LC_006

スレッド特徴量に基づくマルチコアプロセッサスケジューリング

Thread Scheduling Based on the Thread Characteristics for Multi-Core Processors

船矢祐介* 小寺功* 滝沢寛之*† 小林広明*†
Yusuke Funaya Isao Kotera Hiroyuki Takizawa Hiroaki Kobayashi

1 緒言

マイクロプロセッサの性能向上は，これまで主にクロック周波数の向上と，命令レベル並列性（ILP）を利用した並列処理能力の向上によって行われてきた．前者には半導体の微細化やパイプラインの段数を増やすといった方法が，後者にはスーパースカラや VLIW といった方法が挙げられる．しかし，チップの高集積化，高周波数化に伴う消費電力や発熱量の増大が深刻であること，また，従来から指摘されている ILP の限界 [1] によって演算ユニットの利用率が悪化していることなどから，従来の設計での性能向上は行き詰まりを見せている．

そこで今，ILP に代わってスレッドレベル並列性（TLP）が注目されている．TLP では，独立な，あるいは依存性の低い複数のスレッドから並列処理可能な命令を抽出するため，より高い並列度を実現できる．この TLP を利用する方式として次の 2 つがある．1 つは，従来のシングルコアに Simultaneous Multi-Threading（SMT）[2] を用いる方式である．SMT は，1 つのコアで複数のスレッドを同時に実行可能にする技術で，もともと利用率の悪かった演算ユニットの利用率を上げることが可能となる．もう 1 つは，プロセッサをシングルチップマルチコアにする方式である．マルチコアは，複数のコアを使い，複数のスレッドを同時に実行可能にするアーキテクチャである．2 つの方式は，共にクロック周波数を上げずに実行性能を上げることができ，従来の設計の問題を解決できると期待されている [3]．

本報告では，2 つの方式を両立するアーキテクチャであるマルチコアで，各コアでは複数のスレッドを同時に実行する SMT が可能なプロセッサに焦点を当てて考える．このアーキテクチャによってプロセッサのリソースを無駄なく利用し，過剰なリソース投入を行うことなくプロセッサ全体の性能を向上させることを狙う．現在シーケンシャルなコードから投機的に並列実行できるスレッドを生成し TLP を利用する試みが行われている [4]．このように同一プロセスに属する複数のスレッドの場合，各スレッドは同じような特徴を持つため，どのコアにどのスレッドを割り当てるかというスレッドスケジューリングは単純でよく，現在のように OS による動的な割り当てで十分である．しかし異なるプロセスに属する複数のスレッドを扱う場合，スレッドは異なる特徴を持ち，スレッドスケジューリングの結果によって実行性能が大きく変わることが予想される．従って，プロセッサ全体として最も高い性能を発揮するためには，今まで扱われてこなかったスレッド固有の特徴に注目し，各スレッドの特徴に応じた最適なコア内・コア間スレッドスケジューリングを行う必要がある．

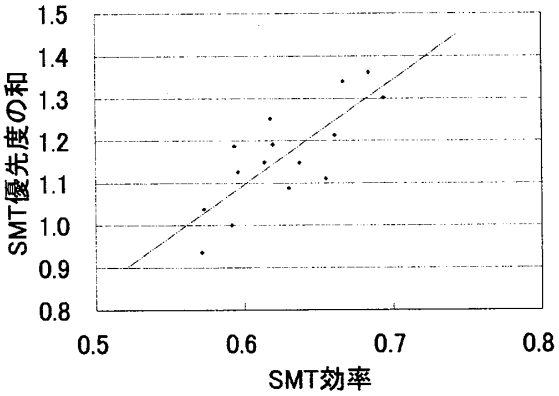

図 1 SMT 優先度の和と SMT 効率の相関関係

そこで本報告では，スレッドの特徴量を定義し，各スレッドの特徴を定量的に評価する．さらに，その特徴量に基づいたスレッドスケジューリング手法を提案し，その有効性をシミュレーションにより評価する．

2 スレッドスケジューリング

複数のスレッドを実行するとき，シングルコアの SMT ではマルチコアに比べて同時実行するスレッドの組み合わせによって実行性能が大きく変化する．マルチコアでは演算ユニットの競合がないので，各スレッドは単独の時と同等の実行性能を発揮できる．それに対しシングルコアの SMT では，各スレッドが演算ユニットを共有するため，演算ユニットの利用率が高いスレッドの組み合わせでは，演算ユニットに対する競合が発生し，性能が悪化する可能性がある．

