2008－ARC－179 (6)
2008／8／5

CMPの統計的モデリングによる実行時最適化手法

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，複数のプロセッサコアを1チップに搭載した，チップマルチプロセッサ（CMP）が主流となっている．CMPでは，一般的に複数のコアがラストレベルキャッシュ，チップ・メモリ間バスやメモリバンクなどのリソースを共有することになる．これら共有リソース上で競合が発生すると，チップ全体のトータルの性能が低下する，あるいは各プロセスの性能低下の公平さ（Fairness）が保たれないなどといった問題が生じる．本稿では，どのような振る舞いをするプログラムが他のコアで実行されているプログラムにどの程度の影響を与えるのかを統計的に解析し，その情報を基に各コアの周波数を調整することでリソース競合の影響を制御し，効率的なプログラム実行を提供する手法を提案する．

Dynamic Optimization for CMP by Efficient Regression Modeling

Hiroshi Sasaki,[†] Masaaki Kondo[††] and Hiroshi Nakamura[†††]

Recently, a single chip multiprocessor (CMP) is becoming an attractive architecture due to its advantage of achieving high throughput and low power. In CMPs, multiple processor cores share several hardware resources such as cache memories, memory buses, and main memory banks. Significant performance and/or fairenss degradation occurs in the case of resource contention. In this paper, we statistically analyze the relationships between the characterisitcs of the program behavior and its performance sensitivity when other programs are running simultaneously on other cores. We propose a novel dynamic voltage and frequency scaling (DVFS) technique which change the frequency of each core individually according to the statistical analysis so as to control the shared resource utilization to offer an effective computing on CMPs.

1. はじめに

近年，汎用マイクロプロセッサのアーキテクチャとして，複数のプロセッサコアを1チップに搭載した，チップマルチプロセッサ（chip multiprocessor: CMP）が主流となっている．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を行うことでクロック周波数の向上に頼ることなく高性能（高スループット）化を達成できるため，消費電力あたりの性能に優れるアーキテクチャである．一般的にCMPではリソース有効活用の観点から，複数のプロセッサコアが，あるメモリ階層以下にあるキャッシュやバス，主記憶であるDRAMのバンクなどを共有することが多い（図1に一般的なCMPの構成を示す）．しかし，これらの共有リソースに対して複数のプロセッサコアが同時にアクセスし競合が発生すると，性能が低下することがある．特に，プロセッサと主記憶の性能差が拡大している近年においては，メモリ階層において競合が発生すると性能への影響が非常に大きく，深刻な問題となっている．

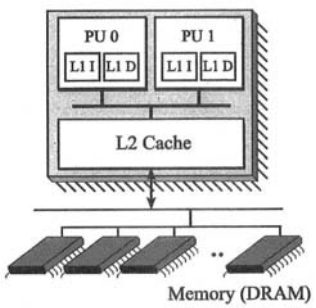

図1 CMPの概要

上述したようなリソース競合の発生により，(1) トータルスループットの低下，(2) *Fairness*（各スレッドの，競合による性能低下の公平さ）の低下，(3) 性能

† 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
†† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
††† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

予測性の悪化，といった問題が生じる．これまでにも，リソース競合の影響の緩和を目的としていくつかの手法が提案されている．例えば，キャッシュ上での競合を防ぐために，キャッシュを論理的なパーティションに分割する手法[5),7),13)]，主記憶SDRAMのバンク上での競合に対処するためのメモリアクセススケジューリング手法[10)~12)]などが提案されている．従来手法により，ある程度リソース競合の影響を緩和させることができると思われるが，今後より多くのプロセッサコアが1チップに集積されると競合による影響はさらに深刻になるため，今まで以上に柔軟かつ効果的にリソース競合の影響を制御できる手法が必要になると考えられる．

我々は，これまでに動的電源電圧制御（DVFS: dynamic voltage and frequency scaling）手法を用いてCMP上の各コアの周波数を独立に制御し，fairnessおよびトータルスループットを向上させつつ，命令あたりの消費エネルギーを削減する手法を提案している[8)]．また，各コアの実行スピードをOSによるタイムスライスの割り当て量の調節によって仮想的に制御し，CMPにおいてリソース競合の影響を柔軟に制御し効率的な実行環境を提供する手法を提案している[16)]．文献[16)]では実行スピードの調整を，あらかじめ統計的な解析手法によって構築した競合の影響を予測するモデルに基づいて行なう．本稿では，文献[16)]と同じように競合の影響を予測するモデルを統計的に構築し，そのモデルに基づいてCMP上でコア毎にDVFSを適用し，効率的な実行を提供する手法を提案する．また，文献[16)]における統計モデルを改良すること，およびCMPでのリソース競合の影響の定性的な理解を深めることを主たる目的として，コア毎にDVFSを適用することが可能な実機のプラットフォーム上で解析を行う．解析にはリソース競合の度合いを任意に変更することができるシンセティックなプログラムを用い，ハードウェアイベントの取得には文献[16)]と同じように，近年のほとんどのプロセッサに搭載されているパフォーマンスモニタリングカウンタ（Performance Monitoring Counter: PMC）を用いる．解析の結果，周波数を変更した場合の性能をPMCの値から予測する非常に精度の高いモデル式を構築できることが分かった．

