64ビット ARM 機械語コードの自動並列化に向けた コンパイラ中間言語生成の検討

長尾 聡希[†] 大津 金光[†] 横田 隆史[†]

[†]宇都宮大学 工学部 基盤工学科 情報電子オプティクスコース

1 はじめに

64ビット ARM プロセッサは、モバイル端末で広く用いられており、組み込み機器等でも今後利用が広がることが期待される。これらのプロセッサはマルチコア化されているため、アプリケーションプログラムの高性能化のために、マルチコアプロセッサの複数コア上で並列処理を行うことは、有効な手段の1つである。しかし、プログラムを並列化するためには、逐次処理コードを並列処理コードに作り替えなければならず、これは開発者にとって容易でない。

そこで、プログラムの逐次処理コードを入力とし、自動で並列処理コードへの変換を行う、自動並列化コンパイラ技術が重要となっている。しかしながら、これらの多くは高水準言語で記述されたソースコードを並列化の対象とするため、ソースコードを入手できないプログラムに対して並列化を行うことができない。

この問題に対し、我々は、プログラムの機械語コードをコンパイラ基盤 LLVM[1] の中間表現である LLVM IR で記述されたコードに変換し、IR コードに対し分析と並列化を行うバイナリレベル自動並列化システムを開発している。

本稿では、64ビット ARM 機械語コードに対するバイナリレベル自動並列化システムの使用を可能にすることを目的として、64ビット ARM 機械語コードから LLVM IR コードへの変換手法について検討する。

2 機械語プログラムを基にした自動並列化

機械語プログラムを基にした自動並列化の実現するバイナリレベル自動並列化システムは、機械語コードから LLVM IR への変換、IR コードの解析及び並列化指示文の挿入 [2]、並列 IR コードの生成、並列 IR コードの機械語コードへのコンパイルの4つの過程を経て、逐次処理コードを並列処理コードに変換する。64ビット ARM 機械語プログラムを基にした自動並列化の実現には、1つ目の過程である、64ビット ARM 機械語コードから LLVM IR コードへの変換の実現が必要となる。

LLVM IR コードの解析と並列化指示文の挿入を行うプログラムは、C 言語で記述されたコードを、コンパイラ基盤 LLVM の C 言語向けフロントエンドである Clang にて LLVM IR コードに変換したものを対象に

並列化を行うことを想定して開発されている。そのため、同じソースプログラムに対して Clang の生成する IR コードをオリジナルとし、それと同等の IR コードを生成することが出来れば、自動並列化が問題なく行えることが期待できる。

機械語コードからの変換により得られる IR コードと、Clang の生成する IR コードを比較して、制御構造の複雑化、メモリアクセスの増加、コードサイズの増加の3つの変化がある場合、並列化処理が難しくなると考えられ、これらの変化は小さいことが望ましいと考えられる。

そこで本稿では、先述の差が小さい IR コードを、オリジナルと同等の IR コードとし、これを 64ビット ARM 機械語コードからの変換により得ることを目標に、オリジナルと同等の IR コードを得ることのできる 64ビット ARM 機械語コードから LLVM IR コードへの変換手法を検討する。

3 機械語コードから LLVM IR への変換手段

64ビット ARM 機械語コードを LLVM IR コードに変換する手法として、機械語コードから LLVM IR コードへの変換を行う既存のソフトウェアの使用を検討する。機械語プログラムを基にした自動並列化を行うために、変換ソフトウェアは、64ビット ARM の機械語コードから LLVM IR コードへの変換が可能であることと、変換後の LLVM IR コードから任意のアーキテクチャの機械語コードへの再コンパイルが可能であることの2つの条件を満たしたものでなければならない。これらに当てはまるものとして、McSema[3]、rev.ng[4]の2つのソフトウェアの利用を検討する。ただし、本稿執筆時点では不具合により、rev.ng は 64ビット ARM の機械語コードから LLVM IR コードへの変換を行う場合に限り、変換後の機械語コードへの再コンパイルができない。

4 LLVM IR コードの評価

各ツールを使った変換により得られる LLVM IR コードが、オリジナルと同等のものであるかを評価するため、McSema と rev.ng から得られた IR コードの比較を行う。

本実験では、C 言語で記述された行列積の計算 (512x512) を行うプログラムを 64ビット ARM 向けにコンパイルした機械語コードに対し、McSema と rev.ng にて LLVM IR コードへの変換を行い、それぞれ得られた IR コードについて、元の C コードを Clang を用いて IR コードに変換したものととの比較を行う。

比較を行う項目は、制御フローグラフ、ロード命令

Investigation of Compiler Intermediate Language Generation Method for Automatic Parallelization from 64-bit ARM Machine Code

[†]Satoki Nagao, Kanemitsu Ootsu and Takashi Yokota
Information, Electronics, and Optics Course, Department of Fundamental Engineering, School of Engineering, Utsunomiya University ([†])

