数理モデル化と問題解決　38－12
（２００２．３．５）

自動並列化コンパイラPROMIS用ユニモジュラ変換の設計と実装

石内寿子† 山口智美‡ 庄野逸‡ 城和貴†
j-ken@alice.ics.nara-wu.ac.jp
† 奈良女子大学理学部情報科学科 ‡ 奈良女子大学大学院人間文化研究科

概 要

自動並列化コンパイラにおいて、並列度を増加させるための最適化手法として様々なループ変換法が提案されてきた。しかし、これらの方法は個々に開発されてきたため、それぞれ適応条件や効果が異なる。そのため、コンパイラに複数のループ変換法を実装しても、その中からどの変換が最適であるかを決定することが難しい問題である。一方、ユニモジュラ変換という、標準的なループ変換のいくつかの組み合わせによる変換を一度に行える変換がある。この一度の変換で、並列化のための最適解を求めることができるのである。このユニモジュラ変換を、イリノイ大学で開発された自動並列化コンパイラ PROMIS に付け加えることが本研究の目的である。

The Design and Implementation of Unimodular Transformations for the Parallelizing Compiler PROMIS

Hisako Ishiuchi† Tomomi Yamaguchi‡ Hayaru Shouno‡ Kazuki Joe†
† Department of Information & Computer Sciences, Nara Women's University
‡ Graduate School of Human Culture, Nara Women's University

概 要

For parallelizing compilers, many loop transformations have been proposed as optimization methods to exploit parallelism. However, since these methods have been designed separately, each method has its own conditions and effect to be applied. Therefore, even if we implement many loop transformations in a compiler, it is difficult to determine which combinatorial use of the transformations is optimal. Another transformation, which is called Unimodular, has the same effect to the combination of some transformations. Some transformation may get the optimal combination regarding to parallelism. In this paper, we describe the implementation of the unimodular transformation to the Parallelizing Compiler PROMIS, which is developed at the University of Illinois.

1 はじめに

本研究室では、イリノイ大学で開発され共有メモリシステムを対象としている自動並列化コンパイラPROMIS[1]を、分散メモリシステムを対象とした、PROMIS-NWU に拡張する研究開発に着手している [2]。

自動並列化コンパイラの中間表現には一般に様々な形のものが開発されている。その中間表現の多様性は、最適化の手法ごとにある特定の中間表現を対象としなければいけないという、複雑な操作を要求する。そこで、PROMIS では最適化の手法に関わらず１つの中間表現を利用できるように統一的中間表現 Universal Intermediate Representation (UIR) が考案され実装された。

上でも述べたように、この PROMIS は共有メモリシステムを対象としている。一般的にも、自動並列化コンパイラは、共有メモリシステムを対象としていて、分散メモリシステムを対象とした自動並列化コンパイラは開発途上にある研究分野である。この分野の研究に、本研究グループは取り組んでいる。その実現性については、PROMIS の UIR である Hierarchical Task Graph(HTG) を Task and Variable Representation Graph(TVRG)[3] に置き換えることで達成できると提案している。

この新しい中間表現、TVRG をもつ PROMIS-NWU に新たな並列化手法を加えるのが本サブテーマの目的である。しかしながら、PROMIS-NWU の開発がまだ完了していないために、現在、PROMIS 上でその実装作業を行っている。ただし、PROMIS-NWU は PROMIS の完全上位互換であるので、PROMIS への実装が完了すれば PROMIS-NWU への移植は簡単に行うことができる。

並列化手法に関しては、PROMIS 自体にも、いくつかのループ変換はすでに実装されている。しかし、これらのループ変換は、適応させる際にどの組み合わせが最適であるかの評価が難しい従来のループ変換である。

これらはそれぞれ独自に研究されてきたため、それぞれ異なった適応条件や変換のルールがあり、変換の組み合わせの効果や適応可能かという問題の予測を困難にしている。そのため、どの変換の組み合わせが任意のループネストに最適となるかを決定する有効な方法はまだ確立していない。PROMIS に実装されているループ変換も、このような問題から、それらの変換の組み合わせを選択する有益な実行メカニズムはない。

