2004－ARC－159　(20)
2004／7／31

OSCARチップマルチプロセッサ上でのデータ転送ユニットを用いたデータローカライゼーション

中野　啓史[†]　内藤　陽介[†]　鈴木　貴久[‡]
小　高　剛[†]　石坂　一久[†]
木村　啓二[†]　笠原　博徳[†]

現在，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ている．これらの CMP アーキテクチャにおいても，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は依然存在する．筆者等はこのメモリウォールの問題に対処し，高い並列性を抽出し効果的な並列処理を実現するために，マルチグレイン並列処理との協調動作により実効性能が高く価格性能比の向上を可能にする OSCAR CMP を提案している．この OSCAR CMP は，集中共有メモリ (CSM) に加え，プロセッサ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)，プロセッサコア間の同期やデータ転送にも使用する 2 ポートメモリ構成の分散共有メモリ (DSM)，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，FORTRAN プログラムをループ・サブルーチン・基本ブロックを粗粒度タスクとする．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い，プログラムのデータローカリティを抽出するデータローカライゼーション手法について述べる．データ転送は，コンパイラにより自動生成した DTU によるデータ転送命令を用いてバースト転送を行う．

Data Localization using Data Transfer Unit on OSCAR Chip Multiprocessor

Hirofumi Nakano[†], Yosuke Naito[†], Takahisa Suzuki[‡], Takeshi Kodaka[†], Kazuhisa Ishizaka[†], Keiji Kimura[†] and Hironori Kasahara[†]

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, centralized shared memory (CSM) to support dynamic task scheduling, and data transfer unit(DTU) for asynchronous data transfer. 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. Data is transferred in burst mode using automatically generated DTU instructions.

1　はじめに

半導体技術の進歩に伴い，複数のプロセッサコアをチップ上に搭載する Chip Multiprocessor(CMP)，たとえば Power4[1)]，UltraSPARCIV[2)] や MP98[3)] が注目を集めている．これらのアーキテクチャでも従来から問題となっているメモリウォールの問題は依然存在し，これを克服することが性能向上のためにさらに重要となっている．

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[4)] や，ループ分割後のタスクの垂直実行 [5)] が提案されている．配列間のキャッシュコンフリクトを削減する手法として配列間の padding[6)] が提案されている．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 や modulo unrolling[7)] が，FlexRAM 上では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 [8)] 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，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 [9)] と協調動作する

[†]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）富士通研究所システム LSI 開発研究所プロセッサソリューション開発部

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，集中共有メモリ (CSM)，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)，プロセッサコア間の同期やデータ転送にも使用する 2 ポートメモリ構成の分散共有メモリ (DSM)，そしてプロセッサコアと非同期にデータ転送が可能なデータ転送ユニット (DTU) を持つ．LDM サイズを考慮し，粗粒度タスクを分割するループ整合分割 [11),12)] 後，並列性とデータローカリティを最大限に抽出するようにスケジューリングを行い，データローカライゼーションを実現している．さらに DTU によるデータ転送命令をコンパイラによって自動的に生成し，バースト転送を使い，効率の良いデータ転送を実現している．

本稿では，粗粒度タスク並列処理 [6),13)] におけるループ整合分割 [11),12)] を利用した，データ転送ユニットを用い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．本手法を OSCAR Fortran マルチグレイン並列化コンパイラ [14)] 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 OSCAR CMP アーキテクチャおよび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．

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 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．

