SIMP（単一命令流/多重命令パイプライン）方式に基づくスーパスカラ・プロセッサ『新風』の性能評価†

久我守弘†† 入江直彦††*
村上和彰†† 富田眞治††

本論文は，SIMP（単一命令流/多重命令パイプライン）方式に基づく試作スーパスカラ・プロセッサ『新風』の性能評価を行っている．SIMP 方式は，均質型スーパスカラ・プロセッサの1実現方式であり，オブジェクト・コードの互換性，および，命令パイプライン数に関する性能のスケーラビリティ保持を目的としている．『新風』の設計に際しては，様々な項目に関して選択肢決定を行った．すなわち，複数命令の供給方法，データ依存および制御依存といった命令間依存関係への対処法，分岐命令への対処法，正確な割込み/分岐の保証方法，命令パイプライン数，などである．このうち，分岐命令への対処法，命令間依存関係への対処法，および，命令パイプライン数が性能に与える影響をシミュレーションにより評価した．『新風』は4本の命令パイプラインを備え，分岐命令への対処法として分岐ターゲット・バッファを用いた動的分岐予測，また，命令間依存関係への対処法として Tomasulo アルゴリズムをそれぞれ採用している．シミュレーションの結果，通常の単一命令パイプライン・プロセッサに比べて約 1.85 倍の性能向上が得られることが判明した．また同時に，いくつかの問題点が明らかになったので，これらボトルネックに関する考察を行っている．

1. はじめに

一般に，単一パイプライン・プロセッサの処理能力は，次式で表現される．

（プログラムの実行時間）
＝（実行すべき命令数）×CPI
×（サイクル・タイム）．

CPI: Cycles Per Instruction.

従来の RISC プロセッサは CPI を1に近づけ，サイクル・タイムを短縮することによって性能向上を図ってきた．しかしながら，現在 RISC プロセッサの CPI がほぼ1に近づきつつあり，将来シリコン技術の限界によってサイクル・タイムの短縮も困難になると考えられている．そこで，命令レベル並列処理により単一プロセッサの処理能力を向上させるアーキテクチャとして，以下の3つの方式が考えられている[12),15)]．

① VLIW (Very Long Instruction Word) 方式：複数の機能ユニット (functional unit) を駆動するための複数のオペレーションフィールドを有する VLIW 命令により，命令レベル並列処理を行う．並列に実行可能なオペレーションの抽出は，コンパイル時に静的コード・スケジューリングにより行う．命令間の依存関係に関してハードウェアは一切関知しないためハードウェア構成が非常に簡単である反面，処理性能が最適化コンパイラの能力にのみ依存するという欠点がある．

② スーパスカラ (Superscalar) 方式：プログラムから複数の命令を同時にフェッチした後，命令解読，オペランド・フェッチ，実行と並列に処理していく．命令間の依存関係に起因するハザードをハードウェアによって回避するとともに，命令間に内在する並列性の抽出をプロセッサ内部で実行時に行う．さらに，必要に応じて命令の順序換えを行う動的コード・スケジューリングを導入する場合もある．

③ スーパパイプライン (Superpipeline) 方式：従来の命令パイプラインの個々のステージをさらに細分化し，パイプラインピッチを短縮することにより高速化を図る．単体プログラムよりも複数プログラムで時分割共有されるような“パイプライン共有型 MIMD”的な用法（マルチスレッド・スーパパイプライン方式）において，多重プログラミングのスループット向上に効果がある．

VLIW およびスーパスカラ方式は，複数の命令ないしオペレーションを同時に発行 (issue) することで，CPI を1以下にすることを狙う．一方，スーパパイプライン方式は，動作周波数を上げることで性能

† Performance Evaluation of the Superscalar Processor【ʃímpu:】Based on the SIMP (Single Instruction stream/Multiple instruction Pipelining) Architecture by MORIHIRO KUGA, NAOHIKO IRIE, KAZUAKI MURAKAMI and SHINJI TOMITA (Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University).

†† 九州大学大学院総合理工学研究科情報システム学専攻
* 現在 日立製作所
Hitachi Ltd.

向上を狙ったものである．しかし，これら3方式の潜在的な性能自体は本質的に等価である[12]．そこで，性能以外の点についてスーパスカラ方式を他の2者と比較する．まず，VLIW 方式とは異なり，従来のパイプライン・プロセッサと命令セットアーキテクチャ・レベルの互換性を保つことができる．さらに，動的コード・スケジューリングにより，実行時にしか判明しない依存関係について対処可能であり，静的コード・スケジューリングの能力を相補うことができるという特長がある．一方，スーパパイプライン方式に対しては，これよりも動作周波数を低く抑えることが可能であるため実現が容易である．また，スーパパイプライン方式が多重プログラミングのスループット向上に効果があるのに対して，スーパスカラ方式は単体プログラムの応答速度向上に効果がある．

以上の点から，筆者らはスーパスカラ方式に着目し，その1実現方式として SIMP (Single Instruction stream/Multiple instruction Pipelining：単一命令流/多重命令パイプライン) 方式を先に提案した[1]．SIMP 方式は，同一機能，構造の命令パイプラインを複数本設けるもので，オブジェクト・コードの互換性，および，命令パイプライン数に関する性能のスケーラビリティ保持を目的とする．筆者らはまた，本方式に基づく試作プロセッサとして，4本の命令パイプラインを備えるスーパスカラ・プロセッサ『新風』を開発してきた[2)-6)]．プロセッサ開発と並行して，『新風』のソフトウェア・シミュレータを作成し，その性能予測を行うとともに，『新風』設計の際に考慮した種々の選択肢についての評価も行ってきた[2),4)]．

