEDO“先進ヘテロジニアスマルチプロセッサ技術” 及び NEDO“リアルタイム情報家電用マルチコア技術” 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] 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] 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).

[13] 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).

[14] APC: http://www.apc.waseda.ac.jp/.

[15] 笠原: 並列処理技術, コロナ社 (1991).

[16] 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).

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

[18] 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).

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

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

[21] 小高, 中野, 木村, 笠原: チップマルチプロセッサ上での MPEG2 エンコードの並列処理, 情報処理学会論文誌, Vol. 46, No. 9, pp. 2311–2325 (2005).