



# Simultaneous Multithread (SMT) アーキテクチャの現状と今後

東京農工大学工学部情報コミュニケーション工学科  
中條 拓伯 [nakajo@cc.tuat.ac.jp](mailto:nakajo@cc.tuat.ac.jp)

東京農工大学工学部情報コミュニケーション工学科  
河原 章二 [shoji@nj.cs.tuat.ac.jp](mailto:shoji@nj.cs.tuat.ac.jp)

和歌山大学情報システム工学部情報通信システム学科  
上原 哲太郎 [tetsu@sys.wakayama-u.ac.jp](mailto:tetsu@sys.wakayama-u.ac.jp)

東京農工大学工学部情報コミュニケーション工学科  
並木 美太郎 [namiki@cc.tuat.ac.jp](mailto:namiki@cc.tuat.ac.jp)

## 背景：高性能プロセッサの必要性

パーソナルコンピュータ (PC) が家庭や、小中学校などの教育機関に浸透し、また Windows98/Me/XP/MacOS など、使い勝手のよいオペレーティングシステム (OS) を利用できるようになってきた。その上で、ネットサーフィンにより必要な情報を引き出ししたり、かつてはテキストのみのやりとりであった電子メールも、カラー画像や音声はもとより、動画の送受信も可能な時代が到来した。さらに、単に PC をネットワークに接続するだけでなく、デジタルカメラやデジタルビデオ、DVD プレーヤ、高品位カラープリンタ、その他さまざまな電子機器が PC に接続されるようになってきた。また、今や爆発的な普及をみせた携帯電話においても、通話用途以上に、メール機能に加えて World Wide Web (WWW) のブラウザ機能や、静止画、動画の閲覧、鑑賞といった用途が広まりつつある。

そのような状況の中においては、汎用プロセッサ<sup>☆1</sup>や、機器に組み込まれる組み込み用プロセッサ<sup>☆2</sup>の性能向上が切に望まれ、単にクロックスピードを向上させるだけでは対応できなくなっている。

かつて、汎用の大型計算機で利用されていたマルチタスク技術<sup>☆3</sup>は、UNIX の普及と、それをベースにした PC 用の OS の進化とともに今や当たり前のように利用できるようになり、PC 内で多数のプロセスやスレッド<sup>☆4</sup>が実行されるようになった。

一般ユーザが Windows 環境で普通に PC を用いた場合には、アプリケーション実行の前には約 20 個の OS スレッドが実行されているといわれている。大容量のメモリを搭載し、バックグラウンドでエディタなどのアプリケーションをあらかじめ立ち上げているようなユーザの場合はさらにその数が多くなるであろう。その状態でインターネットブラウザを起動し、有用な PDS や、画像などを含むさまざまな情報をダウンロードしている間に MP3 の音楽を聞いたり、PDF ファイルを印刷し

☆1 汎用プロセッサ (General Purpose Processor) : パーソナルコンピュータ (PC) やワークステーションに搭載される、ワープロ、データベースなどの事務処理や科学技術計算などの一般的な目的に使用されるプロセッサ。

☆2 組み込み用プロセッサ (Embedded Processor) : 電子機器、OA 機器、携帯機器などに組み込まれ、主にその制御用として使用されるプロセッサ。

☆3 マルチタスク (Multi-task) : コンピュータシステムにおいて、複数の処理 (タスク) を同時に実行すること。厳密には高速に逐次処理を実行し、短時間に複数のタスクを次々に切り替えることで、あたかも複数のタスクが同時実行されているようにみえるようにするためのオペレーティングシステムの技術である。

☆4 プロセス (Process)、スレッド (Thread) : オペレーティングシステム (OS) において、仮想メモリ空間などの資源割当ての単位をプロセスと呼び、そのプロセス内で生成されるプログラムの実行単位をスレッドと呼ぶ。また、コンパイラにより抽出された複数の命令列を 1 つの処理単位として投入されるアーキテクチャサイドからみたスレッドもあり、スレッドについては、厳密な定義はまだ存在せず、本稿では OS により生成されるスレッドを、特に OS スレッドと呼んでいる。

