計算機アーキテクチャ 105－7
ハイパフォーマンス
コンピューティング 50－7
（1994．3．10）

ハイパースカラ・プロセッサ・アーキテクチャ
— ハイパフォーマンス・プロトタイプ・プロセッサの設計および予備性能評価 —

宮嶋浩志† 斎藤靖彦 弘中哲夫† 村上和彰

九州大学 大学院総合理工学研究科 情報システム学専攻

〒816 福岡県春日市春日公園 6-1

†: 九州大学 工学部 情報工学科

E-mail: {miyajima, saitoh, hironaka, murakami}@is.kyushu-u.ac.jp

ハイパースカラ方式を採用した，ハイパフォーマンス・プロトタイプ・プロセッサの設計を行った．ハイパースカラ・プロセッサでは，内部の命令レジスタに通常のスカラ命令をロードすることで，VLIWプログラムを自己形成し，それを実行することでループに内在する命令レベル並列性を活用する．プロトタイプではハイパースカラ・プロセッサの実現可能性を示すことを目的とし，また，ハードウェア・コストを極力抑えるためベクトルレジスタの採用を見送り，スカラレジスタのみを採用した．本稿では，プロトタイプの設計仕様について述べる．さらに，プロトタイプを基本としたモデルに対する予備性能評価を行った結果を示している．

Hyperscalar Processor Architecture
— Design and Performance of High-Performance-Prototype Processor —

Hiroshi Miyajima† Yasuhiko Saitoh Tetsuo Hironaka† Kazuaki Murakami

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
Kasuga-shi, Fukuoka 816 Japan

†: Department of Computer Science and Communication Engineering
Faculty of Engineering
Kyushu University

E-mail: {miyajima, saitoh, hironaka, murakami}@is.kyushu-u.ac.jp

We designed the Hyperscalar processor prototype. Hyperscalar processor architecture exploit instruction-level parallelism by executing VLIW instructions, which are self-created by loading several conventional scalar instructions to its instruction registers. The purpose of designing is to substitute the efficiency of Hyperscalar processor architecture. In this paper, we present the specification of the prototype, and show the results of the performance evaluation.

1 はじめに

ハイパースカラ・プロセッサ (*hyperscalar processor*) とはスーパースカラ・プロセッサ, VLIW プロセッサ, および, ベクトル・プロセッサのそれぞれの長所を含有するアーキテクチャである [1].

ハイパースカラ・プロセッサ・アーキテクチャとは, 簡約すれば,

- 命令長および命令フェッチ巾はスーパースカラ・プロセッサ, または, 通常のパイプライン・プロセッサと同程度だが,
- 機能ユニット対応に (1 個以上の) ユーザ可視の命令レジスタを設け, それに (解読済みの) 命令をロードすることで VLIW プログラムをプロセッサ内部に自己形成し, あたかも VLIW プロセッサの如く振舞い,
- さらに, 自己形成した VLIW 命令のループにより, ベクトル・データに対して**擬似ベクトル処理** (ベクトル命令の処理内容をスカラ/VLIW 命令のループで模擬する) あるいは**ソフトウェア・パイプライニング処理**を施す,

ことを目的としたプロセッサ・アーキテクチャである.

このような構成をとることにより, ハイパースカラ・プロセッサはスーパースカラ・プロセッサが得意とする命令レベル並列度がさほど高くない非科学技術計算分野, VLIW プロセッサおよびベクトル・プロセッサが得意とする命令レベル並列度が非常に高い科学技術分野までを対象アプリケーションとするプロセッサ・アーキテクチャである.

しかしながら, このハイパースカラ・プロセッサを実現するに際して, 解決しなくてはならない課題がいくつか残されている. 主な課題として,

- ハイパースカラ・プロセッサに最適なコンパイレーション手法の確立.
- ハイパースカラ・プロセッサのアーキテクチャ構成のバリエーションと, それらが性能の対して与える影響の評価.

がある.

ハイパフォーマンス・プロトタイプ・プロセッサ (以後, プロトタイプ) は,

- ハイパースカラ・プロセッサの実現可能性を示す.
- 設計上の課題を明確にする.

ことを主な目的としている. 現在は, CAD(Computer Aided Design) ツール PARTHENON[4] を用いてアーキテクチャ記述を行っている.

本稿では, プロトタイプの設計仕様について述べ, その予備性能評価の結果を示す. まず, 2章でハイパースカラ・プロセッサ・アーキテクチャの概要について述べる.3章でプロトタイプの基本仕様について述べ,4章ではプロトタイプの評価, および, 他の命令レベル並列処理方式との比較を行う. 最後に 5章で本稿をまとめる.

2 ハイパースカラ方式

図 1にハイパースカラ・プロセッサの基本構成を示す. 以下, ハイパースカラ・プロセッサの概要について述べる.

