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

2003－HPC－96　(8)
2003／10／16

キュー計算原理による SMT 型マルチスレッド並列キュープロセッサの提案と設計

奥村義智[†☆]　佐々木博敏[†]　BEN A. ABDERAZEK[†]
繁田聡一[†]　吉永　努[†]　曽和将容[†]

本論文では，キュー計算モデルによる SMT の概念を取り入れたマルチスレッド並列キュープロセッサを提案する．マルチスレッド並列キュープロセッサは，並列キュープロセッサの持つ命令レベルの並列性（ILP）だけでなく，マルチスレッドによるスレッドレベルの並列性（TLP）を利用できる．ILP と TLP を併用することで，実行するアプリケーションに適した粒度の並列性を提供することができるため，計算資源の稼働率を上げることができる．

また，提案するマルチスレッド並列キュープロセッサをハードウエア記述言語である Verilog-HDL により記述し，シミュレーションで動作確認を行った．

Proposal and design of an SMT type Multithread Parallel Queue Processor for the queue calculation principle

YOSHITOMO OKUMURA[†☆], HIROTOSHI SASAKI[†], BEN A. ABDERAZEK[†],
SOICHI SHIGETA[†], TSUTOMU YOSHINAGA[†] and MASAHIRO SOWA[†]

In this paper, we propose Multithread Parallel Queue Processor (MPQP) by integrating SMT abstraction into the queue calculation model. MPQP can provide both Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP). Namely, MPQP has ability to offer an appreciate granularity of parallelism for wide range of applications. So, MPQP can improve usability of calculation resources.

We described proposed MPQP by using Verilog-HDL which is a hardware description language, then, we made simulations to validate our design.

1. はじめに

情報技術の驚異的な進歩は，コンピュータシステムの超高速技術に支えられている．その核たる技術がプロセッサ技術であり，さらなる技術向上が求められている．従来の提案されてきた手法では非常に細分化された技術が要求され，改善に改善を重ねるといったような設計手法に陥ってしまい，今日まで抜本的な解決には至っていない．

本論文では，並列キュープロセッサによる命令レベルの並列性（ILP : Instruction Level Parallelism）とマルチスレッド処理によるスレッドレベルの並列性（TLP : Thread Level Parallelism）という２つの並列性を引き出すためのマルチスレッド並列キュープロセッサ（MPQP : Multithreaded Parallel Queue Processor）を提案する．

MPQP は並列キュープロセッサ（PQP : Parallel Queue Processor） をベースにしたプロセッサであり，キューと呼ばれる FIFO（First In First Out）の記憶領域に計算の中間結果を格納するキュー計算モデルに基づいている．

MPQP は，キューマシンアーキテクチャの小さなプログラムサイズ，簡単なハードウエア構成，命令のアウトオブオーダー並列実行という利点を

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, University of Electro-Communications
☆ 現在，ソニー株式会社
Presently with Sony Corporation

保持しながら，マルチスレッド処理を併用することにより，高速な処理を可能にするプロセッサである．同時マルチスレッディング（SMT：Simultaneous Multi Threading）を利用したプロセッサでは，スレッド間で資源の共有を行うことが可能であることなどから，次世代のプロセッサ[1]として有望視されている．本論文で提案する MPQP は従来の SMT プロセッサに比べ，より多くの資源を共有できるという特徴を持っている．

3. マルチスレッド並列キュープロセッサの提案

2.1 キュー計算モデル

プロセッサの計算モデルには，ランダムアクセス計算モデル，スタック計算モデル，キュー計算モデルの 3 つがあり，各計算モデルに基づいて設計されたマシンをそれぞれランダムアクセスマシン, スタックマシン, キューマシン[2), 3), 4)] と呼ぶ．

ランダムアクセス計算モデルは，レジスタにランダムにアクセスするため，演算に使用するレジスタを明示的に指定することが必要である．スタック計算モデルはスタックと呼ばれる LIFO（Last In First Out）の記憶領域に計算の中間結果を格納する計算モデルである．データの挿入と取り出しはスタックトップに対してだけ行われるので，レジスタを明示的に指定する必要がない．キュー計算モデルはキューと呼ばれる FIFO の記憶領域に計算の中間結果を格納する計算モデルである．全ての命令がキューの先頭（QH : Queue Head）からデータを取り出し，キューの末尾（QT : Queue Tail）に計算結果を格納するモデルである．このモデルもレジスタを明示的に指定する必要がない．

