データフローグラフを用いた複数命令のブロック化

1L－5

吉瀬謙二[*1], 中村友洋[*1], 金指和幸[*2], 田中英彦[*1]
[*1]東京大学工学部, [*2]富士通 (株)

1　はじめに

既存のマイクロプロセッサはプログラムの一部から並列性を取り出しているに過ぎず, プログラムが持つ並列性を十分に利用できていない。

そのため, 我々は内在する並列性を効果的に利用できる, 新しいタイプのプロセッサの研究を行なっている。

本稿では, 並列性を解析する範囲 (命令バッファ) を大きくすることによって, 命令レベル並列度がどのように変化するか調査し以下の結果を得た。(1) プログラムには多くの並列性が存在する。(2) プログラムの広範囲にわたる解析により十分な並列度を得ることができる。

2　実行モデル

2.1　実行モデルの定義

命令レベル並列度の効果的な利用を可能とする実行モデルを, 図 1 に定義する。

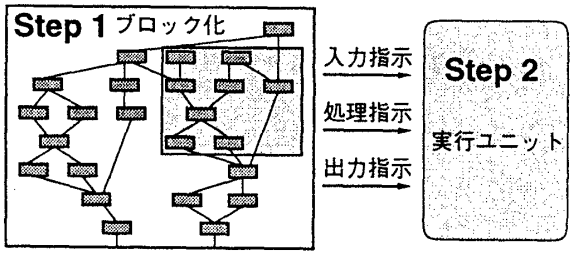

図 1: 実行モデルの定義

本実行モデルはブロック化と実行という 2 つのステップから成る。

ステップ 1. ブロック化

ブロック化ユニットは, データフロー解析を用い実行ユニットの実行単位となるブロックを取り出す。

(ブロック化の方針は後述)

ステップ 2. ブロックの実行

実行ユニットは, 分割されたブロックと必要な情報 (入力指示, 処理指示, 出力指示) を受けとり計算を実行する。

2.2　実行ユニットの構成要素と役割 (図 2 参照)

入力装置 は各演算装置がメモリ内のどのデータを使用するかという指示 (入力指示) により配線を構成する。

処理装置 は演算要素の組合わせ (処理指示) に従って, 演算器を構成し高さ h のブロックの演算を h サイクルで実行する。

出力装置 は演算結果をメモリのどこにしまうかという指示 (出力指示) に従って配線を構成する。

メモリ は演算の結果を保存し, 必要なデータを供給する。

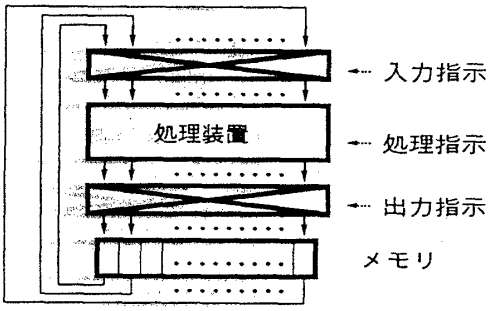

図 2: 実行ユニット構成

3　シミュレーション

先に定義した実行モデルの理想的実行における性能を, シミュレーションにより評価する。理想的実行は, ブロック化に要する時間, 入力装置, 出力装置, メモリ参照によるオーバヘッドを無視し, 処理装置で演算実行を行なっている時間のみを考えた実行時間である。また以下の仮定を設ける。

3.1　シミュレーションにおける仮定

(1) 高速なメモリが容量無制限で使用可能
(2) 分岐予測 100%での投機実行と理想的なフェッチ機構
(3) 全ての命令のレイテンシーは 1 クロックサイクル

3.2　シミュレーションでの実行モデル (図 3 参照)

1. RISC プロセッサの実行コードから連続した n 個の命令をフェッチする。ただし 解析範囲 n は後の評価対象とする。(n のサイズをプログラム全体とすることで RISC コードによる制限を排除できる。)

2. フェッチされた n 命令においてデータフロー解析をおこなう。仮定 (1) より, 真の依存関係以外のデータ依存関係を解消する。仮定 (2) より, データフローグラフは制御依存をもたない。

3. 得られたデータフローグラフから処理装置の実行単位となるブロックを作成し, 実行ユニットでブロック実行に必要となったクロック数を測定する。

Separating Dataflow Graph into blocks
Kenji KISE[*1], Tomohiro NAKAMURA[*1],
Kazuyuki KANAZASHI[*2], Hidehiko TANAKA[*1]
[*1]Faculty of Engineering, University of Tokyo
[*2]Fujitsu Limited

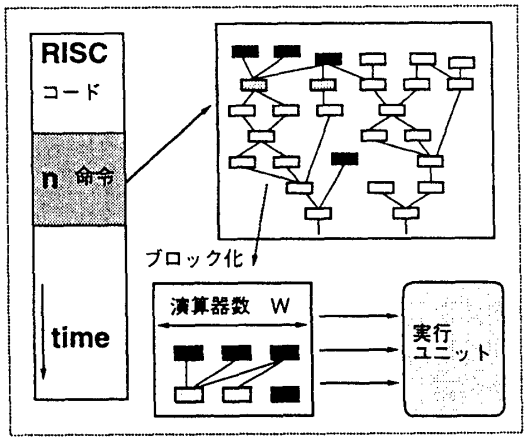

図 3: シミュレーションでの実行モデル

3.3 ブロック化の方針 (図 4 参照)

