OSCAR CMP 上でのスタティックスケジューリングを用いたデータローカライゼーション手法

中野　啓史[†]　　小　高　剛[†]
木村　啓二[‡]　　笠原　博徳[†]

E-mail: {hnakano,kodaka,kimura,kasahara}@oscar.elec.waseda.ac.jp

近年の集積度向上に伴い，1 チップ上に複数のプロセッサを集積するチップマルチプロセッサ・アーキテクチャの実用化が進められている．筆者等はこれまで，1 チップ上で複数粒度の並列性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を指向した，OSCAR チップマルチプロセッサ (OSCAR CMP) を提案してきた．OSCAR CMP はチップ内のプロセッサ・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)，プロセッサ間共有データを格納する 2 ポート構成の分散共有メモリ (DSM) を搭載し，コンパイラがデータ配置を適切に制御する．本稿では，データを共有するループやサブルーチン等の粗粒度タスクを同一プロセッサで連続的に実行することでデータローカリティ最適化を図るデータローカライゼーション手法の，OSCAR CMP に対する適用について述べる．さらに，OSCAR CMP にデータローカライゼーション手法を適用して評価した結果を，共有キャッシュアーキテクチャやスヌープキャッシュアーキテクチャと比較し，現在の OSCAR CMP 用の単純なコード生成に対する改善点の考察も行う．

Data Localization Scheme using Static Scheduling on Chip Multiprocessor

Nakano Hirofumi[†]　, Kodaka Takeshi[†]　, Kimura Keiji[‡]
and Kasahara Hironori[†]

E-mail: {hnakano,kodaka,kimura,kasahara}@oscar.elec.waseda.ac.jp

Recently, chip multiprocessor architecture that contains multiple processors on a chip becomes popular approach even in commercial area. The authors have proposed OSCAR chip multiprocessor (OSCAR CMP) that is aimed at exploiting multiple grains of parallelism hierarchically from a sequential program on a chip. OSCAR CMP has local data memory (LDM) for processor private data and distributed shared memory having two ports for processor shared data to control data allocation by a compiler appropriately. This paper describes data localization scheme for OSCAR CMP which exploits data locality by assigning coarse grain tasks sharing same data on a same processor consecutively. In addition, OSCAR CMP using data localization scheme is compared with shared cache architecture and snooping cache architecture. Then, current naive code generation for OSCAR CMP is considered using evaluation results.

1　はじめに

集積度の向上に伴いチップ上のトランジスタを有効に利用して，スケーラブルな性能向上ができるアプローチとして，チップマルチプロセッサ (CMP) が近年注目を集め，さらに IBM の Power4[1] プロセッサのような商用化も始まっている．これらの CMP アーキテクチャでは，チップ上の限られたメモリ素子をいかに効率よく使うかが，これまでのマイクロプロセッサ以上に性能向上に向けて大きな課題となる．

メモリの有効利用に関する研究は，従来からキャッシュ最適化手法等で広く研究されてきた．たとえば，マルチプロセッサ用に複数のループリストラクチャリングを統合して行い，さらにデータローカリティの有効利用も図る Affine Partitioning[2)~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．また，シングルプロセッサ上で粗粒度タスク間のデータローカリティを利用したキャッシュ最適化手法 [6] も提案されている．さらに，演算器とローカルメモリを持つ処理要素である tile を多数チップ上に集積する MIT の Raw Architecture では，各 tile からのメモリアクセスの衝突を避けるコンパイル手法として，equivalence-class unification 及び module unrolling[7] 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に加え，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と協調動作する，OSCAR チップマルチプロセッサ (OSCAR CMP) を提案している [8]．この OSCAR CMP 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) を持つ．これらのメモリをコンパイラが適切に使用する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，プログラムの持つ並列性とデータローカリティの両方を最大限に活用する．

