

## 行列に基づく Out-of-Order スケジューリング方式の評価

五島 正裕<sup>†</sup> 西野 賢悟<sup>†</sup> 小西 将人<sup>†</sup>  
 中島 康彦<sup>†</sup> 森 眞一郎<sup>†</sup>  
 北村 俊明<sup>†</sup> 富田 眞治<sup>†</sup>

Out-of-order スーパー标ーラ・プロセッサは、命令スケジューリングのため、オペランドの有効性を追跡する *wakeup* と呼ぶロジックを持つ。我々は、命令間の依存関係を表す行列を読み出すことで *wakeup* を実現する方式と、その行列を狭幅化することにより遅延を IPC に対するペナルティに転化する手法を提案した。しかし、行列を用いた類似の方式が DEC Alpha 21264 などて採用されている。本稿では、これらの違いを明らかにする。富士通株式会社から提供された 0.18  $\mu\text{m}$  CMOS プロセスのデザイン・ルールに基づいてこれらの方式のロジックを設計し、Hspice によって遅延を測定した。その結果、我々の方式の回路遅延は、21264 の方式の 1/2 程度以下であることが分かった。

### Evaluation of Matrix-based Out-of-Order Scheduling Schemes

MASAHIRO GOSHIMA,<sup>†</sup> KENGO NISHINO,<sup>†</sup> MASAHITO KONISHI,<sup>†</sup>  
 YASUHIKO NAKASHIMA,<sup>†</sup> SHIN-ICHIRO MORI,<sup>†</sup> TOSHIAKI KITAMURA<sup>†</sup>  
 and SHINJI TOMITA<sup>†</sup>

An out-of-order superscalar processor has a logic called *wakeup*, which manages availability of the data for instruction scheduling. We have proposed a new scheduling scheme which substitutes association of the tags by reading a matrix which represents dependences between instructions, and a method to change the delay of the matrix into IPC penalties. A similar scheme based on matrices, however, has already adopted in DEC Alpha 21264. This paper clarifies the difference between them. We designed the logic circuits guided by a design rule of a 0.18  $\mu\text{m}$  CMOS process provided by Fujitsu Limited, and calculated the delays by Hspice. The evaluation result shows that the circuit delay of our scheme is less than half of that of the scheme adopted in 21264.

#### 1. はじめに

現在の out-of-order スーパー标ーラ・プロセッサでは、命令スケジューリングを行うウィンドウ・ロジックの遅延が動作周波数を制限する主要因の 1 つとなりつつある。そのため、その遅延を短縮する研究がさかに行われるようになってきている<sup>1)~10)</sup>。

ウィンドウ・ロジックのうちでは、*wakeup* と呼ばれる部分が特にクリティカルになると予測されている<sup>1),2),6),7)</sup>。命令の実行に必要なデータが揃っていない命令はウィンドウ中で『眠る』ことになるが、*wakeup* はこれらの命令を『起こす』処理である。従来から採用されてきた連想方式では、データに付けられたタグの連想処理によって *wakeup* を行う。この連想処

理のため、*wakeup* の遅延がクリティカルになっていた<sup>1),2),6),7)</sup>。

そこで我々は、この連想処理を廃し、行列を用いて *wakeup* を行う方法を提案した<sup>1),2)</sup>。また DEC Alpha 21264 では、行列を用いた類似の方法がすでに採用されている<sup>3)~5)</sup>。これらの方式は、命令間の依存を表す行列を用いるという点で一致しているが、依存の表現の方法は異なる。前者では生産者から消費者への依存を表す単一の行列を用いる。一方後者では、命令が参照する物理レジスタを介して、生産者から物理レジスタへ、物理レジスタから消費者への依存をそれぞれ表す 2 種類の行列を用いる。そこで本稿では、前者を直接方式、後者を間接方式と呼ぶことにする。

本稿の目的は、これらの 2 方式の違いを定性的、定量的に明らかにすることにある。以下まず 2 章では、その準備として out-of-order スケジューリングの原理と従来の連想方式についてまとめる。そして 3 章と 4

<sup>†</sup> 京都大学  
 Kyoto University

章で間接方式と直接方式について詳しく述べ、その定量的な違いを明らかにする。定量的な比較は 5 章で行う。

2. 連想方式

前述のとおり間接方式では、2 種類の行列を用いて *wakeup* を行う。しかし、文献 4), 5) などでは、下流の 1 種類のみが記述されているため、直接方式との違いが一見して不明瞭になっている。これは、*wakeup* という言葉が曖昧に用いられているためである。本章では、従来の連想方式の説明を行うとともに、*wakeup* のより正確な定義を与える。

2.1 Out-of-Order スケジューリングの原理

Out-of-order スーパースケラ・プロセッサは、論理的なレジスタとは別に、各命令の out-of-order な実行結果を保存するための物理レジスタを必要とする。物理レジスタは、リオーダ・バッファ、あるいは、物理レジスタ・ファイルなどによって実現される。

Out-of-order スケジューリングは、この物理レジスタを用いたデータ駆動型の計算と見なすことができる。すなわち、命令ウィンドウ中で『眠っている』命令は、ソースに割り当てられた物理レジスタにデータが書き込まれることによって『起こされ』、プログラム・オーダとは独立に実行を開始することができる。

2.2 連想方式の原理

従来から用いられてきた連想方式では、この物理レジスタの番号に基づいて *wakeup* を実現する。連想方式では、物理レジスタの番号はタグと呼ばれ、その連想処理によって *wakeup* すべき命令を検索する。

Out-of-order スケジューリングの処理は、以下で述べる 5 つのフェーズに従って進む；ただし、命令  $I_l/I_r$  の結果を、命令  $I_c$  がそれぞれ左/右のソースとして消費する場合を考える。  $I_l/I_r/I_c$  は、それぞれ、命令ウィンドウの  $l/r/c$  番エントリに格納され、また、  $I_l/I_r$  には、その結果の書き込み先として物理レジスタの  $L/R$  番が割り当てられるとする：

- (1) *Rename* フェッチされた命令に対して、レジスタ・リネーミングが施される：  
 デスティネーション まず、デスティネーションに対して、物理レジスタの 1 つが割り当てられる。この物理レジスタの番号を *tagD* と呼ぶ。  
 ソース 同時に、左/右のソースに現在割り当てられている *tagD* が求められる。これを、*tagD* と

区別して、*tagL/R* と呼ぶ。

$I_l/I_r/I_c$  に関しては、 $tagL[c] = tagD[l] = L$  ,  
 $tagR[c] = tagD[r] = R$  となる。

