マイクロコントローラ向けスクリプト言語 MicroPython の組込みシステムへの適用性評価

2016SE007 藤田一希　2017SE114 板田怜子
指導教員：横山哲郎

1　はじめに

組込みシステムは低いハードウェアコストが要求されることが多いため，実行効率が良くメモリ使用量の少ない C/C++ 言語が使われていることが多い．C 言語は Java やスクリプト言語と比較すると，ソースコードをコンパイラにより機械語のコードに変換するため実行が高速であるというメリットがある．また，バックグラウンドで動く処理がなくアセンブリ言語と同様の処理を簡潔に記述することができるというメリットもある．

C 言語の問題点として Java やスクリプト言語に比べて生産性が低いことがあげられる．具体的には，プログラムの記述量が多く，言語仕様にないものは一から作成する必要があるので，開発にかける時間が増える．また IoT 機器だとエンドユーザーによるプログラミングが求められるケースがあり，C 言語では実現が難しいという点もある．C 言語の問題点を解決するために，Python のマイクロコントローラ向けの処理系である MicroPython や，同様に Ruby のマイクロコントローラ向けの mruby や mruby/c が開発されている．これらの処理系の定量的な性能評価はこれまでなされていない．

本研究では C 言語とマイクロコントローラ向けのスクリプト言語の処理系を比較して組込みシステムへの適用性を評価する．マイクロコントローラ向けのスクリプト言語としては，MicroPython[1] を用いた．

2　研究課題

本研究では以下を研究課題とする．

- MicroPython の実行性能の計測と評価.
- MicroPython のメモリ使用量の計測と評価.
- MicroPython の C 言語との可読性・生産性の比較.

3　背景技術

3.1　軽量スクリプト言語を用いた組込みシステムの実現

組込みシステムの機能は年々向上し，その構造は大規模かつ複雑なものへと変化している．一方，機能の向上に対してシステム自体の開発期間は短縮されつつある．そのような状況により，短い期間で高いパフォーマンスを提供できるシステムの開発に対応したプログラミング言語の開発が求められる．

短期間での組込みシステム開発の効率向上の手段として既存のスクリプト型言語を軽量化し，組込みシステム上で実行可能な仮想マシンを用いてプログラムを実行する方法がある．軽量化したスクリプト言語を軽量スクリプト言語と呼ぶ．仮想マシンを用いることにより，機器への依存性が低いことから多種多様な環境に対応できるという利点がある．また，ガベージコレクション機能の保有，簡潔な文法によるプログラムの記述が可能であり，既存の多くの言語がオブジェクト指向言語に対応しているため，生産性を向上することが可能である．図 1 に軽量スクリプト言語を取り入れたシステム構成を示す．

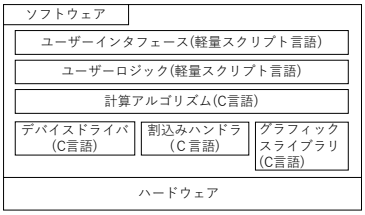

図 1　軽量スクリプト言語を取り入れた組込みシステム

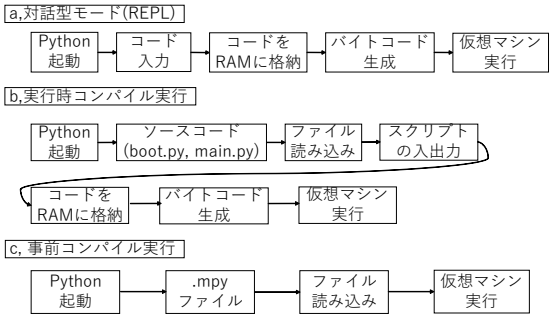

図 2　MicroPython の実行パターン

3.2　軽量スクリプト言語

これまで，幾つかの軽量スクリプト言語が研究・開発されている．本節では関連研究として，mruby と Elixir について説明する．本研究で扱う MicroPython については次節で説明する．

