計算機アーキテクチャ 120－5
（1996．10．31）

ジェットパイプラインの並列化命令スケジューリングに関する一検討

仲池卓也† 佐々木毅人† 片平昌幸†
小林広明† 中村維男†

ジェットパイプラインは、ベクトル処理と命令レベル並列処理を併用することによって、高速演算を可能にするアーキテクチャである。したがって、ジェットパイプラインのコンパイラは、ベクトル命令とスカラ命令が混在したコードを並列化する必要がある。しかし、ベクトル命令とスカラ命令は、性質が異なるため、VLIW 計算機などで用いられている並列化手法を、そのまま適用することはできない。本稿では、スーパスカラプロセッサなどで用いられているディスパッチスタック法を基に、ベクトル命令とスカラ命令を融合した並列化手法を提案し、シミュレーションによりその効果を確認する。

A Study on Parallelizing Scheduling for Jetpipeline

TAKUYA NAKAIKE,† TAKEHITO SASAKI,† MASAYUKI KATAHIRA,†
HIROAKI KOBAYASHI† and TADAO NAKAMURA†

Jetpipeline is an architecture based on instruction-level parallelism(ILP), which utilizes vector and scalar processing to achieve high performance. Therefore, the compiler for Jetpipeline must parallelize vector and scalar instructions of programs. However, since vector instructions take more cycles to complete their execution than scalar instructions, it is not suitable to use parallelizing methods used in VLIW machines. In this paper, we propose a parallelizing method for Jetpipeline by improving the dispatch stack method to parallelize the vector and scalar instructions.We show the effectiveness of the proposed parallelizing method for Jetpipeline through simulation experiments.

1. はじめに

近年、多様化するアプリケーションプログラムを高速に実行するために、様々なアーキテクチャが提案されている。科学技術計算の分野では、数値計算に多く見られる繰り返し構造を高速実行するために、主にベクトル処理方式が用いられる[1]。また、分岐命令を多く含むプログラムは、スーパースカラ[2]やVLIW[3]などの命令レベル並列処理方式によって効果的に実行される。

ジェットパイプラインは、ベクトル処理方式と命令レベル並列処理方式を効果的に組み合わせたアーキテクチャである[4][5]。そのため、科学技術計算、および非数値計算を含むアプリケーションプログラムの両方を高速に実行できる。ジェットパイプラインでは、ベクトル処理と命令レベル並列処理を併用するため、コンパイラがベクトル化、および並列化を行う。ベクトル化は、通常のベクトルプロセッサで用いられている手法をそのまま適用することができる。しかし、並列化は、ベクトル命令とスカラ命令の混在したコードに対して、効果的に行う必要がある。したがって、ベクトル命令、およびスカラ命令の並列性を考慮した並列化手法が必要になる。

本稿では、科学技術計算に多く見られるループ構造におけるベクトル命令とスカラ命令を融合した効果的な並列化手法について考察する。その考察に基づき、ディスパッチスタック法[6]を改良したジェットパイプラインのための並列化手法を提案する。そして、シミュレーションにより提案する並列化手法の有効性を確認する。

第2章では、対象になるジェットパイプラインアーキテクチャの構成について述べる。第3章では、ジェットパイプラインのための並列化手法について述べる。第4章では、シミュレーションによる性能評価、および考察を行う。第5章は、まとめである。