- (2) *Dispatch* その後命令は、ウィンドウに格納される。  
 $I_l/I_r$  がまだ実行されていない場合  $I_c$  は、 $I_l/I_r$  の実行を待って、ウィンドウ内で『眠る』ことになる。
- (3) *Wakeup* 連想方式の *wakeup* は、タグに基づく連想検索によって行われる。後述する *select* によって  $I_l/I_r$  の発行が許諾されると、 $tagD[l/r]$  から  $L/R$  が読み出され、ウィンドウ内のすべての *tagL/R* からの連想検索が行われる。*tagL/R[c]* の内容が  $L/R$  と一致するので、左/右のソースが利用可能であることを表すフラグ *rdyL/R[c]* がセットされる。
- (4) *Select* *rdyL/R* がともにセットされている命令が発行可能な命令である。*Select* では、発行要求が調停され、選択された命令の発行が許諾される。
- (5) *Issue* 発行を許諾された命令の情報が命令ウィンドウから読み出され、実行ユニットに送られる。

2.3 連想方式の wakeup ロジック

図 1 に、連想方式の *wakeup* ロジックのブロック図を示す。連想方式の *wakeup* ロジックは、*tagD* を

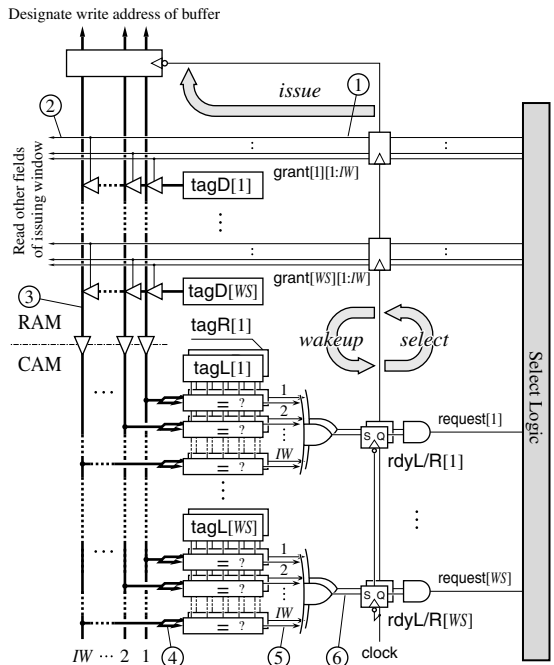


図 1 連想方式の wakeup ロジック  
 Fig. 1 Wakeup logic of associative scheme.

簡単のため各命令は左/右 2 つのソースを持つとしたが、3 つ以上のソースを持つ場合にも容易に拡張できる。

格納する RAM 部と, tagL/R をキー, rdyL/R を内容とする CAM 部からなる. 同図中,  $IW$  は命令発行幅 (Issue Width) を,  $WS$  はウィンドウ・サイズを表す.

RAM 部からは, tagD が読み出される. この RAM 部は, 通常の RAM とは異なり, 行デコーダを持たない. 読み出しワードラインには, パイプライン・ラッチを介して, *select* ロジックからの発行許諾信号 *grant* が直接接続される. 読み出される tagD は, (1) 書き込むべき物理レジスタを指示するために上部のパイプライン・ラッチに, (2) *wakeup* のために下部の CAM 部に, それぞれ送られる. したがって, tagD の読み出しは, *issue* と *wakeup* の両方に含まれることになる. なお, 上部のパイプライン・ラッチに送られた tagD は, 適当なサイクル数だけ遅延され, 物理レジスタの書き込みアドレスとして使用される.

CAM 部では, 比較入力ポートに入力された tagD と一致する tagL/R が連想検索され, 対応する rdyL/R がセットされる. CAM 部の出力側では, rdyL/R を記憶するレジスタの出力信号が, 組合せ回路的に *select* の入力 *request* に接続されている.

*Select* ロジックは,  $WS$  個の *request* を調停し, *wakeup* ロジックに対して  $IW$  組の *grant* をアサートする. このように, *wakeup* と *select* は, 緊密なフィードバック・ループを形成している.

ロジックの動作

図 2 (L-1) に, *wakeup*, *select*, *issue* の命令パイプラインでの位置づけとその動作の様子を示す. 同図は, MIPS R10000 のパイプライン構成に準ずる<sup>11)</sup>. 図中, Exec は実行を, RF $\rightarrow$   $\rightarrow$  RF は物理レジスタ・ファイルに対する読み出しと書き戻しを表す. 同図は, タイミングが最もクリティカル, すなわち, レイテンシが 1 である命令  $I_p$  の次のサイクルで  $I_c$  が実行される場合を示している.  $I_p$  が生成したデータは, オペランド・バイパスを通して  $I_c$  の実行に使用される.

ロジックの動作は, 以下のようにまとめられる:

$A_2$  *Select* ロジックが  $I_p$  を選択する.

- $B_1$  *grant*[ $p$ ] がアサートされ,  $I_p$  に対する *issue* と同時に,  $I_c$  に対する *wakeup* が行われる. RAM 部から読み出された tagD [ $p$ ] は, *wakeup* のために CAM 部に入力され, rdyL/R [ $c$ ] がセットされる.
- $B_2$  *request* [ $c$ ] がアサートされ,  $I_c$  も選択の対象となる. 実際に  $I_c$  が選択されると,  $C_1$  では *grant* [ $c$ ] がアサートされることになる.

ロジックのパイプライン化

命令スケジューリングの 5 つのフェーズのうち, *rename*, *dispatch*, および, *issue* は, 必要であれば, パイプライン化を施すことによってシステムのクリティカル・パスから外すことができる. パイプライン化の代償は, 分岐予測ミス・ペナルティの増加であり, 通常受け入れられる. 実際現存するスーパースケーラ・プロセッサでは, *rename* の遅延のため, デコード・ステージに複数サイクルをあてるのが普通である.

一方, *wakeup* と *select* は, 実際上パイプライン化できない. 図 2 (L-2) に, *wakeup* に 1 サイクル余分にかけた場合の命令パイプラインの様子を示す. *Select* から *wakeup* へのフィードバックにより, ( $A_2$ )  $I_p$  に対する *select* が終わった後にしか, ( $B_1$ )  $I_c$  に対する *wakeup* は開始できない. その結果,  $I_c$  の発行は 1 サイクル遅れ,  $I_p$  と back-to-back に実行できなくなる.

同図では,  $I_p$  が生成したデータは, オペランド・バイパスを通る必要はなく, レジスタ・ファイルを介して  $I_c$  の実行に使用される. すなわち, *wakeup* と *select* に 1 サイクルより多くを割り当てることは, IPC の観点からは, レイテンシが 1 である演算器 — 通常の構成では ALU からのオペランド・バイパスを取り除くことと等価である. それによる IPC の悪化は最大 15% 程度にもなり<sup>1),2)</sup>, 動作周波数の向上に見合わない可能性が高い. したがって, レイテンシが 1 であるパスに関しては, *wakeup* と *select* からなるループ 1 週の遅延は 1 サイクル以内でなければならないと結論づけられる. 逆にいえば, このループ 1 週の遅延がクリティカルである場合には, それがシステムの動作周波数を決定することになる.