2. 関連研究

従来より，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[7)]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[13)]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[5)]では，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文献[14)]で提案されている．さらに，文献[1)]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，文献[6)]では，QoSを導入した共有キャッシュの制御手法について述べられている．

文献[3)]は，キャッシュ競合の影響を考慮したOSによるプロセススケジューリング手法を提案している．これは，コア数以上のアクティブに動作するプロセスに対し，各プロセスのタイムスライス量を性能低下率にあわせて変化させ，fairnessを向上させるものである．この手法は，コア数以上のプロセスが動作している場合しかタイムスライス量の調整ができないため，アクティブに動作しているプロセス数がコア数以下である場合には効果がない．

文献[10)~12)]では，CMPの主記憶DRAMアクセスのスケジューリングにおいて，トータルスループットを向上させるためのQoS制御手法が提案されている．

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 multithreading（SMT）プロセッサでは，リソースの使用率を考慮した最適化の研究が活発に行なわれている[15)]．また，メモリアクセスなど長いレーテンシのストールが発生した際に，異なるスレッドに切り変えて実行することでスループット向上を狙うswitch on event（SOE）型のマルチスレッドプロセッサにおけるfairness向上手法も提案されている[4)]．また，SMT上での共有リソース競合のモデリング手法が文献[9)]で述べられている．

3. CMPにおけるリソース共有の影響

CMPでは，図1に示すように，複数のプロセッサコアがメモリバスや主記憶を共有するのが一般的である．また，図のようにチップ内のL2キャッシュを共有する場合も多い．このように，複数のPUでリソースを共有する場合，各コア上で動作するプロセスの性能は，共有リソース上での競合の状況に大きく依存する．

リソース競合が性能に与える影響を調べるため，*Intel Core2 Extream QX6700*（以下Core2-QX6700）を搭載した実機のマシンにより評価を行なった．Core2-QX6700は，4コアを持つCMPであるが，実際には4 MB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個搭載したものであり，今回は2プロセスを同時に実行した場合について評価を行なった．なお，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が，今回はキャッシュ以外の共有リソース上での競合も影響が大きいことを示すために，L2キャッシュを共有しない場合について評価を行なった．したがって，メモリバ

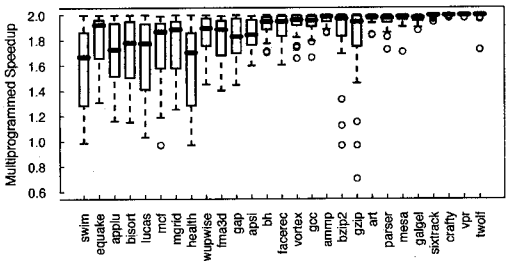

図 2 Core2-QX6700 における multiprogrammed speedup

ス（FSB）以下のメモリ階層が共有リソースとなる．

図 2 に，SPEC2000 ベンチマーク，および Olden ベンチマークのいくつかのプログラムの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 multiprogrammed speedup を示す．multiprogrammed speedup は，CMP における評価指標の一つであり，それぞれのプロセスを単独で実行した場合に対して，複数プロセスで同時に実行した際の各プロセスの性能比を，全プロセスで合計したものである．図は，対象プログラム（横軸に示すプログラム）と，全プログラムとの組み合わせにおける multiprogrammed speedup を，“箱ひげプロット（box-and-whisker plot）”で表わしており，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，ひげ部分の線が，箱部分の上下それぞれから $1.5 \times IRQ$ 範囲内のデータを示している．その範囲外のデータは，はずれ値としてそのデータポイントがそれぞれ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の L2 キャッシュミス率の高い順に並んでいる．

