

# OpenCL

## 並列プログラミング

マルチコアCPU/GPUのための標準フレームワーク

池田成樹◎著



本書で取り上げられているシステム名／製品名は、一般に開発各社の登録商標／商品名です。本書では、™ および® マークは明記していません。本書に掲載されている団体／商品に対して、その商標権を侵害する意図は一切ありません。本書で紹介している URL や各サイトの内容は変更される場合があります。

## はじめに

ある時期までは高性能コンピューティング、つまりスーパーコンピューターを使っただけの演算処理には関心がなかった。理由は簡単で、スーパーコンピューターを使えるような立場ではないので、実際に動かしてみる機会がなかったのと、iPhone アプリを作成するのに夢中だったからである。

それに、数値計算の奥深さについてはかねてから話には聞いていたので、その世界に踏み込むのは恐れ多いとも感じていた。

だが、ある機会に GPGPU や GPU コンピューティングの話聞き及んで、まだ枯れきっていない技術であることを知って少し興味を持った。その後 OpenCL を知り、Mac OS X の新バージョン (Snow Leopard) では、標準搭載されることも知った。さらに、OpenCL は組み込みや携帯機器にも適用される方向性があるという説明で iPhone でも OpenCL が稼働する可能性が出てきたことを把握すると、これはやっておかないといけないということで OpenCL に関していろいろ調べ始めた。

その過程で NVIDIA 社の GPU コンピューティング環境である CUDA の存在を知り、こちらも資料を探したりして、GPU コンピューティングというものとその方向性に関しての理解を深めることができた。

そこで感じたことの結論は、これからの時代に GPU コンピューティングはプログラム開発技術者の必須に近い技術であるということだ。それに、この先において GPU と呼ばれるものの概念はさらに拡張されていくと思われる。

およそ 15 年ほど前には、オブジェクト指向を知らないで商売にならないみたいな風潮があった。GPU コンピューティングの基本となる概念である並列プログラミングはそこまでではないかもしれないが、今後重要なパラダイムとして無視することはできないのは明らかだろう。

たまたまではあるが、昨年末に日本の国家予算の審議でスーパーコンピューターに対する予算の査定についての論争が話題になった。GPU コンピューティングはその消費電力に対する性能と、導入コストに対する性能で、従来の CPU クラスタによるコンピューティングより優位にある。これからさらに GPU コンピューティングに関する事例や研究が増えてノウハウが蓄積されていくだろう。その段階になって改めて予算に対する疑問を行ったとしたら、おそらくその質と量に関して違った見解がまとまってくるのではないだろうか。

本書は並列コンピューティングになじみや経験のない読者を前提にしている。OpenMP や MPI などの CPU ベースの並列処理環境に関する経験も前提にしていない。マルチスレッドプログラミングの経験があったほうが理解はスムーズだろうが、これも経験を前提にはしていない。本書を読むとマルチスレッドプログラミングと OpenCL プログラミングはかなり異質なものであることがわかるだろう。

また、CUDA に関する知識や経験も前提にしていないが、OpenCL の仕様策定で概念として下敷きとなっているので、参考になる点が多いのは確かである。

なお、筆者は数値計算の専門家ではない。その点はあらかじめお断りしておく。付録の参考文献リストで参考にした数値計算関係の書籍や論文を挙げておいた。従来 CPU で行われてきた並列化に対する最適化は、そのまま OpenCL に持ち込むことは難しいが、基本的なテクニックとしては有効である。

また、紙面の都合と、チューニングには搭載している GPU のスペックを把握しないと効果が薄いことから、実行性能の観点での最適化というトピックには踏み込んでいない。これに関しては機会を与えられるならば、取り組んでみたいと考えている。そして執筆段階においては、本文で言及した NVIDIA 社の Fermi 世代の GPU に関して正式な製品としての発売などに関する情報は出てこなかったもので、当然試用するチャンスもなかったことを申し添えておく。Fermi 世代の GPU が普及する段階になると GPU コンピューティングの様相は大きく変わってくると予想しているので、筆者としてはその発売を心待ちにしている。