例として，キュー計算モデルで *a*+*b* という計算式の計算する様子を図 1 に示す．一番上のキューは初期状態の空のキューである．最初の命令（*load a*）でデータ *a* をキューに入れ，次の命令（*load b*）でデータ *b* をキューに入れる．加算命令（*add*）では 2 つのデータを必要とするのでキューの先頭から *a* と *b* が取り出され，加算が行われ，*a*+*b* という結果がキューの末尾に挿入される．

キュー計算モデルでは，レジスタを明示的に指定する必要がないため，個々の命令長が短くなり，全体としてプログラムサイズも小さくなる．また，前後の命令間にデータ依存関係があることが少ないため，並列実行とアウトオブオーダー実行を実現できる．

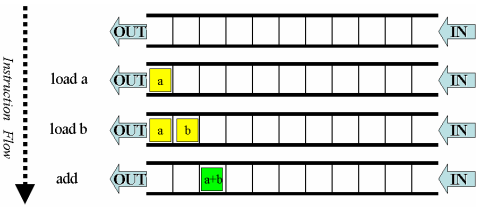

図 1　キュー計算モデル

Fig.1　Queue calculation model

2.2 並列キュープロセッサに適応する SMT とその利点

マルチスレッドとは 1 つのプロセス内に複数のスレッド（最小の実行単位）を同時に独立動作させる処理のことをいう．マルチスレッドにおける代表的な技術に，チップ・マルチプロセシング方式[5)]，タイムスライスマルチスレッディング方式，SMT 方式[6), 7)] がある．

提案する MPQP がこれらのマルチスレッドの中で SMT 方式を取り入れたのは, SMT が計算資源を共有するという特徴を持っているからである．内部資源を物理的に増設することなく, ILP を有効に使えるアーキテクチャと TLP を有効に使えるアーキテクチャを組み合わせること可能にする．例えば，TLP が小さく，ILP が大きいプログラムには多くの資源を ILP 用に用い，TLP が大きく ILP が小さいプログラムには，TLP 用に多くの資源を用いることが可能である．

さらに，PQP に SMT 方式を取り入れることで，SMT の持つ計算資源の共有に加えて，中間結果格納用のキューまで共有することが可能になる．それは，キュープロセッサはレジスタを動的に割り当てるからである．レジスタにスレッド名を付けなくてもプロセッサユニークになりレジスタ名がぶつかることはない．

レジスタ資源の共有は，アプリケーションプログラムの ILP，TLP 含有特性に，レジスタレベルま

で適用可能なことを示している．並列キュープロセッサではレジスタリネーミングが不必要であるなどの特徴を考慮すると，ハードウエアを複雑にすることなく，ILP，TLP により適応するプロセッサが得られる．

3. マルチスレッド並列キュープロセッサの設計

3.1 プロセッサアーキテクチャ

MPQP は，複数のプロセッサ要素（PE : Processor Element）で構成されている．PE は並列に動作し，スレッドを並列に実行する．今回の設計では 4 つの PE により４スレッドの並列実行としている．各 PE にはパイプラインを構成する各ステージが割り当てられ，それぞれの PE で３命令並列に処理する．

物理キューは一本だけ用意する．マルチスレッド実行時には論理的に 4 つに区切られ，各 PE に対して独立したキューを与える．シングルスレッド実行時には論理的に区切らず，長い 1 本のキューとしてそのまま用いる．

MPQP のプロセッサアーキテクチャを図 2 に示す．MPQP は各 PE に対し，物理的にフェッチ，命令分割，キュー割付，命令分配のステージをそれぞれ設ける．演算を行う実行ステージと実行の発行を行うイシューステージ，メモリ空間は共有である．

ここで問題となるのが，実行ステージに対して十分な命令を供給するかという点であるが，これを解決するためにアウトオブオーダー実行を用いる．アウトオブオーダー実行とは命令を出現順序に関係なく実行するものである．命令は消費するデータの生産が完了されていなければ実行できない．前後の命令の依存関係がない時は，ある命令の消費するデータの生産が未完了で，後続の命令のデータの生産が完了している場合には，後続の実行を先に行う． その後，命令のデータの生産が完了し，実行を行ったとしても支障はない．このとき，キュー内部の位置は互いに独立で，命令間の干渉はない．

