計算機アーキテクチャ 124－3
（1997. 6. 27）

大規模データパスプロセッサの構想

中村友洋† 吉瀬謙二† 辻 秀典†
安島雄一郎† 田中英彦†

本稿では、今後のデバイス技術の進歩を仮定した上で、新しいマイクロプロセッサ・アーキテクチャとして大規模データパスプロセッサの構想を述べる。従来のプロセッサでは利用できる並列性の限界などにより、その性能に限界が見え始めている。また、半導体技術の進歩により1チップ内に大規模なデータパスを構築して実行する方式（大規模データパス方式）が現実的になってきた。本稿では、このような背景のもと、大規模データパスプロセッサの開発を目指して行なっている初期的検討と今後必要となる技術について述べる。

Feasibility Study of A Very Large Data Path Processor Architecture

TOMOHIRO NAKAMURA,† KENJI KISE,† HIDENORI TSUJI,†
YUICHIRO AJIMA† and HIDEHIKO TANAKA†

This paper presents the plan of Very Large Data Path (VLDP) Processor on the assumption that VLSI architecture will grow at the same rate. Because today Microprocessors can use only local parallelism and so on, thier performance will reach the ceiling by and by. Because of the continuous improvement of semiconductor technology, it will become possible in near future to construct a large data path in one chip. Based on these, preliminary studies of VLDP architecture are briefly discussed.

1. はじめに

マイクロプロセッサは1971年の4004[5]誕生以来これまで四半世紀の間、デバイス技術の進歩とアーキテクチャの進歩という両輪に支えられて発展を続けてきた。デバイス技術に関してはMooreの法則でいわれるように3年間でチップのトランジスタ数が4倍になるというペースがほぼこれまで続いている。それに伴ってオンチップ・キャッシュの搭載、命令レベル並列性を利用するスーパースカラ方式・VLIW方式などのチップが登場してきた。

近年のプロセッサ・アーキテクチャにおいては、スーパースカラ方式が主流となり、ローカルな命令レベル並列性をより利用するために、out-of-order実行、投機的実行、データフォワーディングなどの機構が追加されている。さらに2次キャッシュのオンチップ化なども見られる。しかし、数年前からスーパースカラ方式の限界が様々指摘されている[10]。

また21世紀に向けてより長期的な視点から、大規模なVLSIの利用方法に関する議論が様々行なわれている[16]。オンチップ・マルチ・プロセッサや、DRAM・ロジック混載などによるメモリとCPUの1チップ化もしくはシステムオンチップと呼ばれるアプローチなどがその代表例である。これらの技術により既存の並列計算機やコンピュータ・システムの機能に相当するものが少数のチップにより構成され、例えば並列計算機における通信オーバーヘッドや、PC・ワークステーションなどにおけるメモリ・ウォール問題などが改善されることで高性能化が期待されている。しかしながら、これらはCPUとしては従来のものを使うものが多い。よってCPUのもつ性能自体を改善するするというのではなく、チップを単位として見たときの性能を改善するアプローチであるといえる。

本稿では、まず今後おおよそ10年後のデバイス技術を仮定し、従来のプロセッサが抱える問題点を簡単に考察した上で、CPU自体の性能向上を目指したプロセッサの構想について述べる。第2節ではデバイス技術、既存プロセッサの抱える問題を述べ、アーキテクチャの将来像について現在研究が進行中のものをまとめる。第3節では大規模データパス方式について示し、第4節で初期的検討および今後必要とされる技術について述べる。

† 東京大学大学院工学系研究科
Graduate school of Engineering, The University of Tokyo

2. 背　　景

2.1 デバイス技術

ハードウェアの規模に関して3年で4倍になるというMooreの法則はよく知られている。現在までのところDRAMにおいてもlogicにおいてもこの法則はほぼ成り立ってきている。プロセス技術は基本的には歩留まり技術であり、チップ作成に関するコスト削減の技術である。Mooreの法則は技術予測であるため物理的制約を考慮に入れてない。この先、物理的制約によりデバイス技術の飽和が起こる可能性もある。