図 2 より，対象プログラムの L2 キャッシュミス率が高い場合には，multiprogrammed 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る．また，この評価では L2 キャッシュを共有していない 2 コア上で対象プログラムを実行しているため，これらの性能低下はメモリバス，およびメモリバンクの競合により生じたものである．したがって，キャッシュを共有していない場合でも競合の影響は深刻であり，キャッシュを共有する場合には，キャッシュ上での競合も発生することから性能への影響はさらに大きくなる．

このように，CMP ではリソース競合の影響で，実行しているプロセスの性質に依存して性能が大きく低下してしまう場合がある．次節では，このリソース競合の影響を緩和させるための提案手法について述べる．

表 1 評価に用いたマシンの仕様

	Phenom-9850
CPU	AMD Phenom X4 9850
- コア数	4
- 周波数	2.50 GHz
- L1 キャッシュ	64+64 KB per core
- L2 キャッシュ	512 KB per core
- L3 キャッシュ	2 MB
主記憶	DDR2-SDRAM
- コントローラ	Dual channel DDR2-1066 MHz
- 周波数	166 MHz (DDR2-667)
- サイズ	2.0 GB

4. 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，以降より *AMD Phenom X4 9850*（以下，Phenom-9850）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．Phenom-9850 は，512 KB の L2 キャッシュをそれぞれのコアに搭載するネイティブな 4 コアの CMP（その点で前述の Core2-QX6700 とは異なる構成である）であり，2 MB の L3 キャッシュを共有している．また，メモリバス以下のメモリ階層も共有リソースとなっている．Phenom-9850 は各コアの周波数および電源電圧を独立に制御することができ，本プロセッサを用いることにより，文献[8] においてはシミュレーションで評価していた手法を実機上で評価することが可能となる．

評価用のソフトウェアとして，OS には Linux（Kernel: 2.6.25）を，カウンタ値の取得には perfmon2 インターフェース[2] を用いる．Phenom-9850 は 412 種類のイベントを PMC を用いることによって取得することができる．コアあたり 4 つの PMC を搭載しており，同時に最大で 4 種類のイベントを取得可能である．また，周波数は Linux の cpufreq ドライバ経由で設定し，2.50 GHz および 1.25 GHz の 2 通りを設定することができる．本稿では，リソース競合の影響の解析に 2 コアを用い，周波数は設定可能な 2 通りを用いる．

5. CMP 向け実行時最適化手法

本節では提案手法である，CMP においてリソース競合の影響を制御し，効率的なプログラムの実行を目指す DVFS 手法について述べる．

5.1 概　　要

あるコア上で実行しているプロセスの共有リソースへのアクセス率が高い場合（例えば，ラストレベルキャッシュミス率が相対的に高いなど），当該プロセスの性能低下に比べて，他のコア上で実行されているプロセスの性能低下が競合により大きくなる可能性がある．この場合，もともと高い命令スループットを達成できるプロセスの性能低下が大きいと，トータルス

ルーブットが大きく低下してしまったり，fairnessが保たれないといった問題が生じる．

提案手法は，各コアの周波数をDVFSによって制御することによって，共有リソースへのアクセス率を調整し競合の発生をコントロールすることで，リソース競合における種々の問題の解決を狙うものである．例えば，共有リソースアクセス率の高いプロセスの実行スピードを遅くすれば競合の発生が抑制され，他のプロセスの性能低下を改善することができる．

5.2 統計的モデリングによる競合の影響の予測

1章でも述べたように，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性能が大きく異なり，その振る舞いは複雑である．したがって，リソース競合が性能に与える影響の度合い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[17]，本稿ではこの手法を応用することで競合による性能低下の解析を行う．

5.2.1 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(x_i)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，yは従属変数，x_iは独立変数，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．

$$y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + e \qquad (1)$$

本稿のモデリング手法では，単独で最高周波数において実行した際に比べてある周波数におい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PMC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，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．以下に交互作用付きの線形回帰モデルを示す．

$$y = \beta_0 + \sum_{i=1}^{n-1} \beta_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \beta_{i,j} x_i x_j + e \qquad (2)$$

5.2.2 学習のためのデータサンプリング

本節では，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．学習には，任意に共有リソースへのアクセス率を変化させることが可能なシンセティックなプログラムを用いる．このプログラムを用いて様々なアクセス率を定常的に持つプログラム郡を用意し，これらを学習用のプログラムとしてモデリングを行う．このことにより，各サンプルデータがプログラム内の振る舞いの変化などの影響を受けることがなくなり，解析結果の理解が容易になり，また精度のよいモデルが構築できると考えられる．