このような従来のループ変換が抱えている問題に

対して、また別の観点から並列化手法を考案した研究がある。この方法は、ユニモジュラ行列を使用することから、ユニモジュラ変換と呼ばれ、1980年代中旬にUtpal Banerjeeによって考案された[4]。ユニモジュラ変換では、各ループネストのイタレーション空間によって、ループネストを1つの全体としてとらえ、それに写像変換を適用するのである。したがって、上述したような各変換の組み合わせを考え評価をしなくてもよい。また、実際には、従来のloop interchangeやloop skewing、loop reversalの組み合わせによる変換をユニモジュラ変換は行っている。よって、3つの変換よりも、このユニモジュラ変換を実装することが、実装コストの低下につながる。このユニモジュラ変換を汎用的な形で実装することが本研究の目的であり、その設計と実装方針についての報告を行う。

2 ユニモジュラ変換

ユニモジュラ変換とは、データ依存により並列実行できないループネストをユニモジュラ行列による線形写像により、プログラムの意味は変えずに、並列性を増加させる変換法である。

ループネスト（深さm）において、各ループのindex変数の値（index値）はm次元のベクトルとして表すことができる。つまり、index値は、m次元空間上の点（index point）とみなすことができる。このindex pointを、並列性が増加するように同じm次元空間に写す、m次元空間からm次元空間への線形写像を考える。

この写像に関しては、一般にindex変数は整数値をとることから、写されたindex変数も整数値をもつことが望ましい。また、index値を写すことによって実行順序を変えるだけであるので、1対1写像でなければならない。以上のことから、整数行列であり、かつ、行列式が± 1のユニモジュラ行列がこの写像を定める行列として最適であると考えられた。

写像変換が有効である条件は、プログラムの意味を変えないということである。変換により実行順序は変えることになるが、プログラムの意味は変わらないように、つまり、データの依存関係が壊れないようにするということである。そのような条件を満たす変換を行うユニモジュラ行列が存在するとき、その行列による変換は有効であると言う。

有効な変換のもとで、目的に合ったユニモジュラ行列を見つけていく。その目的とは、inner loop parallelizationと、outer loop parallelizationである[5]。

inner loop parallelizationとは、ループネスト中の内側のループを並列実行可能なように変換を行うことである。これを満たすユニモジュラ行列の制約条件は、各行列成分に対する下限のみである。つまり、上限がないのでそのような性質を満たすユニモジュラ行列は無限に存在し、inner loop parallelizationは常に可能であることがわかっている[5]。

outer loop parallelizationは、一番外側のループからいくつかのループを並列実行できるように変換することである。つまり、全ての距離ベクトルのレベルがl $(2 \leq l \leq m)$であるとき、外側から$l-1$個のループが並列実行可能ということである。これは、写像後の距離ベクトル全てについて、ベクトルを各行に並べた行列（依存行列）を作り、その行列のランクをrとすると、$m-r$個のループが並列実行可能ということになる。すなわち、$r=m$の場合は、並列実行できるような変換を行うユニモジュラ行列は存在しないということである。inner loop parallelizatinと比較すると、このように制約上限は厳しくなるが、並列度の増加はこちらの方が大きく有効な方法である。制約条件に関しては、ここまで写像後の依存行列のランクによって決まると述べてきたが、写像前の依存行列からでも条件を考えることができる。なぜなら、写像はユニモジュラ行列によって行うので、写像前の依存行列のランクと写像後の距離行列のランクは等しいことが保証されているからである。

3 設計

このユニモジュラ変換を新たなループ最適化の1つの手法としてPROMISに実装する。その際の設計方針をこの章で述べる。以下のものが、ユニモジュラ変換の実装において必要になってくる。

