2007－ARC－174 (15)
2007／8／1

CMPにおけるリソース競合に着目した性能の解析とモデリング

佐々木 広† 近藤正章† 中村 宏†

近年，汎用マイクロプロセッサのアーキテクチャとして主流となっているチップマルチプロセッサ（CMP）では，複数のコアがL2キャッシュやメモリバスなどを共有するのが一般的であるが，それらの共有リソースが同時にアクセスされるとリソース競合が発生し，性能が大幅に低下してしまう．この競合に対処し，効率的な実行を提供するためには，競合による性能低下の影響を解析し，またモデリングすることが重要であると考えられる．本稿では，CMPにおいて独立のプログラムを複数のコア上で同時に実行させる場合を対象とし，リソース競合が性能に与える影響について実機のマシンを用いて解析を行ない，統計的な手法によりモデリングを行なう．本稿で作成したモデルにより，正確に競合が性能に与える影響をモデリングできることがわかった．

Analysis and Modeling of Multiprogramming Performance of Chip MultiProcessors focusing on Resource Contentions

Hiroshi Sasaki,† Masaaki Kondo† and Hiroshi Nakamura†

In single chip multiprocessors (CMPs), multiple processor cores usually share several hardware resources such as cache memories, memory buses, and main memory banks. Performance degrades significantly if those resources are accessed simultaneously and resource contention occurs. In this paper, we analyze the performance impact caused by resource contentions and construct a performance prediction model using hardware performance counters. In order to construct the model, we use a statistical method to find relationships between the performance impact and performance counters. We show that the constructed model can predict the performance impact very accurately.

1. はじめに

複数のプロセッサコアを1チップに搭載するチップマルチプロセッサ（Chip MultiProcessor: CMP）は，シングルタスクの並列処理，あるいは複数タスクの並行処理により高い性能が得られることが期待されている．しかしCMPではリソース有効活用の観点から，複数のコアがL2キャッシュやメモリバス，主記憶であるDRAMバンクなどを共有するのが通常であり，それらの共有リソースが同時にアクセスされるとリソース競合が発生し，性能が大幅に低下してしまうことがある．

従来より，キャッシュやメモリバスだけでなく，命令キューや演算器などコア上の大半のリソースを共有するため，リソース競合が性能に及ぼす影響が非常に大きいSimultaneous Multi-Threading（SMT）プロセッサでは，リソースの使用率を考慮したスレッドの実行制御手法[7),8)]や，競合のモデル化に関する研究が行なわれてきた[9)]．一方，CMPにおいても，特に共有するキャッシュ上での競合性ミスによる性能低下を防ぐために，キャッシュを論理的なパーティションに分割する手法[1)~3)]や，キャッシュ競合の影響を予測するモデル[6)]が提案されている．

このように，CMPにおいてはキャッシュ上での競合による性能低下に対し，その対処手法などが提案されてはいるものの，他の共有リソースにおける競合などに関し，十分に検討は行なわれていない．近年では，プロセッサと主記憶の性能差が拡大しており，メモリバスやDRAMバンクなどで競合が発生すると，より主記憶アクセスのレイテンシが増大してしまうため，それら共有リソース上での競合も性能への影響が非常に大きいと考えられる．また，今後，より多くのプロセッサコアが1チップに搭載されるようになると，これまで以上に競合の影響が深刻になると予想される．したがって，競合が性能に与える影響に関し，より包括的な解析や，さらには競合の影響をモデリングしつつ，そのモデルを基に効率的な実行を提供することが今後必要になると考えられる．

本稿では，CMPにおいて独立のプログラムを複数のコア上で同時に実行させる場合を対象とし，リソース競合が性能に与える影響について，実機のマシンを用いつつ，その解析とモデリングを行なう．モデリングに際しては，共有リソースが複数存在し，競合の影響はプログラムの組合わせに応じて多様な振舞いを示すことから，競合が性能に及ぼす影響は非常に複雑で

† 東京大学 先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

あることが予想され，計算機の振舞いを定性的に理解しつつ，モデルを立てるのは非常に困難であると考えられる．そこで本稿では，文献[16]を基にした統計的学習手法により，モデルを作成する．本手法は，種々のプログラムを実行し，その際のハードウェアカウンタの値を取得することで，その値と競合による性能低下の関係を統計的に学習しモデルを作成するものである．これにより，例えば対象のマシンが変わった場合でも，同じように学習を行なうことで，モデルを作成することが可能となる．