米国半導体工業会調査のデバイス技術に関するロードマップ[8]によれば、今後想定されるデバイス技術は表1の通りである。

表1　デバイス技術ロードマップ
Table 1　Semiconductor technology roadmap

年	1998	2001	2004	2007
最小加工寸法 (μm)	0.25	0.18	0.13	0.10
MPU Tr.数 (/chip)	28M	64M	150M	350M
チップ内周波数 (MHz)	450	600	800	1000
DRAM bit数	256M	1G	4G	16G

表1によれば、およそ10年後には数億トランジスタを利用できることになる。現在の主なプロセッサは数百万〜一千数百万トランジスタであるので、非常に大規模なデバイスを利用できることが分かる。このようなデバイスをどのように利用するかが今後のプロセッサを考える上で重要である。

一方DRAMとlogicそれぞれを見てみると、そこにはメモリウォールと呼ばれる問題が存在する。図1[6]のようにDRAMとlogicのプロセスではDRAMの速度向上率がlogicに比べて低いため、今後一層その差が広がることも考慮しなければならない。最近のプロセッサにおいても図2[14]のように大量のキャッシュメモリを載せてメモリウォール緩和を行なっている。

2.2 既存プロセッサの問題

本節では既存プロセッサの代表としてスーパスカラプロセッサを考える。スーパースカラプロセッサは命令レベル並列性を利用して性能を得る構造である。しかし、様々な問題によりさらなる性能向上に限界が見え始めている。ここでは制御依存関係による制約とメモリウォール問題の2点についてのみ取り上げる。

制御依存関係による制約　プログラム中の分岐命令による制御依存によって命令レベルの並列性抽出に制限が加わる[9]。これを緩和するためこれまでに様々な分岐予測機構[13][3]や条件付き実行[4]などが研究されてきた。しかし、それぞれの性能には限界が見えてきており、より積極的な投機実行機構を設けたり、より大きなレベルでの並列性の利用を考えなければならない。

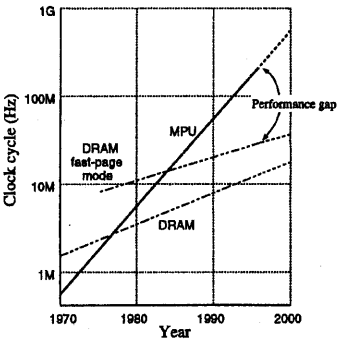

図1　MPUおよびメモリの動作速度の変化
Fig. 1　Historical Trend of MPU and DRAM Clock Cycles

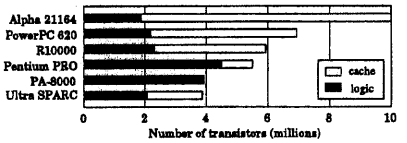

図2　全トランジスタに対するlogicの占める割合
Fig. 2　Allocation of transistors in microprocessors

メモリウォール問題　図1に示したようにCPUとメモリの速度差のために十分なデータがCPUに供給できない状況が生じてきた[12]。並列性を利用するほどこの影響は大きく現れる。プリフェッチ技術[2]の利用等によりこれを緩和することが行なわれている。

2.3 アーキテクチャの将来像

今後のプロセッサ・アーキテクチャを考える上で重要となるのは、大規模なデバイスをいかに利用するかという点である。ここでは表1の2007年におけるデバイス技術を仮定し、アーキテクチャの将来像として次の3つのアプローチについて述べる。

チップ内並列化　現在のスーパースカラ方式やその延長のものが含まれる。スーパースカラよりも広い範囲の並列性を使うものとしてはタスクと呼ばれる単位の並列性を利用するマルチスカラプロセッサ[11]、複数のスレッドを並列に実行するマルチスレッドアーキテクチャ[20][18][7]、従来のマルチプロセッシングを1チップで行なうマルチプロセッサ・オンチップ[1]などが挙げられる。これらは、これまでの並列化技術の延長と考えられるが、マルチスレッドアーキテクチャやマルチプロセッサ・オンチップなどでは、通常は並列に実行するようにかかれたアプリケーションと専用のコンパイラが必要であり、これらが充実しているとはいえないのが現状である。

