第4章 行列演算を例に試行錯誤したら処理時間が30ms→21msに

GPUの能力を使い切るトライ

鈴木 量三朗

ここでは具体的な例として，GPUがその性能を発揮できそうな処理として行列演算を行い，理論性能にどこまで迫れるか試してみます．GPUで実行するカーネル・プログラムを**リスト1**に示します．

このプログラムは行列の1つ1つの要素を並列に計算します．CUDAは`for`文の中身を並列化するので，このプログラム自身には`for`文で使うインデックス用の変数は出てきません．その代わり，与えられた変数`gridDim`，`blockDim`，`blockIdx`，`threadIdx`の値をもとに，自身の場所（インデックス）を特定し必要な積和演算をします．

カーネルを呼び出すホスト側では，メモリ・アクセスを考えブロック分割をします．今回は，256×256の行列の積の演算を32×32のブロックで区切るようにしてみます．この区切り方がデバイス上でのメモリ・アクセスを規定します．また，CUDAでは最大のスレッド数が1024なので，正方であれば32×32が最大になります．カーネルをホストから呼ぶコードを次に示します．

```
naiveMatrixMul<<<grid, threads>>>(d_C, d_A, d_B);
```

理論性能をフルに使いきるのは難しい

● リソースの5%も使えていない

筆者の環境では512回の繰り返しで処理時間は30ms程度でした．これは，おおよそ554GFLOPSの処理速度になります．

この計算にはホスト・デバイス間のデータ転送時間が入っていません．また，512回の繰り返しに同じメモリ領域を使っているため，キャッシュ・ヒットを考えると純粋な計算時間でもありません．おおよその目安と考えてください．NVIDIAのサンプル・プログラムでも似たような計測をしているのでここではそれに合わせました．

GeForce RTX 3060は32ビット浮動小数点数の積和演算で最高性能が12.74TFLOPSですから，5%も使っていないことになります．

リスト1　行列演算のカーネル

```
__global__ void naiveMatrixMul(float *C, float *A,
                                          float *B)
{
  int bx = blockIdx.x;
  int by = blockIdx.y;

  int tx = threadIdx.x;
  int ty = threadIdx.y;

  unsigned int c_index_x = bx * blockDim.x + tx;
  unsigned int c_index_y = by * blockDim.y + ty;
  unsigned int width = gridDim.x * blockDim.x;
  unsigned int height = gridDim.y * blockDim.y;
  assert(width == height);

  unsigned int a_index = c_index_y * width;
  unsigned int b_index = c_index_x;
  float result = 0.0;

  for( unsigned int i = 0 ; i < width; i++ ) {
    result += A[a_index] * B[b_index];
    a_index += 1;
    b_index += width;
  }

  C[c_index_y * width + c_index_x] = result;
}
```

● メモリの使い方とアルゴリズムに問題がある

処理性能が低い理由と，その改善点は幾つか考えられます．

- 配列Bのアドレス・アクセスがとびとびで効率が悪い
- 事前に計算すればよいことをカーネル内で計算している

さらなる性能向上を考えるとメモリ・アクセスを向上させる必要があるでしょう．行列の(x, y)という要素が必要とする領域を**図1**に示します．ここで，隣の($x+1$, y)を考えると，その必要とする領域は**図2**のようになります．それらは重なっている部分もあるのですが，重なっていない部分もあります．