計算機アーキテクチャ 129－3
（1998．5．15）

キューマシン計算モデルに基づく スーパスカラ・プロセッサの設計

岡本秀輔† 前田敦司† 曽和将容†

キューマシン計算モデルは、キューの先頭にあるオペランドを取り出して、演算を行い、結果をキューの末尾に挿入する形で計算を進めるモデルである。これまでにこの計算モデルの特徴を利用して、関数型言語における並列関数呼び出しを高速化する研究がなされて来た。本報告ではこれに対して手続き型の高級言語を対象として、モデルを式の評価に適用し、手続き呼び出しによって作られる木をソフトウエア・スタックで管理するスーパスカラ・プロセッサの設計について述べる。まず、キューマシン計算モデルの概要と特徴についてまとめた後に、このモデルを指向する命令セットアーキテクチャ、デコードと命令発行の原理を説明する。ついで高速な切り替えを可能とするマルチスレッド実行および割り込み処理の可能性についてまとめる。

SuperScalar Processor based on Queue Machine Computation Model

SHUSUKE OKAMOTO,† ATUSI MAEDA† and MASAHIRO SOWA†

The queue machine model of execution is an evaluation scheme for expression trees, in which input operands of operations are taken from head of queue, and its result is put onto tail of queue. The previous study with this model is the automatic parallel execution of functional programming languages. In this paper, we describe a design of superscalar processor. Our processor adopts this model to calculate expressions, and it adopts usual software stack for procedure calls. We also describe a possibility of fast threads switch on our processor.

1. はじめに

キューマシン計算モデル[5]は、スタック・マシンのそれと対照的にキューを用いて計算を進めていく。スタック・マシンでは、スタックの先頭にあるオペランドを取り出して演算を行い、その結果を再びスタックの先頭に置く。これに対して、このモデルではキューの先頭にあるオペランドを取り出して、演算結果をキューの末尾に挿入する。

これまでにこの計算モデルの特徴を利用して、関数型言語における複数関数呼び出しの並列処理を高速化する研究がなされて来た[5),6)]。並列関数呼び出しによって動的に作られる木を、スタックではなくキューで管理することにより、密結合型並列計算機において高い性能効果を得ている。この方式ではレジスタ・マシンである従来型のプロセッサを用いて式の評価を行ない、並列関数呼び出しによる木をキューで処理する。

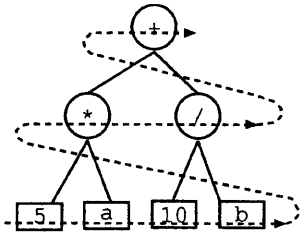
本報告では手続き型の高級言語を対象として、上記方法とは逆に、キューマシン計算モデルを式の評価に適用し、手続きおよび関数呼び出しによって作られる木を従来方式のソフトウエア・スタックで管理するスーパスカラ・プロセッサの設計について述べる。

† 電気通信大学 大学院 情報システム学研究科
The Graduate School of Information Systems,
The University of Electro-Communications

2. キューマシン計算モデル[5]

はじめに提案されているキューマシン計算モデルの概要と重要な特徴についてまとめる。

キューマシン計算モデルにおいて用いられる命令の列は、式の木を根から幅優先に探索して、その探索順序とは逆順に葉と節をならべて作る。例えば、図1にあるような式の木に対しては、矢印にそって命令を並べることで同図の下にある命令列となる。この命令列において、即値（または変数）はその値をキューの末尾に挿入する命令、演算子は必要なオペランドをキューの先頭から順にとりだして結果をキューの末尾に挿入する命令とすると、各命令ごとに図2のようにキューの中身が変遷して計算が行われる。例えばこの図の5番目の命令'*'は、前の状態のキューの先頭から'5'と'a'を取り出し、計算結果として'5*a'をキューの末尾に挿入している。この例の通り、スタック・マシ