システム・オンチップ　ここではCPUとメモリが1チップに融合したものをシステム・オンチップの代表とする。先に述べたように最近では、CPU・メモ

リ間がボトルネックとなることが多く、システム・オンチップは、従来のプロセッサにおけるピンネックやバンド幅、レイテンシの問題を解決することを目指して研究されている[15]。PPRAM[17] は大容量の汎用メモリ、複数の汎用プロセッサ、外部インタフェースを1チップ化したもので、チップ内並列化もおこなったシステム・オンチップであるといえる。

大規模 CPU　ここでは大規模 CPU として、多数の演算器を持つプロセッサを考えることにする。例えば、多数の演算器を相互に接続し、そこにデータパスを構築する実行方式[19]があげられる。このようなアプローチは CPU 自体の性能向上を目指すものである。よってデバイス規模が許せば、大規模 CPU をマルチプロセッサ・オンチップやシステム・オンチップの単位 CPU とすることも考えられる。

3. 大規模データパス方式

本研究では、CPU 自体の性能向上を目指す大規模 CPU の方向で、次々世代のマイクロプロセッサを開発していくことにした。大規模 CPU としては、多数の演算器を持つプロセッサを考える。このようなプロセッサを従来のスーパースカラ構造の大規模化として考えるのは困難である。すでに数個の ALU をもったスーパースカラ・プロセッサにおいても性能向上限界が見えており、より多くの ALU を用意しても有効に利用できるとはいえない。そこで、プログラムの実行方式から考え直す必要がある。プログラムの処理の本質はデータフローであり、これを効率的に実行することを考えなければならない。現在のスーパースカラ等では、プログラムのコントロールフローを中心にして実行を行なっているため制御依存関係による制約等で性能向上は限界にきていることは先に述べた通りである。そこで、多数の演算器を相互に接続し、そこにデータパスを構築する実行方式を考える。このような多数の演算器によってデータパスを構築可能なプロセッサ (大規模データパス–VLDP–プロセッサ) を開発していく。

VLDP アーキテクチャでは、可能な限りデータフローをハードウェアで実行することを考える。データパスを多数の演算器の相互接続によって構築し、相互接続網でデータフロー・グラフを実現することで、オーバーヘッドの少ない処理ができる。このような多数の演算器の相互接続網を ALU-Net と呼ぶことにする。

図3は、複数の分岐命令をまたいで命令スケジューリングを行なった場合に抽出可能な平均命令レベル並列性である☆。図3で、all はすべてのパスを投機実行する場合、ideal は分岐予測が 100%成功して必要なパスのみを実行できた場合である。図3によれば、ideal 状態で7個の分岐命令をまたげば並列度が9程度になる。all 状態で並列度の向上が見られないのは

☆ 本稿の全データは SPEC92 を用いたシミュレーションによる

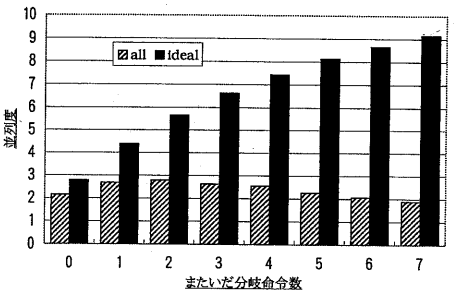

図3　分岐をまたいだスケジューリングと並列性の関係
Fig. 3　Parallel Gain with Speculation

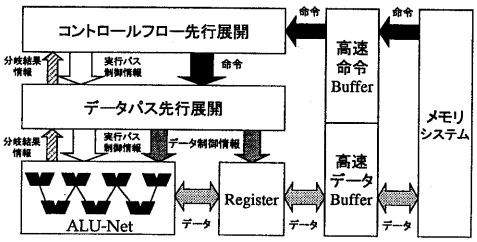

図4　VLDP アーキテクチャ・ブロック図
Fig. 4　VLDP Architecture Block Diagram

無駄な投機実行による影響である。よって ALU-Net を使って効率的に処理を行なうには、制御依存関係が解消された状態のデータパスを多数用意し、各パスごとにデータ依存関係に基づいてデータパスを作り、それを ALU-Net 上でまとめて処理を行なえばよい。そのためには、まず制御依存関係を解消し、多数のデータパスを用意するための機構。次にこれらのデータパスのデータ依存関係に基づいて ALU-Net 上にデータパスを構築するための機構と、それを実際に実行する機構が必要である。これらの機構を VLDP アーキテクチャではそれぞれ次のように呼ぶことにする。