演算器数 (ブロックの幅) w 高さ h とするとブロック内に最大 $w \times h$ の命令を入れることができる。ブロック内の命令数を $w \times h$ に近付ければ全体の実行時間を短縮することができるので, ブロック化ではブロックの隙間をなくすように命令を配置する。

演算器数 w 以上の並列度が存在する時は実行可能な命令から w 個の命令を選択する必要がある。その命令を実行することによって多くの依存関係が解消される命令の優先度を高くして, 命令の選択をおこなう。

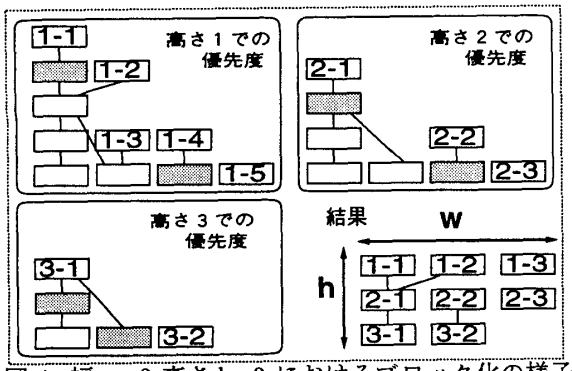

図 4: 幅 w=3 高さ h=3 におけるブロック化の様子

4 実験結果

図 5,6 に実験の結果を示す。プログラムは SPECint92 より compress, espresso を使い, 最初の 100 万命令のトレースデータを評価の対象とした。図は演算器数を変化させたときの, 解析範囲 n に対する並列度の変化を示す。また参考として演算器を無限個, 解析範囲を無限大としたときの並列度 (最大並列度) を破線で示した。

compress は逐次実行の影響が強く比較的小さい解析範囲で最大並列度に対して高い割合の並列度を抽出できているが, espresso では並列実行できる部分がプログラムの広い範囲に分布しているため解析範囲が 10 万以下の場合には十分な並列度を得ることができていない。

プログラム全体からデータフロー解析を行なった結果 2 つのプログラムとも, 解析範囲を 100 万命令に設定しプログラム全体から解析を行なうことで, 演算器数を最大並列度と同程度に制限しても, 最大並列度の 99%以上の性能を獲得できることがわかった。

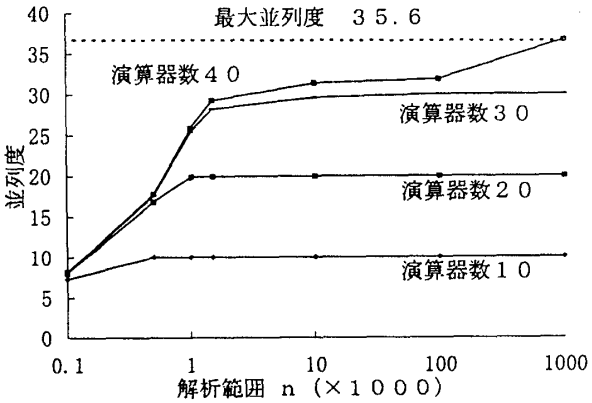

図 5: compress 100 万命令の実行

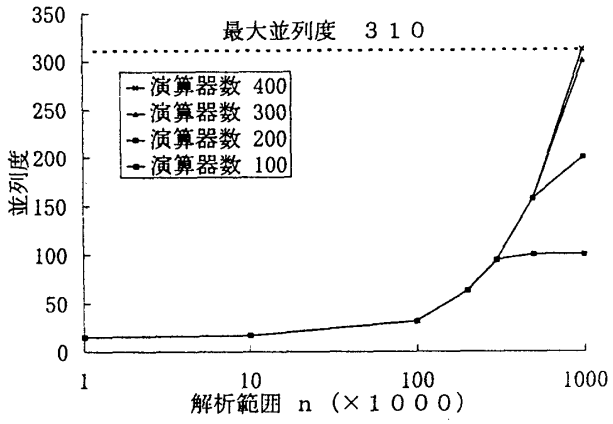

図 6: espresso 100 万命令の実行

5 まとめ

プログラムに内在する命令レベル並列度をどれだけ利用できるかは, 命令の依存関係を解析する範囲 (命令バッファ) の大きさに依存する。

既存のマイクロプロセッサの延長路線では, 同時にフェッチできる命令数, 命令バッファのエントリ等がこの解析範囲を制限するため, これらのエントリ数を大幅に増加させない限り, 並列度を効果的に利用することはできない。

今後, 内在する並列度を効果的に利用できる, 新しいタイプのプロセッサ方式に関し検討をおこなっていく。

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] 田中英彦 ここいらで, 計算機アーキテクチャを再考しよう 情処研報, ARC-108-6