ている間にも、実行されるOSスレッド数はさらに伸びていく。

プロセッサの高速化技術で近年効力を発揮したものはスーパスカラ<sup>☆5</sup>であるが、この技術は、プロセッサ単体が1つのプログラムを実行する場合のピーク性能を高めることには成功した。しかしながら、現状における命令レベル並列性 (ILP: Instruction Level Parallelism) では、今後スーパスカラにおけるこれ以上の性能向上を見込むのは難しい。

さらに、前述のような多数のOSスレッドが存在し、それらが高速に切り替える必要があるような場合には、さまざまな要因 (分岐予測<sup>☆6</sup>が外れたり、キャッシュミスが生じるなど) により、性能低下を余儀なくされてしまう。

今後、十分にILPを引き出すこととともに考慮しなければならないのが、スレッドレベルの並列性 (TLP: Thread Level Parallelism) であり、ILPとTLPをうまく併用した形でシステムを実現することが必要不可欠なものとなると考えられる。

その解の1つが、本稿で紹介するSMT (Simultaneous Multithreading) 技術であり、さまざまな研究機関とともに、プロセッサベンダも開発および実用化に向けて本腰を入れている。

本稿では、まず市場におけるプロセッサの動向について触れ、次に、SMTアーキテクチャの説明を行い、SMTアーキテクチャの基本概念とともに、商用ベース、研究レベルにおけるSMTアーキテクチャの研究事例について簡単に紹介する。さらにSMTアーキテクチャの性能の鍵を握るコンパイラやオペレーティングシステム (OS) の技術的な問題点に触れ、最後にまとめとして、今後の方向性について考察する。

## 最新プロセッサの方向性

### ■クロックの高速化

世界最初のマイクロプロセッサであるIntel 4004が1971年2,250個のトランジスタを集積し、10 $\mu$ のプロセ

ス技術、750KHzのクロック周波数、0.06MIPSの演算能力で登場して以来、半導体技術やCAD技術の利用をもとに、大型計算機において培われたアーキテクチャ上のさまざまなイノベーションがマイクロプロセッサに組み込まれてきた。そして、同じIntel社のPentium4は、現在2,350万トランジスタを集積し、2GHzの動作周波数で稼働していることから、トランジスタ数では100万倍以上、周波数についても約3,000倍もの向上をこの30年間で果たしたことになる。

特にここ数年の動作周波数の向上は目覚ましく、Intelをはじめ、AMD、SUN、IBMなどのプロセッサベンダは、こぞって主にクロック周波数の引き上げと、アーキテクチャの一部を改善することに頼って性能向上を目指してきた。

### ■複数プロセッサの利用とオンチップマルチプロセッサ (CMP) 化

ハイエンドのPCにおいて、2個～4個のプロセッサを搭載した対称型マルチプロセッサ (Symmetric Multi-Processor: SMP) <sup>☆7</sup>はすでにサーバとしての利用を目的に浸透してきており、また、ここ数年の間にデュアルプロセッサ対応の安価なマザーボードが登場して、一般ユーザもその並列処理機能を利用できるようになってきた。

半導体のプロセスルールが今後さらに精細化し、10億トランジスタの時代に突入する以前に、すでにいくつかのプロセッサベンダは、1つのチップの中に複数のプロセッサを搭載するオンチップマルチプロセッサ (CMP) 技術を取り入れた。たとえばそのCMP化を視野に入れた、AMDのHammerをはじめ、汎用プロセッサの分野でもCMPの製品が現れつつある。

CMP技術は、ネットワークプロセッサなど、定型的な処理が中心の特別用途プロセッサにおいてはすでに一般的となり、また研究レベルにおいても、Stanford大学のHydraプロジェクトや、名古屋大学のSKY、九州大学のFUCE、並列・分散処理研究推進機構 (PDC) において、慶應義塾大学を中心にCMPの内部キャッシュ方

☆5 スーパスカラプロセッサ (Superscalar Processor) : バイプライン方式のプロセッサにおける、命令フェッチ、命令デコード、実行などのためのユニットをそれぞれ複数用意し、単一プロセッサにおいて、複数の命令コードを同時にフェッチ、デコードし、その命令間に依存関係がなければ自動的にハードウェアにより実行順序制御を行い、1クロックサイクルで同時に複数の命令を実行する方式の1つである。

