Vol.2021-EMB-58 No.4
2021/11/18

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

板田 怜子[1]　藤田 一希[1]　横山 哲郎[1]　本田 晋也[1]

概要：組込みシステムは低コストが要求されるため，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．しかし，C/C++言語の問題点として Python や Ruby といったスクリプト言語に比べて記述の抽象度が低く生産性が低いことが挙げられる．この問題を解決するために，MicroPython や mruby といった既存のスクリプト言語を軽量化した処理系が開発されている．MicroPython とは Python3 と互換性のあるスクリプト言語であり，マイクロコントローラ上で動作することを目的として開発されている．本研究では MicroPython を対象として，組込みシステムの要件を満たすか評価した．具体的には，MicroPython の機能と組込みシステムの要求から，評価項目および評価方法を設計した．評価項目としては，実行時間評価，メモリ使用量評価，割込み応答時間評価，ガベージコレクションの実行時間への影響評価，最適化機構評価，C 言語呼出し機構評価の項目を選定した．選定した項目の評価手法を提案し，複数種類のマイクロコントローラを用いて評価を実施した．

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．2017 年度に行われた日本の組込みシステム開発に従事しているエンジニアを対象としたアンケートでは，「業務でよく使っている」言語として最多の 36％のエンジニアが C 言語を使っているという報告がある [1].

C 言語はスクリプト言語と比較すると，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機械語と同等の処理を記述することができるというメリットもある．一方，C 言語の問題点としてスクリプト言語に比べて生産性が低いことが挙げられる．具体的には，プログラムの抽象度が低く記述量が多くなる傾向があり，言語仕様にないものは 1 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 機器等では，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，その場合，C 言語はコンパイラ等の用意が必要となり，実現が難しいという点もある．

前述の 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．2014 年の ACM のブログ記事によればその年において米国の主要大学の計算機科学学部では Python が第一言語として最も採用されている．前述の日本のエンジニアを対象した調査においても，軽量スクリプト言語の一種である mruby を業務でよく使っていると回答したエンジニアが 4.5％存在する．この様な状況から，今後スクリプト言語を使える開発者が増加すると予想される．そのため，開発者の確保という観点からも組込みシステム開発へのスクリプト言語の適用は重要となってくると考えられる．特に MicroPython は Python をベースにしているため，有力な候補となる．一方，MicroPython は処理系の定量的な性能評価はこれまでなされていないという問題がある．組込みシステムは一般にリソース制約が厳しく，リアルタイム性が要求されるシステムも存在するが，それらに適用できるかは不明である．

本研究は，MicroPython の組込みシステムへの適用性の評価として，MicroPython の定量的な評価を実施することを目的とする．まず MicroPython の組込みシステム向けの機能について整理する．次に MicroPython を適用する組込みシステムを仮定し，その特性を整理する．特性としては，リソース制約とソフトリアルタイム性について着目した．これらの情報から，MicroPython の評価項目を抽出し，その項目の評価方法を定める．最後に，定めた評価方法に従って評価を実施する．評価は，C 言語のプログラム

[1] 南山大学 理工学部

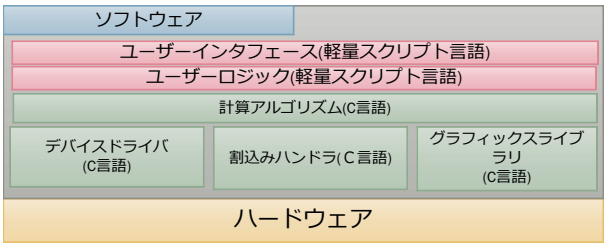

図 1 軽量スクリプト言語を取り入れたシステム構成

と比較可能な項目については比較を行う．評価環境については，組込みシステム向けのマイクロコントローラの性能は近年上昇している．これまでは 100 MHz 程度の動作周波数が主流であったが，400 MHz に近いものも登場している．スクリプト言語の実行は低速であるため，このような高速なマイクロコントローラの利用は不可欠であると考えられる．そのため，評価では従来の 100 MHz 程度のものと 400 MHz の高性能なマイクロコントローラの 2 種類を用いた．

2. 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．また，プロトタイプシステムを短期間で作成して使用感等を早く評価したいという要求や，GUI の実現のために開発効率のよい言語を使用したいという要求もある．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．開発効率が求められるユーザーインタフェースやユーザーロジックに軽量スクリプト言語を用い，性能が求められる計算アルゴリズム・デバイスドライバ・割込みハンドラ等は C 言語で記述し，組み合わせることを想定している．

2.1 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 [3][4][6]．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 [5]．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. MicroPython

MicroPython は Python 3 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 [2][7].