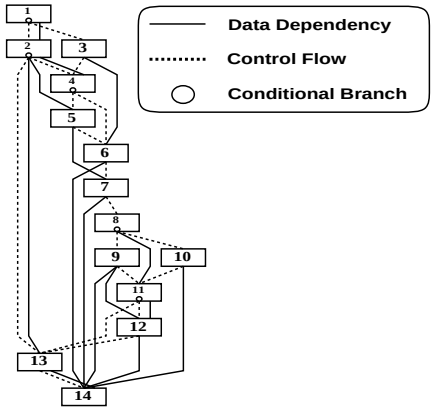

図 1: マクロフローグラフの例

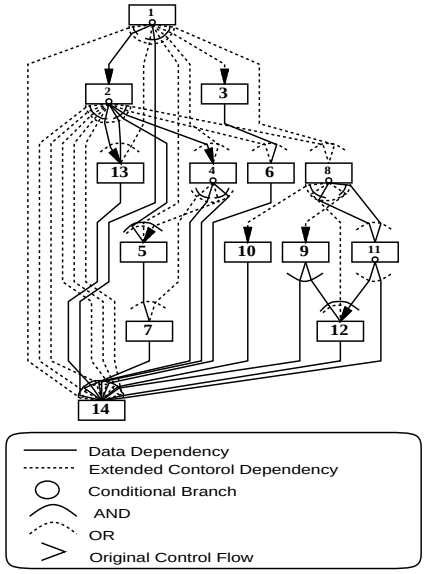

図 2: マクロタスクグラフの例

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．

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

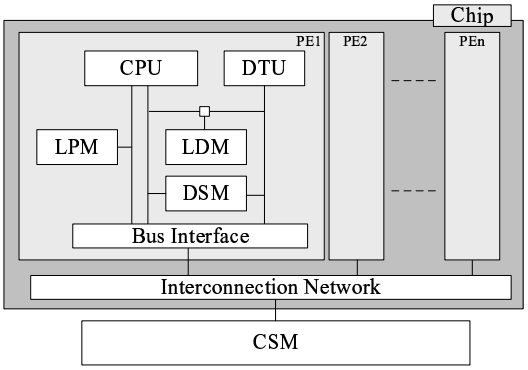

図 3: OSCAR Chip Multiprocessor アーキテクチャ

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 OSCAR CMP 上でのデータローカライゼーション

データローカライゼーション [9),11)] では，OSCAR コンパイラは各プロセッサに近接した高速なメモリを有効利用するためにデータローカリティの最適化を行う．データローカライゼーションは大量の配列データを共有するループを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)[11),12)] する．本稿では，OSCAR コンパイラは分割されたループをデータローカリティを最大化するように最早実行可能条件を考慮して静的にループの実行順序を最適化する．データローカライゼーションを行っても削減できなかったデータ転送はコンパイラによって自動的に生成した DTU 転送命令により転送を行う．

3.1 OSCAR Chip Multiprocessor (CMP)

OSCAR CMP アーキテクチャを図 3 に示す．OSCAR CMP は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用を保持する 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．さらに本稿で集中共有メモリがチップ外に接続されている．

3.2 データ転送ユニット (DTU)

OSCAR CMP 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

1. 転送コマンド
2. 転送元アドレス
3. 転送先アドレス
4. 転送領域サイズ
5. 転送終了通知フラグアドレス

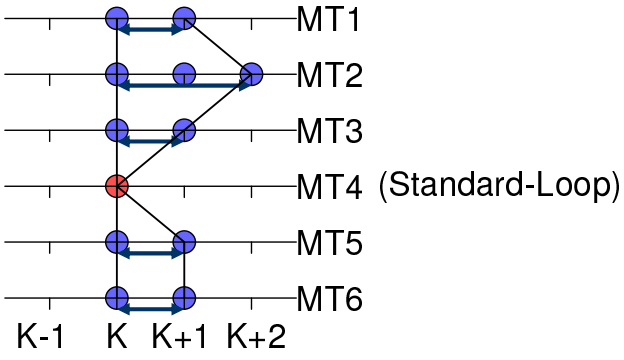

図 4: Inter Loop Dependence 解析の結果

である．これらを指定することで連続領域の一回分の転送を行うことができる．さらに

1. 転送元ストライド長
2. 転送先ストライド長
3. 転送回数

を設定可能である．これらを設定することで，転送先，転送元の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コンパイラが自動的に生成している．

DTU の起動には二種類の方法がある．一つはコンパイラにより生成された上述のパラメータをローカルメモリ上に格納し，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，もう一つは CPU が転送パラメータ値を直接 DTU のレジスタに設定する方法である．転送の終了は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することで，割り込み処理を使わずに同期をとることが可能である．

3.3 データローカライゼーション

各 PE 上の LDM の有効利用に向けてプログラムからデータローカリティの抽出を行うためにループ整合分割 (Loop Aligned Decomposition: LAD)[11),12)] を用いたデータローカライゼーション手法を適用する．