5 a 10 b * / +

図1 キューマシンの命令列

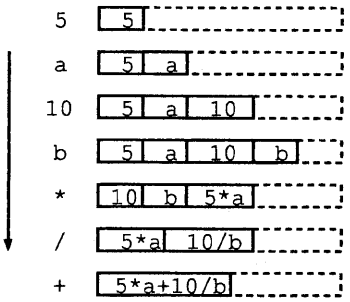


図2 実行中のキューの変化

ンと類似して演算命令は0オペランドで構成することができる。

キューマシン計算モデルのひとつの特徴は、評価すべき式の木を幅優先で探索して命令列を作成しているために、隣接する命令間でそれぞれの処理内容の独立性が高くなることである。したがって、キューの定義にそってデータの出し入れを逐次的に行うのではなく、先入れ先出し方式で処理したときと同じデータ配置にすることを条件に、ランダムにアクセスすることで、独立した命令の処理順序を任意にすることができる。例えば、先の図2では5番目と6番目の演算命令('*'と'/')は同時に行うことができる。したがって、このモデルのプロセッサを設計したときには、命令のスーパスカラ実行やアウトオブオーダ実行を利用することができる。

もうひとつの特徴は、これも木の幅優先探索に関係するが、複数の木を一括して処理する命令列を扱うことが可能な点である。独立した複数の木を横に並べ、それが根の方でつながっているがごとく命令列を生成し処理することで、木の見かけの幅を広げることができる。これによって命令列中に連続して同時処理できる命令の数を増やすことが可能となる。また、高さ方向の相対的な関係をずらして木を並べることにより、同時処理できる命令の数を単に増加させるだけでなく調節することもできる。

3. 命令セット・アーキテクチャの設計

ここではキューマシン計算モデルを用いたプロセッサの命令セット・アーキテクチャに対する設計思想について述べ、実行されるプログラムのサイズが小さくなる可能性を示す。アーキテクチャは、後の章で述べる幾つかのアイデアを検証するためのもので、現在のところFPGAによる実装を計画している。

3.1 キューとスタックの役割

手続き型の高級言語プログラムをコンパイルして実行することを考慮すると、プロセッサにはスタックのデータ構造が必要となってくる。それは手続きの呼び出し順序が深さ優先で木をたどる形をとっているためである。また、再帰的な手続き呼び出しに関する木は葉の位置が動的に決まるために、手続き呼出しを行なう高級言語の実行では、キューマシン計算モデルのように木を葉の方から処理していくことが難しい。

式処理用のデータ構造と手続き呼び出し処理用のデータ構造という観点で既存のマシンを眺めてみると次の様になっている。Pコードを機械語とする仮想計算機のスタック・マシンでは、式の処理と手続き呼び出しの処理に対して一つのスタックを用いて処理を行う。駆動レコード (activation record) を手続き処理の単位としてスタックに積み、スタックの先頭で式の処理を行う。プロセッサとして実現される一般的なスタック・マシンでは、データ・スタックとスタックの先頭を保持するレジスタを用いて式の処理を行い、リターン・スタックに手続きの戻り番地を保持することで手続きの処理を行う[1]。また、現代のレジスタ・マシンでは式の処理をレジスタ上で行い、手続きをメモリ中に作るソフトウエア・スタックで処理することが多い。ただしこれら全てにおいて、CやPASCALにおける駆動レコードの扱いは、レコード内のランダムなアクセスを含むことから、純粋なスタックだけではうまくいかない。

キューマシン計算モデルを実現するプロセッサの構成においても、式の処理と手続き処理に関して幾つかの選択肢が考えられるが、ここでは、手続き型の高級言語プログラムの実行を考えて、命令セット・アーキテクチャとしては、式処理用のキューとメモリ中につくるソフトウエア・スタックの構成を考える。プログラムから操作できるのは、基本としてキュー、ランダムアクセス可能なメモリ、プログラム・カウンタ (PC)、スタック・ポインタ (SP)、フレーム・ポインタ (FP) とする。また、1ワードが32ビット。論理アドレス空間は32ビットを仮定する。