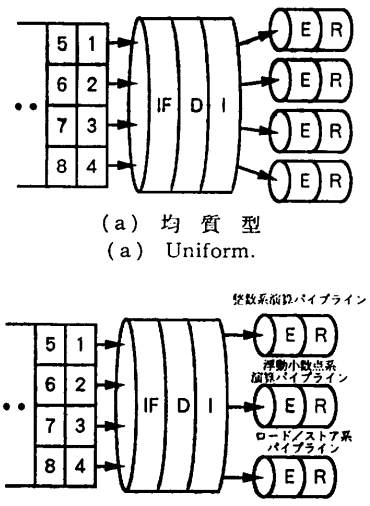
本稿では，まず，『新風』設計上の選択肢について述べた後，3章で『新風』の概要について述べる．続いて，スーパスカラ・プロセッサの構成，制御方式および命令パイプライン本数を変化させたときの処理性能への影響を調べたシミュレーションに関して，4章で評価方法を述べる．5章では，シミュレーション結果の考察を行い，『新風』の問題点を明らかにする．

2. 『新風』設計上の選択肢

スーパスカラ・プロセッサの構成・制御方式には種々の選択肢が存在する．『新風』を設計するにあたり，以下の項目に関する選択肢決定を行った．

2.1 機能ユニットの均質性

スーパスカラ・プロセッサは複数の命令を同時に実行する（この命令数をスーパスカラ度という）．すなわち，複数の命令パイプラインが存在する．このためには，複数のユニットを備える必要があるが，その機能の均質性に関して以下の2種類が存在する（図1参照）．



(a) 均質型
(a) Uniform.

(b) 非均質型
(b) Nonuniform.

IF：命令フェッチ(Instruction Fetch)，D：デコード (Decode)，I：命令発行 (instruction Issue)，E：命令実行 (Execute)，R：格納 (Retire)

図1 機能ユニットの均質性
Fig. 1 Uniformity of the functional units.

① 均質型 (uniform)：すべての機能ユニットの演算機能が同一である．すなわち，すべての命令パイプラインが同一機能，構造である．図1(a)は，スーパスカラ度4の均質型スーパスカラ・プロセッサの例である．

SIMP 方式自身が均質型スーパスカラ・アーキテクチャであり，よって『新風』の機能ユニットは均質である．

② 非均質型 (nonuniform)：各々の機能ユニットの演算機能が均質でなく，命令の種類によりディスパッチされる機能ユニットが決まる．図1(b)は整数/論理演算，浮動小数点数演算，および，ロード/ストアの3つの機能ユニットを持つ例である．これらは同時に動作可能であるが，同一の機能ユニットを使用する命令が複数存在する場合，本質的に競合が発生しうる．

2.2 多重命令供給

スーパスカラ・プロセッサでは，スーパスカラ度に応じて命令キャッシュの命令供給能力を多重化する必要がある．このとき，命令供給多重度 n のスーパスカ

ラ・プロセッサにおいて，n命令バウンダリに揃っていないn個の命令をフェッチする場合，命令の並び換えを行うか否かの選択肢が存在する．命令並び換えを行う場合はさらに，キャッシュのラインクロスを許すか否かの選択肢も存在する[5]．

『新風』では，命令の並び換えは行うが，ラインクロスは許していない．

2.3 命令間依存関係への対処

2.3.1 データ依存

データ依存には，フロー依存，出力依存，および，逆依存の3種類がある．これらに起因するハザードを回避するため，命令発行をブロックする等の制御が必要となる．これには，少なくとも以下の3種類の方式がある[7]．

① インタロック制御：フロー依存，出力依存および逆依存のいずれかが存在すれば，パイプラインをインタロックして命令発行をブロックする．よって，命令の実行開始順序は in-order である．

② Thornton のスコアボード・アルゴリズム：出力依存および逆依存のいずれかが存在すれば，命令発行をブロックする．フロー依存(ソースオペランド不在)に対してはブロックする必要はなく，命令発行可能である．このとき，レジスタに設けた1ビットのスコアボードにより，先行命令からソースオペランドを受け取り，フロー依存が解消され次第実行を開始する．よって，命令の実行開始順序は out-of-order になる．

③ Tomasulo のアルゴリズム：フロー依存，出力依存および逆依存のいずれかが存在しても，命令発行をブロックする必要はない．これは，タグによりレジスタの名前換え (renaming) を行うことにより可能となっている．フロー依存関係にある命令は，タグ一致を検出することで，先行命令からソースオペランドを受け取り，フロー依存が解消され次第実行を開始する．したがって命令の実行開始順序は out-of-order となる．

2.3.2 制御依存

データ依存同様，分岐命令に起因する制御依存の存在も，後続命令の実行を阻害する．特に，out-of-order 実行制御を行う場合，その影響は大きい．制御依存関係により限定される out-of-order 実行可能な命令範囲に関しては，次の3種類の選択肢が存在する．

① lazy execution：分岐命令を越えては out-of-order 実行を行わない．つまり，out-of-order 実行を行う範囲を基本ブロック内に限る．この方式の実現は容易であるが，基本ブロック内の命令数が少ない場合 out-of-order 実行の効果が小さいという欠点がある．

② eager execution：分岐命令を越えて out-of-order 実行を行う．この方式では，分岐予測を行い予測パスの命令に対しても out-of-order 実行を許す．分岐予測が外れた際には，分岐命令以降のすでに実行した命令，および，その実行結果を無効にしなければならない．

③ greedy execution：eager execution 同様，分岐命令を越えて out-of-order 実行を行う．ただし，次のように拡張している；

- 個々の命令の各ソースオペランドについて，複数のフロー依存関係（最大，先行分岐命令数＋1個）を認識しておき，最も先行するフロー依存（最尤フロー依存）が解消され次第，実行を開始する．
- 分岐命令が実行される度に（必要なら）命令の選択的無効化（2.4 節参照）を行う．このとき，制御依存の解消に伴い，新たな最尤フロー依存が発生する可能性がある．その場合，命令の再実行（バックトラック）を行う．