- プログラムからループネストを取得
- 変換に必要な情報の取得
- inner loop parallelizationかouter loop parallelizationで行うかの決定
- 選択された方法での、ユニモジュラ行列を求めるための行列操作
- 求められたユニモジュラ行列によるループネストのイタレーション空間の写像変換

3.1 ループネストの抽出

プログラム中のループネストは、PROMISのUIRであるHTGから取得する。HTGとは、具体的には、ループと関数に基づいてプログラムを階層構造で表したタスクグラフである。このHTGにおいてループは1つ1つループノードとして表されている。従って、ループネストを表すためには、ループをリ

スト化し1つのループネストとすればよい。また、プログラム中のループネストに対して、まとめて変換が行えるようにしたいので、図1のようにループネストもさらにリスト化しておくことにする。

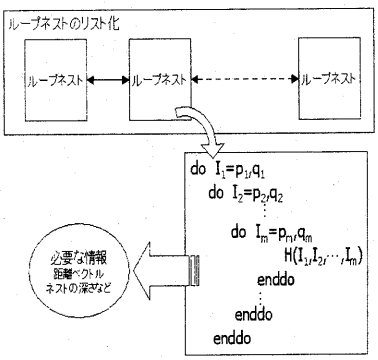

図1 ループネストのリスト化

3.2 変換に必要な情報の収集

変換に必要な情報は、具体的には (1) ループネストの深さ (2) ループインデックスの上限・下限 (3) 依存の距離ベクトル (4) 依存行列のランクの4つが必要である。(1) は、ループネストを構築する際、ループをリストに接続していくときに数えればよい。(2) は、ループをリスト化する際にループノードから情報を取り出しておく。(3) は、PROMIS の Data Dependence Graph が (DDG) 保持している情報である。DDG とは、データ依存関係をグラフ化したものであり、仕様上は HTG に含まれるものであるが、実装上は、HTG とは分離した形で組み込まれている。依存元、依存先という情報、データ依存のタイプなどともに距離ベクトルの情報も含んでいるので、ここから取得する。(4) は、(3) で取り出した距離ベクトルからループネストごとに依存行列を作りランクを求める。つまり、行列操作になるので、ここで PROMIS 内から取り出さなければならないものは何もない。

3.3 変換方法の選択

inner loop parallelization か outer parallelization の選択は、前章で述べたように、依存行列から求められたランクによって決まるので、ここでも、また PROMIS 内から取り出さなければならないものはない。ループネストの深さ m とランク r を比較し、$r = m$ の場合には、inner loop parallelization を $r < m$ の場合には outer loop parallelization を選択する。

3.4 ユニモジュラ行列の取得

outer loop parallelization、inner loop pallelization どちらにおいても、ユニモジュラ行列を見つけるには、数学的に証明されている行列理論に従い、行列演算を行っていく。

3.5 ユニモジュラ行列による変換

イタレーション空間の写像変換とは、インデックス変数のとる上限・下限を変えることに他ならない。ここでは、既に取り出している情報である元のループネストの上限、下限をユニモジュラ行列で写すことになる。よって、新たに必要な情報はない。

4 実装方針

これまで述べてきたように、新たなループ変換手法としてユニモジュラ変換を加え、PROMIS を拡張する。PROMIS では、拡張作業を Extensible compiler Interface(ECI) を利用し、容易に行える設計になっている。本研究も、この ECI を利用して上の設計に従い実装を行う。

4.1 ループネストの構築

まず、ループネストをループノードからリスト化する。ループノードは、上で述べたように HTG から取り出す。HTG は、プログラムの代入文や関数、ループなどをノードとしてそれをリスト化し、さらに、階層構造を成しているタスクグラフである。これらのノードは HtgBase のサブノードとしてそれぞれ存在している。そこで、このリストを順にたどっていき、ループノードにあたるものを探していく。PROMIS の HTG では、ループノードは2種類ある。GeneralLoop と FixedIterLoop の2つのクラスである。この2つの違いは、ループのイタレーション回数が固定されているかどうかである。ユニモジュラ変換は、先に述べたようにループのインデックス変数が一般的に整数値を取ることに着目して考え出された手法であるので、対象となるのは FixedIterLoop の方である。それをリストにつなげていき、1つのループネストを構築する。