2.4 *Wakeup* の定義

次章からは, 行列を用いた 2 つの方式, 間接方式, 直接方式について述べる. 両方式の比較のためには, 前述のとおり, *wakeup* の処理を正確に定義する必要がある.

*Wakeup* を, 連想方式の CAM 部だけと定義している文献も多い<sup>4)~7)</sup> が, 本稿では, 本章で述べてきたとおり, RAM 部と CAM 部, を *wakeup* と定義して

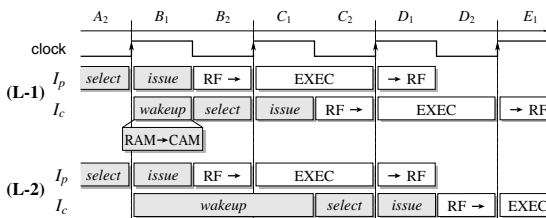


図 2 命令パイプラインにおける *wakeup*, *select*, *issue*  
Fig. 2 The *wakeup*, *select* and *issue* phases in the instruction pipeline.

いる。

本稿の定義は、以下の理由により、より妥当である；システムの動作周波数を決定するうえで重要であるのは、前述のループ 1 周の遅延である。そのうち *select* の処理は *wakeup* の方式とはほぼ独立であるが、残りの部分、すなわち、RAM 部と CAM 部の形式は、後述するように、互いに強く依存しており、どちらか一方だけを取り上げることは意味がない。また、5 章で述べるように、RAM 部の遅延は CAM 部の遅延の 1/2 程度もあり、決して無視することはできない。

3. 間接方式<sup>3)~5)</sup>

間接方式は、*grant* から *request* を求めるにあたって、割り当てられた物理レジスタが利用可能かどうかを表すベクトル *prrdy* を経由する。*grant* から *prrdy*、*prrdy* から *request* を求めるには、それぞれ、*IR*、*RI* の 2 つの行列を用いる。

3.1 間接方式の原理

*IR*/*RI* は、それぞれ、命令とそのデスティネーション/ソースとなる物理レジスタとの関係を表す。したがって、それぞれ、*WS* 行 × *NR* 列である。理由は後述するが、*NR* = 2 · *WS* 程度とすることが普通である。

図 3 に、間接方式の概念図を示す。同図は、2.2 節と同様に、ウィンドウ・エントリ *l/r* 番に格納された命令 *I<sub>l</sub>/I<sub>r</sub>* が生産する結果を、物理レジスタ *L/R* 番を介して、ウィンドウ・エントリ *c* 番に格納された命令 *I<sub>c</sub>* が消費する場合を示す。行列 *IR* の *l/r* 行では、それぞれ、*L/R* 列の 1 要素のみが 1、それ以外は 0 と

なっている。一方、行列 *RI* の *c* 行では、*L/R* 列の 2 要素が 1、それ以外は 0 となっている。

命令の発行が許諾されるとまず、行列 *IR* に従って、*grant* から *prrdy* を求める。発行が許諾された命令に対応するすべての行を列ごとに OR した得られる行ベクトルが、セットすべき *prrdy* を表す。図 3 では、*IR* の *l/r* 行によって、*prrdy*[*L/R*] がそれぞれセットされる。

次いで、行列 *RI* に従って、*prrdy* から *request* を求める。*c* 行では *L/R* 列が 1 であるので、物理レジスタ *L/R* 番がともに利用可能であれば、*I<sub>c</sub>* が発行可能である。すなわち、*prrdy*[*L/R*] がともに 1 であれば、*request*[*c*] を 1 とする。

3.2 間接方式のロジック

図 4 に、間接方式のロジックを示す。ロジックは、図 3 と 1 対 1 に対応するアレイによって実装される。

*IR*

*IR* に対応する部分は、*NR* b × *WS* word、*IW*-write、*1*-read の RAM の読み出しによって実現される。ただし、連想方式の RAM 部と同様に、行デコーダは必要なく、*grant* が直接ワードラインに接続される。

また、通常の RAM とは異なり、発行が許諾される命令に対応して、複数のワードラインが同時にアサートされる。そのためリード・ポートは、必然的にシングル・ビットラインとなる。ダブル・ビットラインとしても、1 であるセルがある行では、両方のビットラインがプルダウンされてしまい、いずれ差動出力を得ることはできないからである。そのため、シングル・エンドのセンスアンプを用いる必要がある<sup>12)</sup>。

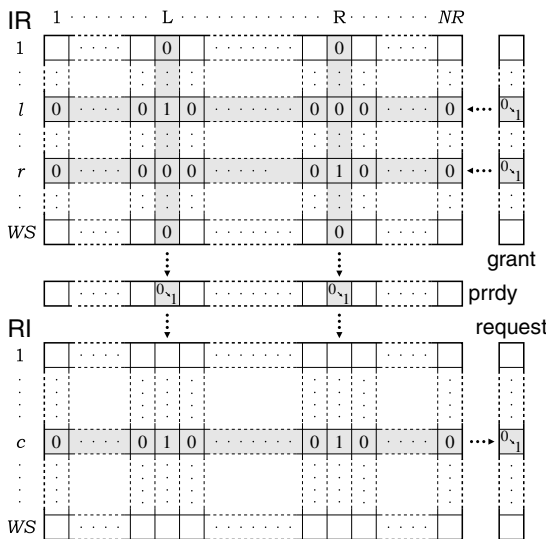


図 3 間接方式の概念図

Fig. 3 Indirect scheme.

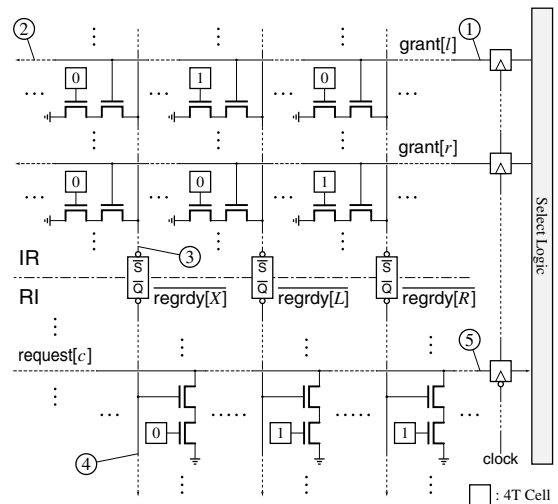


図 4 間接方式のロジック

Fig. 4 Logic of indirect scheme.

