JAEA-Testing
2009-001

JAEA-Testing

科学技術計算のための GPU 利用方法調査

Study on Availability of GPU for Scientific and Engineering Calculations

坂本 健作　小林 清二

Kensaku SAKAMOTO and Seiji KOBAYASHI

システム計算科学センター

Center for Computational Science & e-Systems

July 2009

Japan Atomic Energy Agency　日本原子力研究開発機構

本レポートは独立行政法人日本原子力研究開発機構が不定期に発行する成果報告書です。
本レポートの入手並びに著作権利用に関するお問い合わせは、下記あてにお問い合わせ下さい。
なお、本レポートの全文は日本原子力研究開発機構ホームページ（http://www.jaea.go.jp）
より発信されています。

独立行政法人日本原子力研究開発機構　研究技術情報部　研究技術情報課
〒319-1195　茨城県那珂郡東海村白方白根 2 番地 4
電話 029-282-6387, Fax 029-282-5920, E-mail:ird-support@jaea.go.jp

This report is issued irregularly by Japan Atomic Energy Agency
Inquiries about availability and/or copyright of this report should be addressed to
Intellectual Resources Section, Intellectual Resources Department,
Japan Atomic Energy Agency
2-4 Shirakata Shirane, Tokai-mura, Naka-gun, Ibaraki-ken 319-1195 Japan
Tel +81-29-282-6387, Fax +81-29-282-5920, E-mail:ird-support@jaea.go.jp

JAEA-Testing 2009-001

科学技術計算のためのＧＰＵ利用方法調査

日本原子力研究開発機構　システム計算科学センター
坂本　健作・小林　清二*

（2009 年 4 月 30 日受理）

近年，GPU(Graphic Processing Unit)を科学技術計算に用いる事例が増えている．GPU は CPU(Central Processing Unit)と比べて，高い浮動小数点演算性能とメモリバンド幅を有していると言われている．そこで，開発環境 CUDA(Compute Unified Device Architecture)を用いて，GPU による科学技術計算の有用性を調査した．調査の結果，原子力研究でも用いられる行列，FFT(Fast Fourier Transform)，流体力学等の計算で利用価値が高いことがわかった．一方，高性能を引き出すためには高度なプログラミング技術が必要であり，また，科学技術計算に必要な倍精度演算性能やメモリ誤りの検出等の機能が不十分であることがわかった．

原子力科学研究所(駐在):〒319-1195 茨城県那珂郡東海村白方白根 2-4
*（財）高度情報科学技術研究機構

Study on Availability of GPU for Scientific and Engineering Calculations

Kensaku SAKAMOTO and Seiji KOBAYASHI*

Center for Computational Science & e-Systems
Japan Atomic Energy Agency
Tokai-mura, Naka-gun, Ibaraki-ken

(Received April 30,2009)

Recently, the number of scientific and engineering calculations on GPUs (Graphic Processing Units) is increasing. It is said that GPUs have much higher peak floating-point processing power and memory bandwidth than CPUs (Central Processing Units). We have studied the effectiveness of GPUs by applying them to fundamental scientific and engineering calculations with CUDA (Compute Unified Device Architecture) development tools. The results have shown as follows: 1)Computations on GPUs are effective for such calculations as matrix operation, FFT (Fast Fourier Transform) and CFD (Computational Fluid Dynamics) in nuclear research region. 2)Highly-advanced programming is required for bringing out high performance of GPUs. 3)Double-precision performance is low and ECC (Error Correction Code) in graphic memory systems supports are lacking.

Keywords : GPGPU, CUDA.

*RIST(Research Organization for Information Science and Technology)

目　　次

Contents

This is a blank page.

1. はじめに

近年，GPU(Graphics Processing Unit)を汎用演算に用いる事例が増えており，生命科学，金融工学，流体解析，宇宙科学，データベースなど多方面に利用されるようになった．

計算科学は「理論」及び「実験」と並ぶ第三の科学であり，実験では不可能な問題を扱う原子力研究においては特に重要な手法である．原子力研究における計算分野としては核計算，熱流動，電磁気，構造，材料等，幅広い分野にわたっており[1)]，日本原子力研究開発機構（以下，原子力機構）における原子力研究においてもGPUを利用できる分野は多いと思われる．

NVIDIA社が提供するCUDA(Compute Unified Device Architecture)[2)]は一般的なプログラミング言語であるC言語をベースにした統合開発環境であり，同社のGPUをHPC分野に容易に利用できるようになった．これによりGPUコンピューティングの敷居が低くなり，一般の利用が加速されるようになった．

本調査では，NVIDIA社が提供するGPUと開発環境CUDAを用いて，GPUの性能調査及び科学技術計算のためのGPU利用方法調査を行い，原子力研究分野におけるGPU利用の有用性と課題を報告する．

性能調査及び利用方法調査に用いたハードウエア及びソフトウエアを表１－１に示す．

表１－１　性能試験調査に用いたハードウエア及びソフトウエア

ハードウエア	
CPU	Intel Core2 Duo E6750 2.66GHz
メモリ	2GB 800MHz DDR2-SDRAM
GPU	NVIDIA GeForce 8600GT
インターフェイス	PCI-Express
ソフトウエア	
OS	Windows Vista Business
開発環境	Visual Studio 2005
コンパイラ	Intel Visual Fortran & C ver.10.1
数値演算ライブラリ	Intel MKL ver.10.0
GPU コンパイラ	CUDA ver.2.0

2. アーキテクチャ

GPUの性能を十分に引き出すためには，GPUのアーキテクチャを理解する必要がある．ここでは，調査に用いたGeForce 8600GTのアーキテクチャについて述べる．

GPUは固定的なパイプライン(Pipeline)方式から，より自由で汎用的な用途に利用できる演算器を多数搭載するユニファイドシェーダー(Unified Shader)方式を採用することによって，高度な画像処理への要求を満たしてきた．これにより，自由なプログラミングが可能になりグラフィック以外の汎用計算用途に使われるようになった．