プロセッサのコア数よりも多い数のスレッドを実行する場合，スレッドの一部をシングルコアの SMT で実行することになる．そのためコア内・コア間スレッドスケジューリングでは，まず SMT で実行するのに適切なスレッドの組み合わせを見つけることが非常に重要である．

3 SMT 優先度に基づくマルチコアスレッドスケジューリング手法

本報告では，適切なスレッドスケジューリングを実現するために，スレッド特徴量として SMT 優先度を提案し，さらに SMT 優先度に基づくコア内・コア間スレッドスケジューリング手法を提案する．

* 東北大学大学院情報科学研究科
† 東北大学情報シナジーセンター

3.1 SMT 優先度

適切なスケジューリングのためには，あるスレッドの組み合わせが SMT を利用して同じコアで実行するのにどの程度適切であるかを表すスレッド特徴量が必要である．そこでまず，SMT の実行効率を定義する．2つのスレッドを1コアで SMT を利用して実行した場合と，2つのスレッドにそれぞれ1コアずつ割り当てて2コアを使って実行した場合の実行性能を IPC（Instructions Per Cycle）で計測する．後者の，スレッドを2コアで実行したときの性能を最大性能と考え，それに対する SMT 実行時の性能の比率を式 (1) に示す SMT 効率として定義する．ただし IPC_{1core}，IPC_{2cores} はそれぞれ1コア時の IPC，2コア時の IPC を表す．

$$\text{SMT 効率} = \frac{\mathrm{IPC}_{1core}}{\mathrm{IPC}_{2cores}} \tag{1}$$

これによって，SMT 効率が高い程その2つのスレッドの組み合わせは SMT に適していると定義できる．しかし，SMT 効率は2つのスレッドの組み合わせに対しての定義であり，個別のスレッドからは得られない．このため SMT 効率を直接スケジューリングに用いることはできない．そこで，単独のスレッドから SMT 効率を予測する特徴量を定義する．SMT 効率が低下するのは，同じコアで実行する2つのスレッドが競合し，共有する演算ユニットをうまく使えない場合である．原因は2つ考えられる．

まず演算ユニットの利用率が高く演算ユニットの空きが少ない場合である．あるスレッドが演算ユニットを使用している間もう一方のスレッドはその演算ユニットを利用することができない．

次に演算ユニットへ直接命令を発行する各命令ユニットのキューに空きがなく，命令発行がストールしてしまう場合である．各命令ユニットのキューとは整数演算ユニット・浮動小数点演算ユニットのリザベーションステーションおよびロードユニット・ストアユニットのバッファを指し，これらをまとめて命令キューと呼ぶことにする．あるスレッドの命令が命令キューの1つを占有した場合，たとえその演算ユニットが空いていたとしても，もう一方のスレッドはその演算ユニットを使うことができない．実際，Intel のハイパースレッディングテクノロジでは，片方のスレッドが利用可能な命令キューのエントリ数に上限を設定し，片方のスレッドが完全にストールするという最悪の事態を防止するようにしている [5]．しかしその上限はハードウェアのキューサイズによって決められる固定されたものであり，スレッドが必要としているエントリ数とは無関係である．エントリを多く必要とするスレッドはエントリが不足してパフォーマンスが低下する，一方エントリをあまり必要としないスレッドは，過剰に割り当てられたエントリを使用しないで遊ばせることになる．この問題を解決するためには，ハードウェアによる固定的な資源制約を与えなくとも適切な資源配分が可能なスレッドスケジューリング技術が必要不可欠である．

以上2つの原因を定量的に表すために，予備実験にて事前に1つのスレッドを1つのコアで実行し，その結果を分析した．

表1　本実験で使用する主なパラメータ

CPU 数	3
SMT コンテキスト	2
フェッチ命令数	8
発行命令数及びリタイア命令数	4
ロードユニット及びストアユニット数	2
整数演算ユニット数	3
浮動小数点演算ユニット数	2