**RI**

RI は、各行に対応する  $WS$  個の wired-AND として実現される。request は、あらかじめ high にプリチャージされ、接続された  $NR$  個のプルダウン・スタックによってプルダウンされる。図 4 の場合、 $c$  行  $X$  列のセルの値は 0 であるため、 $\overline{\text{prrdy}[X]}$  の値にかかわらずプルダウン・スタックは ON にならず、request の値に影響を及ぼさない。L/R 列では、セルの値が 1 であるため、 $\overline{\text{prrdy}[L/R]}$  のどちらか一方が high であると request[c] はプルダウンされる。逆に、 $\overline{\text{prrdy}[L/R]}$  がともに low であれば、request[c] は high に保たれる。

なお、図 4 の例では、IR と RI はまったく同じ構造のセルを用いて実現されている。

**prrdy**

RI において、request[c] が high に保たれるには、 $\overline{\text{prrdy}[L]}$  と  $\overline{\text{prrdy}[R]}$  が同時にアサートされる必要がある。一方で、grant[l] と grant[r] は、一般には異なるサイクルでアサートされる。そのため prrdy の位置には、単なるパイプライン・ラッチではなく、1b のレジスタを置く必要がある。

その一方で、RI と select ロジックの間には、連想方式における rdyL/R (図 1) のようなレジスタは必要なく、パイプライン・ラッチを置けばよい。

**3.3 連想方式との関係**

間接方式における IR/RI の各行は、基本的には、前章で述べた連想方式における物理レジスタ番号、すなわち、タグをデコードしたものである。IR/RI は、連想方式の RAM 部/CAM 部と 1 対 1 に対応している。

連想方式では、 $IW$  個のタグに対する連想処理を、それぞれ  $IW$  個の部分回路で行っていた。一方間接方式では、タグをデコードすることによって、1 個の回路でまとめて処理している。間接方式のロジックは、連想方式のロジックに対して、以下のように簡単化されている：

RAM 部と IR 読み出しポート数が  $IW$  本から 1 本に削減される。

CAM 部と RI 一致比較が単純な和積に変わるとともに、入力ポート数が  $IW$  本から 1 本に削減される。

しかしその一方で、ビット幅は  $\lceil \log_2 NR \rceil$  b から  $NR$  b へと大幅に増加している。これらの得失が遅延に与える影響については、5 章で詳しく述べる。

**4. 直接方式<sup>1),2)</sup>**

間接方式が IR, RI の 2 種類のを逐次的にアクセスするのに対して、文献 1), 2) で提案した直接方式では

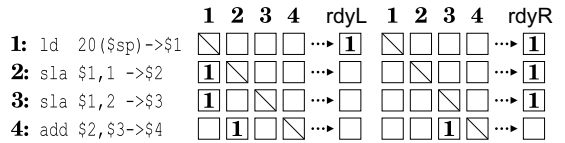


図 5 直接方式の依存行列  
Fig. 5 Dependence matrices of direct scheme.

直接 grant から request を求める。また、文献 1), 2) では、行列の実効サイズを縮小しアクセスを高速化する手法もあわせて提案している。

文献 1), 2) で提案した方式では、ソース・オペランドの数だけの行列を必要としていたが、本稿ではそれらを単一の行列に単一化する方法を提案する。以下まず 4.1 節で文献 1), 2) に基づいて直接方式の原理について説明した後、4.2, 4.3 節で単一化について述べ、4.4 節では単一化した行列の高速化について述べる。

**4.1 直接方式の原理**

図 5 に、直接方式の依存行列の概念図を示す。行列は、rdyL/R 用に各 1 つずつ用意された  $WS \times WS$  の行列である。ただし、対角要素は使用しない。ウィンドウ・エントリ  $p$  番の命令  $I_p$  の実行結果をウィンドウ・エントリ  $c$  番の命令  $I_c$  が消費する場合、 $c$  行  $p$  列の要素は 1、そうでなければ 0 とする。

Wakeup においては、発行される命令に対応する列の行ごとの OR を求めれば、セットすべき rdyL/R を表す列ベクトルを求めることができる。

この処理は、間接方式における IR と転置の関係にあり、IR と同様のロジックによって実現される。

**4.2 ウィンドウ・エントリと物理レジスタの寿命**

行列の単一化について理解するためには、ウィンドウ・エントリと物理レジスタの寿命の違いについて理解しておく必要がある。そこで本節でまず、それらの寿命についてまとめた後、次節で行列の単一化について述べる。

さて、ある命令に対して、それが格納されるウィンドウ・エントリと、その命令のデスティネーションとして割り当てられる物理レジスタは、異なる寿命を持つ。ウィンドウ・エントリと物理レジスタは、フェッチされた命令に対してほぼ同時に割り当てられるが、開放のタイミングが異なる。投機失敗時の回復の方式などにも依存するが<sup>4)</sup>、基本的には、それぞれ以下のタイミングで開放される：

ウィンドウ・エントリ 命令の実行が終了すれば開放してよい。

物理レジスタ 命令の実行が終了した時点ではなく、以下のように、当該物理レジスタを参照する命令

がウィンドウ内に存在しなくなるまで解放できない：

リオーダ・バッファ方式 当該エントリがバッファからリタイアするときまで

レジスタ・リネーミング方式 論理レジスタを上書きする次の命令が完了するときまで<sup>11)</sup>

この寿命の違いは、以下のように、ウィンドウ・エントリと物理レジスタは個別に管理されるべきであることを意味する。

ウィンドウ・エントリは、サイクル・タイムとのトレードオフと観点からいって、物理レジスタに比べ高価な資源である。ウィンドウ・エントリの数、すなわち、 $WS$  は、*wakeup*, *select* の遅延に直接影響するからである。

仮に、ウィンドウ・エントリと物理レジスタを 1 対 1 に対応させ、1 つの機構によって一括管理するとすると、ウィンドウ・エントリにも物理レジスタの寿命が強制されることになる。すなわち、対応する物理レジスタの寿命が尽きるまで、物理レジスタより高価なウィンドウ・エントリまで再利用できなくなってしまう。

以上の理由から、ウィンドウ・エントリと物理レジスタを一括管理することは受け入れ難い。実際、現存するプロセッサでは、その管理コストにもかかわらず、ウィンドウ・エントリと物理レジスタを個別に管理している。物理レジスタの数  $NR$  は、 $NR = 2 \cdot WS$  とすることが多い<sup>3),11)</sup>。

#### 4.3 行列の単一化

文献 1), 2) で示した直接方式では、ソースごとに 1 つの行列を必要とした。一方間接方式では、ソースの数にかかわらず、1 組の行列で済む。ソースの数は、 $RI$  の各行の 1 の数として表れる。そこで直接方式でも、間接方式と同様に、1 つの行列への単一化を試みる。

