制御依存関係による並列度利用の限界に関する定量的評価

吉瀬謙二, 中村友洋, 辻秀典, 田中英彦
東京大学工学部

1 はじめに

プログラムの広い範囲を対象にデータフロー解析とプログラム変換を行なうことで, プログラムに内在する並列度の利用が可能となる。しかし, 汎用プログラムには多数の分岐命令が存在するため簡単に解析範囲を広げることができない。最新のマイクロプロセッサは, 実行される可能性の高い命令をバッファ (命令ウィンドウ) に格納し, 実行可能となった命令をアウト・オブ・オーダ発行することで命令レベルの並列度を利用する。

我々は汎用アプリケーションを対象に実行コードがもつ様々なレベルの並列度の利用を目指しており, 命令ウィンドウで決定される解析範囲を大きくしていくアプローチとともに, プログラム中の独立なフローを抽出する手法を検討している。

本稿では, 広範囲の解析を可能とするために基本ブロックを解析の基本単位とし, ブロックレベル並列度の限界, ブロック単位の解析範囲を広げて得ることができる並列度を測定する。これよりブロックレベル並列度の利用に関する考察をおこなった。

2 トレースデータを用いた解析

プログラムから並列に実行できるフローの利用を考えるとき命令ごとの依存関係を求めていては解析が複雑になり, 大域的な評価が困難になる。この問題を解決するため, 解析の単位としてトレースデータから得られる基本ブロックを考える。

解析するプログラムは SPECint92 から compress, espresso をとりあげる。これらのプログラムは SPARC Station20 の上で gcc コンパイラを用いコンパイルされたものである。実際の解析ではスカラプロセッサのシミュレーションで得られたトレースデータを使った。なお解析する命令数は 200 万命令で打ち切った。

2.1 ブロック長とブロック内並列度

表 1 に解析の単位となる基本ブロックに関するデータとして, 実行されたブロックの数, ブロックの長さ (ブロック内の命令数), ブロック内のデータフロー解析により得られた並列度を示す。ただし全ての命令のレイテンシは 1 ステップと仮定している。

Extracting basic block lebel parallelism
Kenji KISE, Tomohiro NAKAMURA,
Hidenori TSUJI, Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

ブロック内で得られる並列度は 3 以下と非常に小さいが静的な解析で抽出可能であるため, ブロック間にまたがる並列度と比較しこれらの並列度の利用は容易である。

ブロック内のデータフロー解析により得たクリティカルパスの長さを各ブロックのコスト情報として保存しておき, ブロック間の依存関係とブロックのコストを用いて以降の実行ステップ数を計算する。

	ブロック数	ブロック長 (命令)	並列度
compress	254,982	7.84	2.19
espresso	377,244	5.30	1.58

表 1: 200 万命令の解析による基本ブロックの特性

2.2 ブロックレベル並列度の限界

基本ブロックを解析の単位として考え (ブロック間で命令の移動を禁止) ブロックレベルの並列度を利用する。

ブロック間のデータ依存関係をエッジとし, ブロックをノードとしたデータフローグラフを作成する。ただし各ノードには先に計算したブロック内のコスト情報を付加しておく。得られたデータフローグラフからクリティカルパスのコストを求め, それを利用して並列度を計算した結果を表 2 に示す。

	実行ステップ数	並列度 (平均)
compress	340,982	5.87
espresso	365,399	5.47

表 2: ブロックレベル並列度の限界

データフローグラフ作成の際に制御依存関係を排除しているため, 表 2 の並列度はブロックレベルの並列度を利用した場合の限界を示す。現実にはこの条件を満たすことはできないが, 並列に実行可能で計算に必要なブロックをフェッチ時に指定することができれば, 5〜6 程度の並列度を利用できる可能性がある。

2.3 ブロックレベルのウィンドウ

1 エントリに 1 ブロックの情報を格納するウィンドウを用いブロックレベルの並列度を利用することを考える。ウィンドウ内を十分なブロックで満たすため, ウィンドウサイズと同じ段数の分岐予測をおこなう。ウィンドウ内のブロックを解析し同時に実行可能なブロックを発行することでブロックレベル並列度を得る。また全ての分岐予測が当たると仮定する。

ウィンドウサイズを 1 から 100 まで変化させたときの総実行ステップ数を図 1 に示す。

compress では 5 以下のウィンドウサイズで, espresso では 10 以下で, 明らかなステップ数の減少がみられるがそれ以降では変化がゆるやかになっていくのがわかる。

