2D−1

ガード付きPDGを用いた命令レベル並列アーキテクチャのための最適化コンパイラの実現

楊 巍† 古関 聰‡ 小松 秀昭‡ 深澤 良彰†
†早稲田大学理工学部 ‡日本IBM(株) 東京基礎研究所

1 はじめに

EPICアーキテクチャでは、分岐予測ミスによるパイプラインの乱れを防ぐために、命令の条件付き実行機能を取り入れている。コンパイラが実行条件機能を有効に利用できるために、全ての命令にプレディケートを付加する。

本研究では、ガード付きPDG(Program Dependence Graph)を用いることによって命令の条件付き実行のためのプレディケートを生成する。さらに、実行条件を投機的移動することによってプログラムのクリティカルパスの圧縮を行う手法を提案する。なお、アーキテクチャのモデルは米INTEL社のIA-64のMERCEDとする。

2 本研究の概要

MERCEDでは、1つの比較命令によって互いに排他的な2つの述部(図1のp_1とp_2)が生成される。この2つの述部は各分岐先の命令の実行条件として用いられる。結果として生成されたコードに条件分岐がないため、2つの分岐部の命令が並列実行できる。IA-64では、全ての命令が一つの実行条件しか持てない。

本研究では、実行条件の情報をPDG[1]に追加することでガード付きPDGを生成する。さらに、クリティカルパス上の実行条件を投機的移動する際に、その内容が論理式にならないようにプログラムに対して変形を施す必要がある。この変形処理によってクリティカルパスを短縮することが可能となる。

3 ガード付きPDGの生成

本研究では、GCCにIA64用のサブルーチンを組み込み、実装を試みた。GCCでは、プログラムへの最適化が全てRTLという中間言語の上で行われる。本手法では、まずRTLをもとに関数ごとにCFGを生成する。

Realization of Optimizing Compiler using Guarded Program Dependence Graph for Instruction-Level Architecture
Wei Yang†, Akira Koseki‡, Hideaki Komatsu‡, and Yoshiaki Fukazawa†
†School of Science & Engineering, Waseda University
‡IBM Japan.Ltd. Tokyo Research Laboratory

レジスタの再利用による逆依存を防ぐために、制御フローグラフ(CFG)に対して静的単一代入(SSA[3])を施す。その後に制御依存グラフ(CDG)とPDGを生成し、CDGとPDGに基づきガード付きPDGを構築する。CFGを構築する際に検出された基本ブロックや最内ループの情報は関数ごとにRTLと別途に保存しておく。

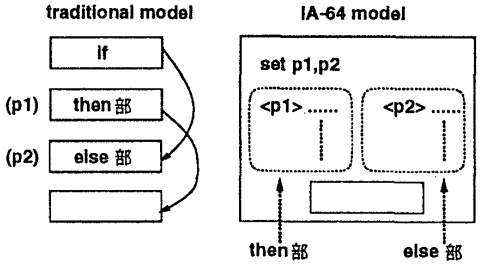

図1: MERCEDの条件付き実行モデル

4 実行条件の投機的実行

本研究では、ガード付きPDGを生成した後に、以下の手順で実行条件の投機的移動を行う。

1. ガード付きPDGからクリティカルパスを選ぶ。複数のクリティカルパスが存在する場合は、そのうちの任意の1本を選ぶ。
2. クリティカルパス上の最後の実行条件p_nをパスの先頭に移動し、$(p_i)R_i = 1$と$(p_i)R_i = 0$の命令をガード付きPDGの先頭に挿入し、p_iの値を汎用レジスタに書き込む。
3. 移動後の実行条件p_nの値が他の各実行条件の論理和になる。その論理和を$R_i(i = 1, .., n)$を用いて計算する。この処理を実行条件が1つの論理値になるまで繰り返す。
4. 汎用レジスタの内容を実行条件レジスタに反映し、命令の実行条件を更新する。

図2には、実行条件の投機的移動を行った例を示す。$abcdefghi$をクリティカルパスとし、プロセッサの並列

度を 8 とする。$abcdefghi$ 上の実行条件 p_8 をパスの先頭に投機的移動した場合、p_8 の値が各実行条件の論理和となる。p_8 を実行条件としたiが p_1 から p_8 までのパス上の命令との依存がなければ、i を p_8 の直後に持ちあげることによって、a から i の実行時間が 9 マシンサイクルから 7 マシンサイクルに減少し、クリティカルパスが短くなることが分かる。

