2006－ARC－167（10）
2006－HPC－105（10）
2006／2／27

マルチコア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転送オーバラップ

宮本孝道[†]　中川正洋[†]　浅野尚一郎[†]
内藤陽介[†]　仁藤拓実[†]　中野啓史[†]
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であり，それに伴い発生するデータ転送によるオーバヘッドを減少させなければならない．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いて，DTU を利用したデータ転送オーバラップを考慮したタスクスケジューリングアルゴリズムとデータ転送スケジューリング手法によるデータ転送オーバラップ手法について述べる．提案手法を OSCAR コンパイラに組み込み，JPEG2000 エンコーディングプログラムに適用して評価を行った結果，4PE ではほとんど全てデータをローカルメモリに割り当てた 1PE での処理と比べ 2.86 倍の速度向上率が得られた．

Data Transfer Overlap of Coarse Grain Task Parallel Processing on a Multicore Processor

Takamichi Miyamoto[†], Masahiro Nakagawa[†], Shoichiro Asano[†], Yosuke Naito[†], Takumi Nito[†], Hirofumi Nakano[†], 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, 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 processor is important for reducing large memory access overhead. Futhermore, hiding data transfer overhead among local or distributed shared memories of processors and centralized shared memory is important. On the memory architechture, the data transfer is specified. 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 proposes and evaluates a static data transfer scheduling algorithm aiming at overlapping data transfer overhead. As the results, the proposed scheme controlled by OSCAR compiler gives us 2.86 times speedup using 4 processors for JPEG2000 encoding program against the ideal sequential execution assuming that the all data can be assigned to the local memory.

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，消費電力の増大等が顕在化し，今後の進展が難しくなっている．これらの問題に対処するためマルチコアプロセッサが注目を集

[†]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

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ことにより，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低電圧で動作させ，適切に並列処理することで，高性能化，低消費電力化が実現可能なアーキテクチャとして期待されている．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残り，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．特にローカルメモリを持つアーキテクチャではメモリ間のデータ転送命令を明示的に発行するとともに，プログラムの並列性を十分引き出すためにはデータ転送オーバヘッドの隠蔽が重要となる．

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),7)]．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オンチップあるいはオフチップ集中共有メモリ (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．

本稿ではマルチコアプロセッサにおけるデータ転送に焦点を当て，各プロセッサエレメント (PE) 近傍のメモリであるローカルメモリあるいは分散共有メモリのサイズを十分大きなものとして扱うことで，マルチコア上で発生する集中共有メモリと PE 近傍のメモリ間の転送，自 PE の近傍のメモリから他 PE の近傍のメモリへの転送となる PE 間転送のデータ転送オーバヘッドの最適化を対象問題とする．

データ転送の隠蔽手法として Direct Memory Access Controller(DMAC) を用いたデータプレロード・ポストストア (PLPS) 手法 [8)~11)] によりプログラムの演算処理とデータ転送のオーバラップによるデータ転送オーバヘッド隠蔽手法が提案されている．従来，DMAC の利用として，マルチメディアプロセッサに対してのプログラミングとして off-chip memory から on-chip memory への転送 [12)] を行う手法がよく用いられている．

本稿では OSCAR マルチグレイン並列化コンパイラにより，プログラムを大域的に解析し，プログラム全域のデータ転送オーバヘッドをプロセッサコアと非同期に動作可能な DTU を用いて隠蔽する自動化手法をマルチコアを対象として提案し，評価を行う．

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 章ではデータ転送オーバラップ手法について述べる．第 5 章では本手法の性能評価を JPEG2000 エンコーディングプログラムを用いて行う．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

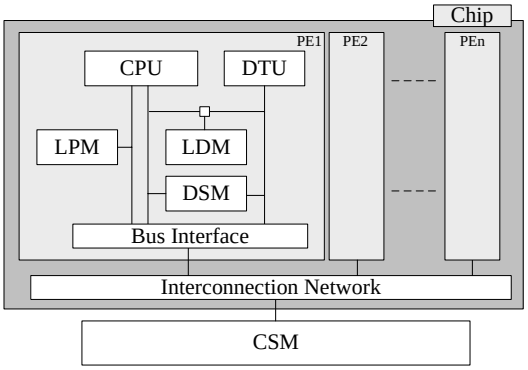

図 1: OSCAR マルチコアアーキテクチャ

の全ての PE はバスやクロスバといった Interconnection Network によってオンチップあるいはオフチップ集中共有メモリ (CSM) に接続されるが本稿ではオフチップ CSM を仮定し評価を行う．