データローカライゼーションにおいては，データ依存エッジで接続された繰り返しブロックを Target Loop Group(TLG) としてグループ化する．

MT の分割のために，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．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．標準ループの k 番目のイタレーションが依存する，または標準ループの k 番目のイタレーションに依存する各ループのイタレーションを図 4 に示したように解析する．

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．ここで N は LDM サイズを考慮して決定される．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．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 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データ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割り当てられる．

ループ整合分割前の MTG の例を図 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 MT2,

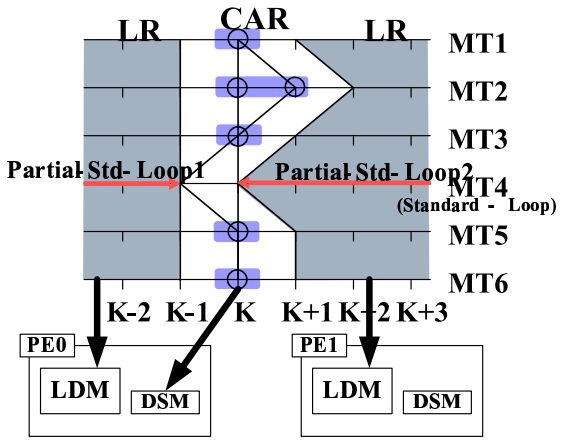

図 5: ループ整合分割例およびメモリへの割り当て

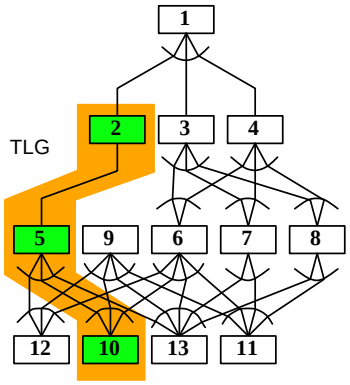

図 6: LAD と DLG の例

5, 10 はそれぞれ図 7 中の MT2, 4, 6 そして 8，MT11, 13, 15 そして 17，MT22, 24, 26 そして 28 にそれぞれ 4 つに分割されている．

図 7 において，斜線のかかった MT がループ整合分割によって生成された MT である．各帯はそれぞれ同一のプロセッサに割り当てられる DLG を表す．たとえば DLG0 は MT2, 11 そして 22 を含む．これは MT2, 11 そして 22 が同一プロセッサ上で連続的に実行されることを意味する．

図 5 に示すように，従来 CSM に配置されていた CAR 内でアクセスされる配列について DSM に配置するように変更を行った．CAR は複数の LR に依存する，あるいは依存されるので，データローカリティは利用できない．そこで，CAR の開始時に CAR 内でアクセスされる配列データを DTU を用いて DSM にロードし，CAR の終了時に生きている配列データを DTU を用いて CSM にストアを行い，CAR 内の配列のメモリアクセス時間を減少させる．

3.4 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 MT は PE に静的に割り当てられる．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[14] をベースとし，タスク割り当ての際に DLG 中の MT を可能な限り連続的に割り当てるように変更してある．

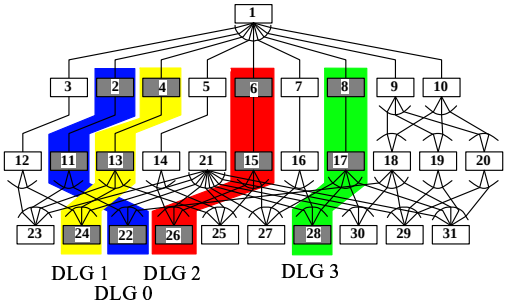

図 7: LAD 後の MTG

3.5 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 にマッピングする．まず，コンパイラは各 TLG の DLG 中でアクセスされる総配列データ量を計算する．次にコンパイラは各 TLG 毎に LDM 上に DLG_Slot と呼ばれる領域を確保する．DLG_Slot のサイズはその TLG の DLG 中でアクセスされる配列データの最大サイズとする．3.4 節で述べたように同一 DLG 中の全ての MT は同一プロセッサ上に連続的に割り当てられるので，同一 DLG に属する MT 内でアクセスされる全てのローカル化された配列データは一つの DLG_Slot を介して授受される．