2. ジェットパイプインアーキテクチャ

2.1 ジェットパイプラインの構成

ジェットパイプラインの構成を図1に示す。ジェッ

† 東北大学大学院 情報科学研究科
Graduate School of Information Sciences, Tohoku University

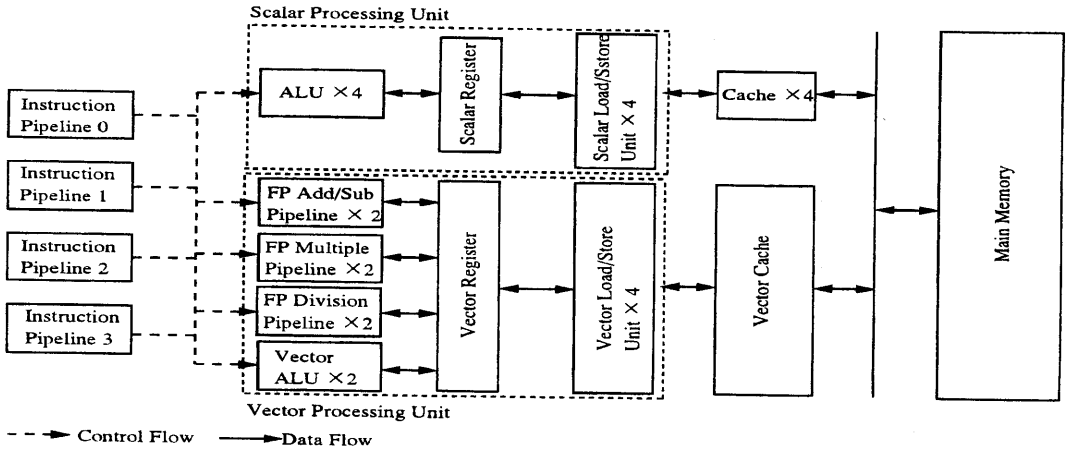

図1 ジェットパイプラインの構成
Fig. 1 Structure of Jetpipeline

トパイプラインは、4本の命令パイプラインによって、複数の処理ユニットを並列に動作させることができる。命令パイプラインに命令を発行するための並列化は、3章で述べるコンパイラによって行なわれる。処理ユニットは、ベクトルプロセシングユニットとスカラプロセシングユニットに大別できる。

ベクトルプロセシングユニットは、4つのベクトルロード/ストアユニット、2つのベクトル整数演算用ALU、2つの各浮動小数点演算用パイプライン、そしてベクトルレジスタから構成される。ベクトルレジスタの要素数は128であり、1ベクトル命令で128の演算、もしくはロード/ストアを実行することができる。ベクトル命令は、命令フェッチ、命令デコードを命令パイプラインで行い、その後の各ベクトルユニットの制御は、各ベクトルユニットの制御装置で行う。したがって、多くのサイクルに渡りベクトル命令が、命令パイプラインを占有することがないため、各命令パイプラインをストールさせる必要はない。

スカラユプロセシングニットは、4つのロード/ストアユニット、4つの整数演算用ALU、そしてスカラレジスタから構成される。したがって、ベクトル命令が実行されている間に、スカラプロセシングユニットで命令レベル並列処理を行うことが可能である。

2.2 ジェットパイプラインの演算性能

2.1節で述べた構成から、ジェットパイプラインは、ベクトル処理と命令レベル並列処理によって性能を向上させることが可能である。 図1から、ジェットパイラインの実行サイクル数T_Jは、式 (1) で示すことができる。

$$T_J = \frac{\alpha T_S}{\beta S_V} + \frac{(1-\alpha)\gamma T_S}{S_P} + (1-\alpha)(1-\gamma)T_S \quad (1)$$

式 (1) で、αはベクトル化率、βはベクトル処理の並列度、T_Sは逐次処理の実行サイクル数、S_Vはベクトル処理の速度向上率、γは並列化率、S_Pは命令レベル並列処理の速度向上率を表す。式 (1) から、ジェットパイプラインの速度向上率S_Jは式 (2) で表される。

$$S_J = \frac{T_J}{T_S} = \frac{1}{\frac{\alpha}{\beta S_V} + \frac{(1-\alpha)\gamma}{S_P} + (1-\alpha)(1-\gamma)} \quad (2)$$

式 (2) から、S_Jを向上させるためには、S_V、S_P、α、β、γを向上させればよいことがわかる。S_Vは1ベクトル命令あたりの平均速度向上率であり、その値は各ベクトル演算用パイプラインの性能によって決まる。したがって、演算パイプラインの性能が向上すればこの値も向上し、ジェットパイプラインの総合性能も向上すると考えられる。S_Pは、1サイクルあたりに実行可能な最大の命令数であり、命令パイプラインの数、および処理ユニットの数によって決まる。これまでの研究[7]により、プログラム中の命令レベル並列性が4以上になることは稀であるため、命令パイプラインの本数を4本以上に増やしても、性能向上は少ないと考えられる。