3.1 MicroPython による組込みシステムの記述

組込みシステムは，GPIO 等の周辺回路を直接操作する必要がある．MicroPython では，周辺機器ごとにクラスを用意しており，そのクラスを介して操作を時実現する．例えば，GPIO では，**図 2** のような操作で行う。

また，組込みシステムは，外界からのイベントを取得するために，割り込みを使用する．MicroPython では，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．

図 3 に MicroPython での割込みハンドラの記述例を示す．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．図 3 において，1 行目から 4 行目で割込みハンドラとなる関数を定義し，6 行目から 8 行目でポート D2 へ立ち上がり入力があれば，前述の関数を呼び出すように登録している，10 行目以降にメイン処理を記述している．

3.2 MicroPython の実行方式

図 4 に MicroPython の 2 種類の実行方式を示す．実行時コンパイル実行ではフラッシュメモリからファイルを読

```
1:  import pyb, machine
2:  led1 = pyb.Pin('D0', pyb.Pin.OUT_PP)
3:
4:  while True:
5:      led1.high() # LED1 を ON にする
6:      sleep(500) # 500 ms 待つ
7:      led1.low() # LED1 を OFF にする
8:      sleep(500) # 500 ms 待つ
```

図 2 MicroPython での GPIO の操作例

```
1:  def exti15_10_irqhandler(line):  # 割込みハンドラ
2:      led1.high()
3:      utime.sleep_us(1)
4:      led1.low()
5:
6:  def switch_push_init():  # 割込みハンドラの登録
7:      global ext
8:      ext = pyb.ExtInt("D2", pyb.ExtInt.IRQ_RISING,
        pyb.Pin.PULL_DOWN, exti15_10_irqhandler)
9:
10:  machine.disable_irq()
11:  switch_push_init()
12:  machine.enable_irq()
13:
14:  while True:
15:      pass
```

図 3 MicroPython での割込みハンドラの記述例

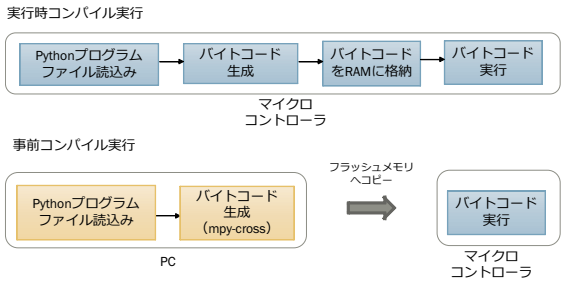

図 4 MicroPython の実行方式

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．マイクロコントローラは一般に ROM 容量に対して RAM 容量が小さい. この実行方式は，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．事前コンパイル実行では，PC 上で，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，mpy ファイルをマイクロコントローラの ROM（フラッシュメモリ）にコピーして実行する．

3.3 ガベージコレクション（GC）

MicroPython の実行系は GC の機構を持つ．但し，実行するタイミングおよび，使用しているアルゴリズムは不明である．

3.4 最適化機構

MicroPython はプログラム実行時にバイトコードを生成して仮想マシン上で実行するため，実行速度は高速ではない．実行速度を高速化するための手段として以下の 2 つの方法が提供されている．

- ネイティブコードエミッタ
- バイパーコードエミッタ

ネイティブコードエミッタについて説明する．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 の機械語を生成する．しかし，性能の向上と引き換えに，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．

次にバイパーコード・エミッタについて説明する．バイパーコード・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．

3.5 C 言語呼び出し機構

MicroPython は，C 言語の関数を呼び出すことが可能である．その一例として，ulab がある．Python で数値計算を行うプログラムを記述する時に NumPy と呼ばれるライブラリが使用されることが多い．しかしながら，MicroPython では NumPy を動作させることが不可能である．そこで，NumPy と同等の機能を提供するライブラリとして micropython-ulab（以下 ulab）が提供されている．ulab の制約として以下のものが挙げられる．

- 計算に使用可能な配列は 2 次元まで
- 配列形状の自動変換は不可
- 使用できる配列の型は uint8, int8, uint16, int16, float

ulab を用いることにより 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．

4. 評価項目及び評価手法

本章では，MicroPython の組込みシステムへの適用性を評価するための評価項目について説明する．まず，組込みシステムで求められる特性について述べた後，評価項目について示し，最後にその評価項目を評価する評価方法について説明する．

4.1 組込みシステムの特性

組込みシステムに求められる性質として代表的なものとして，リアルタイム性，制約条件下での動作，高い信頼性がある [8].