5 最適化

SSA 変換した後生成したガード付き PDG に対して、レジスタ割付けとコードスケジューリングを行う。この段階で基本ブロックの移動や分割が行われるため、それらの処理に合わせて、基本ブロックや最内ループの情報を更新する。

5.1 マシンモード

GCC では、多数のハードウェアへの対応を実現するために、独自のフォーマットで記述されるターゲットマシンの定義ファイルが用意されている。これらのを利用して、アーキテクチャ及び OS ごとにモジュールを生成し、RTL で扱える命令セットに特定し、最適化を行われた後の RTL からターゲットマシンのアセンブリコードを生成することによってハードウェアおよびシステムの違いを吸収する。今現在は IA-64 のハードウェアに関する詳細データがまだ明らかになっておらず、様々な可能性を予想しながら作成するしかない。

5.2 レジスタ割り付け

本研究では、レジスタ割付けの手法として、文献 [2] の方法を採り入れる。クリティカルパスにおける実行条件の投機的移動を行う際に、大量のレジスタが必要となる。汎用レジスタおよび実行条件レジスタの割付けについて、以下のように行う。

1. レジスタの優先度
   実行条件レジスタが汎用レジスタより優先度が高いものとする。実行条件レジスタに対してカラーリングを行う際に、スピルが発生する場合、$(p_i)R_i = 1$ と $(p_i)R_i = 0$ のような命令を用いて、実行条件レジスタの内容を汎用レジスタに退避させる。一方、汎用レジスタに対するカラーリングはレジスタ生存グラフ [2] を利用する。

2. スピルアウトの順番
   実行条件レジスタと汎用レジスタが混在する場合、汎用レジスタを優先的に処理する。汎用レジスタ同士に対してはレジスタ生存グラフ [2] で割り付けを行うとする。実行条件レジスタのスピルアウトについては、まず実行条件が偽となるレジスタを候補として選び、同様に偽となる実行条件を持つレジスタが複数ある場合、参照する命令数の少ないものからスピルアウトする。

6 おわりに

本研究では、ガード付き PDG を用いることによって、予測困難な分岐および分岐予測ミスによる高いペナルティの問題を回避し、MERCED の並列性を生かすコードが生成できることが分かった。さらに、実行条件の投機的移動によって、クリティカルパスを圧縮し、プログラムの実行時間を短縮させる手法を提案した。今後、本手法の改良および評価を行っていきたい。

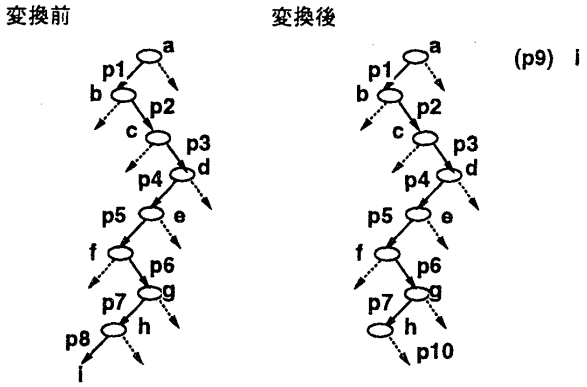

図 2: 実行条件の投機的移動

参考文献

[1] J. Ferrante, K.J. Ottenstein, J.D. Warren, "The Program Dependence Graph and Its Use in Optimization", ACM Transactions on Progamming Languages and Systems, Vol.9, No.3, pp319-349(1987)

[2] 百瀬浩之, 古関聰, 小松秀昭, 深澤良彰：" 命令レベル並列アーキテクチャのためのコードスケジューラ及びレジスタアロケータの協調技法", 情報処理学会論文誌,Vol.38,No.3,1997, pp.584-594.

[3] R. Cytron, J. Ferrante, N.K. Rosen, M.N. Wegman, F.K. Zadeck, "An Efficient Method of Computing Static Single Assignment Form", Conference Record of the 16th ACM Symposium on the Principles of Programming Languages, pp25-35(1989).