一方、α、β、γの値は、コンパイラによって向上させることができる。α、β、γの値を変化させた際の式 (2) から求めた速度向上率を**図2**、**図3**に示す。ただし、S_V、S_Pの値は一定である。 図2から、γの値は、αの値が小さい場合に速度向上率に影響を与えていることがわかる。 図3から、βの値は、αの値が大きい場合に速度向上率に影響を与えていることがわかる。したがって、ベクトル化率に拘らず効果的な速度向上率を得るためには、β、γを向上させるための並列化手法が重要になる。β、γは、プログラムの持つ並列性によって制限される場合がある。しかし、β、γをコンパ

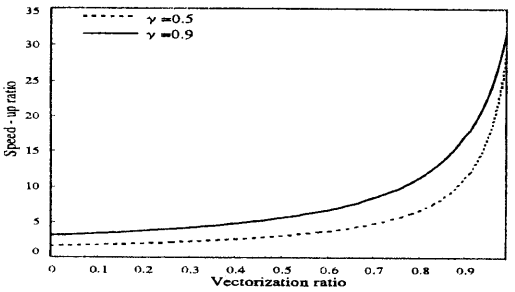

図2　ジェットパイプラインの速度向上率と並列化率
Fig. 2　Speed-up ratio of Jetpipeline and parallelize ratio

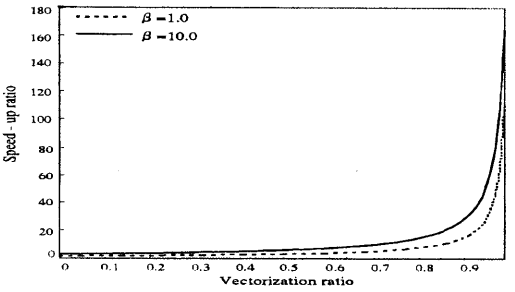

図3　ジェットパイプラインの速度向上率とベクトル並列化率
Fig. 3　Speed-up ratio of Jetpipeline and vector parallelize ratio

イラによって最適な値に近付けることは重要である。

3. ジェットパイプラインのためのコンパイラ

ジェットパイプラインのコンパイラは、字句解析、構文解析など通常のコンパイラが行う処理に加えて、ベクトル化、および並列化を行う。

ベクトル化では、ループに関する依存解析などを行うことにより、ベクトル化可能なループが決定される。そして、ベクトル化の対象になるループ内の演算に対して、ベクトル命令を生成する。

並列化では、命令間の依存関係を考慮して4本の命令パイプラインに命令を発行する。ジェットパイプラインでは、ベクトル命令とスカラ命令が混在したコードを並列化するため、ベクトル処理方式と命令レベル並列処理方式の両方の並列化技術を含んだ並列化手法が必要である。

本章では、ベクトル処理方式と命令レベル並列処理方式の並列化技術について考察し、ジェットパイプラインのための並列化手法を提案する。

3.1　ベクトル処理のための並列化技術

通常のベクトル計算機は、以下に示す命令スケジューリングを行っている[8]。

（1）　より多くのベクトル命令を並列実行させる。

```
for(i = 0; i < n; i++){
   a[i] = p * b[i] + q * c[i] + r * d[i];
}
```

図4　ベクトル化可能なループの例
Fig. 4　Example code able to vectorize

（2）　アドレス計算などのためのスカラ命令を、ベクトル命令と並列実行させる。

(1) は、依存のないベクトル命令や、チェイニングが可能なベクトル命令を並列実行させることを意味する。(2) は、スカラ命令のみを実行するとベクトル化率が低下し、全体の性能が低下するため、可能な限りスカラ命令をベクトル命令と並列実行させることを意味する。