リアルタイム性はシステムによって定められた時間制約に従って動作する性質である．リアルタイム性にはハード (厳しい)，ソフト (緩い) リアルタイム性がある．ハードリアルタイム性とは，必ず定められた時間内で処理を完了することが求められる性質である．例えば医療機器や自動車など，時間制約を満たさない場合に人命等に関わる大きな問題となるシステムで要求される．ソフトリアルタイム性とは，規定時間内に処理が終わらなくても多少許容されるものである．例えば音楽プレーヤーなど，利用者の満足度は下がるが，大きな問題にはならない機器で要求される．

制約条件下での動作とは，コストダウン要請による低性能かつメモリ容量の小さいマイコンの使用，低消費電力，厳しい温度条件下でシステムを実現する必要があることである．信頼性に関しては，システムの誤作動が機器の誤作動に直結してしまうので，システムによっては高い信頼性が求められる．また，システム改修に高いコストがかかることや，PL 法の対象に含まれるため，出荷時に高い品質が求められる．

4.2 評価項目の抽出

評価項目の設計に関して，軽量スクリプト言語を適用する組込みステムの性質について仮定を行う．軽量スクリプト言語を使用する目的として，まずプロトタイプシステムへの適用が考えられるため，信頼性は重要でないと言える．また，ハードリアルタイム性が必要なシステムでは C 言語等のコンパイル型の言語を使うべきであると考える．一方，GUI 等を実現する場合でも，ある程度のリアルタイム性は必要となるため，ソフトリアルタイム性は必要とする．IoT 等のプロトタイプへの適用を考慮すると，使用可能なプロセッサは組込みシステム向けのマイコンとなる．これらは，PC と比較して性能が低く，メモリ容量も小さい．そのため，これらの制約条件下で動作する必要がある．以上から，軽量スクリプト言語を適用する組込みステムは次の特性を持つと仮定する．

- ソフトリアルタイム性（予測可能性）
- 制約条件下での動作（性能・メモリ）

これらの要求される特性と MicroPython の機能から，次のような評価項目を選定した．

(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 言語呼出し機構評価

(a) は制約条件下での動作（性能）についての評価である．性能を評価し，制約条件下での動作が可能かを評価する．既存のプログラムは C 言語で記述されていることが多いため，同一の処理を C 言語と MicroPython で記述して同じマイコンにおける実行性能を比較する．

(b) は，制約条件下での動作（メモリ）についての評価である．前述のように組込みシステム向けのマイコンのメモリ容量は PC と比較して小さいため，これらで動作するか実行時に必要なサイズを調査する．

(c) は，リアルタイム性についての評価である．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であり，組込みシステムにおいては重要視される性能である．MicroPython には前述のように割込みハンドラを記載する機能があるため，この機能のリアルタイム性を評価する．

(d) は，リアルタイム性の特性を満たすかを評価する．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．軽量スクリプト言語のランタイムによっては，インクリメンタル GC を実装している場合がある．しかしながら，MicroPython の GC のアルゴリズムは不明であるため，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) は，制約条件下での動作（性能）の特性を満たすかを評価する．実行性能が低い組込みシステム向けマイコン上で MicroPython の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．

(f) は，制約条件下での動作（性能）の特性を満たすかを評価する．MicroPython を高速化する手法である C 言語呼び出し機構を用いることで，どの程度性能が向上がするか評価する．

4.3 各評価項目の評価方法

(a) 実行時間評価に関しては，下記に示す 2 種類のプログラムを C 言語と MicroPython で記述して同一のマイコンにおける実行性能を比較する．それぞれ要素数を 6, 12, 24 として評価する．

- 再帰クイックソートプログラム
- 非再帰クイックソートプログラム

(b) メモリ使用量評価に関しては，実行時コンパイル実行と事前コンパイル実行の 2 通りの計測方法を行う．測定方法は，測定対象のプログラムのコードサイズを少しずつ増加させ実行時コンパイルおよび事前コンパイル可能性サ