調査に用いたGeForce 8600GTの基本アーキテクチャであるG80デバイスのアーキテクチャを図２－１及び図２－２に示す．図２－１に示すとおり，G80デバイスは多数のテクスチャプロセッサクラスタ(Texture Processor Cluster)からなっており，高速なグローバルメモリ(Global Memory)との通信が可能となっている．また，図２－２に示すとおり，各テクスチャプロセッサクラスタは2つのマルチプロセッサ(Streaming Multiprocessor)からなり，各マルチプロセッサはそれぞれ8つのストリーミングプロセッサ(Streaming Processor)と呼ばれる演算器からなる．このストリーミングプロセッサは32ビットの浮動小数点演算(MUL)と浮動小数点積和演算(MAD)を同時に行い，全体で 膨大な数のスレッド実行が可能となっている．そのほか，共有メモリ(Shared Memory)，L1キャッシュ(L1 Cache)，スペシャルファンクションユニット(Special Function Unit)，テクスチャフィルタ(Texture Filter)及び命令ユニット(Instruction Unit)等からなる．

メモリアーキテクチャを図２－３に示す．8つのストリーミングプロセッサは，それぞれ32bitのレジスタを有する．また，8つのストリーミングプロセッサが共有にアクセス可能な共有メモリを有する．この共有メモリは，グローバルメモリよりもさらに高速なメモリであり，16KB／16バンクからなっている．共有メモリはレジスタと異なりプログラムで領域を定義してアクセスできるメモリであり，この高速なメモリバンド幅を有効に利用することで高速な計算が可能となる．各マルチプロセッサは，それぞれコンスタントキャッシュ(Constant Cache)とテクスチャキャッシュ(Texture Cache)を有する．各マルチプロセッサは32bitのレジスタセットを有する．この２つのキャッシュへ書き込む機能はあるが，通常は読み取り専用となっている．各マルチプロセッサはデバイスメモリ(Device Memory)によりデータを共有する．

CPUとGPUのハードウエア構成の違いを図２－４に示す．CPUとGPUの演算器，コントロールユニット及びキャッシュの数の圧倒的な違いが，GPUにおける高速な演算を可能にしていることがわかる．

調査に用いたGeForce 8600GTの仕様とパフォーマンスを表２－１に示す. ストリーミングプロセッサの数が32となっており，このことからマルチプロセッサの数は32÷8＝４ということになる. またメモリバンド幅を見ると, 22.4GB/sと非常に高速であることがわかる．

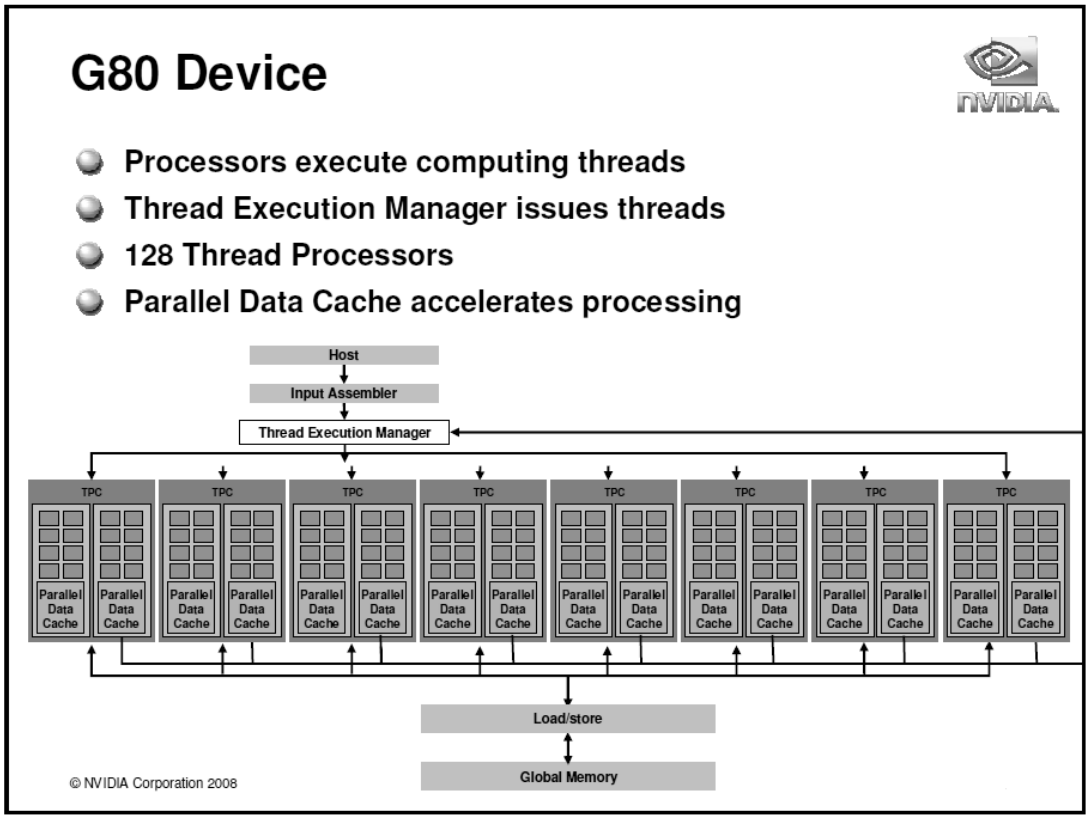


図２－１　G80 アーキテクチャ [3)]