(1)、(2) の要素は、ジェットパイプラインの並列化でも含まれる。このような並列化を 図4 に示すようなコードに適用すると、 図5 のようにスケジューリングされる。 図5 における最初のスカラ処理は、p、q の値をベクトルレジスタに代入するために、p、q の値をスカラレジスタへ代入している。配列の値をベクトルレジスタにロードするためのアドレス計算、およびストライドの計算は、他のベクトル命令と並列に実行されている。そのため、全体の実行サイクル数にカウントされるスカラ処理は、最初のスカラ処理のサイクル数のみである。したがって、 図5 に示すようにスケジューリングを行うと、見かけ上ベクトル命令だけが実行されているようになり、非常に効率的に処理される。また、 図5 から、チェイニングにより多くのベクトル命令が並列に実行され、全体の実行サイクル数が短縮されていることがわかる。

ジェットパイプラインでは、命令レベル並列処理をによって、 図5 中のスカラ処理の実行サイクル数を減少させることができる。通常のベクトルプロセッサでは、スカラ処理を逐次的に実行しなければならないため、スカラ処理の実行サイクル数が大きくなり、全体の実行サイクル数が増加する。この現象は、ベクトル化率の低いプログラムにおいてより顕著に見られる。したがって、ジェットパイプラインでは、次節で述べる命令レベル並列処理のための並列化技術も重要になる。

3.2　命令レベル並列処理のための並列化技術

本節では、スカラ命令のための並列化について考察する。スカラ命令の並列化では、より多くの命令を並列実行させることが重要になる。スカラ命令のための並列化技術の一つに、ディスパッチスタック法が挙げられる[6]。ディスパッチスタック法は、命令間の依存関係のみを考慮し、命令の種類に関係なく並列化を行う。したがって、そのアルゴリズムは簡潔であり、命令セットに拘らず幅広く適用できるため、スーパスカラプロセッサではハードウェアで実装されている。

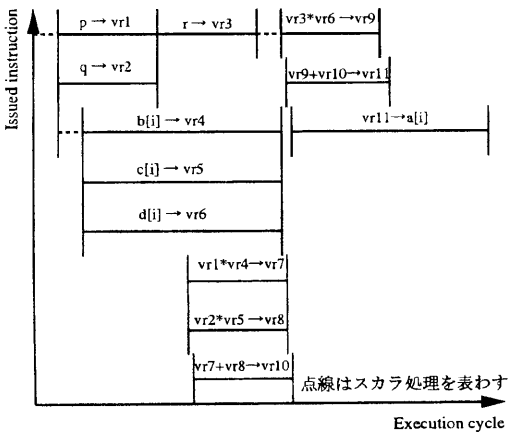

図5 理想的なスケジューリング
Fig. 5 Ideal scheduling for vector processor

以上の利点から、ジェットパイプラインでは、ディスパッチスタック法を用いて並列化を行う。しかし、ジェットパイプラインは、ハードウェアの複雑さをなくし、より高速な演算を達成するために静的な並列化を行うため、ディスパッチスタック法をソフトウェアで実現している。その利点は、スタックサイズの上限を設ける必要がないため、ハードウェアで実現する場合よりも多くの並列性を検出できることである。

3.3 ジェットパイプラインのための並列化技術

前節で述べたように、ジェットパイプラインでは、ディスパッチスタック法を用いて並列化を行っている。しかし、ディスパッチスタック法は、命令の種類を考慮しないため、多くの実行サイクル数を要するベクトル命令間の依存や資源競合を考慮するすることが不可能であった。したがって、従来のディスパッチスタック法による並列化では、 **図6** に示すような待ちループを挿入することにより、ベクトル命令の依存関係、および資源競合を解消してきた。図6中で、%vmcr はベクトルカウンタレジスタであり、各ベクトルユニット毎に備えられている。ベクトルカウンタレジスタの値は、ベクトル命令中の1つの演算を終える毎にデクリメントされ、その値がゼロになった時点でループ処理を終了する。したがって、ベクトル命令の実行が終了するまで、次の命令は実行されない。