また，前節で述べたとおり，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，独立変数(x_i)をそれぞれのコアにおけるPMCの値としている．また，文献[17]で提案しているように，設定可能な周波数ごと（本稿においては2.50 GHzおよび1.25 GHzの2つ）に回帰式を求めることによって，周波数を変更した際の性能比の予測を行うことが可能となる．学習用のデータはそれぞれのコアでPMCの値を一定の時間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．本論文では，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，2.50 GHzで実行した際の10億サイクル（1.25 GHzのときは5億サイクル）ごとにシグナルを送り，その都度PMCの値を取得することにする．それぞれの組み合わせにおいて10サンプルを取得した時点（約4秒）でサンプリングを終了し，学習用のデータとしてはこれら10サンプルの平均を取ったものを用いる（ただし，定常的なプログラムを使用しているため，10サンプル間の差異はほとんど見られない）．

5.2.3 モデリング

前節の手法でサンプリングしたデータから，重回帰分析を用いてモデリングを行う．モデリングに際して，実行時に性能の予測を行なうことを想定した場合，用いるカウンタの個数は対象とするプラットフォームで各コアが同時に取得できるカウンタの個数に制限される．例えば，2つのカウンタを用いる場合のモデルは$x_{(i,\,c)}$をコアiにおけるカウンタcとして，以下の式(3)となる．なお，本モデルではそれぞれのコアで同じ種類のカウンタを用いることとしている．

表 2　用いたカウンタにおける寄与率

プラットフォーム	カウンタ（回数）	寄与率
Phenom-9850 (2.50 GHz)	L2 トータルキャッシュミス，L3 トータルキャッシュミス，実行命令数	0.79（学習：2.50 GHz） 0.76（学習：2.50 GHz，1.25 GHz）
Phenom-9850 (1.25 GHz)	L2 トータルキャッシュミス，L3 トータルキャッシュミス，実行命令数	0.78（学習：1.25 GHz） 0.76（学習：2.50 GHz，1.25 GHz）

$$y = \beta_0 + \beta_1 x_{(0,\,1)} + \beta_2 x_{(0,\,2)} + \beta_3 x_{(1,\,1)} + \beta_4 x_{(1,\,2)} + \beta_5 x_{(0,\,1)} x_{(1,\,1)} + \beta_6 x_{(0,\,1)} x_{(1,\,2)} + \beta_7 x_{(0,\,2)} x_{(1,\,2)} + \beta_8 x_{(0,\,2)} x_{(1,\,1)} \quad (3)$$

最終的にモデルに用いるカウンタは，全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. 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．Phenom-9850 においては 412 種類のイベントを測定することが可能であるが，今回はモデルに用いるイベントとして，前もって行った実験で共有リソースでの競合をある程度表わすことが分かっている，L2 トータルキャッシュミス，L3 トータルキャッシュミス，および実行命令数とした（残り 1 つは時間間隔を測定するための，トータルサイクルである）．これらのイベントは定性的に考えても競合に強く影響があるイベントであることはほぼ自明であり，性能低下率を予測する上で最もよいカウンタである保証はないものの，初期の解析としては十分であると考えられる．

なお，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，すなわち 4 つとした．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，前章の式 (3) となる．なお，本モデルではそれぞれのコアで同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，共有リソースへのアクセス率を任意に設定可能であるシンセティックなプログラムとしては，倍精度のベクトル積の演算を行うプログラムを用いる．このプログラムは入力のベクトルのサイズを変えることによって，再び同じデータをアクセスするまでに触るデータサイズが変わり，それによってキャッシュのヒット率，ひいては共有リソースである L3 キャッシュや，それ以下のメモリ階層へのアクセス率が変わるというものである．本モデリングにおいては，ベクトルサイズを 1,000 から 76,000 まで 3,000 おきに，また 80,000 から 200,000 まで 5,000 おきに変化させ（51 データ），それら全ての組み合せを実行することによってデータのサンプリングを行った．また，ターゲットの周波数を 2.50 GHz とした場合には，相手のコアの周波数は 2.50 GHz のみを用いて学習を行う場合と，2.50 GHz および 1.25 GHz の両方の場合を学習に用いる場合を評価しており，これは相手の周波数がいかなる値であろうとも，共有リソースへの時間あたりのアクセス率によって性能の低下率が見積もれるのか，それとも相手の周波数が異なる場合には別のモデル式を構築しなければならないのかどうかを確認するためである．以上から，1 つの周波数に対して，学習に用いるサンプル数は同じ周波数のみの場合で $51 \times 51 = 2601$ であり，両方の周波数を用いる場合で $51 \times 51 \times 2 = 5202$ となる．