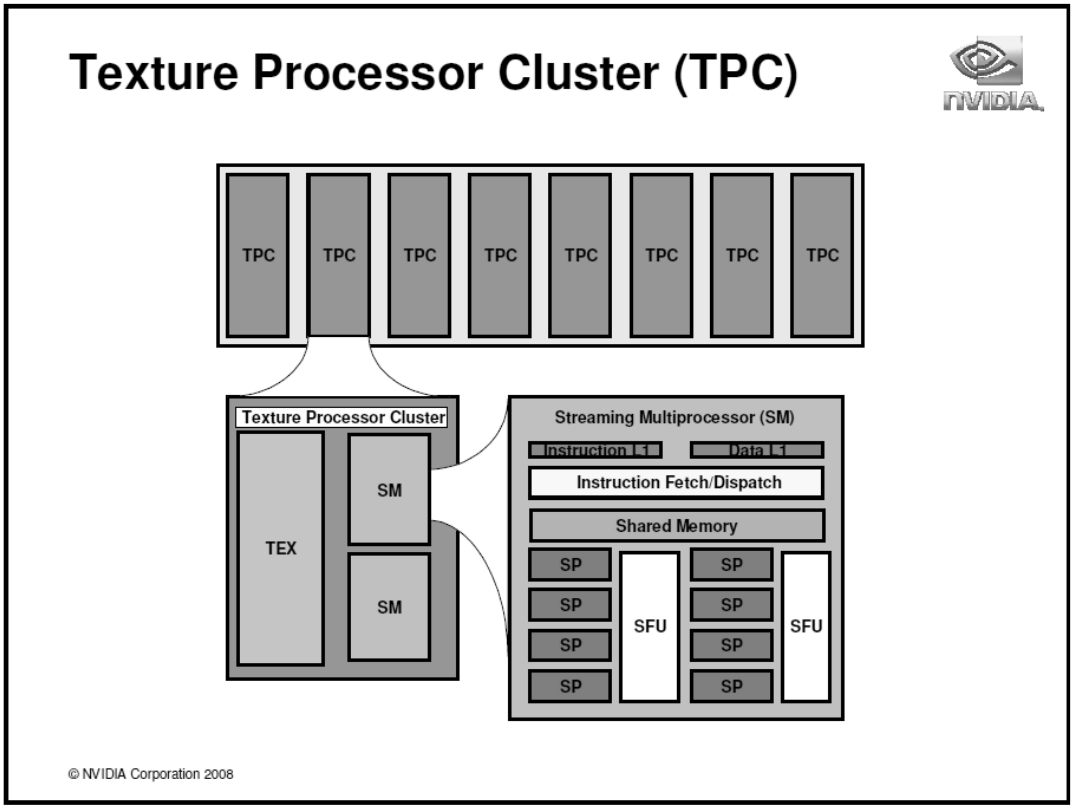


図２－２　G80 アーキテクチャ(詳細) [3)]

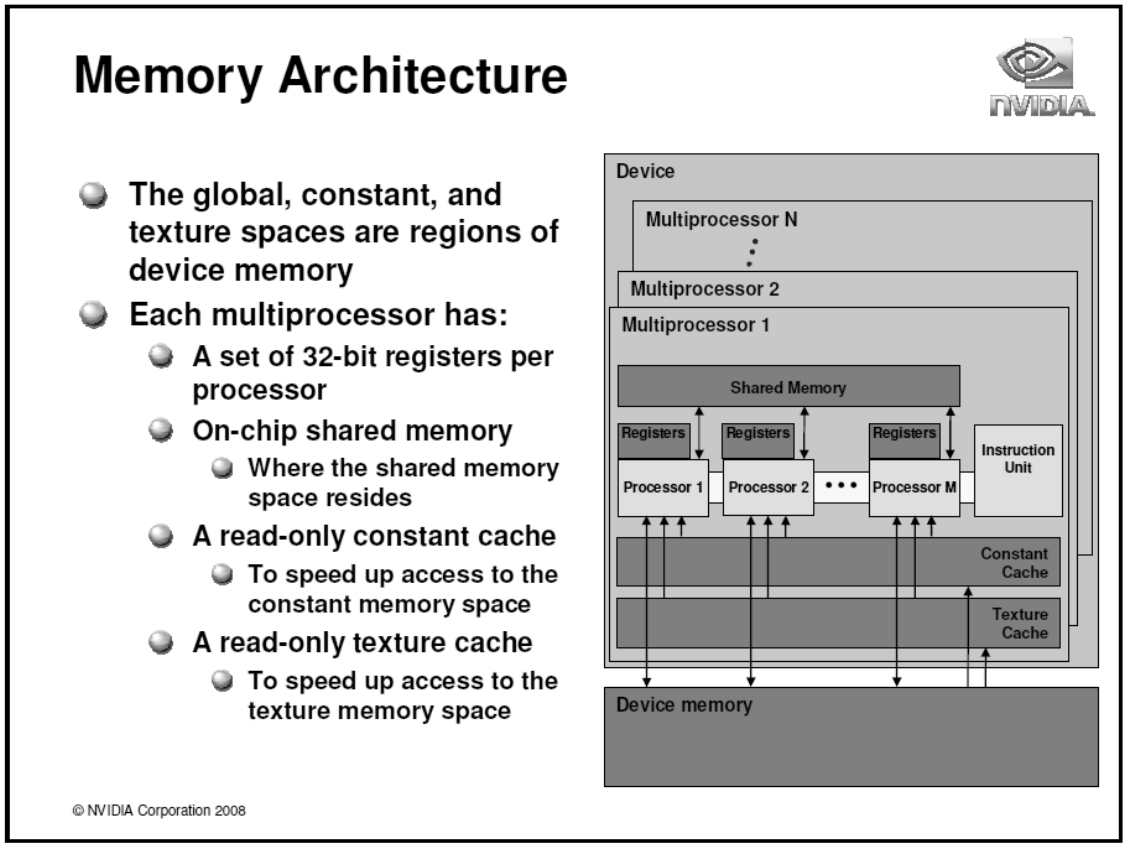


図２－３　メモリアーキテクチャ [3)]

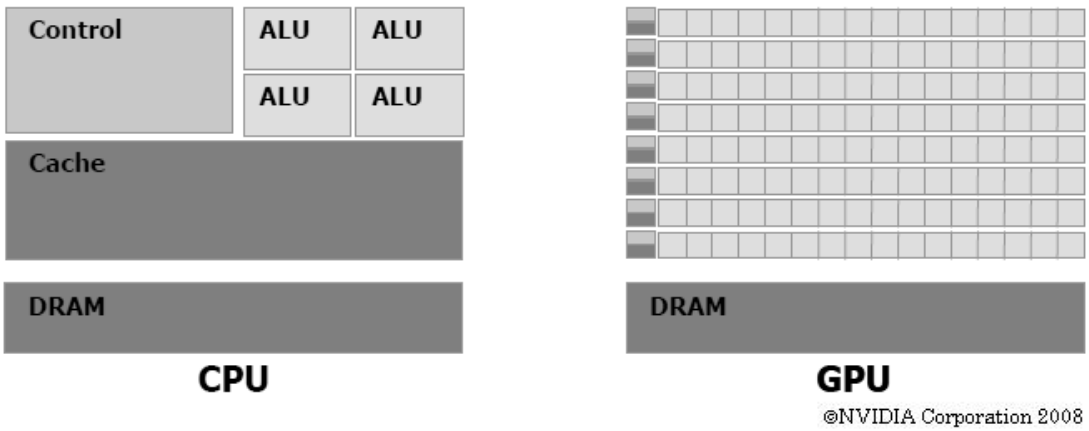


図２－４　CPU と GPU の構造比較 [4)]

表２－１　GeForce 8600GT の仕様とパフォーマンス [5)]

ストリームプロセッサ	32
コアクロック（MHz)	540
シェーダークロック（MHz)	1190
メモリクロック（MHz)	700
メモリサイズ(MB)	256
メモリインターフェイス	128-bit
メモリバンド幅(GB/秒)	22.4
テクスチャ充填率(10 億/秒)	8.64

3. ＣＵＤＡプログラミング

ここでは，CUDA のプログラミングモデルについて述べ，実際の行列積プログラムについてコンパイル・実行方法及び計算結果を示す.

