2009 年 9 月 7 日 高性能コンピューティング学講座ゼミ合宿発表資料

共有資源の競合に着目した CMP 向け実行フェーズスケジューリング手法についての研究

発表者： 0853009 久米正人 (近藤研究室)

1 イントロダクション (from 中間発表)

現在主流となっているプロセッサのアーキテクチャの 1 つに，複数のプロセッサコアを 1 つのチップ上に搭載したチップマルチプロセッサ (CMP) と呼ばれるものがある．CMP に搭載された複数のコアで並列処理あるいは並行処理を行うことによって，クロック周波数の向上に頼らず高い演算性能を達成することができる．

演算性能の向上を目指し，CMP に搭載されるコア数は年々増加傾向にある．ただし，キャッシュやバスなどの資源は複数のコアで共有することが一般的である．そのため，多くのコアを搭載した CMP においては，これら共有資源の競合が起こりやすいといえる．しかし，共有資源の競合が生じると，チップ全体の性能が低下し，期待される演算性能が十分に得られない．このため，CMP の持つ演算性能を最大限に引き出すためには，共有資源の競合をできる限り回避することが重要である．

このため，共有資源の競合に関する研究は従来より行われてきた．しかし，さらに細粒度の視点から見ると，一般的には共有資源の要求率は 1 つのプロセス内でも変化する．このことを考慮すると，より細粒度な実行フェーズ単位でスケジューリングを行うことで，より効果的な実行が行えると考えられる．

そこで本研究では，このような共有資源の競合に着目しつつ，プロセスを実行フェーズで分割した細粒度でのスケジューリング手法を提案する．

2 研究の手法

図 1 に本提案手法によるプロセス実行の概要を示す．図は，プロセス#0〜#3 が 2 つのコア上で実行される様子を示したものである．ここで，1 つのコア上では 1 つのプロセスしか同時に実行されないものとする．全てのプロセスの動きの制御を司る **Manipulator** という親プロセスが存在する．Manipulator が，Polling Interval ごとにその時点でのパフォーマンスカウンタを得る．次に，Manipulator は得られたパフォーマンスカウンタのデータから，それぞれのプロセスの次のフェーズでの要求率の予測を行う．予測で求められた要求率をもとに，なるべく 100%になるようなプロセスの組み合わせを選択する．そして，選択されたプロセスを実行させ，その他のプロセスは停止状態にする．これを繰り返して，本提案手法を実現する．なお，Polling Interval ごとに分割したプロセスの単位のことを**実行フェーズ**と呼ぶ．

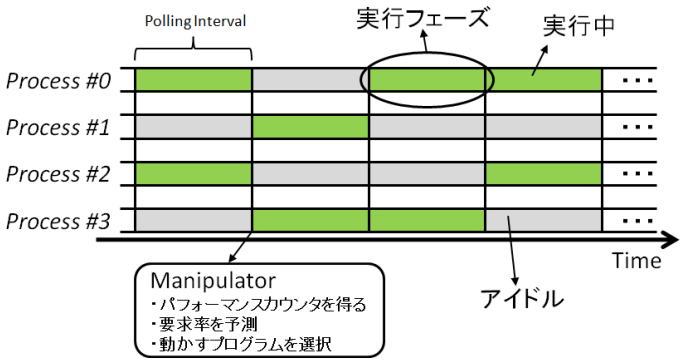

図 1: 本提案手法によるプロセス実行の概要

3 最近の実験

この実験は，提案手法が従来手法に比べて優位な結果を示すケースを見つけるための実験である．

3.1 実験に使うプログラム

配列アクセスの間隔をラインサイズの 64 バイトよりも十分大きな 256 とし，プログラム prg0 を組んだ．これを実験で扱うプログラムとする．このプログラムは，実行するとキャッシュミス，ヒットを一定時間ごとに繰り返す．prg0 のソースコードを以下 prg0.c に示す．

prg0.c（細部省略）

```
 1 int i, j, k;
 2 char a = 0, *U;
 3 U = (char *)malloc(268435456 * sizeof(char));
 4 for(i = 0; i < 268435456; i++) U[i] = 1;
 5 for(k = 0; k < 6; k++){
 6   for(j = 0; j < 800; j++)
 7     for(i = 0; i < 268435456; j += 256)
 8       a += U[i]; // cache miss
 9   for(j = 0; j < 58; j++)
10     for(i = 0; i < 268435456; i++)
11       a += U[i]; // cache hit
12 }
13 printf("solution = %d\n", a);
14 free(U);
```