本稿では，複数のCMPを搭載した実機により，まず実際の競合による性能への影響を解析する．次に統計的モデリング手法について述べ，同じく実機のマシンを対象に競合の影響を実際にモデリングした結果を示す．

2. 関連研究

メモリ階層だけでなく，演算器などのリソースも共有するSimultaneous Multi-Threading (SMT)プロセッサでは，リソース競合による性能の解析やリソース競合のモデル化に関する研究が広く行われている[13)12)9)]．

また，従来より，CMPの共有キャッシュのリソース競合を削減するためにキャッシュを分割し，性能やFairnessを向上させる手法が提案されている．文献[1)]では，共有L2キャッシュを分割することでキャッシュミスの削減，あるいはFairnessの向上を狙う手法が提案されている．文献[2)]では動的にキャッシュを分割し，utilityに基づいて各アプリケーションに領域を割り当てる手法を提案している．また，文献[3)]キャッシュを分割する際の種々のポリシーについて検討が行なわれている．共用キャッシュをOSから制御するための拡張について[5)]で提案されている．さらに，文献[6)]において，キャッシュ競合の影響を予測するモデルが提案されている．

また，プロセッサのキャッシュサイズやウェイ数，演算器の数など広大な設計パラメータが性能にどのような影響を与えるかを統計的な手法を用いることによってモデリングする研究も行われている[15)14)]．

3. 評価環境

本稿では，リソース競合に着目した性能の解析とモデリングを行なうために，*Intel Core2 Extreme QX6700*（以下，Core2-QX6700），および*AMD Athlon 64X2 3800+*（以下，Athlon-64X2）を搭載したPCを用いる．評価に用いたマシンの仕様を表1に示す．

Core2-QX6700は，4MBのL2キャッシュを共有するデュアルコアプロセッサを，1パッケージ内に2個搭載した4コアを持つCMPであり，例えば2つのプロセスを同時に実行する場合，どの2コア上でプロセスを実行するするかにより，L2キャッシュを共有する場合と非共有の場合の両者を評価することが可能である．また，メモリバス（FSB）以下のメモリ階層は共有リソースとなっている．本稿では，2プロセスを同時に実行する場合を評価することとし，キャッシュ共有，非共有の両者について評価を行なう．またAthlon-64X2は，コア毎に512KBのL2キャッシュを持つデュアルコアプロセッサであり，メモリバス以下が共有リソースである．

表1 評価に用いたマシンの仕様

	Core2-QX6700	Athlon-64X2
CPU	Intel Core2 Extreme QX6700	AMD Athlon 64X2 3800+
- 周波数	2.66GHz	2.0GHz
- L2 サイズ	4MB × 2	512KB × 2
- メモリバス	FSB 1066MHz	5.3GB/s
M/B	Intel D975XBX2	ASUS M2A-VM
主記憶	DDR2-SDRAM	DDR2-SDRAM
- 周波数	667MHz (PC667)	667MHz (PC667)
- サイズ	1GB	1GB

OSにはLinux-2.6.18を用い，カウンタ値の取得にはPAPI (Performance Application Programming Interface)[11)10)]を用いる．Core2-QX6700は33種類のパフォーマンスカウンタを持ち，コアあたり同時に2つのカウンタ値を，またAthlon-64X2は29種類のパフォーマンスカウンタを持ちコアあたり同時に4つのカウンタ値を取得可能である．

4. CMPにおけるリソース競合による性能低下の解析

リソース競合が性能に与える影響を調べるため，Core2-QX6700を搭載したマシンを用いて評価を行なった．前節で述べたように，Core2-QX6700ではL2キャッシュを共有する場合と共有しない場合を評価することが可能であるため，ここでは両者を評価する．

図1にL2キャッシュを共有しない場合（non-L2-shared），また図2にL2キャッシュを共有する場合（L2-shared）の，SPEC2000ベンチマークのほぼすべて，およびOldenベンチマークの中の6つのプログラム群の全組み合わせにおける，2プロセスを同時に実行した場合の各プログラムのMulti-Programmed Speedupを示す．Multi-Programmed Speedupは，それぞれのプロセスを単独で実行した場合に比べて，複数プロセスで同時に実行した際の性能比を，全プロセス（本評価では2プロセス）で合計したものである．図は，対象プログラム（X軸）と，全プログラムとの組み合わせにおけるMulti-Programmed Speedupを，“箱ひげグラフ（box-and-whisker plot）”で表わしており，各対象プログラム毎に，箱部分が*IQR (Inter-Quartile Range)*と呼ばれる中央50%範囲内のデータを，またひげ部分の線が，箱の上下の終端から$1.5 \times IQR$範囲内のデータを示している．その範囲外のデータは，は