3.6 データ転送計算

次に，コンパイラはデータ転送情報の計算を行う．本稿では，全てのデータ転送は MT の開始あるいは終了のタイミングで始められる．

同一 DLG 中の MT でアクセスされる全てのローカル化された部分配列は LDM を介して授受される．$DLG_j (1 \leq j \leq n.\ n$ は分割数) に属する MT_i $(1 \leq i \leq N_{MTtotal}$: $N_{MTtotal}$ は全 MT 数) と DLG_j に属さない $MT_k (1 \leq k \leq N_{MTtotal})$ との間および MT_i と TLG の外側階層を表す $Layer_{outside}$ との間でそれぞれデータ転送が必要となる．これらのデータ転送は次のように計算できる．まず，配列の定義使用連鎖 (DU_Chain[生産者][消費者]) を求める．“load” は CSM から LDM へのデータ転送，“store” は LDM から CSM へのデータ転送とそれぞれ定義すると，それぞれ下の計算式で求められる．

$$load[MT_i] = \cup(DU\_Chain[MT_{outside}][MT_i] - DU\_Chain[MT_{outside}][pred])$$

ここで $MT_{outside}$ は DLG_j 以外の MT または $Layer_{outeside}$ を意味し，$pred$ は DLG_j 中の MT かつ MT_i の先行 MT を意味する．

$$store[MT_i] = \cup(DU\_Chain[MT_i][MT_{outside}])$$

ここで，$MT_{outside}$ は DLG_j 以外の MT または $Layer_{outside}$ を意味する．

次に CAR である MT のデータ転送計算について述べる．CAR である MT の開始と終了時点で生きている配列データを転送する必要があるので，CAR である MT_{CAR} の “load” と “store” は下記のようになる．

$$load[MT_{CAR}] = \cup(DU\_Chain[MT_j][MT_{CAR}])$$
$$store[MT_{CAR}] = \cup(DU\_Chain[MT_{CAR}][MT_j])$$