4.2 情報の抽出

変換に必要な情報は各ループネストごとに定まるものなので、構築した際に、その情報を取り出しておく。さらに、それらの情報をもったループネスト1つ1つをリスト化する。

その情報の取り出し方に関しては、UIRから取り出さなければならないものはループインデックスの上限・下限と依存の距離ベクトルである。

インデックス変数の上限・下限の値には、FixedIterLoopクラス中のUpperBound()、LowerBound()というメソッドでアクセスできる。

次に依存の距離ベクトルは、前章で述べたようにDDGが保持している情報である。DDGは設計上HTGに含まれている。その概略図を下に示しておく。DDGは式を表すノードと依存関係を表す矢印(Data Dependence Arc：DDArc)からなる。このDDArcのデータメンバにdistance_vectorsというリストがあるので、これにアクセスしてループネスト内の依存構造を調べる。

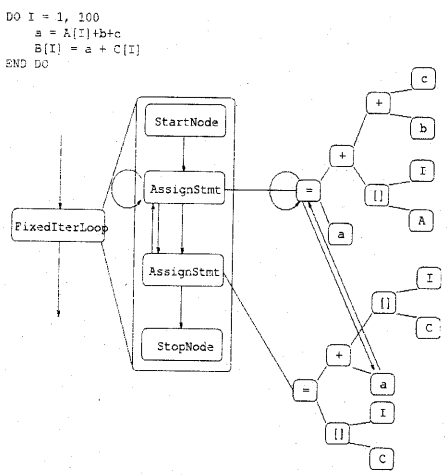

図2 HTG、DDGの概略図

5 まとめ

本稿では、自動並列化コンパイラPROMISに対して追加したループ並列化手法の1つ、ユニモジュラ変換の設計方針と実装方針について報告した。今後は、PROMIS-NWUへの実装と、ユニモジュラ変換から従来のループ変換の最適な組み合わせを求める機能を追加する予定である。

既に述べたようにPROMIS-NWUはPROMISの上位互換である。よって、PROMISに実装したこのユニモジュラ変換の移植は簡単に行える。

次に、ループ変換の組み合わせの最適解とは、ユニモジュラ変換によって行われた、loop interchange、loop skewing、loop reversalの組み合わせを求めるということである。これらは従来の変換方法であるので、はじめに述べたように、これらの変換の中からの組み合わせとその適応順序をどのように決めれば、並列度がどれだけ上がるかを予測し、その最適解を求めるのが困難である。

ユニモジュラ変換では、変換に用いた行列からこれらの組み合わせを求めることができることが数学的に証明されているので、その組み合わせを求め、最適解とする。

以上のような課題に今後取り組み、ユニモジュラ変換のPROMIS-NWUにおける実装を完成させる。

参考文献

[1] H, Saito., N, Stavrakos., C, Polychronopoulos. and A, Nicolau.: The Design of the PROMIS Compiler, Int'l J. of Parallel Programming, Vol28, No.2, pp.195-212 (2000)

[2] 山口智美, 石内寿子, 岩坂麻美, 羽田昌代, 庄野逸, 城和貴：自動並列化コンパイラPROMIS-NWUの概要 情報処理学会 計算機アーキテクチャ研究会, ARC-144-14, pp.79-84

[3] M, Haneda., H, Shouno and Joe, K.: Task and Variable Representation Graph: An Intermediate Representation of Parallelizing Compilers for Distributed Shared Memory Systems, Int'l workshop on Advanced Compiler Technology for High Performance and Embded Systems, pp.47-55 (2001)

[4] U, Banerjee.: Loop Transformations for Restructuring Compilers, Kluwer Academic, (1993)

[5] U, Banerjee.: LOOP PARALLELIZATION, Kluwer Academic, (1994)