まず全ての命令が滞りなく実行される理想の場合を考える．この時，複数同時発行のプロセッサにおける理論的な最大処理能力を

$$\mathrm{C}_{ideal} = \text{実行にかかったサイクル数} \times \text{命令発行幅} \tag{2}$$

とする．これを用いて演算ユニットの利用率を次の式で定義する．

$$\mathrm{C}_{busy} = \frac{\text{実際に発行された命令数}}{\mathrm{C}_{ideal}} \tag{3}$$

次に，あるサイクルで1つの命令が発行されなかった原因を，命令キューの場合（C_{instQ}）とそれ以外（C_{other}）に分けると，次式が成立する．

$$\mathrm{C}_{busy} + \mathrm{C}_{instQ} + \mathrm{C}_{other} = 1 \tag{4}$$

前述の考察より，あるスレッドが SMT に適しているということは C_{busy} と C_{instQ} が小さい，つまり C_{other} が大きいということである．よって C_{other} を SMT 優先度として式 (5) で定義する．

$$\text{SMT 優先度} = 1 - \mathrm{C}_{busy} - \mathrm{C}_{instQ} \tag{5}$$

図1は，4節で使用する6種類のアプリケーションの中から2種類を組み合わせてマルチコアシミュレータ上で実行し，得られた SMT 効率を横軸に，SMT 優先度の和を縦軸にとったグラフである．図1が示すように，両者には強い相関関係が見られるので，2つのスレッドの SMT 優先度の和が高いほど，そのスレッドの組み合わせは SMT に適していると推測できる．

3.2 スレッドスケジューリング手法

SMT 優先度を用いたスレッドスケジューリング手法を提案する．スケジューリングは次の手順で行う．

1. 事前にプロファイリングによって個々のスレッド特徴量として SMT 優先度を取得する．
2. 同時に実行するスレッドの数がプロセッサのコア数を越える場合，SMT 優先度の和が最大となる組み合わせを選択し，1コアにスケジューリングする．
3. 残りのスレッドの数が残りのコアの数以下になるまで2を繰り返し，それ以降の残りのスレッドを残りのコアに1つずつ割り当てる．

4 性能評価

4.1 実験方法

提案手法の有効性をシミュレーションによって検証する．シミュレーションにはマルチコアシミュレータ SESC[6] を使用する．SESC は，スーパースカラのプロセッサコアを複数持つマルチコアプロセッサの動作をシミュレーションする．パイプラインは，フェッチ，デコード，命令発行，演算実行，リタイアメントの 5 ステージからなる．演算実行ステージには整数演算や浮動小数点演算など 5 種類の演算ユニットがあり，アウトオブオーダで演算を実行する．SESC は SMT に対応しており，複数のスレッドの命令を同時に発行することができる．本実験で用いる主なパラメータを表 1 に示す．SMT コンテキストは 1 コアで同時実行できるスレッドの数を示す．また，ベンチマーク用のアプリケーションを用いて，1 アプリケーションを 1 スレッドとして実行する．使用するアプリケーションは，SPEC CPU2000[7] に含まれている gzip, gcc, mcf, mesa, art, equake の 6 種類である．

シミュレーションでは 3 コアのマルチコアプロセッサで 4 スレッドを同時に実行する．6 種類のベンチマークからスレッドとして実行する 4 つを選ぶ組み合わせは，全部で 15 組あり，さらに選んだ 4 スレッドを 3 コアにスケジューリングする組み合わせは 6 種類ある．今回はその組み合わせすべてに対してシミュレーションを行い，提案手法の有効性を評価する．

提案手法で用いる SMT 優先度には，特徴量として C_{busy} と C_{instQ} の 2 つが含まれている．そこで，提案手法と C_{busy} だけを用いる手法を比較し，2 つの要素の有効性も検証する．なお，本実験では提案手法を「手法 1」，C_{busy} だけの手法を「手法 2」と呼ぶことにする．

4.2 実験結果および考察

手法 1 および手法 2 が，ある 1 組の中の 6 種類の組み合わせで何番目に性能がよかったかを IPC で比べた結果を表 2 に示す．手法 1 は 15 組中 11 組で最も性能がよく，残りの 4 組でも 2 位であった．3 位以下になった組はなく，最悪な場合でも 2 番目に性能がよい組み合わせを選択できている．一方，手法 2 が最も性能がよかったのは 5 組しかなく，3 位以下になった組も 3 組あった．これより C_{busy} だけを考慮した手法 2 よりも，C_{instQ} も考慮に加えた提案手法のほうが有効であるといえる．