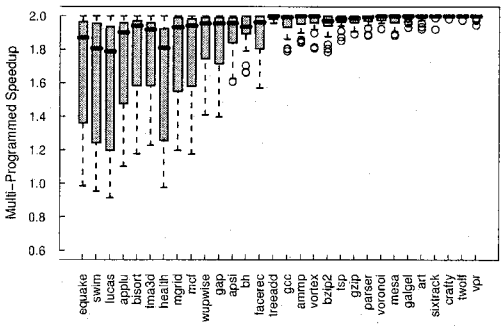

図 1　non-L2-shared の場合の Multi-programmed Speedup

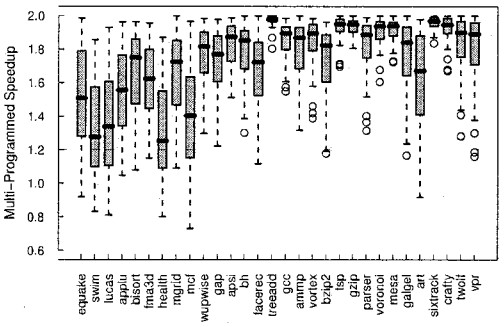

図 2　L2-shared の場合の Multi-programmed Speedup

ずれ値としてデータポイントがプロットされている．また箱内部の横線は，中央値を示している．なお，対象プログラムは，左から単独で実行した場合に L2 ミス率が高い順に並んでいる．

図 1 の non-L2-shared の場合を見ると，対象プログラムの L2 キャッシュミス率が高い場合には，Multi-Programmed Speedup が大きく低下してしまうことがわかる．本評価では，独立に実行可能な 2 つのプログラムを実行しているため，この性能低下はリソース競合によるものであり，キャッシュミス率がある程度高いプログラムでは，リソース競合の影響が大きいことがわかる．またこの場合，L2 キャッシュは非共有であるため，これらの性能低下がメモリバス，およびメモリバンクの競合により生じていることがわかる．

また，図 2 の L2-shared の場合を見ると，non-L2-shared の場合に比べて Multi-Programmed Speedup の低下が大きいことがわかる．これは，メモリバス，メモリバンクでの競合に加えて，L2 キャッシュで競合が生じ，L2 キャッシュミス回数そのものが増加してしまうためである．特に，Multi-Programmed Speedup の値が 1 以下，つまり同時に実行するよりも，逐次的に実行した方が性能が良い場合もいくつかあることから，L2 キャッシュを共有する場合には，競合の影響が深刻であることがわかる．

次に，図 1 と図 2 を比較すると，図 1 の non-L2-shared の場合は，左からほぼ順に IQR が広い順に並んでおり，また中央値も右にいくにつれて 2.0 に近づいていることがわかる．一方，図 2 の L2-shared の場合は左の方に広い IQR のアプリケーションが偏ってはいるが，galgel，art や vpr など右の方にも広い IQR を持つアプリケーションもあり，その傾向は一様ではない．また，ひげ部分の線が non-L2-shared と比べて長く，様々な性能低下率を持つ組み合わせがあることが一目でわかる．このことから，non-L2-shared の場合には，メモリバスやメモリバンクでリソース競合は，L2 キャッシュミス率と大きな相関があると思われる．一方，L2-shared の場合における要因はより複雑で，コアあたりの L2 キャッシュのサイズが小さくなることやアクセスパターンに依存するコンフリクトの影響による性能低下が大きく，L2 キャッシュミス率に着目しているだけでは正確な性能予測ができないことなどがわかる．

5. モデリング

本節では，リソース競合に着目した性能のモデリングのための統計手法，および学習のためのプロファイリングについて述べる．前節で述べたように，CMP はその構成や，同時に実行するプログラムの組み合わせによって，単独で実行した場合に比べてその性能が大きく異なることがあり，その振る舞いは非常に複雑である．したがって，リソース競合が性能に与える影響を，プログラムの特徴や，CMP の構成などから定性的に予測することは非常に難しいと考えられる．我々は，このような定性的に振る舞いを理解することが困難な事象に対して，重回帰分析を用いた統計的な学習を用いることによってモデリングを行う手法を提案しており[16]，以下に述べるように，この手法で用いた統計手法を改良することによって，CMP における性能のモデリングを行う．