詳細は文献 3) を参照されたい（ただし，“条件付き先行実行”と呼んでいる）．

2.3.3 選択肢の組合せ

2.3.1 項および 2.3.2 項で述べた選択肢をまとめると，以下の7種類の組合せが可能となる．

① インタロック＋lazy execution
② インタロック＋eager execution
③ Thornton＋lazy execution
④ Thornton＋eager execution
⑤ Tomasulo＋lazy execution
⑥ Tomasulo＋eager execution
⑦ Tomasulo＋greedy execution

上記7種類の方式は，後者ほど out-of-order 実行の自由度が大きくなるが，それだけ構成が複雑となる．したがって，性能およびハードウェアコストとのトレードオフで選択を行う必要がある．

『新風』では，⑦を採用している．

2.4 分岐命令への対処

分岐命令は，2.3.2 項で述べたように命令実行を阻害するのみならず，命令フェッチをも阻害する．すなわち，分岐するか否か（条件分岐の場合），および，分岐先アドレスが確定するまで次にフェッチすべき命令が決まらない．このような分岐命令に起因する制御

依存への対処法には種々の方式があるが[5]，最も一般的な方式の1つに分岐予測がある．

一般に分岐予測を行う場合，その予測が外れた際，誤って命令パイプラインに投入した命令を無効化してパイプラインを復元する必要がある．この復元処理の方法には，次の2つの選択肢が存在する．

① パイプライン・フラッシュ (pipeline flush)：分岐予測を外した分岐命令以降のすべての命令を単に無効化する．そして，当該分岐命令の分岐結果に従って，正しい命令を再フェッチする．

② 選択的命令無効化 (selective instruction squashing)[5]：無効化すべき命令を選択し，それのみを無効化する．もし有効な命令が残らなかった場合にのみ，命令の再フェッチを行う．本方式は，スーパスカラ・プロセッサにおいて効果がある．これは，分岐命令以降に命令パイプラインに投入されうる命令の数が多く，分岐予測がはずれていても正しい命令が含まれている可能性があるからである．さらに，greedy execution においては，本方式の採用が前提である．

『新風』では，分岐ターゲット・バッファ方式の動的分岐予測を行う．また，greedy execution を行っているので，復元処理として選択的命令無効化を採用している．

2.5 正確な割込み/分岐の保証

スーパスカラ・プロセッサに限らず，一般に命令実行終了順序の out-of-order を許す場合，後続命令が先行命令を追い越してレジスタおよびメモリ内容を更新する可能性がある．このとき，内部割込みが発生し，当該割込みを引き起こした命令の後続命令がすでにレジスタ内容等を更新していると，適切な割込み処理，および，割込み処理後のプログラム再開ができない場合がある．このような割込みを“不正確な割込み (imprecise interrupt)”と呼ぶ．IBM 360/91 のような仮想記憶をサポートしていないプロセッサでは不正確な割込みを許容できたが，仮想記憶をサポートするプロセッサにおいては，少なくともページフォルトは正確な割込みでなければならない．

また，eager および greedy execution を行うプロセッサにおいては，分岐予測パス中の命令が対応する分岐命令に起因する制御依存の解消を待たずに，レジスタ内容等を更新してしまう危険性がある．このような不正確なマシン状態を防ぐ方法として，次のような選択肢が可能である．

① conditional mode：分岐予測パス中の命令は，対応する分岐命令に起因する制御依存が解消されるまで，レジスタ内容等を更新しない．

② reorder buffer[9),10)]：命令の実行終了順序は out-of-order だが，レジスタ内容等の更新順序は in-order とする．この reordering のために，レジスタ等への格納データを一旦バッファリングする．更新条件は conditional mode と同様である．

③ checkpoint repair[8)]：マシン状態をある定まったチェックポイント時点で保存しておき，内部割込みあるいは分岐予測の外れが生じた際には，最新の保存マシン状態を元にパイプラインを復元する．

④ in-order enforce[14)]：内部割込みを引き起こす可能性のある命令が存在する場合は，out-of-order 実行を禁止して in-order とする．

『新風』では，reorder buffer を用いている．

2.6 選択肢の決定

以上 2.1〜2.5 節で述べた選択肢について，『新風』を**表1**にまとめる．また，比較のため，商用スーパスカラ・プロセッサ Intel 80960 CA[13)]，および，IBM RS/6000[14)] の諸元も表1に示す．

3. 『新風』の概要

3.1 『新風』の構成

『新風』は均質型スーパスカラ・プロセッサであり，4本の同一の命令パイプラインを有する．各命令パイプラインは，命令ブロックフェッチ (IF)，命令解読 (D)，命令発行 (I)，実行 (E)，および，リタイア (R) の5ステージから成る (**図2**)．プロセッサは以下の主要ユニットから構成される (**図3**)．各ユニットの詳細については，文献3)，5) を参照されたい．

① マルチバンク命令キャッシュ (MBIC)
② 命令ブロック供給ユニット (IBSU)
③ 命令パイプライン・ユニット (IPUs)
④ 依存解析機能付きレジスタファイル (DHRF)
⑤ マルチポート・データキャッシュ (MPDC)
⑥ 命令パイプライン・チェイニング網 (IPCN)

3.2 命令パイプライン処理過程

命令パイプラインの各ステージについて，その処理の概要を以下に示す．I，EおよびRステージにおける out-of-order 実行制御の詳細については，文献3) を参照されたい．

(1) IF ステージ

命令ブロック供給ユニット (IBSU) は，マルチバンク命令キャッシュ (MBIC) から連続する4個の命

表 1 『新風』および商用スーパスカラ・プロセッサの諸元
Table 1 Specifications of several superscalar processors.