3.2 命令フォーマットの構成

キューマシン計算モデルは0オペランドで演算命令を構成することが可能であるために、1つの命令長を短くすることができる。これによって、プログラムサ

5 1	7	0	
1x	dsp:14		Type A
01	op:6	imm:8	Type B
00	op1:6	00 op2:6	Type C

図3　命令フォーマット

イズが小さくなり、命令メモリ・コストを下げ、命令キャッシュのヒット率を向上させるという、スタック・マシンと同様の利点を享受できるはずである。

しかし、プログラム中では即値を指定する必要があるために、単に命令の種類の数に必要なビット数で命令長を決めることはできない。これに加えて即値と命令を別にすると、相対アドレス指定などの加算命令が増えることになる。逆に命令中に即値を埋め込むと命令長そのものが大きくなる。したがって、これら双方をうまく利用する必要がある。また、命令長はメモリ・アクセスの単位である8, 16, 32ビットから選択しないと、命令がページ境界をまたいで不都合も生じ得る[4]。さらに、可変長命令でフェッチの単位と命令長とが対応せずに、1回の命令フェッチでデコードが完了しない場合ができると、デコードが複雑になる上に、逐次的にならざるを得なくなる。結果としてスーパスカラ実行が難しくなる。

これらの点を考慮して以下の議論では図3の3タイプのフォーマットで構成される命令セットを前提として考える。ここでは命令長は16ビットとして、上位2ビットで各タイプを分ける。

Type Aの命令は無条件分岐と手続き呼出し用の命令である。無条件分岐命令は指定した14ビットのディスプレイスメントでPC相対アドレスを計算して分岐を行なう。手続き呼出し命令はこれに加えて分岐前のPCの値をキューに挿入する。これらの分岐命令で扱えない分岐は、即値ロード命令およびキューの先頭の値を分岐先アドレスとする分岐命令を用いる。

Type Bの命令は8ビットの即値を用いる命令群で、即値を用いた演算命令、ロード／ストア命令、条件分岐命令、16／32ビットの即値ロード命令等である。即値ロード命令はPC相対アドレスで指定される場所に格納された即値をキューに挿入する。これはスタックマシンやSuperH RISC Engine[4]を参考にした。

Type Cは即値を必要としない命令群であり、単項／2項演算命令、キューの先頭の値を分岐先アドレスとする分岐命令、そしてNOP(No OPeration)命令である。命令長を他と揃えるために必ず2命令を指定するが、2命令の実行は順序づけられている。

上記の構成の命令セットでは、一般のRISCプロセッサの命令とは異なり、レジスタ指定がなく、即値以外のオペランドは、キューの先頭および末尾に固定されている。したがって、レジスタをオペランドとして指定するプロセッサのプログラムに比べて、プログラムサイズを大幅に削減できることが期待される。

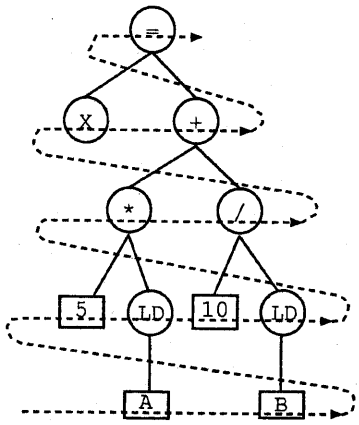


図4　ロード・ストアを考慮した木

4. スーパスカラ・プロセッサの構成

キューへのアクセスを先頭と末尾への逐次的な操作ではなく、データ配置の前後関係を保ちながら、先頭と末尾以外のデータへも同時にアクセスできるようにすることで、プログラムのスーパ・スカラ実行が可能となる。ここではキューの実現および命令発行の原理を中心に、アウト・オブ・オーダ実行を行なう、スーパスカラ・プロセッサの構成を述べる。

