QJavaプロセッサの設計とVerilogシミュレータによる動作検証

阿部俊輔　繁田聡一　Ben A. Abderazek
吉永努　曽和将容
電気通信大学　大学院情報システム学研究科

1. はじめに

Javaは、実行環境がプラットフォームに依存しないという長所によって広く普及しているが、実行速度が遅いという問題点がある．Javaの実行速度を向上させる1つの手法として、バイトコードレベルの並列性を抽出する研究が行われている[3]．しかし、Javaが採用しているスタック計算モデルでは、隣接するバイトコード間に依存関係が発生しやすく、バイトコードレベルの並列実行に適さない．一方、計算の途中結果をキューに格納するキュー計算モデルには、隣接命令間にデータ依存関係が発生しにくいという特徴があり、バイトコードレベルの並列実行に適していると考えられる．そこで、我々はJavaのスタック計算モデルから並列キュー計算モデルに置き換えたQJavaを提案している[1,2]．本研究ではその実行環境として、QJavaバイトコードをハードウェアで直接実行するQJavaプロセッサの設計を行い、動作検証を行った．

2. 並列キュー計算モデル

2.1. キュー計算モデル

キュー計算モデルとは、キューと呼ばれるFIFO(first in first out)のデータ構造を計算結果の格納場所として用いて計算を行う計算モデルである．Javaで採用されている、スタックと呼ばれるLIFO(last in first out)のデータ構造を計算結果の格納場所として用いる、スタック計算モデルと対照的な計算モデルである．

図2は、(a+b)/(c-d)を計算する命令列をキューを用いて計算をする様子を表したものである．キュー計算モデルではデータの取り出しはキューの先頭から、データの挿入はキューの末尾から行う．

2.2. 並列キュー計算モデル

図1の例では、最初の4つのload命令や、4番目のadd命令と5番目のsub命令の間には、データ依存関係がない．従って、キューに挿入されている先頭以外のデータと同時にアクセスすることを許せば、これらの命令は同時に実行できる．このように、キュー内の複数のデータを同時にアクセスできるように拡張した並列キューを用いて計算を行う計算モデルを「並列キュー計算モデル」と呼ぶ．

Design of a QJava Processor and Verification by Verilog Simulator
Graduate School of Information Systems, University of Electro-Communications

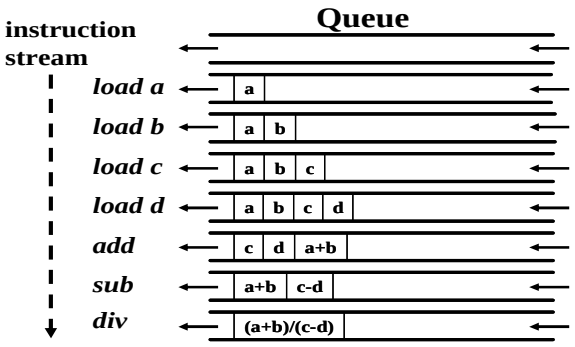

図1　キュー計算モデルによる命令実行の様子

3. QJava

QJavaとは、Javaのスタック計算モデルを、並列キュー計算モデルに置き換えたものである．よって、Javaのオペランドスタックはオペランドキューに置き換えられ、そこにデータを格納することになる．

3.1. QJavaバイトコード

QJavaバイトコードは、Javaのソースファイルを我々が開発した「QJavaコンパイラ[2]」でコンパイルすることによって生成される．QJavaのバイトコードは基本的にJavaのバイトコードと一緒であるが、キュー計算モデルを実現するために、いくつかのバイトコードの動作定義を変更した．また、Javaバイトコードにはない命令を新たに追加した．

- push命令：Javaバイトコードでは、スタックトップへの挿入であるが、QJavaバイトコードでは、キューの末尾への挿入となる．
- dup命令：Javaバイトコードでは、スタックトップの値をスタックトップに複製している．この動作をスタックトップからオペランドを1つ取り出して同じ値をスタックトップへ2つ挿入していると解釈し、QJavaバイトコードでは、キューの先頭からオペランドを1つ取り出して同じ値をキューの末尾に2つ挿入する
- rotate命令：QJavaバイトコードには、キューの先頭にあるオペランドをそのままキューの末尾に移動させる命令を新たに追加する．この命令は、キューに挿入されたデータの順序の入れ替えが必要な場合に用いられる．

3.2. QJavaの実行環境

QJavaバイトコードの実行環境の提供方法として、QJVM(QJava Virtual Machine)やQJIT(QJava Just In Time)コンパイラをソフトウェアにより実装する方法をと、バイトコードを直接実行できるQJava

プロセッサをハードウェアにより実装する方法が考えられる．

本研究では、QJava バイトコードの並列実行を実現するために、後者の方法を採用した．

4. QJava プロセッサ

QJava プロセッサは、QJava バイトコード直接実行するプロセッサである．並列実行可能なバイトコードを同時に実行するスーパースカラープロセッサとして設計した．パイプラインは、命令フェッチ、命令デコード、キュー割付、命令発行、実行、ライトバックの 6 段である．

