計算機アーキテクチャ 150－5
（２００２．１１．２８）

最適実行多重度に基づく SMT プロセッサのジョブスケジューリング方式

大河原　英喜[†]　安里　彰[†]

近年、次世代マイクロアーキテクチャとして、SMTプロセッサが注目を浴びている。SMT では、1 CPU 上で複数スレッドを同時実行することでスレッドレベル並列性の有効活用を図るが、その一方、過剰なスレッド同時実行を行うと、スレッド間で資源競合を起こして性能が低下する問題が知られている。本論文では、最適なスレッド実行多重度を動的に測定して制御することで、過剰なスレッド同時実行による性能低下を避ける手法について述べる。SPEC 2000 を用いて性能評価を行った結果、従来の SMT では資源競合によって 70 ～ 80% の性能低下が発生する場合でも、本手法を用いることで、性能低下を 35 ～ 45% に抑えることができた。

Dynamical Control of Multithreading Level on a SMT Processer

Hideki Okawara[†] and Akira Asato [†]

Recently the SMT (Simultaneous Multi-Threading) architecture has been gaining in popularity. This architecture allows multiple threads to run at the same time exploiting thread level parallelism effectively. But executing excessive threads simultaneously may cause inter-thread contentions and the performance degradation. In this paper we propose and evaluate a new technique to avoid this problem by measuring and controlling the adequate multi-threading level dynamically. We show that our technique can reduce by half the performance degradation compared to usual SMT.

1. はじめに

近年のプロセッサでは、パイプライン段数増加によるクロック高速化や、集積度向上による CPU 資源増加によって大幅に性能が向上している。その反面、命令レベル並列性の限界から、有効利用されていない CPU 資源も多く、ハードウェア資源に見合った性能向上が得られていないのが現状である。そこで、CPU 上で実行されるスレッドを頻繁に切り替えることで、スレッドレベル並列性の有効活用を図る MT (Multi-Threading) プロセッサの研究が行われてきた。さらに、1 サイクル当りにも複数スレッドから命令フェッチを行う様に拡張された SMT (Simultaneous Multi-Threading) プロセッサ[1)2)] の研究も行われてきた。ここ数年では、Intel 社 Xeon の hyper-threading[3)] に代表される様に製品化が行われており、IBM 社や SUN 社からも SMT プロセッサの開発計画が発表されている。

MT/SMT プロセッサでは、図 1に示す様に 1CPU 上に複数のハードウェアコンテキスト（レジスタ、プログラムカウンタ等）を持ち、コンテキストスイッチを行うオーバーヘッドなしに、実行スレッドの切り替えを行う。演算器やキャッシュなどの計算機資源を複数スレッド間で共有することで、資源の有効活用が図られ IPC 向上が期待される。その反面、スレッド間で資源競合を起こす可能性もあり、同時実行されるスレッドによっては、性能が低下するという問題点が報告されている[4)]。特に、過剰なスレッド同時実行が行われることによって L2 キャッシュミス率が増加する場合には、性能が大きく低下することが知られている[5)]。

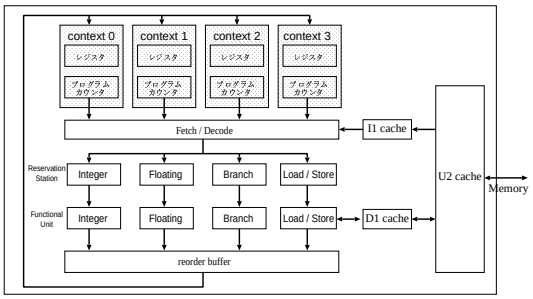

図 1　SMT プロセッサの概念図

本論文では、この問題に対して、SMT プロセッサの動作状態を基に最適なスレッド実行多重度を決定し、動的に実行多重度を変化させることで、過剰なスレッド同時実行による性能低下を避ける手法について述べる。まず初めに、2節において関連研究について述べ、3節で本論文の提案する手法について説明する。その後、性能評価を行った結果を 4節で示し、5節、6節でまとめる。

† (株) 富士通研究所
FUJITSU LABORATORIES LTD.

