社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－113　（２）
2004／1／22

資源共有型VLIWプロセッサの性能評価

荒本　雅夫[†]　湯山　洋一[†]　樋口　昭彦[†]　岡澤　潤香[††]　小林　和淑[†††]
小野寺秀俊[†]

† 京都大学大学院情報学研究科　〒606-8501 京都市左京区吉田本町
†† 日本IBM　〒520-2392 滋賀県野洲群野洲町大字市三宅800
††† 東京大学大規模集積システム設計教育研究センター　〒113-8656　東京都文京区弥生2-11-16
E-mail: †{masao,yuyama,higuchi,junka,kobayasi,onodera}@vlsi.kuee.kyoto-u.ac.jp

あらまし　我々はSMT技術をVLIWプロセッサに適用した資源共有型VLIWプロセッサを提案する．変数の依存関係の解析はコンパイラで静的に行い，実行資源の割り当てはハードウェアで動的に行う．従って，ハードウェアは単純なまま，実行資源あたりの処理能力を向上させることができる．評価実験の結果，4つの実行資源がある場合，通常のプロセッサよりも資源使用効率を2.6倍に向上させることができる見通しが得られた．
キーワード　VLIW，SMT，資源効率

Performance Evaluation of a Resource-Shared VLIW Processor Array

Masao ARAMOTO[†], Yoichi YUYAMA[†], Akihiko HIGUCHI[†], Junka OKAZAWA[††],
Kazutoshi KOBAYASHI[†††], and Hidetoshi ONODERA[†]

† Graduate School of Informatics, Kyoto University
†† IBM Japan
††† VLSI Design and Education Center, The University of Tokyo
E-mail: †{masao,yuyama,higuchi,junka,kobayasi,onodera}@vlsi.kuee.kyoto-u.ac.jp

Abstract　We propose a Resource-Shared VLIW Processor Array which is a VLIW processor on the SMT technology. In the proposed processor array, a compiler (software) statically analyzes the dependency of a variable , and a hardware dynamically assigns execution resources. Therefore, hardware can be utilized as much as possible while its structure becomes very simple. Experimental results show that hardware resources of the proposed processor are used 2.6 times more frequent than a conventional VLIW processor when four parallel resources are prepared.
Key words　VLIW, SMT, resource utilization

1. はじめに

近年，高度な機能を持つ組み込み機器が普及してきている．例えば携帯電話では通話だけではなく様々なアプリケーションを実行できることが求められている．現在，組み込み機器の機能を実現するためには専用回路(ASIC)を設計することが一般的である．しかし設計の長期化，製造コストの増大，バグ発生率の増加などという問題により，今後も専用回路で対処することが困難になりつつある．従って，様々な機能の組み込み機器を実現するためには今後，高性能で設計後に機能を変更することができるデバイスが求められると考えられる．このようなデバイスとしてはプロセッサ[1,2]，リコンフィギュラブルデバイス[3,4]，FPGA[5,6]などが挙げられる．

一般的にプロセッサは専用回路よりも性能が低い．プロセッサの性能を向上させるためには，動作周波数を向上させる方法と，アーキテクチャを工夫して並列度を向上させる方法が考えられる．プロセッサの動作周波数を高くするためには，電源電圧を高くしなければならなくなり，性能あたりの電力が増加する．特に携帯の組み込み機器に使用するためには，低消費電力化は非常に重要な制約となる．一方，並列度を向上させる方法では，プログラムから得られる並列度が低いと，活用できないハードウェア部が現れ，資源の使用効率が低下するという問題がある．

我々はこの問題に対処するために資源共有型VLIWプロセッサを提案する．これはVLIWにSMT(Simultaneous Multi-Threading)技術[7–9]を適用したものである．我々の提案する