6.1 学習結果

学習の結果を表 2 に示す．なお，各周波数において，同一周波数のみのデータから学習した場合と，異なる周波数のデータも一緒に用いた場合の両方について結果を示している．表から，どちらの周波数においても非常に高い寄与率を達成していることがわかる．寄与率はどちらもおよそ 0.80 弱程度であり，これは相関係数 0.90 弱程度を意味しており，非常に予測精度が高いといえる．また，異なる周波数のデータを用いた場合においても，同一の周波数のみの結果とほぼ変わらない結果を示しており，これ以上新たなモデルを構築する必要なく比較的容易に各周波数ごとにモデル式を構築する文献[17] における手法を用いることができると考えられる．

また，今回は全てのカウンタのデータをサンプリングしてその中から最も寄与率の高いカウンタを選ぶという手法を用いていないが，L2 トータルキャッシュミス，L3 トータルキャッシュミス，および実行命令数のカウンタを用いることで非常に高い寄与率を達成することができた．412 種類全てのイベントについて学習を行えば，さらに寄与率の高いイベントの組み合わせを発見できる可能性がある．本評価より，実行時に高い精度で周波数を変更した場合の性能が予測できることが分かったものの，コア数が増加するに従って学習に必要な時間が指数的に増えるという問題や，モデル式の項数が指数的に増えることによってその計算時間が無視できなくなることなどの問題が起こることが分かっている．これらの問題を克服し，CMP 上で効率的な DVFS 手法を提案することが今後の課題である．

7. おわりに

我々は，CMP 上で複数のプロセスを同時に実行した際に，コアごとに DVFS を用いることによって共有リソースでの競合を緩和し効率的な実行を提供する手法を提案している[8]．本稿の提案手法は，性能の低下率を統計的なモデルから予測することによって周波数を変更するものであり，そのための統計的手法に関する解析を行った．

Phenom-9850 を用いた実機のプラットフォーム上で，シンセティックなプログラムを実行して得られたデータを用い，交互作用を考慮したモデルで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分かった．本モデルを用いて実行時に DVFS をコア毎に行い，CMP において効率的な実行を提供することが今後の目的である．

また，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，3 コア，4 コアにおいても適用できるようなモデルおよび学習方法の提案なども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) D.Chandra, F.Guo, S.Kim, and Y.Solihin. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In *HPCA-11: Proceedings of the 11th International Symposium on High Performance Computer Architecture*, pages 340–351, 2005.
2) S.Eranian. perfmon2. http://perfmon2.sourceforge.net/.
3) A.Fedorova, M.Seltzer, and M.D. Smith. Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 25–38, 2007.
4) R. Gabor, S. Weiss, and A. Mendelson. Fairness and Throughput in Switch on Event Multithreading. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 149–160, 2006.
5) L.R. Hsu, S.K. Reinhardt, R.Iyer, and S.Makineni. Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource. In *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 13–22, 2006.
6) R. Iyer. CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 257–266, 2004.
7) S. Kim, D. Chandra, and Y. Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 111–122, 2004.
8) M.Kondo, H.Sasaki, and H.Nakamura. Improving Fairness, Throughput and Energy-Efficiency on a Chip Multiprocessor through DVFS. *SIGARCH Comput. Archit. News*, 35(1):31–38, 2007.
9) T. Moseley, D. Grunwald, J. L. Kihm, and D. A. Connors. Methods for Modeling Resource Contention on Simultaneous Multithreading Processors. In *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*, pages 373–380, 2005.
10) O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 146–160, 2007.
11) O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 63–74, 2008.
12) K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing Memory Systems. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 208–222, 2006.
13) M.K. Qureshi and Y.N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 423–432, 2006.
14) N.Rafique, W.-T. Lim, and M.Thottethodi. Architectural Support for Operating System-Driven CMP Cache Management. In *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 2–12, 2006.
15) A. Snavely and D. M. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 234–244, 2000.
16) 近藤正章, 佐々木広, 中村宏. トラクションコントロール実行: CMP 向けプロセス実行制御方式の提案. In *SACSIS 08: Proceedings of the symposium on Advanced computing systems and infrastructures*, pages 265–272, 2008.
17) 佐々木広, 浅井雅司, 池田佳路, 近藤正章, 中村宏. 統計情報に基づく動的電源電圧制御手法. In *IPSJ Transactions on Advanced computing systems*, Vol 47, No. SIG18 (ACS16), pages 80–91, 2006.