本稿では，OSCAR CMP 上でのスタティックスケジューリングを用いたデータローカライゼーション手法について述べる．ここで提案するデータロー

[†]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

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

[‡]早稲田大学理工学総合研究センター

カライゼーション手法は三つの処理から構成される．すなわち，(1) ローカルメモリの容量を考慮したループ整合分割，(2) 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うスタティックスケジューリング，(3) スケジューリング後のタスクに対する生死解析情報を用いた CSM–LDM 間データ転送挿入である．本稿では，特にスタティックスケジューリングアルゴリズムとデータ転送挿入アルゴリズムについて詳しく述べる．本手法を OSCAR Fortran マルチグレイン並列化コンパイラ [9] 上に実装し，OSCAR CMP 上で性能評価を行い，L2 共有キャッシュアーキテクチャと L2 スヌープキャッシュアーキテクチャとの比較を行った．

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 章では本手法の評価を行う OSCAR チップマルチプロセッサアーキテクチャ，第 4 章ではデータローカライゼーションを考慮したスケジューリングアルゴリズム，第 5 章ではスケジューリング後のコードに対する OSCAR CMP のメモリアーキテクチャを考慮したデータ転送挿入及びローカルメモリ配置の決定手法，第 6 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．

OSCAR マルチグレイン並列化コンパイラにおける粗粒度タスク並列処理の手順は次のようになる．

1. ソースプログラムを階層的に 3 種類の MT に分割
2. 各階層の MT 間のコントロールフロー，データ依存を解析しマクロフローグラフ (MFG) を生成
3. MFG 上の制御依存とデータ依存を考慮して MT 間の並列性を抽出する最早実行可能条件解析を行いマクロタスクグラフ (MTG) を生成
4. MTG がデータ依存エッジしか持たない場合は，MT はスタティックスケジューリングによって PG にコンパイル時に割り当てられる．一方，MTG が条件分岐などの実行時不確定性持つ場合は，コンパイラがユーザコード中に生成したダイナミックスケジューリングルーチンによって，MT を PG に実行時に割り当てる．

3 OSCARチップマルチプロセッサ

本稿で提案するデータローカライゼーション手法のターゲットアーキテクチャである OSCAR チップマルチプロセッサ (OSCAR CMP) アーキテクチャ (図 1) について説明する．

OSCAR CMP は CPU，データ転送ユニット (DTU)，ローカルプログラムメモリ (LPM)，ローカルデータメモリ (LDM) および分散共有メモリ (DSM) を持つプロセッサエレメント (PE) を複数個相互接続網 (バス結合，クロスバ結合など) で接続し 1 チップ上に搭載した構成となっている．今回の評価では，PE 間相互結合網として 3 本バスを利用している．

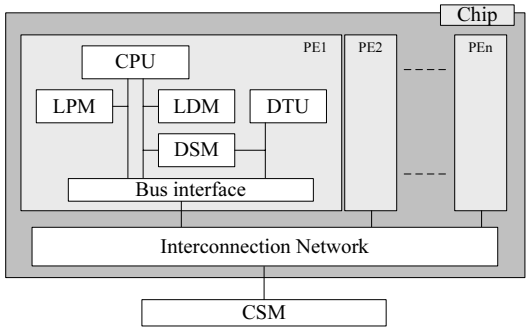

図 1: OSCAR CMP アーキテクチャ

4 データローカライゼーションを考慮したスケジューリングアルゴリズム

この章では配列の生死解析情報を利用した粗粒度タスクスタティックスケジューリングアルゴリズムについて述べる．本手法ではまず，同一の PG へ割り当てられた複数の MT を通して共有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，MT を分割する．次に配列の生死解析情報を用いたローカルメモリ上のデータのシミュレーションを使い，粗粒度タスクスタティックスケジューリングを行う．

4.1 マクロタスク分割

データを共有する複数の MT 間で効率の良いローカルメモリを介したデータの受け渡しを行うには，その共有データ量が 1PE のローカルメモリ容量を越えないように MT を分割する必要がある．そこで，MT 間のデータ共有量と並列性の両方を考慮する分割手法であるループ整合分割 (LAD) [10),11)] を利用して MT 分割を行う．分割後多くのデータ共有量を持ち同一の PG へ割り当てられる MT をデータローカライゼーショングループ (DLG) として定義する．