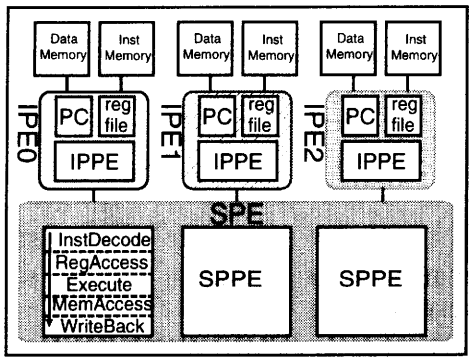

図 1　資源共有型 VLIW プロセッサの概要

プロセッサでは，複雑な並列性の解析はコンパイラで静的に行う．コンパイルの結果，並列性が引き出せず空き資源が発生する場合は別タスクの処理を同時に行い，資源を有効利用する．提案プロセッサは，ソフトウェア，ハードウェアの一方で並列性を解析する従来のプロセッサの中間のアーキテクチャであるといえる．本稿では資源共有型 VLIW プロセッサのアーキテクチャと，そのアーキテクチャで実現可能な性能について検討する．

本稿は以下のような構成になっている．2 章では資源共有型 VLIW の概要について述べる．3 章では性能評価のためのシミュレーション方法について述べる．4 章ではその実験結果について述べる．最後に 5 章で結論を述べる．

2. 資源共有型 VLIW

本章では，資源共有型 VLIW プロセッサの概要について述べる．まず資源共有型 VLIW プロセッサのアーキテクチャと動作の概要について述べる．次に提案プロセッサが従来のプロセッサよりも優れている点について説明する．

2.1 アーキテクチャの概要

資源共有型 VLIW プロセッサのアーキテクチャの概要を図 1 に示す．このプロセッサは主に以下の構成要素から成り立っている．

- IPE : Independent Processor Element
  プロセッサ毎に独立な資源
- SPE : Shared Processor Element
  プロセッサ間で共有可能な資源
- PPE : Pipelined Processing Element
  命令の実行に必要な資源

独立な資源である IPE には命令フェッチ部やレジスタファイルが含まれる．PPE は命令デコーダ，実行ユニットが含まれる．PPE の内部はパイプライン構成になっており，1 サイクル毎に異なるスレッドの命令を処理することできる．従って IPE に接続する SPE 内の PPE 数を変化させることで，プロセッサの並列度を動的に変化させることができる．なお，IPE 内の PPE を IPPE，SPE 内の PPE を SPPE と呼ぶ．

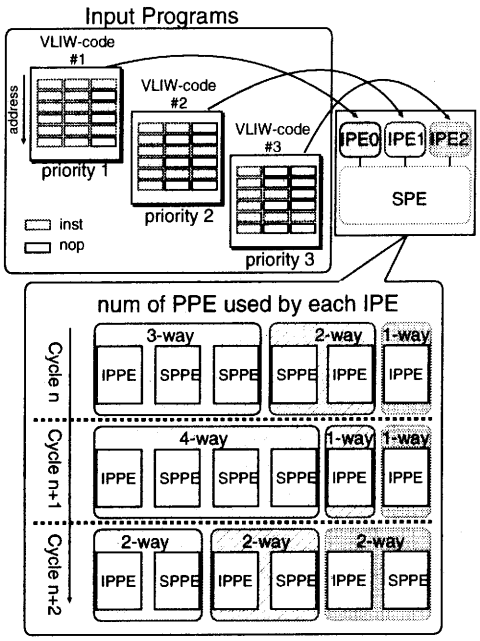

図 2　資源共有型 VLIW プロセッサの動作の概要

図 2 に，資源共有型 VLIW プロセッサが動作する様子を示す．プロセッサへ入力するバイナリコードは VLIW の形式で与えられる．つまり，変数の依存関係などはあらかじめ静的に解決されている．このような VLIW コードをそれぞれの IPE に与える．プロセッサは入力された VLIW コードに従って実行可能な命令を同時に実行する．プログラムのコンパイル時に変数の依存関係が解消できず並列性が引き出せなかった場合，VLIW コードには NOP(No OPeration) 命令が挿入される．NOP 命令はプログラムの実行には全く影響を与えない．従って実行資源があるにもかかわらず，資源を活用できなくなる．あるタスクで NOP 命令があった場合，SPPE を別のタスクの処理に使用することで従来では無駄になっていた資源を有効利用することができる．このような技術を SMT(Simultaneous MultiThreading) という [7–9]．実行資源を共有しているので，複数のタスク間で資源競合が発生する可能性がある．そのような場合はタスク毎に決められている優先度を利用する．優先度の高いタスクは優先的に資源を利用でき，優先度の低いタスクは 1 サイクル待ち，次のサイクルで命令を実行する．