ハイパースカラ・プロセッサの構成は基本的にはスーパスカラ・プロセッサと変わらない. 命令長および, 命令フェッチ巾は 1 またはスーパースカラ・プロセッサ等と同程度とする. 例えば, 命令長は 32 ビットで命令フェッチ巾 1〜4 命令程度で構わない. これにより, VLIW 方式の短所であるコード・サイズの増加および命令キャッシュの低使用効率といった問題を解決する.

ハイパースカラ・プロセッサのスーパースカラ・プロセッサとの間の本質的な相違点は, 並列動作可能な各々の機能ユニット (FU) 毎に 1 個以上のユーザ可視の命令レジスタ (IR) を設けた点である. 機能ユニット数を f(図 1の例の場合 $f = 7$), 各機能ユニット当たりの命令レジスタ数を rとすると, 計 $f \times r$個 (図 1の例の場合 $7 \times r$個) の命令レジスタが存在する. 命令レジスタが構成するアドレス空間は $f \times r$の 2 次元配列となる. このとき, IRの各行はあたかも, f個のフィールドからなる 1 個の VLIW 命令のように見える. また, 命令レジスタ全体では, r命令から成る 1 個の VLIW プログラムのように見える.

ハイパースカラ・プロセッサは通常動作時 (**Normal** モード) はプロセッサ内にある $f \times r$個の命令レジスタを用いず, スーパスカラ・プロセッサまたは, 通常のスカラ・プロセッサとして動作する. そして, 高い並列度を持つループなどを実行する時は, 通常の命令フェッチをやめ $f \times r$個の命令レジスタにロードされた命令をあたかも r個の VLIW 命令からなるループであるかのように f個の機能ユニットを用いて実行する (**Turbo** モード) プロセッサである.

Turbo モードで実行する $f \times r$個の命令レジスタの命令をディスパッチする方法として次の 2 つの方法が考えられる.

- **専念ディスパッチ**：IRディスパッチ開始命令によりデータ・キャッシュからの命令ディスパッチを開始し, IR ディスパッチ終了条件を満たすまで命令レジスタへの命令ディスパッチを続ける. つまり, メモリ上のデータをレジスタへロードする場合と同等の方法で処理される.
- **並行ディスパッチ**：通常の命令フェッチと並行して, 命令キャッシュから命令レジスタへディスパッチする命令をロードする. **Normal** モードで実行される命令をそのまま命令レジスタにディスパッチすることで実現される.

3 プロトタイプの基本仕様

3.1 検討項目

プロトタイプでは,

- ハイパースカラ・プロセッサの実現可能性を示す.
- 設計上の課題を明確にする.

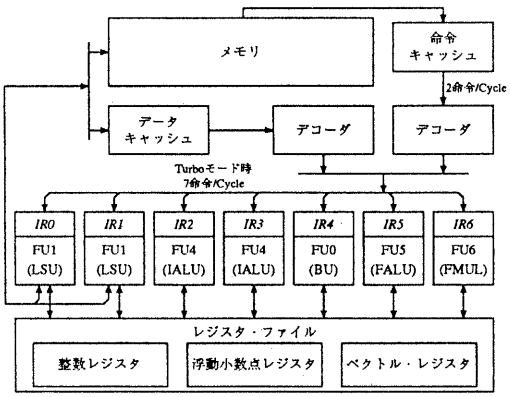
ことを主な目的としている. また, 現在の最先端の VLSI プロセッサの回路規模が 300 万トランジスタ程度であることを踏まえ, 同程度以下の回路規模を目標に設計を行う.

設計にあたって, 以下の点について, あらかじめ検討する.

- アドレス計算の高速化への対処法
- 機能ユニット構成と必要な回路規模

3.1.1 アドレス計算の高速化

我々は文献 [3] において, ソフトウェア・パイプライニングに関する性能評価を行った. 評価結果より,gcc の生成したオブジェクト・コードがアドレス計算に必要以上に整



FU　：機能ユニット　　BU　：分岐ユニット
IR　：命令レジスタ　　FALU：浮動小数点加減算ユニット
LSU：ロード/ストア・ユニット　FMUL：浮動小数点乗除算ユニット
IALU：整数加減算ユニット

図 1: ハイパースカラ・プロセッサの基本構成

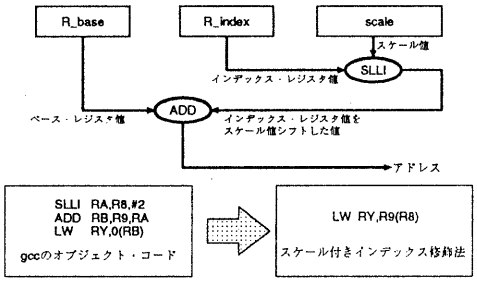


図 2: スケール付きインデックス修飾

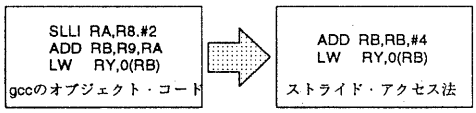


図 3: ストライド・アクセス法

数演算命令を用いているため，整数演算命令がループ・ボディの大きさ (以下 *III(Iteration Initiation Intarval)*[3] を決定していることがわかった．整数演算命令を削減することができれば，より高い性能を得ることができる．