3.1 CUDA のインストール方法

CUDA の開発環境は NVIDIA 社のサイトから無料でダウンロード可能である [6]．OS としては，Windows，Linux 及び MacOS をサポートしている．コンパイラとして，Windows では Microsoft　Visual Studio/C++が，Linux では gcc がインストールされている必要がある．CUDA ドライバ，CUDA ツールキット及び CUDA SDK コードサンプルの順にインストールする. これにより，後述する GPU 用に最適化された数値演算ライブラリ CUBLAS 及び CUFFT もインストールされる. また，CUDA Visual Profiler をインストールすると，たとえば CUDA プログラム実行時のメモリアクセスの一貫性（コヒーレント）を検証することができ，パフォーマンスの改善に役立つ．マニュアルや実際の適用事例の紹介も充実しており，ダウンロードや参照が可能である.

CUDA プログラミングモデルの基本的な構成を図３－１に示す．CUDA で作成されたアプリケーションは，インストールされた CUDA ドライバ，CUDA ランタイムライブラリ及び CUDA ライブラリを通して，GPU のメモリ領域の確保やスレッド実行を行うことが可能である．CUDA プログラミングは C 言語とほぼ同じインターフェイスであり，CUDA で作成されたオブジェクトは C コンパイラからリンクして CPU で動作するロードモジュールを作成できる.

3.2 CUDA プログラミングモデル

CPU側をホスト(Host)，GPU側をデバイス(Device)と呼ぶ．図３－２に示すとおり，ホストからデバイス上でカーネル(Kernel)が起動される，カーネルの中で膨大な数のスレッド(Thread)並列実行が行われるが，一度に実行されるカーネルは１つとなっている．GPU スレッドはCPUスレッドと比べて非常に軽量でありオーバヘッドがほとんどないことが特徴である．スレッドのまとまりをブロック(Block)と呼び，さらにブロックのまとまりをグリッド(Grid)と呼ぶ．カーネルはグリッド内で起動される，１つのマルチプロセッサで実行されるブロックは１つであるが，複数のブロックが同時に存在できる.

図３－３にカーネルのメモリアクセスを示す．ブロック内のスレッドは，共有メモリを介したデータの共有や実行の同期によって互いに協調できる．しかし，別のブロック内のスレッドとは協調できない．カーネルの入出力データはグローバルメモリに置かれる．ホストはグローバルメモリを読み書きできるが共有メモリを読み書きできない．単一ブロック内のスレッドによって共有メモリが共有される．この共有メモリはレジスタ並みに高速であり，プログラミングによって効率的に利用するとプログラム実行パフォーマンスが向上する.

スレッドは 32 スレッド単位で実行される．この単位をワープ(Warp)と呼ぶ．マルチプロセッサに含まれる８つのストリーミングプロセッサは４サイクルに渡って同じ命令を実行する．コアクロックがシェーダークロックに比べて非常に小さいことが理由であると思

われる．したがって，32 スレッド＝1Warp単位のスレッド並列実行によりプログラム実行パフォーマンスが向上することとなる．

3.3 プログラム実行の流れ

CUDA SDKコードサンプルには，実際のプログラムに応用可能なさまざまなサンプルが用意されている．ここでは，MatrixMulという行列積（Ｃ＝A×B)を計算するプログラムを参考に，実行の流れを解説する．

基本的に，プログラムはCPU側で実行され，GPU側のカーネルを呼び出す形でスレッド並列実行が行われる．図３－４にスレッド並列実行の流れを示す．

CUDAプログラミングにおけるプログラムの基本的な流れは以下(1)～(5)の通りである．各項にはプログラムで用いられる代表的な命令を記した．

図３－５にソースコード(MatrixMul_kernel.cu)を示す．

なお，実行に用いたメインプログラムMatrixMul.cuはGPUを使わずCPUでGPUをエミュレーションするコードを生成したり，デバッグ機能を組み込んだコードを生成したりできる．このため，CUDAの命令の前にCUDA_SAFE_CALLなどの関数をかぶせて記述されているので注意が必要である．

(1)GPUのデバイスメモリにデータ領域を確保

```
cudaMalloc((void**) &d_A, mem_size_A)
```

(2)データをGPU側に転送

```
cudaMemcpy(d_A, h_A, mem_size_A,cudaMemcpyHostToDevice)
```

(3)グリッド数とブロックスレッド数の確保

```
dim3 threads(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid(WC / threads.x, HC / threads.y);
matrixMul<<< grid, threads >>>(d_C, d_A, d_B, WA, WB);
```

(4)カーネル関数の呼び出し

```
matrixMul<<< grid, threads >>>(d_C, d_A, d_B, WA, WB)
```

(5)ホストへデータを転送，領域解放

```
cudaMemcpy(h_C, d_C, mem_size_C,cudaMemcpyDeviceToHost)
cudaFree(d_A)
```

CUDAのスレッド並列プログラミングの特徴は(3)のように，ループを分割する形で行

うのではなく，<<< ... >>>で囲まれた１行の命令を記述するところである．

3.4 プログラムの実行

GPUの演算性能は非常に高く，実行時間を計測するために行列のサイズを十分大きくする必要があり，2,048×2,048の正方行列の積とした．Ｃの１要素に対して2,048回の積和演算を行い，これを2,048×2,048個分計算するので，17,179,869,184回の浮動小数点演算(MUL+ADD）となる．行列のサイズは，図３－６に示すプログラムインクルードファイルMatlixMul.hを変更することにより可能である．

前述のように，テストに用いたMatrixMul.cuは，GPUを使わずCPUでGPUをエミュレーションするコードを生成したり,デバッグ機能を組み込んだコードを生成したりできる．したがって, プログラム開発環境Visual Studioにおいて, Release, EmuRelease, Debug, EmuDebugの４つのモードを選び，実行バイナリの生成と実行ができる仕組みとなっており，エミュレーション実行やデバッグに非常に便利である．

