計算機アーキテクチャ　103－16
設　計　自　動　化　69－16
(1993. 12. 17)

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 ―

弘中 哲夫†　斎藤 靖彦‡　村上 和彰‡

†九州大学 情報工学科

‡九州大学 大学院 総合理工学研究科

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

あらまし

ハイパースカラ・プロセッサ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．そのため，如何にプロセッサ内に自己形成されるVLIWプログラムの並列性高めるかが問題となる．本稿では，ハイパースカラ・プロセッサ用のオブジェクト・コードに適用するソフトウェア・パイプライニングの手法の概略，および，その手法により生成されたオブジェクト・コードを用いてハイパースカラ・プロセッサの性能評価を行った結果を示している．その結果，ベクトル・レジスタの有無に関わらずハイパースカラ・プロセッサが従来型ベクトル・プロセッサと同程度または，それ以上の性能を達成可能なことが示された．

和文キーワード：スーパースカラ・プロセッサ，VLIW，ソフトウェア・パイプライニング，ベクトル・プロセッサ，ベクトル処理，性能評価

Hyperscalar Processor Architecture
— Performance of Software Pipelining —

Tetsuo HIRONAKA†　Yasuhiko SAITOH‡　Kazuaki MURAKAMI‡

†Department of Computer Science and Communication Engineering
Kyushu University

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

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

Abstract

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. So, to improve the performance, instruction-level parallelism in the self-created VLIW instructions must be increased. This paper, describes the way how software pipelining is used to optimize the code for the hyperscalar processor, and also, the results of the performance evaluation are presented. The preformance evaluation shows that the hyperscalar processor would achieve the same or even better performance compared, to the conventional vector processors.

英文 **key word**: Superscalar processor, VLIW, Software pipelining, Vector processor, Vector processing, 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 [2]. 本稿ではハイパースカラ・プロセッサで用いるソフトウェア・パイプライニングの手法と, その手法を評価した結果を示す.

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. 1 つはハイパースカラ・プロセッサに最適なコンパイレーション手法の確立. もう 1 つは, ハイパースカラ・プロセッサ・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を評価である.

本稿では, ソフトウェア・パイプライニングに施したオブジェクト・コードを用い, いくつかの異なるハードウェア構成を持つハイパースカラ・プロセッサについて性能評価を行った結果を示す. 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章で評価に用いたソフトウェア・パイプライニングの具体的なアルゴリズムについて述べる. 4章ではハイパースカラ・プロセッサの評価手法, および, 評価を行った項目について述べ, 5 章では評価結果について述べる. 最後に 6章で本稿をまとめる.

2 ハイパースカラ・プロセッサ

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.

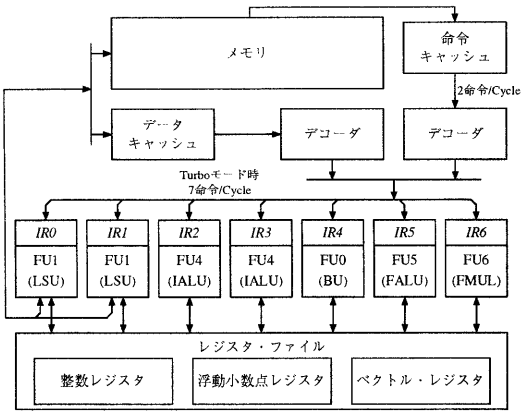
ハイパースカラ・プロセッサの構成は基本的にはスーパスカラ・プロセッサと変わらない. 命令長および, 命令フェッチ巾は 1 またはスーパースカラ・プロセッサ等と同程度とする. 例えば, 命令長は 32 ビットで命令フェッチ巾 1〜4 命令程度で構わない. これにより, VLIW 方式の短所であるコード・サイズの増加および命令キャッシュの低使用効率といった問題を解決する.

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ロード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.