☆6 分岐予測 : バイプラインプロセッサを効率よく駆動させ、プロセッサの性能を向上させるための技術で、以前の条件分岐命令が分岐したか、しなかったかという情報をもとにして、バイプライン中に投入された分岐命令に続く命令列の実行を開始する。予測がヒットした場合は、それらの命令の処理を続けるが、分岐の予測が外れた場合は後続のバイプラインの内容を捨て、命令をフェッチからやり直す必要がある。この予測ミスに伴う処理の遅れが問題となる。

☆7 対称型マルチプロセッサ (Symmetric Multi-Processor: SMP) : マルチプロセッサシステムにおいて、各プロセッサの用途がすべて対等となっているシステムをいう。すべてのCPUから共通にアクセスできるメモリシステムが必要があり、一般に共通のバスに接続された共有メモリが用いられる。

式などについての研究が進められてきた。また、NECのMP98のように、携帯端末をターゲットにした低消費電力のプロセッサが実用段階に入っている。

単体のプロセッサコアで性能が不十分であるならば、複数のコアを用いて性能向上を図ろうとすることは自然な流れであり、従来の並列処理が求めるものをそのままチップの中に実現しようとするものである。しかしながら、単に複数のプロセッサコアを投入するだけで性能向上が見込めるわけではなく、メモリ技術など、その周辺においてさまざまな工夫が必要となってくる。

## ■ Simultaneous Multithreading (SMT) の登場

以上のような状況の中で、単一プロセッサでありながら、性能向上を実現できる Simultaneous Multithreading (SMT) が登場してきた。

マルチスレッド技術<sup>☆8</sup>自体はそれほど新しいものではなく、Tera Computerの創設者の1人であるBurton Smithが1974年に設計を開始し、1982年に最初に出荷したHEPに端を発するといわれている。

マルチスレッド・アーキテクチャの目的は、スレッドがアクセス遅延の大きい命令を実行した場合に切り替えを行うという、遅延を隠蔽するためのものであり、他にもMITのAlewifeやBill DallyのM-Machine、UC BerkleyのTAMなどが挙げられる。

これらのアーキテクチャに関してまとめたリンク集としては、文献1)のページがあり、プロセッサアーキテクチャの動向を知る手助けとなるであろう。

そのような背景の中から、TLPとILPとをうまく融合し、効率的に複数のスレッドを実行し、無駄となっていたリソースを最大限に有効活用するための技術として、SMTが提案された。これは、プロセッサにおいて、マルチタスクOS上で実行される複数のタスク(プロセス、スレッド)を明示することで、複数のパイプライン上で並列処理を行うものである。

複数のリソースを命令レベルで並列に活用する技術としてはVLIW (Very Long Instruction Word) アーキテクチャ<sup>☆9</sup>があり、DSP (Digital Signal Processor) などで有効に利用されつつある。しかしながら、並列に実行可能な命令列をコンパイラにより、1つの命令にまとめておかねばならず、命令セットも異なるため、従来の命令との互換性を保つことは困難である。これに対

して、SMTでは、OSやコンパイラのサポートが必要なものの、各スレッドの命令は既存の命令を流用できる。

かつてAlphaプロセッサの技術がDECからCompaqへと移った頃、将来のAlphaプロセッサ (EV8) にSMTを実装すると発表されたが、EV8が日の目を見る前に開発が終了し、その知的財産権はインテルに委譲された。そして、先頃IntelはSMTを最初に実装したプロセッサ技術として、開発コード「Jackson」と呼ばれるプロジェクトにおいて開発されたHyper Threadingを発表した<sup>7)</sup>。Hyper Threadingは1つのプロセッサに対して、2つのスレッドを割り当てられるようにしたものである。ソフトウェア (OS) 側からみたとき、Hyper Threadingを搭載したプロセッサは仮想的に2個のプロセッサとして見えるようになり、2001年11月にデンバーで開催されたSC2001において実際に2つの異なるアプリケーションを同時に実行するデモが行われていた。

