計算機アーキテクチャ　130－6
（１９９８．　８．　５）

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

稲石 大祐†, 木村 啓二†, 藤本謙作†,
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††

URL: http://www.kasahara.elec.waseda.ac.jp

あらまし　従来のコンパイラによる単一プロセッサ用キャッシュ最適化は個々のループを対象としているため、プログラム全体に比べると局所的な最適化が多く、プログラム全域を対象としたキャッシュ最適化は行われていない。そこで本稿では、最早実行可能条件解析を利用した単一プロセッサ上での FORTRAN プログラムのキャッシュ最適化手法を提案する。OSCAR FORTRAN マルチグレイン自動並列化コンパイラは、FORTRAN プログラムをループ・サブルーチン・基本ブロックの３種のマクロタスク(MT)に分割し、各 MT に最早実行可能条件解析を行いマクロタスクグラフ(MTG)を生成する。MTG は制御依存及びデータ依存に基づく MT 間の実行順序制約、及び MT 間で授受されるデータに関する情報を表現する。本手法ではこのＭＴＧを用いて、先行 MT によってアクセスされたデータにアクセスする後続 MT が先行ＭＴの直後に実行されるよう大域的なコード移動を行い、キャッシュヒット率を向上させる。本手法は、OSCAR FORTRAN マルチグレイン自動並列化コンパイラ中に、最適化された逐次型 FORTRAN を出力するプリプロセッサ機能として実現されている。CG 法プログラムを用いた本キャッシュ最適化手法の性能評価結果を行ったところ 167MHz　UltraSPARC 上で最高 62%の速度向上が得られた。

A Cache Optimization with Earliest Executable Condition Analysis

Daisuke INAISHI†, Keiji KIMURA†, Kensaku FUJIMOTO†,
Wataru OGATA†, Masami OKAMOTO††, Hironori KASAHARA†

Department of Electrical,Electronics and Computer Engineering, Waseda University†,　TOSHIBA Corporation††

Abstract　Cache optimizations by a compiler for a single processor machine have been mainly applied to a singlenested loop. On the contrary, this paper proposes a cache optimization scheme using earliest executable condition analysis for FORTRAN programs on a single processor system. OSCAR FORTRAN multi-grain automatic parallelizing compiler decomposes a FORTRAN program into three types of macrotasks (MT), such as loops, subroutines and basic blocks, and analyzes the earliest executable condition of each MT to extract coarse grain parallelism among MTs and generates a macrotask graph (MTG). The MTG represents data dependence and extended control dependence among MTs and an information of shared data among MTs. By using this MTG, a compiler realizes global code motion to use cache effectively. The code motion technique moves a MT, which accesses data accessed by a precedent MT on MTG, immediately after the precedent MT to increase a cache hit rate. This optimization is realized using OSCAR multi-grain compiler as a preprocessor to output an optimized sequential FORTRAN code. A performance evaluation shows about 62% speed up compared with original program on 167MHz UltraSPARC.

1.はじめに

単一プロセッサ上のキャッシュ利用効率向上はシステムの処理速度に大きく影響するため、キャッシュ最適化の研究が多く行われている。コンパイラによる代表的なキャッシュ最適化としては、キャッシュプリフェッチング[10]のようにアクセスされると予想されるデータを予めキャッシュに読み込んでおくことによりキャッシュヒット率を高める手法や、ループフュージョン、ループインターチェンジ、ブロッキング等のループリストラクチャリング[6,7]のようにループの変形によりキャッシュアクセスを効率的にする手法等が挙げられる。しかしこれらのリストラクチャリングは各々のループ内、もしくはフローグラフ上で隣り合ったループに対して行なわれるため、プログラム全体からみると局所的な最適化になっている。そのため複数のループに跨るもしくはプログラム全域を対象とした広域的な最適化はほとんど行なわれていない。

本稿では、OSCAR FORTRAN マルチグレイン自動並列化コンパイラ[1,3,9]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である最早実行可能条件解析[2,3]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 マルチグレイン自動並列化コンパイラは 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 種類のマクロタスク(MT)に階層的に分割し、MT 間のデータ依存・制御依存を同時に解析することにより各 MT の最早実行可能条件を求め、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「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」手法では、MTG より得られるこれらの情報を用い、先行 MT で定義参照されたデータにアクセスする MT が先行MTの直後に実行されるように MT レベルでの大域的なコード移動を行なうことにより、キャッシュヒット率を向上させる。