Web アプリケーションの分野において，Ruby と Ruby on Rails が多用されている．Ruby を組込みシステム開発でも利用可能にするために，Ruby を軽量化することによって組込みシステム開発に適した mruby と呼ばれる言語が開発された [2][3][5]．組込みシステムを搭載するハードウェアには利用するリソースに関する制限がある．mruby は容量が少ないメモリ上で動作することを目的としている．mruby によって生成されたバイトコードを実行するときに，仮想マシンにおいて 400 KB 程度のメモリを使用する．

Elixir とは，仮想マシン上で動作を行う関数型プログラ

ミング言語である [4]．Elixir は Erlang と呼ばれる仮想マシンで実装されており，並行処理や関数型などの特徴を持っている．さらに，Elixir のプログラムコードは軽量のプロセス上で動作しているため，同一の仮想マシン上で数千のプロセスが起動することがある．また，関数型プログラミング言語であることから，高い保守性や動作が高速であるといったメリットを持つ．

3.3 MicroPython

MicroPython は Python3 と互換性のあるスクリプト言語であり，マイクロコンピュータ上で動作することを目的として最適化されたものである [1][6]．言語仕様はバージョン 3.5 のコルーチン記述用の async/await キーワードや Type Hints 機能を用いている．

以下に MicroPython が備えている特徴について記述する．GPIO 等のデバイスを操作するクラスが事前に用意されており，それらのクラスを用いることでデバイスを用意に操作可能である．また，割込みハンドラはコールバック関数として定義され，ピンの ON, OFF などのイベントの発生によって実行される．なお不要になったメモリは，ガベージコレクションにより自動で回収される．

図 2 で MicroPython の 3 種類の実行パターンを示す．a では入力したコードを仮想マシン上で一行ごとに実行する．b ではフラッシュメモリからファイルを読み込み，入力されたスクリプトをバイトコードに変換する．その後，RAM にコードを格納し仮想マシン上で実行する．RAM でバイトコードを保持するので，サイズが大きいプログラムを実行できないという欠点を持つ．c では mpy-cross と呼ばれるクロスコンパイラによって，Python で記述したソースコード（.py ファイル）をバイトコードである.mpy ファイルに変換している．この方法によって，ボードでソースコードをコンパイルする必要がないので，ロード時間を短縮することが可能である．

MicroPyhton の性能を向上させるための 3 つの方法を説明する．1 つ目は，Python で変数や引数に関する処理を行い，高速な計算能力が必要とされる箇所では C 言語のモジュールを呼び出すことで実行速度を向上させるという方法である．MicroPython のスクリプト言語と C 言語で記述されたモジュールで同じ性能の機能を実装し，C 言語を用いた処理の性能が向上していることを確認する．

2 つ目はネイティブコードエミッタを使用する方法である．ネイティブコードエミッタにより，MicroPython のコンパイラはバイトコードではなくネイティブな CPU オペコードを生成する．しかし，性能の向上と引き換えに，コンパイルを行ったプログラムコードのサイズが大きくなるという欠点を持つ．

3 つ目は Viper コード・エミッタを使用する方法である．Viper コード・エミッタは，ネイティブコードエミッタより最適化が為された機械語命令を生成する．これにより，整数演算とビット演算の実行性能を向上させることが可能となる．しかし，性能の向上と引き換えに，関数が持つ引数や引数値に対する制限，浮動小数点数を使用したときは最適化が為されないという欠点を持つ．

4 評価項目の抽出

評価環境としては ST マクロ社の STM32 と呼ばれる Cortex-M をマイクロコントーラとして持つ性能の異なる複数実行環境を使用する．具体的には，F401（84MHz）・Pyboard（168MHz）・F746（216MHz）・H743（480MHz）の 4 種類を用いた．C 言語は OS なしのベアメタル環境で動作させる．MicroPython は GitHub に公開されている最新のソースコードをビルドして使用した．