しかしながら、SMTは仮想的に複数のプロセッサが動作しているようには見えるものの、CMPほどの並列効果は期待できない。実際、Hyper Threadingでの性能向上は現在のところ10～30%程度であるとみられているが、単一プロセッサでの性能向上としては飛躍的な数値であるとも考えることもできる。

今後は、SMTアーキテクチャをベースとした複数のプロセッサコアを1つのチップに搭載したプロセッサが登場してくるであろう。先に述べたように、定型的な処理を行うネットワークプロセッサなどの組み込み用プロセッサにおいては、すでに多数のコアを1つのチップに投入してきている。

たとえば、Broadcom社は、MIPSベースのコアを2個搭載したネットワークプロセッサBCM1250を発表し、将来はそのコアの数を増やしていく方向にある。しかしながら、汎用プロセッサの場合は、ILP、TLPともに十分な並列性を抽出できなければ多数のSMTコアを内蔵することは実用的でなく、研究すべき課題はまだ数多く存在する。次章ではSMTの基本的な概念やその動向について解説する。

☆8 マルチスレッド (Multi-Thread) : マルチタスクシステムがプロセスを切り替えるには、レジスタの内容を保存し、次に切り替わるプロセスのためのレジスタの値をロードするなど、負荷が大きい。同一プロセス内のOSスレッド間では、切り替えに要する負荷が小さい。このOSスレッドを用いて、同一プロセス内においてマルチタスク処理を可能としたものがマルチスレッドである。

☆9 VLIWプロセッサ (Very Long Instruction Word Processor) : 1つの命令語中に、複数の命令を並べ、それらをすべて同時に実行する形式のプロセッサアーキテクチャである。各命令間の依存関係を事前にコンパイラによって解析し、最適化しておく必要があるため、コンパイラ技術が重要な鍵となる。しかしながら、スーパースカラのような動的な実行順序制御機構が不要となる。

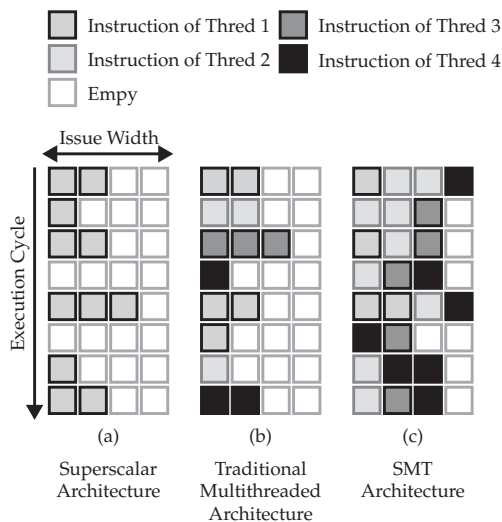


図-1 高性能プロセッサアーキテクチャにおける命令発行

## SMTアーキテクチャ

### ■ SMTアーキテクチャの概念

前述の通り、近年のプロセッサは、半導体技術の進歩により大規模な回路を集積できるようになり、複数のパイプライン、スーパスカラ技術の実装により高速化を目指してきた。しかしながら、コンパイラが十分なILPを引き出せなかったり、分岐予測に失敗したりすることで、パイプラインや演算器などのプロセッサ内のリソースが無駄になることが多かった。

図-1にプログラムの実行速度向上を目指した高性能プロセッサアーキテクチャの命令発行の流れを示す。

図では、横軸がプロセッサの同時命令発行数、縦軸が実行サイクルを示し、上から下へと実行サイクルが進むものとする。図-1 (a) は従来のスーパスカラプロセッサやVLIWプロセッサの命令発行の様子を示している。スーパスカラプロセッサは、全サイクルを通して実行するスレッドは1つである。図-1 (a) 中の空白が示すように、キャッシュミスなどの要因により実行スレッドが停止した場合、命令を発行できないサイクルが生じる。この無駄なサイクルをVerticalWaste (VW) と呼ぶ。一方、命令実行可能なサイクルにおいても、不十分なILPにより、1, 2命令程度しか命令実行ができない場合もある。この命令発行の空白をHorizontal Waste (HW) と呼ぶ。