(1)　コントロールフロー先行展開
(2)　データパス先行展開
(3)　実行（ALU-Net）

図4は VLDP アーキテクチャのブロック図である。コントロールフロー先行展開では、命令キャッシュから命令を読み込み、実行パス制御情報をつけてデータパス先行展開に命令を渡す。データパス先行展開では、実行パス制御情報と命令からデータ制御情報を作りだし、ALU-Net およびレジスタにおけるデータ転送制御を行なう。また実行パス制御情報によって ALU-Net 上にデータパスが構成される。分岐結果は ALU-Net から各制御部に返される。

3.1　コントロールフロー先行展開

コントロールフロー先行展開は VLDP プロセッサにおけるフェッチ機構であり、その動作は以下の2点を満たすものである。

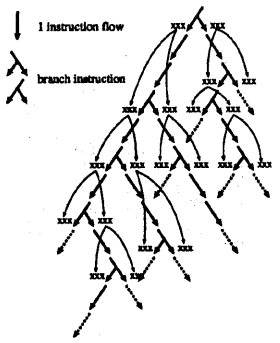

図5　コントロール・フロー先行展開のイメージ
Fig. 5　Image of Control Flow Pre-Construction

動的な分岐情報の利用　分岐予測のための確率情報をコントロールフロー・パス別に管理することによって分岐予測を効率的に行うものである。

資源が許す限りフェッチ動作を行う　すでに述べた効率的分岐制御に基づき多くの分岐を跨って資源が許す限り投機的に命令をフェッチするものである。多くの命令をフェッチすることよってコントロールフローの構築を行い制御依存関係を解消した命令列を抽出することが目的である。さらに、多くの分岐を越えた投機的フェッチにより複数のパスが構築されるため分岐予測がはずれた場合でもその影響が小さいと考えられる。

その動作は 図5 に示すように、命令をフェッチしつつコントロールフロー・パスの構築を行い、このコントロールフロー・パスに基づき情報の管理を行うものである。命令フェッチ時、分岐命令に到達した場合のその先の投機的フェッチは分岐予測の確率情報に基づいて行われる。結果として、投機的にフェッチされるパスとされないパスが存在する図5 に示すようなコントロールフローパスが構築される。このパスのうち命令実行の結果、実行パスが確定し不要となったパスが切り捨てられるとともに、分岐確率情報の更新が行われる。

具体的に 図5 において矢印は 1 命令であり、実線が実際にフェッチされた部分、点線が命令は存在するがその時点ではフェッチされていない部分である。さらに、xxx は分岐命令における分岐確率情報を意味し、細い線の矢印はこの分岐確率情報の依存関係を表している。図5 は、命令のフェッチ状況（コントロールフロー）を表すもので、データ依存関係（データパス）を表しているものではない。さらにフェッチは制御パスの構築に行うものであるため、フェッチに伴いデータ処理は行われない。

コントロールフロー先行展開において管理される情報は、コントロールフローパスの情報（パスの切捨てを行う場合に必要）と分岐確率情報である。これらを総合して環境情報と呼ぶことにする。

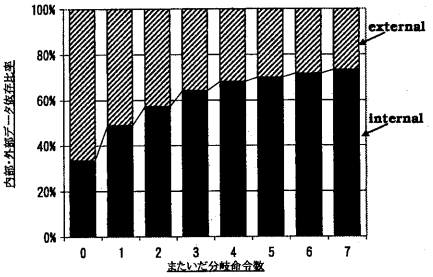

図6　ローカルなデータ・アクセス
Fig. 6　Local Data Access

3.2　データパス先行展開

VLDP では ALU-Net に命令を割り当て接続網上にデータパスを構築し、データを流すことで実行を行う。ここでは、実行すべき命令を解析し ALU-Net への割り当てを行うステージをデータパス先行展開と呼ぶ。