MicroPython の組込みシステム適用を評価するため，MicroPython の機能と組込みシステムの要求から，次のような項目を選定した．

(a) 実行時間評価

(b) メモリ使用量評価

(c) 割込み応答時間評価

(d) GC の実行時間への影響評価

(e) 最適化機構評価

(f) C 言語呼び出し機構評価

（a）は 4 種類のプログラムを 3 種類の実行環境で C 言語と MicroPython で記述して実行性能を比較する．

（b）については，実行時コンパイル実行と事前コンパイル実行の 2 通りの計測を実施する．

（c）については，割込み応答時間とは，プロセッサに割込み要求が入ってから割込みハンドラが動作するまでの時間を示し，組込みシステムにおいては重要視される性能である．評価ボードを 2 つ繋げた時の割込み応答時間について計測を実施し，3 種類の実行環境においてどの程度の差が計測時間に現れるかを確認する．

（d）については，GC 発生時の実行時間は通常の実行時間に比べて増加する．そのため，最悪実行時間が大きくなるという問題が発生する．最悪実行時間とは，あるプログラムの実行を完了させるために必要とされる最も長い実行時間のことである．例えば，車のブレーキに使用されるシステムでは最悪実行時間がどのタイミングで発生するかを確認することは安全性の面において重要である．GC を意図的に発生させるプログラムを実行させることによって，最悪実行時間とその出現頻度の確認を行う．

（e）については，MicroPython プログラムにおいて，最適化手法を適用した場合にどの程度実行時間が変化するかを確認するために，最適化手法としては，ネイティブコードエミッター（native）とバイパーコードエミッター（viper）を用いてバブルソートプログラムの実行時間の比較を行う．

（f）については，micropython-ulab という数値計算ライブラリを用いて各実行環境での計算速度の比較を行った．

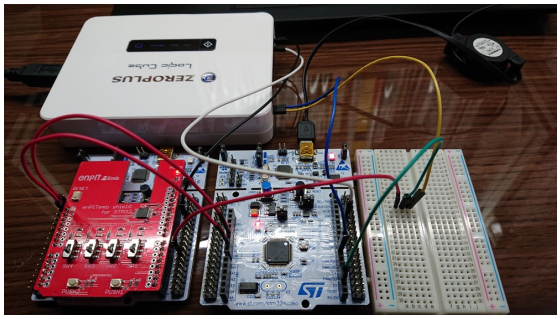

図 3　割込み応答時間の測定環境

表 1　実行時間評価の測定結果 [us] 及び増加率 [%]

	F401			H743		
要素数	6	12	24	6	12	24
C 言語（再帰）	5.2	10.6	25.4	0.6	1.3	2.9
C 言語（非再帰）	4.2	9.8	22.3	0.5	1.0	2.3
MicroPython（再帰）	961	2299	7709	138	322	1054
MicroPython（非再帰）	1122	2036	3876	151	275	514
増加率（再帰）	184	216	432	230	248	366
増加率（非再帰）	267	207	173	302	275	223

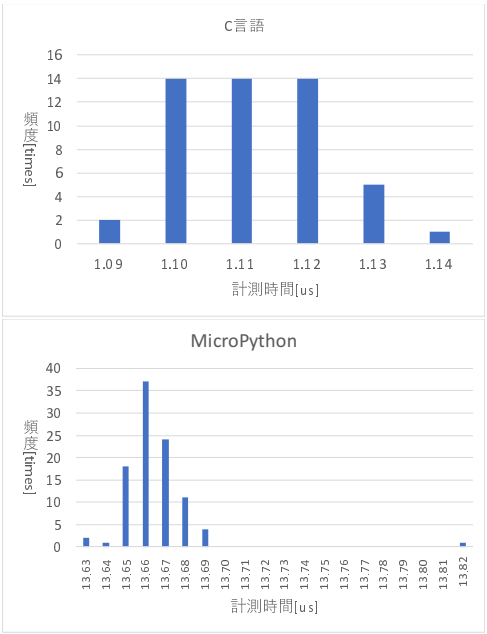

図 4　割込み応答時間の測定結果

5　評価手法

実行時間や割込み応答時間に関するにおいては，C 言語と MicroPython で同じ条件で計測を行う必要がある．各処理系で用意されている時間取得機能は精度が異なるため，本研究では，IO ポートとロジックアナライザを使用した計測方法を用いる．

