Gears OS on Raspberry Pi

145759E 氏名 桃原優 指導教員：河野 真治

1 Gears OS

当研究室では、Code Segment と Data Segment によって構成される Gears OS の開発を行っている。Gears OS は、並列プログラミングフレームワークの Cerium と 分散フレームワークの Alice の開発を通して得られた知見を元に開発を進めている。
Cerium はオブジェクト指向言語である C++ を用いて開発した並列プログラミングフレームワークである。Cell、マルチコア CPU、GPU を用いた並列実行をサポートしている。並列処理の単位として Task を記述し、Task に他の Task との依存関係を設定する事で並列実行を実現するが、データは汎用ポインタで Task に渡されるため、データの依存関係を保証できない。データの正しさや依存関係を保証できていないので、並列処理は行えるが、信頼性が低い。その知見から、並列分散処理 には Code の分割だけではなく Data の分割も必要である事が分かった。
Gears OS では 当研究室で開発している CbC を用いて Data Sement を定義し、実装を行っている。

2 CbC

Gears OS の実装には LLVM/Clang 上に実装した CbC(Continuation based C) を用いる。
CbC は Code Segment を基本的な処理単位とする。C の関数とは異なり返り値を持たないが、Code Segment の宣言は C の関数の構文と同じように行い、型に __code を用いる。
CbC は for 文や while 文といったループ制御構文を持たないので、ループ処理は自分自身への再帰的な継続を行う事で実現する。
現在の Code Segment から次の Code Segment への移動は goto の後に Code Segment 名と引数を並べて記述する。この goto による処理の遷移を継続と呼ぶ。C と異なり、戻り値を持たない Code Segment ではスタックに値を積んで行く必要が無くスタックは変更されない。このようなスタックに値を積まない継続を軽量継続と呼ぶ。この軽量継続により、並列化、ループ制御、関数コールとスタックの操作を意識した最適化がソースコードレベルで行えるようになる。

3 Code Gear と Data Gear

Gears OS では、プログラムの単位として Gear を用いる。Gear は並列実行の単位、データの分割、Gear 間の接続等になる。Code Gear はプログラムの処理そのものであり、任意の数の Data Gear を参照し、処理が完了すると任意の数の Data Gear に書き込む。Code Gear は接続された Data Gear 以外にアクセスできない。Code Segment と同じように Code Gear から次の Code Gear への処理の移動は goto の後に Code Gear の名前と引数を指定する事で実現できる。Data Gear はデータそのものを表す。int や 文字列などの Primitive Data Type を持っている。Gear の特徴として処理やデータの構造が Code Gear、Data Gear に閉じている事にある。これにより、実行時間、メモリ使用量などを予測可能なものにすることができる。

4 Raspberry Pi 上の Gears OS

本研究では、ARM で動くシングルボードコンピュータである Raspberry Pi 上で Gears OS を動かせるようになる事で、ハードウェア上でも信頼性があり、並列実行ができるプログラミングを を記述できるようになる事を目指している。
CbC を Raspberry Pi で動かすためのアプローチの手法を、I と T の形をした図の組み合わせによって説明を行う。I の上部分に cbclang や Xv6 などのソースコード名を、下部分にその機能の記述言語を記してある。T の下にある I は特別で、上に VM 下に VM を乗せている OS が記されている。 T の上部分は左に入力されるファイル、右に出力されるその機能によって出力されるファイルが記され、下部分にその機能の記述言語が記されている。
例として、cbclang のソースコード (I) と、 Raspberry Pi 上の clang (T) を図 1 に示す。

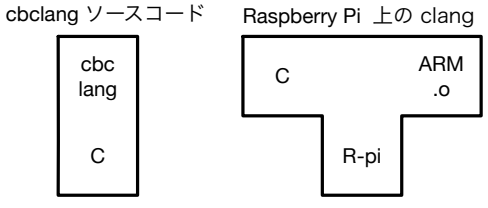

図 1: 図の例

5 Raspberry Pi 上での実装

Raspberry Pi 1 では、メモリが足りないため CbC をコンパイルすることができない。Raspberry Pi 3 だとコンパイルできるが、時間がかかる。qemu によるメモリの拡張もできないので、別の手法で Raspberry Pi 上に CbC を実装する方が好ましい。