イズをを計測する．

（c）割込み応答時間評価に関しては，メイン処理において，空の処理を実行している状態，クイックソートを繰り返し実行している状態において，割込みハンドラの応答時間がどのように変化するか評価する．割込みハンドラは，外部の他のマイクロコントローラから GPIO 経由で発生させた割込み信号により起動させる．

（d）GC の実行時間への影響評価に関しては，5 つの要素からなる 3 つのリストを連結する処理を 10 回実行する関数を用意し，メイン処理でこの関数を繰り返し呼び出す．この関数はメモリを消費し続けるため，GC が定期的に発生すると予想される．この関数の実行時間を計測することで，GC 発生時の関数の実行時間の変化を計測する．

（e）最適化機構評価に関しては，バブルソートのプログラムに対して，最適化を適用しない場合，ネイティブコードエミッターを適用した場合，バイパーコードエミッターを適用した場合についてそれぞれ実行時間を計測する．

（f）C 言語呼出し機構評価に関しては，1 次および 2 次リストの加算処理に関して，MicroPython の演算子での計算と，ulab を用いた計算の 2 つの方法の実行時間をそれぞれ計測して，その結果を比較する．それぞれのプログラムにおいてリストのサイズを変更して計測を行う．

5. 評価

前述の評価手法に従って評価を実施する．まず評価環境を説明したのち，各評価項目の評価結果について示す．

5.1 評価環境

5.1.1 評価対象のマイクロコントローラ

幾つかの評価項目では，プロセッサの種類による違いを評価するため，複数の実行環境を使用する．評価に用いた実行環境を表 1 に示す．評価ボードは ST マイクロ社の STM32 と呼ばれるマイクロコントローラを使用している．MicroPython は複数種類の SoC に対応しているが，主たる対応 SoC が STM32 シリーズであるため，STM32 シリーズを用いた．全ての SoC でプロセッサコアは ARM 社の Cortex-M シリーズである．F401/Pyboard は，Cortex-M4 を搭載しており，H743 は Cortex-M7 を搭載している．Cortex-M4 は家電やセンサー等の小規模な組込みシステム向けである．クロック周波数は 100 MHz 以下と低速であり，キャッシュを持たない．一方，Cortex-M7 は IoT や AI を実行する機器向けであり，Cortex-M4 と比較して，アーキテクチャは複雑であり，キャッシュを持ち，動作周波数は 400 MHz 以上である．

Pyboard は，MicroPython のオフシャルなボードで MicroPython 上のファイルシステムに対して，PC から直接書き込みが可能である．Pyboard は 1024KB のフラッシュメモリ，192KB の RAM の仕様である．そのため，メモリ使用量評価で用いる．

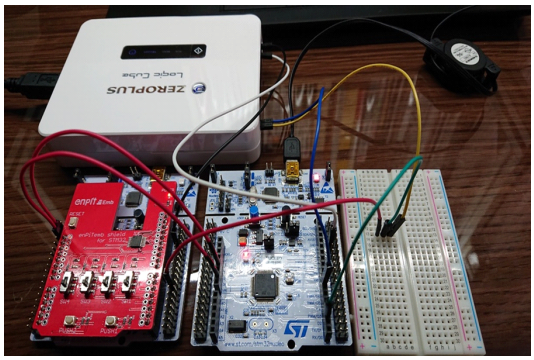

図 5 割込み応答時間の測定環境

5.1.2 計測方法

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．各環境における測定結果から実行オーバヘッドが引くことで，実質の実行時間を求める．

図 5 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を 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．

C 言語と MicroPython，各評価環境における実行オーバヘッドは表 2 の通りである．実行オーバヘッドは，それぞれの環境 (言語) で GPIO ポートを High にした直後に Low にして，その間のロジックアナライザで外部から計測した値である．本章で示す計測結果は，実際の計測結果からこの実行オーバヘッド分を引いた値としている．

5.1.3 C 言語のランタイムと開発環境

C 言語は OS なしのベアメタル環境で動作させる．C 言語の開発環境は表 3 に示す開発環境を用いてビルドする．実行時間評価の各種プログラムは CubeIDE を用いた．割込み応答時間に関しては，CubeIDE には適切なサンプルが無かったため，TrueStudio を使用した．

