2E－6

マクロタスク最早実行可能条件解析を用いたキャッシュ最適化手法

稲石 大祐†, 木村 啓二†, 尾形 航†, 岡本 雅巳‡, 笠原 博徳†
†早稲田大学理工学部電子電気情報工学科, ‡（株）東芝

1 はじめに

単一プロセッサのキャッシュの利用効率向上のために、現在様々な手法が研究されている。コンパイラによる代表的なキャッシュ最適化としては、ブロッキング [4] 等のループリストラクチャリングが挙げられる。これらのリストラクチャリングは各々のループ内に対して行なわれるため局所的なものが多く、複数のループに跨るもしくはプログラム全域に対する最適化はほとんど行なわれていない。

本稿では、OSCAR Fortran マルチグレイン並列化コンパイラ [1, 3] におけるループ・サブルーチン・ベーシックブロック等の粗粒度タスク間の並列性抽出手法（最早実行可能条件解析）を利用した、Fortran プログラムの単一プロセッサキャッシュ最適化手法を提案する。OSCAR Fortran マルチグレイン並列化コンパイラでは Fortran プログラムを RB(ループ)・SB(サブルーチン)・BPA(基本ブロック) の 3 種のマクロタスク (MT) に分割し、各 MT の最早実行可能条件解析からマクロタスクグラフ (MTG) を生成する。この MTG は制御依存・データ依存に基づく MT 間の先行実行順序制約、及び MT 間で授受されるデータに関する情報を提供する。本手法ではこの MT 間制御依存・データ依存情報を用い、キャッシュヒット率を高めるために、先行 MT のデータにアクセスする MT が直後に実行されるよう MT レベルでの大域的なコード移動を行なう。本稿ではこの手法を「最早実行可能条件解析を用いたキャッシュ最適化手法」と呼ぶ。

2 最早実行可能条件解析 [2,3]

OSCAR Fortran マルチグレイン並列化コンパイラにおける粗粒度並列処理手法（マクロデータフロー処理）では、Fortran プログラムは次に示す 3 種類の MT に階層的に分割される。

- BPA (Block of Pseudo Assignment statements): 基本ブロック、および複数の小基本ブロックを融合/分割したブロック
- RB (Repetition Block):最外側ナチュラルループ
- SB (Subroutine Block):サブルーチン

分割された MT 間の制御フロー解析、データ依存解析により各階層でのマクロフローグラフ (MFG) が生成される。しかし MFG は MT 間の制御フローとデータ依存を表したものにすぎず、その並列性（最早実行可能条件）を示すものではない。従って MT 間の並列性を抽出するには、制御依存とデータ依存を同時に解析する必要がある。そこで、制御依存とデータ依存を考慮した MT 間の最大の並列性を表すものとして、各 MT の最早実行可能条件を用いる。マクロタスク i(MTi) の最早実行可能条件とは、MTi が最も早い時点で実行可能となるための条件である。ただし、このマクロデータフロー処理における実行可能条件は、次のような実行条件を仮定して求められる。

1) マクロタスク i(MTi) がマクロタスク j(MTj) にデータ依存するならば、MTj の実行が終了するまでは MTi は実行開始出来ない。

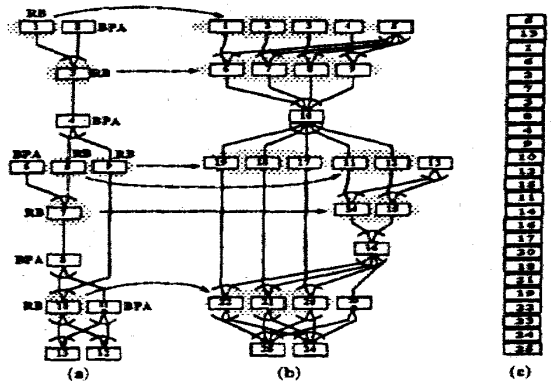

図 1: CG 法プログラムのマクロタスクグラフ

2) MTj の条件分岐先が確定すれば、MTj の実行が終了しなくても、MTj に制御依存だけしている MTi は実行を開始することが出来る。

MTi の最早実行可能条件の一般形は次の通りである。

[(MTi が制御依存する MTj が MTi に分岐する)
AND
(MTi がデータ依存する全てのマクロタスク MTk($0 \leq k < |N|$) の実行が終了する
OR
MTk が実行されないことが確定する)]