Raspberry Pi でコンパイルを行うまでの過程を、図 2 に示す。

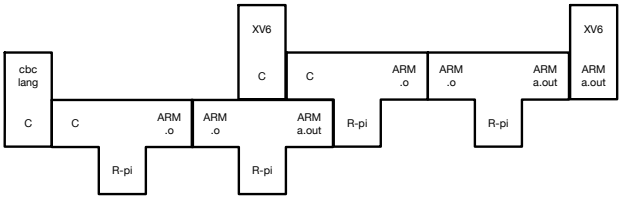

図 2: Raspberry Pi

6 LLVM CrossCompile

OSX 上で CrossCompile する事でコンパイルの時間の問題を解決する方法がある。
CrossCompile とは、別の OS で実行可能なコードを生成するコンパイルの手法である。arm-linux-gnueabihf-gcc を使用し、C で書かれたファイルを CrossCompile することで ARM のコードを生成できる事を確認した。この時、出力は mach-o になるので、mach-o の loader 作ることで Raspberry Pi で動かせるようになる。
まだ実装できていないが、OSX 上で行えるためコンパイルの速度向上が望める。OSX 上で コンパイルを行うまでの過程を、図 3 に示す。

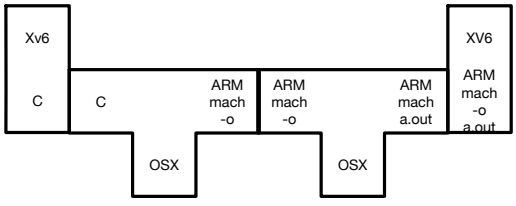

図 3: OSX

7 仮想マシン

CrossCompile と別に Linux 環境で CbC を動かした後に、Raspberry Pi に載せる手法がある。
Linux の LLVM でコンパイルすることができれば elf のコードを書けるようになるので、elf の loader を作る事で、Raspberry Pi で動くようになる。
また、Linux 用の gcc を CbC に書き直す際に、gcc7 に書き直せば linker がそのまま使えるので、Xv6 で動くようになる。
Xv6 とは、マサチューセッツ工科大の大学院生向け講義の教材として使うために、UNIX V6 という OS を ANSI-C に書き換え、x86 に移植した OS である。Xv6 は Raspberry Pi に移植する事ができる。ANSI-C で書かれている Xv6 を CbC に書き直す事で、Raspberry Pi で CbC を動かせるようになる。
Linux 上 で LLVM がコンパイルできない原因はまだ分かってないが、メモリを上げる事でこの方法でもコンパイルの速度向上が望める。Linux 上で コンパイルを行うまでの過程を、図 4 に示す。

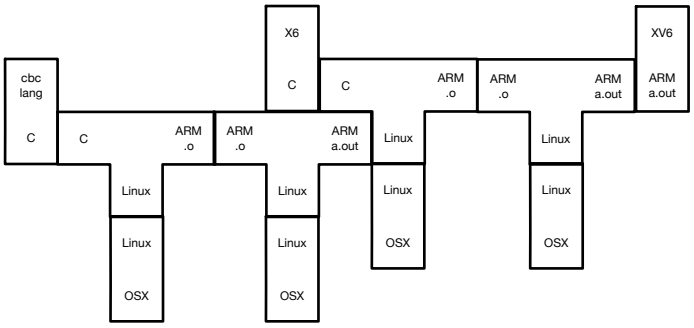

図 4: OSX Linux VM

8 今後の課題

Xv6 で CbC が動くようになれば、Raspberry Pi 以外のハードウェアでの実装も容易になるので、Linux 上での実装を目指して研究を進めていく。Xv6 で CbC が動けば、続けて Linux 上で Gears OS の実装も行なっていく。

参考文献

[1] 徳森 海斗, 河野真治. Llvm clang 上の continuation based c コンパイラ の改良, 2015.

[2] 伊波立樹, 東恩納琢偉, 河野真治. Code gear、data gear に基づく os のプロトタイプ, 2016.

[3] 小久保翔平, 河野真治. Code segment と data segment を持つ gears os の設計, 2016.

[4] The LLVM Compiler Infrastructure. `http://llvm.org`.