計算機アーキテクチャ 132−16
システムソフトウェアとオペレーティング・システム 80−16
ハイパフォーマンスコンピューティング 75−16
（1999. 3. 5）

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実装と評価

鈴木 均　前田 敦司　岡本 秀輔
電気通信大学 大学院 情報システム学研究科

キューマシン計算モデルは、式の木構造を幅優先に探索することで命令列を作り、式の中間結果をFIFOのキューに格納する計算モデルである。これまでにこの計算モデルの特徴を利用したスーパスカラプロセッサのモデル設計がなされてきた。本報告ではプログラム中の繰り返し操作・手続き呼び出しなどにおいて、データ順序の入れ替えやインデックス計算などを行う際に有用なデータ退避用のキューを導入した。まず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動作原理について説明し、続いてデータ退避用のキューについて言及し、PLD化を目指した設計・シミュレーションの結果について述べる。

Implementation and evaluation of SuperScalar Processor based on Queue Machine Computation Model

Hitoshi SUZUKI, Atusi MAEDA, Shusuke OKAMOTO
The Graduate School of Information System,
The University of Electro-Communications

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 a design of superscalar processor model. In this paper, we produce a queue for backup/restore data, which efficiently for loop or procedure call. We describe a result of design and simulation for implementation of PLD device.

1 はじめに

本研究で設計したスーパスカラプロセッサは、式の木構造を幅優先に探索してその過程で現れたノードを命令列を作り、式の中間結果をFIFOのキューに格納するキューマシン計算モデルに基づいている。

これまでにこの計算モデルの特徴を利用したスーパスカラプロセッサのモデル設計がなされてきた[1]。中間結果を格納するメモリに循環配列レジスタを用いてデータへのランダムアクセスを可能にしつつ、レジスタ割付をFIFOのキューのデータ配置に沿って行うことで、スーパスカラ実行を可能にしている。

本研究ではこの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にデータ退避/復元用のキューを新たに追加することで、手続き呼び出し、繰返し操作など操作に対する性能を

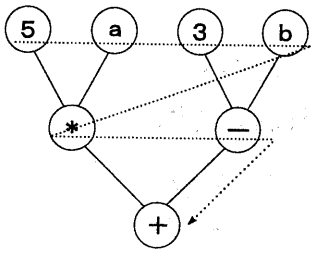

図 1: 式 (5*a)+(3-b) の構文木

向上させたプロセッサを設計した。また、PLD による実装を目指した論理設計を行い、動作の検証を行う。

2 キューマシン計算モデル

はじめに提案されているキューマシン計算モデルの原理と特徴をまとめ、スーパスカラ実行の可能性について述べる。

キューマシン計算モデルは図 1 のような逆木に対し、矢印のように木を幅優先に探索し、図 2 のような命令列を作り出す。命令が実行されていくにつれ、キューの中身は図 2 のように変遷していく。最初の 4 つの即値によって、4 つの値がキューにいれられ、次に*演算子によって '5' と 'a' が取り出され、'5*a' がキューの末尾にいれられる。このようにキューマシン計算モデルでは演算命令が 0 オペランドで構成される。

キューマシン計算モデルでは評価する式の木を幅優先に探索して命令列をつくるために、

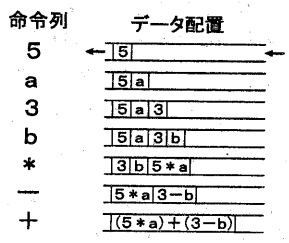

図 2: 式 (5*a)+(3-b) の命令列とキュー

キュー内にデータが多数存在する場合に隣接した命令間の独立性が高い。従って、仮にキュー内のデータに対して FIFO ではなくランダムアクセスが可能であれば、これらの独立性の高い命令のスーパスカラ実行が可能になる。

3 スーパスカラ実行の原理

キュー内で隣接したデータの独立性を利用し、かつ末尾に挿入し先頭から取り出すというキューの定義に沿わずに、キュー内のデータの並び順を保持してデータに対してランダムアクセスを行うことで、スーパスカラ実行を実現できる。