2.最早実行可能条件解析

本章では FORTRAN プログラムの粗粒度並列性を抽出する最早実行可能条件解析[2,3]について述べる。

OSCAR FORTRAN マルチグレイン自動並列化コンパイラ[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[2,4])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。マクロデータフロー処理とはこれらの粗粒度タスク間の並列処理手法である。

- BPA(Block of Pseudo Assignment statements)：基本ブロック、およびダイナミックスケジューリングにおけるデータ転送オーバーヘッドを考慮し複数の小基本ブロックを融合したブロック、または内部に独立したデータ依存グラフを持つ１つの基本ブロックを分割することによって得られるブロック
- RB (Repetition Block)：Do ループや、IF 文による後方分岐等によって生成される最外側ナチュラルループ
- SB(Subroutine Block)：インライン展開が有効に適応できないと判断されたサブルーチン

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[1-4]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。

- 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。
- 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 を含むパスに分岐する)
AND
(MTi がデータ依存する全てのマクロタスク MTk(0≦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)]

AND の前の条件が制御依存に起因する実行確定条件であり、AND の後の条件がデータ依存に起因する

データアクセス可能条件である。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(b)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。点線のエッジは拡張された制御依存を表し、実線のエッジはデータ依存を表す。この拡張制御依存エッジは、通常の制御依存だけでなく、MTi のデータ依存先行タスクが実行されないための条件も表している。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。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。MTG では、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方に位置するような 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。例えばあるプログラムが12個の MT に分割され(図1(a))、最早実行可能条件解析の結果、図1(b)のような MTG が生成されたとする。MT 番号はオリジナルプログラムでの出現順に相当するので、MT9 はオリジナルプログラム中では MT8 の次に実行されるはずであるが、MTG 情報より MT9 の最早実行可能条件は MT4 の終了(データ依存)のみであることがわかるため、MT4 の直後に MT9 のコードを移動させることが可能となる(図1(c))。同様に MT11 はどの MT にも依存していので、例えば図1(c)のようにプログラムの先頭に移動させることも可能である。

このように最早実行可能条件解析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。最早実行可能条件解析はデータ依存と同時に制御依存を解析しているため、解析結果より得られる MTG を用いた MT 実行順序の並べ替えは、プログラムの意味を変化させることはない。

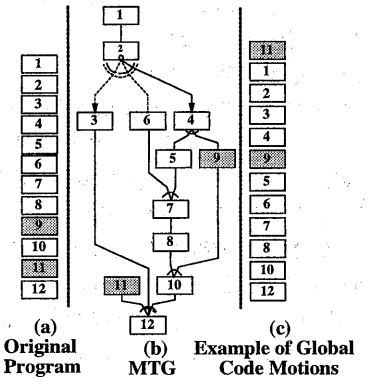

図 1 大域的なコード移動の例

3.2 キャッシュ最適化

本節では大域的な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。図1の MTG において MT4 で定義もしくは参照される配列 A が MT9 で定義もしくは参照されるとする。オリジナルプログラムの実行順序で実行する場合、MT4 の実行時にキャッシュ上に読み込まれた配列 A は、MT5～8 を実行する過程で MT9 の実行前にキャッシュから追い出されてしまう可能性がある。しかし MTG の情報から MT9 を MT4 の直後まで移動させられることがわかるため、MT4 と MT9 が連続して実行されるよう MT を並べ替えれば、MT9 は配列 A をキャッシュ上に存在させたまま利用することが出来る。このように MT 間で授受されるデータに関する情報を用いて、最早実行可能条件が満たされる範囲内

で、キャッシュが効率的に利用されるように MT を並べ替える(コードを移動する)ことにより、従来為し得なかったプログラム全体を対象とした大域的なキャッシュ最適化を実現する。

本手法を実現するにあたって、マクロデータフロー処理で用いられてきた従来の MTG に不足している情報を付加する必要がある。本来 MTG とは最早実行可能条件解析によって引き出された MT 間の並列性を示したものであるため、従来の解析と同時に入力依存解析も行い、以下の情報を本手法のために生成した。

- MTi で定義もしくは参照されているデータを定義もしくは参照する MT
- その MT と MTi とで共通に定義もしくは参照されているデータ(共有データ)とその量

この情報を用いてキャッシュが効率的に利用されるよう MT を並べ替える(コードを移動する)のであるが、基本的には「MTi の直後に移動するべき MT の決定」を効果的に行う問題となる。今回は以下に示すような決定法を用いた。以下でレディ 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。

1) MTi との共有データ量が最も多い 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

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、2)及び 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に設定されている。

3.3 マクロタスク分割

本節では本手法を効果的にするための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 種類の 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よりはるかに大きい量のデータを定義･参照する場合もある。この場合その MT 内部でキャッシュラインのリプレイスが生じてしまい、本手法を効果的に用いるのは難しい。

