式の分割による並列アルゴリズムESHのスーパースカラ・プロセッサでの評価

4D－2　金丸 弘樹　宇野 総一　西川 英史　岩根 雅彦
九州工業大学 工学部

1. はじめに

式の分割による並列スケジューリングアルゴリズムESH(Expression Scheduling Heuristic)は、並列計算機などへの適用により速度向上が得られる。[1] 今回、これをアウト・オブ・オーダー実行を行うスーパースカラ・プロセッサに適用したので、その効果についてここで報告する。

2. ESH

同一優先順位の演算子で構成される多項式は、結合則・可換則により右辺の式の評価順序が拘束されない。そこで、変数が参照可能となる時間を考慮して二項演算を生成していけば、多項式の演算がもっとも早く完了する可能性が高い。ESHでは、まず同一優先順位を持つ演算を1つのノードとしてプログラムを分割し、依存解析を行いタスクグラフを生成する。タスクグラフ中には、同一優先順位の演算をもつ多項式(未決定ノード)とすでに二項演算になっている式(決定ノード)が含まれる。未決定ノードから決定ノードへの変換は、式の右辺に現れる変数の定義時間が早いものから順に二項演算として抽出することで行う。 すべてのノードが決定ノードに変換されたらスケジューリングは完了する。

3. 実験

3.1 環境

今回、実験用のプロセッサとしてPowerPC604[3][4]を用意した。これは、浮動小数点、分岐処理、ロード・ストアユニットをそれぞれ1つと3つの整数ユニット(シングルサイクル整数ユニット×2、マルチサイクル整数ユニット×1)をもつスーパースカラ・プロセッサである。構造は、図1のようになっており、4命令同時のフェッチが可能で、各実行ユニットはそれぞれ独立にアウト・オブ・オーダーで実行を行う。ディスパッチ・ユニットは各実行ユニットに最大4命令を同時に発行できるが、命令順に発行を行うため実行ユニットが使用中であるとストールする。このため、各ユニットには、リザベーションステーションが2つ用意されている。実行ユニットの結果は、レジスタに書き込まれるのを待つことなくリネームバッファによって完了と同時に使用できる。レジスタファイルへの書き込みは、ライトバックで命令が発行された順に行われる。図2にPowerPC604の主な命令の命令レイテンシを示す。

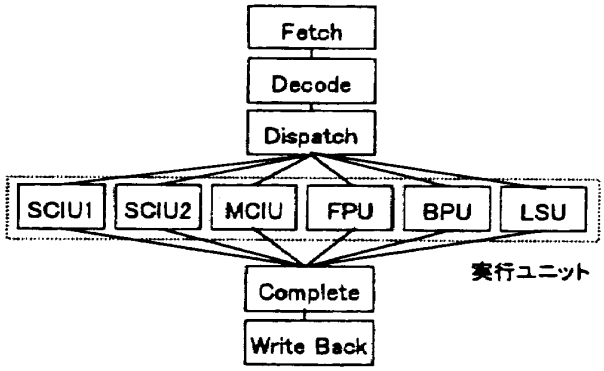

図 1 PowerPC604 構成図

命令	add	subf	mullw	lwz	stw
使用するユニット	SCIU	SCIU	MSU	LSU	LSU
レイテンシ(cycle)	1	1	3	3	3

図 2 PowerPC604の主な命令レイテンシ

3.2 評価

```
int module1(void) {
    int p,q,r,s,t,u;

    p = p * q;
    r = r - s;
    t = p + q + r + s;
    q = q + t;
    s = s * t;
    u = p - q - r - s;
}
```

(a) Module 1

```
int module2(void) {
    int x1,x2,x3,x4,t;

    x1 = ( x1 +x2 +x3 -x4) * t;
    x2 = ( x1 +x2 -x3 -x4) * t;
    x3 = ( x1 -x2 +x3 +x4) * t;
    x4 = (-x1 +x2 +x3 +x4) * t;
}
```

(b) Module 2

図 3 評価に使用したソースコード

図3に示すようなソースコードを用いてベンチマークテストを行った。なお、図3(b)においては、演算をすべて浮動小数点演算に置き換えたものについても評価している。(これをModule3とする)計測はPowerPC604内蔵のパソコン上で行い、先のコードを市販の C/C++

The Evaluation of Expression Scheduling Heuristic (ESH) Algorithm on Super Scalar Processor
Hiroki Kanamaru, Souichi Uno, Eiji Nishikawa, Masahiko Iwane
Department of Electrical Engineering, Faculty of Engineering, Kyusyu Institute of Technology

コンパイラ [5]（以下MCC）によりコンパイルして出力されたコードと、人手によるESHのスケジューリングを行ったコードとの実行速度の比較を行った。なお、このESHを適用するにあたって、並列計算機などでは考慮していたプロセッサ間の通信時間を省略した。これは、スーパースカラ・プロセッサが機能ユニット間の値の受け渡しに時間を必要としないためである。さらに、比較の簡単のためESHのコードについてはレジスタの割り当てなどを可能な限りMCCのコードに合わせている。また、MCCにおいては、レベル4の最適化を行った。[5]