しかし、並列化した後に待ちループを挿入することによって、余分な待ちループが挿入され、実行サイクル数が増加することがある。この理由は、並列化の際に同種のベクトル命令や、依存関係のあるベクトル命令が集中してスケジューリングされ、資源競合や依存関係を解消するために多数の待ちループが挿入されるためである。例えば、 図4に示すコードを従来のディスパッチスタック法を用いて並列化した後に、待ちループを挿入すると、 **図7** のようになる。 図7

```
    vfmul %vr,%vr2,%vr3 ;vector multiply
wl: cmpi  0,%vcmr       ;wating loop
    jc    nz,wl
    nop
```

図6 待ちループ
Fig. 6 Waiting loop

表1 ベクトルユニット資源表
Table 1 Vector unit resource table

vector unit	work	source1	source2	dest	cycle
VLOAD1	1	0	0	4	5
VLOAD2	1	0	0	5	5
・	・	・	・	・	・
・	・	・	・	・	・
VALU1	1	0	0	1	10
VALU2	1	0	0	2	10

で、並列化する際に、VALU を使用するスカラ値のベクトルレジスタへの代入命令が、3つ連続してスケジューリングされているため、r→vr3 を実行するために、待ちループが挿入される。その結果、後続の命令の発行が遅れている。また、並列化の際に、実行サイクル数を計算していないため、チェイニングのタイミングが不明になり、 図5の理想的なスケジューリングに較べて、多くの待ちループが挿入され、全体的に実行サイクル数が増加する。

リソース競合による余分な待ちループの挿入を避けるためには、並列化時にベクトルユニットの使用状況を把握し、競合を起こす命令の発行を遅らせ他の命令を発行すれば良い。依存関係のあるベクトル命令をチェイニングするためには、依存を起こすベクトル命令の実行サイクル数をカウントし、先行するベクトル命令の結果が1つ以上出力された時点で、後続のベクトル命令を発行する必要がある。そこで、これらのスケジューリングを達成するために、 **表1** に示すベクトルユニット資源表を本スケジューリングに導入する。表1中で、work はベクトルユニットが使用されていれば1になり、それ以外では0になる。また、source1、source2、dest は、ベクトル命令が使用するソースレジスタ、およびデスティネーションレジスタを示し、ベクトル命令間の依存関係を考慮する際に用いられる。cycle は、ベクトル命令の実行サイクル数を表し、命令パイプラインに命令が4つ発行される毎に1増加する。cycle の値は、チェイニングのタイミングの判定や、資源競合を避けるために使用される。ベクトルユニット資源表を用た並列化手法を以下に示す。

step1

通常のディスパッチスタック法と同様に、スタックから依存関係の解消された命令を選出する。

step2

発行可能な命令から、ベクトル命令を選出する。

step3

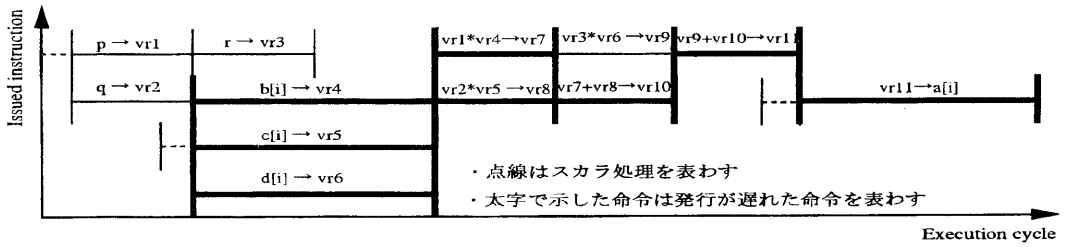

図7　並列化後の待ちループ挿入
Fig. 7　Waitinig-loop insertion after parallelizing

発行するベクトル命令が使用する資源が、使用されていないかを確認する。もし、使用されていれば、そのベクトル命令はスタック内に残される。

step4

発行するベクトル命令と依存のある(正依存)ベクトル命令が実行されている場合、先行する命令のcycleをベクトルユニット資源表で調べる。cycleの値が、ベクトルユニットのスタートアップタイム以上であれば、後続のベクトル命令を発行する。cycleの値が、スタートアップタイム以下である場合、後続のベクトル命令をスタック内に残す。4本の命令パイプラインに命令が発行された場合、workが1であるベクトルユニットのcycleを1増やす。