テスト環境におけるプログラム実行は，Visual StudioにおいてReleaseモード及びEmuReleaseモードで作成したバイナリを，Windowsのコマンドプロンプトを起動して実行した．ReleaseモードではGPU上でカーネルが実行され，EmuReleaseモードでは，CPU上でエミュレーション実行が行われる．

実行時間(ms)はプログラムから出力される．GPU(Releaseモード)及びCPU(EmuReleaseモード)における結果は，それぞれ次の通りであった．

GPU: 967.026245(ms)
CPU: 3277409.00(ms)

GPUでは約１秒，CPUでは約 55 分であり，GPUの実行速度は非常に大きいことがわかる．ただし，CPUの実行におけるプログラムをみると，CPUの性能を最大限に引き出しているプログラムではなく，さらに実行時間が長いため実行中に他のタスクが割り込みをおこなっていため不利な状況である．

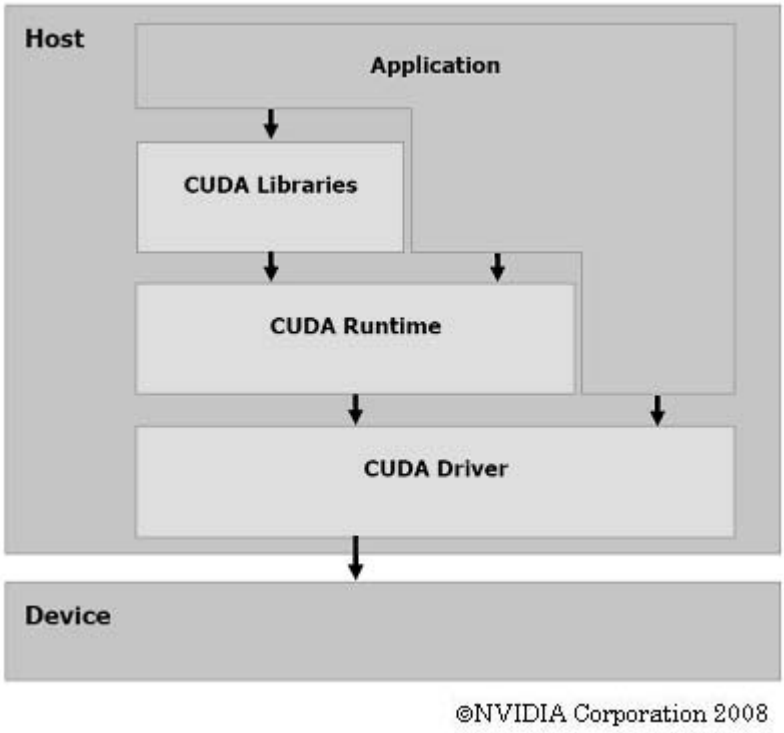


図３－１　CUDAプログラミングモデルの基本的な構成 [4)]

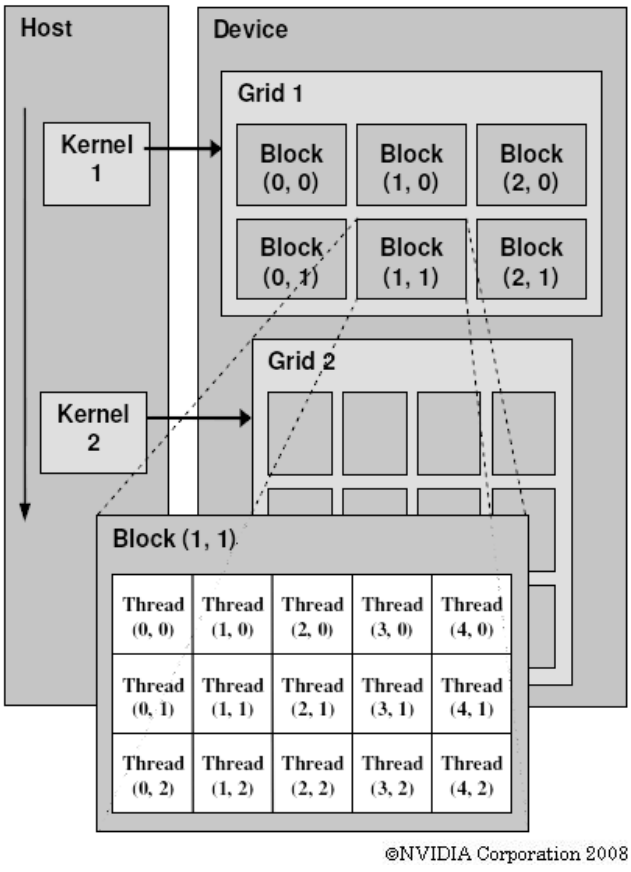


図３－２　CUDAプログラミングモデル [5)]