2.2 提案アーキテクチャの利点

提案アーキテクチャが従来のプロセッサよりも優れていると考えられる事項について説明する．

- **ハードウェア，ソフトウェアの両方で最適化可能**

スーパースカラプロセッサでは並列性の解析をハードウェアで動的に行うために，従来のコンパイル技術が流用できるという利点がある．しかし，ハードウェアコストや消費電力が高くなるという問題がある．VLIW プロセッサでは並列性をコンパ

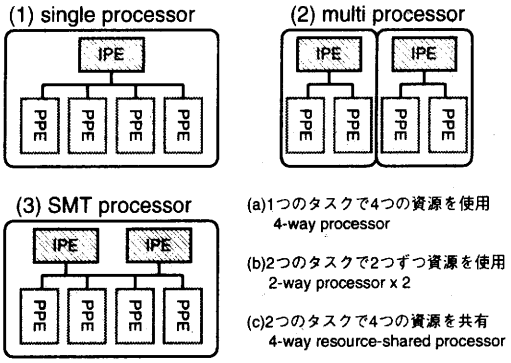

図 3　実行資源の使用方法の違い

表 1　実行資源の使用方法の違いによる性能変化

評価モデル	single processor		multi processor		SMT processor	
	Cycle	Util	Cycle	Util	Cycle	Util
タスク 1	1130	30%	1324	63%	1130	70%
タスク 2	1723	54%	2859	58%	2266	70%
合計	3000	42%	2859	60%	2266	70%

イラで静的に決定するために，ハードウェアが単純になるという利点がある．しかし，コンパイル技術が複雑になる．コンパイラで並列性を充分に引き出せなければ，プログラム実行時のハードウェア資源がうまく活用できないという問題がある．

我々の提案するプロセッサは，変数の依存関係の解析はコンパイラで静的に行う．そのために従来のスーパースカラのような複雑なハードウェアが不要となる．変数の依存関係が解消された入力プログラムに対し，実行資源の割り当てはハードウェアで動的に行う．そのために従来の VLIW プロセッサでは活用できなかった空き資源をうまく活用することができる．

このように提案プロセッサはハードウェアとソフトウェアの両面から最適化することができるのが利点の一つである．

- **資源の使用効率を改善可能**

例として，4 つの実行資源を，図 3 のように

（1） 4-way プロセッサ (シングルプロセッサ)

（2） 2-way プロセッサ ×2 個 (マルチプロセッサ)

（3） 4-way 資源共有型プロセッサ (SMT プロセッサ)

として利用する場合について考える．

これらのプロセッサで，ある 2 つのタスクを実行する際の実行時間と資源の使用効率を表 1 に示す．ただし，(1) の 4-way プロセッサはマルチタスク OS の機能を利用し，タスクスイッチにより 2 つのタスクを実行すると仮定する．(1) の 4-way プロセッサでは，それぞれのタスクの実行時間自体は短いが，1 度に 1 タスクしか実行できない．そのため，タスク全体の実行時間はそれぞれのタスクの実行時間に，タスクスイッチのオーバーヘッドを加えた 3000 サイクルとなる．また，命令発行幅が広くなると，プログラムから並列度を引き出すことが難しいので，資源使用効率が低い．(2) の 2-way プロセッサ ×2 個では，それぞれのタスクを同時に実行する．タスク全体の実行時間は実行時間が長いタスク 2 の 2859 サイクルとなる．(1) の場合よりも命令発行幅が狭く並列性の解析が簡単になるために，資源の使用効率も (1) よりはよい．(3) の資源共有プロセッサでは，タスク 1 は (1) と同じ速度で実行できるが，もう一方のタスクに必要な時間は (1) より約 500 サイクル増加する．しかしタスク全体で考えると実行時間は 2266 サイクルとなり，3 つの中で最も高速となる．資源使用効率についても 3 つの中で最もよい．

