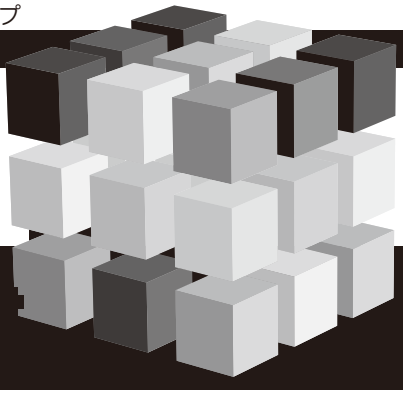
画像処理プログラムを例に…
最適化による処理高速化にトライ

ご購入はこちら

ラズパイ×Open CL GPUアプリ作り



第2回　RGB/グレースケール画像変換アプリの最適化

本橋 弘臣

表1　1次元の浮動小数点数の加算処理（chap8/opencl-add-float-opt3）との比較
前回作成したopencl-rgb2gsは，リード/ライトともに1バイトで1ピクセルぶんしか演算処理ができなかった

プログラム	①グローバル・ワーク・サイズの設定方法	②メモリ・アクセス量		③カーネル演算量
		リード	ライト	
opencl-add-float-opt3	1D	32バイト	16バイト	少ない
opencl-rgb2gs	2D	4バイト	4バイト	多い
opencl-rgb2gs-opt1	2D	16バイト	16バイト	多い

ラズベリー・パイ4/5には，SoC内蔵GPUとしてVideoCore VI/VII（ブロードコム）が搭載されています．このGPUは，OSにUbuntu 24.10を選択し，オープンソースのclvkというOpenCL互換レイヤ・ソフトウェアを組み合わせることで，OpenCL 3.0によるGPGPUプログラミングを行うことが可能です．

本連載では，clvkを使ったGPGPUプログラミング環境を使って，実際のアプリケーション・プログラムを開発します．clvkを使ったGPGPUプログラミング環境は，次に示すサポート・ページの手順で構築できます．

https://interface.cqpub.co.jp/202506_pigpu/

本連載で使用するプログラムのソースコードは本誌ウェブ・ページからダウンロードできます．

https://www.cqpub.co.jp/interface/download/contents2025.htm

参考文献(1)では，本環境の詳細や，OpenCLプログラミングの基礎知識，最適化手法などについて解説しています．（編集部）

RGB/グレースケール画像変換…②

uchar4×4（series2/opencl-rgb2gs-opt1）

● プログラムの概要

ここでは，前回（第1回，2025年8月号）解説したopencl-rgb2gsプログラムの改良版として，カーネル関数を1回実行する度に，一気に4ピクセル分の演算処理を行うように変更したプログラムを作成します．

この変更により，opencl-rgb2gs-opt1プログラムの動作は，**表1**の②メモリ・アクセス量の観点ではかなりopencl-add-float-opt3に近づくことになります．

リスト1　変更後（opencl-rgb2gs-opt1）のカーネル・コード

```
001:__kernel void kernel_func(__global uchar4
                                  *buff, uint stride)
002:{
003:  uint  x_gid = get_global_id(0) * 4;
                          // X方向のグローバルID値を4倍
004:  uint  y_gid = get_global_id(1);
005:  uchar4  in1_u8x4, in2_u8x4, in3_u8x4, in4_u8x4;
006:  uchar   gs1, gs2, gs3, gs4;
007:  size_t  off;
008:
009:  // 処理対象のピクセルデータのオフセット・アドレスを求める
010:  off = x_gid + (y_gid * stride / sizeof(
                                          *buff));
011:
012:  // メモリから RGBA 画像データを4ピクセル分読み込む
013:  in1_u8x4 = buff[off    ];
014:  in2_u8x4 = buff[off + 1];
015:  in3_u8x4 = buff[off + 2];
016:  in4_u8x4 = buff[off + 3];
017:
018:  // RGBA データをグレースケール(GS)データに変換する
019:  gs1 = (in1_u8x4.s0 / 4 + in1_u8x4.s0 / 16) …;
020:  gs2 = (in2_u8x4.s0 / 4 + in2_u8x4.s0 / 16) …;
021:  gs3 = (in3_u8x4.s0 / 4 + in3_u8x4.s0 / 16) …;
022:  gs4 = (in4_u8x4.s0 / 4 + in4_u8x4.s0 / 16) …;
023:
024:  // グレースケール化した RGBA 画像データをメモリに書き込む
025:  buff[off    ] = (uchar4)(gs1, gs1, gs1, 0xff);
026:  buff[off + 1] = (uchar4)(gs2, gs2, gs2, 0xff);
027:  buff[off + 2] = (uchar4)(gs3, gs3, gs3, 0xff);
028:  buff[off + 3] = (uchar4)(gs4, gs4, gs4, 0xff);
029:}
```

● カーネル・コードの内容

リスト1に変更後のカーネル・コードを示します．このカーネル関数の中で，メインの処理である13～28行目については，同じ処理を4回繰り返しているだ