高性能プロセッサモデルにおけるアーキテクチャとOSの協調†

4D−03

河原 章二　中條 拓伯　並木美太郎
東京農工大学工学部情報コミュニケーション工学科

1 はじめに

近年のVLSI技術は，1つのチップに数千万というオーダのトランジスタの実装を可能にし，それに合わせてプロセッサの性能も向上してきた．特に現在主流のスーパスカラプロセッサアーキテクチャは，命令ウィンドを増やし，またチップ上のキャッシュを拡大することで性能を上げてきた．しかし現在，その性能向上には陰りが見え始めている．その要因として単一の命令流（スレッド）に，現行のスーパスカラプロセッサが抽出できる命令レベル並列性（ILP）がさほど存在しないというのが挙げられる．

このような中，Simultaneous Multithreading (SMT) [2] アーキテクチャが注目されている．SMTは1つのチップに複数のスレッドを保持し，それらを同時に実行することで性能向上を得る．

SMTの提案は現在までにいくつかあるが，一方でSMTプロセッサを考慮したオペレーティングシステム（OS）は未だ提案されていない．本論文では，現状のプロセッサモデルのまとめを行なうとともに，SMTプロセッサ上において，どのようなOSのモデルが望ましいか，またどのような課題があるかを議論する．それとともに，OSの実装を目標としたSMTプロセッサのモデルについても議論する．

2 現状のプロセッサモデル

現在のプロセッサは，パイプラインプロセッサをベースにしてそこからいくつかのプロセッサモデルに派生している．まず，ベースとなるパイプラインプロセッサの簡単なモデルを図1に示す．

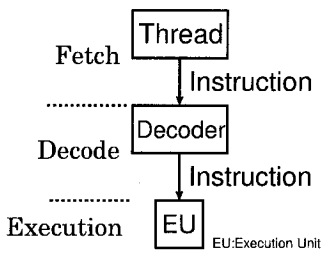

図1: パイプラインプロセッサのモデル

ここでスレッドとは1つの命令流を指す．図中の左の単語，Fetch，Decode，Executionはパイプラインステージを示している．また，矢印は命令の流れを示している．そしてEUはExecution Unitの略であり，ALUなどの実行ユニットを意味する．

パイプラインプロセッサにおいて，命令は1つのスレッドから1命令フェッチされ，その命令はデコードユニットを介して実行ユニットに送られる．これがベースとなるパイプラインプロセッサの基本的なモデルである．

2.1 スーパスカラプロセッサ

スーパスカラプロセッサとは，1つのスレッドから複数の命令をフェッチし，もし同時実行できる命令（依存関係が無い）があれば，1サイクル中にそれらを実行するプロセッサのことである．図2に簡単なモデルを示す．

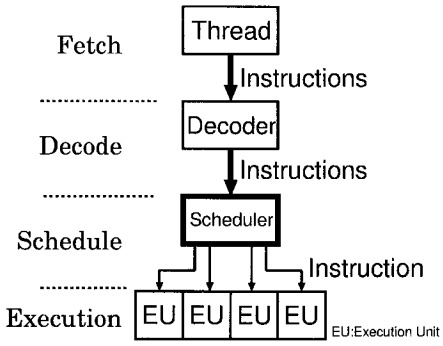

図2: スーパスカラプロセッサのモデル

同時実行（並列実行）できる命令はハードウェアが抽出しそれらを実行する．並列実行可能な命令を抽出，スケジューリングするのが図中のスケジューラの役目である．このために，新たにスケジューリングのためのパイプラインサイクルが追加される．現在，一般的に最も普及しているプロセッサモデルである．

2.2 VLIWプロセッサ

VLIW（Very Long Instruction Word）プロセッサとは，コンパイラ（ユーザ）が並列実行可能な命令をあらかじめ指定しておき，プロセッサは既にコンパイラによってスケジューリングされた命令を並列に実行する．図3にVLIWプロセッサのモデルを示す．