最後になるが、製品写真の提供や GPU コンピューティングに対する取り組みの話を聞かせてくださった NVIDIA 社日本法人広報の中村かおり氏と、同様にお話を聞かせてくださった OpenCL の仕様策定団体であるクロノス・グループ広報の河西仁氏にこの場を借りて御礼を申し上げたい。

本文においての数学的な記述に関しては、数学科の学生である高岡邦行氏に原稿を読んでもらって貴重な意見をいただいたのと、現役の研究者の豊沢聡氏にも意見を願った。御礼申し上げたい。

また、いつもながらきついスケジュールで協力をいただいた編集の鈴木光治氏と本書の企画を快諾していただいたカットシステムの石塚勝敏氏にも御礼申し上げます。

2010 年 1 月 調布にて Steely Dan を聴きながら 池田成樹

## ■使用した機材について

メインのマシンは MacBook Pro であり、サブのマシンは自作マシンである。以下にマシンのスペックを紹介しておく。

- MacBook Pro

Intel Core 2 Duo 2.5GHz

メインメモリ 4GB

Geforce 8600M GT VRAM 512MB

OS は Snow Leopard をインストールしている。

- 自作マシン

Intel Pentium 4 3.0GHz HT 対応

メインメモリ 1GB

GeForce 8400GS (PCI 接続) VRAM 512MB

かなり古い世代のマザーボードなので拡張スロットに PCI Express が存在しない。GPU カードは PCI で接続されているので、本来の性能は発揮できていないはずである。

OS は Linux (Ubuntu 9.10) と Windows XP SP3 をインストールして切り替えてブートしている。

## ■サンプルプログラムについて

下記のサイトで本書で扱ったサンプルプログラムのソースコードをダウンロードできる。

<http://www.daikichi.net/>

# 目次

はじめに.....	iii
-----------	-----

## 第1章 「計算機」としてのコンピューターアーキテクチャ概説.....1

1.1 CPUの性能向上.....	2
1.1.1 ムーアの法則.....	2
1.1.2 プロセッサの性能向上手法と障害.....	4
1.1.3 インテルアーキテクチャの変遷.....	5
1.1.4 マルチコアの時代へ.....	15
1.2 GPUの歴史と機能の変遷.....	17
1.2.1 GPU 誕生以前.....	17
1.2.2 GPU 誕生.....	18
1.2.3 プログラマブルシェーダ.....	19
1.2.4 統合型シェーダ.....	21
1.3 GPU コンピューティングへの道筋.....	23
1.3.1 GPU の演算能力.....	23
1.3.2 GPGPU.....	24
1.3.3 CUDA.....	26
1.3.4 GPU コンピューティングへ.....	27

## 第2章 並行処理と並列処理.....33

2.1 マルチプロセスとマルチスレッド.....	34
2.1.1 マルチプロセス.....	34
2.1.2 マルチスレッド.....	36

2.2	並行なのか、並列なのか？ .....	37
2.3	パフォーマンスの問題 .....	38
2.3.1	アムダールの法則 .....	38
2.3.2	グスタフソンの法則 .....	40
2.4	並列化におけるアプローチ .....	41
2.4.1	データ並列化 .....	41
2.4.2	タスク並列化 .....	42
2.5	排他制御 .....	43
2.5.1	データアクセスが競合すると問題がある場合 .....	43
2.5.2	ロックによる同期 .....	45
2.5.3	"Think Parallel!"—並列処理を考える指針 .....	47

## 第3章 OpenCL のアーキテクチャ.....49

3.1	OpenCL 概説 .....	50
3.2	プラットフォームモデル .....	52
3.2.1	フリンによる並列処理の分類 .....	53
3.2.2	SPMD と MPMD .....	54
3.3	実行モデル .....	56
3.3.1	NDRange .....	56
3.3.2	ワークアイテム .....	56
3.3.3	ワークグループ .....	57
3.3.4	ID を表す形式と取り得る値 .....	57
3.3.5	コンテキスト .....	59
3.3.6	コマンドキュー .....	60
3.3.7	カーネルの分類 .....	62
3.4	メモリモデル .....	62
3.4.1	メモリアクセスの分類 .....	62
3.4.2	ホストからのメモリアクセス手法 .....	64
3.4.3	メモリアクセスの一貫性 .....	65