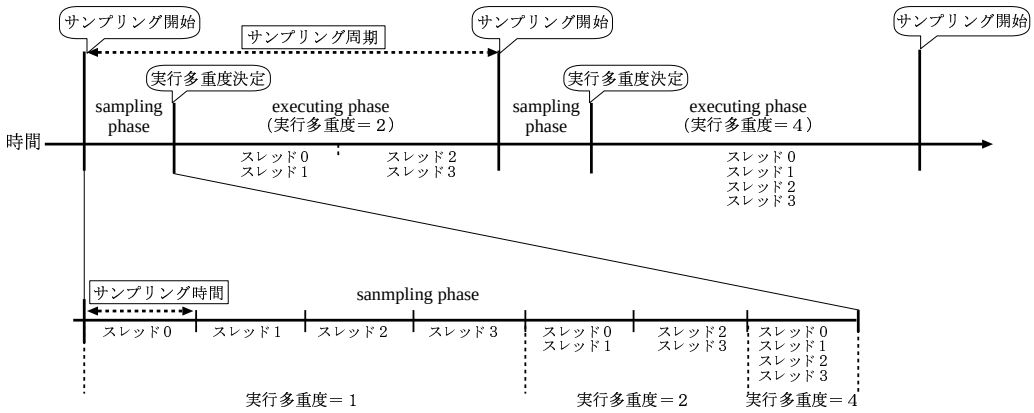

図 2 DCML のタイミングチャート

2. 関 連 研 究

2.1 ジョブスケジューリング方式に関する研究

SMT プロセッサにおけるスレッド間の資源競合について注目して、ジョブスケジューリング方式に関する関連研究[6)7)8)9)] などが行われている。これらの研究は、同時実行するスレッドの組合わせによって性能が影響を受ける点に注目し、SMT プロセッサのコンテキスト数以上のスレッドが存在する場合に、資源競合を起こさない相性の良いスレッドの組を選択してコンテキストスイッチを行うものである。

例えば Tullsen らによる研究[6)] では、まず、コンテキストスイッチの際に適当な組合わせのスレッドを割り当てて、統計情報 (IPC や INT/FP 演算器上での資源競合) を測定する sample phase を繰り返す。コンテキストスイッチを繰り返していくつかの組合わせに関して統計情報が得られた後は、それらの情報を基に最適なスレッドの組合わせを選択してコンテキストスイッチを行うことで性能向上を図っている。

同様に、Ledlie らによる研究[8)] では、各スレッドの演算器使用率（命令ミックス）や、load/store 命令数、L1 キャッシュミス率を測定しておき、演算命令バランスの異なるスレッドを選択したり、キャッシュミス率の多いスレッドと少ないスレッドを選択することで、特定の CPU 資源に負荷が集中しない様にコンテキストスイッチを行っている。

前述したこれらの関連研究は、コンテキストスイッチを行う際に、休止状態にあるスレッド群の中から CPU に割り当てるスレッド選択を行うものである。そのため、実際にコンテキストスイッチで割り当てられたスレッド間で資源競合が発生してしまう場合や、十分な数の休止状態スレッドが存在しない場合には効果を示さない。それに対して、本論文で提案する手法は、コンテキストスイッチで CPU に割り当てられたスレッド群を実行する際に、動的に実行多重度を変化させることで、過剰なスレッド同時実行による資源競合を防ぎ、性能低下の低減を図るものである。

2.2 SMT に適したキャッシュメモリに関する研究

SMT プロセッサにおけるスレッド間干渉によるキャッシュミス増加を防ぐ研究として、Devadas らによる研究[10)] が挙げられる。この研究では、キャッシュを適当なパーティションに区切り、常に全スレッドで共有するのでなく、スレッドごとに必要に応じた量のキャッシュブロックを動的に割り当てることでキャッシュ干渉を防ぐ。同様の効果は、ページカラーリングによってアドレスマッピングを工夫することでも期待できる。

しかし、キャッシュパーティショニングでは、一定の総容量のキャッシュ上で複数スレッドが同時実行される点は変らないため、特に容量性ミスが多い場合には効果が得られない。それに対して、本論文で提案する手法では、同時実行されるスレッド数を減らすことで、容量性キャッシュミスの増加も防ぐことが期待される。