4.2 スケジューリングアルゴリズム

データを共有する MT を異なる PG へ割り当てた場合，PG 間でデータ転送が必要になる．データ転送を最小化するためには，ある PG_i に既に割り当てられた MT_j とデータ共有量の多い MT_k は並列性を損なわない範囲で同一の PG_i に割り当てる必要がある．そこで，PG と MT の組合わせ毎に次のようにデータ転送ゲインを定義し，PG 間データ転送の最小化を行う．ここで，PG_i と MT_j のデータ転送ゲイン $Gain_{ij}$ は， PG_i のローカルメモリ上のデータと MT_j とのデータ共有量として定義する．ただし，スケジューリング時のローカルメモリ上のデータのシミュレーションは配列の生死解析情報に基づき決定する．

以上の前提をもとに本手法で実装したスケジューリングアルゴリズムである，DLG を考慮したデータ転送ゲイン/CP/MISF スケジューリング法 [11)] は以下の通りとなる．ここで，DLG_MT とは DLG に属する MT とする．

まず，先行制約が満たされた DLG_MT がある場合，こちらを優先的にプロセッサに割り当てる．DLG_MT のスケジューリングプライオリティは以下のようになる．(1)PG に最後に割り当てられた

DLG_MT と同じ DLG に属する MT，(2)CP/MISF のプライオリティ．

次に DLG_MT 以外の MT のスケジューリングプライオリティは次のようになる．(1) データ転送ゲインが最大の PG と MT の組合わせ，(2) 複数の組合わせ候補があれば CP/MISF のプライオリティ．

5 OSCAR CMP のメモリアーキテクチャを考慮したデータ転送挿入及びローカルメモリ配置

本章では第 4 章で述べたスタティックスケジューリングアルゴリズムを適用した MTG 内の各 MT に対する，配列の生死解析情報に基づいたデータ転送の計算およびローカルメモリ配置について説明する．なお，今回の評価では本手法による DSM への配列の配置および DTU を使用した配列の転送は利用しない．そのため全てのプロセッサ間データ転送は CSM を介して行い，かつ CSM–LDM 間のデータ転送は CPU のロード/ストア命令によって処理される．さらに，現在本手法は粗粒度並列性のみを利用しているので，MT 割り当て単位である 1PG あたり 1PE とする．

データ転送計算の前準備として次の六つの配列の範囲情報すなわち配列領域を計算しておく．まず，MT_i と MT_j が同じ DLG に属するとき，MT_i が生産し，MT_j が消費する配列 A_p を st_inside_{ijp}, ld_inside_{jip} とそれぞれ定義する．また，MT_i と MT_k が同じ DLG に属さないとき，MT_i が生産し，MT_k が消費する配列 A_p を $st\_outside_{ikp}$, $ld\_outside_{kip}$ とそれぞれ定義する．さらにデータローカライゼーション適用 MTG の外側から生きて入り，MT_i で消費される配列 A_p を ld_outer_{ip}，MT_i で生産され，適用 MTG の外側へ生きて出る配列 A_p を st_outer_{ip} とそれぞれ定義する．

次に，計算された配列領域及び DLG 内の MT での定義・参照回数から，次のようにローカル化の対象から除く配列を決定する．すなわち，各 DLG に生きて入るが DLG 内でたかだか一回しか使用されず，なおかつ各 DLG から生きて出るがたかだか一回しか定義されない配列は，ローカル化配列の対象としない．これらの配列をローカル化対象配列として選択すると CSM に直接アクセスする場合と比較し，一旦 LDM へストアする分だけ余計に時間がかかってしまうからである．これらのローカル化の対象外となった配列を先に計算した ld_inside, st_inside, ld_outside，st_outside，ld_outer および st_outer から削除する．

