4R-02　マルチプロセッサシステム上でのキャッシュ最適化を考慮した粗粒度タスクスタティックスケジューリング手法

中野啓史 †，石坂一久 †，小幡元樹 †，木村啓二 †，笠原博徳 † ‡
† 早稲田大学理工学部電気電子情報工学科，‡ アドバンスト並列化コンパイラプロジェクト

1　はじめに

キャッシュの有効利用はシステムの性能向上に大きく貢献するため，キャッシュ最適化に関する研究は従来から広く行われている．コンパイラによるキャッシュ最適化としては，キャッシュプリフェッチング [9] のようにアクセスが予測されるデータを予めキャッシュに読み込んでおく方法や，ループフュージョン，ループインターチェンジ，ブロッキング等，ループリストラクチャリング [6, 7] によりローカリティを高める手法及び，これらの手法を統合したアフィンパーティショニング [8] が提案されている．しかし，これらはそれぞれのループ内，あるいは隣接するループ間での局所的な最適化である．本稿では OSCAR Fortran マルチグレイン並列化コンパイラ [1, 3] における粗粒度並列処理手法（マクロデータフロー処理）を利用し，データ共有量の多いタスクをなるべく同一プロセッサに割り当てると共に実行順序の最適化を行い，キャッシュの有効利用を図るマルチプロセッサシステム上での粗粒度タスクスタティックスケジューリング手法を提案する．

2　粗粒度タスク並列処理

粗粒度タスク並列処理ではプログラムをサブルーチン、ループ、基本ブロックの三種類のマクロタスク (MT) [1, 2, 3] に分割し，それらの MT 間の並列性を利用する．

OSCAR Fortran マルチグレイン並列化コンパイラ [1, 3] では MT として次の三種類の MT を生成する．

- BPA (Block of Pseudo Assignment statements):
  基本ブロック，もしくは複数の小基本ブロックを融合したブロックあるいは並列性を考慮して分割されたブロック
- RB (Repetition Block):最外側ナチュラルループ
- SB (Subroutine Block):サブルーチンコール

分割された MT から制御フロー解析，データ依存解析によりマクロフローグラフ (MFG) が生成される．この MFG を元に最早実行可能条件解析が行われ，MT 間の並列性を抽出したマクロタスクグラフ (MTG) を得る．本論文で提案する手法は，この MTG を用いて，MT をコンパイル時にプロセッサクラスタ (PC: PE を論理的にグループ化したもの) に割り当てるスタティックスケジューリングを行う．

3　キャッシュ最適化を考慮したスケジューリングアルゴリズム

本稿で提案するキャッシュ最適化手法はシングルプロセッサ用のキャッシュ最適化 [5] をマルチプロセッサ用に拡張したものである．

ある MT_i の実行終了時にはその MT_i 内で定義・参照されたデータがキャッシュ上に存在する可能性が高い．そこで，その MT_i と共有データ量 (同じデータを定義・参照する量) の多い MT_j を MT_i の直後に同一 PC で実行すればキャッシュの有効利用が図れる．

本稿で提案するアルゴリズムは 2 つの手順から構成される．まず 3.1 で説明するマクロタスク分割を行い，続いて 3.2 で説明するアルゴリズムによりスケジューリングを行う．

3.1　マクロタスク分割

MT がキャッシュ容量を越えるデータを定義・参照する場合，最初にアクセスしたデータがキャッシュから追い出されてしまい，キャッシュの有効利用が難しくなる．そのため，キャッシュ容量を考慮した MT の分割が必要となる．この様子を図 1 を用いて説明する．

複数の MT がキャッシュ容量以上の共有データをアクセスする場合を考える．図 1 で網線部はキャッシュ容量を示し，破線はキャッシュ上に存在するデータの遷移を示す．MT_i と MT_j は配列 A, B を共有するループであり，図 1 では MT_i と MT_j が定義・参照するデータ量全体はキャッシュ容量を越えている．この時 MT_i と MT_j を連続して同一 PC に割り当てたとしても，MT_i がアクセスするデータ量がキャッシュ容量を越えているため，MT_j でアクセスされる共有データが MT_i の終了時にキャッシュ上に残っている保証はない (図 1(a))．さらに共有データの一部が残っていたとしても，MT_j の実行においてそのデータをアクセスする前に追い出してしまう可能性もある．このような場合には，キャッシュ容量以上のデータをアクセスする MT をキャッシュ容量以内のデータをアクセスする複数の MT にループイタレーション方向で分割した後，後述のようなデータ共有を考慮したスケジューリングを行えば，キャッシュヒット率の向上が期待できる．そこで，図 1(b) のように MT_i と MT_j を，分割後の MT でアクセスするデータ量がキャッシュに収まるように分割 ($MT_i{}^1 \sim MT_i{}^n$, $MT_j{}^1 \sim MT_j{}^n$) し，分割後の MTG を用いて $MT_i{}^1$ と $MT_j{}^1$, $MT_i{}^2$ と $MT_j{}^2$,..., $MT_i{}^n$ と $MT_j{}^n$ がそれぞれ同一 PC に連続して割り当てられるようにすれば，異なるループ間でキャッシュを効率的に利用することができる．