3. 提 案 手 法

3.1 DCML (Dynamical Control of Multithreading Level)

本論文では、過剰なスレッド同時実行による性能低下を防ぐためのジョブスケジューリング手法、DCML (Dynamical Control of Multithreading Level) を提案する。DCML では、一定サンプリング周期ごとに統計情報を採取して、SMT プロセッサを動作させる最適なスレッド実行多重度を決定する。4 コンテキストの SMT（以後、4SMT と表現する）を例にとって、DCML のタイミングチャートを図 2に示す。DCML では以下の 2 つの処理フェーズを繰り返す。

sampling phase 実行多重度を変化させてスレッド群を実行し、各実行多重度での統計情報を測定する☆。sampling phase 終了時には、採取した統計

☆ 実行多重度が 2 以上の場合にはスレッドの組合わせが何通りか考えられる場合もあるが、ここでは、sampling phase の時間短縮のためにラウンドロビン方式による 1 通りの組合わせのみサンプリングを行う。

情報を基に最適な実行多重度を選択する。選択指標としては、IPC やキャッシュミス率、フェッチストールサイクル、演算器等の CPU 資源の不足によるパイプラインストールサイクル等が考えられる。

executing phase sampling phase で決定された実行多重度に基づいて SMT プロセッサ上でスレッドを実行する。OS は、実行多重度に応じて executing phase の時間を time sharing して、ハードウェアコンテキスト上のスレッド群を実行する。図 2の例では、一回目のサンプリングでは実行多重度として 2 が選択されており、executing phase を 2 等分して 2 スレッドずつ同時実行している。二回目のサンプリングでは実行多重度として 4 が選択されており、time sharing は行わずに常に 4 スレッドが同時実行されている。

実行多重度を動的に変化させる仕組みとしては、OS 側でハードウェアコンテキスト上に割り当てられているスレッドの動作状態／待機状態を切り替えて制御する。プロセッサ側では、従来通り、動作状態のスレッドのみから命令フェッチを行うことで、同時実行スレッド数を調整することができる。

図 2中に示されているサンプリング時間の長さに関して、以下のトレードオフが存在する。

- サンプリング時間が長い場合、サンプリングされる統計情報の信頼度は高いが、サンプリングオーバヘッドが大きいといった問題点がある。
- サンプリング時間が短い場合にはオーバーヘッドは小さいが、局所的な統計情報しか得られないために、統計情報の信頼度が低いという問題点がある。

また、図 2中に示されているサンプリング周期（サンプリングを行って実行多重度を選択する間隔）に関しては、以下のトレードオフが存在する。

- サンプリング周期が長いと、適切な実行多重度が選択された場合の性能向上は高いが、不適切な実行多重度が選択された場合に性能が低下する。また、コンテキストスイッチやスレッド生成／終了による CPU に割り当てられているスレッドの変化や、同一スレッドでも処理フェーズによる性質の変化に対する対応が遅れるといった問題点がある。
- サンプリング周期が短いと、適切な実行多重度が選択された場合の性能向上の効果が低くなる。その一方、適切な実行多重度が選択されなかった場合や、スレッド割り当てや処理フェーズの動的な変化に対して、柔軟に対応することができる。

サンプリング時間とサンプリング周期の最適なバランスを求めることは難しいが、4節では何通りかの性能評価を行った中から、適当な結果に関して示す。

4. 性能評価

本節では、DCML による性能向上に関する性能評価結果を示す。まず初めに、評価条件について 4.1節で述べる。4.2節では、従来の SMT プロセッサの性能比較を行い、本論文で問題としている過剰なスレッド同時実行による性能低下の様子を示す。その後、4.3節において DCML による性能改善を示す。

4.1 評価条件

性能評価には、我々が開発した SPARC V9 アーキテクチャベースのソフトウェアシミュレータである SEEDS シミュレータ[11] を用い、トレースドリブンシミュレーションによる評価を行った。評価トレースとしては SPEC2000 ベンチマーク（CINT 12 種類、CFP 14 種類）を用いた。各ベンチマーク、60 本のトレースファイルの中から、文献[12] の手法を用いて、全体の挙動を表現するに足る複数本のトレースファイル群を選出してシミュレーション評価を行い、ベンチマーク全体の挙動を推定した。各シミュレーションに用いたトレースファイルは 40M 命令である。