5.1 重回帰分析

重回帰分析は多変量解析の一つで，従属変数 (y) と独立変数 (xi) との関係を調べ，関係式を導くことによって独立変数から従属変数の予測を行う手法であり，その関係式は一般的に式 (1) で表される．ここで，β_0 は定数項，β_i はそれぞれの独立変数にかける重み（偏回帰係数），e は誤差項である．

$$f(y) = \beta g(x) + e = \beta_0 + \sum_{i=1}^{p} \beta_i g_i(x_i) + e \quad (1)$$

5.1.1 重回帰分析における交互作用

本モデリング手法では，単独で実行した際に比べて複数プロセスで同時に実行した際の性能比を従属変数，それぞれのコアで実行しているプログラムの振る舞いを示すパフォーマンスカウンタの値を独立変数とする．

ここで，CMP において複数のコアで同時にプロセスを実行している際にはリソース競合が起こり，その影響によって性能が変化することは述べたが，同時にそれぞれのカウンタ値にも影響を与えることが考えられる．例えば，あるコアで L2 キャッシュミスが頻発した場合に，他のコアは競合によって性能が低下し，その結果として L2 キャッシュミス回数にも変化が出てくるような場合がそれにあたる．通常の重回帰分析ではある独立変数 x_1 の値が変化しようがしまいが，他の独立変数 x_2 と従属変数 y の関係は一定であると仮定されているため，上記のような事象を表すことができない．これは独立変数間の交互作用の問題と呼ばれ，この交互作用をモデルに組み込むためには，独立変数同士の積を新たな独立変数 $x_3 = x_1 x_2$ として投入する手法が用いられる事が多く，本手法でもコア間の競合による影響をモデル化するためにこの手法を用いる．

5.1.2 Restricted Cubic Splines (RCS)

基本的な重回帰分析では，従属変数はすべての説明変数に対して線形に振る舞うことを仮定する．この仮定によって表現可能な関係は非常に制限されているため，非線形性を取り入れた様々な手法が提案されている．CMP における性能のモデリングを行う際にも，説明変数のいくつかの領域によって，それぞれ従属変数と異なる関係を持つ非線形な関係が考えられるため，複雑なモデル化が可能な Restricted Cubic Splines (RCS) と呼ばれる，制限付きの三次スプライン関数を用いたモデリングを導入する．スプライン関数は区分的多項式であり，*knots*（結び目）と呼ばれる複数の端点によって分割されている．RCS は *knots* において傾きと曲率が等しく，また曲線の両端が直線であるという制限を持ち，k 個の *knots*☆ を持つ RCS は以下の式 (2)〜(5) で表される．

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{k-1} x_{k-1} \quad (2)$$

$$x_1 = x \quad (3)$$

$$\begin{aligned} x_{j+1} = {} & (x - t_j)_+^3 \\ & -(x - t_{k-1})_+^3 (t_k - t_j)/(t_k - t_{k-1}) \\ & +(x - t_k)_+^3 (t_{k-1} - t_j)/(t_k - t_{k-1}) \end{aligned} \quad (4)$$

$$(u)_+ = \begin{cases} u: & u > 0 \\ 0: & u < 0 \end{cases} \quad (5)$$

5.2 学習のためのデータサンプリング

本節では，3 節に示した 2 つの CPU における 3 つのプラットフォーム（Core2-QX6700 は L2 キャッシュ共有と非共有の 2 つ）において，統計的な学習による性能のモデリングを行うために必要なデータのプロファイリング手法について述べる．それぞれのプラットフォームにおいて，様々な未知のアプリケーションを同時に実行した場合における性能の低下率を正確に予測するため，学習には様々な特徴を持つアプリケーションを広く用い，大量のサンプルデータを取得する必要がある．