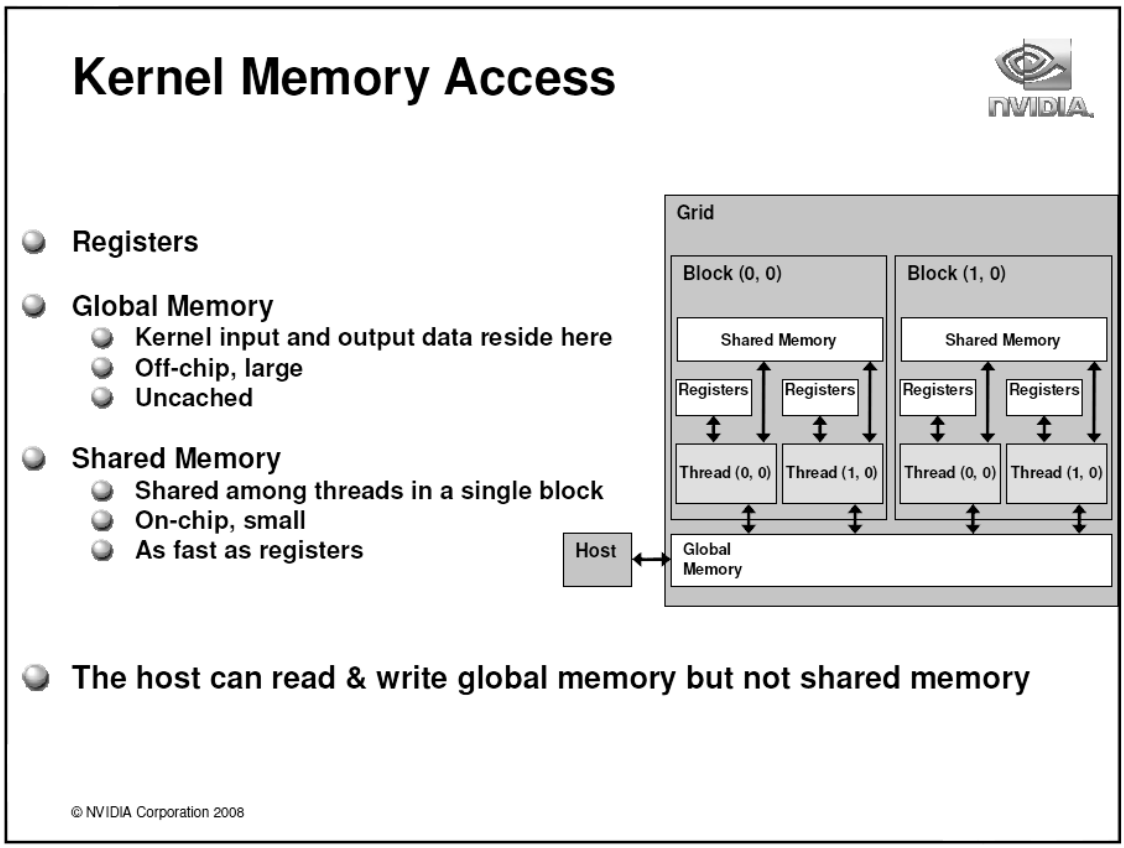


図３－３　カーネルのメモリアクセス[5)]