アドレス計算を高速化するためには，以下の選択肢が考えられる．

①**整数 ALU の追加**：整数 ALU を追加すると，レジスタ・ファイルのポート数がソース，デスティネーション合計で 3 本増加する．ポート数の増加，および，整数 ALU の追加により，ハードウェア・コストが増加する．

②**アドレス計算手法の改善**：アドレス計算命令数を削減する方法として以下に示す方法が考えられる．

(a)ロード／ストア命令のアドレシング・モードにスケール付きインデックス修飾を導入する．(ハードウェアによる対処)

(b)ストライド・アクセス法を採用する．(ソフトウェアによる対処)

アドレス計算手法の改善に関して，2 つの方法を比較検討する．

(a)**スケール付きインデックス修飾**：スケール付きインデックス修飾アドレシング・モードをロード/ストア命令に付加する．図 2に示すように，ロード/ストア・ユニットにシフタを付加することで，以前は複数の命令を組み合わせて実行していたアドレス計算をロード/ストア・ユニットで 1 命令で実行できる．反面，インデックス・レジスタ値を読み出すために，ロード／ストア・ユニット 1 個につき整数レジスタ・ファイルに対して 1 ポート増加する．シフタの追加によってもハードウェア・コストが増加する (図 2参照)．

(b)**ストライド・アクセス法**：ストライド・アクセス法は，図 3の例で，インデックス (R8) を 2 ビットシフト後，ベース (R9) に加算する，という gcc の生成した 2 命令を，ベースにストライド (4) を加える 1 命令に置換する方法である．コンパイラによりアドレス計算の最適化を行うことにより命令数を減らす (図 3参照)．

スケール付きインデックス修飾を用いる方法では，ロード/ストア・ユニットに新たにシフタを追加し，レジスタ・ファイルへ 1 ポート追加せねばならない．しかし，整数 ALU で行っていたアドレス計算命令をロード/ストア命令が吸収することになるので，文献 [3] での検討において *III* を最小化する際にボトルネックとなっていた整数演算命令を大幅に減少させることができる．整数 ALU の個数を 2 個から 1 個に減らしても，整数演算命令が *III* を最小化する際のボトルネックとなる可能性は低い．ロード/ストア・ユニットにおいて，整数レジスタ・ファイルに対するポート数が増える．しかし，整数 ALU を 1 個にすることで全体としては整数レジスタ・ファイルのポート数は減少する (図 4参照)．

図 4，図 5に各方式を採用した時のレジスタ・ファイルの構成をそれぞれ示す．スケール付きインデックス修飾法の場合レジスタ・ファイルは 11 ポート，ストライド・アクセス法の場合は 12 ポート必要である．

将来のレイアウト設計の負担を考慮して，レジスタ・ファイルのポート数が少ない**スケール付きインデックス修飾法を導入**する．

3.1.2 機能ユニット構成

分岐／ジャンプ・ユニットは，他の機能ユニットと並列に動作するメリットが少ない．レジスタ・ファイルのポート数を削減するため，整数レジスタ・ファイルにポートを持つ整数 ALU と統合する．

文献 [3] での性能評価，および，4章での評価に用いている DLX[5] コンパイラは整数乗算命令を使用せずに，オブジェクト・コードを生成する．必要以上にレジスタ・ファイルのポート数を増加させないため，整数乗算器は整数 ALU と統合する．

以上の検討より，Turbo モードでは，

①整数 ALU + 整数乗算器 + 分岐／ジャンプ (以後，IALU) × 1
②浮動小数点 ALU(FALU)× 1
③浮動小数点乗除算器 (FMUL)× 1

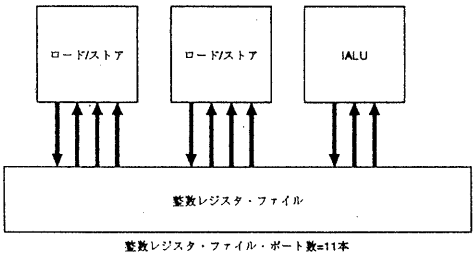


図 4: スケール付きインデックス修飾法での構成法

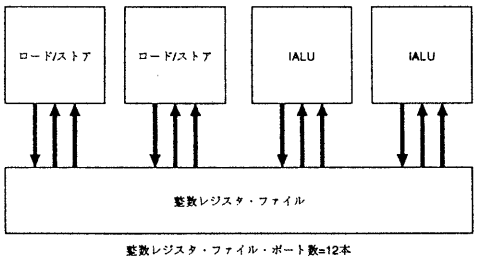


図 5: ストライド・アクセス法での構成図

④ロード／ストア (L/S)× 2

が並列に動作可能とする.