これに対処する方法を説明するために、まずプログラム中に大規模収束ループがある場合を考える。図 1 において、MT4 が実行時間の大半を占める収束ループであり、大量のデータを定義・参照する場合、図 1(b)の MTG を用いて MT を並べ替えても MT4 自体は変形されないので実行時間は短縮されない。このような場合の対処法としては階層化が有効である。マクロデータフロー処理は階層型[5]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。先の例では、MT4 を階層化しそのボディー部の MT を並べ替えれば、本手法によって速度向上を引き出すことができる。

次にデータを共有している複数の MT がキャッシュサイズ以上の共有データを定義・参照する場合を考える。例えば、MTi と MTj が共有するデータ量が多く、ともにキャッシュサイズを超える量のデータを定義・参照するループであるとする。この場合、MTi の直後に MTj が実行されるよう並べ替えたとしても、MTi が定義・参照するデータ量がキャッシュサイズを超えているため、MTi の終了時に MTj でアクセスされる共有データがキャッシュ上に残っている保証はなく、実行時間を短縮させられるとは限らない。さらに、共有データの一部がキャッシュ上に残っていたとしても、MTj の実行において、そのデータを定義・参照する前に追い出してしまう可能性もある。このような場合の対処法としては MT 分割が有効である。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 にループイタレーション方向で分割すれば、本手法によるキャッシュの効率向上が期待できる。先の例では、MTi と MTj を、分割後の MT で定義・参照されるデータ量がキャッシュに収まるように n 分割(MTi-1～MTi-n、MTj-1～MTj-n)し、分割後の MTG を用いて MTi-1, MTj-1, MTi-2, MTj-2, ...,MTi-n, MTj-n の順序で実行されるよう並び替えれば、キャッシュを効率的に利用することができる。

この MT 分割においては、MT 間の並列性が高い方がコード移動の自由度が高まるため、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。また共有データ量が大きい MT 群を分割する場合に

は、分割後のMTGを用いた並べ替えによってキャッシュの高効率化が望めるように考慮して分割すべきである。例えばキャッシュサイズを超える配列A(1000)がMT1,MT2で定義・参照され、MT1ではA(1:1000)を、MT2ではA(1:800)を定義・参照するとする。均等に2分割するとMT1-1でA(1:500)、MT1-2でA(501:1000)、MT2-1でA(1:400)、MT2-2でA(401:800)となり、並べ替えたとしても、配列Aの定義・参照範囲がずれているためにキャッシュ上の配列Aを効率的に利用できない。したがって、配列の定義・参照範囲を考慮して、分割後のMT間で定義・参照範囲が等しくもしくは非常に近くなるように分割する必要がある。先の例ではMT1-1でA(1:400)、MT1-2でA(401:1000)となるように分割できればキャッシュ上の配列Aを効率的に利用することができる。このような分割には、マクロデータフロー処理におけるデータローカライゼーションで用いられる整合分割[8]を応用することができる。

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、及び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、OSCAR FORTRANマルチグレイン自動並列化コンパイラを、FORTRANを出力するプリプロセッサとして用いるという実装形態を採った。すなわちOSCARコンパイラを拡張し、MT並べ替え処理及び逐次型FORTRAN出力処理を付加することで、FORTRANプログラムからMTレベルでのキャッシュ最適化大域的コード移動を行い、逐次型FORTRANプログラムを出力するプリプロセッサ方式とした。逐次型FORTRANを出力することにより、局所的なキャッシュ最適化に関しては従来の市販コンパイラに実装されている最適化を利用することができる。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。

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(コンパイルオプションは同じ)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 (167MHz Ultra SPARC)上で行なう。UltraSPARCの1次キャッシュはDキャッシュが16Kバイトのダイレクトマッピングで I キャッシュが16Kバイトの2ウェイセットアソシアティブ、ラインサイズはともに32バイトである。2次Unifiedキャッシュは512Kバイトのダイレクトマッピングでラインサイズは64バイトである。コンパイラはSunSoftのFORTRAN77 4.0を用い、コンパイルオプションには-xarget=native, -O5を指示した。-xarget=nativeはターゲットマシンの各種パラメータに応じたプロセッサ、キャッシュ構成、命令セットに関する最適化を、-O5は実行時間短縮化の最高レベルの最適化を指示するものである。