さらに手法 1 と手法 2 を詳細に調査したところ，15 組中 8 組でスレッドスケジューリングの結果が同じであった．提案手法が C_{instQ} も考慮しているにもかかわらず手法 2 と同じスケジューリングになったのは，実行した 4 つのスレッドが，C_{busy} が低いスレッド 2 つと，C_{busy} が高いスレッド 2 つに明確に分かれる組み合わせであった．例えば，4 つのスレッドが gzip, gcc, mesa, equake の場合が挙げられる．図 2 に示した各ベンチマークの C_{busy} を見ると，gzip と gcc は非常に低く，mesa と equake は非常に高い．逆に，実行した 4 つのスレッドのうち 3 つ以上のスレッドの C_{busy} が近い組み合わせでは，手法 1 で示した提案手法の方が高い性能となった．例えば，4 つのスレッドが mcf, mesa, art, equake の場合，図 2 より mesa, art, equake の 3 つの C_{busy} が近い値である．

表 2 6 通りの組み合わせ中の各手法の順位の回数

順位	1 位	2 位	3 位	4 位	5 位	6 位
手法 1	11	4	0	0	0	0
手法 2	5	3	4	0	3	0

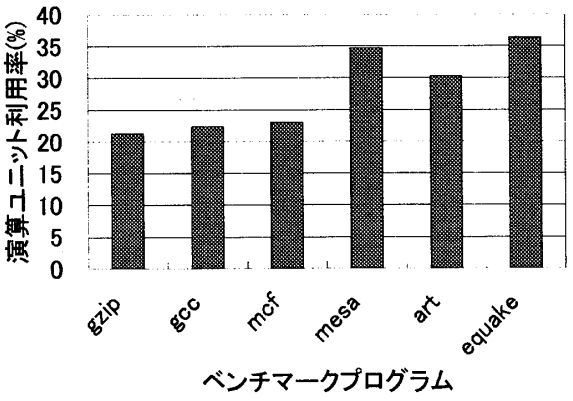

図 2 各ベンチマークの演算ユニットの利用率

以上の結果より，C_{busy} が SMT 効率に及ぼす影響は大きく，SMT 優先度の中で大きなウェイトを占めているといえる．そのため C_{busy} に大きな差があるスレッドの組み合わせでは手法 2 と同じスケジューリング結果となった．しかし C_{busy} が同じ程度のスレッドの組み合わせを選ぶときは，SMT 優先度の中の C_{instQ} が有効に機能することから，提案手法のほうがより的確に SMT 効率を予測し適切なスケジューリングができることが示された．

次に 15 組のシミュレーション結果の平均値を図 3 に示す．それぞれの組で最も性能のよかったスレッドの組み合わせを性能の上限値として「最良」で示し，逆にそれぞれの組で最も性能の悪かったスレッドの組み合わせを性能の下限値として「最悪」で示している．

提案手法である手法 1 は「最良」にはわずかに及ばないものの，かなり近い実行性能を達成している．表 2 で示した通り 11 組で最も性能のよい組み合わせを選択できただけでなく，2 番目の性能の組み合わせを選んでしまった 4 組でも，その性能は最もよい組み合わせに非常に近かったためである．提案手法によってどのようなスレッドの組に対しても最適，もしくは最適に非常に近い組み合わせが選択できたといえる．一方，手法 2 はいくつかの組で性能が非常に低い組み合わせを選んでしまったため，平均して手法 1 よりも低い性能となった．

最後にそれぞれのコアごとの実行性能を図 4 に示す．図 3 と同様に 15 組の平均値である．3 つのコアのうち SMT を用いて 2 スレッドを同時実行した 1 つのコアの性能を「SMT」で示し，1 スレッドのみ実行した 2 つのコアの性能は合計して「Multi-Cores」で示す．