次にローカルメモリ配置について述べる．LDM サイズを ldm_size，DLG 内でアクセスされる各ローカル化配列の最大サイズの合計を dlg_size とすると $\lfloor ldm\_size/dlg\_size \rfloor$ がある瞬間に同時に配置可能な DLG の個数となる．さらに dlg_size に分割された LDM の領域の内部を各配列の最大サイズで分割し使用する．本手法では各 PE には複数の DLG が割り当てられるので，これらの分割された LDM 上の領域をスケジューリング結果に従い，各 DLG 内のローカル化配列によって使い回すことになる．スケジューリング結果によっては，ある DLG に属する MT_i の割り当て後に同一 DLG の後続 MT を割り当てることができず，他の DLG に属する MT_j が割り当てられることもある．この場合，LDM に空きがなければ CSM への一時的なデータの待避が必要となるが，DLG 内でアクセスされる全ローカル化配列を待避させるのではなく，MT_j に必要なローカル化配列領域を確保するために必要なデータのみを待避させることで不要なデータ転送を生成しない．

さて，前述のスタティックスケジューリングの結果に基づき，実際に必要なデータ転送を計算していく．ここで，配列 A_p を分割して，DLG_l 用のローカル化配列としたものを A_{pl} として表す．DLG_l に属する MT_i のローカル化配列 A_{pl} に関するデータ転送について以下のように場合分けを行い計算する．

まず，DLG_l 用のローカル化配列 A_{pl} の領域が既に PE 内の LDM に割り当てられている場合，すなわち MT_i に先行する MT によって A_{pl} の一部もしくは全てをロードされている場合，DLG_l に属する MT_i に必要となるロードは，$ld\_outside_{ijp}$ (MT_j は任意の先行 MT)，ld_outer_{ip} のうち，先行 MT によってまだロードされていない配列領域および ld_inside_{ikp}(MT_k は DLG_l に属するが，MT_i の実行前に一旦 CSM へ待避される先行 MT) である．一方ストアは $st\_outside_{ijp}$ (MT_j は任意の後続 MT) および st_outer_{ip} となる．

次に LDM に DLG_l 用のローカル化配列領域 A_{pl} の領域が割り当てられていないとき，DLG_l に属する MT_i に必要となるデータ転送は，LDM に A_{pl} 用の領域が空いているか否かで二つに分けられる．LDM に A_{pl} 用の領域が空いている場合，MT_i に必要となるロードは $ld\_outside_{ijp}$ (MT_j は任意の先行 MT)，ld_outer_{ip} および ld_inside_{ijp} (MT_j は任意の先行 MT) となる．一方ストアは $st\_outside_{ijp}$ (MT_j は任意の後続 MT) および st_outer_{ip} となる．

次に LDM に A_{pl} 用の領域が空いていない場合は LDM 上に既に割り当てられている他の DLG_k のローカル化配列 A_{pk} を LDM から CSM へ待避する必要がある．LDM から CSM へ待避することになった DLG_k に属する MT_m に必要なストアは st_inside_{mnp} (MT_n は DLG_k に属する MT_m の後続 MT) となる．次に MT_i に必要なロードは $ld\_outside_{ijp}$ (MT_j は任意の先行 MT)，ld_outer_{ip} および ld_inside_{ijp} (MT_j は任意の先行 MT) となる．一方ストアは $st\_outside_{ijp}$ (MT_j は任意の後続 MT) および st_outer_{ip} となる．

以上のようにコンパイラが配列の定義・参照関係および生死解析情報を元に，適切なデータ転送を挿入することでデータのコヒーレンスを保つ．最後にローカル化配列をリネーミングすることでデータローカライゼーションを実現している．

6 性能評価

本章では本論文で提案したスケジューリングアルゴリズムおよびデータローカライゼーションの性能評価について述べる．

6.1 評価環境

性能評価に用いた OSCAR CMP と L2 共有キャッシュおよび L2 スヌープキャッシュアーキテクチャについて述べる．OSCAR CMP には本手法を，L2 共有キャッシュおよび L2 スヌープキャッシュアーキテクチャには配列間のキャッシュコンフリクトを最小化するキャッシュ最適化である padding[12] をそれぞれ適用した．さらに使用したベンチマークアプリケーションについて述べる．

