第8章 工夫次第で処理速度は10倍以上に！
ボトルネックになりがちなコピー処理の高速化も

最適化③…浮動小数点数対応と非同期実行，ゼロ・コピー処理

本橋 弘臣

表1　OpenCLに用意されている最適化手法
本稿では浮動小数点数への対応を行った上で，非同期実行化とゼロ・コピー処理の活用について紹介する

GPUの性能最適化手法	変更箇所	備　考
ローカル・ワーク・サイズの最適化	ホスト・コード	説明済み
処理ベクタ長の最適化	カーネル・コード	説明済み
GPU 処理の非同期実行化	ホスト・コード	本稿で説明
ゼロ・コピー処理の活用	ホスト・コード	本稿で説明

ここまでで，まだ説明していない最適化方法を含めると，OpenCLではGPUの最適化手法として**表1**のような方法が用意されています．

本稿では，GPUで次の処理を行うプログラムを題材に，処理性能を最適化していきます．

- 機能：大量のfloatデータに対して加算処理を行う（C = A + B）
- 処理対象のデータ個数：変数a，b，c各データが全て128M個（512Mバイト）

1 浮動小数点数（float）への対応と最適化

コード①：最適化前のベース

・データ型：float，・lws：1，・gws：128M

● ソースコードの内容

▶データ型をfloatに変更

最適化を行う前のシンプルなコードを最初に示します．プログラムはchap8/opencl-add-floatに格納されています．まずカーネル・コードは**リスト1**（a）（次頁）のように至って単純なコードです．次に，ホスト・コードの主要部分を**リスト1**（b）に示します．このコードとは，カーネル・コードで取り扱うデータの型がfloatに変わっただけで，第2部 第6章までに登場したサンプル・プログラムと比べて，特に新しい要素は含まれていません．

▶GPU処理時間の測定＆表示機能を追加

リスト1（b）には示していませんが，実際のホスト・コードにはGPUで実行している各処理ステップごとの処理時間を測定して，プログラムの最後で処理時間を表示する機能が追加されています．このため，ホスト・コードの先頭では，**リスト1**（c）のように，まずベンチマーク・テストの実行回数分のイベント・オブジェクト変数を宣言しています．event変数が2次元配列になっているのは，次に示す4つのGPU処理ステップごとの処理時間を測定するようにしているからです（**表2**）．

表2　新たにコードに追加したGPUの処理時間測定機能における測定項目
4つの処理ステップごとに処理時間を測定する．そのためevent変数が2次元配列になっている

測定項目	使用する変数
GPU入力データのコピー（mem_obj_a：ホスト→VRAM）	event[][0]
GPU 入力データのコピー（mem_obj_b：ホスト→VRAM）	event[][1]
カーネル実行	event[][2]
GPU 出力データのコピー（mem_obj_c：VRAM→ホスト）	event[][3]

● 実行結果…GPU処理速度は0.8Gバイト/s

リスト2にこのopencl-add-floatプログラムの実行結果を示します．ホスト・メモリ上のバッファをCPUが初期化するのに1761msの時間が掛かっているので，CPU→Memoryの書き込み速度は0.9Gバイト/sに過ぎなかったことが分かります．一方でGPUのカーネル・コードが128M個のfloat値（AとB）を加算して，加算結果をVRAMに書き込むのに平均で