MPQ の命令には 1 Byte と 3 Byte の 2 種類の命令長を持った命令がある．1 Byte の命令はオペコードのみを有する命令である． キュー計算モデルによりオペランドなしでオペコードのみの命令が可能となる．3 Byte の命令は 1 Byte のオペコードと 2 Byte のオペランドを有する命令である．命令で直接ある程度の大きさの数値を指定しなくてはならないような命令に対応するために用意する．

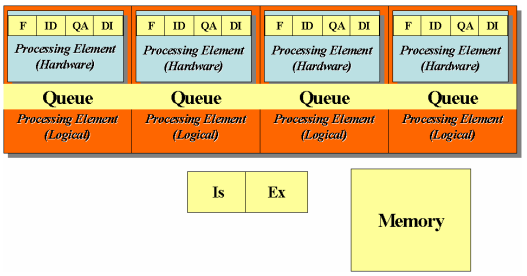

図 2　プロセッサアーキテクチャ

Fig. 2　Processor architecture

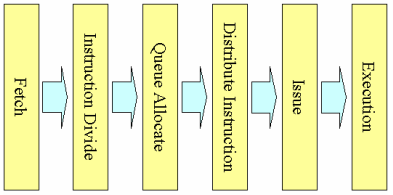

図 3　パイプライン

Fig. 3　Pipeline

3.2 パイプライン

MPQP のステージ構成は,図 3 に示すように,「フェッチ (F: Fetch)」,「命令分割 (ID: Instruction Divide)」,「キュー割付 (QA: Queue Allocate)」,「命令分配 (DI: Distribute Instruction)」,「イシュー (Is: Issue)」,「実行 (Ex: Execution)」の 6 段のパイプラインからなっている．各 PE は 3 命令分フェッチすることで，各 PE はそれぞれ実行するスレッドで 3 命令並列に処理する．

各ステージの説明を以下に示す．

3.2.1 フェッチ

フェッチはメモリから命令を読み出し，プロセッサに取り込む動作を行う．ただし，今回は命令キャッシュから読み出しているものとするため，命令の読み出しは 1 クロック内で可能であるものとする．

最も長い命令長を 3Byte としたことから，この命

令を 1 回で取り込むことができるように，フェッチ幅を 3Byte とする．

3.2.2 命令分割

フェッチされた段階では 1Byte と 3Byte の 2 種類の命令長を持った命令が混在しているため，一つ一つの命令に切り分ける．命令分割にはフェッチされた命令を一時，FIFO バッファに入れる．

3.2.3 キュー割付

キュー割付は命令分割によって切り分けられた命令に対して，キュー内部のどの位置にアクセスするかの決定を行う．且つ，命令を使用される演算器の種類ごとに振り分ける．命令分割から送られてくる命令を一時，FIFO バッファに入れる．

3.2.4 命令分配

命令分配は各 PE によって割り付けられた命令を演算器ごとに分配し，演算器の持つリザベーションステーション(RS, Reservation Station)と呼ばれる FIFO バッファに命令を入れる．命令の分配はラウンドロビンアルゴリズムで行う．

3.2.5 イシュー

イシューは演算器ごとに設けられ，RS の先頭にある命令に必要なデータがそろっているかフラグを参照し，実行可能か判断する．このとき同時に演算器の実行状態も見て，命令の実行が可能であれば演算器に命令を発行する．

3.2.6 実行

実行は命令の処理を行うステージである．イシューステージで発行された命令を受け取り，その命令の処理に必要なデータをキューに要求する．キューからデータを受け取ると，命令の実行に取り掛かりつつ，受け取ったデータのあった物理キューアドレスに該当するフラグを倒し，データが消費されたことを表す．命令の実行が完了し，結果をキューに書き込む場合，キューに書き込み要求と共にデータを送り，書き込み先のアドレスのフラグをたててデータが生産されたことを表し，命令の実行を完了する．

3.3 マルチスレッド実行

MPQP でキューを用いたマルチスレッド実行の様子を図 4 に示す．まず，fission 命令を実行し，キューを分割する．キューを PE 毎に区切り，プロセッサをマルチスレッド実行できる状態にする．スレッドは PE 毎に割り当てられたキューを使って並列に実行する．実行後は，シングルスレッド実行ができる状態に戻すために，fusion 命令を行う．fusion 命令により，fission 命令で分割されたキューを再び 1 本のキューにする．このような実行方法であるために，マルチスレッド実行だけでなく，シングルスレッド実行にも対応できる．