2.1 データ転送ユニット (DTU)

今回の評価で仮定する 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．OSCAR DTU では連続領域転送，転送元，転送先でストライド長が異なるストライド転送，SCATTER, GATHER 転送が可能であり DTU 命令は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．

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．

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．

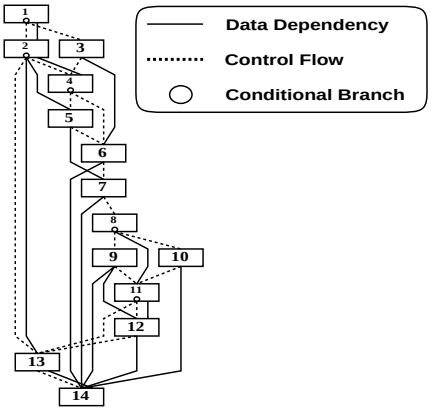

図 2: マクロフローグラフの例

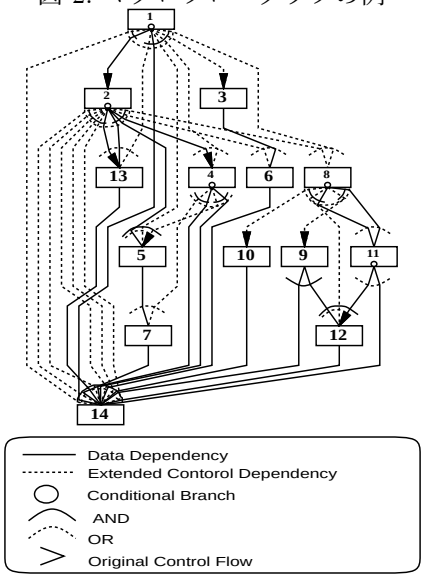

図 3: マクロタスクグラフの例

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，コントロール依存とデータ依存を考慮した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依存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．

4 マルチコアにおけるデータ転送オーバラップ手法

本稿ではプロセッサの演算処理とのオーバラップを考慮した配列データ転送駆動タイミングのコンパイラ内での自動決定を行う．本自動化手法は，コンパイラ内において生成されたマクロタスクグラフに対して，配列範囲データ依存解析，タスクスケジューリング，データ転送スケジューリングを行う．

4.1 データ転送オーバラップ

マルチコア上で対象とするデータ転送として，他 PE の分散共有メモリへのデータ転送 (DSM_STORE)，PE にローカルなメモリから集中共有メモリへのデータ転送 (CSM_STORE)，集中共有メモリから PE にローカルなメモリへのデータ転送 (LM_LOAD) を考える．

データ転送オーバラップは，図 4 で示すようにデータ転送をプロセッサの演算処理と並行して行うことで，データ転送時間をプログラムの処理時間から隠蔽する技術である．

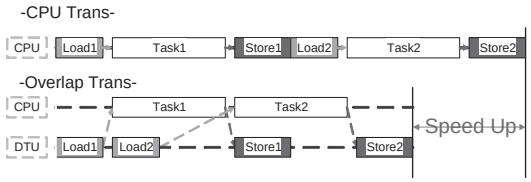

図 4: データ転送オーバラップ

4.2 MT 間配列範囲データ依存解析

メモリアーキテクチャにおいてマクロタスクの効率の良い実行のために PE のローカルメモリへ明示的にデータ転送を行わなくてはならない．配列に対するデータ転送範囲計算はデータ転送量の削減の観点から非常に重要となる．

マクロタスクグラフのエッジがデータ依存エッジのみの場合，マクロタスク間における配列データ依存範囲はマクロタスク間のデータ転送範囲となる．

以下に，マクロタスク間の配列範囲データ依存解析の計算手順を述べる．ここで，配列範囲データ間の演算として，差を'−'，積を'∩'，和を'∪' でそれぞれ表す．

step1. 全ての MT における配列の U，D を求める．ここで MT_i の U_i，D_i を以下のように定義する．

$$U_i = In_i \cap \left(ExposedUse_i \cup (MayMod_i - Kill_i)\right)$$
$$D_i = Out_i \cap MayMod_i$$

