FRPによるGPU上の計算の実現

櫻井義孝・森口草介・渡部卓雄(東京工業大学)

組込みシステム向けFRP言語 XFRP

関数リアクティブプログラミング言語（FRP）はプログラムの状態を**時変値**として表現してリアクティブシステムを構築するプログラミングパラダイムである．組込みシステム向けFRP言語としてXFRPがある． XFRPはEmfrp [Sawada2016]の後続言語でありEmfrpの解説は別に示している．

XFRPの問題点

XFRPで画像のような大きなデータを扱うプログラムを記述すると，システムの応答性能が低下する可能性がある．
本研究ではGPUを搭載するような組込みシステムにおいてGPUを用いて処理を高速化するために，
XFRPにGPU上で動作する時変値を導入する．

GPU Node

FRPによるGPU上の計算の実現のために，GPU上で計算する時変値を導入する．これを**GPUノード**と呼ぶ．
GPUノードは以下のようにノード配列として定義される．

```
gnode grayScale@960*1280 : Int = red[self] * 0.2126 + green[self] * 0.7152 + blue[self] * 0.0722
```

これはGPUノード**grayScale**を定義する．このGPUノード**grayScale**は960*1280個の**時変値の配列**になる．
GPUノードの更新式の定義にはキーワード**self**を使用できる．このキーワード**self**は配列の添字を意味し，更新式はself番目の配列の更新式の定義になる．そのため，上のように定義されたGPUノードは次のように指定された個数のノードを定義していることになる．

```
gnode grayScale0 : Int = red0 * 0.2126 + green0 * 0.7152 + blue0 * 0.0722
gnode grayScale1 : Int = red1 * 0.2126 + green1 * 0.7152 + blue1 * 0.0722
gnode grayScale2 : Int = red2 * 0.2126 + green2 * 0.7152 + blue2 * 0.0722
...
```

GPU Nodeの実装

本研究ではGPUを使用可能にするためにXFRPをCUDAコードを変換する．
下にサンプルアプリケーションとしてカメラから入力される画像をGPUを用いてグレースケール化するようなアプリケーションと，アプリケーション上でGPUを用いて計算される時変値がどのように変換されるかを示す．

```
module GrayScale
in image@960*1280: Int
out grayImage@960*1280: Float

# Separate inputs to RGB
gnode red@960*1280: Int   = (image[self] >> 16) & 255
gnode green@960*1280: Int = (image[self] >> 8) & 255
gnode blue@960*1280: Int  = image[self] & 255

# Degamma Correction on RGB
gnode red_rg@960*1280: Float   = pow(red[self] / 255.0, 2.2)
gnode green_rg@960*1280: Float = pow(green[self] / 255.0 , 2.2)
gnode blue_rg@960*1280: Float  = pow(blue[self] / 255.0 , 2.2)

# Gray Scale
gnode grayscale_rg@960*1280: Float = 0.2126 * red_rg[self] +
                                     0.7152 * green_rg[self] +
                                     0.0722 * blue_rg[self]

# Gamma Correction on GrayScale
gnode grayscale@960*1280: Float = pow(grayscale_rg[self], 1.0/2.2)

# Copy to Output
node grayImage@960*1280: Float = grayscale[self]
```

GPU上で計算されるノードgrayscaleは次のようなCUDAのコードに変換される．

```
float grayscale[2][1228800];
float* g_grayscale[2];   //GPU上のメモリ

// GPU上のメモリの初期化処理
void setup_grayscale(){
  for(int i=0;i<2;i++)
    cudaMalloc((void**)&g_grayscale[i],
                  1228800*sizeof(float));
}

// GPUで実行される関数(Kernel関数)
__global__ void grayscale_kernel(float* grayscale,
                                 float* grayscale_rg)
{
  int self = blockIdx.x * blockDim.x + threadIdx.x;
  if(self < 1228800){
    grayscale[self] = pow(grayscale_rg[self],
                                       (1. / 2.2));
  }
}

// GPU上の時変値grayscaleの更新関数
void grayscale_update(){
  // Kernel関数の呼び出し
  grayscale_kernel<<<dim3(2400),dim3(512)>>>
        (g_grayscale[turn], g_grayscale_rg[turn]);

  // CPU側へのメモリの転送
  cudaMemcpy(grayscale[turn],
             g_grayscale[turn],
             1228800 * sizeof(float),
             cudaMemcpyDeviceToHost);
}
```

性能評価

GPU Nodeを利用したXFRPの性能を確認するため，上のソースコードで実装される画像のグレースケール化アプリケーションを用いて性能評価を行った．評価環境にはJetson Nano (GPU:128コアMaxwell, CPU:ARM A57)を使用した．

画像サイズ (px)	960x1280	480x1280	480x640
変換時間 (ms)	163.58	82.136	41.052