スーパスカラプロセッサは動的，つまりハードウェアが並列実行可能な命令を抽出して実行するのに対し，VLIWは静的にスケジューリングされた命令を実行する．よって，スーパスカラプロセッサのようなスケジューリングのためのハードウェアは必要ない．このため，スーパスカラプロセッサと比べてハードウェアが複雑にならずにすむ．現在，VLIWはDSPなどの，静的にスケジューリングしやすい定型処理をターゲットにしたプロセッサに採用されている．

---

†Cooperation between Operating System and Architecture Based on High Performance Processor Model

Shoji Kawahara (shoji@nj.cs.tuat.ac.jp)
Hironori Nakajo (nakajo@cc.tuat.ac.jp)
Mitaro Namiki (namiki@cc.tuat.ac.jp)
Department of Computer, Information and Communication Sciences , Tokyo University of Agriculture and Technology

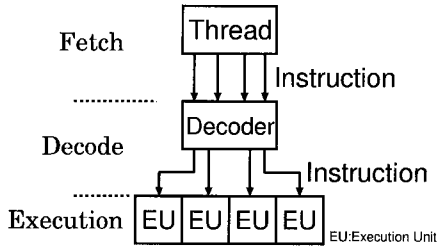

図 3: VLIW プロセッサのモデル

2.3 オンチップマルチプロセッサ

オンチップマルチプロセッサ（CMP）は，1つのチップに複数のプロセッシングエレメント（PE）を実装したプロセッサモデルである．図4に CMP のモデルを示す．

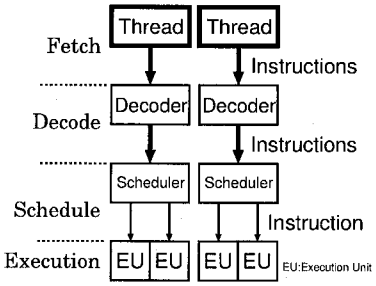

図 4: オンチップマルチプロセッサのモデル

CMP が今まで挙げてきたプロセッサモデルと異なるところは，複数のスレッドを1つのプロセッサで扱える点にある．このようなプロセッサアーキテクチャを，オンチップマルチスレッドアーキテクチャという．図4では各 PE はスーパスカラプロセッサになっているが，これは普通のパイプラインプロセッサでも，または VLIW プロセッサでも構わない．また，図では PE は2つだが，更に4つや8つであっても構わない．重要なことは複数の PE が1つのチップに実装されている点にある．複数のスレッドから命令をフェッチし，それらを各 PE に流すことにより複数スレッド実行を可能にしている．これは従来のマルチプロセッサを1つのチップで実現したものといえる．これにより，従来のシングルスレッドをターゲットにしたプロセッサモデルに比べ，より高い性能向上が得られる．

2.4 SMT プロセッサ

SMT プロセッサは複数のスレッドからフェッチしてきた命令を1つのパイプラインに流し，同時に実行するプロセッサモデルである．図5に SMT プロセッサのモデルを示す．

CMP，SMT プロセッサはともにオンチップマルチスレッドアーキテクチャであるが，両者の違いは CMP はハードウェアが PE 単位で区切られているのに対し，SMT は1つのハードウェアリソースを複数のスレッドで共有している点にある．このため，SMT プロセッサは CMP と比べてハードウェアリソースを無駄なく活用できる．

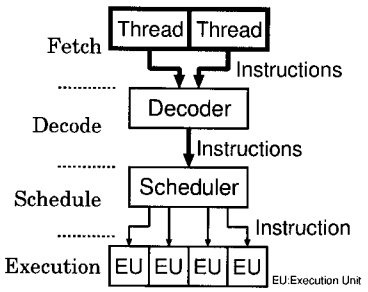

図 5: SMT プロセッサのモデル

2.5 プロセッサの命令実行状態

上述した各プロセッサモデルの命令実行状態のまとめを図6に示す．

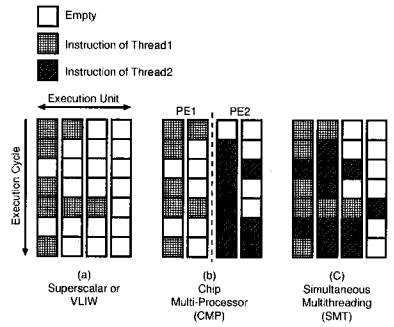

図 6: 各プロセッサモデルの命令実行状態

図の四角1つ1つが実行ユニットを意味し，横が実行ユニットの総数，縦が実行サイクルの流れを意味する．例えば（a）の1番上の段が意味するものは，このプロセッサは1サイクル中に最大4つの命令を同時に実行でき，1サイクル目でスレッド1の2つの命令がプロセッサの実行ユニット（Execution Unit）によって実行される，ということである．全サイクルを通した実行命令数が，即ちそのプロセッサの性能の指標といえる．