性能評価では同一の構成のプロセッサコアを用い，ネットワークおよびメモリアーキテクチャを変更し

評価を行った．OSCAR CMP のネットワークおよびメモリアーキテクチャについては第 3 章の通りである．また，L2 共有キャッシュモデルのアーキテクチャを図 2 に，L2 スヌープキャッシュモデルのアーキテクチャを図 3 にそれぞれ示す．

今回の評価ではプログラムの実行開始時に配列を全て CSM に配置し，OSCAR CMP では 5 章で述べたコンパイル時のデータ転送命令に従い，LDM と CSM 間でロード・ストアを行う．また，各 DLG 中の MT によってアクセスされるデータ量が十分 LDM に入るようにコンパイル時にループ整合分割されている．評価に用いた OSCAR CMP の各メモリのパラメータを表 1 に，L2 共有キャッシュモデルの各キャッシュのパラメータを表 2 に，L2 スヌープキャッシュモデルの各キャッシュのパラメータを表 3 にそれぞれ示す．

計測にはクロックレベルの精密なシミュレータを用いる．評価ベンチマークにはシミュレーション時間短縮のため，SPEC 95fp から配列サイズを ref のデータセットである 513 から 65 に，収束ループの回転数を 750 から 200 にそれぞれ縮小した Tomcatv および配列サイズを ref のデータセットである 513 から 257 に，収束ループの回転数を 900 から 4 にそれぞれ縮小した Swim を用いる．このとき，Tomcatv のプログラム中の全データサイズは約 231KB，Swim は約 3.3MB となる．また，本評価では全てのループ回転数を定数として評価を行った．

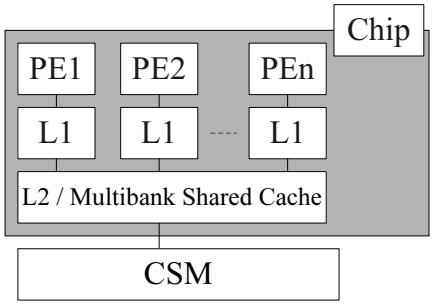

図 2: L2 共有キャッシュアーキテクチャ

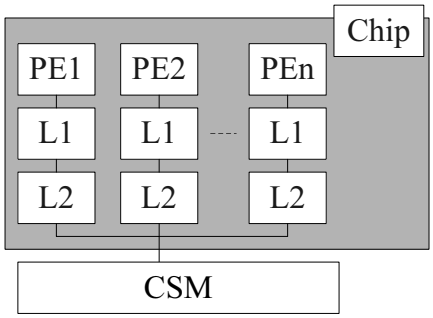

図 3: L2 スヌープキャッシュアーキテクチャ

表 1: OSCAR CMP のメモリパラメータ

LDM	サイズ: 256KB レイテンシ: 1, 2, 4 clocks
DSM	レイテンシ (ローカル): 1, 2, 4 clocks レイテンシ (リモート): 4, 8, 16 clocks
CSM	レイテンシ: 20, 40, 80 clocks バス幅: 64bit

表 2: L2 共有キャッシュモデルのキャッシュパラメータ

L1 データキャッシュ	サイズ: 16KB ラインサイズ: 32B 4-way, ライトスルー レイテンシ: 1, 2, 4 clocks
L2 キャッシュ	サイズ: 1MB(1-4pe), 2MB(8pe) ラインサイズ: 64B 4-way, ライトバック レイテンシ: 5, 11, 23 clocks
CSM	レイテンシ: 20, 40, 80 clocks バス幅: 64bit
L1 キャッシュ無効化に要するレイテンシ	2, 5, 11 clocks

表 3: L2 スヌープキャッシュモデルのキャッシュパラメータ

L1 データキャッシュ	サイズ: 16KB ラインサイズ: 32B 4-way, ライトスルー レイテンシ: 1, 2, 4 clocks
L2 キャッシュ	サイズ: 256KB ラインサイズ: 64B 4-way, ライトバック レイテンシ: 5, 11, 23 clocks
CSM	レイテンシ: 20, 40, 80 clocks バス幅: 64bit
キャッシュコヒーレンス制御に要するレイテンシ	2, 5, 11 clocks
L2 キャッシュ間データ転送に要するレイテンシ	2, 5, 11 clocks