3.2 全体構成

プロトタイプの全体構成を図 6に示す. プロトタイプは, 3.1.2節で述べた FU を装備する. 分岐／ジャンプ・ユニットは, 分岐先の決定を早くするため ID ステージに装備し, それ以外の FU は, EX ステージに装備する. Normal モード時は, 同じ FU を使用する命令同士の組合せ以外は 2 命令同時実行可能である (並列度 2).

Turbo モード時は, IALU, FALU, FMUL, L/S_0, L/S_1 が並列動作可能である (並列度 5).

Normal モードおよび Turbo モードの場合の各ステージのパイプライン処理過程を以下に示す.

•Normal モード

①IF : 命令キャッシュからの命令フェッチ (1 ステージ).
②ID : 命令デコード, 分岐処理およびオペランド・フェッチ (1 ステージ).
③EX : 実行およびメモリ・アクセス (2〜4 ステージ).
④WB : レジスタ・ファイルへの書込み (1 ステージ).

•Turbo モード

①ID : IR から命令フェッチ, 命令デコード, 分岐処理およびオペランド・フェッチ (1 ステージ).
②EX : 実行およびメモリ・アクセス (2〜4 ステージ).
③WB : レジスタ・ファイルへの書込み (1 ステージ).

Turbo モードでは ID ステージにおいて, IR から各 FU に命令がディスパッチされる. ディスパッチされた命令は, 同一クロック内に Normal モードの ID ステージでの処理と同じ処理をする. EX ステージ以降の動作は, Normal モード時の動作と同じである.

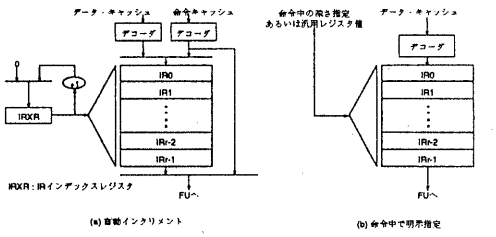


図 7: IR の構成

EX ステージは FU によりパイプライン段数が異なる. 各 FU の EX ステージにおけるパイプライン段数を以下に示す.

•IALU : 2 段.

•FALU : 4 段.

•FMUL : 4 段.

•L/S : 4 段.

プロトタイプでは, 割込みはサポートしない. ハザードはコンパイラにより静的に解消することを前提とする.

3.2.1 IR 構成

IR に命令をロードするとき, インデックスを自動的にインクリメントする方式 (図 7(a)) は IR の深さ i=0 から始めて命令がロードされるたびに自動的に 1 ずつインクリメントするため, 命令が NOP であっても IR に対してロードを行う必要があり非効率である. しかし, 命令中で IR の深さを指定 (図 7(b)) できるようにすれば, 全ての IR に NOP をセットする命令を設けて, あらかじめ全ての IR に NOP を設定しておき, NOP 以外の命令だけを専用のロード命令を用いて IR にロードすればコード・サイズを削減できる.

並行ロード方式 : 通常の命令実行と同時に IR にも命令をロードするのでコード・サイズの削減につながる. しかし, IR の深さ指定は自動インクリメントを用いざるを得ないので, 命令が NOP であっても IR に対してロードしなければいけないため, コード・サイズ削減の効果が減殺される.

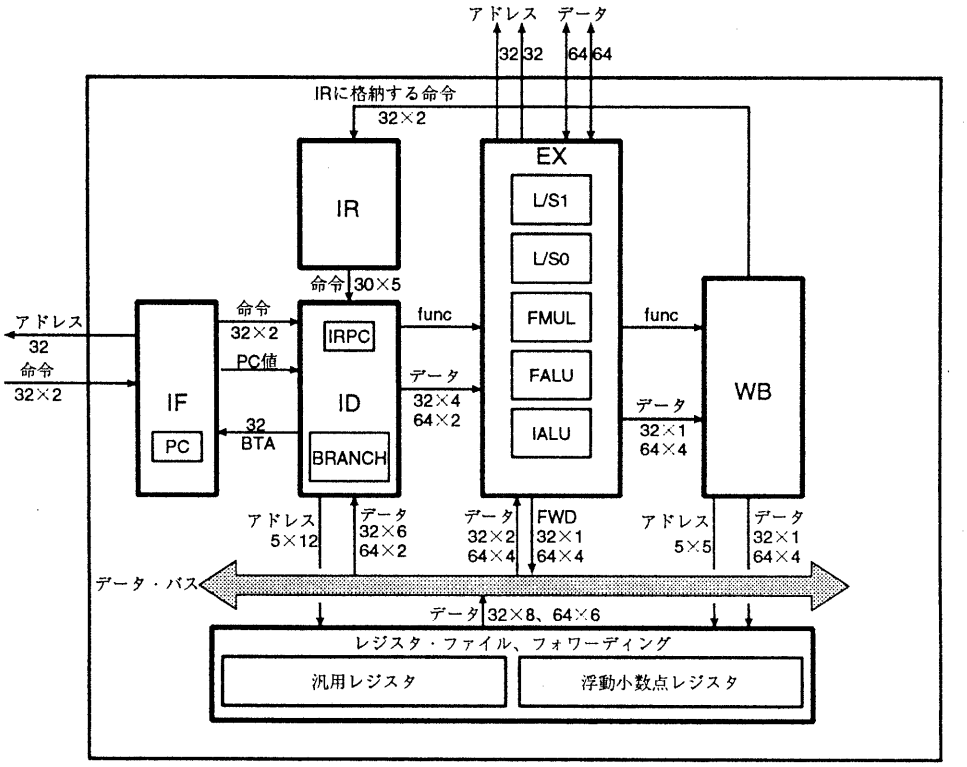
専念ロード方式 : 専用ロード命令中で IR の深さ指定が可能なので, 自動インクリメント方式による NOP ロードのデメリットはない. さらに, 命令中で深さを即値により直接指定するのではなく, レジスタを指定することにより深さを間接的に指定することで, IR の段数 (深さ) の変化にも対応できる. よって, プロトタイプでは**専念ロード方式を採用する**.