図6の（a）はスーパスカラプロセッサと VLIW プロセッサの命令実行状態である．両プロセッサは，全サイクルを通して1つのスレッドからフェッチしてきた命令を実行する．よって，1サイクル中に1命令しか実行できない時もあれば，キャッシュミスなどのプロセッサのストールにより，全く命令を実行できないサイクルも存在する．

（b）は CMP の実行状態を示しており，各 PE がそれぞれ別のスレッドを実行することにより，(a) よりも実行ユニットの稼働率が改善されるのが分かる．

そして（c）は SMT の実行状態を表しており，CMP

と違い，1 つのプロセッサ（CMP における PE）を複数のスレッドが共有しているのが分かる．複数のスレッドからフェッチしてきた命令群を一緒にスケジューリングすることにより，CMP と比べてより効率的にハードウェアリソースを使用できる．

3　SMT プロセッサ

ここでは SMT プロセッサの概要について述べる．前節では SMT プロセッサの簡単なモデルについて述べたが，ここでは SMT プロセッサの具体的な形態を示す．

図 7 に SMT プロセッサ内部の概略を示す．

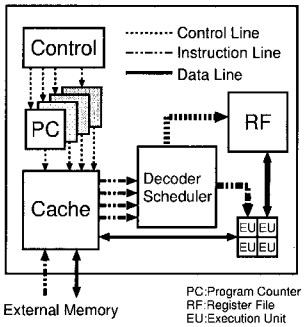

図 7: SMT プロセッサの概要図

SMT プロセッサは，チップ内に複数のプログラムカウンタ（PC）を保持し，それらを制御することで複数スレッドの実行を可能にしている．その他のハードウェアリソース，例えば実行ユニットは全てのスレッドが共有する．このようにスレッド制御ユニットなどのわずかなコストで，SMT プロセッサは複数スレッド実行の性能向上のみならず，ハードウェアリソースを有効に利用することが可能になる．しかし，SMT プロセッサは従来のスーパスカラプロセッサの機能を拡張したものであり，スーパスカラプロセッサが本来持っている，ハードウェアの複雑さの問題 [1] をより増大させる恐れがある．

現在，Intel が HyperThreading という SMT アーキテクチャを用いたプロセッサを商用化しつつある [3]．このプロセッサは，論理的に 2 つ PE が結合した SMP (Symmetric Multi-Processor) として動作する．これは提案初期の SMT[2] のアーキテクチャに近く，比較的簡単なハードウェアロジックの付加により，性能向上が得られる．

研究レベルでは，1 つのプログラムを複数のスレッドに並列化させ，それらを SMT プロセッサで実行するという手法が模索されている．この複数スレッドの実行はプロセッサに新たなパラダイムを与える．単純な並列化の場合，サブルーチンコールやループのイタレーションを全てスレッドに変換し，並列実行させるという手法が挙げられる [4]．また，分岐命令が現れた時に Taken，Not Taken の両方のパスを実行することも可能である [5]．さらに，単なるプログラムの並列化だけでなく，通常のプログラム実行と同時に，分岐予測の改善，プリフェッチ，キャッシュ制御などの副次的な操作を行なうことも提案されている [6, 7]．

4　オンチップマルチ SMT

前述の通り現状の SMT プロセッサは，商用レベルでは SMP に見立てたものに，研究レベルではシングルプログラムの高速化を目指したものと位置付けられている．商用レベルにおける SMP と見立てた SMT の使用は，従来のオペレーティングシステム（OS）との親和性がよいという利点がある．例えば Microsoft 社の WindowsNT や，フリーで配布されている Linux などは SMP に対応しており，ほとんどプログラムを改変することなく SMT プロセッサに移行できる．