選択肢・諸元		プロセッサ		
		『新 風』	i 80960 CA	IBM RS/6000
命令供給	命令供給多重度	4	4*1	4
	命令アラインメント	サポート	サポート	サポート
	ラインクロス	不可	可	可
	ラインサイズ	16命令	8命令	16命令
機能ユニット	均質性	均質	非均質	非均質
	種類・数	①汎用×4 • 整数加減算，ロード/ストア，分岐 • 整数乗算 • 浮動小数点加減算 • 浮動小数点乗算	①整数加減乗除算×1 ②ロード/ストア×1 ③分岐，制御×1	①整数加減乗除算，ロード/ストア×1 ②浮動小数点加減乗除算×1 ③分岐×1 ④制御×1
	命令発行多重度	4	3	4
依存関係対策	データ依存	Tomasulo アルゴリズム	インタロック制御	インタロック制御*2
	制御依存	greedy	eager	eager
命令実行制御	実行開始順序	out-of-order	in-order*3	in-order
	実行終了順序	out-of-order	out-of-order	out-of-order
分岐命令対策	分岐対処	動的分岐予測 (分岐ターゲット・バッファ)	静的分岐予測	動的分岐予測 (not-taken 予測)
	復元処理	選択的無効化	パイプライン・フラッシュ	パイプライン・フラッシュ
正確な割込み/分岐	割込みの正確/不正確	正確	不正確*4	正確
	正確な割込み機構	reorder buffer	—	• in-order enforce • checkpoint repair*5
	正確な分岐機構	reorder buffer	conditional mode	conditional mode

*1 先頭命令が偶数アドレスの場合4命令，奇数アドレスの場合3命令.
*2 浮動小数点演算命令と浮動小数点レジスタ・ロード命令間の逆依存および出力依存関係は register renaming により解消する.
*3 命令の組合せ次第では，out-of-order となる場合もある.
*4 仮想記憶未サポート．ただし，命令の逐次実行により，正確な割込みを保証する動作モードがある.
*5 制御レジスタ用.

令を命令ブロックとしてフェッチし，各命令パイプライン・ユニット (IPU) のDステージに命令を投入する．命令ブロックをフェッチする際，命令ブロックの先頭アドレスが4命令バウンダリでない場合（命令ミスアラインメント）は，命令の並び換えを行い無駄な no-op が生じないようにする．ただし，命令ブロックがキャッシュラインを跨ぐような場合には，次ライン分の命令を no-op とする．つまり，ラインクロスを許さない．次にフェッチすべき命令ブロックの決定は分岐ターゲット・バッファ (BTB) 方式の動的分岐予測によって行う．フェッチした命令ブロック内の分岐命令が過去において BTB に登録されているか否かを調べ，登録されている場合には予測アルゴリズムに従って予測パスを決定する．登録されていない場合には分岐しないと予測する．分岐予測が外れた際には，選択的命令無効化によりパイプライン復元処理を行う．IF ステージの詳細な動作は，文献 5) を参照されたい．

(2) Dステージ

投入された命令のデコードを各 IPU で同時かつ独立に行う．命令内にある種々の命令フィールドを解読し，解読結果（操作の種類，オペランドの種類など）をレジスタ・ファイル (DHRF) へ送る．

(3) Iステージ

Dステージでの解読結果に従って，命令間の依存関

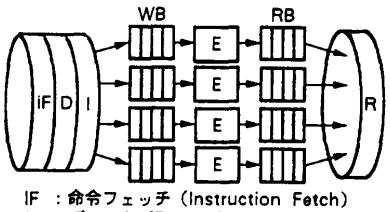


図 2 評価モデル (スーパスカラ度4)
Fig. 2 Simulation model.

係を DHRF において解析する．そして，ソースレジスタの現在値をレジスタファイルから読み出し，データおよび制御依存関係を表す実行制御情報を付加して，Eステージに対し命令の“発行”を行う．実行制御情報は複数先行命令との間のフロー依存関係および制御依存関係を表現するもので，greedy execution による out-of-order 実行を可能としている．

以上の IF, D, I ステージは命令ブロック単位で in-order に処理される．

(4) Eステージ

各 IPU は，整数 ALU (IALU：ロード/ストア，分岐も含む)，整数乗算器 (IMUL)，浮動小数点 ALU (FALU)，浮動小数点乗算器 (FMUL) の4個の機能ユニットを持つ．各機能ユニットはさらにパイプライン化されており，Eステージ全体としては並列演算パイプラインとして動作する．また，機能ユニットの前段にはオペランドの待合せを行う供給バッファ (WB)，後段には reorder を行う格納バッファ (RB) を設ける．WB および RB はそれぞれ4段のキューであり，各段は WB と RB が対になって動作する．よって，両者をまとめて，供給格納バッファ (WRB) と呼ぶ．WRB は out-of-order 実行および正確な割込み/分岐を保証するために設けたバッファである．WB