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

3 ソフトウェア・パイプライニング

3.1 基本アルゴリズム

本節では今回の評価に用いたソフトウェア・パイプライニングの具体的なアルゴリズムについて述べる. ソフトウェア・パイプライニングの基本的アルゴリズムとしてモジュロ・スケジューリングを用いた [1]. 以下にスケジューリング手法の概略を示す.

ここでは各記号を次のように定義する. Rはシステム内に保持される全機能ユニット数を保持するベクトルである. $R(k)$ はシステム内に含まれる機能ユニット kの個数である. また, 基本ブロック内のすべての命令とその依存関係を *DAG*(*Directed Acyclic Graph*) $G=(V,E)$ で表す. この時, 各 e $(e \in E)$ は依存関係を保証するための最小遅延時間 $d(e)$ を持ち, 各命令ν $(\nu \in V)$ は機能ユニットを占有する時間 $l(\nu)$ をもつ. また, $\rho_\nu(i)$ はνが各ステップ iで占有する機能ユニット数を示す.

1. ソフトウェア・パイプライニングするための *III* (*Iteration Initiation Interval*) Sの上限 S_{max}と下限 S_Rを求める. *III* の下限 S_Rは 1 イタレーション実行に最大必要な資源数, および, プロセッサが提供する資源数から求まる最小の実行時間を次式より計算し, S_Rとした.

$$S_R := max_k \left\lceil \frac{\sum_{\nu \in V, 0 \le i < l(\nu)} \rho_\nu(i,k)}{R(k)} \right\rceil . \qquad (1)$$

*IIIS*の上限 S_{max}は通常のリスト・スケジューリングを行い, このとき 1 イタレーション実行に必要なサイクル数を S_{max}とした. この値を以上の *III* でソフトウェア・パイプライニングすることは無意味である.

2. *III* を $s := S_R$と置く.
3. *III* を sとしたモジュロ・スケジューリングを行う [1]. モジュロ・スケジューリングとは通常のリスト・スケジューリング・アルゴリズムにおいて資源の制約に関してのみ, 次式$\overline{\rho}^s_\nu(i)$ を適用したものである.

$$\overline{\rho}^s_\nu(i) = \sum^{j \in Z} \rho(i+js). \qquad (2)$$

4. 3のモジュロ・スケジューリングにより資源の制約, および, データ依存関係をすべて満たすことに成功し, かつ, $s < S_{max}$ を満たすとき, スケジューリングを終了する. あるいは, $s \le S_{max}$が満たされなくなった場合は, ソフトウェア・パイプライニング不可能なループとしてスケジューリングを終了する.
5. *III* を $s := s+1$ として再び 3を行う.

3.2 逆依存の解決

より小さな *III* でソフトウェア・パイプライニングすることは, より高い機能ユニット使用率を達成する上で重要である. しかしながら, レジスタ値のライフタイムが長い場合にはイタレーション間で生じる逆依存のため, より小さな *III* でソフトウェア・パイプライニングを行うのは困難である. そこで, このような逆依存関係を解消し, より小さな *III* でのスケジューリングを可能にするため, ソフトウェアおよびハードウェアによる解決策として, 従来次のような手法が用いられていた.

1. モジュロ・バリアブル・エクスパンション (*Modulo Variable Expansion*) : 各イタレーション毎に異なるレジスタを用いるようにすることでイタレーション間の逆依存を解消する. 具体的にはループ・アンローリングを適用した後, ソフトウェア・パイプライニングを適用することで実現する. ただし, アンローリングを行うため, コード・サイズが大きくなるという欠点が存在する.
2. ベクトル・レジスタの導入 : 各イタレーション毎でベクトル・レジスタの異なる要素をアクセスするようにすることで逆依存を解消する. 具体的にはレジスタ値のライフタイムが S_Rより長いものに対し, ベクトル・レジスタを割り当てることで実現できる. 本方式の場合モジュロ・バリアブル・エクスパンションと異なりループ・アンローリングを必要としないのでコード・サイズは大きくならない.