3.5	プログラミングモデル .....	67
3.5.1	データ並列モデル .....	67
3.5.2	タスク並列モデル .....	67
3.5.3	同期.....	68
3.6	OpenCL フレームワーク.....	69
3.6.1	OpenCL フレームワーク.....	69
3.6.2	OpenCL ランタイム.....	70

## 第4章 OpenCL プログラミング.....71

4.1	プラットフォームとデバイス情報の取得.....	72
4.2	カーネルプログラムの作成.....	101

## 第5章 OpenCL による数値計算 (1) .....115

5.1	浮動小数点数による演算.....	116
5.1.1	浮動小数点数の表現 (IEEE 754).....	116
5.1.2	浮動小数点数表現における誤差の発生.....	118
5.2	並列処理による数値計算.....	122
5.2.1	級数の総和計算.....	122
5.3	行列の計算.....	135
5.3.1	行列の積.....	135
5.3.2	行列の成分への参照とループ処理.....	136
5.3.3	行列の積演算の並列化 .....	140

## 第 6 章 OpenCL による数値計算 (2) ……153

6.1 LU 分解.....	154
6.1.1 LU 分解の用途.....	154
6.1.2 LU 分解の方法.....	157
6.1.3 LU 分解のプログラミングへの適用.....	164

## 付録 A OpenCL リファレンス……195

A.1 データ型.....	196
A.1.1 スカラー値型.....	196
A.1.2 ベクター値型.....	200
A.2 型変換とキャスト.....	206
A.2.1 暗黙の型変換.....	206
A.2.2 明示的なキャスト.....	206
A.2.3 明示的な変換.....	207
A.2.4 他データ型への再解釈.....	210
A.3 演算子.....	213
A.4 修飾子.....	214
A.5 関数リファレンス.....	216
A.5.1 ワークアイテム関数.....	216
A.5.2 数学関数.....	217
A.5.3 整数値の関数.....	227
A.5.4 共通関数.....	230
A.5.5 幾何関数.....	232
A.5.6 比較関数.....	233
A.5.7 ベクター値のロード/ストア関数.....	236

**A.6 同期関数.....241**

- A.6.1 バリア同期関数.....241
- A.6.2 メモリフェンス関数.....242
- A.6.3 非同期グローバル、ローカルメモリ間コピー関数.....243

**付録 B OpenCL SDK のインストール.....245**

- B.1 SDK のダウンロードサイト.....246
- B.2 Windows の場合.....247
- B.3 Linux の場合.....248
- B.4 Mac OS X の場合.....248

**付録 C プログラムのビルド.....249**

- C.1 Windows でのビルド.....250
- C.2 Linux でのビルド.....257
- C.3 Mac OS X (Snow Leopard) でのビルド.....257

**引用・参考文献.....263****索引.....267**



## 1.2 GPUの歴史と機能の変遷

まず、GPUの誕生に至る過程とその機能の発展について解説する。

### 1.2.1 GPU誕生以前

そもそも、PCの世界ではVRAM（ビデオメモリ）に直接CPUがアクセスして描画を行っていたのを補助するために、グラフィックアクセラータと呼ばれるチップを搭載することでその負担を軽減していた。

だがこの時代はあくまでも2Dでの描画がその対象であり、3Dに関する演算はCPUによって行わせる必要があった。3D演算をグラフィックチップで処理するのは、ジオメトリエンジンという名称で最初SGI社が開発した。SGI社による3D演算の基盤はOpenGLに引き継がれていく。

そのため、そういった3D演算をグラフィックチップで行う処理環境はワークステーションと呼ばれる機種やゲームセンターに置いてあるゲームマシン（バーチャファイターなどで遊んだ記憶がある読者は多いと思われる）での世界だった。

1994年にソニー・コンピュータエンタテインメント社が開発したプレイステーションは家庭用のゲーム機にジオメトリエンジンを搭載し、3D表示のゲームを家庭で楽しむコンセプトを定着させた。その人気と普及に関しては詳しく述べる必要はないだろう。

PCでも同じように3D表示を楽しみたいという需要に応えるために、マイクロソフト社は1996年にDirect3Dを発表した。そもそもMS-DOS上で開発されることが多かったゲームソフトをWindowsで開発することを促すために、1995年にDirectXを発表していたが、それに3Dグラフィック表示機能を追加するためのものだった。