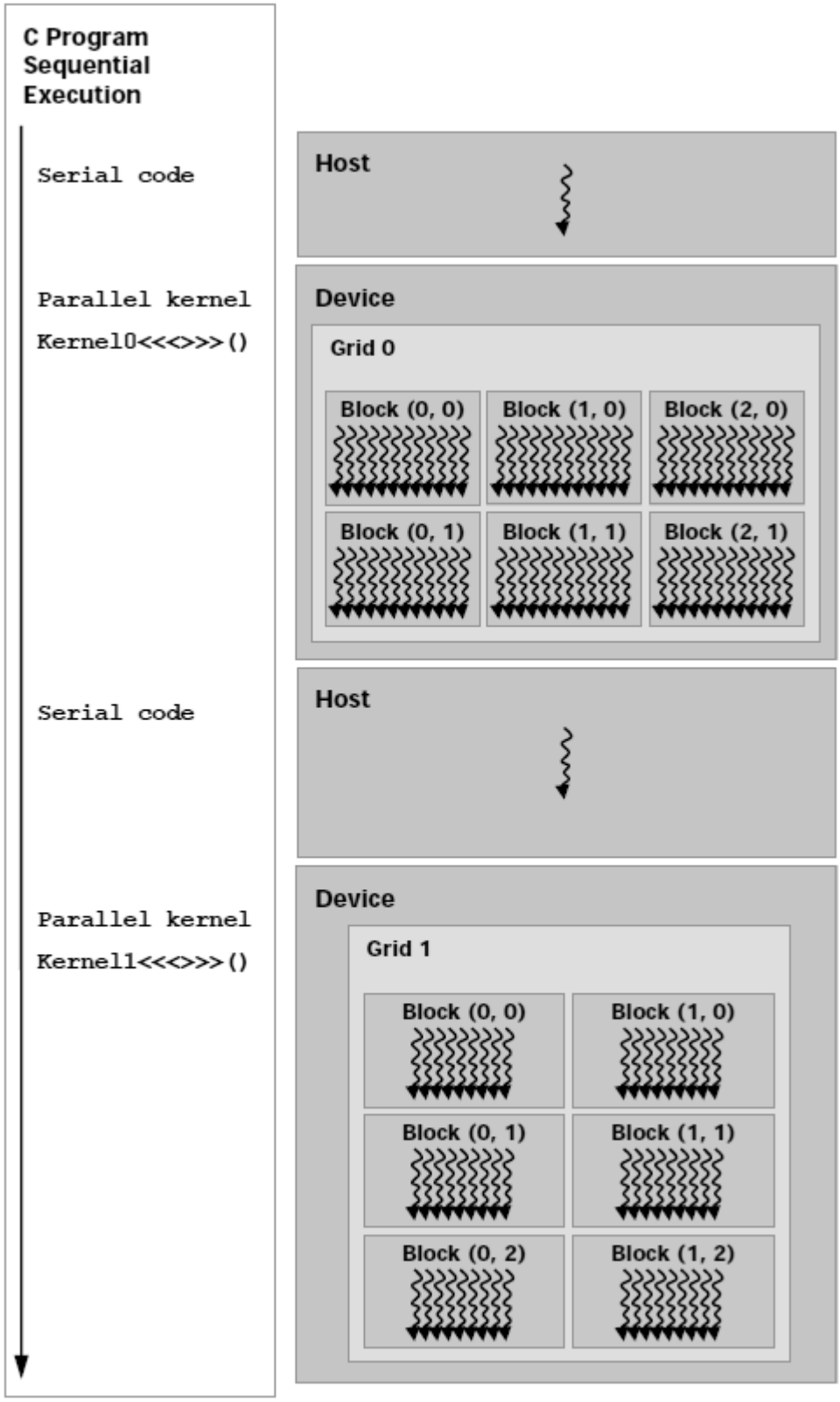


図３－４　スレッド並列実行の流れ [4)]

```
 1  * Copyright 1993-2007 NVIDIA Corporation.  All rights reserved.
 2  *
 3  * NOTICE TO USER:
 4  *
 5  * This source code is subject to NVIDIA ownership rights under U.S. and
 6  * international Copyright laws.  Users and possessors of this source code
 7  * are hereby granted a nonexclusive, royalty-free license to use this code
 8  * in individual and commercial software.
 9  *
10  * NVIDIA MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THIS SOURCE
11  * CODE FOR ANY PURPOSE.  IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR
12  * IMPLIED WARRANTY OF ANY KIND.  NVIDIA DISCLAIMS ALL WARRANTIES WITH
13  * REGARD TO THIS SOURCE CODE, INCLUDING ALL IMPLIED WARRANTIES OF
14  * MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
15  * IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL,
16  * OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
17  * OF USE, DATA OR PROFITS,  WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE
18  * OR OTHER TORTIOUS ACTION,  ARISING OUT OF OR IN CONNECTION WITH THE USE
19  * OR PERFORMANCE OF THIS SOURCE CODE.
20  *
21  * U.S. Government End Users.  This source code is a "commercial item" as
22  * that term is defined at  48 C.F.R. 2.101 (OCT 1995), consisting  of
23  * "commercial computer  software"  and "commercial computer software
24  * documentation" as such terms are  used in 48 C.F.R. 12.212 (SEPT 1995)
25  * and is provided to the U.S. Government only as a commercial end item.
26  * Consistent with 48 C.F.R.12.212 and 48 C.F.R. 227.7202-1 through
27  * 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire the
28  * source code with only those rights set forth herein.
29  *
30  * Any use of this source code in individual and commercial software must
31  * include, in the user documentation and internal comments to the code,
32  * the above Disclaimer and U.S. Government End Users Notice.
33  */
34
35 /* Matrix multiplication: C = A * B.
36  * Device code.
37  */
38
39 #ifndef _MATRIXMUL_KERNEL_H_
40 #define _MATRIXMUL_KERNEL_H_
41
42 #include <stdio.h>
43 #include "matrixMul.h"
44
45 #define CHECK_BANK_CONFLICTS 0
46 #if CHECK_BANK_CONFLICTS
47 #define AS(i, j) CUT_BANK_CHECKER(((float*)&As[0][0]), (BLOCK_SIZE * i + j))
48 #define BS(i, j) CUT_BANK_CHECKER(((float*)&Bs[0][0]), (BLOCK_SIZE * i + j))
```

図３－５　matrixMul_kernel.cu(1/3)[6]

```
49 #else
50 #define AS(i, j) As[i][j]
51 #define BS(i, j) Bs[i][j]
52 #endif
53
54 ////////////////////////////////////////////////////////////////////////////
55 //! Matrix multiplication on the device: C = A * B
56 //! wA is A's width and wB is B's width
57 ////////////////////////////////////////////////////////////////////////////
58 __global__ void
59 matrixMul( float* C, float* A, float* B, int wA, int wB)
60 {
61     // Block index
62     int bx = blockIdx.x;
63     int by = blockIdx.y;
64
65     // Thread index
66     int tx = threadIdx.x;
67     int ty = threadIdx.y;
68
69     // Index of the first sub-matrix of A processed by the block
70     int aBegin = wA * BLOCK_SIZE * by;
71
72     // Index of the last sub-matrix of A processed by the block
73     int aEnd   = aBegin + wA - 1;
74
75     // Step size used to iterate through the sub-matrices of A
76     int aStep  = BLOCK_SIZE;
77
78     // Index of the first sub-matrix of B processed by the block
79     int bBegin = BLOCK_SIZE * bx;
80
81     // Step size used to iterate through the sub-matrices of B
82     int bStep  = BLOCK_SIZE * wB;
83
84     // Csub is used to store the element of the block sub-matrix
85     // that is computed by the thread
86     float Csub = 0;
87
88     // Loop over all the sub-matrices of A and B
89     // required to compute the block sub-matrix
90     for (int a = aBegin, b = bBegin;
91              a <= aEnd;
92              a += aStep, b += bStep) {
93
94         // Declaration of the shared memory array As used to
95         // store the sub-matrix of A
96         __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
97
98         // Declaration of the shared memory array Bs used to
```

図３－５　matrixMul_kernel.cu(2/3)[6]

```
 99            // store the sub-matrix of B
100            __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
101
102            // Load the matrices from device memory
103            // to shared memory; each thread loads
104            // one element of each matrix
105            AS(ty, tx) = A[a + wA * ty + tx];
106            BS(ty, tx) = B[b + wB * ty + tx];
107
108            // Synchronize to make sure the matrices are loaded
109            __syncthreads();
110
111           // Multiply the two matrices together;
112            // each thread computes one element
113            // of the block sub-matrix
114            for (int k = 0; k < BLOCK_SIZE; ++k)
115                  Csub += AS(ty, k) * BS(k, tx);
116
117            // Synchronize to make sure that the preceding
118            // computation is done before loading two new
119            // sub-matrices of A and B in the next iteration
120            __syncthreads();
121      }
122
123      // Write the block sub-matrix to device memory;
124      // each thread writes one element
125      int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
126      C[c + wB * ty + tx] = Csub;
127 }
128
129 #endif // #ifndef _MATRIXMUL_KERNEL_H_
130
```

図３－５　matrixMul_kernel.cu(3/3)[6)]

```
   * Copyright 1993-2007 NVIDIA Corporation.    All rights reserved.
 1 #ifndef _MATRIXMUL_H_
 2 #define _MATRIXMUL_H_
 3
 4 // Thread block size
 5 #define BLOCK_SIZE 16
 6
 7 // Matrix dimensions
 8 // (chosen as multiples of the thread block size for simplicity)
 9 #define WA (3 * BLOCK_SIZE) // Matrix A width
10 #define HA (5 * BLOCK_SIZE) // Matrix A height
11 #define WB (8 * BLOCK_SIZE) // Matrix B width
12 #define HB WA    // Matrix B height
13 #define WC WB    // Matrix C width
14 #define HC HA    // Matrix C height
15
16 #endif // _MATRIXMUL_H_
```

図３－６　matrixMul.h[6)]

4. FORTRAN インターフェイスと CUDA ライブラリの利用

科学技術計算に用いられるプログラミング言語としては FORTRAN が用いられることが多い．CUDA の利用環境として，FORTRAN から簡単に利用できる数値演算ライブラリ CUBLAS 及び CUFFT が提供されている．ここではこれらを利用して前章と同様の規模の行列積計算について BLAS ライブラリ(sgemm)を用いて計算した．また，比較のために，CPU 側でも Intel MKL を用いて高速計算を行った．

4.1 FORTRAN インターフェイス

CUDA で作成されたオブジェクトコードは，直接 FORTRAN から利用することは可能である．しかし，C 言語と同様に受け渡し方法や配列におけるデータの格納方法に注意する必要がある．

4.2 CUDA ライブラリの利用

CUDA には広い用途に使用できるライブラリ CUBLAS と CUFFT がある．これらは FORTRAN からも呼び出し可能である．また．CUDA ドライバを直接操作することなしに利用可能である．それぞれ線形代数の基本サブルーチンである BLAS(Basic Linear Algebra Subroutine)と高速フーリエ変換を行う FFT(Fast Fourier Transform)のライブラリであり，科学技術計算での利用頻度が高い．