評価対象のアーキテクチャパラメータを表 1に示す。SMT プロセッサの実行多重度に関しては、4.2節では 1SMT/2SMT/4SMT/8SMT の 4 通りを評価するが、表 1中のパラメータは共通である。

表 1 評価対象のアーキテクチャパラメータ

命令フェッチ幅	8 命令／サイクル
演算器	(int,fp,ldst,br)=(6,6,4,2)
リザベーションステーション	(int,fp,ldst,br)=(32,32,24,16)
リネーミングレジスタ	(int,fp)=(64,64)
I1/D1 キャッシュ	128kB, 2way, レーテンシ 1 サイクル
U2 キャッシュ	4MB, 4way, レーテンシ 13 サイクル
メモリアクセス	平常時レーテンシ 220 ～ 250 サイクル

4.2 予備評価

従来の SMT プロセッサにおける資源競合の問題点を示すために、従来の 1SMT ～ 8SMT のシミュレーションを行い、実行多重度が増加した際の性能比較を行った。ここでは、ハードウェアコンテキスト数分のプロセスが同時実行される状況を仮定して評価を行った。

各ベンチマークにおける 1SMT ～ 8SMT のチップトータルの IPC を比較した結果を図 3に示す。上段には CINT、下段には CFP の各ベンチマークの IPC が示されている。各ベンチマークにおける IPC を比較すると、大半のベンチマークでは、実行多重度が最も高い 8SMT の性能が最も良いことがわかる。しかし、必ずしも 8SMT の性能が最善ではなく、gzip, gcc においては 4SMT の性能が最も良く、mcf, twolf, galgel, art では 2SMT の性能が最も良い。特に gzip, galgel, art では 8SMT における性能低下が著しく、最も性能が良い