4.1 動的レジスタ割り付け方式

データ構造としてキューを実現するには、一般に、リンクト・リストまたは循環配列が使われる。提案方式ではレジスタを循環配列として扱うことで、キューを実現するとともに、ランダムアクセスを可能とする。循環配列ではデータが挿入されると配列上の一つの位置が割り当てられ、読み出されるとその位置が解放される。この特徴を利用して、提案するスーパスカラ・プロセッサでは、0オペランド命令をデコードした際に、循環配列を実現するレジスタの番号を割り当て、内部では通常のレジスタ・オペランドを持った命令として実行を進める。

図4はロード／ストアを考慮して先の図1を再構成したものである。図5の最左列が対応する命令列を示す。同図では、レジスタを10個使った場合と4個使った場合のレジスタ割り当てを、内部命令とともに示す。Adr命令はアドレスをロードしてキューに挿入する命令、Lit命令は8ビットの即値をキューに挿入する命令、Mult命令, Div命令, Add命令はそれぞれ2項演算命令、Load命令,Store命令はワード単位のロードとストアをそれぞれ示す。ロードとストアの内部命令における“(R0)”という表記は、レジスタの値をアドレスとしたメモリへのアクセスを示す。

循環配列に沿ったレジスタ割り当てでは、デスティ

命令列	10 レジスタ	6 レジスタ
Adr A	R0=A	R0=A
Adr B	R1=B	R1=B
Lit 5	R2=5	R2=5
Load	R3=(R0)	R3=(R0)
Lit 10	R4=10	R4=10
Load	R5=(R1)	R5=(R1)
Mult	R6=R2*R3	R0=R2*R3
Div	R7=R4/R5	R1=R4/R5
Adr X	R8=X	R2=X
Add	R9=R6+R7	R3=R0+R1
Store	(R8)=R9	(R2)=R3

図5 レジスタ割り付けの例

ネーション・レジスタが0,1,2,... と順に割り当てられ、同様にソース・レジスタが割り当てられる。最後のStore命令は2つのレジスタをソースとしていて、書き込み先はメモリである。したがって、10レジスタの場合は前のAdd命令に続いて、R8,R9と割り当てられ、6レジスタの場合はR2,R3と割り当てられている。

複数命令のレジスタ割り当てを1サイクルで行なうに当たって、例えば4命令デコーダの各々をD0,D1,D2,D3とすると、レジスタ割り当てのための情報は循環配列の先頭と末尾の情報に加えて次のようになる。

- D0はD0への入力
- D1はD0とD1への入力
- D2はD0,D1とD2への入力
- D3はD0,D1,D2とD3への入力

これを実現するためには命令のコード化において、ソース・オペランドとデスティネーション・オペランドの数ごとに命令をまとめるような考慮をする必要がある。

4.2 命令発行機構

レジスタ割り当ての済んだ内部命令は発行待ちのバッファに送られる。命令発行ではすべてのソース・オペランドが揃い、デスティネーション・オペランドが書き込み可能になっていることを保証する必要がある。提案方式では、循環配列に沿ったレジスタ割り当ての特徴を用いる。

循環配列を用いてキューを実現した場合に、配列の各要素は書き込みと読み出しとが必ず交互に行なわれる。したがって、循環配列を構成するレジスタは各々のアクセスごとに、読み出し可能状態と書き込み可能状態の2状態を遷移する。この特徴を利用すると、命令が発行可能であることを保証するには次の条件を満たせば良い。

- 読み出し可能条件
  - ソース・レジスタの全てが読み出し可能
  - 先行する完了前のどの命令もソース・レジスタに書き込まない
- 書き込み可能条件
  - デスティネーション・レジスタが書き込み可能
  - 先行する完了前のどの命令もデスティネーション・レジスタに書き込まない