ハイパースカラ・プロセッサでこれらの手法を用いることを考えた場合, プロセッサ内の命令レジスタ・サイズが限られていることからコード・サイズが大きくなる 1の適用は望ましくない. また, 2の場合にしても, ベクトル・レジスタの導入によるハードウェア・コストの増加が見込まれる.

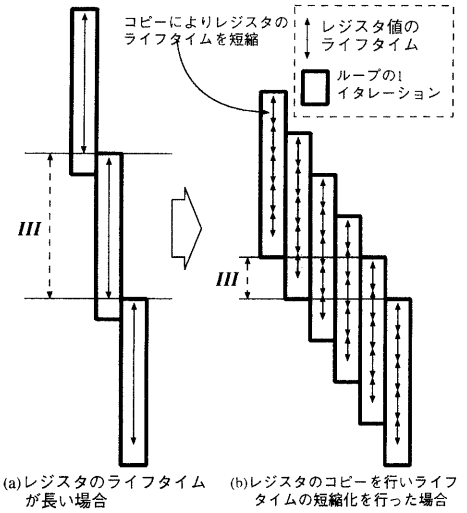


図 2: ステージ・バランシング (*SB*)

そこで, ハイパースカラ・プロセッサで行うソフトウェア・パイプライニングが要求する小さなコード・サイズを満たし, ベクトル・レジスタの導入によるハードウェア・コストの増加を伴わない逆依存の解決法として独自の手法である**ステージ・バランシング** (*Stage Balancing*) を用いた. 以降**ステージ・バランシング**を *SB*と呼ぶ. この手法はアンバランスに長いステージ・レイテンシをもつパイプライン・プロセッサにおいて, パイプライン・レジスタを適時挿入することでより, より小さいパイプライン・ピッチでのパイプライン処理を達成することと等価である.

図 2に *SB*の概念を示す. *SB*はレジスタ値のライフタイムが原因となる逆依存関係を防止するため, S_Rより, 長いライフタイムを持つレジスタ値に対して適時コピー命令を挿入し, ライフタイムを常に S_R未満にする手法である.

*SB*はモジュロ・スケジューリングと同時に実行される. 具体的には 3.1節の 3を次のように変更して行う. モジュロ・スケジューリング実行時に S_Rより長いライフタイムを持つレジスタ値 R_{long}があった場合. S_Rを越えない最長のライフタイムが得られる位置で R_{long}を他のレジスタ (R_{copy}) にコピーする命令をプログラム中に挿入する. コピー命令挿入と同時に S_Rの更新する. さらに, コピー命令以降にスケジューリングされる R_{long}参照命令はすべて, R_{copy}を参照するように変更され, スケジューリングされる. また, S_Rが更新された結果 $s \leq S_R$が成立しなくなった場合, 現在の s を *III* に用いたスケジューリングをあきらめる.

4 評価法

ハイパースカラ・プロセッサの命令レジスタを有効に活用する方法として次の 2 つ手法が存在する [2].

- 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 評価法および評価項目

以下の手順に従い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・コードを生成以外はアルゴリズムにしたがって人手によりハイパースカラ・プロセッサに適するように最適化を行った.

1. gcc 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. *DAG* の生成.
6. 最内ループに対するソフトウェア・パイプライニング.

評価は 3節で述べたソフトウェア・パイプライニング手法の内, 次の 2 つを評価の対象とした.

- *HPS(SR)*: 逆依存関係への対処法として *SB*を用い, スカラ・レジスタのみを使用して最適化されたオブジェクト・コード.
- *HPS(VR)*: 逆依存関係への対処法としてベクトル・レジスタを導入し, 最適化したオブジェクト・コード.

なお, 命令レジスタへの命令の格納はいずれの場合も専念ロードにより行った.

この他に, メモリ・バンド幅が性能に与える影響を評価するため, ロード/ストア・パイプライン本数を 1～2 と変化させて評価を行った. 故に評価モデルは以下の 4 つである.