本モデリングにおいては，従属変数 (y) を単独で実行した際に比べて複数プロセスで同時に実行した際の性能比，説明変数 (x_i) をそれぞれのコアにおけるパフォーマンスカウンタの値としている．学習に用いたアプリケーションの組み合わせ数は 30 個のアプリケーションから重複を許した 2 つを選ぶので，${}_{30}C_2 + 30 = 465$ 通りである．また，これらのアプリケーションは振る舞いが最初から最後までで一定でないものがほとんどであり，様々なフェーズに分かれている．このようなプログラム内の細粒度の振る舞いを学習に用い，モデルに反映させるために，それぞれのコアでパフォーマンスカウンタの値を一定間隔ごとにサンプリングし，それぞれを異なる学習用のサンプルデータとする．サンプリングの際には，2 つのプログラムを同時に異なるコアで実行しはじめ，30 ms ごとにハードウェアのタイマ割り込みによってシグナルを送り，その都度パフォーマンスカウンタの値を取得する．これらのアプリケーションはそれぞれ長さが異なっており，特定のアプリケーションの影響が大きくなることを防ぐため，それぞれの組み合わせにおいて 400 サンプルを取得した時点（約 12 秒）でサンプリングを終了とする．また，性能の低下率を算出するために，単独で実行した際の性能が必要となるため，同じように 30 ms 間隔でサンプリングを行う．

5.3 予測精度の評価について

得られたデータを用いてモデルを構築した後に，そのモデルの精度について評価を行い，モデリング手法の有効性について議論を行う．この評価には leave-one-out cross validation 法と呼ばれる手法を用いる．これは，全サンプルから 1 つのサンプルを除き，残りのサンプルでモデルを構築したときに，あらかじめ抜いたサンプルを正確に予測することができるかを検定する手法である．本稿では，全サンプルからある 1 つのアプリケーションを含むすべての組み合わせのサンプルを除き，モデルの精度について評価する．

6. モデリングおよび評価

前節で述べたようにそれぞれのプラットフォームにおいてサンプリングしたデータをもとに，統計的な学習を用いてモデリングを行う．モデリングに際して，例えば本手法によって構築されたモデルを OS が用いて動的に最適化を行うこと場合を想定し，用いるカウンタの個数はそれぞれのプラットフォームで同時に取得できるカウンタの個数とした（Core2-QX6700 はそれぞれ 2 つ，Athlon-64X2 はそれぞれ 4 つ）．例えば，それぞれ 2 つのカウンタを用いる場合のモデルは，$x_{(i,\ j)}$ をコア i におけるカウンタ j として，以下の式 (6) となる．なお，本モデルではそれぞれのコアで同

☆ 一般的に *knots* の数は多ければ多いほど正確な近似が可能となるが，オーバーフィッティングを考慮し，3 または 4 の値が用いられることが多く，本モデリングにおいては 4 で統一した．

じカウンタを用いることとし，それぞれのコアの同じカウンタ間の相互作用のみを考慮している．また，性能の低下率を見積もるためには，単独で実行したときと，複数のプロセスを実行したときで，命令数をそろえ，かかったサイクル数の比率を計算する必要がある．このため，1つのプログラムの組み合わせあたり得られた400サンプルを10サンプルにマージし，全体としては $465 \times 10 = 4650$ サンプルを学習に用いる．

$$y = \beta_0 + \beta_{(0,\,1)}x_{(0,\,1)} + \beta_{(0,\,2)}x_{(0,\,2)} + \beta_{(1,\,1)}x_{(1,\,1)} + \beta_{(1,\,2)}x_{(1,\,2)} + \beta_3 x_{(0,\,1)}x_{(1,\,1)} + \beta_4 x_{(0,\,2)}x_{(1,\,2)} \quad (6)$$

また，モデルに用いるカウンタは，文献[16]で用いた手法と同じように，すべてのカウンタ値を用いて全通りのモデリングを行い，寄与率の最も高いカウンタの組み合わせを選択する．

6.1 学習結果

学習の結果を表2に示す．表から，3つのプラットフォームに共通してRCSを用いた方がより高い寄与率を達成していることがわかる．一番寄与率の高いモデルはAthlon-64X2においてRCSを用いた場合で，このときの寄与率は0.75（相関係数 =0.87）であり，非常に予測精度が高いといえる．またAthlon-64X2においてはRCSを用いない場合においても寄与率が高く，その1つの大きな理由として，リソース待ちによるストール数☆という，リソース競合による性能低下を予測する上で非常に重要となるカウンタの値が取得可能であることが挙げられる．また，他に選択されたカウンタを見てもロード命令の頻度の指標となるL1データキャッシュのカウンタが選択されており，定性的にも妥当だと考えられる．