また，シングルスレッド実行を行うとき，キューレジスタをスレッド数の乗数分扱うことが可能となる．分割していない状態の１本のキューを使用することで，従来よりもレジスタ資源を大きく得ることができる

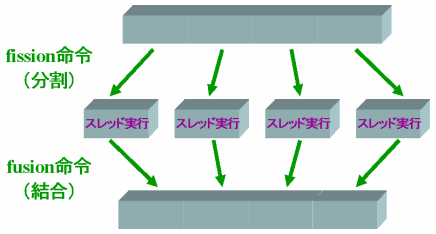

図 4 マルチスレッド実行

Fig. 4 Multithread execution

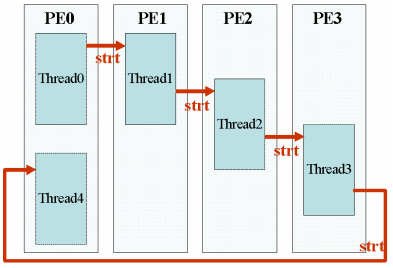

図 5 スレッド発行

Fig. 5 Thread issue

3.3.1 スレッド発行

スレッドに対して発行する側のスレッドを親スレッド，発行される側のスレッドを子スレッドとする．スレッド発行は，親スレッド内でスレッド発行命令 strt 命令が実行するときに行う．

スレッド発行の様子を図 5 に示す．親スレッド内でスレッド発行命令を行う位置は自由に決める

ことができる．発行されたスレッドは，親スレッドが実行する PE の隣の PE で実行する．なお，PE はリング状に結合する．

3.3.2 スレッド間処理の同期

スレッド間の処理の同期を取るために wake 状態と sleep 状態の状態を設ける．wake 状態はスレッドの処理が行われている状態である．sleep 状態はスレッドの処理が停止されている状態である．

スレッドが発行されたときの初期状態は wake 状態である．wake 状態にあるスレッドが sleep 状態になるためには，スレッド自身が sleep 状態への移行命令を実行し，sleep 状態になる．逆に，sleep 状態のスレッドが再び wake 状態に戻るためには，他のスレッドが当スレッドに sleep 状態解除命令を実行し，wake 状態になる．また，スレッドから他のスレッドが wake 状態であるか sleep 状態であるかを見ることができる．

3.3.3 スレッド間通信

スレッド間データ通信は，メモリもしくはフレームを介して行う．メモリ空間は，全スレッドより共有されている．フレームは，全スレッドより共有されたレジスタによって構成される記憶領域空間であり，メモリよりも高速なアクセスを提供する．これらの共有された記憶領域空間を介することでスレッド間のデータ通信は実現される．

4. シミュレーション

MPQP をハードウエア記述言語である Verilog-HDL によって記述し，シミュレータ上でプログラム実行を行い，動作検証を行った．

初期状態のキューの結合状態のままで処理する場合（シングルスレッド処理：ST）と，キューの分割状態で処理する場合（マルチスレッド処理：MT）を比較することで MPQP の性能評価を行った．

4.1 条件と環境

MPQP の性能測定では，それぞれのプログラムの終了したクロック数を測定し，クロック数の逆数をパフォーマンス値として定義する．このとき，すべての命令を逐次に実行した場合のパフォーマンス値の Serial を 100%とした場合で比較した比をパフォーマンス比とする．全ての命令を逐次実行した場合のパフォーマンス値の Serial は ST のプログラムで実行された命令数と，MPQP が 1 つの命令を処理するのにかかる平均クロック数の積で求める．なお，シミュレーション環境は表 1 に示した通りである．

表 1　シミュレーション環境

Table 1　Simulation environment

ハードウエア	CPU	Intel Pentium4 Processor 2.20GHz
	メモリ	PC2100 510MB DDR SDRAM
ソフトウエア	OS	Microsoft Windows XP Professional Edition Version 2002
	Verilog-Compiler	Icarus Verilog Compiler Version 0.6.1
	Verilog-Simulator	Icarus Verilog vvp runtime engine

4.2 シミュレーションプログラム

シミュレーションプログラムは，Simple，8FFT のプログラムを使用した．

- Simple

単項演算のみで構成される数式を 4 つだけ計算するプログラムである．極めて小さなプログラムであり，1 つの計算式を 1 つのスレッドとすることで，スレッド発行のオーバーヘッドの影響が出やすいことが予想される．MT モードに対するワーストケースとして用意した．