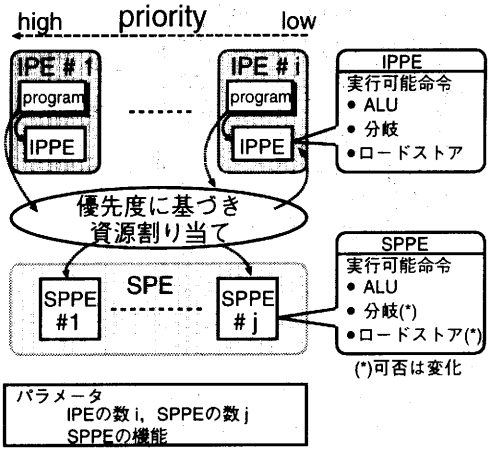

図 4　資源共有型 VLIW プロセッサのシミュレーションモデル

このように，プロセッサの実行資源を複数のタスク間で共有して使用すると，実行資源の使用効率が向上し，合計のタスクの実行時間が短くなる．

- **あるタスクだけを高速処理可能**

(3) では，一方のタスクから 4 命令同時に発行可能の場合，4 つの実行資源を使用することも可能である．(2) の場合は，1 つのタスクで 2 つの実行資源のみしか使えないためにそれ以上の高速化は不可能である．

3. シミュレーション条件

本章では資源共有型 VLIW プロセッサのシミュレーションの条件について述べる．最初に，資源共有型 VLIW プロセッサのシミュレーションモデルについて述べる．次に入力として用いるデータの生成方法について説明する．

3.1 資源共有型 VLIW プロセッサのモデル

図 4 に実験に用いる資源共有型 VLIW のシミュレーションモデルを示す．これは C++で記述しており，システムの詳細な構造は特に意識していない．IPE モジュールにはプログラムファイルと IPPE が 1 つ存在する．サイクル毎に発行可能な命令を調べ，発行可能命令数を外部に出力する．IPE モジュール毎にある優先度は，1 番目のものが最が高く，i 番目のものが最も低い．スケジューラは複数の IPE からの処理要求を受けとり，IPE の優先度と SPE の機能に応じて，IPE に命令発行許可を出す．SPE は中に存在する SPPE の個数によって 1 サイクルに処理できる命令数が変化する．もし資源が足りない場合は，優先度の高い IPE の命令を実行する．優先度の低い IPE

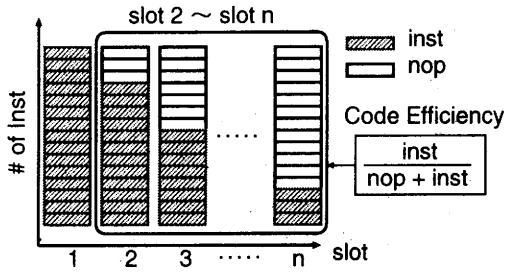

図 5　仮想的な命令コード

は 1 サイクル待ち，次サイクルで命令を実行する．IPPE ではすべての命令が実行可能とする．SPPE では分岐，ロードストア命令の実行の可否はパラメータで決める．

このシミュレータにおいて，IPE の数，SPPE の数，SPPE の機能を変化させて，共有資源 SPE 全体の IPC(Instruction Per Cycle) と資源の使用効率を評価する．