このプログラム prg0 は，後述する Core i7 940 (k008) の環境において，キャッシュヒット部分とキャッシュミス部分が両方とも 10 秒になるように作られている．単独で実行すると，このセットを 6 回繰り返し，約 120 秒で終了する．

3.2 実験手順

Core i7 940 (k008) 上にて 8 個の prg0 を同時に動作させ計測を 10 回行い，その平均を結果とした．

3.3 予測の条件，プログラム選択

予測の条件は，以下のとおりとする．

・条件（カウンタ値は L2 キャッシュミス値を使用）
(a) 過去 15 個のカウンタのデータから最小二乗法を用いて近似式を求め，次のフェーズのカウンタ値を予測
(b)1 個前のデータをそのまま使用

得られた予測値よりどのように動かすプログラムを選択するかについては,（予測値）/（あり得る最大ミス回数）=占有率とし，各コアからプログラムを 1 つずつ選んでなるべく占有率を 100%になるように選択する．残りのプログラムは止める．よって，クアッドコアなので同時に最大 4 つのプログラムが動く．

3.4 実行時間に関する実験

この実験において，実行時間とは 8 つの prg0 のうち，最も遅かったものの実行時間とした．この実験結果を表 1 に示す．

表 1: 実験結果 (実行時間)

条件	実行時間 (sec.)	標準偏差（σ）
単純 8 個	508.15	9.59
(a)	461.61	5.25
(b)	458.69	6.53

prg0 においては，ミスし続けるかヒットし続けるかの 2 パターンしかない．そのため，要求率は 100%か 0%のどちらかである．よって 1 つ前のデータを使う (b) が最も予測の精度が高くなる．

予測の精度が高くなると，実行時間が短縮されるという結果から，予測の精度を高めることに重点を置いて今後の研究を続ける．

3.5 IPC に関する実験

この実験においては，8 つの prg0 のうち，1 つが終わるまでの間で IPC を測定した．IPC は以下，式 (1) で算出した．

$$IPC = \frac{Instructions\_retired}{Elapsed\_time} \times \frac{1}{2.93 \times 10^9} \quad (1)$$

この実験結果を表 2 に示す．

表 2: 実験結果 (IPC)

条件	IPC	標準偏差（σ）
単純 8 個	0.03010	0.0011
(a)	0.02950	0.0008
(b)	0.02954	0.0012

IPC でみたときには，実行時間に関する実験結果と異なる．これは，実験方法に問題があったために正当な IPC を測定できなかったと考えられる（8 個全てが終わるまでの時間で IPC を測定することが望ましい）．

3.6 参考: Core i7 940 の情報

参考として，Core i7 940 に関する情報を以下，表 3 に示す．

表 3: Core i7 940 のキャッシュの情報 [1]

コア数	4
動作周波数	2.93GHz
L1-D	Private, 32KB, 8-way assoc.，64-byte line
L1-I	Private, 32KB, 4-way assoc.，64-byte line
L2	Shared, 256KB, 8-way assoc.，64-byte line
L3	Shared, 8MB, 16-way assoc.，64-byte line

4 今後の予定

- SPEC などのベンチマークで測定を行う
- 実行の公平性を測定する
- 予測値と実測値の差（予測の精度）を測定する
- スケジューリング機構（Manipulator など）をもっと作りこむ

5 まとめ

今回は，提案手法が従来手法に比べて優位な結果を示すケースを見つけるための実験を行った．これより得られた知見として，**予測の精度が高くなると，実行時間が短縮される**ことが挙げられ，本研究においては予測の精度が重要である．

参考文献

[1] 大原雄介, 【特集】完全版!! 「Core i7」極限検証 - 内部アーキテクチャ解析編, http://journal.mycom.co.jp/special/2008/nehalem02/menu.html, 毎日コミュニケーションズ, 2008.