手法 1 と手法 2 を比較すると，SMT で 2 スレッドを実行し

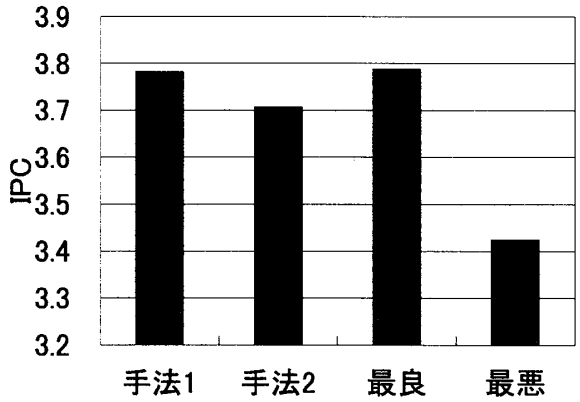

図3　プロセッサ全体の実行性能（IPC）の比較

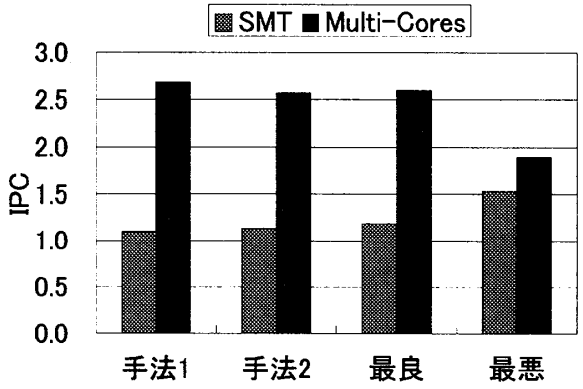

図4　プロセッサ内のコアの実行性能の比較

たコアの実行性能の差はほとんどなく，1スレッドを実行した2つのコアの実行性能では手法1の方が高い．これは1スレッドずつマルチコアで実行する時に，共有2次キャッシュでの競合などの理由により単独実行時と同等の性能を出せないスレッドの組み合わせがあるためと考えられる．提案手法で用いるSMT優先度はSMT効率を予測する特徴量である．またSMT効率の定義からSMTに適していないと判断されたスレッドはマルチコアで実行したときに高い性能が得られると判断されたのと同じである．よってSMT優先度を用いることでSMTで実行すべきスレッドとマルチコアで実行すべきスレッドの両方が適切に選択され，その結果プロセッサ全体としての実行性能が手法2よりも高くなることが明らかになった．

以上の結果から，マルチコアプロセッサの潜在能力を最大限に引き出すという目的に対し，提案手法のSMT優先度に基づくマルチコアスレッドスケジューリングが有効であることが示された．

5　結言

シングルチップマルチコアで，各コアではSMTが可能なプロセッサにおいて，プロセッサの潜在能力を最大限に引き出すということが本研究の目的である．そのためにはスレッドの特性を考慮したスケジューリングが必要である．その実現のために本報告ではスレッドの特徴量としてSMT優先度を提案し，さらにそれを用いたスレッドスケジューリング手法を提案した．また評価実験によって提案手法の有効性を示した．

今後の課題として，各命令ユニットのキューに空きがなく命令発行がストールしてしまうことが，SMT優先度に基づくスケジューリングにどの程度影響を及ぼしていたのかを定量的に評価することが挙げられる．また，本報告とは異なる実験条件，例えば違うアプリケーションを使用した場合やコア数を増やした場合の性能評価，提案手法の実装が挙げられる．特に実装については，事前にSMT優先度を取得する方法，取得したSMT優先度をスケジューラに知らせる方法などが大きな課題になる．また本報告ではスケジューリングにかかるオーバーヘッドを考慮していないため，スケジューリングのオーバーヘッドを考慮した性能評価が必要である．さらに今後は，性能向上に加えて消費電力の削減にも貢献するスケジューリング手法へ改良することを予定している．そのためには積極的にSMTを利用して少ない数のコアでスレッドを実行し，使われていないコアの電力供給を停止するといった方法が考えられる．

参考文献

[1] Wall, D.W., "Limits of Instruction-Level Parallelism," Proc. of ASPLOS-IV, 1991.

[2] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, Dean M. Tullsen, "SIMULTANEOUS MULTITHREADING:A Platform for Next-Generation Processors," IEEE Micro, 1997.

[3] 笠原博徳, 木村啓二, "マルチコア化するマイクロプロセッサ," 情報処理学会, Vol.47, No.1, pp.10-16, 2006.

[4] W.Liu et al., "POSH:A TLS Compiler that Exploits Program Structure," Proc.ACM SIGPLAN Symp.Principles and Practice of Parallel Programming, March 2006.

[5] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A.Koufaty, J. Alan Miller, Michael Upton, "Hyper-Threading Technology Architecture and Microarchitecture," Intel Technology Jornal Q1, Vol.6, Issue 1, pp.4-15, 2002.

[6] SESC, "http://sesc.sourceforge.net"

[7] Standard Performance Evaluation Corporation, "http://www.spec.org"

謝辞

本研究に対し，日頃貴重なご助言をいただくスタンフォード大学マイケルフリン教授，東北大学中村維男教授，後藤英昭助教授，鈴木健一講師，江川隆輔博士に感謝する．本研究の一部は，文部科学省科研費基盤研究（B）課題番号18300011の援助を受けている．