3.2　仮想的な入力データ

提案アーキテクチャの性能は入力プログラム (バイナリコード) の命令がどれくらい詰まっているか (効率がいいか) に依存する．そこで評価用に命令の詰まり具合を自由に変更できる仮想的なデータを用いる (図 5)．このデータの命令の詰まり具合を表わす指標として，コード効率というものを定義する．1 番目のスロットの命令は必ず IPPE で実行される．2 番目以降のスロットの命令は SPPE で実行される可能性がある．従って SPE の性能を評価する際には，2 番目以降のスロットの命令の詰まり具合が重要になる．仮想的なデータは 1 番目のスロットの命令数を slot1 とすると式 1 のようにスロットが進むにつれて徐々に命令数が少なくなると仮定する．

$$\mathrm{code}(n) = \mathrm{slot1} \times a^{n-1} (0 < a < 1) \qquad (1)$$

このような命令コードにおいて，2 番目以降のスロットの命令数の割合をコード効率と定義する．

3.3　プログラムのコンパイル方法

実験には C 言語で記述されたプログラムを用いる．まず，C 言語のプログラムを GCC でコンパイルする．その結果のアセンブリ列を解析し，同時に実行可能な命令を VLIW の形式で出力する．今回は簡単のために，インオーダでスケジューリングを行う．命令を分岐命令，メモリアクセス命令，その他の命令に分類し，SPPE で実行できる命令の制約に従って並列性を解析する．テストプログラムとして 7 種類のプログラム (ave, bubble, fir, heap, idct, matrix, qsort) を用いる．これらのプログラムをコンパイルし，IPE が 1(つまり資源を共有しない通常のプロセッサ) のモデルを用いてシミュレーションした．結果の IPC を図 6 に，実行資源の使用効率を図 7 に示す．命令発行幅が増加すると IPC は増加する．しかし，プログラムから得られる並列性には限界があるために IPC は徐々に飽和し，ある値に収束する．従って，逆に実行資源の使用効率は低下する．

今回，スケジューリングには単純なアルゴリズムしか用いていない．アウトオブオーダスケジューリングやトレーススケジューリング [10] などを行うと並列度は改善できると思われるが，並列度は最大でも 7 程度であると言われている [11]．

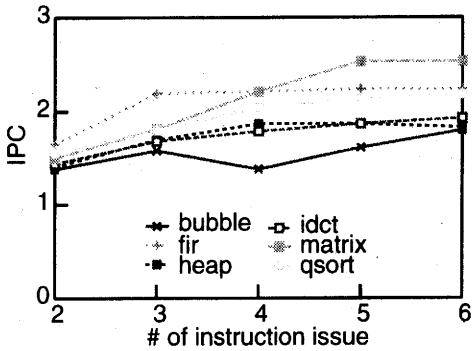

図 6　命令発行数と IPC の関係

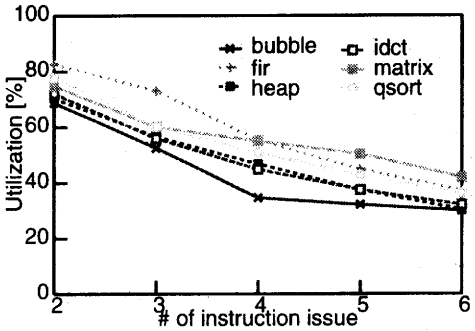

図 7　命令発行数と実行資源の使用効率の関係

4.　実験結果

本章では共有資源 SPE の使用効率に関する実験結果について述べる．最初に仮想的なデータを用いた評価を行い，提案プロセッサの性能を評価する．次に実際のプログラムを用いた評価を行う．最後に実行資源 SPPE の機能を限定した場合の性能について考察する．

4.1　仮想的なデータを用いた評価

まずは，命令の詰まり具合を自由に変化させて評価を行った．実験は入力の仮想的なデータのコード効率が低い場合 (25%) と高い場合 (75%) の 2 種類について行った．図 8 に SPE 全体の IPC を示す．また図 9 に SPE 全体の資源使用効率を示す．