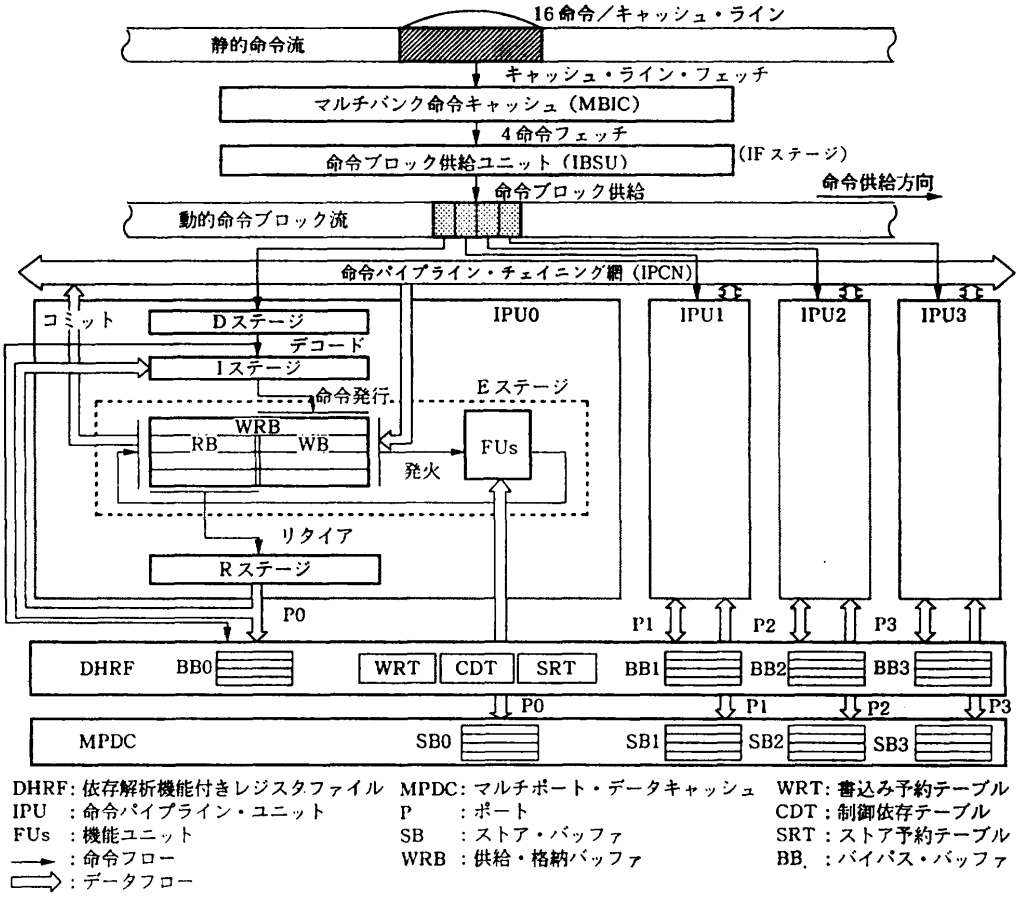


図 3 『新風』の構成
Fig. 3 Block diagram of the【ʃímpu:】processor.

は IBM 360/91 における reservation station[7] に，また，RB は文献 9) における reorder buffer に相当する．Iステージから発行された命令は，まず機能ユニットの前後に設けた供給格納バッファ（WRB）に登録される．WRB 中の命令は，Iステージで作成された実行制御情報に基づいて out-of-order 実行制御される．すなわち，最尤フロー依存が解消され次第命令実行が“発火”され，その実行結果は WRB に一旦格納される．そして，その命令に関するすべての依存関係が解消され次第，当該演算結果は後続命令のソースオペランドとして使用できるようにチェイニング網（IPCN）経由で全パイプライン・ユニット（IPU）およびレジスタ・ファイル（DHRF）に送られる．これを“コミット”と呼ぶ．発火およびコミット順序はいずれも out-of-order である．

（5） Rステージ

命令ブロックを構成する4個の命令がすべてコミットされたら，レジスタ・ファイル（DHRF）およびマルチポート・データキャッシュ（MPDC）において実行結果の格納を行い，その命令ブロックを命令パイプラインから“リタイア”させる．リタイアの順序は命令ブロック単位で in-order である．

3.3 演算遅延

命令パイプラインは，2サイクル/ステージの2サイクル・パイプラインである．一方，Eステージの演算パイプラインは，1サイクル・パイプラインとして動作する．各命令の演算遅延（operation latency）は，以下のとおりである；

- 整数加減算/論理演算：1サイクル．
- 整数乗算：3サイクル．
- 浮動小数点加減算（単精度）：4サイクル．
- 浮動小数点乗算（単精度）：5サイクル．
- ロード/ストア：4サイクル．
- 分岐：3サイクル．

さらに，上記の値に，WB における発火遅延1サイクル，および，RB におけるコミット遅延1サイクルを加えた値が，Eステージ所要時間となる．

4. 性能評価方法

4.1 シミュレータ

性能評価は，ソフトウェア・シミュレーションにより行った．本シミュレータは，『新風』の機械語プログラムを入力として，個々の命令のパイプライン処理過程を忠実にシミュレーションする[2]．本シミュレータは，『新風』プロセッサだけでなく，種々の構成，制御方式ないしスーパスカラ度を採るスーパスカラ・プロセッサをシミュレーション可能としている[4]．

4.2 評価項目

2章で述べた選択肢のうち，以下の項目に関して評価を行う．

（1） 分岐命令への対処（2.4 節参照）：

- 分岐予測：BTB による動的分岐予測を行う vs. 行わない．
- 復元処理：パイプライン・フラッシュ vs. 選択的命令無効化．

（2） スーパスカラ度：1，2，4，および，8．

（3） 命令間依存関係への対処（2.3 節参照）：2.3.3 項で述べた以下の7種類の組合せ．

① インタロック＋lazy execution
② インタロック＋eager execution
③ Thornton＋lazy execution
④ Thornton＋eager execution
⑤ Tomasulo＋lazy execution
⑥ Tomasulo＋eager execution
⑦ Tomasulo＋greedy execution

