XX-XX

卒業論文

LLVMコンパイラ基盤を用いた
ベクトル化コード生成についての検討

2022年3月

永池 晃太朗

宇都宮大学工学部
情報工学科

内容梗概

日本語アブストラクト

Consideration of Vectorized Code Generation
Using LLVM Compiler Infrastructure

Kotaro Nagaike

Abstract

ENGLISH abst

目 次

第1章　はじめに

FPGA(Field Programmable Gate Array) はユーザによって回路の再構成が可能な LSI であり，目的の処理をハードウェアとして実装可能なデバイスである．近年 FPGA の大容量化，高性能化によって大規模な回路が実現可能になった．これにより FPGA は自動運転を始めとする組込み分野での利用増加が期待されている．

FPGA を用いたハードウェア開発は HDL(Hardware Description Language) による RTL(Register Transfer Level) 設計が広く用いられている．RTL は FPGA 上に構成する回路の信号の流れや制御構造を直接設計できる一方，動作検証やデバックが難しく，短期間で複雑な処理の開発は困難である．[1] そのため，FPGA による開発期間を短縮する方法として，専用ハードディスクとプロセッサを用いたソフトウェアによる処理を組み合わせる方法が考えられる．FPGA 上のハードウェアリソースを用いて実装するプロセッサのことをソフトコアプロセッサという．一般的に組込みシステムではそのコストやサイズ，消費電力などに制限があるためハードウェア資源に制約があることが多く，メモリバンド幅が限られることからメモリシステムの性能が高くないことが多い．メモリシステムの性能が低いと演算性能を高くしてもメモリアクセスに時間がかかり，結果としてシステム全体の性能はメモリシステムの性能によって左右される．[2] そこで，ソフトコアプロセッサによる処理を考える．すべての処理を専用ハードウェア回路で行うのではなく，高い性能の求められない汎用的な処理についてはソフトコアプロセッサにて行うことにより開発する専用ハードウェア回路を削減でき，ソフトコアプロセッサの性能が向上して専用ハードウェア回路を削減しても求められた性能に達することができればハードウェアの開発コストを抑えることができる．

組込み分野では AI 技術に注目が集まっている．独立行政法人情報処理推進機構の調査によると，将来強化/新たに獲得したい技術として組込み/IoT 関連企業の 46%が AI 技術を挙げている．[3]

AI 技術の応用としては画像認識などの画像処理が行われる．画像処理では画像を構成する画素に対して同じ処理を行うようなものが多い．このように複数のデータに対して同じ演算を行う処理については，単一命令で複数データの処理を行う SIMD(Single Instruction, Multiple Data) や，複数命令を並列に実行する MIMD(Multiple Instruction, Multiple Data) による並列処理で高速化が可能である．[**?**, **?**] MIMD は複数の制御を並列化することができるため，異なる処理を同時に実行するアプリケーションでは有効である．しかし，プロセッサに複数の制御ユニットをもたせる必要があるた

め SIMD と比較するとハードウェアのコストが大きくなる．一方 SIMD は単一の制御ユニットで複数の演算ユニットを並列動作させるため制御ユニットのコストは低くなる．データ並列処理では複数のデータに対して同じ処理を実行するため SIMD によって並列処理が可能である．

SIMD による並列処理を行う場合，SIMD 命令は 1 命令で演算するデータ数が決まっている．そのため演算性能を上げるために同時演算数を変更すると機械語コードを作り直す必要がある．異なる同時演算数でも同一の機械語コードを利用可能とするためには，機械語コードが同時演算数に依存しないスケーラブルなベクトル拡張が必要である．スケーラブルなベクトル拡張により機械語コードを変更することなく，必要に応じて容易に同時演算数を増やし高性能化することが可能となる．

スケーラブルなベクトル拡張を実現したものとしてオープンな命令セットアーキテクチャである RISC-V[5] をベクトル拡張したベクトル拡張付き RISC-V[6] が提案されている．ベクトル拡張付き RISC-V は組込み機器に広く用いられている ARM のベクトル拡張である ARM SVE(Scalable Vector Extension)[7] の命令セットを参考に組み込み向けに RISC-V に拡張したものである．しかし，ベクトル拡張付き RISC-V に対応したコンパイラが存在していない．そこで解決策としてベクトル拡張付き RISC-V のベクトル命令のアセンブリコードを得るためのコンパイラの開発を検討した．