6.2 性能評価結果

Tomcatv の CSM レイテンシが 20 の時の性能評価結果を図 4，40 を図 5，80 を図 6 にそれぞれ示す．また，Swim の CSM レイテンシが 20 の時の結果を図 8，40 を図 9，80 を図 10 にそれぞれ示す．図中の縦軸は 1PE 時の OSCAR CMP の実行クロック数を 1 としたときの速度向上率を，横軸はプロセッサ数をそれぞれ表す．図中の oscar は本手法を適用した OSCAR CMP の速度向上率を，shared と snoop は padding を適用した L2 共有キャッシュモデルと L2 スヌープキャッシュモデルの速度向上率をそれぞれ表す．

図 4，5，6 において Tomcatv はループ並列性，データローカリティともに高く，OSCAR CMP には本手法を，L2 共有キャッシュモデルおよび L2 スヌープキャッシュモデルには padding を適用しているので，4PE までは全てスケーラブルな性能向上を示している．8PE 時，OSCAR CMP と L2 スヌープキャッシュモデルはスケーラブルな性能向上を示すのに対し，L2 共有キャッシュモデルは著しく性能が低下している．これは L2 共有キャッシュモデルではラインコンフリクトにより L2 キャッシュミスが増加したためである．一方，8PE 時 L2 スヌープキャッシュモデルでは L2 キャッシュミスは全て初期参照ミスによって引き起こされている．これは一旦データをキャッシュに載せた後，ローカリティを最大限抽出できていることを表す．次に OSCAR CMP と L2 スヌープキャッシュモデルの結果を比較するとほぼ同様か若干 OSCAR CMP が低い性能を示すことが分かる．この原因としては OSCAR CMP ではデータ転送の際にバースト転送を行っていない点と配列の生死解析が不十分なために不要なデータ転送を行っている点が挙げられる．一方 L2 スヌープキャッシュモデルでは L2 のラインサイズである 64Byte 単位でのバースト転送を行っている．データ

転送回数と CSM のレイテンシから簡単なモデルを作成し，バースト転送により実現され得る推定速度向上率を図 7 に示す．図中縦軸は各レイテンシ毎に 1PE 時の OSCAR CMP の実行クロック数を 1 としたときの速度向上率を，横軸はプロセッサ数をそれぞれ表す．oscar_est はバースト転送により得られる OSCAR CMP の推定速度向上率を表す．図より OSCAR CMP 上でバースト転送を実現すれば，多くの場合スヌープキャッシュモデルとほぼ同等かそれ以上の性能の達成が予想できる．また，手動で不要なデータ転送を除去した結果更なる速度向上が得られたので，配列の生死解析を強化し，不要なデータ転送を除去することで更なる性能向上が得られることが予想できる．

Tomcatv と同様に Swim もまたループ並列性，データローカリティの高いプログラムである．図 8，9，10 より，L2 スヌープキャッシュモデルはスケーラブルな性能向上を示し，OSCAR CMP はそれに比べ低い性能を示し，L2 共有キャッシュモデルは 8PE 時性能低下を示していることが分かる．この原因として OSCAR CMP ではバースト転送を行っていない，コード生成系の実装上の不備によりバス幅の 64bit ではなく 32bit での転送を行っている，また配列の生死解析が不十分なために不要なデータ転送を行っているという三点が挙げられる．特に 8PE 時の差が大きいのは OSCAR CMP では PE 間相互結合網として 3 本バスを利用しているので，不要なデータ転送によりバスのコンフリクトの頻度が高まっていることが原因と考えられる．また，L2 共有キャッシュモデルではラインコンフリクトによる L2 キャッシュミスが増加し，性能が著しく低下している．以上を踏まえ，データ転送回数と CSM のレイテンシから簡単なモデルを作成し，64bit のバス幅をフルに使ったバースト転送により実現され得る推定速度向上率を図 11 に示す．図中 oscar_est はバースト転送により得られる OSCAR CMP の推定速度向上率を表す．図より 64bit のバースト転送によっても 8PE 時 L2 スヌープキャッシュモデルに大きく差を開けられていることが分かる．そこで，不要なデータ転送除去のために配列の生死解析を強化する必要があることが分かる．