（4） 供給格納バッファ（WRB）コスト：WRB における発火およびコミット遅延が発生しない場合 vs. 発生する（1サイクルずつ）場合．

上記以外の選択肢に関しては，表1に示したとおりである．なお，命令/データ・キャッシュのヒット率は 100% を仮定した．

4.3 ベンチマーク・プログラム

bsort（バブルソート），div（引き戻し法による除算），max（最大値検索），sieve（素数計算），liv 5〜7（リバモアループ #5〜7：単精度），vsum（配列要素の加算：単精度）の8種類のプログラムを用いる．いずれも，ループ再構成，静的コード・スケジューリング等による最適化は施していない．

5. 性能評価結果

5.1 シミュレーション結果

図4および**図5**にシミュレーション結果を示す．これらは，以下の評価項目と性能向上比との関係を表す；

- 分岐命令への対処
- 命令間依存関係への対処
- スーパスカラ度
- 供給格納バッファ（WRB）コストの有無

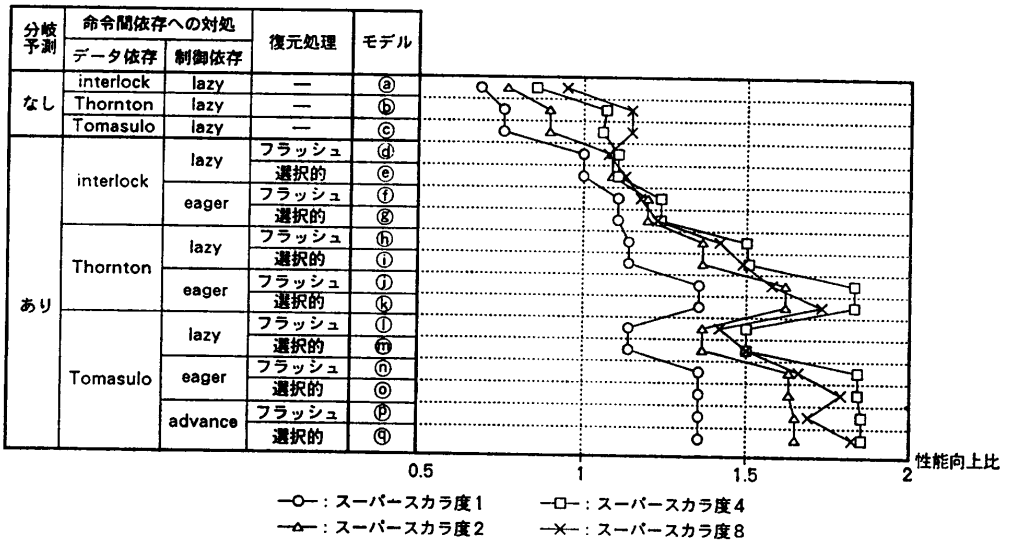


図 4 性能向上比 (WRB コストを加味した場合)
Fig. 4 Speedups (incl. 1-cycle cost for WRB).

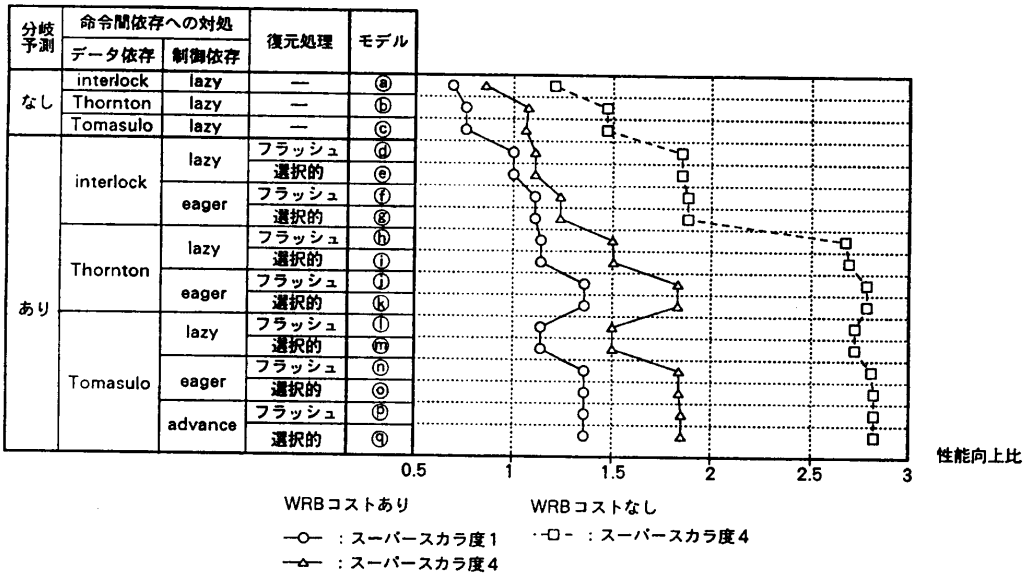


図 5 性能向上比 (WRB コストを加味しない場合)
Fig. 5 Speedups (excl. 1-cycle cost for WRB).

性能向上比は，標準的な単一パイプライン・プロセッサであるスーパスカラ度1のモデルⓓを基準とした値で，全ベンチマーク・プログラムの調和平均値である．

『新風』は，スーパスカラ度4のモデルⓠに相当する．図4より，単一パイプライン・プロセッサに比べて約 1.85 倍の性能向上が可能である．

5.2 評価項目に関する解析

(1) 分岐命令への対処 (図4参照)

まず，分岐予測の効果について見てみると，分岐予測を行う場合 (ⓓ～ⓠ) と行わない場合 (ⓐ～ⓒ) とでは，分岐予測を行う方が常に高い性能向上比を得ている．分岐予測ヒット率自体は 90～98% の範囲にあり，これは文献 5) の分岐予測ヒット率 90% に一致している．ちなみに，文献 11) によると，分岐予測ヒット率 100% に比べてヒット率 85% の場合，10～45% 程度性能向上比が低下する．次に，分岐予測が外れた際の復元処理方式について見てみると，パイプライン・フラッシュ (ⓓ, ⓕ, ⓗ, ⓙ, ⓛ, ⓝ, ⓟ)