本論文では，第 2 章で現在の MIQS プロセッサについて述べる．第 3 章でベクトル拡張付き RISC-V 命令の生成のために利用したコンパイラ基盤である LLVM について述べる．第 4 章で実際に命令生成のための実装について述べる．第 5 章では実際のソースコードからアセンブリコードの出力を行った結果について述べる．

第2章　　MIQSプロセッサ

本章ではベクトル処理機能を持つソフトコアプロセッサであるMIQSプロセッサについて述べる．

2.1　RISC-V

RISC-Vはカルフォルニア大学バークレイ校が新たに開発したRISCの設計思想に基づく命令セットアーキテクチャ(ISA)である．RISC-Vはオープンな ISAであり，ライセンスが不要である．

従来のISAでは，後方バイナリ互換性を維持するために過去に拡張した命令すべてを実装する必要がある．しかし，このようなISAでは命令数が増加し複雑になる．そこでRISC-Vではモジュール式ISAを採用している．[8] RISC-Vのシステムの機能を独立したモジュールとして分け，アプリケーションに応じて拡張機能を組み込むかを選択できる．RISC-Vには必ず組み込まなければならない基本命令セット (RV32I) の他に，主な拡張機能として乗算及び除算 (RV32M)，単精度浮動小数点 (RV32F)，倍精度浮動小数点 (RV32D) 等がある．

RISC-Vの基本命令は6つの命令フォーマットで表すことができる．基本命令の命令フォーマットを図に示す．R形式は2つのソースレジスタを扱う形式，I形式は12ビットの即値を扱う命令，S形式はストア命令，B形式は条件分岐命令，U形式は20ビットの即値を扱う命令，J形式は無条件ジャンプ命令の形式である．命令形式が単純であるため命令のデコーディングが単純化される．

2.2　MIQS概要

ベクトル拡張付きRISC-Vは機械語コードが同時演算数に依存しないスケーラブルなベクトル拡張を実現したISAであり，既存のRISC-VをARM SVEを参考にベクトル拡張したものである．ベクトル拡張付きRISC-Vではスケーラブルなベクトル拡張を行うためにプレディケートによるループ制御がある．プレディケートを用いたループ処理はループカウンタと処理する全データ数を比較してループカウンタが全データ数より小さい間対応するプレディケートレジスタをTrueにする．この命令によってベクトル処理で余りの要素がある場合，余りの要素の部分に対応するプレディケートの要素のみをTrueにする．これによって余りの要素が変化してもすべてのデータ数分ベクトル処理を行うことができ，同時演算数に依存しない機械語コードが実現できる．

ベクトル命令はベクトルロード，ストア命令，ベクトル演算命令，ベクトル制御命令の 3 つに分けられる．RISC-V にはカスタム命令用にオペコード領域が 4 つ用意されているがそのうちの 2 つを利用しており，1 つをベクトルロード，ストア命令，もう一方をベクトル演算命令とベクトル制御命令に使用している．ベクトルロード，ストア命令のアドレスはベースとなるスカラレジスタの値にオフセットを加えることで計算する．ベクトル演算命令は，プレディケートあり演算命令，プレディケートなし演算命令，即値による演算命令に分けられる．浮動小数点命令は必要なハードウェア資源が多いためサポートしていない．ベクトル拡張付き RISC-V は基本命令に関して RV32I のみ組み込んでいる．

レジスタ構成はベクトルレジスタ v0-v31，ベクトルマスク制御に用いるためのプレディケートレジスタ vp0-vp7，プレディケートレジスタ同士の論理演算に用いるプレディケートレジスタ vp8-vp15，RISC-V の汎用レジスタ x0-x31，プログラムカウンタとなっている．ベクトルレジスタの長さは $128\times2^n (0 \leq n \leq 4)$ ビットで表され，汎用レジスタの幅は 32 ビットとなっている．

2.3 ベクトル拡張付き RISC-V

ベクトル拡張付き RISC-V は機械語コードが同時演算数に依存しないスケーラブルなベクトル拡張を実現した ISA であり，既存の RISC-V を ARM SVE を参考にベクトル拡張したものである．ベクトル拡張付き RISC-V ではスケーラブルなベクトル拡張を行うためにプレディケートによるループ制御がある．プレディケートを用いたループ処理はループカウンタと処理する全データ数を比較してループカウンタが全データ数より小さい間対応するプレディケートレジスタを True にする．この命令によってベクトル処理で余りの要素がある場合，余りの要素の部分に対応するプレディケートの要素のみを True にする．これによって余りの要素が変化してもすべてのデータ数分ベクトル処理を行うことができ，同時演算数に依存しない機械語コードが実現できる．