以上を踏まえた上でローカルメモリモデルとキャッシュモデルの有利不利について考察してみる．ローカルメモリモデルの有利な点は明示的にデータ転送の制御が可能，ストライドアクセスに対し，容量を有効に活用できる，生死解析情報の利用による不要なデータ転送の除去，一定のアクセスレイテンシの保証が挙げられる．一方不利な点は明示的な制御のためにコンパイラによるより強力な解析が必要な点が挙げられる．次にキャッシュモデルの有利な点は明示的な制御を必要としない，padding 等のキャッシュ最適化と組み合わせることで容量全域にわたって有効に利用できる点が挙げられる．不利な点としては一定のアクセスレイテンシを保証できない，死んでいる変数に対しても本来不要な変数の書き戻しが発生する点が挙げられる．

7 まとめ

本稿では，OSCAR CMP 上でのスタティックスケジューリングを用いたデータローカライゼーション手法について述べた．本手法を OSCAR Fortran マルチグレイン並列化コンパイラ上に実装し，OSCAR CMP 上で性能評価を行った．その結果 SPEC

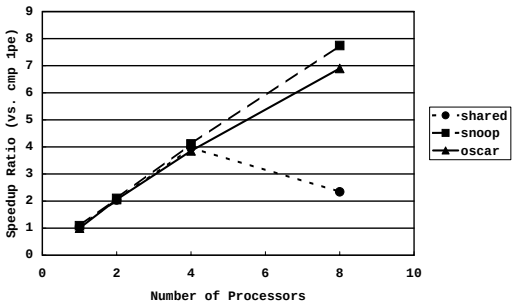

図 4: CSM アクセスレイテンシ 20 の時の Tomcatv の速度向上率

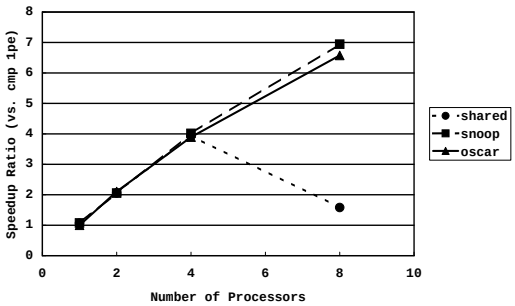

図 5: CSM アクセスレイテンシ 40 の時の Tomcatv の速度向上率

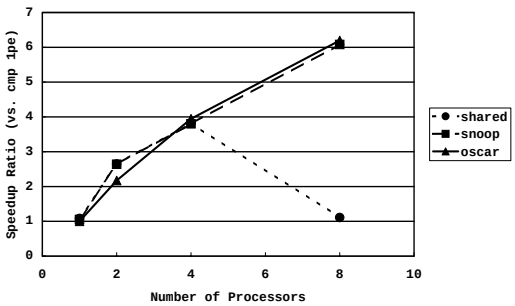

図 6: CSM アクセスレイテンシ 80 の時の Tomcatv の速度向上率

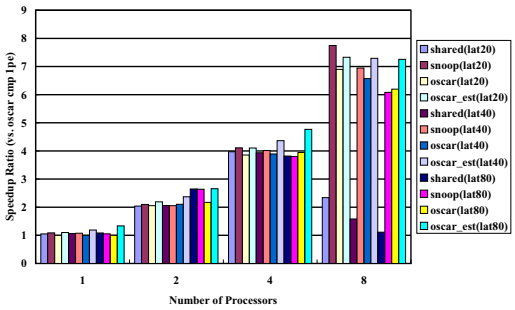

図 7: バースト転送を考慮した時の Tomcatv の推定速度向上率

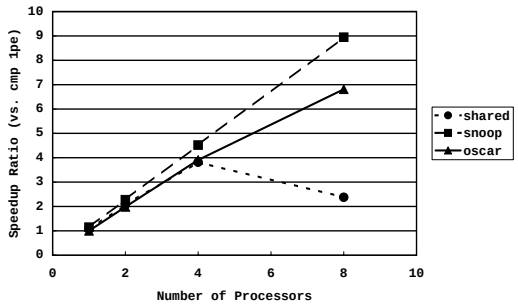

図 8: CSM アクセスレイテンシ 20 の時の Swim の速度向上率

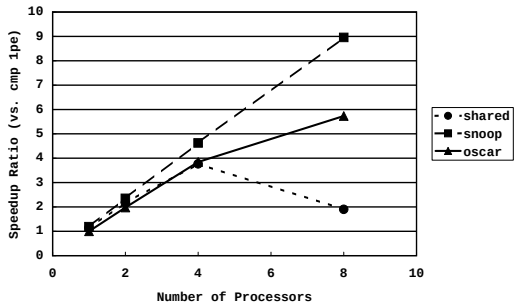

図 9: CSM アクセスレイテンシ 40 の時の Swim の速度向上率

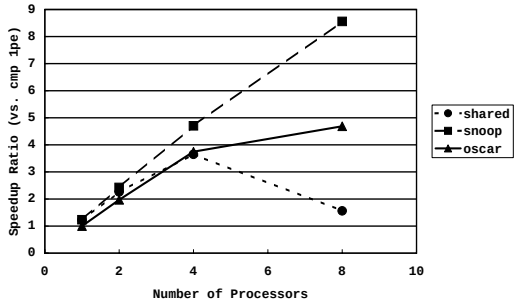

図 10: CSM アクセスレイテンシ 80 の時の Swim の速度向上率

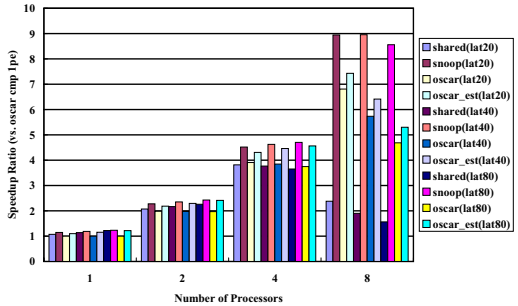

図 11: バースト転送を考慮したときの Swim の推定速度向上率