しかし SMP と見なすならば，ハードウェアが複雑化する SMT としなくとも，CMP を用いて実現することが可能である．一方で，現在研究レベルである単体のプログラムの並列実行の場合，CMP はハードウェアリソースが各 PE に分断され，複数スレッドに並列化できないプログラムの時には期待したほどの性能が引き出せない場合がある．我々はこれらのことから，OS との親和性を考慮したシステム全体のスループットの向上，ハードウェアの複雑化の回避，単一プログラムの実行速度の向上，これらを目指したアーキテクチャ，オンチップマルチ SMT というアーキテクチャについても今後研究を進める．

オンチップマルチ SMT は，SMT プロセッサを PE とし，その PE を複数個，1 つのチップ上で実現することにより，SMT プロセッサの SMP として動作する．複数のプロセスは OS によってそれぞれの PE に割り振られ，それらは OS によってスケジューリング，管理される．また各 PE に割り振られたプロセスは，コンパイラ（ユーザ）によって複数スレッドに分割され，その PE 内で並列実行される．またプロセス間の通信，つまり PE 間の通信は 1 つのチップ上にあるという利点をいかし，チップ上の共有メモリないし共有レジスタによって高速に行なわれる．

5　SMT を考慮した OS の資源管理モデル

ここでは SMT を考慮した時の OS のモデルについて議論する．

従来，OS ではプロセッサの仮想化モデルとして，Mach 以降，プロセスとはマルチプログラミングの環境下で保護を行う実行単位，スレッドとは資源を共有しあう実行単位，という 2 種類のモデルが採用されてきた．OS から見れば，CMP も SMT も基本的には OS の論理スレッドとして仮想化できる．

従来の資源管理モデルの延長でとらえると，

1. **複数の実スレッドを実行する 1 つの SMT プロセッサを，1 つのプロセスとして仮想化，プロセス全体をスイッチする方式**
2. **実スレッドを論理スレッドとして仮想化する方式**

の 2 つが考えられる．

前者は，アドレス空間を共有しあう従来のスレッドとプロセスの関係を，そのまま対応つける考え方である．1 つのプロセス内で生成されるスレッドは実スレッドに対応し，実スレッドの生成はコンパイラが並列化を行う．この SMT をマルチプロセッサにしたアーキテクチャでは，さらに，この複数のプロセッサを用いて，実スレッドの個数を増やすことができる．これにより，資源共有の観点からは，高性能が期待できる．

しかし，プロセス内の論理スレッドを実スレッド以上に割り付けようとすると，何らかの工夫が必要となる．ループのような均質構造を有する反復処理ならば，コンパイラでの最適化が期待できるが，サーバ処理のような非均質構造のマルチスレッド処理では，プログラマが明示的にforkなどのスレッド生成命令を発行する可能性があり，個数を限定することはできない．このような理由から，後者のような論理スレッドを導入する必要が生じる．

従来のOSのスレッド管理は，コンテキストのsave/restoreは必須であるため，細粒度並列性を生かすための従来のスレッドよりも，さらに軽量なコンテキストの導入が不可欠である．オンチップマルチSMTでは，個々のPE間でのスレッド割付が性能を左右する．

上記のことから，SMTおよびオンチップマルチSMTにおけるOSの目標は，数10命令から1000命令程度の細粒度スレッドに向けた

1. **本質的課題として，SMT向けのプログラミングモデル**
2. **技術的課題として軽量な論理スレッドの制御方式特に，関数呼出し程度の数命令で論理スレッドをスイッチする方式**
3. **SMTをオンチップで複数結合させるたときの資源の統合的な管理**

