第1章

マイコンやるなら押さえておきたい

プログラム実行速度

藤井 裕也

リスト1　メイン・プログラムから呼び出される行列演算用の関数

```
void inline gemm(float *a, float *b, float *c,
           const int p, const int q, const int r) {
  for (int i=0; i<p; i++) {
    for (int j=0; j<r; j++) {
      for (int k=0; k<q; k++) {
        c[i*r+j] += a[i*q+k]*b[k*r+j];
      }
    }
  }
}
```

ESP32でアプリケーションを開発する場合，さまざまなプログラミング言語が使えます．開発者は自身の好みや，処理の内容と言語の得意不得意が合っているか，言語に必要なライブラリが備わっているか，などによって実際に使用する言語や開発環境を選んでいると思います．

ここでは，言語選択の一助となるよう，行列積の計算を題材に，ESP-IDFとArduino IDE，MicroPythonによる開発・実行環境の違いによるプログラムの性能差を比べてみます．

比較の条件として，Arduino IDEはデフォルトの設定，ESP-IDFは公式のサンプル・プログラムである`hello_world`と同じオプションで試してみます．

比較に使った，行列演算用のプログラムを**リスト1**に，メインのプログラムの抜粋を**リスト2**に示します．

Arduino IDEの実力

● 予想を裏切る結果…Arduino IDEの方が純正ESP-IDFより高速

同じコンパイラを使っているので，オプションの差はあれどどちらもあまり変わらないと予想していましたが，Arduino IDEの方が純正ESP-IDF開発環境より高速という意外な結果となりました．

- Arduino IDE：5,668μs
- ESP-IDF：13,728μs

要因は以下の2つです．

①デフォルトのクロック周波数の差
②ループに使われる命令の差

リスト2　メイン・プログラムの計測処理部分

```
const int p=59;
const int q=60;
const int r=61;
float *a = (float*)malloc(p*q*sizeof(float));
float *b = (float*)malloc(q*r*sizeof(float));
float *c = (float*)malloc(p*r*sizeof(float));

:
//初期化処理

int64_t prev_time = esp_timer_get_time();
gemm(a, b, c, p, q, r);
printf("val: %f, time: %lld\n", c[0],
                (esp_timer_get_time()-prev_time));
```

▶①デフォルト・クロック周波数の差

Arduino IDEではデフォルトのクロック周波数は最大の240MHzになっていますが，ESP-IDFのサンプル・プログラムの設定（`sdkconfig`）では160MHzになっています．

▶②ループに使われる命令の差

ESP32は，通常の分岐命令の他にzero-overheadループと呼ばれる命令を持っています．単純なループであれば，これを使って分岐命令のオーバヘッド注1なしで実行できます．

コンパイル・オプションを変えることでコンパイラによるプログラム最適化の程度を指示できます．

gccでは，`-O3`を指定してもこのzero-overheadループ命令を使ってくれないケースがあるのですが，g++を使うように変更するだけで`-Os`でもzero-overheadループ命令を使うようになる場合があり，今回はそのケースでした（xtensa-esp32-elf-gcc 5.2.0時点で確認）．

Arduino IDEではスケッチをC++のプログラムとして扱うため，コンパイルにはg++が使われるに対して，ESP-IDFのサンプルである`hello_world.c`はCのプログラムなので，通常はgccが使われます．

● 条件をそろえるとだいたい同じくらい

試しにESP-IDFでもg++を使うように変更してみると，zero-overheadループが使われるようになりま

注1：目的とする処理に直接関係のない2次的な作業．