ベクトル命令はベクトルロード，ストア命令，ベクトル演算命令，ベクトル制御命令の 3 つに分けられる．RISC-V にはカスタム命令用にオペコード領域が 4 つ用意されているがそのうちの 2 つを利用しており，1 つをベクトルロード，ストア命令，もう一方をベクトル演算命令とベクトル制御命令に使用している．ベクトルロード，ストア命令のアドレスはベースとなるスカラレジスタの値にオフセットを加えることで計算する．ベクトル演算命令は，プレディケートあり演算命令，プレディケートなし演算命令，即値による演算命令に分けられる．浮動小数点命令は必要なハードウェア資源が多いためサポートしていない．ベクトル拡張付き RISC-V は基本命令に関して RV32I のみ組み込んでいる．

レジスタ構成はベクトルレジスタ v0-v31，ベクトルマスク制御に用いるためのプレディケートレジスタ vp0-vp7，プレディケートレジスタ同士の論理演算に用いるプレディケートレジスタ vp8-

vp15，RISC-V の汎用レジスタ x0-x31，プログラムカウンタとなっている．ベクトルレジスタの長さは $128\times2^n(0 \leq n \leq 4)$ ビットで表され，汎用レジスタの幅は 32 ビットとなっている．

第3章　　LLVM

3 章！ 黙々と書こう.

3.1　LLVM 概要

ここは 3.1 節です.

3.2　LLVM による自動ベクトル化機能

ここは 3.2 節です.

第4章　　ベクトル拡張付きRISC-Vコンパイラ

ここは4章です．終わるまで振り返るな…　文の推敲は後でもできる．

4.1　LLVMバックエンドにおける独自命令実装手法

ここは4.1節です．

4.2　ベクトル拡張付きRISC-V命令の定義

ここは4.2節です．

第5章　　評価と考察

5 章．終盤，もう少し．

5.1　生成されるアセンブリコードの評価

ここは 5.1 節です．

5.2　非対応命令についての考察

ここは 5.2 節です．

第6章　　おわりに

論文のまとめと課題．お疲れ様でした．

謝辞

本研究の機会を与えていただき，また，日頃から貴重な御意見，御指導いただいた，馬場 敬信教授，大津 金光准教授，大川 猛助教，横田 隆史教授，に深く感謝致します．そして，本研究において多大な御力添えを頂いた，Boaz Jessie Jackin 氏をはじめとする研究室, オプティクス教育研究センターの方々に感謝致します.

参考文献

[1] 三好健文 : “FPGA 向けの高位合成言語と処理系の研究動向,” コンピュータソフトウェア. Vol.30, No.1, pp.76-84, 2013.

[2] Samuel Williams, et al.: “Roofline: an insightful visual performance model for multicore architectures,” Communications of the ACM. Vol.52, No.4, pp.65-76, 2009.

[3] 独立行政法人情報処理推進機構 社会基盤センター: “2020 年度組込み/IoT 産業の動向把握等に関する調査,” 2021.

[4] デイビッド・A・パターソン, ジョン・L・ヘネシー著 成田 光彰 訳: “コンピュータの構成と設計 第 5 版,” 日経 BP 社. 2017.

[5] Andrew Waterman, Krste Asanovi: “The RISC-V Instruction Set Manual Volume I: Unprivileged ISA,Document Version 2.2,” 2017.

[6] Yoshiki Kimura,et al: “Proposal of Scalable Vector Instruc-tion Set for Embedded RISC-V Processor,” Proc. 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), Vol.1, pp.435-439, 2019.

[7] Nigel Stephens, et al: “The ARM Scalable Vector Extension,” IEEE Micro, Vol.37, No.2, pp.26-39, 2017.

[8] デイビッド・パターソン (著), アンドリュー・ウォーターマン (著), 成田 光彰 (訳): “RISC-V 原典オープンアーキテクチャのススメ,” 日経 BP 社, 2018.

[9] 平石康祐, 橋本瑛大, 大津金光, 大川猛, 横田隆史: “SIMD 拡張ソフトコアプロセッサのための効率的なメモリシステムの検討,” 情報処理学会第 80 回全国大会講演論文集, pp.115-116, 2018.