ここで，In_i は MT_i に生きて入る配列，$ExposedUse_i$ は MT_i で定義前に参照される前方露出参照される配列，$MayMod_i$ は MT_i で定義される可能性のある配列，$Kill_i$ は MT_i で確実に定義される配列，Out_i は MT_i から生きて出る配列を表す．

step2. 全ての MT をコントロールフロー順に辿り，以下の手順を繰り返す．

step3. ある MT_i からの配列データ依存範囲を計算する．ここで，MT_i の D_i を D_valid_i とする．

step4. MT_i からコントロールフローを辿り，比較対象となる MT を MT_j とする．

step5. D_valid_i と MT_j の U_j を比較を行い，重なる配列範囲をフロー依存範囲 FD_{i-j} とする．

$$FD_{i-j} \quad = \quad D\_valid_i \cap U_j$$

step6. MT_i からコントロールフロー順に MT_j，MT_k に対してフロー依存 FD_{i-j}，FD_{i-k} が得られた場合，MT_j から MT_k に対して，FD_{i-j} と FD_{i-k} の重なる配列範囲が入力依存範囲 ID_{j-k} となる．

$$ID_{j-k} \quad = \quad FD_{i-j} \cap FD_{i-k}$$

step7. D_valid_i から MT_j の D_j との重なる範囲を除き，残存する範囲である D_valid_i を以降用いる．

step8. step4 から step6 を D_valid_i の範囲が空になるまで繰り返す．ただし，比較演算結果が不定となる MT や分岐 MT に到達した場合にはその MT までの D_valid_i を全て集中共有メモリへの書き出しを行う．

本配列範囲データ依存解析を図 5 に示す．

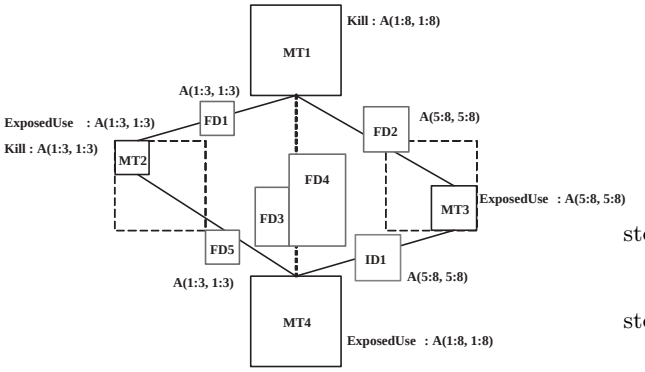

図 5: 配列範囲データ依存解析

4.3 データ転送オーバラップを考慮したスタティックスケジューリング

配列範囲データ依存解析により図 6 のような MTG が得られる．図 6 では，ノードが MT，ノードの隣の数字が CP 長を示し，MT 間のエッジ上のフロー依存 FD，入力依存 ID が配列範囲データ依存情報を示している．EMT は MTG の出口ノードであり，EMT が実行可能になった時がプログラムの終了を示す．

本稿で行うデータ転送オーバラップを考慮したスタティックスケジューリングアルゴリズムを以下に示す．本アルゴリズムはマルチプロセッサスケジューリング問題におけるヒューリスティックアルゴリズム ETF/CP 法に対してデータ転送オーバラップ手法 (PLPS) を適用したアルゴリズムである．

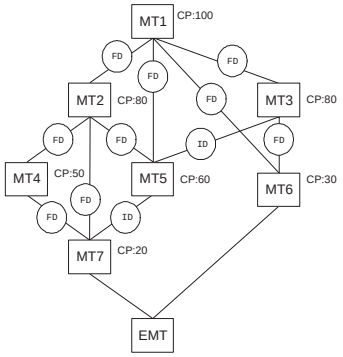

図 6: スケジューリング対象 MTG

step1. 各 MT から出口ノードまでの最長パス長（CP 長）の大きい順に各 MT に優先度をつける．

step2. 先行タスクが全て割り当てられたレディタスク集合に対して，各レディタスクを各プロセッサに割り当てた場合を考える．

step3. あるレディタスク MT_i をあるプロセッサ PG_j に割り当てた場合のデータ転送オーバラップを考慮した最早実行終了時刻を計算する．MT_i を PG_j に割り当てた場合に MT_i 実行に必要となるデータ転送を配列範囲データ依存解析のフロー依存・入力依存からデータ転送を算出し，オーバラップを考慮したデータ転送タイミングを決定する．