行列を単一化するには、基本的には、各行列を要素ごとに OR すればよい。図 5 の例では、単一化された行列の第 4 行では、第 2, 3 列の 2 要素が 1 となる。

単一化された行列のロジックは、図 6 のようになる。図 4 に示した間接方式の  $RI$  部と基本的には同じだが、アレイの幅は  $NR$  から  $WS$  へと大幅に削減される。削減の程度については、次節で詳しく述べる。

ただし、 $RI$  の場合と同様に、*grant* が異なるサイクルでアサートされることへの対処が必要である。それには、2 つの対処法が考えられる。1 つ目は、やはり  $RI$  と同様に、行列の前にレジスタを置くことである。しかしこの方法は、直接方式ではうまくいかない。以下、1 つ目の対処法がうまくいかない理由と、2 つ目の対処法について順に述べる。

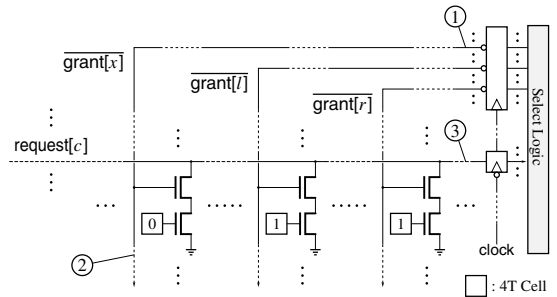


図 6 直接方式のロジック  
Fig. 6 Logic of direct scheme.

#### 対処法 1

間接方式の *prrdy* と同様、行列の前にレジスタ *irdy* を置くことにしよう。

その役割を果たすためには、*irdy* も、*prrdy* と同様に、それに依存する命令がウィンドウ内に存在している間セットされていなければならない。すなわち *irdy* は、前節の説明に従えば、ウィンドウ・エントリより長い、物理レジスタと同様の寿命を持つとすることができる。すなわち、格納された命令  $I_p$  の実行が終了した後、ウィンドウ・エントリ  $p$  番は原理的には解放してよいが、*irdy* の  $p$  番は解放できない。

そのため、解放されたウィンドウ・エントリ  $p$  番も、*irdy* の  $p$  番が解放されるまで、再利用することはできない。したがって、ウィンドウ・エントリも、*irdy* と同じ物理レジスタの寿命を強制されることになる。このことは、前節で述べたとおり、通常受け入れられない。

#### 対処法 2

2 つ目の対処法として、*grant* された命令に対応する列をリセットすることが考えられる。たとえば、*grant[l]* と *grant[r]* が、この順序で別のサイクルにアサートされた場合、以下ようになる：

- (1) *grant[l]* がアサートされると、 $l$  行のスタックによって *request[c]* はプルダウンされる。
- (2) そのプリチャージ・フェーズで  $l$  列をリセットする。以降では、*grant[l]* にかかわらず  $l$  行のプルダウン・スタックは ON にならない。
- (3) したがって、*grant[r]* がアサートされる時には、*request[c]* は high に保たれる。

列をリセットするには、専用のポートを用意するのが最も低コストであろう。各セルに対しては、小型の  $n$ MOS ゲートを 1 つ追加するだけでよい。

正確には、リオーダ・バッファ方式の物理レジスタと同じ寿命を持つ。

なおこの方法は、行列上に保存された依存関係を発行時に破壊するため、投機失敗時の状態回復の方法が制限される。しかし、このことに依存しない効率の良い回復法も提案されている<sup>4)</sup>。

4.4 行列アクセスの高速化

本節では、文献 1), 2) で提案した行列アクセスの高速化手法について述べる。そのうちの一部は間接方式にも適用可能であり、その方法については 4.5 節で述べる。

4.4.1 行列の分散化

実際のプロセッサでは、命令ウィンドウは、集中化された単一のロジックとして実装されるのではなく、演算器のクラスごとに設けられたリザベーション・ステーションや命令キューとして分散実装されることが多い。実際、最近のレジスタ・リネーミング方式のプロセッサの多くは、整数 (INT), ロード/ストア (LS), 浮動小数点 (FP) 命令の系統ごとにウィンドウを分散化している。DEC Alpha 21264 は INT, LS 用と FP 用の, HP PA-8000 は INT, FP 用と LS 用の, それぞれ 2 つの, MIPS R10000 は INT 用, LS 用, FP 用の 3 つのサブウィンドウを持つ<sup>3),11),13)</sup>。このような、命令の系統ごとのウィンドウの分散化は非常に重要である。わずかな IPC のペナルティを犠牲に、ロジックの実効サイズを大幅に縮小できるからである。

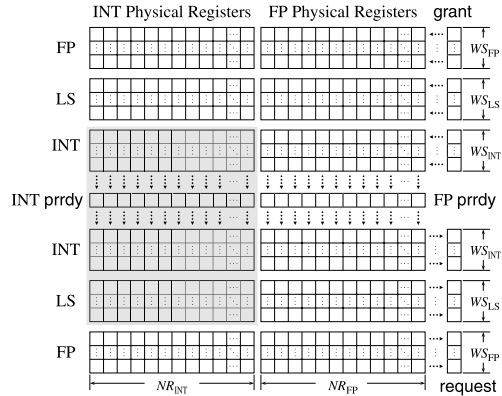
R10000 と同様の分散化を施した場合の行列の様子を図 7 (b) に示す。R10000 では、INT, LS, FP の各サブウィンドウの発行幅とサイズはそれぞれ、 $IW_{INT} = IW_{LS} = IW_{FP} \equiv IW' = IW/3 = 2$ ,  $WS_{INT} = WS_{LS} = WS_{FP} \equiv WS' = WS/3 = 16$  である。

分散化が行列に与える第 1 の効果は、書き込みポート数の削減である。行列に対する書き込みは行に対して行われる。ウィンドウを分散化した場合、たとえば INT 命令を *dispatch* するとき書き込まれる行は、図 7 (b) では上から  $WS_{INT}$  行に制限される。したがってこの部分のセルの書き込みポート数は、 $IW$  から  $IW_{INT}$  へと  $1/3$  に削減される。LS, FP についても同様である。

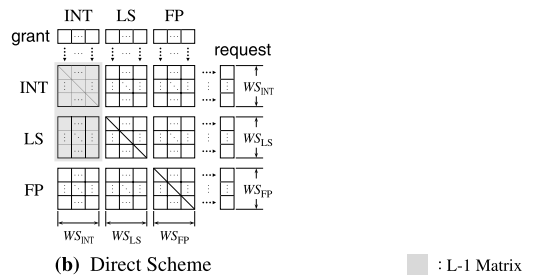
また、R10000 のように INT のみのサブウィンドウを持つ場合、次項で述べる多階層化が可能になる。