- *HPS(SR)1*, *HPS(VR)1*: ロード/ストア・パイプラインが 1 本の場合.
- *HPS(SR)2*, *HPS(VR)2*: ロード/ストア・パイプラインが 2 本の場合.

表 1に評価モデルとして使用したハイパースカラ・プロセッサの仕様を示す. なお, レジスタ構成, および, メモリ構成について次のような仮定を行った.

表 1: 評価モデルの仕様

並列度	**Normal** モード	1
	Turbo モード	6 または 7
整数加減算ユニット	本数	2
	開始間隔サイクル数	1
	所要サイクル数	1
	演算スループット	1 単精度演算/サイクル
浮動小数点加減算ユニット	本数	1
	開始間隔サイクル数	1
	所要サイクル数	4
	演算スループット	1 単精度演算/サイクル
浮動小数点乗算ユニット	本数	1
	開始間隔サイクル数	1
	所要サイクル数	4
	演算スループット	1 単精度演算/サイクル
ロード/ストア・ユニット	本数	1 または 2
	開始間隔サイクル数	1
	所要サイクル数	4
	バンド巾/ユニット	4 バイト/サイクル
分岐ユニット	本数	1
	開始間隔サイクル数	1
	所要サイクル数	1
整数レジスタ	個数	32
	語長	32 ビット
浮動小数点レジスタ	個数	32
	語長	32 ビット
汎用ベクトルレジスタ	個数	0 または 32
	語長	32 ビット
	レジスタ長	16 語

- レジスタ構成
  ハイパースカラ・プロセッサのレジスタ構成, 特に機能ユニット間での共有形態とベクトル・レジスタの参照法に関して文献 [4] でさまざまな検討が行われているが, ここではコード・スケジューリングを容易にするため, 次のように仮定した.
  - レジスタの共有形態 : 機能ユニット間で「読出し (R) & 書込み (W) 完全共有」
  - ベクトル・レジスタのポインタ構成 : 各命令毎にベクトル・レジスタを読出し, 書込みを行うためのポインタを持ち, ポインタのインクリメントは暗黙的に行われる. ポインタのラップアラウンドも可能である.
- 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.

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.

5 評価結果

図 3に評価モデル *HPS(SR)1*, *HPS(VR)1*, *HPS(SR)2*, および, *HPS(VR)2* 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ため, スループットは最大 2FLOPC となる.

5.1 ハイパースカラ・プロセッサの性能

スカラ・レジスタのみに *SB*を適用した場合と, ベクトル・レジスタを導入した場合の性能を比較を行うため, 評価モデル *HPS(SR)* と評価モデル *HPS(VR)* の性能に着目する. ベクトル・レジスタを導入することで評価モデル *HPS(SR)1* と評価モデル *HPS(VR)1* 間で最小 -0.8%～最大 47.7%相乗平均で 3.9%, *HPS(VR)1* の性能が勝っている. 評価モデル *HPS(SR)2* と評価モデル *HPS(VR)2* 間で最小-2.3%～最大 47.7%相乗平均で 9.9%で *HPS(VR)2* の性能が勝っている. このようにベクトル・レジスタを導入することの優位性はロード/ストア・パイプライン数の増減に関係なく見られた. しかしながら, 両者の性能差は LFK1, LFK7 を除くとわずかである. LFK1, LFK7 にみられた性能差は *SB*適用により挿入されたコピー命令が *III* を増加させたためである.

次に, メモリ・バンド幅の変化に対する性能への影響に着目すると, 評価モデル *HPS(SR)1* と評価モデル *HPS(SR)2* 間で最小 -0.4%～最大 82.6%相乗平均で 6.5%, 評価モデル *HPS(VR)1* と評価モデル *HPS(VR)2* 間で最小 -0.0%～最大 80.0%相乗平均で 9.9%, 性能が勝っている. しかしながら, 図 3からもわかるように, メモリ・バンド幅の変化による性能向上はあまり大きくない. これは, 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多くの整数演算命令を用いているため, 整数演算命令数が *III* を決定しているためである. そのため, アドレス計算手法の改善, または, 整数演算器数を増加させることで性能をより高めることができる.