ここで，MT_i 実行に必要なデータ転送とは PG_k $(k \neq j)$ で実行される MT_l からのフロー依存範囲 FD_{l-i} が PG_j の DSM への STORE となり，同一 PG に割り当てられた場合にはデータ転送は発生しないものとする．

データ転送オーバラップを考慮したデータ転送駆動タイミング決定方法は，MT_l から MT_i へのデータ転送が存在する場合，対象データ転送における生産 MT が MT_l，消費 MT が MT_i であり，データ転送は生産 MT の終了時刻から消費 MT の開始時刻までの間でタスク処理とオーバラップして転送が可能なタイミングを決定する．

このように決定された全てのデータ転送の終了時刻と割り当て PG_j の空き時刻から MT_i の最早実行可能時刻を求め，MT_i の実行クロックを加算した時刻が最早実行終了時刻となる．

step4. 全てのレディタスクとプロセッサの組み合わせから最早実行終了時刻の最も小さいタスクとプロセッサの組み合わせを割り当てる．

step5. 4. において複数のタスクとプロセッサの組み合わせが存在する場合は，1. で作成した優先度の高い組み合わせを割り当てる．

step6. 2 から 5 を，全てのタスクが割り当てられるまで繰り返す．

4.4 タスクスケジューリング結果に基づいたデータ転送スケジューリング

第 4.3 節のタスクスケジューリング結果から全てのデータ転送が既知となる．これらのデータ転送を，以下に示す優先度決定アルゴリズムによりデータ転送の優先度を決定する．

step1. 生産 MT 終了時刻から消費 MT 開始時刻までのデータ転送の余裕度が短いデータ転送を選択

step2. 1 で等しい場合，転送先 MT の CP 長が大きいデータ転送を選択

step3. 2 で等しい場合，コストの大きいデータ転送を選択

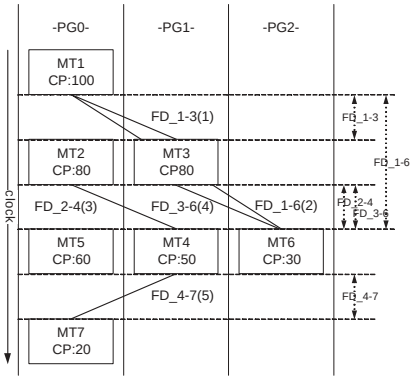

図 7: タスクスケジューリングとデータ転送優先度

図 6 の MTG に対して PG 数 3，インターコネクションネットワークは 1 本バスとした場合のタスクスケジューリング結果とデータ転送優先度決定結果を図 7 に示す．

図 7 では各 PG への MT の割り当てと PG 間データ転送を示し，MT 番号と共にクリティカルパス長を示し，タスクスケジューリング結果の右側にデータ転送の余裕度を示している．データ転送の右隣の括弧で囲まれた数字が決定されたデータ転送の優先度の順番を示したものである．

決定された優先度順にデータ転送の駆動タイミングを決定する．各データ転送に対して，対象アーキテクチャである OSCAR マルチコアのインターコネクションネットワークの占有状態，対象メモリの利用状態，使用 DTU の利用状態等を考慮してデータ転送の駆動タイミングを決定する．ただし，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．