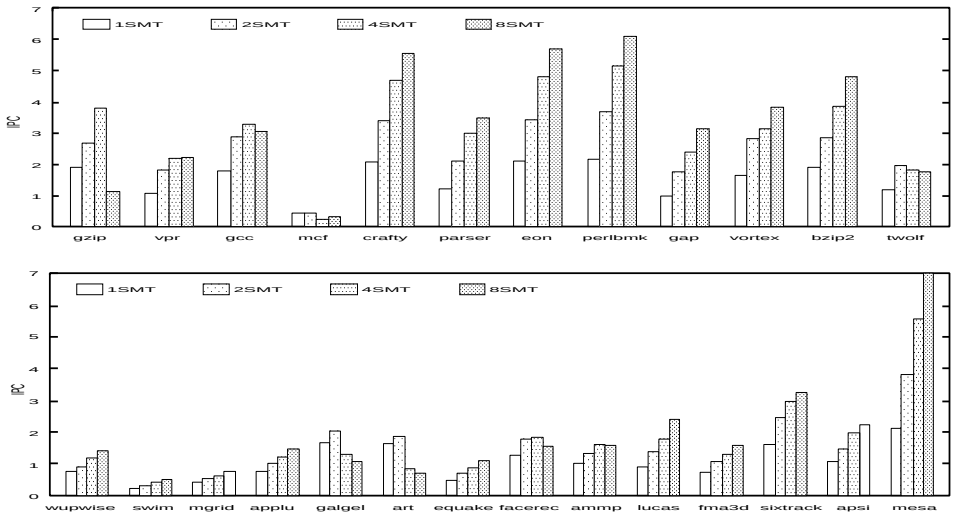

図 3　従来の SMT プロセッサの IPC 比較

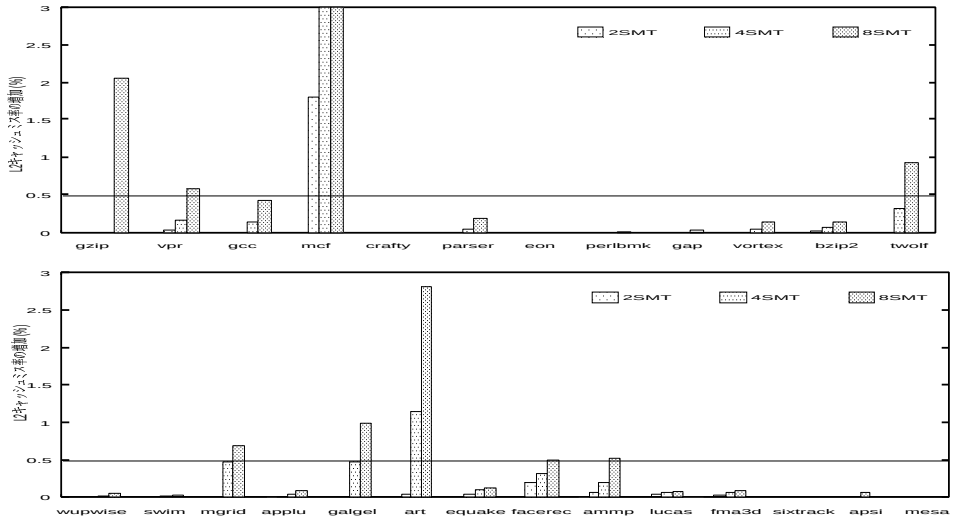

図 4　SMT プロセッサの実行多重度による L2 キャッシュミス率の増加

場合と比較して 70 ～ 80% の性能低下を起している。

性能低下の大きな要因としては、L2 キャッシュミス率の増加が考えられる。1SMT と比較した 2SMT ～ 8SMT での L2 キャッシュミス率の増加の様子を図 4に示す。上段には CINT、下段には CFP の各ベンチマークに関して、1SMT の L2 キャッシュミス率との差分が示されている。図 3において実行多重度が増加して IPC が低下しているケースには、L2 キャッシュミス率が大きく増加していることがわかる。実行多重度が増えた場合に、L2 キャッシュミス率が約 0.5% 以上増加すると IPC が低下している傾向が見て取れる。

この様に、資源競合による性能低下の度合いは、アプリケーションの性質と L2 キャッシュ構成（容量、連想度）とに大きく依存すると考えられ、一概に最適な実行多重度を固定することはできない。

4.3　DCML による性能改善に関する評価

4.2節では、SMT プロセッサの実行多重度を上げても性能が低下する場合があることを示した。ここでは、8SMT に DCML を適用することによって、如何に性能が改善されるかについて評価する。

4.3.1　実行多重度決定アルゴリズム

実行多重度を決定する選択指標として、ここでは IPC と L2 キャッシュミス率の 2 通りの評価を行った。以下に、実行多重度決定アルゴリズムについて説明する。

IPC　各実行多重度における IPC を比較し、チップトータル IPC が最も高い実行多重度を用いる。

L2 キャッシュミス率　原則、8 スレッドを同時実行する。但し、各実行多重度における L2 キャッシュミス率を比較して、L2 キャッシュミス率が 0.5% 以

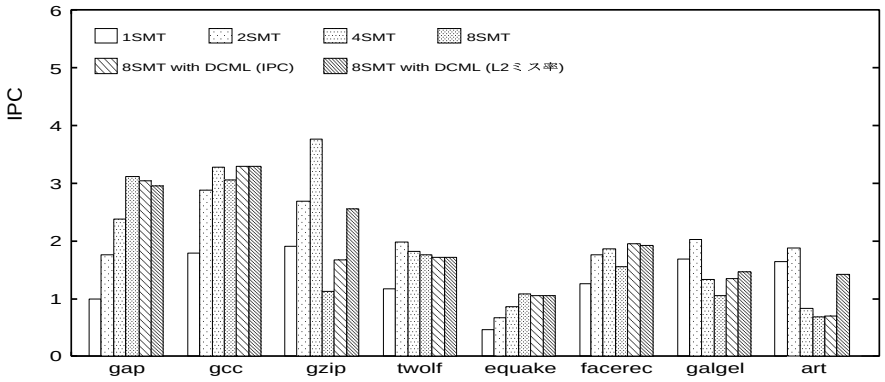

図 5 DCML による 8SMT の性能低下の低減

上、増加している場合には実行多重度を下げる。0.5% という閾値は、図 4の結果に基づいて設定した。

例えば選択指標として L2 キャッシュミス率を用いた場合に、実行多重度が 2 のキャッシュミス率が 0.6% で、実行多重度が 4/8 のキャッシュミス率が 1.2% という統計情報が得られれば、DCML では最適な実行多重度として 2 を選択する。