3.1 循環配列レジスタ

キューのデータ構造の実現として循環配列レジスタを用い、ランダムアクセスを可能にする。循環配列レジスタは図 3 のようにキューの先頭を表すポインタ QH と末尾を表すポインタ QT を持っており、QH–QT 間のレジスタがキュー内にあるデータを示している。図 3 の (a) の状態時にデータをキューから一つ取り出し、2 つ格納したとすると、QH,QT の示す位置は (b) のようになる。各レジスタはこのポインタの移動に合わせて、レジスタ内のデータが有効である読出し可能状態、無効である書込み可能状態に必ず交互に変化する。

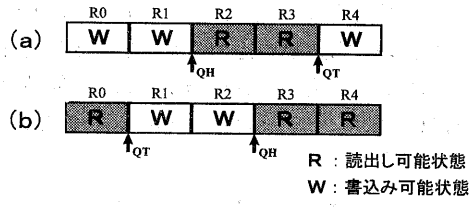

図 3: 循環配列レジスタ

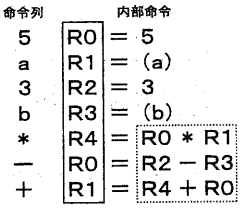

図 4: レジスタ割付された内部命令

3.2 レジスタ割付機構

0 オペランド命令はプロセッサに取りこまれた後、循環配列レジスタの番号をオペランドとして指定する内部命令へ変換される。

内部命令の読出しレジスタは QH の示す位置、書込みレジスタは QT の示す位置から順に割り付けられていく。図 4 はレジスタが 5 つの場合に図 1 に対してレジスタ割付を行った結果を内部命令と共に示したものである。循環配列に沿ったレジスタ割付では書込みレジスタは、R0 から順番に割付けられ、読出しレジスタの場合も同様である。このようにレジスタ割付は順番に割り付けるだけの単純なものになる。

3.3 命令発行機構

レジスタ割付の済んだ内部命令は発行待ちバッファに送られる。命令発行機構は発行待ちバッファ内の命令から、全ての読出しオペランド、書込みオペランドが使用可能なものを探し、発行を行う。前述の循環配列レジスタが、交互に読出し可能状態、書込み可能状態に変化することを踏まえて、オペランドが使用可能であることを保証するには、次のような条件を満たせば良い。

- 読込みレジスタの使用可能条件
  - レジスタが読出し可能状態
  - 先行命令がレジスタに書き込まない
- 書込みレジスタの使用可能条件
  - そのレジスタが書込み可能状態
  - 先行命令がレジスタに書き込まない

留意すべき点は 2 つ目の条件が読出し、書込みのどちらの場合も同じ“書込み”に関する条件となっていることである。書込みレジスタに関して、書込み可能であって先行命令による書込みが起こらないということは、即ち先行命令による読出しが終了しているということと等価である。命令発行機構はレジスタ状態と、先行命令の書込みを調査するだけでよく、比較的単純な構造になる。

キューマシン計算モデルはこのようなレジスタ割付機構と命令発行機構を備えることで、スーパスカラ実行を可能にする。

4 一時キューの導入

プログラム中の繰返し操作では、ループカウンタとなる値をループ内部で複数回にわたって使用し、更にカウンタの更新などを行うため、ループカウンタ値が任意の位置で複数回使用できることが望ましい。また手続き呼び出しにおける引数も、手続き内では複数回、任意の位置で使用されると考えられるので、同様に順序入れ換え、あるいは退避、複製などが必要となる。

このようなデータの一時退避、複製、任意位置での取り出しを高速に行うためにデータ退避用のレジスタを別途に用意することを考える。汎用レジスタを用いることは、レジスタ割付方式、命令発行機構の単純性という本プロセッサの利点に反する。そこでデータ退避/復元用のレジスタも循環配列レジスタで構成し、FIFO のキューとすることで、レジスタ割付方式、命令発行機構を統一することにした。以降はいままでの演算を行うためのキューを演算キュー、データ退避/復元用のキューを一時キューを呼ぶ。