従来のレジスタオペランドによる演算はレジスタを介した集中型のデータ交換と見ることができる。しかし、レジスタアクセスに制御が集中することは、将来多数の演算器を利用する際にハードウェア的な障害となると考えられる。特に VLDP では、コントロールフロー先行展開により複数実行パスの命令がフェッチされる。ここで従来のプロセッサと同様にレジスタによるオペランド供給のみで実行を行うと仮定する。フェッチ命令数の増加や、全後続実行パスで解放されるまでリネームレジスタを保存しなければならないために起こるリネームレジスタのライフタイム伸長によって、必要レジスタ数が増加することが予想される。

そこで VLDP では、ローカルなアクセスを ALU-Net 上の接続として吸収し、データを直接演算器間で転送する。図6 は複数の分岐命令をまたいで命令フェッチを行なった場合に、それらの命令間に存在するデータ依存関係を調べたものである。internal とはデータ依存がフェッチされた命令間にあるもの、external とはデータ依存がフェッチされた命令以外の命令との間にあるものである。図6 によれば、分岐命令を多くまたぐ程、データ依存がフェッチされた命令間にある確率が高くなり、ALU-Net 上の接続で吸収できる確率が高くなる。このようなデータ転送を行なうことで、従来では演算前後に制御を行って解決しなければならなかったレジスタアクセスを、演算前に解決可能な ALU-Net の制御に置き換えることができ、制御に関するレイテンシを削減できる。ここでさらに演算器がデータを出力する際に次の演算器の発火を行うことにより制御の分散化を実現できるため、多数の演算器を持つ大規模 CPU の実装が可能になる。

一方、ALU-Net は加算器、乗算器、論理演算器、シフタ等基本的な回路を多く集め、相互に接続したものである。レイテンシによる実行速度の低下を防ぐため、接続網にはゲート数が少ないことが要求される。

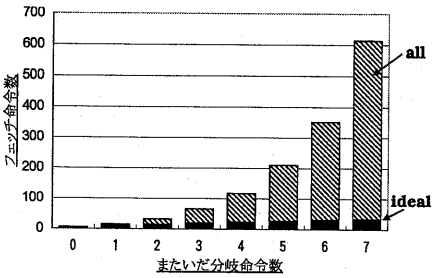

図7　分岐をまたいだ投機的命令フェッチ
Fig. 7　Speculative Instruction Fetch

4. 考　　察

4.1　コントロールフロー先行展開

分岐制御機構の検討（分岐予測の方針）　コントロールフロー先行展開はその中で用いる分岐予測機構によりその命令展開の効率が決定される。図7は複数の分岐命令をまたいで命令フェッチを行なった場合の命令フェッチ数である。allはすべてのパスを投機フェッチする場合、idealは分岐予測が100%成功して必要なパスのみをフェッチする場合である。命令展開の効率をあげるには、フェッチする命令を減らしてidealに近付けることが必要である。これは投機的フェッチ機構におけるコスト最小化に関する研究であるといえる。

実装方式の検討　コントロールフロー先行展開の実装方式における検討項目として、環境情報の保存と更新の方法があげられる。コントロールフローの展開時に必要とされる分岐確率等の情報およびパス管理を行うための情報等の管理。さらに、これらの情報は動的変化するものであるため、その更新および削除等の操作。実装においてはこれらの具体的な検討が必要である。コントロールフロー先行展開においては、多量の命令を投機的にフェッチすることを目的としているため、その機構の大規模化が考えられる。しかし、機構の大規模化は情報管理を複雑にするため、集中的に管理、処理すればその部分がクリティカルパスになり得る。逆に無理に分散的に管理させれば大規模化は容易となるが、そのためのオーバーヘッドが生じる可能性がある。よって、規模を考慮した管理方法の検討が必要である。さらにその規模は先に述べた分岐制御機構にも依存しているため、これは、情報管理方法と投機的フェッチを行う規模、さらには分岐制御機構の効率を考慮に入れた実装に関する研究であるといえる。