- 8FFT

8 点 FFT の整数部分の演算を行う．求められる 8 つの解に対する計算の一つ一つをスレッドとする．8 点のサンプリングデータは即値入力によってあらかじめ定められた定数として入力する．データ処理に関してスレッド化することで得られる利点を見ることを目的とした．

4.3 結果と考察

MPQP をシミュレータ上で動作させ，MPQP が動作することを確認し，前節で説明した各プログラムについてのシミュレーション結果より算出したパフォーマンス比を図 6 に示す．

Simple と 8FFT の両方のプログラムにおいて，ST と MT は共に逐次実行時の値 Serial を上回った．これは MPQP がシングルスレッド実行時及びマルチスレッド実行時の双方で，命令を並列実行することで高速性を発揮していることを表している．

ST と MT を比較すると Simple では ST が良い性能を得て，8FFT では MT が良い性能を得たことがわかる．

Simple では MT のパフォーマンス比が ST の値を下回った．キューの分割，スレッド発行のためのオーバーヘッドがプログラム処理全体に対して占める割合が大きいためである．

8FFT では 1 つの解を出すために，サンプリングデータを 8 つ使用する．64 個のデータを入力しなくてはならない．この時，MT モードでは 3Byte のフェッチ幅を持つ PE が 4 基存在することで 12Bytet のフェッチ幅を得る．命令を並列実行することができる機会が増えたためパフォーマンスが上昇した．

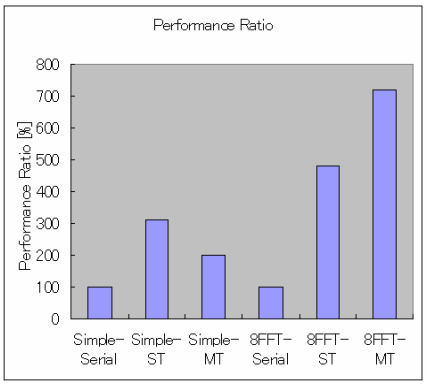

図 6　パフォーマンス比

Fig. 6　Performance Ratio

5. おわりに

本論文では，並列キュープロセッサの高速化のアプローチとして，ILP だけでなく TLP にも着目し，従来の PQP に SMT を取り入れた MPQP を提案した．

本手法では，プロセッサを増やすことで TLP を向上させる手法とは異なり，1 つのプロセッサ上に複数のスレッドを設けることで TLP を向上させる．複数のスレッドが同時に実行され，計算資源は共有できるので稼働率の向上が望める．さらに，レジスタ部にキューを用いているためにシングルスレッド実行とマルチスレッド実行の両方に対応しながら，シングルスレッド実行時では分割していない状態の 1 本のキューを使用することで，従来よりもレジスタ資源を大きく得ることができる．

今回設計した MPQP の動作確認を行うために，ハードウエア記述言語である Verilog-HDL によって記述し，シミュレーションを行った．これは MPQP が設計可能であるかを見る基本設計を行ったものであるから，今後は最適なハードウエア設計を行い，処理能力等の性能を評価する予定である．

参考文献

1) Susan J Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, Dean M. Tullsen: *Simultaneous Multithreading: A Platform for Next-Generation Processors*, IEEE Micro, (1997).

2) Bruno R. Preiss: *Data Flow on a Queue Machine*, The Institute of Electrical and Electronic Engineers, (1985).

3) 奥村義智，吉永努，曽和将容: キューマシン用 C コンパイラ QCC, 電子情報通信学会 情報・システムソサイエティ大会, p.17, (2001).

4) 奥村義智，吉永努，曽和将容: キューマシン用並列化 C コンパイラ，情報処理学会研究報告 2002-ARC-149，p.127, (2002).

5) B.J. Smith: *Architecture and Applications of the HEP Multiprocessor Computer System*, SPIE Real Time Signal Processing IV, pp.241-248, (1981).

6) D.Tullsen, S.Eggers, H.Levy: *Simultaneous Multithreading Maximizing On-chip Parallelism*, 22nd Annual International Symposium on Computer Architecture, pp.392-403, (1995).

7) D.Tullsen, S.Eggers, J.Emer, H.Levy, J.Lo, R.Stamm: *Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor*, 23nd Annual International Symposium on Computer Architecture, pp.191-202, (1996).