特筆すべきは、読み込みと書き込みの双方の条件で、先行する完了前の命令における書き込み対象を調べれば良い点である。書き込み可能条件において、デスティネーション・レジスタが書き込み可能状態で、かつ完了前の先行命令がこれに書き込まないということは、当該レジスタに対する読み出しを行う先行命令は、その読み出しを終了していることを意味する。よって、対象としている命令がレジスタに書き込む順番になったということになる。

8レジスタで割り付けを行なった内部命令の発行の例を図6により示す。ここでは1サイクルに4命令ずつデコードされて発行バッファに挿入される。発行バッファは5命令以上保持できるとする。即値命令は発行の次のサイクルには完了し、その他の命令は1サイクルの後に完了するものと仮定する。発行条件を保証するために次の情報を保持する3種のベクタを使用する。

- 発行バッファ内にある先行命令の書き込みレジスタの累積情報(IW)
- 実行中の命令の書き込みレジスタの情報(EW)
- レジスタの状態(RS)

図6ではクロック・サイクルごとの命令発行の可否と、それを決定するためにベクタの状態が示されている。IWは発行バッファのエントリごとにあり、あるビットが1の場合に対応するレジスタに先行命令が書き込むことを表す。同様にEWにおいては、実行中の命令が書き込もうとするレジスタに対応するビットが1となる。RSは0が書き込み可能状態、1が読み込み可能状態を示す。

1クロック・サイクル目における先頭の3命令はいずれも書き込むのみである。これらはどれもデスティネーション・レジスタに対応するRSが書き込み可能(0)で、かつ各IWとEWで対応するビットが0であるために発行可能と判断される。4番目の命令はR0をソース・レジスタとするが、対応するRSが書き込み可能状態なので発行できず、次のクロック・サイクルに持ち越される。

2クロック・サイクル目では3つの書き込みが完了し、RSのR0～R2が読み込み可能状態となる。バッファ内の先頭の1と3番目の命令では、ソース・レジスタが読み込み可能、かつ先行する命令がこのレジスタに書き込まない。加えて、デスティネーション・レジスタは書き込み可能で、先行する命令がこのレジスタに書き込まないので発行可能となる。2番目の命令は書き込む条件を満たしているので、これも発行可能となる。残りの2命令は、ソース・レジスタが書き込み可能状態なので発行できない。以降、同様の方法で

CL	可否	内部命令	IW 543210	EW と RS	543210
1	○	R0=A		EW	000000
	○	R1=B	000001	RS	000000
	○	R2=5	000011		
	×	R3=(R0)	000111		
2	○	R3=(R0)		EW	000000
	○	R4=10	001000	RS	000111
	○	R5=(R1)	011000		
	×	R0=R2*R3	111000		
	×	R1=R4/R5	111001		
3	×	R0=R2*R3		EW	101000
	×	R1=R4/R5	000001	RS	010100
	×	R2=X	000011		
	×	R3=R0+R1	000111		
	×	(R2)=R3	001111		
4	○	R0=R2*R3		EW	000000
	○	R1=R4/R5	000001	RS	111100
	×	R2=X	000011		
	×	R3=R0+R1	000111		
	×	(R2)=R3	001111		
5	○	R2=X		EW	000011
	×	R3=R0+R1	000100	RS	000000
	×	(R2)=R3	001100		
6	○	R3=R0+R1		EW	000000
	×	(R2)=R3	001000	RS	000111
7	×	(R2)=R3		EW	001000
			000000	RS	000100
8	○	(R2)=R3		EW	000000
			000000	RS	001100

図6　命令発行の例

正しいタイミングでの命令発行がなされていく。

この方法の問題点としては、レジスタの依存関係の種類によっては、1クロック・サイクルの無駄が生じ得る点である。例えば、3クロック・サイクル目の1と3番目の命令である。1番目の命令がR2を読みだし、これに続いて3番目の命令がR2に書き込む。パイプライン実行などを考慮に入れれば、この2命令を同時に発行しても正しい実行を行なうことができる。