コード効率が 25%，IPE 数が 1 のとき，SPPE 数が増加するにつれて，SPE の IPC は比例的に増加する．これは本実験ではコード効率を一定にして評価しているためである．しかしプログラムのコード効率が低いために，SPPE が 8 の場合でも IPC は 2.0 程度にしかならない．

一方，IPE 数を増加する，つまり複数の IPE で実行資源を共有すると IPC が向上している．それに伴い，資源の使用効率も向上している．例えば，SPPE が 4 の場合，IPE を 1 から 4 に増加すると，IPC は 1.0 から 3.0 へと約 3 倍向上する．同様にコード効率が 75%の SPPE が 4 の場合，IPE を 1 から 4 に増加させると，IPC は 3.0 から 3.9 へと約 1.3 倍向上する．

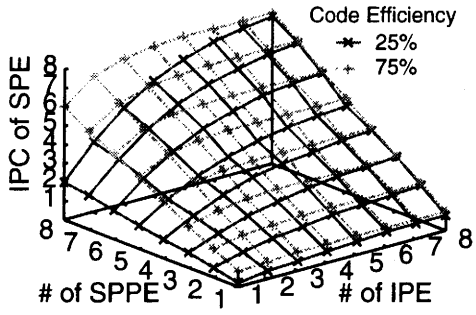

図 8　仮想的なデータを用いた際の IPC の変化

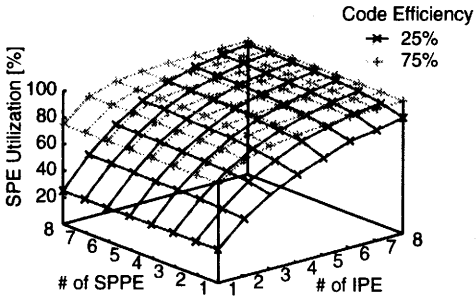

図 9　仮想的なデータを用いた際の使用効率の変化

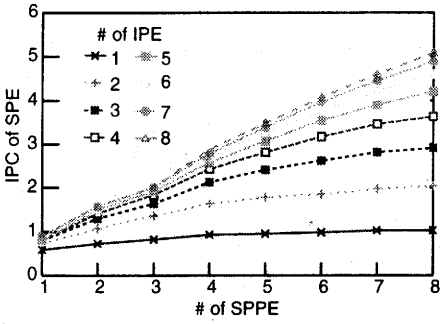

図 10　実プログラムを用いた際の IPC の変化

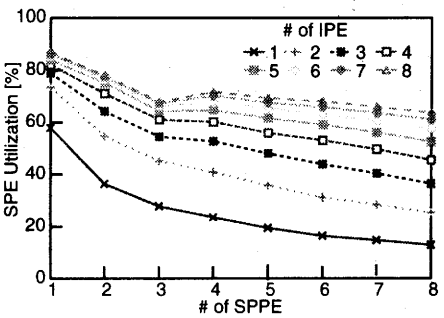

図 11　実プログラムを用いた際の使用効率の変化

コード効率が高い場合は IPC，資源の使用効率がはじめから高いために，IPE 数の増加により SPE の使用効率は向上はするが効果は少ない．

この結果から提案プロセッサは特にコード効率が低い場合に効果的である言うことができる．

4.2　実プログラムを用いた評価

実際のプログラムを用いて評価を行った．本実験では，SPPE は全ての命令を実行可能であるとした．また，入力プログラムには 3.3 節で述べたものを用いる．これらのプログラムを複数の IPE で様々な組み合せで実行した結果の平均値を実験結果として示す．SPE の IPC の変化を図 10，SPE の使用効率の変化を図 11 に示す．これらの図では，横軸で SPPE の数，グラフ中の点で IPE の数を表現している．図 10 を見ると，IPE 数が 1 の場合，SPPE 数を 4 以上増加させても IPC があまり向上していない．これは入力プログラムをコンパイルした結果の並列度が低かったからである (図 6，図 7)．SPPE 数を増加させても IPC は一定なので，追加した SPPE が活用されることはなく，SPE の使用効率は悪化する (図 11)．しかし，このような入力プログラムに対してでも，IPE 数を増加させ，複数のタスクで実行資源を共有することで，IPC が向上している．例えば，SPPE が 4 の場合，IPE を 1 から 4 個に増加させると，IPC は 0.9 から 2.4 へと約 2.6 倍向上する．