(b) では、TeraのMTAやMITのAlewifeなどに代表

される、マルチスレッド・アーキテクチャにおけるプログラムの実行の様子を示す。これらのアーキテクチャは、複数のスレッドを切り替え、各サイクルにおいてそれぞれのスレッドに含まれる命令を実行する。これによりVWに関しては解決しているものの、依然としてHWの問題が残されたままである。これは、マルチスレッド・アーキテクチャが実行サイクルにだけ着目しており、1サイクル中の命令発行数はスーパスカラと同様にILPにのみ頼っているからである。

そこで、(c) に示される実行形態を持つSMTアーキテクチャが登場した。(a), (b) と大きく異なる点は、1サイクル中に複数のスレッドの命令を同時に実行していることである。これにより、VWのみならずHWをも解決することができ、プロセッサ内部のリソースを有効に活用することで性能向上を図ることが可能となる。

### ■ SMTアーキテクチャの研究事例と動向

SMTアーキテクチャに関しては文献2) のページに、Washington大学においてTullsenらを中心にAlphaをベースとしたSMTに関する研究成果<sup>3)</sup>とともに、さまざまな最新の話題が提供されている。

元々SMTアーキテクチャは複数のプログラムを同時に実行するアーキテクチャとして提案されていた。しかしその後、シングルプログラムをSMTプロセッサで動作させたときのシミュレーション結果が報告され、並列化することによって、そのシングルプログラムの高速化の可能性を示している。その結果、5つのSPLASH-2ベンチマークプログラムを用いた場合、SMTプロセッサのInstructions Per Cycle (IPC) は平均5.91, 最大6.83となることが報告され、その潜在的な性能が示された。

また、Chappellらや佐藤らは、主スレッドに対してサブスレッドが最適化を行うことにより、主スレッドの実行速度を改善する手法を提案している<sup>4)</sup>。

SMTアーキテクチャは、ハードウェアの複雑化が問題点として挙げられているが、十分にスレッドが生成できるのであれば、Out Of Order (OOO) 実行を行う必要がないと主張している研究もある。この報告は興味深く、In Order実行の機構を視野に入れることにより、スーパスカラのハードウェアの複雑さを回避する可能性を示すものである。

汎用プロセッサの動向としては、Hyper Threading機能を搭載したXeonのデュアルプロセッサ機がすでに販

- 6 -

キテクチャに委託することとなる。OSは動的な資源管理と保護に責任を持ち、今後は、OSと一体化した言語処理系、コンパイラ支援を考えていく必要がある。

さらに、アーキテクチャに最も近い立場のOSとしては、複数のスレッドの制御をアーキテクチャサイドで行うべきか、OSの役割として扱うかの切り分けを今後検討していく必要があり、そこでさまざまなトレードオフを解決していくことが重要となるであろう。

## ■コンパイラからみたSMTアーキテクチャ

SMTアーキテクチャの性能を握る最も重要な技術がコンパイラ技術であるというのは共通の認識である。現在、ほとんどのOSで何らかのスレッドライブラリが使えるようになり、またJavaのようにThreadが言語仕様に取り込まれた環境が普及したことによって、SMTアーキテクチャの本領が発揮され、その真価が問われるようになってきている。しかしながら、以下に述べるように、現在までに培われてきた自動並列化技術はSMTアーキテクチャにそのまま適用できるわけではない。

自動並列化（特にCやFortranなどの逐次言語の並列化）の技術は以下の3つに分類される<sup>☆10</sup>。

1. 細粒度並列化：VLIWやスーパースカラプロセッサを対象にした、命令スケジューリングの延長としてのコンパイル技術。一般性が高いが、性能の向上には特にコンパイラが重要な役割を果たす。
2. 中粒度並列化：ループ単位の並列化。配列要素など比較的規則性があり、アクセスするデータアドレスに対する密な参照が前提で、大規模数値演算を対象としている。
3. 粗粒度並列化：タスクやプロセス単位、サブルーチン単位の並列化で、プログラミング言語によるサポートが重要である。