5.2 Turbo モードが実行時間に占める割合

表 2に各 LFK の総実行時間に占める **Turbo** モードでの実行時間の割合を示す. 評価モデルに使用したハイパースカラ・プロセッサでは **Turbo** モード時に並列度 6 または 7 で動作し, **Normal** モード時に並列度 1 で動作する. そのため, 演算性能を十分に発揮するためには **Normal** モードでの処理時間にくらべ **Turbo** モードでの処理時間が十分に大きい必要がある. **Normal** モードで処理されるコードとしては,

- 最内ループ以外のコード,
- 命令レジスタへ専念ロードを行うためのコード,

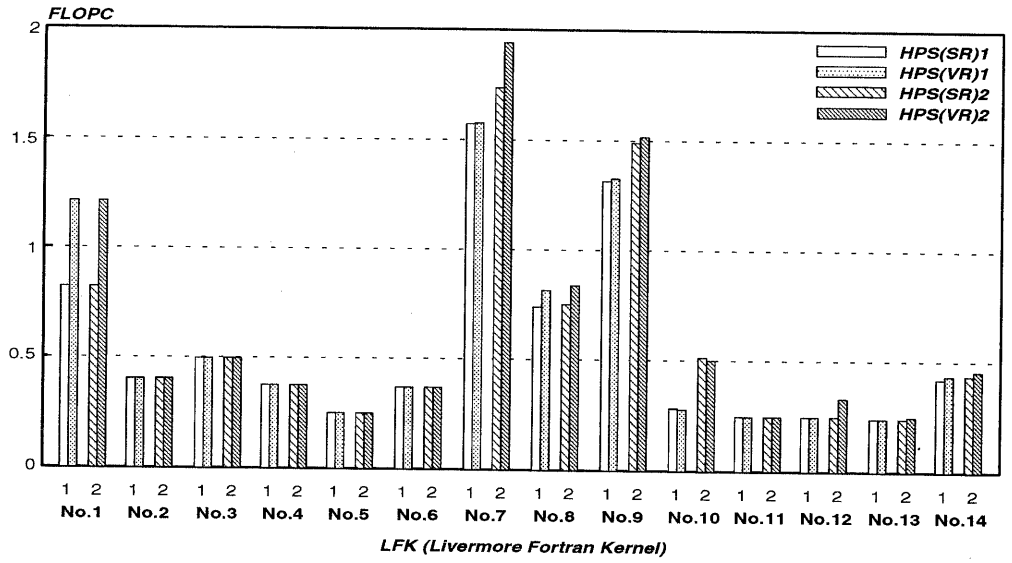


図 3: ハイパースカラ・プロセッサの性能

表 2: Turbo モードが実行時間に占める割合

LFK	*HPS(SR)1*	*HPS(VR)1*	*HPS(SR)2*	*HPS(VR)2*
1	98.2	96.4	98.2	96.4
2	49.1	49.1	49.1	49.1
3	98.6	98.6	98.6	98.6
4	91.6	91.6	91.6	91.6
5	99.4	99.4	99.4	99.4
6	62.2	62.2	62.2	62.2
7	97.9	98.0	97.1	96.6
8	94.2	89.9	94.1	83.4
9	82.3	83.1	75.5	76.9
10	93.5	91.8	88.1	85.3
11	98.5	98.8	98.5	98.8
12	98.5	98.6	98.1	97.4
13	83.9	84.0	84.2	83.9
14	98.0	98.1	97.9	98.2

表 3: LFK 実行に必要な命令レジスタ段数

LFK	*HPS(SR)1*	*HPS(VR)1*	*HPS(SR)2*	*HPS(VR)2*
1	6	4	6	4
2	6	6	6	6
3	4	4	4	4
4	5	5	5	5
5	8	8	8	8
6	4	4	4	4
7	10	10	9	8
8	47	41	46	41
9	11	11	9	9
10	31	31	16	16
11	4	4	4	4
12	4	4	4	3
13	26	26	26	25
14	12	12	12	12

- ソフトウェア・パイプライニング処理におけるプロローグおよびエピローグのコードである.