4.1 一時キューの操作

一時キューに対する操作は、自分自身あるいは演算キューとのデータ転送および即値入力の

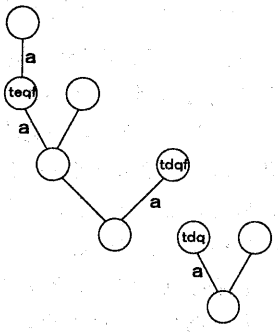

図 5: 一時キューの利用方法

みとする。一時キュー内のデータに対して演算を許すと、演算命令が使用するキューのエントリの組み合わせが増えるため、命令数を増やすかあるいはキューを選択するためのフィールドを命令に設けなければならなくなる。どちらにせよ命令長の増大と複雑化を招く原因となり、キューマシン計算モデルの特徴である0オペランド演算命令の利点を損なってしまう。

4.2 利用方法

一時キューの利用方法は主にデータの一時退避、複製、任意位置での取り出しである。図5ではまず左の'teqf'命令で、aを一時キューに退避すると共に直下の演算命令にオペランドを供給する。その後'tdqf'命令によって、一時キューに退避したデータを残しつつ取り出し演算に用いている。最後に右の独立した式でaを一時キューから取り出す。

このように一時キューを用いることで、データの退避と任意の位置への復元や、データの複製が比較的簡単に行えるようになる。

5 命令セット

キューマシン計算モデルではオペランドを指定する必要の無い0オペランド演算命令を構成することが可能なため、ひとつの命令の命令

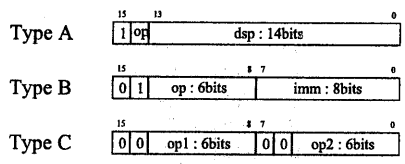

図 6: 命令フォーマット

長を短くすることができる。しかしプログラムでは即値を指定する必要が出てくるため、本研究では図6のような3種類の命令フォーマットに分け、命令長は16ビットと定めた。Type-Aは無条件分岐と手続き呼び出し用の命令フォーマットで14ビットのディスプレイスメントを備える。Type-B命令は8ビットの即値を用いる命令群で即値代入や即値演算、条件分岐命令、ロード/ストア命令などである。Type-Cは即値を必要としない命令群で単項演算、2項演算、特殊レジスタ操作、NOP、HALTなどである。Type-Cは即値指定が必要が無いために命令長は8ビットと短く、このため2命令をあわせて16ビットとし、Type-A、Type-Bと命令長を揃える。

6 ハードウェア構成

設計するプロセッのハードウェア構成を基本構成を、図7に示す。プロセッサはデコーダ(Decoder)、発行ユニット(Issue)、フェッチユニット(fetch)、レジスタファイル(RF)といくつかの機能ユニットから成り立っている。これらはおおよそパイプラインのフェイズ毎に機能するように分けられている。機能ユニットは2つの算術/論理演算ユニット(ALU)、ロード/ストアユニット(LSU)、乗算ユニット(MU)、分岐ユニット(BU)から成る。これらはそれぞれ並列に動作し、最大度数5のスーパスカラ実行を可能にする。