を課題としてとらえる必要がある．

6 OSを考慮したSMTプロセッサ

ここでは，OSの実装を目標としたSMTプロセッサのモデルを議論する．

基本的に多数のスレッドを実行するSMTプロセッサは，従来のプロセッサと比較して，より外部メモリのバンド幅が問題になってくる．これに関連して，従来のキャッシュをそのままSMTに当てはめるのも問題になっている[8]．また前述の通り，OSを用いてコンテキスト切り替えを行なう場合，コンテキストのsave/restoreが必須であり，メモリバンド幅は更に重大な問題になってくる．そして，コンテキスト切り替えはレジスタの退避のため，システム全体で実行サイクルを大きく消費する作業の1つである．このため，例えばこのコンテキスト切り替えのためのコンテキスト切り替え専用のキャッシュ（C-cache）をプロセッサ内部に実現するなどして，コンテキスト切り替えによるメモリバンド幅への影響を取り除く必要がある．このC-cacheのため，プロセッサはOSがこのキャッシュを直接アクセスできるように新たな命令を用意するのが望ましい．また，このC-cacheのトレードオフとして，従来の1次キャッシュ，2次キャッシュの削減が考えられる．

そしてSMTプロセッサ上において，本来OSが管理するプロセスを管理する機構だけでなく，コンパイラやユーザが生成するスレッドをOSが管理できるようにするためのアーキテクチャによるサポートが必要である．前述の通り，OSはシステム全体の資源を管理する役割があり，当然プロセッサの資源も管理する．しかしコンパイラやユーザがOSのシステムコールを介することなく，SMTプロセッサのスレッド生成命令を実行できるアーキテクチャの場合，資源の管理者たるOSにそのスレッド生成を何らかの形で通知する必要がある．これを回避する方法としては，スレッド生成を行なう時は必ずOSを介するというモデルにすればよいが，本来SMTプロセッサに期待される軽量なスレッド生成・消滅の実現が困難になる．このため，軽量なスレッド生成・消滅，そしてOSの資源管理を両立させる場合はプロセッサ側の何らかのサポートが必要になってくる．

この点で，オンチップマルチSMTを考えた場合，SMTであるPE内部はコンパイラまたはユーザのスレッド生成空間，そしてチップ全体はOSのプロセス管理空間という切り分けができる．しかし，PEの内部に生成できるスレッド数以上をコンパイラやユーザが生成しようとした場合，コンパイラやユーザはOSまたはアーキテクチャのサポートが必要となり，これはアーキテクチャ，OS，コンパイラ（ユーザ），全てにまたがる課題の1つである．

このようにSMTプロセッサにおいてOSを考慮した場合，メモリバンド幅などのハードウェアとしての課題，OSの管理機構をサポートを検討する必要がある．

7 まとめ

本論文では，現在実現されているいくつかのプロセッサモデルの説明を行なった．また最近注目されているSMTアーキテクチャの内部モデルを示し，SMTプロセッサとOSを協調させるための課題を述べた．今後は，ここで述べた課題を検討し，SMTアーキテクチャとOSを密接に協調させた，より強固なシステムの構築を目指す．さらにはオンチップマルチSMTの検討も今後行なう．

参考文献

[1] S. Palacharla, N.P. Jouppi, and J.E. Smith: Complexity-effective superscalar processors, 27th Int'l Symp. on Computer Architecture, 1997

[2] D. M. Tullsen, S. J. Eggers, and H. M. Levy: Simultaneous multithreading: Maximizing on-chip parallelism, in Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp.392-403, 1995

[3] http://www.intel.com/

[4] 河原 章二, M. Yankelevsky, 中條 拓伯, C. Polychronopoulos: オン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(JSPP'2001), pp.39-46, 2001

[5] S. Wallace, B. Calder, and D. M. Tullsen: Threaded Multiple Path Execution, Proc. of Int'l Symp. on Computer Architecture(ISCA98), pp.238-249, 1998

[6] R. S. Chappell, J. Stark, S. P. Kim, S. K. Reinhardt, and Y. N. Patt: Simultaneous Subordinate Microthreading (SSMT), Proc. of Int'l Symp. on Computer Architecture(ISCA99), pp.186-195, 1999

[7] 佐藤 寿倫, 有田 五次郎: KIT COSMOS プロセッサ：背景と着想 IEICE Technical Report CPSY99-115, pp.69-76, 2000

[8] S. Hily, and A. Seznec: Standard Memory Hierarchy Does Not Fit Simultaneous Multithreading, In Workshop on MultiThreaded Execution, Architecture and Compilation, Colorad State Univ. Technical Report CS-98-102, 1998