と選択的命令無効化（ⓔ, ⓚ, ⓘ, ⓚ, ⓜ, ⓞ, ⓠ）とでは，ほとんど性能に差がないことがわかる．これは選択的無効化が，ほとんどのベンチマークにおいて，1万命令実行しても数回から数十回程度しか発生しないため，その効果が現れないからである．

以下，分岐予測を行う場合（ⓓ〜ⓠ）を解析の対象とする．

（2） 命令間依存関係への対処（図4参照）

まずデータ依存への対処法を見ると，スーパスカラ度4において，インタロック制御（ⓓ, ⓔ, ⓕ, ⓖ）の性能向上比は約 1.1〜1.2 であるが，インタロック制御以外（ⓗ〜ⓠ）の性能向上比は 1.5〜1.9 である．つまり，後者は前者に比べて約 25〜70% 程度性能がよい．文献 11) には，20〜65% 程度の差が見られる．

しかし，インタロック制御以外の方式（ⓗ〜ⓠ）に関して，Thornton のアルゴリズム（ⓗ〜ⓚ）と Tomasulo のアルゴリズム（ⓘ〜ⓠ）を比較してみても，その差はほとんどない．一方，文献 7) は，Cray-1 スカラ・ユニットにおいて，Tomasulo のアルゴリズムは Thornton のアルゴリズムに比べて 20% 程度性能がよいと報告している．この違いはレジスタ数の違い（『新風』は汎用および浮動小数点レジスタが各 48 個，Cray-1 はアドレスおよびスカラ・レジスタが各 8 個），すなわち，逆依存と出力依存の発生頻度の違いに起因するものと考えられる．

次に，インタロック制御以外の方式（ⓗ〜ⓠ）に関して，制御依存への対処法の効果をスーパスカラ度4において見てみる．このとき，データ依存解消アルゴリズムに依らず，lazy execution（ⓗ, ⓘ, ⓛ, ⓜ）の性能向上比は約 1.5，eager/greedy execution（ⓙ, ⓚ, ⓝ, ⓞ, ⓟ, ⓠ）の性能向上比は約 1.8 となっている．つまり，後者は前者に比べて 20% 程度性能がよい．しかし，eager execution（ⓝ, ⓞ）と greedy execution（ⓟ, ⓠ）との差はほとんどない．これは，greedy execution におけるコミット条件に問題があったためと考えている．すなわち，設定したコミット条件が，フロー依存関係にある先行命令から後続命令へのデータ・バイパスを阻害し，後続命令の先行実行の効果が得られなかったためである．

（3） スーパスカラ度（図4参照）

スーパスカラ度に関しては，いずれのモデル（ⓐ〜ⓠ）においてもスーパスカラ度4でほぼ飽和することがわかる．同様の傾向が，文献 12) でも報告されている．これは，ベンチマーク・プログラムに内在する並列度が性能を決める支配的な要因であり，その平均並列度が2〜3程度しかないことに起因する．

また，図4ではスーパスカラ度8の場合の方が4の場合よりも性能が低下しているが，これは供給格納バッファ（WRB）の段数とリタイア処理に原因がある．スーパスカラ度が8にもなると，実行終了が遅い命令によってリタイアが妨げられ，4段の WRB はすぐにオーバフローを起こしてパイプラインがインタロックしてしまい，命令の並列実行が妨げられたためである．

5.3 ボトルネックの解明

スーパスカラ・プロセッサの性能向上比を評価したものには，文献 11), 12) 等がある．今回われわれが得た『新風』の予測性能向上比は，これらに比べて低い．その原因として，以下のボトルネックの存在が明らかになっている．

① リタイア・ボトルネック：正確な割込み/分岐を保証するために，格納バッファ（RB）を用いてレジスタ内容等の更新を in-order に行う．このレジスタ内容等の更新処理は命令パイプラインの最終段のR（リタイア）ステージに相当し，これをもって命令は命令パイプラインから抜け出ると同時に供給格納バッファ（WRB）を解放する．しかしながら，キュー動作を行う WRB の獲得・解放を容易に制御するため，リタイアは命令ブロック単位という制限が付く．よって，命令ブロック中に他よりも実行の遅い命令が存在すると WRB がなかなか解放されず，結局 WRB が一杯になってI（発行）ステージがインタロックされることになる．

② コミット・ボトルネック：greedy execution におけるコミット条件[4]が，フロー依存関係にある先行命令から後続命令へのデータ・バイパスを阻害している．通常の eager execution においては，先行命令はその実行結果を直ちに後続命令にバイパスできる．しかし，greedy execution においては，制御依存が解消されてからでないとバイパスできない．分岐命令が早期実行される場合は問題ないが，3.3 節で述べたように，『新風』では分岐命令の演算遅延が3サイクルもある．なお，今回の性能評価においては，eager execution にも greedy execution と同じコミット条件を適用している．よって，実際には eager execution の方が性能がよいと考えられる．

③ 供給格納バッファ（WRB）ボトルネック：図4

の結果は，WB における発火遅延1サイクル，および，RB におけるコミット遅延1サイクルを演算遅延に加味した場合の値である．WRB コストを無視したシミュレーションを行ったところ，図5に示すように，スーパスカラ度4の場合で 50〜80% 程度の性能向上が認められた．これから，発火遅延およびコミット遅延が演算遅延に隠蔽可能な場合，同程度の性能向上が期待できる．