4.4.2 行列の多階層化

R10000 の構成では、レイテンシが 1 であるパスは、INT から INT, INT から LS の 2 つである。図 7 (b) では、影を付けた部分がこれに相当する。この部分を



(a) Indirect Scheme



(b) Direct Scheme

図 7 行列の分散化

Fig. 7 Decentralization of matrices

取り出し、これを L-1、残りを L-2 行列と呼ぶ。

L-1 アクセスは *select* と合わせて 1 サイクル以内に行う必要があるが、L-2 アクセスはパスのレイテンシに合わせて適当にパイプライン化してよい。図 2 (L-2) は、レイテンシが 2 の場合に当たる。同図では、L-2 アクセスには 1.5 サイクル、すなわち、L-1 の 3 倍の時間をかけており、L-2 がクリティカルになる可能性はきわめて低い。

一方 L-1 は、元の行列から比べると格段に少量化され、その分だけ遅延も短縮される。さらに、L-1 に対しては次項で述べる狭幅化を適用することができる。

4.4.3 L-1 行列の狭幅化

依存する命令間の距離は短い場合が多い。この性質を利用して、L-1 をさらに縮小することができる。各行において、先行する  $w$  命令に対するビットだけを残し、それ以外を L-2 に移すのである。*WakeUp* は、命令間の距離が  $w$  以下の場合には L-1 によって、そうでない場合には L-2 によって行われる。前者を L-1 ヒット、後者をミスと呼ぶ。ミスの場合には、1 サイクルのペナルティが生じる。

図 8 に、L-1 の縮小の様子を示す。左は元々の、右が縮小後の L-1 である。L-2 に移されるセルは、左図中で薄く示してある。メモリのラインのうちでは、書

SPARC など、INT-FP レジスタ間の転送命令がない ISA では、同図中破線で記した部分行列を省略できる。

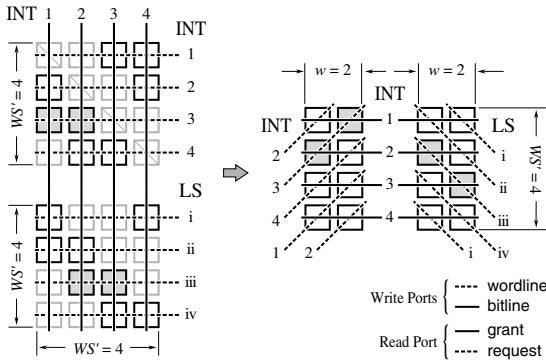


図8 L-1 行列の狭幅化  
Fig. 8 Narrowing of L-1 matrix.

き込みポートのビットラインが  $2 \cdot IW'$  本と最も本数が多い。したがって図8では、それらが直線になるように、L-1に残されるセルを矩形領域に集めている。 $w$ は、L-1を構成するアレイの幅にあたる。そのため、この技術をL-1行列の狭幅化と呼ぶ。

図8右から明らかなように、狭幅化されたL-1のワード、ビットラインは、それぞれ  $w$  個のセルにしか接続されていない。したがって *wakeup* の遅延は、 $WS$  とは独立に、 $w$  によって決まる。

なお、各サブウィンドウ間では、エントリ消費の歩調を揃える *pace-keeping* と呼ぶ操作が必要であり、ウィンドウ・エントリの使用効率が若干悪化する<sup>1),2)</sup>。

$w \geq \min(WS'/4, 64)$  であれば、L-1ミスではなく、主に *pace-keeping* によってIPCは低下し、それは1~2%に抑えられることが分かっている<sup>1),2)</sup>。

#### 4.5 間接方式に対する高速化手法の適用

間接方式のIR, RIに対しても、直接方式と同様の分散化と多階層化を適用することができる。図7(a)に、その様子を示す。同図(a)は、同図(b)と同様、R10000の構成に対するものである。

物理レジスタは、INT, FPのそれぞれに分けられ、それぞれの個数は、 $NR_{INT} = 2(WS_{INT} + WS_{LS})$ 、 $NR_{FP} = 2(WS_{FP} + WS_{LS})$ 、すなわち、 $NR_{INT} = NR_{FP} \equiv NR' = 4 \cdot WS'$  とすることが多い<sup>3),11),13)</sup>。

しかし、L-1の狭幅化は、間接方式には適用することができない。直接方式と同様に、割り当てられるウィンドウ・エントリと物理レジスタとの関係を制限して行列を縮小することは可能だが、直接方式における命令間の近さのような有効なヒューリスティクスが、ウィンドウ・エントリと物理レジスタの間には存在しないからである。

表1 セル面積(面積( $F^2$ )(縦( $F$ )×横( $F$ )))

Table 1 Cell areas (area ( $F^2$ ) H ( $F$ ) × W ( $F$ )).

	×1/2	×1	×2
連想方式	960 (24 × 40)	1360 (34 × 40)	4320 (54 × 80)
間接方式	360 (20 × 18)	400 (20 × 20)	572 (26 × 22)
直接方式	400 (20 × 20)	400 (20 × 20)	572 (26 × 22)

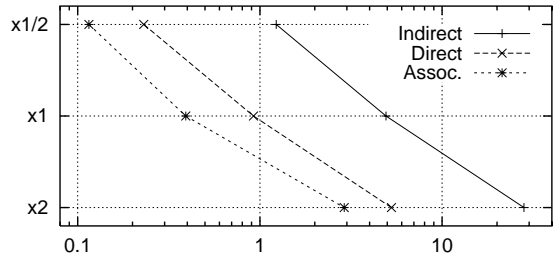


図9 回路面積 ( $\times 10^6 F^2$ )  
Fig. 9 Areas ( $\times 10^6 F^2$ ).