5. マルチスレッド実行支援機構

キューマシン計算モデルには、前述したように、複数の木を並べてそれらを一括して処理する命令列を扱うことができるという特徴を有している。これは依存関係のない任意の木を、同時進行的に処理していくことが可能であることを意味する。この特徴を利用すると、スレッドを動的に生成して、同時進行的に処理することが可能となる。さらに、同時進行的に処理される命令は内部命令に変換されて、スーパスカラ実行が行なわれるので、単一のプロセッサにおいてマルチスレッドが命令レベルで並列実行されることとなる。

図7は2つの木を同時進行で処理する命令列に加えて、新たなスレッドを開始できる点を示している。一

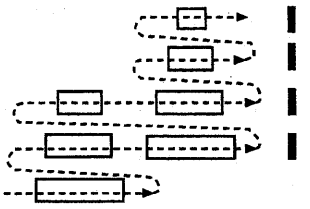


図7　2つの木の処理とスレッド付加可能点

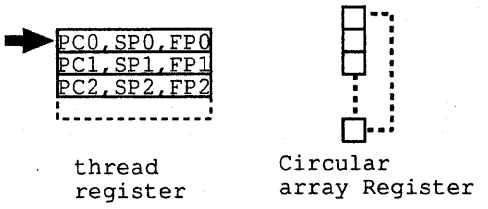


図8　レジスタ構成のイメージ

つのスレッドとして複数の木をキューマシンで処理するには、図の矢印のように並べられた木の同じレベルを全てたどって命令列を生成する。この木のレベルの変わる図中の黒い縦棒のある点でスレッドを切替えることにすると、処理される命令の列は、もともと3つ以上の木を並べたものと同一にすることができる。この切替えが可能になるならば、以降はデコードされてスーパスカラ実行されるので、幅の広い木を、高い並列度で実行することと同じになる。

このようなマルチスレッド実行を可能とするには次の2つが必要となる。

- 木のレベルの変わる点を命令列中に明示する
- 循環配列を構成するレジスタ以外のレジスタ(PC, SP, FP)のセットを複数用意して切替える

図8にレジスタ構成のイメージを示す。図のようにPCなどから構成されるレジスタのセットでリストを構成する。新たなスレッドを生成する場合にはリストの末尾に加える。そして、命令の区切りごとに矢印に当たるリストのポインタを切替える。

このマルチスレッド支援機構の特徴をまとめると、スレッド切替えのオーバヘッドが小さい点と高い並列度での実行が期待できる点である。ただし、リスト状になるレジスタをハードウエアでどのように構成するかという点と、循環配列を構成するレジスタが足りなくなる場合にはどうするかという点を解決する必要がある。

6. 割り込み処理

プロセッサの割り込み処理では、割り込まれたプログラムの状態の待避と復帰が必要である。レジスタ・マシンではプログラム・カウンタの他に、割り込み処理の演算で使用するレジスタがその対象となる。これ

までに述べてきたプログラムおよびプロセッサ構成では、演算目的のレジスタの待避と復帰に関して、いくつかの手法が考えられる。

現在、案として挙げられている方法は以下の3通りである。

- 循環配列の未使用部分で演算を行う
- 割り込み処理スレッドとして処理する
- レジスタをメモリに待避して処理をする

1と2番目の方法について説明する。循環配列の特徴の一つは、配列の使用部分と未使用部分がそれぞれ連続領域となる点である。また、プログラム中には陽にレジスタ指定がなされていないので、実行には循環配列を構成するどのレジスタを割り当てて実行しても良い。したがって、未使用部分に割り込み処理用の空き部分が十分にあれば、割り込み処理プログラムは連続した未使用部分を利用して実行が可能となる。さらに、割り込み処理プログラムが使用するレジスタ数を限定して、その数のレジスタだけで循環配列を構成するようにすれば、未使用部分が存在する限りにおいて、割り込みへの更なる割り込みが可能である。