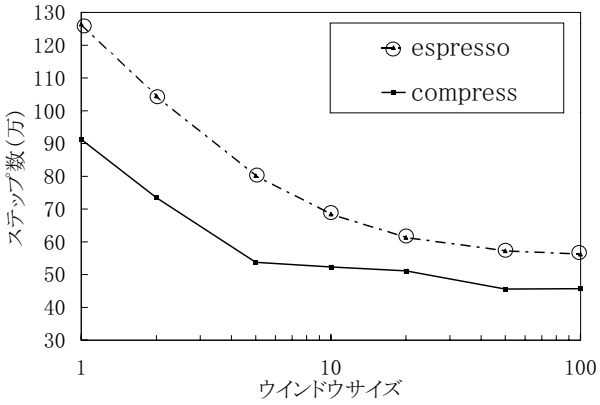

図 1: ウィンドウサイズによるステップ数の変化

表 3 にウィンドウサイズが 10 と 100 の場合における並列度と, ウィンドウサイズが 1 の時の相対性能を 1 とし, ウィンドウサイズが無限大のときの性能向上率を 100%とした時の性能向上率を示す。

	ウィンドウサイズ 10		ウィンドウサイズ 100	
	並列度	性能向上率	並列度	性能向上率
compress	3.82	64.9%	4.38	74.6%
espresso	2.92	53.4%	3.56	65.1%

表 3:ウィンドウサイズとブロックレベル並列度の関係

表からウィンドウサイズ 10 のときに, ウィンドウサイズを増加させて得られるすべての性能向上に対し 50%以上の性能向上率を得ていることがわかる。

2.4 実行ステップ数の推移

ここまでのデータを整理する。それぞれのデータは, 以下で定義する各実行モデルによる総実行ステップ数に対応している。

スカラプロセッサモデル (scalar)：1 ステップに 1 命令実行可能なモデル (比較対象用のモデル)

ブロック実行モデル (block)：実行中の 1 ブロック内のみの並列度が利用可能なモデル

ウィンドウサイズ 10 のブロック実行モデル (b-10)：ウィンドウ内の 10 ブロックを解析し並列に実行できる複数のブロックを同時に実行可能としたモデル

理想的なブロック実行モデル (b-ideal)：同時に複数のブロックを実行可能とし, 制御依存を理想化したモデル

オラクルマシン (oracle)：1 ステップに複数の命令が実行可能であり, 制御依存関係を理想化したモデル

スカラプロセッサモデルでは並列度 1, オラクルマシンの並列度は compress, espresso それぞれ 32.0, 34.6 である。参考としてこれらのモデルを評価に加えた。

それぞれのモデルにおいて compress, espresso の 200 万命令を実行したときの総実行ステップ数を図 2 に示す。

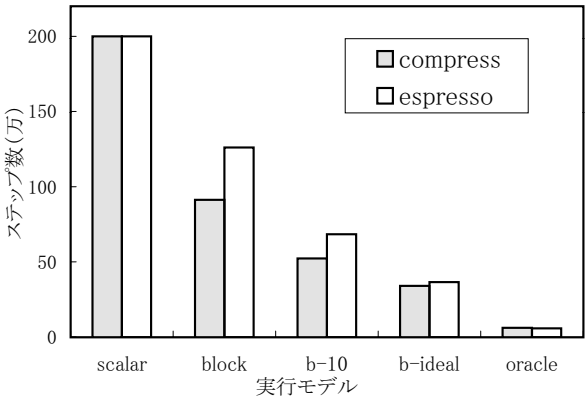

図 2: 実行ステップ数の推移

3 考察

先の実行モデルのうち, 理想的なブロック実行モデルとオラクルマシンは, 制御依存関係の理想化 (分岐の結果が全てわかっている) を仮定しているためトレースデータを用いることではじめてシミュレートできる。これらのモデルは性能向上の限界を示すという点で重要であるが, 実現不可能なモデルである。これらの理想的なモデルから, ブロック間の命令移動を禁止することが並列度利用に関する大きな制約になっていることがわかる。この制約により並列度の限界は compress で 32.0 から 5.9 へ espresso で 34.6 から 5.5 へと大幅に減少する。

ブロックを解析単位とした 3 つのモデルに注目する。ウィンドウサイズ 10 のブロック実行モデルでは, ウィンドウサイズの増加により得られる全性能向上の 50%以上の性能向上を達成できることが判明した。残り 50%の性能向上を得るためには, 広範囲の解析を可能とするテクニックが必要となる。

4 まとめ

基本ブロックを解析の単位とする 3 つのモデルを中心に 5 つの実行モデルを定義し, 各モデルにおいてプログラムの総実行ステップ数を計算した。これより, ブロックレベル並列度を十分に利用するには 10 段以上の分岐予測とさらに広範囲にわたる解析が必要であることを示した。

今後, 多様なプログラムに対する解析をおこなうとともに, スーパースカラ等の実行モデルとの比較をおこなっていく。

謝辞

本研究の一部は文部省科学研究費 (一般研究 (B) 課題番号 07458052「 大規模データパスプロセッサの研究 」) による。

参考文献

[1] 吉瀬, 中村, 金指, 田中　データフローグラフを用いた複数命令のブロック化　情処第 52 回, pp.6-81-82, 1996.