## 5. 回路の評価

富士通株式会社から提供されたCS80Aプロセスのデザイン・ルールに基づいて、各方式の *wakeup* ロジックのレイアウト設計を行った。得られたレイアウトから回路面積を求め、Hspiceによって遅延を求めた。

CS80Aは、ゲート長  $0.18 \mu\text{m}$  のバルクCMOSプロセスで、ゲート、層間絶縁膜は  $\text{SiO}_2$  である。配線は、6層アルミであるが、評価では下3層のみを用いている。

### 5.1 評価モデル

ベース・モデルとしては、MIPS R10000<sup>11)</sup>を用いた。その1/2倍、1倍、2倍の計算資源を持つモデルを評価した。それぞれを、×1/2、×1、×2と呼ぶ。それぞれのパラメータは、 $IW' = 1, 2, 4$  に対して、 $WS' = 8 \cdot IW'$ 、 $NR' = 4 \cdot WS'$ 、 $IW = 3 \cdot IW'$ 、 $WS = 3 \cdot WS'$ 、 $NR = 2 \cdot NR'$  である。

### 5.2 回路面積

間接、直接方式ともに、書き込むべき物理レジスタを指示するtagDを格納するため、連想方式のRAM部と同じものを別途必要とする。したがって、連想方式のCAM部と、間接、直接方式の行列の面積を比較する。

表1にセルの面積を、図9に総面積を示す。

#### セル面積

間接方式のIRとRI、および、直接方式では、ほぼ同一のセルを用いている(3.2節参照)。直接方式では、間接方式のセルに加えてリセット用のポートを必要とする(4.3節)が、セル面積にはほとんど影響を与えなかった。



多ポート・メモリのセルの面積はポート数の 2 乗に比例するため、各方式のセル面積は、基本的には、 $O(IW'^2)$  で与えられる。しかしその影響の大きさは、 $IW'$  の絶対的な大きさに依存する。

間接、直接方式のセルでは、書き込みポート数が  $IW' = 1, 2, 4$  と増加するが、4T セルや読み出しポートなどの定数成分が支配的であるため、 $\times 2$  でも 4 割程度の増加にとどまっている。

一方、連想方式の CAM 部の比較入力ポート数は、INT と LS は INT と LS から、FP は FP と LS からの tagD を受け付けるため、 $2/3 \cdot IW (= 2 \cdot IW')$  となる。書き込みポート数は間接、直接方式と変わらないが、比較入力ポート数は  $2 \cdot IW' = 2, 4, 8$  と増加し、 $O(IW'^2)$  が支配的になる。その影響は表 1 から見てとれる。

### 総面積

セルの総数は、連想方式は  $WS \log NR'$ 、間接方式は  $2 \cdot WS \cdot NR = 16/3 \cdot WS^2$ 、直接方式は  $WS^2$  である。

間接方式と直接方式のセルの面積はほぼ同一であるから、間接方式の総面積は直接方式の総面積のほぼ  $16/3$  倍となっている。

連想方式の総面積は、 $\log NR' = 5, 6, 7$  程度であるので、 $O(IW'^2) \times O(WS \log NR') \simeq O(IW'^2) \times O(WS) = O(IW'^3)$  とできる。

一方、間接、直接方式の総面積は、セル面積の変化が 4 割程度であるので、 $O(WS^2) = O(IW'^2)$  となり、連想方式よりオーダは小さい。図 9 でも、間接、直接方式の曲線の傾きは連想方式のそれよりも若干緩やかになっている。

結局、どのモデルにおいても、直接方式の面積は連想方式の面積のほぼ 2 倍、間接方式の面積は直接方式の  $16/3$  倍となっている。

### 分散化の効果

上述したセル面積のデータは、これ以上のウィンドウの分散化は効果が小さいことも同時に示す。連想方式の RAM 部の比較入力ポート数は、基本的には、 $IW'$  ではなく  $IW$  の関数であり、単純に削減すると深刻な IPC の低下をもたらす。したがって、INT, LS, FP の 3 つのサブウィンドウをさらに分散化する効果は、書き込みポート数のさらなる削減のみである。

間接、直接方式の場合、ウィンドウをたとえば倍に分散化することは、 $\times 2, \times 1$  のセルを  $\times 1, \times 1/2$  に置き換えることを意味する。表 1 から明らかなように、これにはほとんど効果がない。連想方式では、比較入力ポートの影響がより大きいので、効果はさらに

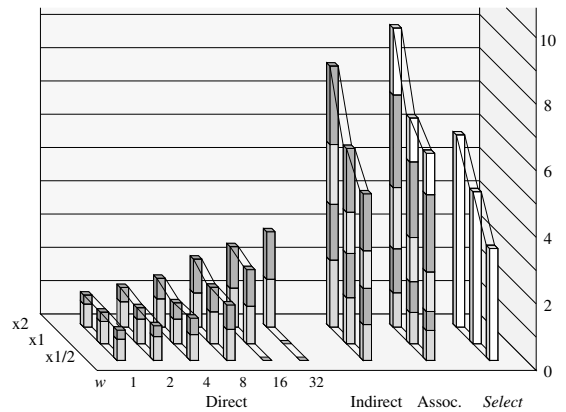


図 10 各方式の回路遅延

Fig. 10 Delays.

小さい。

### 5.3 回路遅延

高速化のため、それぞれの回路にはバンク分割を施した。連想方式の CAM 部は横、間接方式の IR, RI は縦に、2 バンクに分割した<sup>3)</sup>。直接方式では、図 8 右に示したように、INT-INT, INT-LS の 2 つの小行列をそれぞれ 1 つのバンクとしている。

図 10 に、直接 (Direct)、間接 (Indirect)、連想 (Assoc) の各方式の wakeup、ならびに、select ロジックの回路遅延を示す。間接、直接方式は、L-1 行列の遅延である。直接方式の  $w$  は、狭幅化後の L-1 の幅である。Select ロジックの遅延は、固定優先順位の prefix-sum 方式のものである<sup>1), 2), 8)</sup>。縦軸は、F.O.4 インバータの遅延 (60.0ps) で正規化してある。連想、間接、直接方式の各バーは、それぞれ、5, 4, 2 つの部分からなる。下から  $n$  番目の部分は、それぞれ、図 6, 4, 1 に示した各方式の回路図中の、丸数字の  $n$  番から  $n+1$  番で示した点の間の遅延を表す。白色の部分は主にゲート遅延からなり、灰色の部分は配線遅延の影響を受ける。なお直接方式で、 $\times 1$  の  $w = 32$ 、 $\times 1/2$  の  $w = 16$ 、32 のバーが存在しないのは、そのような構成をとりえないためである。 $\times 1$  では、 $WS' = 16$  なので、 $w = 32$  という構成をとりえない。 $\times 1/2$  でも同様である。

#### 間接方式

間接方式の RI の遅延は連想方式の CAM 部のそれに比べて大幅に短縮されている一方で、IR の遅延は大幅に増加している。これは、ビット幅が  $\log_2 NR'b$  から  $NR'b$  へと 1 桁前後増大しているためである。結局、連想方式に対する wakeup 全体の遅延の改善はごくわずかである。

## 直接方式

一方直接方式の遅延は、両者に対して大幅に短縮されている。L-1の狭幅化を行わなくても、その遅延は両方式の1/2以下である。 $w = WS'/4 = 8$ まで狭幅化した場合、IPCは約2%程度低下するが<sup>1),2)</sup>、遅延はさらに半減される。

直接方式の遅延が半分であることは、その程度まで削減しないとクリティカルパスではなくなる、ということではなく、命令発行幅が拡大する将来にわたり、クリティカルパスとならないような設計マージンを提供するものである。