5.3 LFK 実行に必要な命令レジスタ段数

表 3に各 LFK を実行する上で必要となった命令レジスタの段数を示す. ハイパースカラ・プロセッサはプロセッサ内に命令レジスタを設けるため, 命令レジスタ段数をあまり多く取ることができない. 評価モデル *HPS(SR)* と評価モデル *HPS(VR)* を比較するとスカラ・レジスタのみを用いてソフトウェア・パイプライニングを行った場合と, ベクトル・レジスタを用いてソフトウェア・パイプライニングを行った場合では命令レジスタ段に大きな差がないことがわかる. これは *SB* を用いることでベクトル・レジスタを用いない小規模なハードウェアで, ベクトル・レジスタを用いた場合と同程度の小さな命令レジスタ段数で, ベクトル・レジスタを用いた場合に遜色ない性能を達成することが可能なこと示している.

また, 表 3より, ループ・アンローリングを適用せずにソフトウェア・パイプライニングを行う場合には LFK1～14 で必要な命令レジスタの段数は 50 以内であると言える.

5.4 ベクトル・プロセッサとの性能比較

ハイパースカラ・プロセッサとベクトル・プロセッサとの相対的な性能差を評価するため, 筆者らが行ったベクトル・プロセッサ [3] と性能比較を行った. ただし, 評価対象としたベクトル・プロセッサと今回評価対象としたハイパースカラ・プロセッサの間には演算器構成の違いがあり

単純な比較はできない．以下に両者の相違点を示す．

- 演算パイプライン：評価対象ハイパースカラ・プロセッサは整数演算用演算パイプラインと浮動小数点演算用パイプラインがそれぞれ独立している．それゆえ，整数加減算，浮動小数点加減算．整数乗除算，浮動小数点乗除算のパイプラインが4並列で動作可能である．しかしながら，評価対象ベクトル・プロセッサでは整数と浮動小数点は1本のパイプラインで処理する．そのため，加減算用，乗除算用で最大2並列でしか動作しない．
- ロード/ストア・パイプライン：評価対象ハイパースカラ・プロセッサはロード/ストア・ユニット内にアドレス計算用の演算器を持たない．逆に，評価対象ベクトル・プロセッサはアドレス計算用の演算器をロード/ストア・パイプライン内に持つ．

以上から，ハイパースカラ・プロセッサではアドレス計算に使用できる整数演算器が最大2つしかないのに対し，ベクトル・プロセッサは最大4つ持つ．多くのLFKでアドレス計算用の整数演算命令がボトルネックになっていることを鑑みると本評価はハイパースカラ・プロセッサにとって不利な評価となっている．評価対象ベクトル・プロセッサの基本仕様を表5に示す．

図4に筆者らが行ったベクトル・プロセッサ[3]と，ハイパースカラ・プロセッサの性能比較を行った結果を示す．なお，評価の対象に用いたベクトル・プロセッサのモデルは*VP*, *VPM*, *VPMF*の3つである．以下に各モデルについて説明する．

- *VP*: 従来のベクトル・プロセッサに相当するモデルである．文献[3]における*Base*モデルである．
- *VPM*: 従来のベクトル・プロセッサにベクトル命令レベルのマルチスレッド処理機能を付加したモデルである．文献[3]における*Base+M*モデルに相当する．
- *VPMF*: *VPM*モデルのベクトル・レジスタをリングFIFOバッファとして動作可能にしたモデルである．各ベクトル・レジスタのポインタをラップラウンド可能とすることで実現する．文献[3]における*Base+M+F*モデルに相当する．