4.3.2 サンプリング時間と周期について

今回の評価では、統計情報のサンプリング周期は 100M サイクルとした。一般的に、数十〜数百 msec ごとにコンテキストスイッチが行われるため、数 GHz の周波数を想定すると数十〜数百 M サイクルごとには、サンプリングを行うべきであると考えた。そこで、50M 〜 400M サイクルのサンプリング周期でシミュレーションを行い、本論文では、その中から性能バランスの良かった 100M サイクルの結果について示す。

また、サンプリング時間の長さについても何通りかシミュレーションを行い、その中でも性能バランスの良かった 500k サイクルを用いた。sampling phase では、各実行多重度における場合の数で 15 通りについてサンプリングを行うため、sampling phase の長さは 7.5M サイクルとなる。従って、100M サイクルごとに合計 7.5M サイクルのサンプリングを行い、最適な実行多重度を決定することになる。

4.3.3 評価結果

図5に、DCML を適用することによる 8SMT の性能改善の様子を示す。特徴的な例として、CINT から gap, gcc, gzip, twolf の 4 種類、CFP から equake, facerec, galgel, art の 4 種類の結果を示している。図中には、比較のため従来の 1SMT 〜 8SMT における IPC と、実行多重度選択指標に IPC と L2 キャッシュミス率を用いた 2 通りの DCML の IPC が示されている。

図3で 8SMT の性能が大きく低下していた gzip, galgel, art では、DCML を適用することで 2 倍近い性能が得られている。最も性能が良い 2SMT/4SMT と比較しても、従来の 8SMT では 70 〜 80% の性能低下を起しているのに対して、選択指標として L2 キャッシュミス率を用いた場合には、性能低下を 35 〜 45% にまで抑えることができている。その他に、4SMT の方が性能が良い gcc と facerec において性能低下が防がれている。

5. 考　　察

5.1 DCML による性能向上の度合いについて

図 5の結果では、ベンチマークによって性能向上の度合いに違いが見られる。その要因について考察を行う。

gzip, galgel, art DCML によって大きな性能向上が見られる。しかし、従来の SMT で最も性能が良い場合と比較して性能差が残る。その原因として、以下の 2 点が考えられる。

- DCML で決定された実行多重度が、従来の 1SMT 〜 8SMT の中で最適と見られる実行多重度と異なるケースが存在する。例えば art では 2SMT の IPC が最も良いが、DCML では実行多重度として 1 を選択している事が多かったため、1SMT の性能に近くなっている。
- DCML では、選択した実行多重度に基づいて、100M サイクルのサンプリング周期を time sharing して 8 スレッドを実行する。そのため、time sharing された期間に実行されるスレッド数を減らす事はできるが、長期的なサンプリング周期全体で見た場合には、8 スレッド間で資源競合を起す可能性が残る。それに対して、例えば従来の 4SMT では、100M サイクルの間に 4 スレッドしか実行されず、4 スレッド間での資源競合の可能性しかない。そのため、8SMT に DCML を適用することで最適な実行多重度が選択されたとしても、従来の 1SMT 〜 4SMT よりは性能が低下すると考えられる。

表 2に、今回、評価した条件の DCML において最適な実行多重度が選択できたと仮定した、IPC の

理想値を示す。 galgel や art では、最適な実行多重度が選択できれば更なる性能向上が見込めるが、後者の要因によって従来の SMT での最高性能には及ばないことがわかる。

表 2 8SMT に DCML を適用した場合の理想 IPC

	従来の SMT での最高 IPC	DCML 評価の IPC	DCML での理想 IPC
gzip	3.77	2.56	2.59
galgel	2.02	1.45	1.66
art	1.88	1.43	1.54

gcc, facerec 従来の SMT で最も性能が良かった 4SMT と同等の性能が得られている。 facerec では 4SMT より高い性能が得られているが、これはベンチマークの処理フェーズによって最適な実行多重度が変化することに起因すると考えられる。具体的には、 4SMT の性能が良い処理フェーズと、 8SMT の性能が良い処理フェーズが存在する。従来の SMT では実行多重度が固定のため、いずれかの処理フェーズで僅かに性能低下を起していたが、 DCML では処理フェーズの性質にあった実行多重度で実行することできている。

gap, equake 従来、 8SMT の性能が最も良かったため性能改善は見られず、逆に gap ではサンプリングオーバーヘッドや、不適切な実行多重度の選択によって性能が低下している。