3.3 節で述べたように，一般的にプログラムの並列度を向上させるために命令発行幅を増加させると，プログラムのコード効率が悪化する傾向がある．提案プロセッサはそのようなコード効率が低くなったプログラムに対して効果的であると言える．

4.3　実行可能命令の制限による影響

前節の実験では，SPPE ですべての命令が実行可能であると仮定した．しかし，分岐命令を実行できる SPPE を制限しないと信号配線が複雑になる可能性がある．また複数のロード ストア命令を許可するとメモリのポート数が増加するのでハードウェアコストが増加する．このように，分岐命令とロード・ストア命令はハードウェアの構成に大きく影響を及ぼす．本節では，以下のように SPPE で実行できる演算を制限することにより，どの程度 SPE の使用効率が劣化するかについて評価した．

（1）　すべて実行可能 (4.2 節)

（2）　分岐命令不可

（3）　ロードストア命令不可

（4）　分岐命令，ロードストア命令不可

(2)～(4) の場合の SPE 使用効率を (1) の使用効率を 100%として換算したものを図 12 に示す．

まず同じ IPE 数で比較する．例として IPE が 4 の場合を図 13 に示す．命令発行幅が狭い場合に分岐命令を制限するとプログラムの並列性が大きく制限される．そのために，分岐命令を制限すると制限なしの場合の 85%の使用効率となる．さらに命令発行幅が広くなると，並列性はプログラム内の変数の依存関係によって制限されてくるために，分岐命令の制限による影響が少なくなる．メモリアクセス数はプログラムの制御の流れには影響しない．しかし，命令発行幅が広くなると同時に実行するべきロードストア命令が増加するためにメモリアクセス数の制限による影響を受けるようになる．従って，命令発行幅が広くなると分岐命令の制限よりも大きな影響を受けるようになり

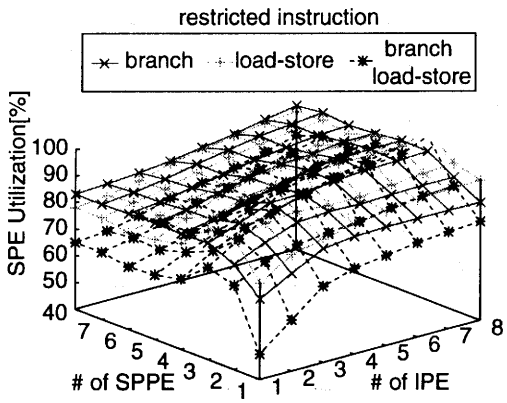

図 12　実行資源の機能制限による使用効率の変化

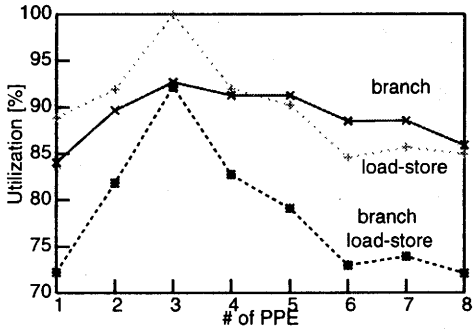

図 13　実行資源の機能制限による使用効率の変化 (IPE 4)

表 2　実行資源の機能制限による使用効率の変化 (SPPE 4)

	制限する命令		
IPE 数	分岐	ロードストア	分岐，ロードストア
1	83.3%	79.6%	66.4%
4	91.2%	91.6%	82.7%
8	96.1%	94.0%	91.0%