各 FU 毎の IR の段数は Turbo モードで実行できるループの III の最大値を決定する. しかし, IR はプロセッサ内に設けるため段数をあまり多く取ることができない. 文献 [3] での検討より, プロトタイプでは, **IR は各 FU 毎に 16 本装備し**,

$$III > IR\text{段数}$$

の場合は, 以下のいずれかの手法をとる.

•ループの分割により, III を小さくする.
•Normal モードで実行する.



数字 ：ビット数あるいはビット数×本数
BTA ：分岐先アドレス
func ：各FUの function code
FWD ：フォワーディング ・データ

図 6: プロトタイプの全体構成

3.2.2 レジスタ・ファイル構成

文献 [3] での検討より，ベクトル・レジスタを導入することの優位性はロード／ストア・パイプライン数に関係なく見られるが，ハードウェア・コストの増加に見合うだけの性能の向上が得られていない．プロトタイプにはベクトル・レジスタは装備せず，以下のスカラ・レジスタを装備する．

- 32 ビット，マルチポート汎用レジスタ：32 本
- 64 ビット，マルチポート浮動小数点レジスタ：32 本

(ただし，文献 [3] では，メモリ・レイテンシを 4 と小さめに仮定しているが，メモリ・レイテンシが大きい場合は，ベクトル・レジスタ導入を検討する必要がある．)

ハイパースカラ方式では，Turbo モード動作時には各 FU が並列に動作し，同時に読出しおよび書込みをレジスタ・ファイルに対して行う．プロトタイプの FU 構成の場合，整数レジスタ 32 本に対して最大で同時に 10〜12 のオペランド・アクセスが行われる可能性がある．バンク・コンフリクトによる性能低下を回避するため論理物理一致型 [2](マルチポート構成) を採用する．

整数演算器は汎用レジスタ，浮動小数点演算器は浮動小数点レジスタにしかアクセスする必要がないので，レジスタ・ファイルを整数レジスタ・ファイルと浮動小数点レジスタ・ファイルに分割する．両レジスタ・ファイル間の転送は，双方にポートを持つロード/ストア・ユニットを用いて，MOVE 命令により行う．

3.3 命令セット・アーキテクチャ

プロトタイプの命令セット・アーキテクチャを以下に示す．命令は，以下の 5 種類に分類される．

- 整数演算命令
- 浮動小数点演算命令
- 分岐／ジャンプ命令
- ロード／ストア命令
- ハイパースカラ方式特有命令

整数演算命令，浮動小数点演算命令，ロード／ストア命令は，Normal モードおよび Turbo モードでの動作は同じである．しかし，分岐／ジャンプ命令，ハイパースカラ方式特有命令には，Normal モードおよび Turbo モードでの動作が異なる命令，および，Normal モードあるいは Turbo モードでしか使用できない命令がある．

3.3.1 整数演算命令

Normal および Turbo モード共通命令である．整数演算は全て，レジスタ-レジスタ間およびレジスタ-即値間で実行される．即値形式では，即値 13 ビットを 32 ビットに符号拡張した値を使用する．以下に示す命令がある．

•加減算命令
•乗除算命令
•論理演算命令
•シフト命令
•比較命令

3.3.2 浮動小数点演算命令

Normal および Turbo モード共通命令である．単精度および倍精度について，以下に示す命令がある．

•浮動小数点加減算命令
•浮動小数点乗除算命令
•データ型変換命令 (整数型，浮動小数点単精度型，浮動小数点倍精度型の相互間)
•比較命令
  –2 つの浮動小数点レジスタの比較を行い，条件が真である時には浮動小数点状態レジスタに 1 を設定し，偽の時は 0 を設定する．