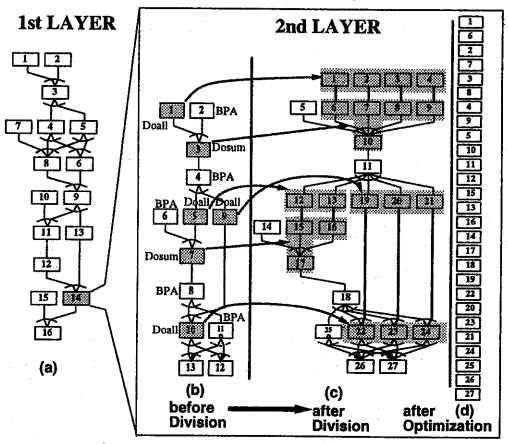

図2　CG法のマクロタスクグラフ

CG法プログラムは図2(a)のようなMTGを生成する。この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いるため、実行時間の大半が収束計算ループ(図2(a)MT14)によって占められ、Enterprise3000の2次キャッシュサイズ(512Kバイト)以上のデータを定義・参照している。よって収束計算ループのボディーを第2階層とし階層的な最適化を行う。図2(b)は第2階層のMTGである。図2(b)のMTGにおいて、MT1とMT3、MT5とMT7、MT9とMT10の各組で共有データ量が多いことが解析結果より得られるが、各組とも2次キャッシュサイズ以上のデータを定義・参照している。したがって、MT1,3、MT5,7、MT9,10の各組で扱われるデータ量に従って2次キャッシュサイズ内に収まるようにMT分割を行う。MT1,3を4分割、MT5,7を2分割、MT9,10を3分割した結果

が図2(c)の MTG である。図2(c)の MTG に対して MT を並べ替え、最適化を行った結果、図2(d)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。共有データ量の多い MT1,6、MT2,7 等が連続して実行されるため、キャッシュ上のデータが効率的に利用され、キャッシュ最適化されている様子がわかる。

CG 法中で用いられる連立方程式の係数行列サイズを変化させて本手法の性能を評価した結果を表1に示す。係数行列サイズに応じて各 MT が2次キャッシュ(512K バイト)に収まるように MT 分割を行った。表1では、異なる係数行列サイズの場合の、オリジナル FORTRAN プログラムが要した実行時間と、本キャッシュ最適化手法を組み込んだプリプロセッサ（OSCAR マルチグレイン並列化コンパイラ）によって出力された最適化 FORTRAN プログラムの実行時間及びその時のスピードアップ率を示している。なお、MT 分割数は数パターンの内、最も性能が高かったものを採用している。

表1 最早実行可能条件解析を用いた
キャッシュ最適化による速度向上

係数行列サイズ	実行時間[s]		速度
	オリジナル	最適化後	向上率
16384(128x128)	1.87	1.64	15.5%
36864(192x192)	9.57	5.91	62.3%
65536(256x256)	17.36	14.05	23.5%
102400(320x320)	37.15	27.25	36.4%
147456(384x384)	65.76	49.99	31.6%
200704(448x448)	101.57	72.84	39.4%
262144(512x512)	130.08	108.26	20.2%

表 1 から分かるように、本手法により、各行列サイズにおいて、実WSサーバ上で約 15%から最高 62%もの処理速度の向上が得られることが確かめられた

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 FORTTAN マルチグレイン自動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単一の UltraSPARC サーバーマシン上で最高60%以上の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。

本研究の一部は、通産省次世代情報処理基盤技術開発事業並列分散分野マルチプロセッサコンピューティング領域研究の一環として行われた。

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]H.Kasahara, H.Honda, M.Iwata, M.Hirota, "A Compilation Scheme for Macro-dataflow Computation on Hierarchical Multiprocessor System", Inter. Conf. on Parallel Processing, Aug.1990.

[5]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,, Vol35, No4, pp.513-521, Apl.1994.

[6]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

[7]David F.Bacon, Susan L.Graham, Oliver J.Sharp "Compiler Transformations for High-Performance Computing", ACM Computing Surveys, Vol.26, No4, Dec 1994

[8]吉田 明正, 前田 誠司, 尾形 航, 笠原 博徳, 'Fortran マクロデータフロー処理における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.35, No.9, pp.1848-1860, Sep.1994.

[9]笠原, 尾形, 木村, 小幡, 飛田, 稲石 "マルチグレイン並列化コンパイラとそのアーキテクチャ支援", 信学技報, TECHNICAL REPORT OF IEICE. ICD98-10, CPSY98-10, FTS98-10, 1998-04

[10]Todd C.Mowry, Monica S.Lam, Anoop Gupta, 'Design and Evaluation of a Compiler Algorithm for Prefetching", APLOS V, Oct 1992