step5

step1で選出した命令中にベクトル命令が残っていれば、step2から繰り返す。

step6

スカラ命令を発行し、スタック内にスカラ命令が残っていればstep1から繰り返す。4本の命令パイプラインに命令が発行された場合、workが1であるベクトルユニットのcycleを1増やす。

step7

スタック内にベクトル命令だけが残っている場合、それらのベクトル命令に対する依存関係や資源競合を、待ちループ、もしくはnopを挿入して解消し、残っているベクトル命令を発行する。4本の命令パイプラインに命令やnopが発行された場合、workが1であるベクトルユニットのcycleを増やす。待ちループが挿入された場合、ベクトルユニット資源表から待ちループのサイクル数を計算し、workが1であるベクトルユニットのcycleを増やす。

以上の並列化を図4に適用すると、図5とほぼ同じように並列化される。

4. 性能評価

本稿では、提案した並列化手法の有効性を確認するために、ベンチマークによるシミュレーションを行なった。使用したベンチマークは、科学技術計算に多く見られるプログラムを集めたリバモア・フォートラン・カーネルの1〜9である。シミュレーションは、ジェットパイプラインのソフトウェアシミュレータ上で行なった。**表2**に各ベンチマークの特徴を示す。一般的に、表2中のベクトル化率、ループ長、演算数は、大きければ、ベクトル処理の効果が増加する。

表2　ベンチマークの特徴
Table 2　Charactristics of Livermore Fortran Kernel

	ベクトル化率	ループ長	演算数
kernel 1	0.99	1000	中
kernel 2	0.97	1000〜1	少
kernel 3	0.55	1000	少
kernel 4	0.53	1000	少
kernel 5	0.00	1000	-
kernel 6	0.00	1000	-
kernel 7	0.99	1000	多
kernel 8	0.99	1000	多
kernel 9	0.97	1000	多

図8に各並列化手法の速度向上率、**図9**に各並列化手法におけるベクトル並列化率を示す。ベクトル並列化率は、式(2)中のβの値を示す。図中のSequentialは命令パイプラインが1本でスカラ命令の並列化が不可能な場合を示し、Dispatch stackは従来のディスパッチスタック法による並列化を示し、Dispatch stack + Resource tableは本稿で提案する並列化手法を示す。Vectorはベクトル処理を示し、Unrollingはループアンローリングを適用して、ベクトル命令、およびスカラ命令の命令レベル並列性を向上させる場合を示す。

Dispacth stack + Vectorの速度向上率は、ベクトル化率の大きいプログラムにおいて、Sequential + Vectorとの差はほとんど見られない。この理由は、ベクトル命令が効果的に並列化されていないためであると考えられる。したがって、図9から、Dispatch stack + VectorとSequential + Vectorのベクトル並列化率の間に差が見られない。しかし、ベクトル化率が低いプログラム、およびベクトル化不可能なプログラムでは、Sequential + Vectorよりも大きい速度向上率を示している。この理由は、ベクトル化率が低いために、命令レベル並列処理の効果が大きくなった