各 MT の最早実行可能条件を求め、さらに冗長な条件を削除すると、図 1(a) に示すようなマクロタスクグラフ (MTG) と呼ばれる無サイクル有向グラフが生成される。MTG において各ノードは MT を表す。実線のエッジはデータ依存を表す。MTG 中のノード内の小円を起点とするデータ依存エッジつまり実線のエッジは、制御依存とデータ依存の 2 つを同時に表している。図 1(a) 中のエッジを束ねている実線のアークは、そのアークによって束ねられたエッジが互いに AND の関係にあることを示す。点線のアークは、そのアークで束ねられたエッジが互いに OR の関係にあることを示す。ノードの内の小円は、MFG と同様条件分岐を表している。このマクロタスクグラフにおいてもエッジの向きは下向きと仮定しており、ほとんどの矢印は省略されている．矢印がついているエッジは、元の MFG 上での分岐方向を表すエッジである。

3 コード移動によるキャッシュ最適化

本章では、前章で解説した最早実行可能条件を用いて、コードの大域的移動を行ない、キャッシュ利用を最適化する手法について述べる。

3.1 大域的なコード移動

本節ではコードを大域的に移動させる方法について述べる。

前述の最早実行可能条件解析を行なうと、オリジナルプログラム中では後に位置するような MT であっても最早実行可能条件さえ満たされていれば、プログラムのどの位置で実行されようともプログラムの意味は正しく保たれる。例えば図 1(a) において MT9 はオリジナルプログラム中では MT8 の次に実行される (MT 番号はもとのプログラムでの出現順に相当する) はずであるが、MT9 の最早実行可能条件は MT4 の終了 (データ依存) のみであるため、MT4 の直後に MT9 のコー

---

* A Cache Optimization with Macto-Task Earliest Execution Condition
Daisuke INAISHI†, Keiji KIMURA†, Wataru OGATA†
Masami OKAMOTO‡, Hironori KASAHARA†
† Department of Electrical, Electronics and Computer Engineering, Waseda University
‡ TOSHIBA Corporation

ドを移動させることが可能である。つまり最早実行可能条件を用いた大域的なコード移動は、最早実行可能条件が満たされる範囲内でMTを任意に並べ替えることで実現される。

3.2 キャッシュ最適化

本節ではコード移動によるキャッシュ最適化手法を述べる。

通常あるマクロタスクMTiの実行が終了した時点では、MTiが定義参照したデータがキャッシュ上に存在する確率は極めて高い。従ってMTiで定義参照したデータを再び定義参照するようなMTをMTiの直後に移動させれば、キャッシュの効率は向上する。このことを常に考慮してMTの並べ替え(コードの移動)を行なうことにより、従来なし得なかった大域的なキャッシュ最適化を実現する。

MTiの直後に移動すべきMTを決定する手続きを図2に示す。ここで共通データとはMT間で共通に定義参照されるデータ、レディMT集合とはMTiが終了した時点で実行可能となる(最早実行可能条件を満たした)MTの集合を表す。この手続きをレディMT集合が空になるまで繰り返すことにより、大域的なコード移動によるキャッシュ最適化を実現する。なお、図2における2)、3)の条件は次のMT選択で選択の幅が広がるようにするためのものである。

1) MTiとの共通データ量が最大のMTをレディMT集合から選出する
2) 1の条件を満たすMTが複数ある場合その中から、そのMTの終了により実行可能となる可能性があるMTの数が最大のMTを選出する
3) 2の条件を満たすMTが複数ある場合その中から、そのMTの終了により実際に実行可能となるMTの数が最大のMTを選出する
4) 3の条件を満たすMTが複数ある場合その中から、MT番号が最小のMTを選出する

図2: MTiの直後に移動すべきMTの決定法

3.3 マクロタスク分割

本節では本手法を効果的にする前処理について述べる。

第2章で述べたようにプログラムは3種のMTに分割されるが、プログラムによっては1つのMTでキャッシュサイズの何倍もの量のデータを定義参照する場合もある。この問題の対処法として2つの方法が考えられる。