また、 $\times 1/2$ から $\times 1$ 、 $\times 2$ へと、 $IW$ 、 $WS$ 、 $NR$ などのパラメータが増加していくに従い、連想方式、間接方式の遅延は急激に増大しているが、直接方式の遅延の増大の割合は比較的緩やかである。

## 分散化の効果

直接方式では、遅延に影響を及ぼすセルの個数は $w$ によって決まる。したがって、同一の $w$ に対する遅延の変化は、書き込みポート数の増減によるセル面積の増減の効果による。グラフから、その影響はわずかであることが分かる。前節において、ウィンドウのさらなる分散化は回路面積に対して効果が小さいと述べたが、遅延に対しても同様であることが示されている。連想、間接方式に対しては直接の証拠は示していないが、同様であることは容易に推測できよう。

## 6. おわりに

本稿では、out-of-order スーパースケラ・プロセッサの行列を用いたスケジューリング方式として、DEC Alpha 21264で採用されている間接方式と、我々が提案した直接方式について述べた。富士通株式会社から提供された $0.18\mu\text{m}$  CMOS プロセスを用いて、それらの回路の面積と遅延を比較した。

間接方式は、2つの行列を逐次的に用いて *wakeup* を実現する。間接方式では、回路は単純化されるものの、回路規模が大幅に増加するため、従来方式に対してほとんど遅延の改善がみられなかった。

一方直接方式は、間接方式の2つの行列を1つにまとめた方式と考えることができる。そのため、間接方式に対して、面積は3/16、遅延は1/2以下になることが分かった。

謝辞 富士通株式会社には、LSIの設計情報をご提供いただいたので、ここに深甚なる謝意を表したい。本研究の一部は、文部科学省科学研究費補助金、基盤研究(B)(2) #13480083、特定領域研究「情報学」 #13224050による。

## 参考文献

- 1) 五島正裕ほか：スーパースケラのための高速な動的命令スケジューリング方式，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.42, No.SIG 9(HPS 3) (2001).
- 2) Goshima, M., et al.: A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors, *MICRO-34* (2001).
- 3) Farrell, J., et al.: Issue logic for a 600-MHz out-of-order execution microprocessor, *IEEE J. of Solid-State Circuits*, Vol.33, No.5 (1998).
- 4) Morancho, E., et al.: Recovery Mechanism for Latency Misprediction, *PACT* (2001).
- 5) Brown, M., et al.: Select-Free Instruction Scheduling Logic, *MICRO-34* (2001).
- 6) Palacharla, S., et al.: Quantifying the Complexity of Superscalar Processors, Technical Report, Univ. of Wisconsin-Madison (1996).
- 7) Palacharla, S., et al.: Complexity-Effective Superscalar Processors, *ISCA24* (1997).
- 8) Henry, D., et al.: Circuits for Wide-Window Superscalar Processors, *ISCA27* (2000).
- 9) Canal, R., et al.: A low-complexity issue logic, *ICS'00* (2000).
- 10) 佐藤寿倫ほか：連想検索を取り除いたスーパー スカラプロセッサ向け命令発火機構，*JSP2001* (2001).
- 11) Yeager, K.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, No.4 (1996).
- 12) Bakogulu, H.: *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley (1990).
- 13) Kumar, A.: The HP PA-8000 RISC CPU, *Hot Chips VIII*, pp.9-20 (1996).

(平成14年1月29日受付)

(平成14年5月10日採録)



五島 正裕 (正会員)

1968年生。1992年京都大学工学部情報工学科卒業。1994年同大学大学院工学研究科情報工学専攻修士課程修了。同年より日本学術振興会特別研究員。1996年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手。1998年同大学大学院情報学研究科助手。高性能計算機システムの研究に従事。2001年情報処理学会山下記念研究賞，2002年同学会論文賞受賞。IEEE会員。



西野 賢悟(学生会員)

1978年生。1994年大府市立大府西中学校卒業。1997年愛知県立旭丘高校卒業。2001年京都大学工学部情報学科卒業。同年より同大学大学院情報学研究所修士課程に進学。プロセッサ・アーキテクチャの研究に従事。2002年情報処理学会論文賞受賞。



小西 将人(学生会員)

1976年生。1992年綾部市立八田中学卒業。1995年京都府立綾部高校卒業。1999年京都大学工学部情報学科卒業。2001年同大学大学院情報学研究所修士課程修了。同年同研究科博士後期課程に進学。並列計算機アーキテクチャの研究に従事。



中島 康彦(正会員)

1963年生。1986年京都大学工学部情報工学科卒業。1988年同大学大学院修士課程修了。同年富士通(株)入社。スーパーコンピュータVPPシリーズのVLIW型CPU, Mアーキテクチャ・命令エミュレーション, 高速CMOS回路等に関する研究開発に従事。工学博士。1999年京都大学総合情報メディアセンター助手。同年同大学大学院経済学研究所助教授。計算機アーキテクチャに興味を持つ。2002年情報処理学会論文賞受賞。IEEECS, ACM各会員。



森 眞一郎(正会員)

1963年生。1987年熊本大学工学部電子工学科卒業。1989年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了。1992年同大学院総合理工学研究科情報システム学専攻博士課程単位取得退学。同年京都大学工学部助手。1995年同助教授。1998年同大学大学院情報学研究科助教授。工学博士。並列/分散処理, 計算機アーキテクチャの研究に従事。2002年情報処理学会論文賞受賞。IEEE, ACM各会員。



北村 俊明(正会員)

1955年生。1978年京都大学工学部情報工学科卒業。1983年同大学大学院博士課程研究指導認定退学。同年富士通(株)入社。汎用コンピュータ, スーパーコンピュータVPPシリーズのVLIW型CPU, Mアーキテクチャ・命令エミュレーション, 米国HAL社においてSPARCプロセッサ等の研究開発に従事。工学博士。2000年京都大学総合情報メディアセンター助教授。計算機アーキテクチャに興味を持つ。2002年情報処理学会論文賞受賞。電子情報通信学会, IEEE, ACM各会員。



富田 眞治(正会員)

1945年生。1973年京都大学大学院博士課程修了, 工学博士。同年京都大学工学部情報工学教室助手。1978年同助教授。1986年九州大学大学院総合理工学研究科教授。1981年京都大学工学部情報工学科教授。1998年同大学大学院情報学研究科教授。計算機アーキテクチャ, 並列計算機システムに興味を持つ。情報処理学会論文賞受賞(1987年, 1992年, 2002年)。電子情報通信学会, 情報処理学会フェロー。著書「並列計算機構成論」, 「並列処理マシン」, 「コンピュータアーキテクチャI」等。電子情報通信学会, IEEE, ACM各会員。