2番目の方法は、走行中のプログラムと同時進行で処理しても良い種類の割り込みに適用できる。割り込み要求とともに新たな割り込み処理スレッド用のプログラム・カウンタ等を割り当て、前述のスレッド切り替えの方式でプログラムを並列実行させる。

7. おわりに

本報告ではキューマシン計算モデルに基づいたスーパスカラ・プロセッサの設計について述べた。

プログラムの特徴としては、レジスタ・オペランドを必要としないために、スタックマシンと同様に命令長が短くなり、プログラムサイズが小さくなる可能性がある。これとは逆にレジスタを指定しないために、ロードとストアの命令が増加して、プログラムサイズが増加する要因もある。この付加的なロードとストアはメモリ・アクセスが増えることも意味する。

レジスタ・マシンではレジスタ上に再利用する値を置いて演算を進めることで、メモリ・アクセスを削減する。提案方式では一度読み出されたレジスタは割り当てが解かれ、その後に別の値が書き込まれる。したがって、値の再利用が難しい。特に関数の引数渡しなどで影響が出る可能性がある。

これに対する方法としては、キューの先頭データを複製して末尾に挿入する命令や、ストアとともに保存される値を再度キューの末尾に挿入する命令など、キューマシン特有の命令を用意することで対処する方法が検討されている。

プロセッサとしての特徴は次の通りである。まず、プロセッサ内部ではレジスタを使った実行を行うにも関わらず、プログラム中にはレジスタの指定をしないことから、プロセッサが持つレジスタ数に自由度が生まれる。プログラムによってレジスタの必要最低数があるために、まったく依存しないわけではないが、この特徴によってレジスタ数の異なるプロセッサ・ファミリが構成できる。

循環配列の特徴を利用したレジスタ割り付けは、従来のスーパスカラ・プロセッサにおけるレジスタ・リネーミングに類似する。一般にレジスタ・リネーミングは、ランダムに生じる空きレジスタを何らかの形で保持しなければならず、ハードウエアが複雑になる。一方、提案方式ではレジスタの割り当てと解放は規則的に行われるので、単純な機構で実現することが可能であると考えられる。

また、命令発行の原理もレジスタへの規則的なアクセスパターンを利用しているので、ハードウエアは従来方式に比べて単純になると考えられる。

マルチスレッド実行の支援機構は、スレッド数に制限ができるものの、1～3クロック・サイクル程度の高速なスレッド切り替えが可能になり得る。この高速なスレッド切り替え機構とスーパスカラ実行とで、マルチスレッドの処理は、単一のプロセッサにおける命令レベルの並列実行に置き換えられる。この特徴はハードウエアによる中粒度から細粒度への並列度変換とも言える。

参考文献

1) P. J. KoopMan, Jr 著, 田中清巨 監訳, 藤井敬雄訳: スタックコンピュータ CISC/RISC とスタックアーキテクチャ, 共立出版 (1994).
2) M. Johnson: 'Superscalar Microprocessor Design' , Prentice Hall(1991).
3) S. Weiss and J. E. Smith 著, 日本アイ・ビー・エム監訳 : PowerPC 解説 POWER から PowerPC へ, International THOMSON Publishing(1995).
4) 荒川文男, 西井修, 中川典夫: 組み込み用途プロセッサー SH, 情報処理 ,Vol.39, No.3, pp.195 – 199(1998).
5) 前田敦司, 中西正和: 新しい計算モデル キューマシンとその並列関数型言語への応用, 情報処理学会論文誌, Vol. 38 No. 3 pp.574 – 583 (1997).
6) 前田敦司, 中西正和: キューマシン方式による並列 Lisp 処理系のスケジューリング手法, 電子情報通信学会論文誌 D-I Vol.J80-D-I No.7, pp.624 – 634(1997).