とストア命令の数を合計したメモリアクセス命令数、コードサイズである。これらの項目は、それぞれ制御構造の複雑化、メモリアクセスの増加、コードサイズの増加がどの程度起きているかを示す評価指標として用いる。また、IR コードの機械語コードへの再コンパイルが可能な McSema については、変換により得られた IR コードに対し x86_64 機械語コードへの再コンパイルを行い、変換前の機械語コードとの実行時間の比較も行う。これは、機械語コードの変換を経て並列化を行ったコードが、Clang から得られる IR コードを並列化した場合に比べて、どれくらいの性能が変化するかを評価するためのものである。また、再コンパイル先のアーキテクチャに x86_64 を選択した理由は、実行時間の計測を容易にするためである。

実験環境を表 1 に示す。本実験では、McSema は master ブランチの、rev.ng は develop ブランチの 2022 年 12 月 2 日時点での最新版を使用した。

それぞれの変換で得られた IR コードから、行列積の計算部分のみを切り出したコードの、コードサイズとメモリアクセス命令数と、再コンパイルしたコードの実行時間を表 2 に示す。括弧内の値は、Clang の結果を 1 とした場合の相対比である。

McSema と rev.ng から得られた IR コードのどちらにおいても、制御フローグラフ上は、Clang のものと同様に 3 重ループの制御構造が確認できたため、制御構造の複雑化は発生していないと考えられる。表 2 より、コードサイズにおいては、Clang のものと比較して、McSema からは 1.56 倍、rev.ng からは 12.50 倍のコードサイズの IR コードが得られ、両者ともにコードサイズの増加が発生した。メモリアクセス命令数においては、Clang のものと比較して、McSema は 1.2 倍、rev.ng は 8.5 倍となったため、両者ともにメモリアクセスの増加が発生したと考えられる。実行時間においては、Clang のものと比較して、McSema は 1.34 倍となったため、McSema を経て並列化したコードは、Clang から得られる IR コードを並列化した場合に比べて、1.34 倍程の実行時間になると予想される。

また、rev.ng から得られた IR コードでは、最適化によって命令の処理順が入れ替わることを防ぐために、元の機械語コード上での 1 命令分の処理が完了する毎に関数呼び出しを行うようになっている。並列化を行うループ内にこういった関数呼び出しが存在する場合、前処理として関数呼び出しの削除が必要となり、並列化処理の妨げとなる可能性がある。

したがって、McSema と rev.ng のどちらも、コードサイズとメモリアクセスの増加が著しく、バイナリレベル自動並列化システムで使用するには適さないと考える。

表 1: 実験環境

CPU	Intel Core i7-11700 2.50GHz 8 コア
OS	Ubuntu 20.04 LTS
メモリ	64GB DDR4
コンパイラ	Clang 14.0.0

表 2: 実験結果

	Clang	McSema	rev.ng
コードサイズ (バイト)	3889 (1.00)	6062 (1.56)	48621 (12.50)
メモリアクセス命令数	31 (1.0)	38 (1.2)	264 (8.5)
実行時間 (秒)	0.600 (1.00)	0.802 (1.34)	動作せず

5 おわりに

本稿では、64 ビット ARM 機械語コードを基にした自動並列化を実現するために、行列積の計算を行うプログラムを 64 ビット ARM 向けにコンパイルした機械語コードに対し、McSema、rev.ng の 2 つのソフトウェアにて LLVM IR コードへの変換を行い、得られた IR コードを Clang から得られたものと比較した。その結果、コードサイズとメモリアクセスの増加が発生していたことから、両者ともにバイナリレベル自動並列化システムで使用するには適さないことが分かった。

今後は、64 ビット ARM 以外の機械語コードから LLVM IR コードへの変換を行うソフトウェアを調査し、それらの中で目標とする IR コードを生成するものに対し、64 ビット ARM 機械語コードからの変換機能を実装する手法などを検討する。

謝辞

本研究は、一部日本学術振興会科学研究費補助金 (基盤研究 (C)20K11726) の援助による。

参考文献

- [1] Chris Lattner, Vikram Adve: “LLVM: a compilation framework for lifelong program analysis & transformation,” International Symposium on Code Generation and Optimization, pp. 75-86, 2004.
- [2] K. Jingu, et al.: “Directive-Based Parallelization of For-Loops at LLVM IR Level,” 2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp.421-426, 2019.
- [3] Artem Dinaburg and Andrew Ruef: “Mcsema: Static translation of x86 instructions to llvm,” ReCon 2014 Conference, 2014.
- [4] A. Di Federico, P. Fezzardi and G. Agosta: “rev.ng: A Multi-Architecture Framework for Reverse Engineering and Vulnerability Discovery,” 2018 International Carnahan Conference on Security Technology (ICCST), pp.1-5, 2018.