5.1.4 MicroPython

使用した MicroPyton のバージョンは，git のコミット ID 27767aafa21754e6cdbf52a719e7ce395f0fe672 時点レポジトリからビルドした物である．メモリ使用量評価において用いた MicroPython のコンパイラ（mpy-cross）は MicroPython と同じバージョンを使用し Windows 10 の WSL 上で実行した．

実行時間評価，割込み応答時間評価，ガベージコレクションの実行時間への影響評価，C 言語呼び出し機構評価

表 1 評価対象のマイクロコントローラ

名称	評価ボード	マイクロコントローラ	CORE	キャッシュ（KB）	周波数（MHz）	FLASH（KB）	RAM（KB）
F401	NUCLEO-F401RE	STM32F401RET6U	Cortex-M4	-	84	512	96
H743	NUCLEO-H743ZI2	STM32H743ZIT6U	Cortex-M7	16（I/D）	480	2048	1024

表 2 実行オーバヘッド [us]

	F401	H743
C 言語	0.2	0.10
MicroPython	12.07	1.55

表 3 C 言語の開発環境

評価項目	統合開発環境	コンパイラ
実行時間計測	CubeIDE v1.4	gcc v7.3.1
割込み応答時間評価	TrueStudio v9.3.0	gcc v6.3.1

表 4 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
再帰	C 言語	5.2	10.6	25.4	0.6	1.3	2.9
	MicroPython	961	2299	7709	138	322	1054
	増加率	184	216	432	230	248	366
非再帰	C 言語）	4.2	9.8	22.3	0.5	1.0	2.3
	MicroPython	1122	2036	3876	151	275	514
	増加率	267	207	173	302	275	223

において，プログラム及び事前コンパイルしたバイトコードの評価環境への転送は，コマンドラインツール ampy を用いた．

5.2 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 4 に示す．再帰クイックソートの実行時間において MicroPython は C 言語に比べて最小で 184 倍，最大で 303 倍低速となった．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において MicroPython は C 言語に比べて最小で 173 倍，最大で 302 倍低速となった．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．

5.3 メモリ使用量評価

Pyboard を対象に評価した結果，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約 31.3 KB，コンパイル不可能な最小のファイルサイズは約 31.5 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．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．

5.4 割込み応答時間

図 6 に，F401 でメイン処理で空のループを実行した場合の，C 言語での割込み応答時間および F401 と H743 で同様の処理を MicroPython で実現した場合の計測結果を示す．F401 において最頻値実行時間は C 言語の方が MicroPython より約 12 倍高速である．表 4 の実行時間評価においては，C 言語と MicroPython の実行時間の差は 200 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，F401 においては最小値と最大値の差は，C 言語の場合は 0.25 us，MicroPyton の場合は 1.13 us となり，MicroPython の方がばらつきが大きい．

次に実行環境間の差異について比較する．MicroPython での最頻値については，F401 は 13.7 us，H743 は 2.7 us であり，F401 に対して，H743 は 4.98 倍高速であった．動作周波数の割合は 5.74 倍なため，動作周波数の割合を少し下回る高速化率となった．実行時間の頻度の分布は実行環境ごとに異なり，F401 が小さい．H743 のばらつきが大きくなった理由としては，H743 はキャッシュを持つためだと考えられる．

次に，クイックソートをメイン処理で実行した場合の測定結果を**図 7** と**図 8** に示す．最頻値については，F401 が約 13.9 us，H743 が約 1.8 us である．空ループの測定結果と比較すると，F401 はほぼ変わらず，H743 は高速化している．一方，ばらつきに関しては，空ループ時の結果と比較すると，H743 のみばらつきが大きくなっており，200 us を超える大きな時間が計測された．GC が動作して低速になっている可能性があるが，一方，F401 で GC が発生しない理由は不明である．この問題の解析は今後の課題とする．