なお，各ベクトル・プロセッサの評価モデルはハイパースカラ・プロセッサの評価モデルと条件を等しくするため，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ものとした．

図4の評価結果から以下のことがわかる．なお，表4には各評価モデルの平均FLOPC値を示す．

- *HPS(SR)2*とベクトル・プロセッサを比較した結果，*HPS(SR)2*は相乗平均で*VP*に対して1.01倍，*VPM*に対して0.75倍，*VPMF*に対して0.68倍性能が良いことがわかった．以下に各々のLFKに対する優劣を示す．
  - *VP*以上の性能を示すのはLFK5〜7, LFK9〜11, および, LFK13〜14の8つのカーネルである．
  - *VPM*以上の性能を示すのはLFK5, LFK7, LFK9〜11, および, LFK13〜14の7つのカーネルである．
  - *VPMF*以上の性能を示すのはLFK5, LFK10, LFK11, および, LFK13の4つのカーネルである．
- *HPS(VR)2*とベクトル・プロセッサを比較した結果，*HPS(VR)2*は相乗平均で*VP*に対して1.08倍，*VPM*に対して0.81倍，*VPMF*に対して0.73倍性能が良いことがわかった．以下に各々のLFKに対する優劣を示す．
  - *VP*以上の性能を示すのはLFK5〜11, および, LFK13〜14の9つのカーネルである．
  - *VPM*以上の性能を示すのはLFK5, LFK7, LFK9〜11, および, LFK13〜14の7つのカーネルである．
  - *VPMF*以上の性能を示すのはLFK5, LFK10, LFK11, および, LFK13の4つのカーネルである．

表4: 各評価モデルの平均性能

FLOPC	*HPS(SR)2*	*HPS(VR)2*	*VP*	*VPM*	*VPMF*
調和平均	0.4085	0.4312	0.2776	0.4605	0.4888
算術平均	0.9529	0.8338	0.6788	0.6550	0.5971

表5: ベクトル・プロセッサの基本仕様

加減算パイプライン	本数	1
	開始間隔サイクル数	1
	所要サイクル数	5
	演算スループット	1倍精度演算/サイクル
乗除算パイプライン	本数	1
	開始間隔サイクル数	1(除算以外) 12(除算)
	所要サイクル数	4(除算以外) 12(除算)
	演算スループット	1倍精度演算/サイクル (除算以外)
ロード/ストアパイプライン	本数	2
	開始間隔サイクル数	1
	所要サイクル数	4
	バンド巾	8バイト/サイクル
ベクトル・レジスタ(データFIFO)	個数	16
	語長	64ビット
	レジスタ長	32語
	バンド巾/レジスタ	1語/サイクル
スカラ・レジスタ	汎用レジスタ	32個
	浮動小数点レジスタ	32個
制御レジスタ	仮想パイプライン制御(VPCR)	16個
	ベクトル長(VLR)	1個