全体の処理の流れは次のようになる。各ユニットにおける処理がそれぞれパイプラインの

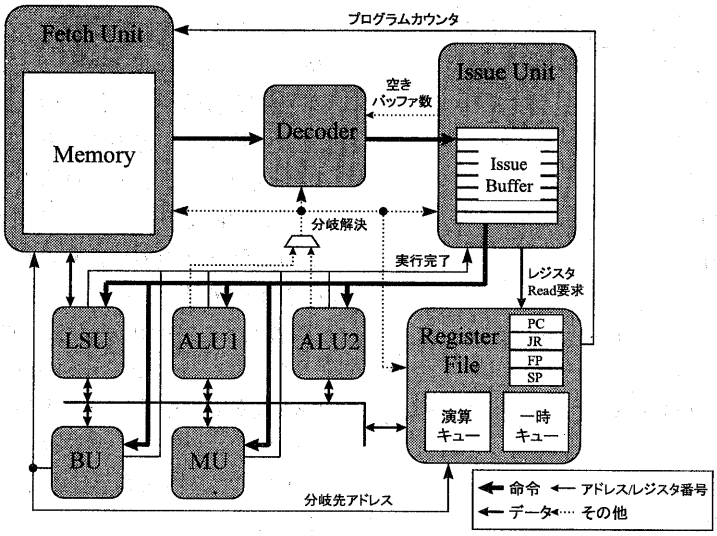

図 7: プロセッサの全体構成図

1ステージを示している。まずフェッチユニットが、メモリから32ビット長の1命令ワード(1〜4個の命令)を取り出し、デコーダへ送る(IFステージ)。デコーダはデコードを行いレジスタ割付を行った内部命令を発行ユニットへ送る(IDステージ)。そこで命令の発行可能チェックが行われ、発行可能な命令が適切な機能ユニットへ発行される(ISステージ)。また、同時にレジスタファイルへは、各機能ユニットへ供給すべきオペランドデータの要求をだし、レジスタファイルはそれを受けて、データを機能ユニットへ提供する。各機能ユニットは必要に応じて、レジスタファイルや、フェッチユニットを介してメモリにアクセスを行う。機能ユニットはそれぞれ独自のパイプラインを備えており、実行終了までの期間が違う。ALUは一般的な算術演算や論理演算を行う。LSUはメモリへのアクセスを伴う命令を処理する。MUは乗算を行うユニットである。BUは分岐命令の際のターゲットアドレス計算を行うユニットである。

7 動作検証と性能評価

7.1 プログラムサイズ

キューマシン計算モデルでは、命令長の短縮によるプログラムサイズが小さくなることが期待される。いくつかのプログラムに対して、MIPSとキューマシンによるプログラムコードの違いを考えたときのプログラムサイズは、現在のところあまり差がでない(表1)。キューマシンによるプログラムは命令長が小さくなるものの、全体の命令数がやや増加する傾向にあり、さらにフェッチを並列度を増加させるために32bit境界で行っているため、NOP命令を多く挿入せねばならない。

命令フェッチを16ビット単位に変更することで、MIPSマシンのプログラムサイズよりも

マシン	命令数	サイズ
MIPS	11	44バイト
キューマシン(32)	12	48バイト
キューマシン(16)	19	38バイト

表 1: プログラムサイズの比較例

ユニット名	使用LE数
フェッチユニット	1654
デコーダ	1373
ALU	419
LSU	204
MU	285
BU	66

表 2: 各ユニットの使用LE数

プログラムサイズは小さくなる可能性がある。

7.2 論理合成

VHDLによるプロセッサの動作記述を行い、ユニット毎に合成を行った。図2に合成した各ユニットの回路規模を示す。LE (Logic Element) はPLDの構成要素で、1LEが約12ゲートに相当する。また、波形シミュレータによるシミュレーションを行い、キューマシンの動作を確認する。

8 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実行を行うプロセッサのPLDによる実装を目標とした論理設計とシミュレーションを行い、いくつかの性能評価を行った。

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におけるプログラムはレジスタセットマシンに比べて命令長が小さくなるが、レジスタの指定がないために命令数は大幅に増加する。また、十分な並列度をプロセッサに供給するために命令フェッチの単位を大きくすることでNOP命令が増え、プログラムサイズが大きくなってしまうこともある。命令フェッチ単位の小さくし、複数命令ワードをフェッチするような構成にすることでこの問題は回避可能である。

また、一時キューの導入はプログラム実行に不可欠ともいえるループ処理、手続き呼び出し処理などのパラメータ授受、またはデータの複製、一時退避などに役立つ。且つ実装には小さなハードウェア追加で済み、演算キューとの動作原理の統一化などの利点を持ちつつ、キューマシン計算モデルによるプログラミングで陥りがちなデータ複製や移動のための性能低下を防ぐことができる。

設計にはVHDLを用い、機能別に個別にデザインすることでハードウェア構成を整理し、動作の安定化を計り、個別に評価しやすくした。実装においては、パイプライン処理、バイパス処理などを取り入れ、プロセッサの高速化を計っている。

今後の課題としては、プロセッサ性能を高めるために現段階では実装していないアルゴリズムや機構を追加していくことである。

参考文献

[1] 岡本秀輔, 前田敦司, 曽和将容: “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”, 情報研報 Vol.98 No.37 pp.13-18(1998).

[2] 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.13 No.3 pp.574-583(1997).

[3] 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).

[4] Michael J. Flynn: “Computer Architecture,Pipelined and Parallel Processor Design”,Jones and Bartlett Publishers.

[5] David A.Patterson, John L.Hennessy(成田光彰 訳): “コンピュータの構成と設計”, 日経BP(1996).