命令フェッチ機構の検討　コントロールフロー先行展開は、命令実行とは独立して連続的にフェッチを行う機構であるため、機構内の検討だけでなくフェッチ能力を向上させる必要性も生じる。メモリデバイスのアクセス速度向上はそれほど望めないため、機構の改善によりそれを行うしかない。その一つの方法として、連続転送を用いることによるフェッチ能力の改善が考えられる。例えば、ソフトウェア的に情報を付加することによってハードウェアがそれを利用するというものが考えられる。さらにはハードウェア機構との複合的利用、メモリ階層の見直しによるこの他のハードウェア的改善等多く考えられる。これは、ソフトウェア的サポートを考慮に入れたメモリシステムの改善に関する研究であるといえる。

4.2　データパス先行展開

4.2.1　低遅延の接続網

接続網の遅延を少なくするために単純な接続網しか使えない場合、演算器間の接続数が制限され演算器の利用効率が悪くなることが予想される。このため、プログラム中で出現頻度の高い典型的な演算器接続パターンを複数実装するなどの対策が必要になる。

4.2.2　依存関係解析とレジスタ

ALU-Net上の接続に割り当てるローカルなデータ依存関係を動的に検出するため、データ依存関係の解析を行う必要がある。基本的に従来のアウトオブオーダ実行のデータ依存解析技術が流用できるが、複数実行パスの命令を扱うための拡張が必要になる。リネームレジスタの扱いなどで異なる実行パスの命令同士が干渉しないように制御する必要がある。

ロードストア以外のデータフローは全てALU-Netに割り当てるのが理想であるが、一つの演算器の出力を多数の演算器が必要とする場合、一対多の複雑な接続・制御を行わなければならない。このため現実的には、低速だが演算器間でグローバルなデータ転送手段としてレジスタを使用せざるを得ない。VLDPにレジスタを用いる場合、以下のような特有の課題がある。

命令の発火機構　ALU-Netに新たに命令を割り当てる場合、ALU-Net上で発火待ちしている命令の出力を使用する時は必ずALU-Net上で接続することが理想である。しかし、接続網の関係で常にALU-Net上で接続できるとは限らない（2オペランドのうち片方しかALU-Netで接続できない実装など）。この場合、ALU-Netに割り当てた命令がレジスタを待つ場合の発火機構を設ける必要がある。発火機構としては、レジスタ管理側で制御する従来型の制御機構と、演算器間で制御を行う機構が考えられる。

リネームレジスタ制御機構　複数の実行パスの命令を扱うため、同じレジスタでも実行パス毎に別に管理しなくてはならない。これはリネームレジスタ機構を拡張し、同じレジスタでも異なる実行パスの場合は別のリネームレジスタを割り当てるようにすることで解決できる。

しかしこの場合、リネームレジスタの解放は単純に一つのレジスタへの代入命令が行われた場合ではない。同一のリネームレジスタが複数の後続実行パス

から参照される場合があるため、後続の実行パス全てでレジスタへの代入が行われるまでリネームレジスタを解放してはいけない。よって、このための監視機構も必要になる。

レジスタの分散実装　レジスタを複数組持ち、実行パスの分岐時にレジスタの組をデュプリケートするのが理想だが、デュプリケートの頻度が高いと実行速度に影響が出る可能性がある。異なる実行パスではデータ依存関係はないため、これを利用したレジスタの分散実装を検討する。

4.3 ALU-Net

ALU-Net はハードウェアコストが高いので、性能を落さない範囲でできるだけ小さくしたい。そのためには図7における all と ideal の差で表される無駄な命令を ALU-Net にできるだけマップしないようにすべきである。つまり、投機フェッチ／デコード／マップ／実行のいずれのステージまで大規模な先行展開を行なうかを検討する必要がある。

5. おわりに

本稿で検討した大規模データパス方式では、低レイテンシでデータ処理を行なう。ALU-Net を用い、CPU 自体の性能向上を目指す。本方式で重要な役割を担うのは、コントロールフロー先行展開とデータパス先行展開である。コントロールフロー先行展開では投機的に多数の命令をフェッチすることでデータ準備等のペナルティを隠蔽するための時間を作るとともに、フェッチした多数の命令が持つ制御依存関係を解消することで、後のステージの解析を容易にする。データパス先行展開はデータ依存解析をおこない、多数の命令を ALU-Net にマップする。今後は、大規模データパスプロセッサの詳細検討とその性能評価を行なう。