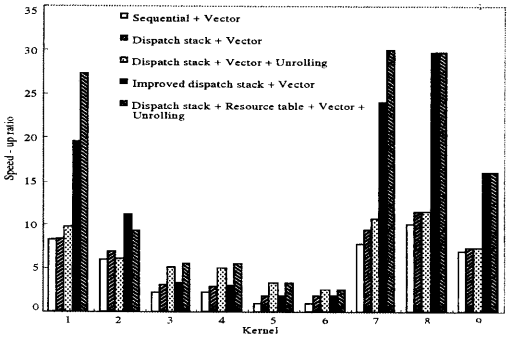

図8　速度向上率
Fig. 8　Speed-up ratio

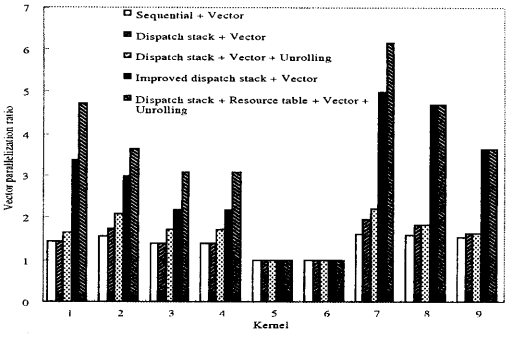

図9　ベクトル並列化率
Fig. 9　Vector parallelization ratio

ためであると考えられる。したがって、ジェットパイプラインは、ベクトル化率が低いプログラムでも、効果的な処理が可能であることがわかる。

Dispatch stack + Resource table + Vector の速度向上率は、ベクトル化率の高いプログラムにおいて、Sequential + Vector、Dispatch stack + Vector と比較して非常に大きい値になっている。この理由は、本稿で提案する並列化手法によって、ベクトル並列化率が向上したためであると考えられる。このことは、図9からも明らかである。図8中のKernel1、およびKernel7の結果から、Dispatch stack + Resource table におけるループアンローリングの効果が、Dispatch stack よりも大きいことがわかる。したがって、本稿で提案する並列化手法は、ベクトル処理の並列性を効果的に引き出すことが可能であることがわかる。また、ベクトル化率の低いプログラム、およびベクトル化不可能なプログラムでも、Dispatch stack + Vector とほぼ等しい速度向上率を示している。したがって、本稿で提案する並列化手法が、ベクトル処理とスカラ処理の両方における並列性を、効果的に引き出していることがわかる。

以上の考察から、ジェットパイプラインは、ベクトル化率に拘らず効果的な処理が可能であることがわかる。しかし、ベクトル化率の低いプログラムにおける速度向上率は、ベクトル化率の高いプログラムの速度向上率よりも低い。したがって、ベクトル化不可能なスカラ部分に対して、ソフトウェアパイプライニング[5]のような高度な並列化手法を適用することによって、より高速な処理を実現することが今後の課題として挙げられる。

5. ま と め

本稿では、ジェットパイプラインのベクトル処理のための効果的な並列化手法を提案した。シミュレーションの結果から、ベクトル化率の高いプログラムにおいて、これまでの並列化手法よりも多くの並列性を検出し、速度向上率を増加させることができた。しかし、ベクトル化率の低いプログラムでは、速度向上率の向上は見られない。したがって、ベクトル化率の低いプログラムをより高速に処理するために、より効果的な並列化手法を開発する必要がある。

参 考 文 献

1) Russell, R.M.: The CRAY-1 computer system, *Communication of the ACM*, Vol.21, No.1, pp. 63–72 (1978).
2) マイクジョンソン: スーパースカラプロセッサ, 日経BP 出版センター (1994).
3) Fisher, J. A.: Very long instruction word architectures and the ELI-512., *In Proceedings of the 10th Annual Symposium on Computer Architectures*, pp. 140–150 (1983).
4) Katahira, M., Shen, H., Kobayashi, H. and Nakamura, T.: A hybrid architecture for instruction level parallelism., *In Proceedings of the High Performance Computing Conference '94*, pp. 317–323 (1994).
5) Katahira, M., Sasaki, T., Shen, H., Kobayashi, H. and Nakamura, T.: Software pipelining for jetpipeline architecture, *In Proceedings of the International Symposium on Parallel Architectures,Alogorithms,and Networks*, pp. 127–134 (1994).
6) Acosta, R. D., Kjelstrup, J. and Torng, H. C.: An instruction issuing approach to enhancing performance in multiple function unit processord, *IEEE Transaction on Computers*, Vol.35, No. 9, pp. 815–828 (1986).
7) Jouppi, N. P.: The nonuniform distribution of instruction-level and machine parallelism and its effect on performance, *IEEE Transaction on Computers*, Vol. 38, No. 12, pp. 1645–1658 (1989).
8) 長島重夫, 田中義一: スーパコンピュータ, オーム社 (1992).