6 おわりに

ハイパースカラ・プロセッサの命令レジスタを有効に活用するためのオブジェクト・コード最適化手法について

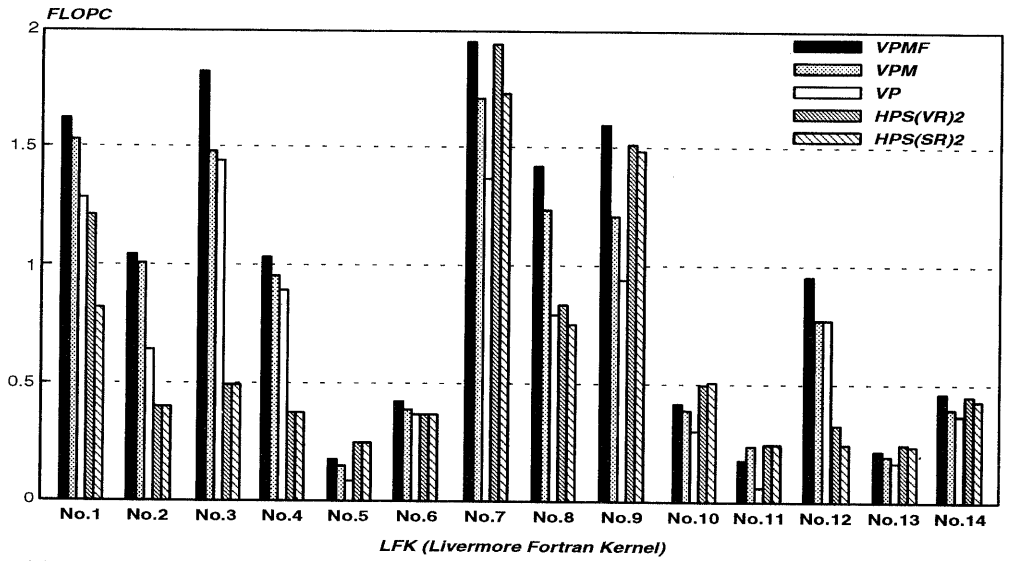


図 4: ハイパースカラ・プロセッサ vs. ベクトル・プロセッサ (ロード/ ストア・パイプライン本数 2)

述べ, さらにそれを用いた場合におけるハイパースカラ・プロセッサの性能を評価した. その結果, 次のことが判明した.

- モジュロ・スケジューリングによるソフトウェア・パイプライニングと, 次の 2 つの逆依存関係解決法を用いることにより, ループ・アンローリングを用いることなく, 従来型のベクトル・プロセッサと同程度の性能を達成することができるがわかった.
  - スカラ・レジスタのみを使用して逆依存に対処するオブジェクト・コード最適化手法であるステージ・バランシング (*SB*) を導入する.
  - ハードウェア的に逆依存関係に対処するため, ベクトル・レジスタを導入する.
- ループ・アンローリングを用いないソフトウェア・パイプライニング手法を用いることで, LFK 実行に必要な命令レジスタ段数は 50 以下であることが判明した.
- ハイパースカラ・プロセッサで LFK を実行場合において, 大並列度で動作する **Turbo** モードと小並列度で動作する **Normal** モードが各々総実行時間に占める割合を求め, ハイパースカラ・プロセッサの有効性を示した.
- ベクトル・プロセッサとの性能比較を行い, 従来型のベクトル・プロセッサと同程度の性能が得られることが判明した. さらに, 評価対象のベクトル・プロセッサが評価対象ハイパースカラ・プロセッサより多くの整数演算ユニットをもつことを考慮するとハイパースカラ・プロセッサの方がより良い性能を持つ可能性があることを示した.

今回は gcc が生成したオブジェクト・コードをそのまま使用してソフトウェア・パイプライニングを行った. そのため, ハイパースカラ・プロセッサの整数演算器構成を十分に考慮に入れたオブジェクト・コードを用いていない.

今後の課題としてソフトウェア・パイプライニングを行う前のコードにおけるアドレス計算法の最適化や, ロード/ストア命令に演算機能つけるなど, ソフトウェア, ハードウェアの両面からアドレス計算の高速化に関する工夫を行ってゆく予定である.

参考文献

[1] Monic S. Lam, "A Systolic Array Optimizing Compiler, " *Kluwer Academic Publishers*, pp.83–124, 1998.

[2] 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.

[3] Hashimoto, T., Murakami, K., Hironaka, T., and Yasuura, H., "A Micro-vectorprocessor Architecture — Performance Modeling and Benchmarking —," *Proc. 1993 Int'l. Conf. on Supercomputing*, pp.308–317, July 1993.

[4] 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 —, " 情報処理学会研究報告,93–ARC– 101–12,1993 年 8 月.