DirectXやDirect3DはあくまでもAPIであり、グラフィックチップそのものを規定するものではなかったが、APIの仕様が拡張されて高度になるのに伴って、グラフィックチップで処理できる演算も高度化されていった。

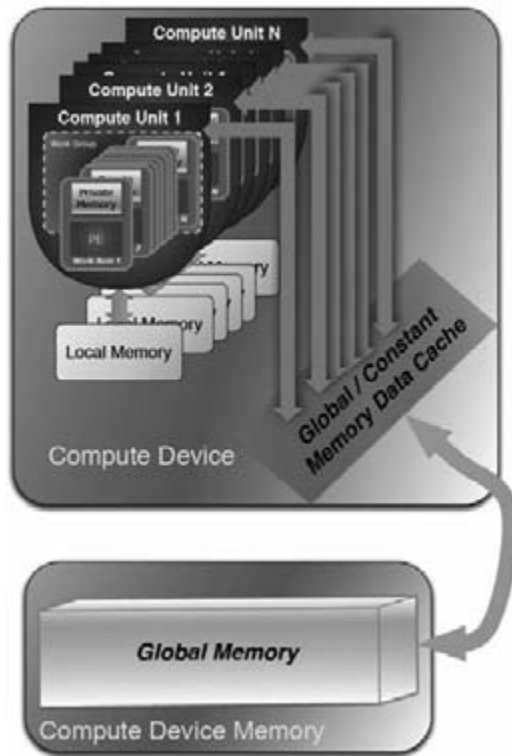


図 3-7 メモリアクセスの概念図

### 3.4.2 ホストからのメモリアクセス手法

ホスト側では、OpenCL API の呼び出しとメモリコマンドのキューへの投入によってメモリの管理を行う。また、ホストと OpenCL デバイス上のメモリは別個のものとして運用される。言い換えると、ホストから OpenCL デバイス上のメモリ空間は独立しており、かつ直接アクセスすることはグローバルメモリが対象といえども許されていない。

ただし、OpenCL API の明示的な使用によって、以下に示す 2 つの方法を通してアクセスすることは可能である。

- データをホスト側のメモリ空間にコピーする

ホスト側にメモリオブジェクトをコピーすることで、ホストからのアクセスを可能にする。デバイスとは双方向でコピーが可能である。つまり、読み込みも書き込みも可能で

番目の引数は文字列の配列を受け取る際の要素数（行数）を指定するが、ソースコードでは1つの文字列にまとめてしまっているため、1を指定する。

読み込んだカーネルプログラムは `cl_program` 構造体によって保持される。

### ■カーネルプログラムのビルド

```
err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

カーネルプログラムをコンパイルして実行可能にするには、`clBuildProgram` 関数で行う。最初の引数は `cl_program` 構造体のポインタを指定している。次の引数はデバイスリストの個数である。次の3番目の引数でデバイスリストを指定するが、`NULL` としているためデバイスリストの個数に0を指定する。

デバイスリストを指定すると、指定されたデバイスに即したコンパイルが行われる。`NULL` を指定した場合は、`cl_program` 構造体に関連づけられたすべてのデバイスに即してコンパイルが行われる。

4番目の引数はコンパイルオプションを指定する。`NULL` であれば、オプションなしである。残り2つの引数はコールバックに関連するものなので、これも `NULL` にして無効にしてある。

### ■カーネルの生成

```
kernel = clCreateKernel(program, "VectorAdd", &err);
```

カーネルオブジェクトを `clCreateKernel` 関数によって生成する。2番目の引数は実行させたいカーネルプログラムにおける関数名である。ホストプログラムのように `main` 関数からと決まっているわけではない。

### ■カーネルへ引数の設定