図 3 に割込み応答時間時のロジックアナライザとマイコンボードの接続を示す．基本的な計測方法としては，計測対象のプログラムの実行前に，GPIO のポートレジスタに'1' を書き込み対応するチップの端子が High とする．計測対象のプログラムの実行後に，ポートレジスタの値を'0' とすることで対応するチップの端子が Low とする．そして，チップの端子をロジックアナライザで観測することで，プログラムの時間を計測する．

6　評価

6.1　実行時間評価

実行環境として F401 と H743 を用いた場合の測定結果を表 1 に示す．再帰クイックソートの実行時間は C 言語に比べて最小で 184 倍，最大で 366 倍 MicroPython が低速となった．実行環境の中で F401 がもっとも増加率が低かった．非再帰クイックソートの実行時間は C 言語に比べて最小で 173 倍，最大で 302 倍 MicroPython が低速となった．実行環境の中で F401 がもっとも増加率が低かった．再帰クイックソートと比較すると，要素数が多くなるほど，C 言語からの増加率が低くなる．これは，MicroPython の関数呼び出しの実行オーバヘッドが大きいためだと予想される．

6.2　メモリ使用量評価

実行時コンパイル実行はコンパイル可能な最大のファイルサイズは 31.291 KB，コンパイル不可能な最小のファイルサイズは 31.554 KB となった．事前コンパイル実行において，評価環境にコピーできたバイトコードの実行可能な最大ファイルサイズは 90.588 KB（188.359 KB）実行不可能な最小ファイルサイズは 90.660 KB（188.493 KB）となった．なお，() 内は，コンパイル元のファイルのサイズを示している．

計測結果から，事前コンパイルの方が大きなサイズで実行することができると言える．また事前コンパイル実行は約 90.588 KB でターゲット実行可能である．そして mpy-cross を用いることで約半分のプログラムサイズにコンパイルすることができる．バイトコードを経てコンパイルする必要がないため RAM の使用量を減らすことができるといえる．また実行時コンパイルでは，RAM の空き容量 165 KB に対してコンパイル可能なサイズが 31.291 KB であったため，使用可能なメモリサイズの 18 %程度のプログラムはコンパイル可能であることが分かった．

6.3　割込み応答時間

図 4 に F401 での C 言語と MicroPython での計測結果を示す．最頻値実行時間は C 言語のほうが約 12 倍高速である．表 1 での C 言語と MicroPython の実行時間の差は 100 倍程度であったのに対して，速度低下率が低い．これは，MicroPython において，割込みハンドラが呼び出されるまでの処理は C 言語で記述された MicroPython の処理系が動作しているためだと考えられる．ばらつきについては，最小値と最大値の差は，C 言語の場合は 0.25us，MicroPyton の場合は 1.13 us となった．