4.1. 構成ユニット

QJava プロセッサは以下のユニットから構成される．

- **フェッチユニット(FU)**

  フェッチカウンタの示すメモリアドレスから毎サイクル 12 バイトずつ命令をフェッチし、内部バッファに格納する．分岐が確定した場合は、分岐ユニットから分岐先のターゲットアドレスを受け取り、そのメモリアドレスから命令をフェッチする．

- **デコードユニット(DU)**

  QJava バイトコードは固定長命令ではないので、フェッチされた命令列を命令長を解析しながら、デコードを行わなければならない．各命令がオペランドキューから消費するデータ数と生産するデータ数、命令の種別を付加した内部形式に変換する．

- **キューヘッド・テイル計算ユニット(QCU)**

  内部形式のデータ消費数と生産数の情報から、その命令が使うべきオペランドキューのヘッドとテイルの値（位置）を計算する．

- **命令発行ユニット(IIU)**

  内部に実行ユニットの種類だけ専用のバッファを持っており、各バッファはキューとして実現されている．デコードの際に付け加えられた命令の種別によって格納するバッファを選択し、対応するバッファのテイルに命令を格納する．また、バッファヘッドの命令が使うデータが揃っていて、且つ演算に使うユニットが busy でない場合に実行ユニットへ命令を発行する．

- **実行ユニット(EXU)**

  命令の処理を実際に行うユニットである。実行ユニットには、ALU、FPU、分岐ユニット(BU)、Load/Store ユニット(LS)、キュー操作ユニット(QC)を含む．それぞれ、キュー管理ユニット(QAU)からデータをもらい、演算結果を QAU へ返す．今回の設計では、ALU、FPU、Load/Store ユニットは複数用意し、それぞれ、4 個、2 個、2 個とした．依存関係がない命令は並列に実行できるようする．

- **キュー管理ユニット(QAU)**

  オペランドキューを管理するユニットである．QAU では演算のソースとなるオペランドキュー内のデータを各実行ユニットに送り、演算結果が返ってきたら、その値をオペランドキューに格納する．

  オペランドキューの各エントリにはデータレディビットと呼ばれる、そのエントリにデータが入っているか否かを示すフラグを用意している．データレディビットが立っている場合はそのエントリのデータは有効で、下がっている場合にはそのエントリのデータは無効である．データを実行ユニットに送る際に、そのエントリのデータレディビットを下げ、データを格納する際に立てる．

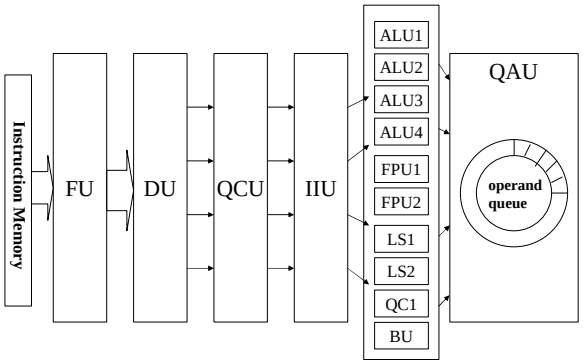

図 2 QJava プロセッサの概要

5. 動作確認

Verilog-HDL を用いて RTL(Register Transfer Level)設計を行った。Verilog コンパイラには IcarusVerilog を使用した。

実行した QJava バイトコードは QJava コンパイラが出したもので、

- load/store を含む算術論理演算
- ループ
- 条件分岐・無条件分岐

などである．シミュレーションにより、これらを含むプログラムに対して、正確に動作することを確認した．

6. おわりに

QJava バイトコードを直接実行する QJava プロセッサの設計を行った．Verilog-HDL を用いて記述し、Verilog シミュレータを用いて動作検証を行った．Load/Store を含む算術論理演算、ループ・条件分岐・無常条件分岐を含むバイトコードについて正常に動作することを確認できた．

現在対応できていない QJava バイトコードがいくつかあり、それに対応させ全ての命令に対して処理を行えるようにすることが課題である．

参 考 文 献

[1]繁田 聡一、王 立強、Ben A. Abderazek、吉永 努、曽和 将容: バイトコードレベルの高い並列性を持つ QJava の提案、先進的計算基盤システムシンポジウム SACSIS 2003, pp.77-80 2003

[2]L.Wang, B.A.Abderazek, S.Shigeta, T.Yoshinaga, and M.Sowa, "QJAVAC: Queue-Java Compiler Design for High Parallelism Queue Java Bytecode", Proceedings of International Technical Conference on Circuits/Systems Computers and Communications, Vol.2, pp.900-903 (2003)

[3]Motoki Kimura, Morgan Hirosuke Miki, Takao Onoye, Isao Shirakawa: Implementation of Java Accelerator for High-Performance Embedded Systems, IEICE TRANCE. FUNDAMENTALS, VOL.E86-A, NO.12 DECEMBER 2000