84%の使用効率となる．このように性能に最も影響のある命令は，SSPE 数によって分岐命令にもロードストア命令にもなりうる．ただし，分岐命令，ロードストア命令の両方を制限した場合両者の影響を強く受け，最大 72%程度にまで性能が劣化する．一方，同じ SPPE 数で比較すると，IPE 数が多いほうが性能が高いことがわかる．例として，SPPE 数が 4 の場合を表 2 に示す．分岐命令とロードストア命令を制限した場合，IPE が 1 では使用効率は 66.4%であるが，IPE を 4，8 と増加させることにより 82.7%，91.0%へと向上させることができる．

このように，実行資源 SPE を共有しない場合は実行資源の機能制限の影響を大きく受ける．しかし，複数の IPE 間で SPE を共有することにより，SPPE の機能制限の影響を小さくすることができる．

5. 結　　論

並列度の高いプロセッサの資源使用効率を高めるために，実行資源を共有した資源共有型 VLIW プロセッサを提案した．資源共有型 VLIW プロセッサでは，複雑な並列性の解析はコンパイラで静的に行い，実行資源の割り当てはハードウェアで動的に行う．そのため，ハードウェア，ソフトウェアの両方で最適化が可能，資源使用効率の向上という利点が見込める．

実験ではアーキテクチャの規模を拡大したときの共有資源 SPE の使用効率について評価を行った．その結果，プログラムのコード効率が高い場合は効果は少ないが，コード効率が低い場合は SPE の使用効率を大きく改善することができることがわかった．実際のプログラムではコンパイルの結果得られるコード効率は低く，共有資源 SPPE 数が 4 の場合，IPE 数を 1 から 4 に増加させることで IPC を 0.9 から 2.4 へと 2.6 倍向上させることができる．実行資源の機能を限定する場合，SPPE 数が少ないときは分岐命令の制限による影響が大きく，SPPE 数が多いときはメモリアクセス数の制限による影響が大きい．しかし，複数の IPE 間で SPE を共有することにより，機能の制限による影響を少なくすることができる．

今後はこのようなアーキテクチャを実装し，さらに詳細な評価を行う必要がある．また，ソフトウェア技術に関して検討することも必要である．現在はインオーダでスケジューリングを行った結果を用いて評価を行った．さらに複雑なアルゴリズムを用いてスケジューリングした結果を用いても，同様な結果が得られるか評価する必要がある．さらにハードウェアの面積，消費電力などについても検討する必要がある．

文　　献

[1] ARM7 THUMB FAMILY. `http://www.arm.com`

[2] MIPS32 K4 Family. `http://www.mips.com`

[3] Herman Schmit, David Whelihan, Andrew Tsai, Matthew Moe, Benjamin Levine, and R. Reed Taylor. PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology. In *the IEEE Custom Integrated Circuits Conference*, Vol. 36, pp. 63–66, 2002.

[4] M. Motomura. A Dynamically Reconfigurable Processor Architecture. In *Microprocessor Forum*, Oct 2002.

[5] `http://www.altera.com/`

[6] `http://www.xilinx.com/`

[7] Hiroaki Hirata, Kozo Kumura, Satoshi Nagamine, Yoshiyuki Mochizuki, Akio Nishimura, Yoshimori Nakase, and Teiji Nishizawa. An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 136–145, 1992.

[8] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *The 22rd Annual International Conference on Computer Architecture*, pp. 392–403, Jun 1995.

[9] 中條拓伯, 河原章二, 上原哲太郎, 並木美太郎. Simultaneous Multithread(SMT) アーキテクチャの現状と今後. *IPSJ Magazine*, Vol. 43, No. 3, pp. 281–7, Mar 2002.

[10] P. Geoffrey Lowney, Stefan M. Freudenberger, Thomas J. Karzes, W. D. Lichtenstein, Robert P. Nix, John S. O'Donnell, and John C. Ruttenberg. The Multiflow Trace Scheduling compiler. *The Journal of Supercomputing*, Vol. 7, No. 1-2, pp. 51–142, 1993.

[11] D. W. Wall. Limits of instruction-level parallelism. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, Vol. 26, pp. 176–189, 1991.