この MT の分割には，MT 間のデータ共有量と並列性の両方を考慮した分割手法であるループ整合分割 [4] を利用する．

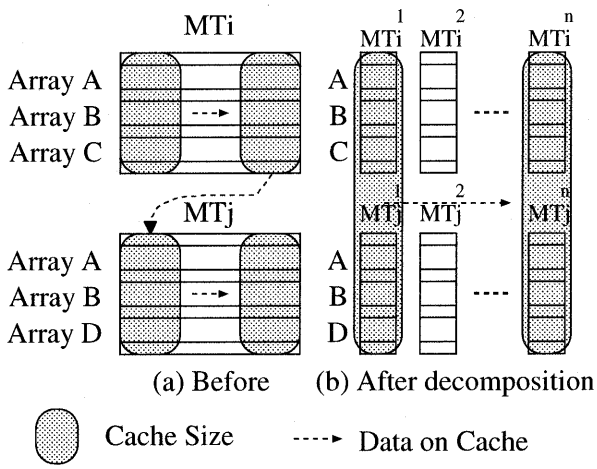

図 1: ループ整合分割

3.2　スケジューリングアルゴリズム

ループ整合分割され，同一プロセッサに割り当てられるべき MT を図 2 のようにデータローカライゼーショングループ

---

* A Static Scheduling Method for Coarse Grain Tasks considering Cache Optimization on Multiprocessor Systems
By Hirofumi Nakano †, Kazuhisa Ishizaka †, Motoki Obata †, Keiji Kimura †, Hironori Kasahara † ‡
† Department of EECE, Waseda University,
‡ Advanced Parallelizing Compiler Project

(DLG) へとグルーピングする．ここで，DLG_i に属する MT_j を $DLG_i\_MT_j$ と呼ぶこととする．スケジューリングアルゴリズムにおける前提条件として，同一の DLG_i に属する MT は全て同一の PC に割り当てることとする．

以上の前提をもとにして，本論文で提案するスケジューリングアルゴリズムは次のように説明できる．

PC_n に最後に割り当てられた $DLG_i\_MT_j$ と最大のデータ共有量を持つ $DLG_i\_MT_k$ を PC_n に割り当てる．データ共有量が同じ MT が複数ある場合は，CP/MISF [3] のプライオリティに従いタスクを割り当てる．これらの処理を未割り当てのタスクがなくなるまで繰り返す．

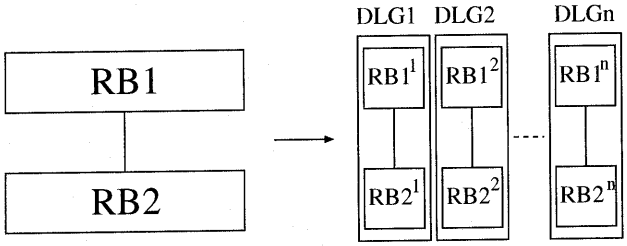

図 2: データローカライゼーショングループ (DLG)

4 性能評価

次に提案手法の性能を OSCAR Fortran マルチグレイン並列化コンパイラに実装し評価を行った結果について述べる．本評価では，OSCAR Fortran コンパイラを逐次 Fortran プログラムを入力とし，OpenMP Fortran プログラムを出力するプリプロセッサとして利用した．生成された OpenMP Fortran プログラムはターゲットアーキテクチャのネイティブコンパイラにより実行バイナリにコンパイルされる．

性能評価プログラムとしては対称帯状係数行列をもつ連立方程式の繰り返し求解法である CG 法 (Conjugate Gradient Method) プログラムと SPECfp95 ベンチマークの 1 つである swim を用いた．オリジナルプログラムとキャッシュ最適化スケジューリングしたプログラムをそれぞれコンパイルし，その実行時間を比較した．コンパイルおよび実行は Sun Microsystems の Ultra80(UltraSparc-II, 4 プロセッサからなる SMP, L1cache16KB+16KB, L2cache4MB) 上で行った．OS は Solaris7，コンパイラは Forte[tm] Fortran 6 update 1，コンパイルオプションはシーケンシャル実行するものには"-fast"を，マルチプロセッサで実行するものでオリジナルのものには"-fast -parallel -reduction"を，マルチプロセッサで動作させるものでキャッシュ最適化を施したものには"-fast -explicitpar -mp=openmp"をそれぞれ用いた．