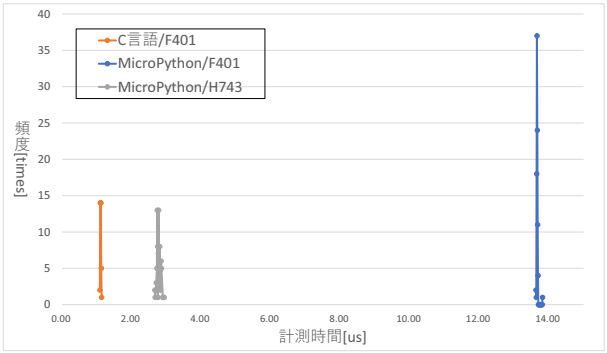

図 6 割込み応答時間の測定結果（空ループ）

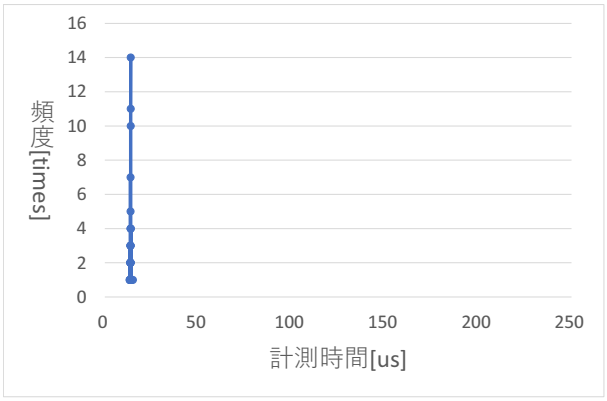

図 7 割込み応答時間の測定結果（クイックソート/F401）

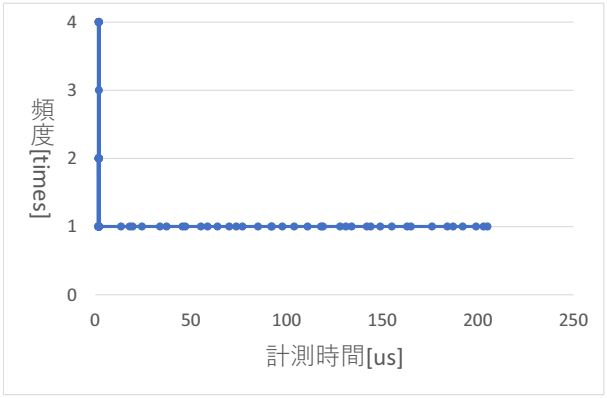

図 8 割込み応答時間の測定結果（クイックソート/H743）

5.5 GC の実行時間への影響評価

図 9 に，リスト操作のプログラムを繰り返し実行した際の 1 回ごとの実行時間の計測時間を示す．

両方の環境において，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．H743 は 159 回に 1 回発生している．発生間隔は残り RAM サイズに依存してると考えられ，RAM サイズが大きいほど間隔が大きいほど間隔が広くなっている．GC 発生時のプログラムの実行時間は F401 は通常時の 5 倍程度，H743 は 16 倍程度増加していることが確認できる．

F401 と比較して，H743 はメモリ容量が大きいので不要になったメモリを回収する頻度が少なく，その代わりに一度に回収するメモリが大きい．そのため通常時と比較した GC の実行時間の割合が大きくなっていると予想される．