95fp の Tomcatv においてキャッシュ最適化である padding を適用した L2 スヌープキャッシュモデルと比較し，ほぼ同等の性能向上が得られることを確認した．また，いくつかの改善を行うことで更なる向上の見込みが得られた．一方で SPEC 95fp の Swim において L2 スヌープキャッシュモデルよりも低い性能を示した．そこで本論文中でいくつかの改善案を示し，今後それらの改善案の評価を行っていく方針である．

8 謝辞

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「自動並列化協調型チップマルチプロセッサ」により行われた．本論文作成に当たり有益なコメントを頂いた，宮田操氏 (STARC)，高橋宏政氏 (富士通研)，倉田隆弘氏 (ソニー)，高山秀一氏 (松下)，安川秀樹氏 (東芝) に感謝いたします．

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] Lim, A. W., Cheong, G. I. and Lam, M. S.: An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication, *Proc. 13th ACM SIGARCH International Conference on Supercomputing* (1999).

[3] Lim, A. W., Liao, S. and Lam, M. S.: Blocking and Array Contraction Across Arbitrarily Nested Loops Using Affine Partitioning, *Proc. of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2001).

[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] 稲石, 木村, 藤本, 尾形, 岡本, 笠原: 最早実行可能条件解析を用いたキャッシュ利用の最適 化, 情報処理学会研究報告 ARC (1998).

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

[8] 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).

[9] 笠原: 並列処理技術, コロナ社 (1991).

[10] 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).

[11] 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).

[12] 石坂, 中野, 小幡, 笠原: ラインコンフリクトミスを考慮した粗粒度タスク間キャッシュ最適化, 情報処理学会研究報告 ARC (2002).