3.3.3 分岐／ジャンプ命令

分岐命令は全て条件付き分岐命令である．分岐条件は命令で指定されたソース・レジスタの値あるいは浮動小数点状態レジスタが 0 か否かをテストする．分岐先アドレスは，命令内の 19 ビット・オフセットを 32 ビットに符号拡張した値とプログラム・カウンタ (PC) を加算して得る．

ジャンプ命令は分岐先アドレスの指定法とリンクの有無で 4 つに分類される．

分岐先アドレスの指定法には，命令内の即値 23 ビットを 32 ビットに符号拡張した値を PC に加算してアドレスを得る方法とジャンプ先のアドレスが格納されているレジスタを指定する方法がある．また，リンクの場合は汎用レジスタ R31 にリターンアドレスを格納する．

分岐命令およびジャンプ命令は Normal モードでは PC を使用するが，Turbo モードでは PC の代わりに **IRPC(命令レジスタ・プログラム・カウンタ)** を使用する．

3.3.4 ロード／ストア命令

Normal および Turbo モード共通命令である．32 ビット汎用レジスタと 64 ビット浮動小数点レジスタに対するロード／ストアを行う．汎用レジスタへ，バイト，およびハーフワードのデータをロードする際には汎用レジスタの下位を使用し，上位部分は命令によって 32 ビットに符号拡張またはゼロ拡張を行う．浮動小数点レジスタへの単精度浮動小数点データをロードする際には浮動小数点レジスタの上位 32 ビットを使用し，下位 32 ビットにはゼロを設定する．倍精度浮動小数点数は単一の浮動小数点レジスタを占有する．

科学技術計算などにおいては参照の局所性が生かされないようなメモリ・アクセスをすることがよくある．参照の局所性が生かされないようなメモリ・アクセス時にはキャッシュ・ミスヒット・ペナルティにより性能が低下する．特に，Turbo モードにおいては性能が大幅に低下する．これを解決する手段としてキャッシュをバイパスしてメモリに直接アクセスする命令を用意した．この他，浮動小数点レジスタ間，汎用レジスタ-浮動小数点レジスタ間の転送命令を用意した．

3.3.5 ハイパースカラ方式特有命令

Turbo モードに関する命令で，ハイパースカラ方式特有の命令である．

•Turbo モードへの移行命令
Nomal モードから Turbo モードへ移行する際に使用する．
  –LDIR(*LoaD Instruction Register*)
  命令をメモリからフェッチし，簡単なデコードを行い使用する FU が決定する．使用する FU の IR に命令をロードする．IR の深さ指定は，LDIR 命令中で指定されたレジスタで，間接的に指定する．Normal モードでのみで有効である．
  –JALR2T(*Jump And Link to Turbo mode*)
  Turbo モード開始命令である．PC の値をレジスタに退避し，IRPC に即値をロードする．Normal モードのみで有効である．
  –FLUSH
  全ての IR に NOP をセットする．Normal モードのみで有効である．
•TQEQZ,TQNEZ(*Turbo mode Quit EQual Zero, Turbo mode Quit Not Equal Zero*)
Turbo モード終了条件判定命令である．条件付き分岐命令で，指定された汎用レジスタが 0 か否かを判定する．条件成立時は IR から FU への命令ディスパッチを中止し，PC に即値をロードして Turbo モードを終了する．条件不成立時は Turbo モードをそのまま続行する．Turbo モードで使用する．

4 予備性能評価

本性能評価ではハイパースカラ方式と他の命令レベル並列処理方式との性能比較を行い，ハイパースカラ・プロセッサの，他方式に対する相対的な位置付けを示す．評価モデルとしては，基本的にはプロトタイプを用いた．

ハイパースカラ・プロセッサの IR を有効に活用する方法として，次の 2 つの方法が存在する．

•ソフトウェア・パイプライニング処理
•擬似ベクトル処理

本評価では命令レジスタを有効に利用するため，ソフトウェア・パイプライニング処理を用いた場合における性能評価を行った．ベンチマーク・プログラムとして LFK(*Livermore Fortran Kernels*)24 の中から LFK1～LFK14 を用いた．

4.1 評価モデル

評価モデルとして，以下の 4 つを比較した．

•**PP(1)** : 並列度 1 の命令パイプライン．
•**SS(2)** : 並列度 2 のスーパースカラ．
•**VLIW(5)** : 並列度 5 の VLIW．
•**HS(2,5)** : Normal モードの並列度 2，Turbo モードの並列度 5 のハイパースカラ．

各モデル間の差異は並列度が異なるだけで，機能ユニット構成は等しいとする．表 1に評価モデルとして使用したハイパースカラ・プロセッサ，**HS(2,5)** の基本仕様を示す．**各モデルともに使用可能なレジスタの本数は制限していない．**メモリ構成については以下のような仮定を行った．