5.2 最適実行多重度決定に用いる選択指標について

DCML で最適な最適実行多重度を決定するために用いる選択指標としては、 IPC より L2 キャッシュミス率を用いた方が性能改善が大きい場合が見られた。特に、 gzip と art において性能差が大きい結果が得られた。 DCML でサンプリングした統計情報を見ると、実行多重度による IPC の差がそれほど明確でないケースも多く、適切な実行多重度が選択されていないことが多い。それに対し、実行多重度による L2 キャッシュミス率の増加の方が顕著に現れるため、より正しく最適な実行多重度を選択することができている。

6. ま と め

本論文では、 SMT プロセッサにおいて過剰なスレッドが同時実行された際にスレッド間の資源競合によって性能低下が起きる問題に対して、最適なスレッド実行多重度で SMT プロセッサを動作させることで性能低下を防ぐ手法を提案した。本論文で提案する DCML(Dynamical Control of Multithreading Level) では、一定周期で各実行多重度におけるプロセッサ統計情報を測定し、その情報を基に最適なスレッド実行多重度を動的に選択する。実行多重度が決定された後は、ハードウェアコンテキスト上にあるスレッド群の間で time sharing して、スレッドの動作状態／休止状態を切り替えて実行する。

SPEC2000 を用いたシミュレーション評価の結果、従来の SMT では実行多重度が増すに連れて L2 キャッシュミス率が大きく増加する場合があり、 2SMT/4SMT と比べて 8SMT の性能が最大 70 〜 80% 低下してしまうケースが見られた。本手法を用いることで 35 〜 45% の性能低下に抑えることができた。

今回の評価では、 DCML によって同時実行スレッド数を変化させても、長期的に見れば一定のサンプリング周期の間に 8 スレッドが実行される点は変らず、性能改善に上限があるという考察が得られた。今後の課題として、選択された実行多重度に応じてサンプリング周期を変化させる等によって、より大きな DCML の効果が得られないかを評価して行きたい。

参 考 文 献

1) Susan Eggers Dean Tullsen and Henry Levy. Simultaneous multithreading: Maximizing on-chip parallelism. *ISCA '95*, 1995.
2) Joel Emer Dean Tullsen, Susan Eggers and Henry Levy. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *ISCA '96*, 1996.
3) Hyper-Threading Technology. http://www.intel.com/technology/hyperthread/.
4) T.Kimbrel F.N. Eskesen, M. Hack and M.S. Squillante. Performance analysis of simultaneous multithreading in a powerpc-based processor. *WDDD*, 2002.
5) S. Hily and A. Seznec. Contention on 2nd level cache may limit the effectiveness of simultaneous multithreading. *IRISA report*, 1997.
6) Allan Snavely and Dean Tullsen. Symiotic jobscheduling for a simultaneous multithreading processor. *ASPLOS IX*, 2000.
7) Dean Tullsen Allan Snavely and Geoff Voelker. Symiotic jobscheduling with priorities for a simultaneous multithreading processor. *Sigmetrics*, 2002.
8) Matthew McCormick Omer Zaki and Jonathan Ledlie. Adaptively scheduling processes on a simultaneous multithreading processor. *technical report*, 2000.
9) Susan Eggers. Henry Levy Sujay Parekh and Jack Lo. Thread-sensitive scheduling for smt processors. *technical report*, 2000.
10) Larry Rudolph G.Edward Suh and Srinivas Devadas. Dynamic cache partitioning for simultaneous multithreading systems. *PDCS*, 2001.
11) 大河原英喜, 河場基行, 安里彰. 計算機システムにおける分岐予測外れパスの影響. 情処研報,ARC-144, Vol. 2001, , 2001.
12) 河場基行, 安里彰. トレース長を考慮したクラスタリングによる評価用トレース選択. コンピュータシステムシンポジウム予稿集, 2002.