ここで MT_j は MT_{CAR} 以外の MT を表す．

4 性能評価

本章では本手法の OSCAR CMP 上でのデータローカライゼーション手法および DTU によるバースト転送の性能評価結果について述べる．

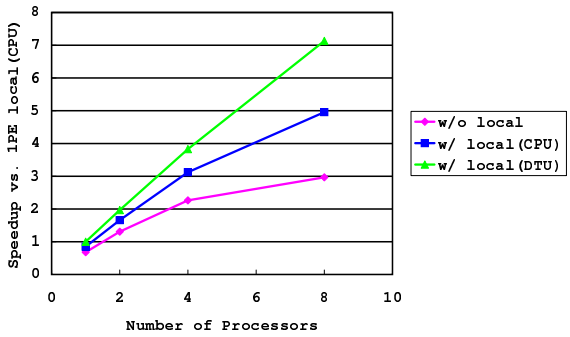

図 8: 400MHz 想定時の OSCAR CMP 上での Swim の性能評価結果

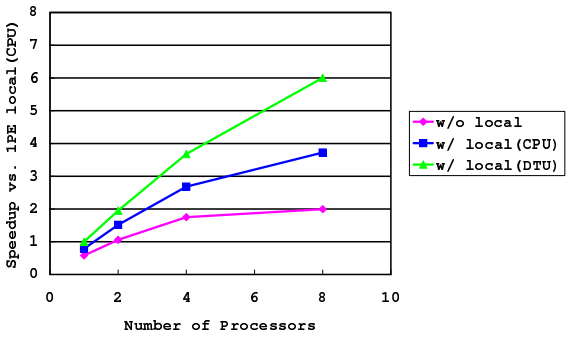

図 9: 1000MHz 想定時の OSCAR CMP 上での Swim の性能評価結果

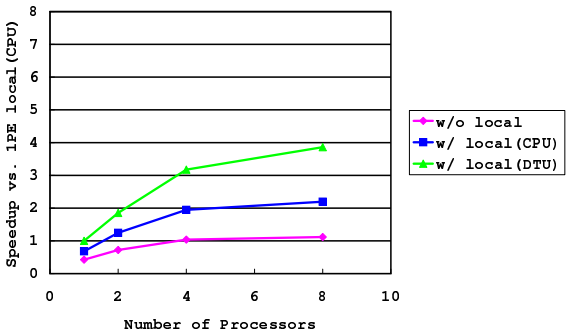

図 10: 2800MHz 想定時の OSCAR CMP 上での Swim の性能評価結果

4.1 評価環境

本手法を SPEC CFP 95 の Swim と Tomcatv を用いて評価を行った．シミュレーション時間短縮のために各ループの回転数と配列サイズを縮小している．本評価で用いた Swim と Tomcatv のデータセットサイズはそれぞれ約 3.3MB と 230KB である．本評価では OSCAR CMP 上の LDM サイズを 256KB とした．また，組み込み用途から科学技術計算での利用を考慮し，低 CPU コア周波数から高 CPU コア周波数までを想定して，表 1 に示す 3 種類のメモリアクセスレイテンシセットを用いて評価を行った．表中の各メモリレイテンシセットの想定プロセッサコア周波数はそれぞれ 400MHz, 1000MHz, 2800MHz である．チップ内のメモリレイテンシは ITRS 20003[15] および CACTI[16] を，チップ外のレイテンシは Elipida Memory 社のデータシート [17),18)] をそれぞれ用いて計算を行った．評価した PE 数は 1, 2, 4 そして 8 である．プログラム実行の初期状態では，全ての分割された配列データは CSM 上に配置されている．DTU のバースト転送時のバースト幅は 64byte とした．バースト幅 64byte 転送時の CPU 転送および DTU のバースト転送による CSM メモリアクセスクロック数を表 2 に示す．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．

表 1: OSCAR CMP のメモリアクセスレイテンシ

	400MHz	1000MHz	2800MHz
CSM latency	24	42	105
LDM latency	1	2	3
DSM latency	1	1	2
LPM latency	1	1	1

表 2: CPU 転送・DTU 転送時の CSM メモリアクセスクロック数

	400MHz	1000MHz	2800MHz
CPU 転送	192	336	840
DTU 転送	45	63	154

4.2 性能評価結果

Swim の想定プロセッサコア周波数が 400MHz の時の性能評価結果を図 8 に，1000MHz の時を図 9 に，2800MHz の時を図 10 にそれぞれ示す．Tomcatv の想定プロセッサコア周波数が 400MHz の時の性能評価結果を図 11 に，1000MHz の時を図 12 に，2800MHz の時を図 13 にそれぞれ示す．図中の縦軸は本手法を適用したときの 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それぞれ表す．図中の “w/o local” が本手法未適用時を，“w/ local(CTU)” がデータローカライゼーションを適用し，CPU によりデータ転送を行った場合を，“w/ local(DTU)” がデータローカライゼーションを適用し，DTU によりデータ転送を行った場合をそれぞれ表す．

図 8, 9 そして 10 において，“w/o local” がスケーラブルな性能向上を示していない．これに対し，“w/ local(CPU)” は 8PE において，400MHz 想定時 5.0 倍，1000MHz 想定時 3.7 倍，2800MHz 想定時 2.2 倍と性能が改善している．これは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 CSM へのメモリアクセスが減少し，“w/o local” において問題となっていた CSM のデータ供給能力不足が改善され，データローカリティの抽出により LDM が有効利用されたからだ．さらに “w/ local(DTU)” は 8PE において，400MHz 想定時 7.1 倍，1000MHz 想定時 6.0 倍，2800MHz 想定時 3.9 倍の速度向上を示している．これは DTU を使ったバースト転送により，データ転送時間が短縮されたからだ．

図 11, 12 そして 13 においても Swim と同様の結果が見られる．まず，“w/o local” がスケーラブルな性能向上を示していない．これに対し，“w/ local(CPU)” は 8PE において，400MHz 想定時 5.7 倍，1000MHz 想定時 4.7 倍，2800MHz 想定時 3.0 倍と性能が改善している．これは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 CSM へのメモリアクセスが減少し，“w/o local” において問題となっていた CSM のデータ供給能力不足が改善され，データローカリティの抽出により LDM が有効利用されたからだ．さらに “w/ local(DTU)” は 8PE において，400MHz 想定時 6.7 倍，1000MHz 想定時 6.3 倍，2800MHz 想定時 5.1 倍の速度向上を示している．これは DTU を使ったバースト転送により，データ転送時間が短縮されたからだ．