1つ目の手法は階層化である。マクロデータフロー処理は階層型に拡張されているため、膨大なデータを扱うMTであってもRBもしくはSBであれば、そのボディー部を再びMTに分割しMTGを生成することができる。これは収束ループが実行時間の大半を占めるようなプログラムにおいて有効である。

2つ目の手法はMT分割である。キャッシュサイズ以上のデータを定義参照するMTを、キャッシュサイズ以内のデータを定義参照する複数のMTに分割すれば、本手法によりキャッシュの効率向上が期待できる。並列性が向上した方がコード移動の自由度が高まるので、分割されたMT間に依存が生じないように分割する方が望ましい。また共通データ量が大きいMT群を分割する場合には、分割後のMT間で共通データ量が大きくなるよう考慮して分割すべきである。このような分割には、整合分割を応用することができる。図1(b)はMT分割を行なった例である。キャッシュサイズ以上のデータを扱うMT1とMT3を5分割、MT5とMT7を2分割、MT9とMT10を3分割している。

4 プリプロセッサでの性能評価

本章では本手法の実装方式、その性能評価について述べる。

本手法の実現方法としては、FortranプログラムをOSCAR Fortranマルチグレイン並列化コンパイラに入力し、MT分割・MTレベルコード移動の結果を逐次型Fortranプログラムとして出力するプリプロセッサ方式を採った。このキャッシュ最適化プリプロセッサを用いて本手法の性能評価を行なった結果について述べる。

性能評価プログラムとして対称帯状マトリクス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)のプログラムを用いる。オリジナルのプログラムと、それをキャッシュ最適化プリプロセッサで最適化したプログラムとを、それぞれコンパイルして実行し、その実行時間を比較する。実行及びコンパイルはSun Microsystems,Inc.のEnterprise3000 (UltraSPARC)上で行なう。コンパイラはSunSoftのFORTRAN77 4.0を使用し、コンパイルオプションで最大限の最適化を行なう。

CG法プログラムは、変数の初期化部分と実際に連立方程式の係数行列を解く収束計算ループから構成されており、実行時間の大半が収束計算ループによって占めらので、収束計算ループのボディーを第2階層とし最適化を行う。図1(a)は第2階層のMTGである。さらにEnterprise3000の2次キャッシュ(512Kbyte)に合わせて、MT分割を行なう(図1(b))。このMTGを用いてコードを移動しキャッシュ最適化を行なった結果、図1(c)に示されるようにMTレベルでのコードの移動がなされた。複数のマトリクスサイズに対して性能評価を行なった結果を表1に示す。この結果から、本キャッシュ最適化プリプロセッサにより平均13%の速度向上が得られることが確かめられた。

表1: キャッシュ最適化プリプロセッサ性能評価結果

マトリクスサイズ	オリジナルプログラムの実行時間[s]	最適化後のプログラムの実行時間[s]	速度向上率[%]
128×128	2.31	2.26	2.43
192×192	10.76	8.06	33.44
224×224	13.10	11.98	9.38
256×256	20.84	19.08	9.21
320×320	40.43	36.07	12.09

5 まとめ

本稿では、最早実行可能条件解析を用いた大域的コード移動による単一プロセッサキャッシュ最適化手法を提案した。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プログラム入力からリストラクチャリングされた逐次型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。プリプロセッサとしての実装による性能評価の結果からCG法プログラムについては平均13%の速度向上が得られ、本手法の有効性が確かめられた。

今後の課題としては、本手法のマルチプロセッサシステム用キャッシュ利用最適化手法への拡張が挙げられる。

参考文献

[1] H.Kasahara, H.Honda, S.Narita, "A Multi-Grain Compilation Scheme for OSCAR", Proc. 4th Workshop on Languages and Compilers for Parallel Computing, Aug. 1991.

[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] Monica.S.Lam, Edward.E.Rothberg, Michael.E.Wolf, "The Cache Performance and Optimizations of Blocked Algorithms", Fourth Intern. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV), April 9-11, 1991