一方，Core2-QX6700を見てみると，L2キャッシュを共有している場合にはRCSを用いずとも高い寄与率（0.52）を達成しており，L2キャッシュミスとL1データキャッシュアクセスという非常に直感に合うカウンタが選ばれている．ただし，RCSを用いた場合は他の2モデルよりも寄与率が低く，その理由としては図2で示したとおり，カウンタには現れないような，データへのアクセスパターンなどによる性能の低下率が大きいためではないかと考えられる．また，L2キャッシュ非共有の場合においてはRCSを用いることによって0.66と高い寄与率を達成している．このとき，一方にハードウェア割り込みというカウンタが選ばれているが，このカウンタはAthlon-64X2のRCSを用いたモデルでも選択されており，ディスクでの競合など性能に大きく影響する，I/O処理の重さなどを表す指標になっていることが想像できる．

なお，3つのプラットフォーム全体を見ると，L1データキャッシュアクセスがすべてのモデルにおいて選択されていることがわかる．直感的にはL2キャッシュミス回数の方が適切な感じを受けるが，L2キャッシュミス回数が少ない場合において，その理由が他のコアのL2キャッシュミス回数が多く，自コアが性能低下してしまったため少なくなっているのか，それともL2キャッシュミスが元々少ないプログラムなのかといった判断がつきづらいからではないかと想像できる．それに対して，L1データキャッシュアクセスはロード命令の頻度を直接的に表すため，多くの場合においてこのカウンタを用いた方が正確な予測ができると考えられるが，この点に関してはさらなる解析が必要である．

☆ Core2-QX6700でもこの名称のカウンタは存在するが，値が常に0でり用いることができなかった．

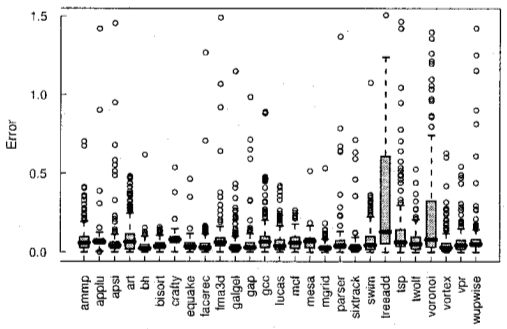

図3 Athlon-64X2における予測誤差（w/o RCS）

6.2 評価結果

次に，学習の結果一番寄与率が高かったAthlon-64X2について，RCSを用いた場合と，用いない場合のモデルについてleave-one-out cross validationによる評価を行った結果を図3および図4に示す．この図は横軸にアプリケーション，縦軸に予測誤差を示した箱ひげグラフとなっている．1つのアプリケーションあたり300点程度の予測を行っているため，はずれ値が多く見えてしまうが，縦軸の上限は1.5 %となっており，IQRの範囲がほとんどのアプリケーションで0.1%以下と狭いことからも，高い予測精度を達成していることがわかる．なお，1.5 %を超えているはずれ値も存在しており，全体に占めるはずれ値の割合が多いアプリケーションはRCSを用いた場合，用いない場合で *treeeadd* の33%，となっており，全体の平均ではどちらも10%弱となっている．また，RCSを用いた方が中央値が小さく，IQRが狭くなっていることからRCSを用いた方が精度の高いモデルを構築できたことがわかる．なお，どちらのモデルにおいても *treeadd* および *voronoi* に対する予測精度が他のアプリケーションと比較して低いが，この2つのプログラムは実行するたびに大きく性能が変動するため，そもそも予測することは難しい．

7. まとめと今後の課題

本稿では，CMP上で複数のプロセスを同時に実行した際にも，高いスループットを維持し，効率的な実

表 2　寄与率の最も大きいカウンタの組み合わせ

プラットフォーム	カウンタ（回数）	寄与率
Core2-QX6700（共有） w/o RCS	L2 キャッシュミス，L1 データキャッシュアクセス	0.52
Core2-QX6700（共有） $w/$ RCS	L2 キャッシュミス，L1 データキャッシュアクセス	0.60
Core2-QX6700（非共有） w/o RCS	L1 データキャッシュアクセス，L1 データキャッシュミス	0.40
Core2-QX6700（非共有） $w/$ RCS	ハードウェア割り込み，L1 データキャッシュアクセス	0.66
Athlon-64X2 w/o RCS	命令発行のなかったサイクル，完了命令， リソース待ちによるストール，L1 データキャッシュアクセス	0.69
Athlon-64X2 $w/$ RCS	ハードウェア割り込み，L1 命令キャッシュアクセス， リソース待ちによるストール，L1 データキャッシュアクセス	0.75