図 8 では決定されたデータ転送の駆動タイミングとバスの占有状態を示している．

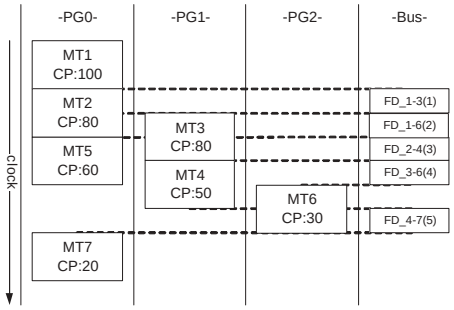

図 8: データ転送スケジューリング

5 性能評価

本章では OSCAR マルチコア上での本データ転送オーバラップ手法の性能評価結果について述べる．

5.1 評価環境

プロセッサコアの周波数は組込み用途を想定した 400MHz として，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，ローカルの DSM は 1 クロック，リモートの DSM は 4 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[13] および CACTI[14] を，チップ外のレイテンシには Elpida Memory 社のデータシート [15),16)] をそれぞれ用いた．

5.2 評価アプリケーション

JPEG2000 エンコーディングプログラムである jj2000[17] を FORTRAN で参照実装したプログラムを用いる．JPEG2000 エンコーディング処理は，DC レベル変換 (DC Level Shift)，離散ウェーブレット変換 (DWT:Discrete Wavelet Transformation)，スカラ量子化 (Quantization)，係数モデリング (Coefficient bit modleling)，算術符合化 (MQ-Coder)，ビットストリーム出力 (Output Bit Stream) の 6 ステージで分けられる．図 9 に JPEG2000 エンコーディングのブロック図を示す．離散ウェーブレット変換ステージでは画像全体に対する処理であり，スカラ量子化，係数モデリング，算術符号化は 64x64 ピクセルのコードブロックと呼ばれる単位で処理を行う．ここで，スカラ量子化は各サブバンドに対して行う処理であるがコードブロックがサブバンド境界を越えて生成されないためコードブロック単位で処理を行うことが可能となる．

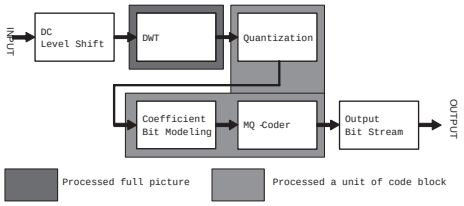

図 9: JPEG2000 エンコーディングのブロック図

本手法の評価では，入力画像として，128x128 ピクセルの画像，800x600 ピクセル (SVGA) の画像をそれぞれ用い，圧縮方式は不可逆変換，ウェーブレットレベルを 3，その他のエンコードパラメータは JJ2000 エンコードのデフォルトパラメータとして評価を行った．

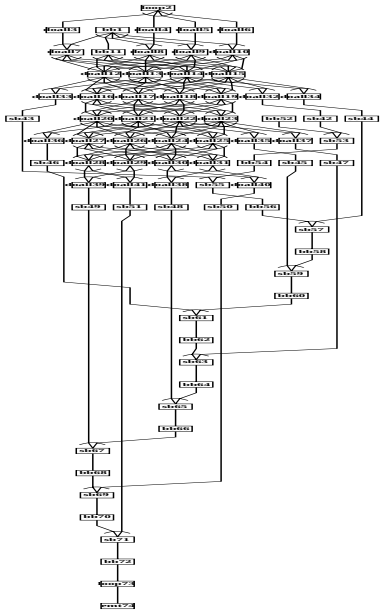

図 10: 128x128 画像サイズにおける JPEG2000 エンコーディングのマクロタスクグラフ例

図 10 に 128x128 ピクセルの画像における JPEG2000 エンコーディングの MTG を示す．本例では DC レベル変換，ウェーブレット変換をタスク分割し，コードブロック毎の処理が各タスクに分割された MTG である．ウェーブレット変換では垂直方向，水平方向の処理間で転置転送が存在するためデータ転送が多く発生しているのが見て取れる．

5.3 性能評価結果

JPEG2000 エンコーディングの性能評価結果を図 11 に示す．図中の横軸には，各画像サイズにおける評価プロセッサ数を示し，縦軸には，1 プロセッサで配列デー

タを集中共有メモリへ置いて処理を行った場合の処理時間に対して各評価の速度向上率を表す．ここでいう処理時間とは第 5.2 節で示したエンコード処理部分である 6 ステージの実行時間である．各バーは，配列データを全て集中共有メモリ (CSM) 上に置いて処理を行った場合 (CSM_Exec)，配列データをローカルなメモリへ CPU により転送を行い処理を行った場合 (CPU_DT)，配列データをローカルなメモリへ DTU を利用して本オーバラップ手法を適用した場合 (Overlap_DT) をそれぞれ表す．