	Module1	Module2	Module3
MCC	26.33	32.39	60.64
ESH	24.31	28.35	54.59

（単位はサイクル）

図 4 各モジュールの実行時間

実験の結果を図4に示す。なお、表中の数字は、計測時間から算出した平均のサイクル数である。

4. 考察

図3(a)のコードの一部をESH、MCCそれぞれについて図5に示す。ここで、図5の点線部は、図3(a)のコードにおける最初の多項式に関する部分である。また、これらのコードの実行過程を図6に示す。この実行過程は、プロセッサの仕様にしたがってシミュレーションを行ったものである。なお、命令レイテンシ以上に実行ユニットが使用されているのは、命令のデータ依存解決のためである。

この図より、ESHコードの実行ユニットの使用時間が、MCCのコードに比べて少なめなことがわかる。特に、最後の乗算命令(mullw)の参照可能時間は、MCCコードにおいては16サイクル目、ESHコードでは15サイクル目と実行サイクルが早くなっていることが確認できる。

```
4:    mullw  r3,r5,r3        mullw  r3,r5,r3
5:    lwz    r7,76(sp)       lwz    r7,76(sp)
6:    subf   r6,r7,r6        subf   r6,r7,r6
7:    stw    r3,64(sp)       add    r8,r5,r7
8:    add    r8,r6,r3        stw    r3,64(sp)
9:    add    r8,r8,r7        add    r8,r8,r3
10:   stw    r6,72(sp)       stw    r6,72(sp)
11:   add    r8,r8,r5        add    r8,r6,r8
12:   mullw  r7,r8,r7        mullw  r7,r8,r7
13:   add    r5,r8,r5        add    r5,r8,r5
```

(a) MCC CODE　　(b) ESH CODE

図 5 最適化されたそれぞれのコード（一部）

これは、図3(a)のコード中、最初の多項式において、変数Pの定義に時間がかかることを、MCCが認識できていないためと考えられる。このような遅延の回避は、ESHが変数の定義時間を考慮しながら2項演算を生成していくために得られる成果である。しかし、図1(a)ソースコードの実測値においては、ESHのコードに比べてMCCのコードが、およそ2サイクル遅延する結果が出ている。誤差であるとも考えられるが、ESHの効果以外のスーパースカラ・プロセッサに対するスケジューラの効果による影響も考えられる。

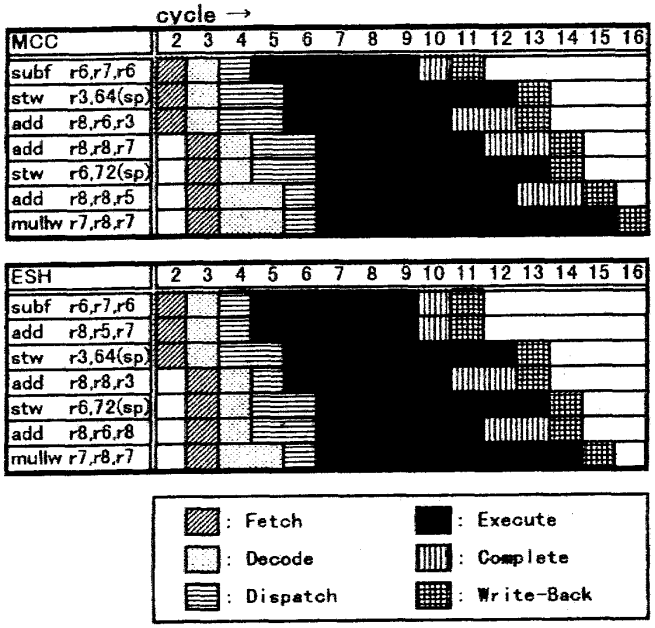

図 6 図4の点線部の実行状態

5. むすび

今回の実験により、アウト・オブ・オーダー実行のスーパースカラ・プロセッサにおいてもESHの効果が期待できることがわかった。しかし、スーパースカラにより、ESH以外による効果が生じる可能性もある。今後、ESHアルゴリズムとスーパースカラ・プロセッサに対するスケジューリングのより良い統合などがあげられる。

参考文献

[1] 岩根 他,「式の分割による並列化アルゴリズムESHとその評価」,情報処理学会論文誌 vol.38, No.9

[2] M.Johnson(村上和彰訳),「スーパースカラ・プロセッサ」, 日経BP , 1994

[3] 「PowerPC604 User's Manual」, Motorola , IBM , 1995

[4] 「PowerPC Microprocessor Family: The Programming Environments」, Motolora , IBM , 1994

[5] 「Motorola C/C++ SDK User's Guide – PowerPC Edition for MacOS·DR4.0－」, Motorola , 1996