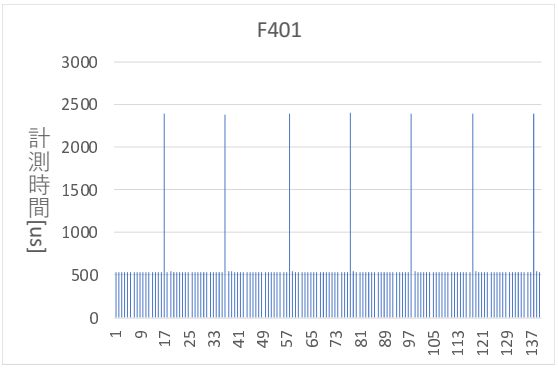

図 5　GC の実行時間への影響の測定結果

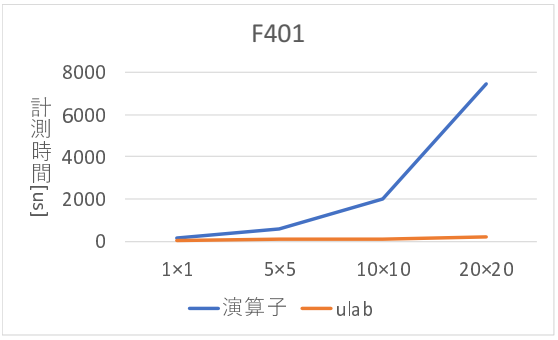

図 6　演算子と ulab を用いた計算の実行時間の比較

6.4　GC の実行時間への影響評価

バブルソートの処理を行うプログラムを F401 で繰り返し実行して実行毎の実行時間を計測した．図 5 に F401 で GC が発生した時の計測時間を示す．

実行環境 F401 において，定期的に実行時間が長い処理時間が発生しており，このタイミングで GC が発生していると，予想される．F401 は 20 回に 1 回 GC が発生している．GC 発生時のプログラムの実行時間は通常時の 5 倍程度増加していることが確認できる．

6.5　最適化機構評価

測定結果から，F401 では native で 1.5 倍，viper で 1.7 倍高速化した．一方，H743 では，native と viper が共に 1.1 倍高速化した．このことから，F401 が一番実行時間が遅く H743 が一番高速であり，各種マイコンボードの実行時間において，viper > native > normal の大小関係が確認できる．

従って，最適化処理においてネイティブコードエミッターを適用したコードのほうがネイティブコードエミッターを適用したコードより実行時間が高速であることが言える．

6.6　C 言語呼び出し機構評価

図 6 は MicroPython で直接計算した場合（演算子）と数値計算ライブラリを用いて計算した場合（ulab）の両方で 1 次元の計算を行った時の実行時間を示したものである．図 6 より，演算子を用いた計算と比べて ulab を用いた計算のほうが実行時間は高速であることが分かる．これは ulab の計算に使用される関数が C 言語で記述されているため，バイトコードを仮想マシン上に送信して計算を行う必要がないからであると考えられる．

計測結果である図 6 に注目する．20 × 20 のリストの計算において，演算子による計算の実行時間は ulab の場合の実行時間の約 35 倍程度であることが分かる．また，F401 の 1 次元の 1 × 1 の計算において，演算子と ulab の計算の実行時間の差は 83 us である．このことからも，従来の演算子での計算より ulab というライブラリを用いた計算の方が高速であると考えられる．

7　おわりに

本研究では，組込み開発における MicroPython のメリットとデメリットについて調査した．評価の結果，C 言語の割込み応答時間は MicroPython の応答時間と比べて約 12 倍高速であることや，プログラムコードへの最適化処理によるプログラム実行時間の高速化などが確認できた．今後の課題として，動的にメモリを確保して使用可能なメモリサイズを明らかにする事や周期処理のリアルタイム性の調査などが挙げられる．

参考文献

[1] 喜家村奨，高橋参吉，稲川考司，西野和典：MicroPython プログラミングで学ぶ情報技術，教育システム情報学会第 44 回全国大会論文集，pp.397–398 (2019).

[2] 永山将成，田中和明：軽量 Ruby のハードウェア制御ライブラリの実装に関する研究，平成 24 年度電気関係学会九州支部連合大会講演論文集，p.401 (2012).

[3] Tanaka, K., Nagumanthri, A. and Matsumoto, Y.: mruby – Rapid Software Development for Embedded Systems, *Proc. 15th Int'l Conf. on Computational Science and Its Applications* (*ICCSA 2015*), pp.27–32 (2015).

[4] 高瀬英希，河上晃治，菊池　豊：関数型言語 Elixir の IoT フレームワーク Nerves に関するリアルタイム性の評価およびその改善の試み，情報処理学会研究会，Vol.7, pp.1–8（2020）.

[5] 佐藤　弾：軽量 Ruby を使った組込みアプリケーションの開発，平成 25 年度電気関係学会九州支部連合大会講演論文集，p.397 (2013).

[6] Nicolas, V., Pierre, V. and Louis, F.: Increase Avionics Software Development Productivity using MicroPython and Jupyter Notebooks, *Proc. 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)* (2018).