スレッドは、元々は上記の粗粒度並列化に適した処理単位であったが、プログラム中のループをサブルーチンに分割して、新たなスレッドとして定義することで、自動並列化コンパイラの中粒度並列化技術の蓄積が利用できる。また、細かく分割されたスレッド内の命令列に対して細粒度の並列化を施すことも可能である。これらの手法はSymmetric Multi-Processor (SMP) において技術の蓄積がなされており、SMTアーキテクチャ

にも適用可能であるが、以下のような点に注意する必要がある。

一般にコンパイラが実装するアーキテクチャにおいて重要視するのは、スレッド生成や同期オーバーヘッドが十分小さいことと、静的な実行時間の予測が可能であることである。

SMPと比較して、SMTはリソース競合が複雑に起きるので実行時間の予測はより難しい。しかし、スレッドの生成や同期オーバーヘッドは小さいので、その利点を活かした手法が考えられるであろう。

また、コンパイラの得意分野である中粒度最適化ではキャッシュの効果があまり期待できないため、主記憶バンド幅が性能のボトルネックになりやすく、特にメモリアーキテクチャの重要性が高くなると考えられる。SMTアーキテクチャとコンパイラの協調のためには、キャッシュ制御命令や投機ロード命令のような、メモリバンド幅の不利を補うような命令は必須であろう。

レジスタ数に関しては、コンパイラとしては多ければメモリアクセスの頻度が削減でき、処理性能は向上する。しかしながら、レジスタファイルのサイズやアクセスするポート数を増やすことはクロックスピードの低下を招き、またOSの節で述べたように、コンテキストの切り替え時のオーバーヘッドが問題となるため、レジスタの数に関しては慎重に検討しなければならない。レジスタの数はアプリケーションにより異なるが、コンパイラ開発者の実際の経験からは16本～32本は必要であると考えられている。

従来コンパイラの研究・開発はアーキテクチャとの結び付きが強く、OSとの連携に関してはやや手薄であった感がある。しかしながら、SMTアーキテクチャに限らず、今後の新しいプロセッサ・アーキテクチャの研究においては、OSとコンパイラの協力体制を確立することが重要であろう。

## ■プログラミング言語からみたSMTアーキテクチャ

今後のSMTアーキテクチャの性能が発揮できるかは、前述の通り、OSやコンパイラが重要な役割を果たすが、それに加えてアプリケーションをマルチスレッド化できるかどうかにかかっており、その作業には膨大

<sup>☆10</sup> 並列性の粒度 (Grain of Parallelism)：複数の処理を同時に実行させることで全体の処理性能の向上を目指したものが並列処理であるが、その処理単位は大きく、粗粒度 (coarse grain, プロセスや手続きレベル)、中粒度 (Medium grain, 主にループなど)、細粒度 (Fine grain, プロセッサの発行する命令レベル) の3つに分類される。この中で、粗粒度と中粒度の中間に位置するスレッドレベル並列性 (Thread Level Parallelism: TLP) と、細粒度の命令レベル並列性 (Instruction Level Parallelism: ILP) が、今後の並列処理において重要な鍵となる。

な時間を要するといわれている。そのためには、プログラミング言語からのサポートも1つの鍵となると考えられる。

従来、プログラミング言語はマシンの性能を引き出すという観点よりも、ユーザのプログラミング支援の方に重点を置いていた。いくらマシンの性能を指向した言語であっても、それを有効に活用できるための言語体系、すなわちプログラミングのしやすさや移植性を考慮したものでなければ、一般ユーザには受け入れられず、SMTアーキテクチャの持つ能力を引き出すこともできなくなる。

また、ユーザがそのプログラミング言語を用いて作成するさまざまなプログラムに対しても、コンパイラはコードを生成し、それを効率よくOSがスケジューリングし、アーキテクチャが高速に実行しなければならない。たとえば、実際のアーキテクチャが扱える個数以上の膨大な数のスレッドを生成した場合においても、システムは対応する必要がある、そのために、スレッドキューをバックアップするためのメモリ領域を確保するなどの工夫が求められる。