謝辞　本研究の遂行にあたり、日本学術振興会の特別研究員制度（「プログラム解析に基づく次世代マイクロプロセッサアーキテクチャの研究」）および文部省科学研究費（一般研究 (B) 課題番号 07458052「大規模データパスプロセッサの研究」）のご支援を頂きました。ここに感謝の意を表します。

参考文献

1) A.Nayfeh, B., Hammond, L., Olukotun, K.: Evaluation of Design Alternatives for Multiprocessor Microprocessor, *23rd ISCA*, pp. 67–77 (1996).
2) Chen, T.-F. and Bear, J.-L.: A Performance Study of Software and Hardware Data Prefetching, *21st ISCA*, pp. 223–232 (1994).
3) Dutta, S. and Franklin, M.: Control Flow Prediction with Tree-Like Subgraphs for Superscalar Processors, *28th MICRO*, pp. 258–263 (1995).
4) et al., S. M.: Effective Compiler Support for Predicated Execution Using the Hyperblock, *25th MICRO*, pp. 45–54 (1992).
5) Faggin, F., Jr., M. E., Mazor, S. and Shima, M.: The History of the 4004, *IEEE MICRO*, Vol. 16, No. 6, pp. 10–20 (1996).
6) Kumanoya, M., Ogawa, T., Inoue, K.: Advances in DRAM Interfaces, *IEEE MICRO*, Vol. 15, No. 6, pp. 30–36 (1995).
7) M.Tullsen, D., J.Eggers, S., S.Emmer, J., M.Levy, H., L.Lo, J., L.Stamm, R.: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *23rd ISCA*, pp. 191–202 (1996).
8) Semiconductor Industry Association(SIA): *The National Technology Roadmap for Semiconductors* (1994).
9) S.Lam, M. and robert P.Wilson: Limits of Control Flow on Parallelism, *19th ISCA*, pp. 46–57 (1992).
10) Smith, M. D., Johnson, M., A.Horowitz, M.: Limits on Multiple Instruction Issue, *3rd ASPLOS*, Vol. 3, pp. 290–302 (1989).
11) S.Sohi, G., E.Breach, S., T.N.Vijaykumar: Multiscalar Processors, *22nd ISCA*, pp. 414–425 (1995).
12) Wilkes, M. V.: The Memory Wall and the CMOS End-Point, *Computer Architecture News*, Vol. 23, No. 4, pp. 4–6 (1994).
13) Yeh, T.-Y. and N.Patt, Y.: Alternative Implementations of Two-Level Adaptive Branch Prediction, *19th ISCA*, pp. 124–134 (1992).
14) Yu., A.: The Future of Microprocessors, *IEEE MICRO*, Vol. 16, No. 6, pp. 46–59 (1996).
15) 宮嶋浩志, 岩下茂信, 村上和彰: 高性能システム・オン・チップ構成法に関する性能評価, 情処研究会 HPC 62-6, Vol. 96, No. 81, pp. 33–38 (1996).
16) 坂井修一: オンチップマルチプロセッシングに関する初期的検討, 情処研究会 ARCH 122-7, Vol. 97, No. 15, pp. 33–38 (1997).
17) 村上和彰, 吉井卓, 岩下茂信: 21 世紀に向けた新しい汎用機能部品 PPRAM の提案, 情処研究会 ARCH 108-8, Vol. 94, No. 91, pp. 1–8 (1994).
18) 鳥居淳, 木村真人, 近藤真己, 鈴木研司, 小長谷明彦: 順序付きマルチスレッドアーキテクチャのプログラミングモデルと評価, 並列処理シンポジウム JSPP '96, pp. 299–306 (1996).
19) 田中英彦: ここいらで、計算機アーキテクチャを再考しよう, 情処研究会 ARCH 108-6, Vol. 94, No. 91, pp. 30–40 (1994).
20) 木村真人, 井上俊明, 鳥居淳, 小長谷明彦: 順序付きマルチスレッド実行モデルの提案とその評価, 並列処理シンポジウム JSPP '95, pp. 99–106 (1995).