CG 法プログラムを用いた性能評価結果を表 1 に，swim を用いた性能評価結果を表 2 にそれぞれ示す．表中，速度向上率とは (オリジナルプログラムの 1 プロセッサ上での実行時間/キャッシュ最適化を適用したプログラムの実行時間) である．表 1 の PE 数 1 の場合に効率が落ちているのは，MT 分割を行った際，今回の評価ではコンパイラへの実装を容易にするため，コードコピーを行っており，コードサイズが増えたためと考えられる．表 1 において PE 数 4 の場合，オリジナルプログラムの実行時間が 8.0 秒であるのに対し，最適化後のプログラムの実行時間が 5.5 秒となり，1.45 倍の速度向上を得た．表 2 において PE 数 4 の場合，オリジナルプログラムの実行時間が 60.6 秒であるのに対し，最適化後のプログラムの実行時間は 19.2 秒となり，3.16 倍の速度向上を得た．最適化によって速度向上が得られたことが分かる．また，表 2 では PE 数 3，4 の場合に最適化後の実行時間はそれぞれ 30.2 秒，19.2 秒となり，オリジナルプログラムのシーケンシャル実行時間である 99.8 秒に比べ 3.30 倍，5.20 倍の速度向上を得ており，スーパーリニアとなっていることが分かる．

表 1: cg 法の実行時間と速度向上率

PE 数	実行時間 [秒]	
	オリジナル (速度向上率)	最適化後 (速度向上率)
1	14.5(1.00)	14.8(0.98)
2	9.3(1.56)	9.1(1.59)
3	8.4(1.73)	7.5(1.93)
4	8.0(1.81)	5.5(2.64)

表 2: swim の実行時間と速度向上率

PE 数	実行時間 [秒]	
	オリジナル (速度向上率)	最適化後 (速度向上率)
1	99.8(1.00)	93.4(1.07)
2	66.4(1.50)	57.9(1.72)
3	62.7(1.59)	30.2(3.30)
4	60.6(1.65)	19.2(5.20)

5 まとめ

本稿では，マルチプロセッサシステム上でのキャッシュ最適化を考慮した粗粒度タスクスタティックスケジューリング手法を提案した．本手法は OSCAR マルチグレイン並列化コンパイラを用いて，Fortran プログラムを入力とし，最適化された OpenMP Fortran プログラムを生成するプリプロセッサとして実現されている．性能評価の結果，本手法を適用したプログラムを 4PE で実行した時，シーケンシャル実行時間に対し，CG 法プログラムで 2.64 倍，swim で 5.20 倍の速度向上が得られ，本手法の有効性が確認された．

今後の課題としてはより多くのプログラムでの評価，及び最適な分割数の決定法が挙げられる．

本研究の一部は NEDO アドバンスト並列化コンパイラプロジェクトにより行われた．

参考文献

[1] H.Kasahara, M.Obata, and K.Ishizaka "Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP", Proc. 12th Workshop on Languages and Compilers for Parallel Computing(2000).

[2] 本多, 岩田, 笠原, "Fortran プログラム粗粒度タスク間の並列性の検出手法", 信学論, J73-D-I(12), Dec. 1991.

[3] 笠原 博徳, "並列処理技術", コロナ社, Jun. 1991.

[4] H.Kasahara, A.Yoshida, "A Data-Localization Compilation Scheme Using Partial-Static Task Assignment", Journal of Parallel Computing ,Vol.24,No.3,pp579-596, May.1998

[5] 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最適化手法", 情報処理学会第 58 回全国大会, 3H-07, Mar., 1999.

[6] M.S.Lam, E.E.Rothberg, M.E.Wolf, "The Cache Performance and Optimizations of Blocked Algorithms", ASPLOS IV, Apr.9-11, 1991

[7] M.Wolfe, "High Performance Compilers for Parallel Computing", The Addison-Wesley Publishing Company, 1996

[8] A. W. Lim, G. I. Cheong and M. S. Lam, "An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication", Proceedings of the 13th ACM SIGARCH International Conference on Supercomputing, June, 1999.

[9] T.C.Mowry, M.S.Lam, A.Gupta, "Design and Evaluation of a Computer Algorithm for Prefetching", ASPLOS V, Oct.1992