•メモリ構成
Turbo モード時におけるメモリ・アクセスはキャッシュをバイパスして直接メモリに対して行われるものと仮定した．また，メモリは完全にコンフリクト・フリーであり必ず一定時間 (4 クロック・サイクル) でメモリ・アクセスを完了すると仮定した．

表 1: **HS(2,5)** モデルの基本仕様

並列度	Normal モード	2
	Turbo モード	5
整数加減算	本数	1
/整数乗算	開始間隔サイクル数	1
/分岐ユニット	所要サイクル数	2
(IALU)	演算スループット	1 単精度演算/サイクル
浮動小数点	本数	1
加減算	開始間隔サイクル数	1
ユニット	所要サイクル数	4
(FALU)	演算スループット	1 単精度演算/サイクル
浮動小数点	本数	1
乗算	開始間隔サイクル数	1
ユニット	所要サイクル数	4
(FMUL)	演算スループット	1 単精度演算/サイクル
ロード/	本数	2
ストア・	開始間隔サイクル数	1
ユニット	所要サイクル数	4
(L/S)	バンド巾/ユニット	8 バイト/サイクル
整数	個数	(32)
レジスタ	語長	32 ビット
浮動小数点	個数	(32)
レジスタ	語長	64 ビット

4.2 評価方法および評価指標

以下の手順に従って，評価に用いたオブジェクト・コードを生成した．基本的には gcc によるオブジェクト・コードの生成以外はアルゴリズム [3] に従って人手により，上記の 4 つのモデルにそれぞれ適するよう，モデルごとに最適化を行った．逆依存関係への対処法としては，ステージ・バランシング (*SB*)[3] を用いた．

①gcc によりオブジェクト・コードを生成．

②基本ブロックの抽出．

③最内ループの検出．

④ループ内不変定数の追い出し．

⑤アドレス計算の高速化 (命令の置換)．

⑥*DAG* の生成．

⑦最内ループに対するソフトウェア・パイプライニング．

(a)**PP(1)** : 並列度 1 でソフトウェア・パイプライニング．
(b)**SS(2)** : 並列度 2 でソフトウェア・パイプライニング．
(c)**VLIW(5)** : 並列度 5 でソフトウェア・パイプライニング．
(d)**HS(2,5)** : 並列度 5 でソフトウェア・パイプライニング．ただし，$III \leq 16$ の条件を満たさない場合，Normal モードで実行するものとし，並列度 2 でソフトウェア・パイプライニング．

評価指標には，浮動小数点演算数および動作周波数の双方を正規化した FLOPC(*floating-point operations per clock cycle*) を用いた．

4.3 評価結果

図 8に評価モデルに対する評価結果を示す．表 1に示すようにいずれの評価モデルも 2 本の浮動小数点演算パイプラインを備えている．各モデルは並列度が異なるため，**PP(1)** モデルはスループットは最大 1FLOPC, **SS(2)**, **VLIW(5)**, および，**HS(2,5)** モデルのスループットは最大 2FLOPC である．

表 2: LFK 実行に必要な IR 段数

LFK	IR 段数	LFK	IR 段数
1	6	8	(40)
2	4	9	10
3	4	10	14
4	4	11	4
5	8	12	5
6	4	13	(22)
7	9	14	11

4.3.1 性能比較

性能比較の結果，以下の事実が判明した．

- **HS(2,5)** モデルは LFK8 以外のカーネルでは，**VLIW(5)** モデルに近い性能を示した．LFK8 では **HS(2,5)** モデルは $III > 16(IR$段数$)$ となり，Turbo モードで実行できないため **SS(2)** モデルと同じ結果になる．
- **HS(2,5)** モデルは LFK3,4,5,6,8,11,12,13 で **SS(2)** モデルと同程度の性能しか示していない．LFK8 以外は，プログラムの性質に起因するものである．ループ・キャリード・ディペンデンシが存在する場合や，使用する機能ユニットに偏りがある場合には性能を十分に発揮できていない．

4.3.2 命令レジスタ段数

表 2に **HS(2,5)** モデルで各 LFK を実行する上で必要となった IR の段数を示す．ハイパースカラ・プロセッサはプロセッサ内に IR を設けるため，IR 段数をあまり多く取ることができない．プロトタイプにおいては，IR 段数を 16 段としている．LFK1～14 の中では，LFK8, および，LFK13 が 16 段以内という制約を満たすことができなかった．本評価では，Normal モードで実行する，という選択をした．ループ分割が可能な場合については，

①ループの分割により IIIを小さくする．
②Normal モードで実行する．

という手順の採用について検討する必要がある．また，本評価においてはループ・アンローリングは適用していないが，IIIの値が小さく (具体的には 8 以下程度)，なおかつループ・アンローリング適用後にソフトウェア・パイプライニングしたほうが良い場合の対処法についても検討する必要がある．