④ 分岐ボトルネック：②でも述べたように，分岐命令の早期実行の可否が性能を左右する．『新風』では，その条件分岐方式として先行条件決定（advanced conditioning）方式[6]を採用しているので，分岐するか否かは早期に決定可能である．しかし，選択的命令無効化を採用しているので，分岐先アドレスを生成し無効化すべき命令を決めるまで，制御依存が解消されない．結局，これに3サイクルかかることになる．

6. まとめと今後

以上，1987年度より開発を行ってきたスーパスカラ・プロセッサ『新風』の性能評価を行った．その結果，単一パイプライン・プロセッサに比べて，約 1.85 倍の性能向上が可能であることが判明した．しかしながら，この性能向上比は決して満足のいく値ではない．

5.2 節および5.3節で行った考察に基づき，以下の高速化および軽量化を施すことが今後の課題として残る．

- 命令間依存関係への対処：最適化コンパイラによる静的コード・スケジューリングを活用することで，実行制御アルゴリズムの簡素化を図る．つまり，Tomasulo+greedy execution→インタロック+eager execution.
- パイプライン復元処理：選択的命令無効化→パイプライン・フラッシュ．
- 正確な割込み機構：reorder buffer→不正確な割込みである（つまり，レジスタ内容等の out-of-order 更新を許す）が，プログラム再開を可能とする新割込みアーキテクチャ．
- 正確な分岐機構：reorder buffer→conditional mode.

謝辞 現在筆者らとともに設計・開発を行っている原 哲也，納富 昭の両氏，および，日頃御討論いただく富田研究室の諸兄に感謝いたします．

なお，本研究は一部文部省科研費試験研究による．

参考文献

1) Murakami, K. et al.: SIMP (Single Instruction stream/Multiple instruction Pipelining): A Novel High-Speed Single-Processor Architecture, *Proc. 16th Ann. Int. Sym. Computer Architecture*, pp. 78-85 (1989).
2) 入江ほか：SIMP（単一命令流/多重命令パイプライン）方式に基づく『新風』プロセッサの高速化技法および性能予測，情報処理学会研究報告，88-ARC-73-11 (1988).
3) 久我ほか：SIMP（単一命令流/多重命令パイプライン）方式に基づく『新風』プロセッサの低レベル並列処理アルゴリズム，情報処理学会論文誌，Vol. 30, No. 12, pp. 1603-1611 (1989).
4) 久我ほか：SIMP（単一命令流/多重命令パイプライン）方式に基づくスーパースカラ・プロセッサ『新風』の性能評価，並列処理シンポジウム JSPP '90 論文集，pp. 337-344 (1990).
5) 原ほか：SIMP（単一命令流/多重命令パイプライン）方式に基づくスーパースカラ・プロセッサ『新風』の命令供給機構，情報処理学会研究報告，90-ARC-80-7 (1990).
6) 原ほか：『新風』プロセッサの条件分岐方式，第 40 回情報処理学会全国大会論文集，7L-6 (1990).
7) Weiss, S. et al.: Instruction Issue Logic in Pipelined Supercomputers, *IEEE Trans. Comput.*, Vol. C-33, No. 11, pp. 1013-1022 (1984).
8) Hwu, W. W. et al.: Checkpoint Repair for High-Performance Out-of-Order Execution Machines, *IEEE Trans. Comput.*, Vol. C-36, No. 12, pp. 1496-1514 (1987).
9) Sohi, G. S. et al.: Instruction Issue Logic for High-Performance, Interruptable Pipelined Processors, *Proc. 14th Ann. Int. Sym. Computer Architecture*, pp. 27-34 (1987).
10) Smith, J. E. et al.: Implementing Precise Interrupts in Pipelined Processors, *IEEE Trans. Comput.*, Vol. 37, No. 5, pp. 562-573 (1988).
11) Smith, M. D. et al.: Limits on Multiple Instruction Issue, *Proc. ASPLOS-III*, pp. 290-302 (1989).
12) Jouppi, N. P.: The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance, *IEEE Trans. Comput.*, Vol. 38, No. 12, pp. 1645-1658 (1989).
13) Intel Corp.: 80960 CA User's Manual, Santa Clara, CA (1989).
14) IBM Corp.: IBM RISC System/6000 Technology, Austin, TX (1990).

15) 横田：次世代 RISC，並列処理を導入し CMOS で 100 MIPS ねらう，日経エレクトロニクス，No. 487, pp. 191-200 (1989).

(平成2年8月20日受付)
(平成3年3月 4 日採録)

久我 守弘（正会員）

1965年生．1987年福岡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．現在，同大学院博士課程に在学中．並列処理，計算機アーキテクチャの研究に従事．

入江 直彦（正会員）

1965年生．1988年九州大学工学部電子工学科卒業．1990年同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．この間，並列処理，最適化コンパイラの研究に従事．現在(株)日立製作所中央研究所に勤務．計算機アーキテクチャの研究に従事．

村上 和彰（正会員）

1960年生．1982年京都大学工学部情報工学科卒業．1984年同大学院工学研究科情報工学専攻修士課程修了．同年富士通(株)本体事業部に入社．主として汎用計算機Mシリーズのアーキテクチャ開発に従事．1987年九州大学工学部助手，1988年同大学院総合理工学研究科情報システム学専攻助手，現在に至る．計算機アーキテクチャ，コンパイラ，並列処理等の研究に従事．著書「計算機システム工学（共著，昭晃堂）」．電子情報通信学会，日本応用数理学会，ACM，IEEE，IEEE-CS 各会員．

富田 眞治（正会員）

1945年生．1968年京都大学工学部電子工学科卒業．1973年同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，1991年京都大学工学部情報工学科教授，現在に至る．計算機アーキテクチャ，並列処理システムなどに興味を持つ．著書「並列計算機構成論」「計算機システム工学」「並列処理マシン」など．電子情報通信学会，IEEE，ACM 各会員．