“w/ local(CPU)” と “w/ local(DTU)” を比較すると性能が大幅に改善している．これは表 2 に示した通り，400MHz, 1000MHz, 2800MHz 想定時それぞれで約 4.3 倍，5.3 倍，5.5 倍の速度向上がバースト転送により得られ，これにより “w/ local(DTU)” は “w/ local(CPU)” に対し，性能が大幅に向上しているからである．

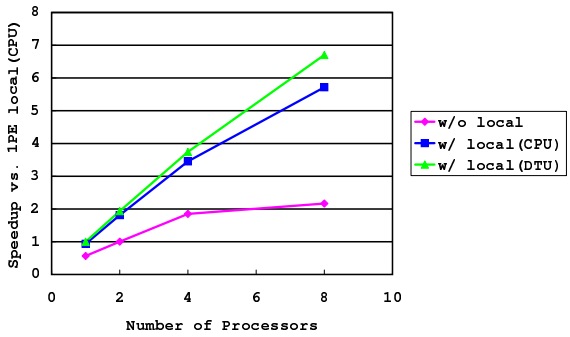

図 11: 400MHz 想定時の OSCAR CMP 上での Tomcatv の性能評価結果

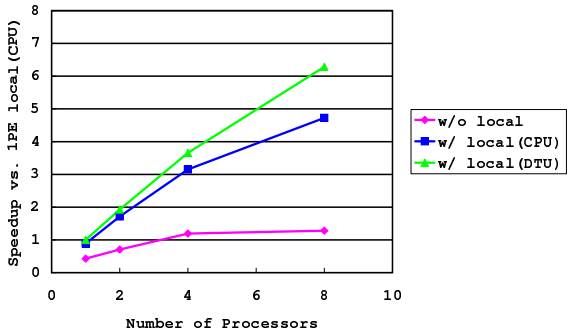

図 12: 1000MHz 想定時の OSCAR CMP 上での Tomcatv の性能評価結果

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および DTU 制御命令自動生成を 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 CFP 95 の Swim 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，CPU 転送と比較し，想定プロセッサコア周波数 400MHz 時，31%，1000MHz 時，38%，2800MHz 時 43%の速度向上をそれぞれ示した．Tomcatv において，400MHz 時 15%，，1000MHz 時 25%,2800MHz 時 42%の速度向上をそれぞれ示した．

本研究の一部は STARC “自動並列化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” 及び日本学術振興会特別研究員奨励費 (# 1501202) によって行われた．

参考文献

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

[2] Krewell, K.: UltraSPARC IV Mirrors Predecessor, *Microprocessor Report* (2003).

[3] Edahiro, M., Matsushita, S., Yamashina, M. and Nishi, N.: A Single-Chip Multiprocessor for Smart Terminals, *IEEE MICRO Magazine*, Vol. 20, No. 4, pp. 12–20 (2000).

[4] 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).

[5] 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).

[6] Ishizaka, K., Obata, M. and Kasahara, H.: Cache Optimization for Coarse Grain Task Parallel Processing using Inter-Array Padding, *Proc. 16th International Workshop on Languages and Compilers for Parallel Computing* (2003).

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

[8] 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).

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

[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] 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).

[12] 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).

[13] 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).

[14] 笠原: 並列処理技術, コロナ社 (1991).

[15] : International Technology Roadmap for Semiconductors 2003 Executive Summar y (2003).

[16] 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).

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

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

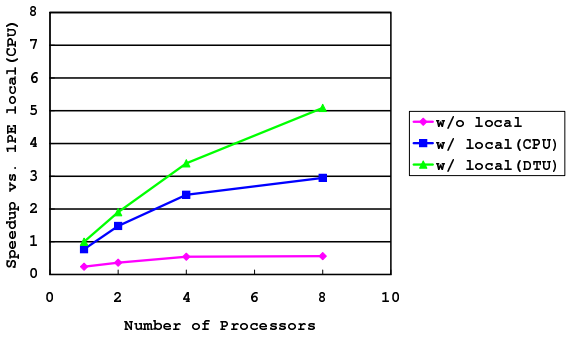

図 13: 2800MHz 想定時の OSCAR CMP 上での Tomcatv の性能評価結果