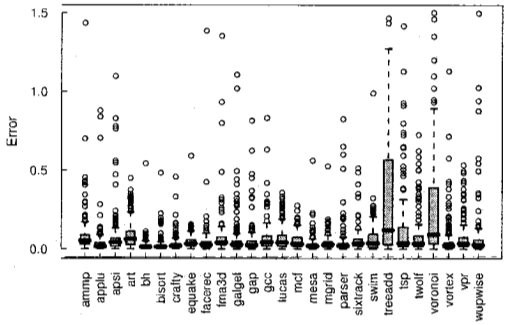

図 4　Athlon-64X2 における予測誤差（$w/$ RCS）

行を提供するために必要である考えられる，共有リソースでのアクセス競合による性能低下をモデリングするための統計的な手法を提案した．

Core2-QX6700 および Athlon-64X2 を用いた実機のプラットフォーム上で，交互作用を考慮した上で，Restricted Cubic Splines (RCS) を用いた非線形なモデルでの重回帰分析を行った．評価の結果，同時に複数のプロセスを実行した際においてそれぞれのコアから得られるデータを用いて正確に，性能の低下率を予測できることがわかった．本モデルは，文献[17] で提案されているトラクションコントロール実行のような，動的な最適化を行う際に非常に有効であるといえる．

今後，さらに統計手法のアルゴリズムを改良する他，さらに多くのベンチマークで提案手法の評価を行う予定である．また，4 コアや 8 コアとチップ上のコア数が増えた場合にも適用できるようなモデルおよび学習方法の提案や，提案手法によるモデリングを用いた CMP における効率的な実行の提案・評価などが重要な課題として挙げられる．

謝辞　本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の研究プロジェクト「革新的電源制御による超低電力高性能システム LSI の研究」，および文部科学省科学研究費補助金 (基盤研究 (A) No.18200002) の支援によって行われた．

参　考　文　献

1) G.E.Suh, et al., "A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning", *In Proc. HPCA*, pp.117–128, Feb. 2002.

2) M.K.Qureshi and Y.Patt,"Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches", *In Proc. MICRO*, pp.423–432, Dec. 2006.

3) L.R. Hsu, S.K. Reinhardt, R.K. Iyer, S. Makineni, "Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource", *In Proc. PACT*, pp.13–22. Sep. 2006.

4) K.J.Nesbit, et al., "Fair Queuing Memory System", *In Proc. MICRO*, pp.208–219, Dec. 2006.

5) N.Rafique, W.-T.Lim, and M.Thottethodi, "Architectural Support for Operating System-Driven CMP Cache Management", *In Proc. PACT*, pp.2–12, Sep. 2006.

6) D.Chandra, et al., "Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture", *In Proc. HPCA*, pp.340–351, Feb. 2005.

7) A.Snavely and D.M.Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multithreading Processor", *In Proc. ASPLOS IX*, pp.234–244, Nov. 2000.

8) K.Luo, et al., "Balancing Throughput and Fairness in SMT Processors", *In Proc. ISPASS 2001*, pp.164–171, Nov. 2001.

9) T.Moseley, et al., "Methods for Modeling Resource Contention on Simultaneous Multithreading Processors", *In Proc. ICCD*, pp.373–380, Oct. 2005.

10) S.Browne, et al., "A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters" *In proc SC*, 2000.

11) PAPI group. PAPI Software Specification, Version 3.0.

12) J.R.Bulpin and I.A.Pratt, " Multiprogramming Performance of the Pentium 4 with Hyper-Threading", *WDDD*, 2004.

13) N.Tuck and D.M.Tullsen, "Initial Observations of the Simultaneous Multithreading Pentium 4 Processor", *In Proc. PACT*, 2004.

14) E.Ipek, et al., "Efficiently Exploring Architectural Design Spaces via Predictive Modeling", *In Proc. ASPLOS*, pp.185–194, 2006.

15) B.C.Lee and D.Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction", *In Proc. ASPLOS*, pp.195–206, 2006.

16) 佐々木，浅井，池田，近藤，中村，"統計情報に基づく動的電源電圧制御手法"，情報処理学会論文誌，Vol.47, No.SIG18 (ACS16), 2006 年 11 月．

17) 近藤，佐々木，中村，"トラクションコントロール実行: CMP 向け実行制御方式の検討"，情報処理学会研究報告，ARC-174, 2007 年 8 月．