5 おわりに

ハイパースカラ・プロセッサのハイパフォーマンス・プロトタイプ・プロセッサの設計仕様，および，予備性能評価について述べた．プロトタイプの基本構成は図 1 の通りである．スケール付きインデックス修飾法をハードウェアでサポートし，命令レジスタ段数は 16 段とした．命令セット・アーキテクチャは，RISC における基本的な命令に，ハイパースカラ方式独自の命令を加えたものとする．さらに，キャッシュをバイパスしてメモリに直接アクセスするロード/ストア命令を設けた．

予備性能評価より，プロトタイプは，

$III \leq IR$本数

の場合は，同等の機能ユニット構成を持つ VLIW プロセッサとほぼ同じ性能が得られることが判明した．

$III > IR$本数

の場合は，以下のいずれかの手法をとる．

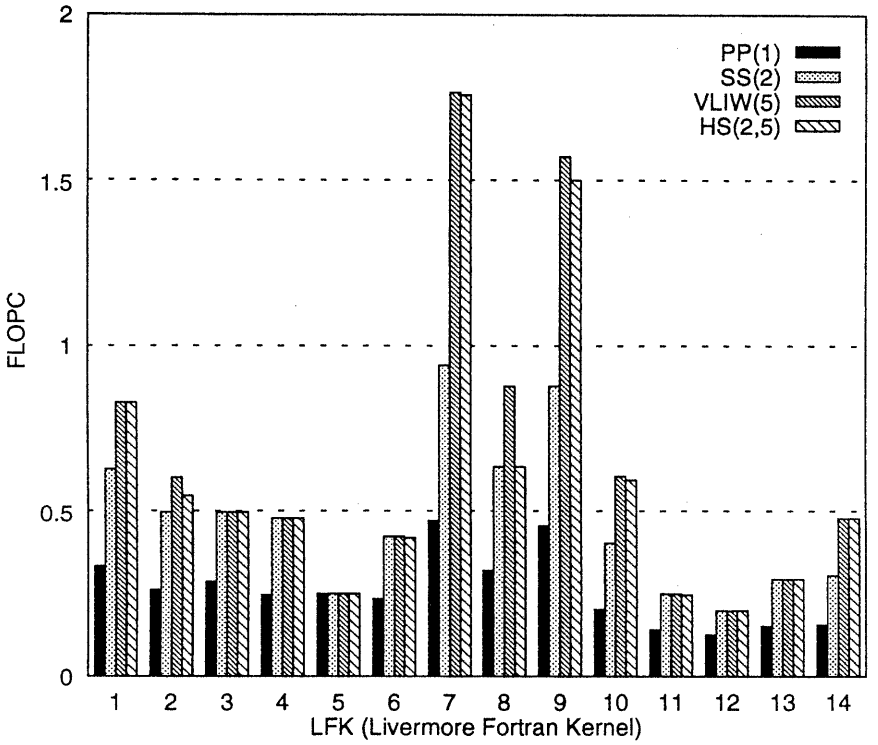


図 8: 4 つのモデルの性能比較

- ループの分割により，IIIを小さくする．
- Normal モードで実行する．

性能の低下を最小限に抑えるために，さらに検討が必要である．

また，本評価ではメモリ・レイテンシを 4 としたが，実際はもっと大きい値をとる．メモリ・レイテンシが大きくなった場合の影響についても早急に評価する必要がある．

謝辞

PARTHENON をご提供頂いた中村行宏氏，小栗清氏をはじめとする NTT コミュニケーション科学研究所の諸氏に感謝致します．

日頃ご討論頂く九州大学 大学院総合理工学研究科 安浦寛人教授，ならびに，安浦研究室の諸氏に感謝致します．

参考文献

[1]村上和彰：“ハイパースカラ・プロセッサ・アーキテクチャ — 命令レベル並列処理への第 5 のアプローチ —”，並列処理シンポジウム JSPP '91 論文集 pp.133–140，1991 年 5 月．

[2]斎藤靖彦，村上和彰，“ハイパースカラ・プロセッサ・アーキテクチャ — 実現上の課題 —，”情処研報，93–ARC–101–12,1993 年 8 月．

[3]弘中哲夫，斎藤靖彦，村上和彰，“ハイパースカラ・プロセッサ・アーキテクチャ — ソフトウェア・パイプライニング処理に関する性能評価 —，”信学技報，VLD93–89，1993 年 12 月．

[4]NTT データ通信 (株)，PARTHENON REFERENCE MANUAL，PARTHENON USERS MANUAL．

[5]Hennessy，J. L. and Patterson，D. A. (富田眞治，村上和彰，新實治男 訳)：*Computer Architecture : A Quantitative Approach*，Morgan Kaufmann Publishers,Inc.,1990．