言語、コンパイラサイドでは、適度な並列度のハードウェアレベルのスレッド数となるように調節するという、従来のマルチプロセッサのための機能に加えて、ビジーウェイト<sup>☆11</sup>などにおいてリソースの消費を抑えるよう注意が必要となる。かつて、高級言語マシンの研究が盛んであった頃と同様に、SMTアーキテクチャにおいては、プログラミング言語はコンパイラ、OSとともにアーキテクチャを熟知した上での設計が求められるかもしれない。

## 今後の方向性

以上のように、SMTアーキテクチャについて、ハードウェア、ソフトウェアの両面から述べてきたが、今後はプロセッサ単体での性能を議論するのではなく、システム全体としての性能について考えていくべきである。

まず、システムアーキテクチャの視点からSMTアーキテクチャに対して求められるものとしては以下のことが考えられる。

- SMTにおいて重要となってくるメモリアーキテクチャ上の工夫。
- SMTの利点を生かすための高速な同期方式の工夫。

- OSとアーキテクチャ間におけるスレッド管理の役割分担。

コンパイラについては、今までのSMPなどにおける並列化技術の蓄積を活用し、SMT固有の問題に焦点をあてていくことで、さまざまな問題解決の糸口が見つかると思われる。

SMTアーキテクチャの今後の利用に関してであるが、Hyper Threadingは細かな多数のトランザクションを処理するサーバマシンをターゲットにしている。この場合、OSレベルで多数のスレッドが内在しており、アーキテクチャ上の工夫と、多少のコンパイラのサポートにより効率のよいシステムの構築が可能となるであろう。

また、現在、SMTアーキテクチャでは、複数のアプリケーションの同時実行が可能となり、単一プログラムの自動並列化による高速化の研究が進みつつある。今後さらに、複数アプリケーションの同時実行とともに、個々の単一プログラムの高速化を目指すならば、前述の通り、今後はアーキテクチャ単独ではなく、OS、コンパイラ、プログラミング言語処理系の研究者らと、高性能システムを実現していくために解決すべき問題を議論し、分担できる部分を明確にして、協調して研究を進める体制が大切であろう。

**謝辞** ソフトウェアに関しまして貴重な意見をくださった、早稲田大学理工学部電気電子情報工学科笠原博徳教授、京都大学大学院情報学研究科八杉昌宏講師、東京工業大学大学院総合理工学研究科合田憲人講師に感謝いたします。また、本稿を作成するにあたり、細部にわたり精査くださり、ご助言をいただいたNECシステムデバイス・基礎研究本部の西直樹氏に感謝いたします。

## 参考文献

- 1) <http://www.cs.wisc.edu/~arch/www/groups.html>
- 2) <http://www.cs.washington.edu/research/smt/>
- 3) Tullsen, D. M., Eggers, S. J. and Levy, H. M.: Simultaneous Multithreading: Maximizing On-chip Parallelism, Proc. of Int'l Symp. on Computer Architecture (ISCA '95), pp.392-403 (1995).
- 4) 佐藤, 中村, 有田: 大規模スーパースカラプロセッサ向け命令発行機構, 信学技報ICD2000-144, pp.107-112 (2000).
- 5) <http://www.csr.d.uiuc.edu/acoral/>
- 6) 河原, Yankelevsky, M., 中條, Polychronopoulos, C.: シングルチップマルチスレッドプロセッサα-Coralのアーキテクチャ, 並列処理シンポジウムJSPP2001論文集, pp.39-46 (2001).
- 7) [http://developer.intel.com/technology/itj/2002/volume06issue01/art01\\_hyper/p01\\_abstract.htm](http://developer.intel.com/technology/itj/2002/volume06issue01/art01_hyper/p01_abstract.htm)

(平成13年12月21日受付)

<sup>☆11</sup> ビジーウェイト (Busy wait) : イベントが発生するまでループを繰り返す処理で、CPU内のパイプラインはもとより、ALU、レジスタなどの重要なリソースを食いつぶす。SMTの場合、このリソース消費は、他のスレッドの実行に悪影響を及ぼす要因となる恐れがある。