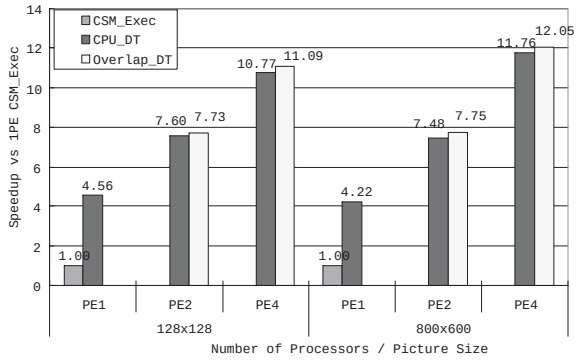

図 11: JPEG2000 エンコーディングの性能評価結果

図 11 の性能評価結果より，1PE で集中共有メモリ上にデータ配置をした処理に対して，プロセッサのローカルメモリにデータを配置し CPU データ転送を行った場合 128x128 画像サイズで 4PE 使用時 10.77 倍，本データ転送オーバラップ手法を適用した場合では 11.09 倍の速度向上が得られ，800x600 画像サイズにおいては，ローカルメモリへデータ配置し CPU データ転送を行った場合，4PE で 11.76 倍，本データ転送オーバラップ手法を適用した場合では 12.05 倍の速度向上が得られた．

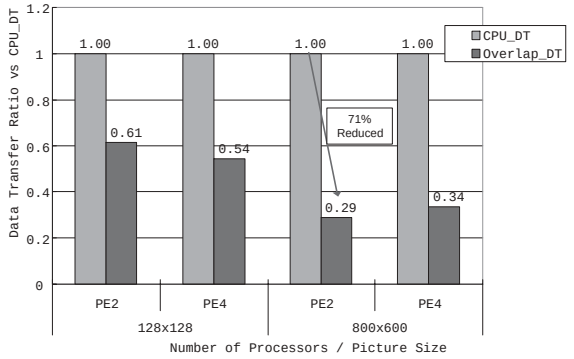

図 12: データ転送コストの削減結果

本アプリケーションではローカルメモリへのデータ配置スケジューリングによってデータ転送が最小化され，4PE の場合ではデータ転送が処理時間に占める割合は約 10%であったためオーバラップによる速度向上は小さな値となっているが，図 12 のデータ転送コストの削減結果から，CPU によるデータ転送に対して本データ転送オーバラップ手法では最大 71%のデータ転送コストを削減したことが確認された．

6 まとめ

本稿では，マルチコア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オーバラップ手法について述べた．本手法を OSCAR マルチグレイン並列化コンパイラに組み込み，OSCAR マルチコア上で性能評価を行った．その結果，JPEG2000 エンコーディングにおいて，集中共有メモリ上にデータを配置して処理を行う場合に対して，ローカルメモリへのデータ配置によるデータ転送の最小化および残ったデータ転送の DTU によるオーバラップにより，エンコード処理が 800x600 画像サイズの時，2PE で 7.75 倍，4PE で 12.05 倍の速度向上が得られることが確認できた．

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”，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] 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 (1999).

[7] 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).

[8] 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, 信学論, Vol. J75-D-I, No. 8, pp. 495–503 (1992).

[9] 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-I), Vol. J75-D-I, No. 8, pp. 495–503 (1988).

[10] 藤本, 橋本, 笠原: データ転送と処理のオーバーラップを用いたデータ転送最小化自動並列化コンパイラ, 電気学会情報処理研究会資料, No. IP-96-24 (1996).

[11] 木村, 古郷, 尾形, 笠原, 橋本: 処理とデータ転送のオーバーラッピングを考慮したダイナミックスケジューリングアルゴリズム, *CPSY97* (1997).

[12] D Kim, R Managuli, Y. K.: Data cache and direct memory access in programming mediaprocessors, pp. 33–42 (2001).

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

[14] 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).

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

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

[17] R Grosbois, D Santa Cruz, J. A. B. B. D. B. F. H. G. M. and Onno, P.: http://jj2000.epfl.ch/.