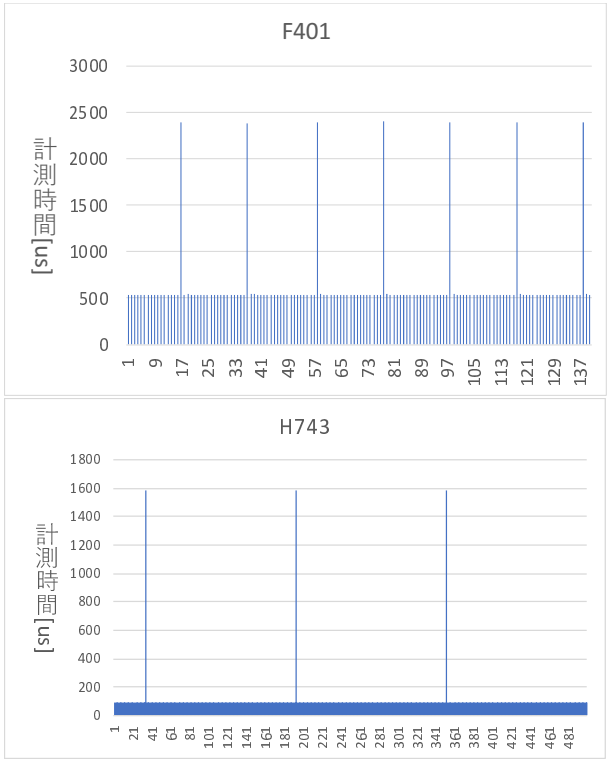

図 9 GC の実行時間への影響の測定結果

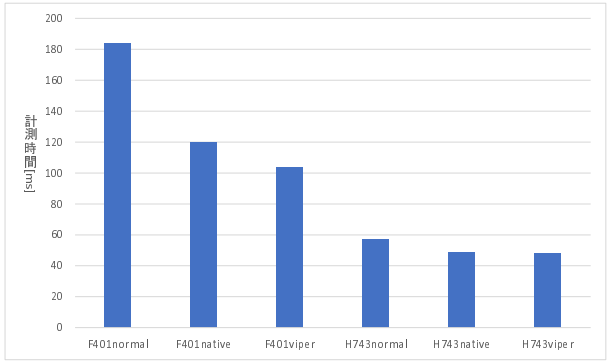

図 10 MicroPython 各種コード最適化 実行時間 [ms]

5.6 最適化機構評価

図 10 に測定結果を示す．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．実行環境により高速化率が異なることが分かった．H743 は動作周波数が高くキャッシュを持つため，最適化を実施しなくてもある程度高速に実行できるためだと予想される．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．

5.7 C 言語呼出し機構評価

図 11 に測定結果を示す．グラフの各線の計測時間の示

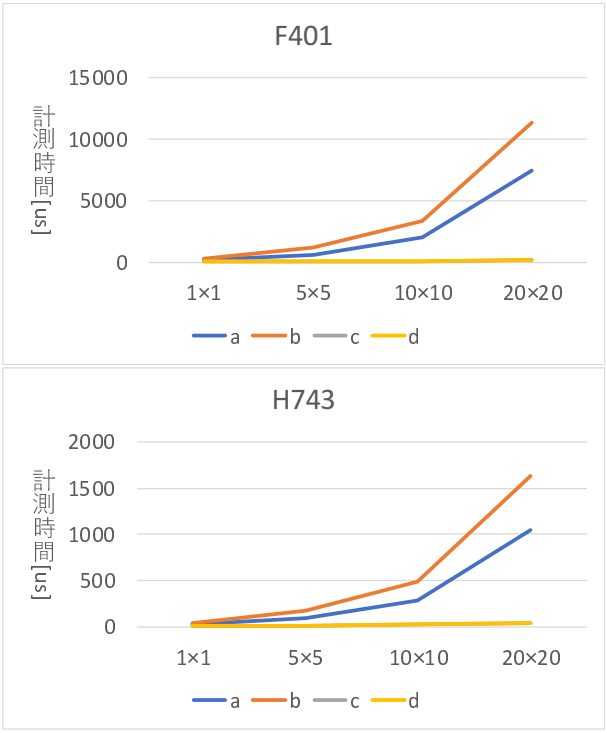

図 11 演算子と ulab を用いた計算の実行時間の比較

す内容は次の通りである．

(a) 演算子による 1 次元リストの演算
(b) 演算子による 2 次元リストの演算
(c) micropython-ulab による 1 次元リストの演算
(d) micropython-ulab による 2 次元リストの演算

図 11 の c と d に関しては計測された実行時間の差が微々たるものであったため，c と d のグラフの線が重なっている．F401 において，グラフが重なっている箇所の数値は次の通りである．1×1 の計算において c の実行時間が 53.9 us であり d の実行時間が 54.0 us，5×5 の計算において c の実行時間が 67.1 us であり d の実行時間が 69.8 us，10×10 の計算において c の実行時間が 93.5 us であり d の実行時間が 101.2 us，20×20 の計算において c の実行時間が 210.7 us であり d の実行時間が 212.2 us である．図 11 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大幅に高速化されていることが確認できた．

6. おわりに

本研究では，組込みシステムへの MicroPython の適用性の評価として，MicroPython の定量的な評価を実施した．評価の結果，MicroPython の応答時間は C 言語の割込み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] MEITEC：組み込み系エンジニアが選んだ「業務で使う」「好きな」言語／ OS は？, fabcross for エンジニア（オンライン），入手先〈https://engineer.fabcross.jp/archeive/170125_embedded.html〉(参照 2020-09-30).
[2] 喜家村奨, 高橋参吉, 稲川孝司, 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).
[3] 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).
[4] Tanaka, K., Nagumanthri, A.D. 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).
[5] 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研究報告，Vol.2020-EMB-7, pp.1–8（2020）.
[6] 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).
[7] 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).
[8] 渡辺 登, 牧野進二：組込みエンジニアの教科書, pp.10–19, pp.26–32(2020).