```
GetAndCHK(clSetKernelArg(kernel, 0, sizeof(cl_mem), (void*)&mem_a));  
GetAndCHK(clSetKernelArg(kernel, 1, sizeof(cl_mem), (void*)&mem_b));  
GetAndCHK(clSetKernelArg(kernel, 2, sizeof(cl_mem), (void*)&mem_c));  
GetAndCHK(clSetKernelArg(kernel, 3, sizeof(cl_int), (void*)&num_elements));
```

カーネル生成時は実行させたい関数名のみを指定したが、引数を渡したい場合は `clSetKernelArg` 関数で指定する。0、1、2、3と指定している数値は指定したい引数の順

数学関数が返す値は丸めモード指定に影響されない。常に最近傍偶数丸めモードを指定した場合と同じ値を結果に得る。

`gentype` は、`float`、`float2`、`float4`、`float8`、`float16` のいずれかを示す。

## (1) 数学関数

### **`gentype acos (gentype)`**

逆余弦（アークコサイン）を求める。

### **`gentype acosh (gentype)`**

双曲線余弦（ハイパボリックコサイン）の逆関数（逆数ではない）を求める。

### **`gentype acospi (gentype x)`**

$\frac{\text{acos}(x)}{\pi}$  を求める。

### **`gentype asin (gentype)`**

逆正弦（アークサイン）を求める。

### **`gentype asinh (gentype)`**

双曲線正弦（ハイパボリックサイン）の逆関数を求める。

### **`gentype asinpi (gentype x)`**

$\frac{\text{asin}(x)}{\pi}$  を求める。

### **`gentype atan (gentype y_over_x)`**

逆正接（アークタンジェント）を求める。

### **`gentype atan2 (gentype y, gentype x)`**

$\text{atan}\left(\frac{y}{x}\right)$  を求める。

### **`gentype atanh (gentype)`**

双曲線正接（ハイパボリックタンジェント）の逆関数を求める。

### **`gentype atanpi (gentype x)`**

$\frac{\text{atan}(x)}{\pi}$  を求める。

### **`gentype atan2pi (gentype y, gentype x)`**

$\frac{\text{atan2}(y,x)}{\pi}$  を求める。

### **`gentype cbrt (gentype)`**

立方根を求める。

## ■ 著者プロフィール

### 池田 成樹 (いけだ・なるき)

1973年神奈川県生まれ。立教大学社会学部観光学科卒。

エンジニアとして企業に勤めるが、人生というものを意識しだして、やむにやまれず退職。ダイキチ・ドットネット有限会社を設立。社会人学生として、とある大学の夜間部で数学を学ぶ。だが、卒業は果たせず。コンピューティングの「極み」を追求する毎日。

### 主な著書

『Java はじめの一步』、『Delphi はじめの一步』、『Unix はじめの一步』(カットシステム 2003年)、『改訂版やさしい Java 入門』、『Delphi 2005 プログラミングテクニック Vol.7』(カットシステム 2005年)、『iPod で Linux したっていいじゃない』(カットシステム 2006年)、『Delphi 2005 プログラミングテクニック Vol.8』(カットシステム 2007年)、『Java GUI プログラミング Java SE 6 対応 Vol.I』(共著 カットシステム 2007年)、『Java GUI プログラミング Java SE 6 対応 Vol.II』(共著 カットシステム 2008年)

# OpenCL 並列プログラミング

## マルチコア CPU/GPU のための標準フレームワーク

2010年2月10日 初版第1刷発行

著者 池田 成樹  
発行人 石塚 勝敏  
発行 株式会社 カットシステム  
〒169-0073 東京都新宿区百人町 4-9-7 新宿ユーエストビル 8F  
TEL (03)5348-3850 FAX (03)5348-3851  
URL <http://www.cutt.co.jp/>  
振替 00130-6-17174  
印刷 株式会社 シナノ

---

本書に関するご意見、ご質問は小社出版部宛まで文書か、[sales@cutt.co.jp](mailto:sales@cutt.co.jp)宛に e-mail でお送りください。電話によるお問い合わせはご遠慮ください。また、本書の内容を超えるご質問にはお答えできませんので、あらかじめご了承ください。

---

- 本書の内容の一部あるいは全部を無断で複製複製(コピー・電子入力)することは、法律で認められた場合を除き、著作者および出版者の権利の侵害になりますので、その場合はあらかじめ小社あてに許諾をお求めください。

Cover design Y.Yamaguchi © 2010 池田成樹

Printed in Japan ISBN978-4-87783-237-7