CUBLAS は，以下の単精度の BLAS 関数をサポートしている．

・実数データ
  ・レベル１（ベクトルーベクトル）
  ・レベル２（行列－ベクトル）
  ・レベル３（行列－行列）
・複素数データ
  ・レベル１
  ・CGEMM

CUFFT は，以下の機能をサポートしている．

・実数データと複素数データの１～３次元変換
・複数の１次元の変換を並列で実行するバッチ
・実数データと複素数データのインプレース及びアウトプレース変換

4.2.1 CUBLAS

CUBLAS の配列構造は FORTRAN に準じた構造となっている．すなわち，配列の初期値は１であり，列メジャーで配列要素を格納する．そのため，FORTRAN から扱いやすいライブラリとなっている．CUDA のサイトには CUBLAS を利用したサンプル [7]が提供されており，行列積 sgemm を利用するソースコードがある．FORTRAN から使用するためのラッパープログラム fortran.c が提供されている．このラッパープログラムは CBLAS を利用する場合に非常に便利である. 既存のコードを修正する場合, fortran.c を用いると，コンパイル時に CUBLAS_USE_THUNKING を定義すると修正が少なくて済む．これにより，関数の呼び出し中に自動的に CPU から GPU にデータをコピーし，結果を CPU に

返し，GPU のメモリを開放する．ただし，関数呼び出しごとにメモリが割り当てられるのでオーバヘッドがかかる．

ここでは，CUBLAS の行列積ルーチン sgemm を利用して，前章と同様に 2048×2048 の行列積を求めた．ソースコードを図４－１に示す．

コンパイル・リンク方法は以下の通りである．

```
icl /O3 /c fortran.c
ifort   /O3 /fpp /DCUBLAS sgemm_speed.f90 fortran.obj cublas.lib
```

実行結果はプログラムから実行時間(ms)と実効性能(MFlops)で出力され，結果は次の通りであった．

time = 0.4212   MFLOPS= 40787.6562

4.2.2. MKL との比較

CUBLAS ライブラリとの比較のために，同じ行列積を数値演算ライブラリ Intel MKL の sgemm を用いて CPU 上で実行した．

コンパイル・リンクの方法は次の通りである．

```
ifort   /fpp /O3 sgemm_speed.f90   mkl_c.lib libguide40.lib
```

実行結果はプログラムから実行時間(ms)と実効性能(MFlops)で出力され，結果は次の通りであった．

time = 1.3104   MFLOPS= 13110.3057

4.3. 考察

テスト環境で用いたGeForce8600GTは，表２－１に示すとおりである．G80デバイスのストリーミングプロセッサは，32ビットの浮動小数点演算(MUL)と浮動小数点積和演算(MAD)を同時に行う能力を持っているので，実効性能のピーク値は次の様に求められる．

３命令(MUL+MAD)×４MP×８SP×1180(MHz) ＝113.28(GFlops)

また，テスト環境で用いたCPU Intel E6750は，2.66GHzなので，実行性能のピーク値は次のように求められる．

８命令×2.66(GHz)＝21.3(GFlops)

ここで，これまでの2048×2048行列積の演算性能結果を表４－１にまとめた．MKLを利用した場合に比べて，CUBLASを利用した場合の実効性能は高いが，ピーク値は36%にとどまっている．ピーク値に近い性能を引き出すためには，MULとMADの演算を同時に行うようなプログラムが必要となるが，現実的には難しいと思われる．

図４－２に行列積のサイズ別実効性能を示す. 横軸のサイズは 1024×1024 を 1024 と表し，

1024～2560 のサイズについて，MKLとCUBLASの実行性能(GFlops)を計測した．
CUBLASの性能が高いがサイズによって安定していない．原因は未調査であるが，GPU の性能を十分引き出すには，GPUの構造を理解し，さらなるCUDAプログラムの改良が必要であると思われる．

```
 * Copyright 1993-2007 NVIDIA Corporation.  All rights reserved.
 1 program matrix_multiply
 2 implicit none
 3 ! Define the floating point kind to be  single_precision
 4 integer, parameter :: fp_kind = kind(0.0)
 5 ! Define
 6 real (fp_kind), dimension(:,:), allocatable ::        A, B, C
 7 real ::         time_start,time_end
 8 real (fp_kind)::        alpha=1._fp_kind,beta=1._fp_kind, c_right
 9 integer::  i,j,m1,m2
10 do m1=128,2560,32
11   allocate(A(m1,m1))
12   allocate(B(m1,m1))
13   allocate(C(m1,m1))
14 ! Initialize the matrices A,B and C
15   A=1._fp_kind
16   B=2._fp_kind
17   C=3._fp_kind
18 c_right= 2._fp_kind*m1+3._fp_kind
19 ! Compute the matrix product  computation
20   call cpu_time(time_start)
21 #ifdef CUBLAS
22   call cublas_SGEMM ('n','n',m1,m1,m1,alpha,A,m1,B,m1,beta,C,m1)
23 #else
24    call SGEMM ('n','n',m1,m1,m1,alpha,A,m1,B,m1,beta,C,m1)
25 #endif
26   call cpu_time(time_end)
27 ! Print timing information
28 print "(i5,1x,a,1x,f8.4,2x,a,f12.4)", m1, " time =",time_end-time_start, "
MFLOPS=",1.e-6*2._fp_kind*m1*m1*m1/(time_end-time_start)
```

図４－１　sgemm_speed.f90(1/2)[7)]

```
29 ! check the result
30          do j=1,m1
31            do i=1,m1
32              if ( abs(c(i,j)- c_right ) .gt. 1.d-8 ) then
33                      print *, "sgemm failed", i,j, abs(c(i,j)- c_right )
34                      exit
35              end if
36            end do
37          end do
38
39   deallocate(A,B,C)
40 end do
41 end program matrix_multiplyly
```

図４－１　sgemm_speed.f90(2/2)[7)]

表４－１　2048×2048 行列積の演算性能結果

プログラム名／ 演算器名／ライブラ名	実行時間 (ms)	実効性能 (GFlops)	ピーク性能比 (%)
sgemm_speed.f90/ GPU /CUBLAS	421	40.8	36.0
sgemm_speed.f90/ CPU /Intel MKL	1311	13.1	61.5
matrixMul.cu / GPU/－	967	17.8	15.7
matrixMul.cu / CPU/－	3277409	－	－

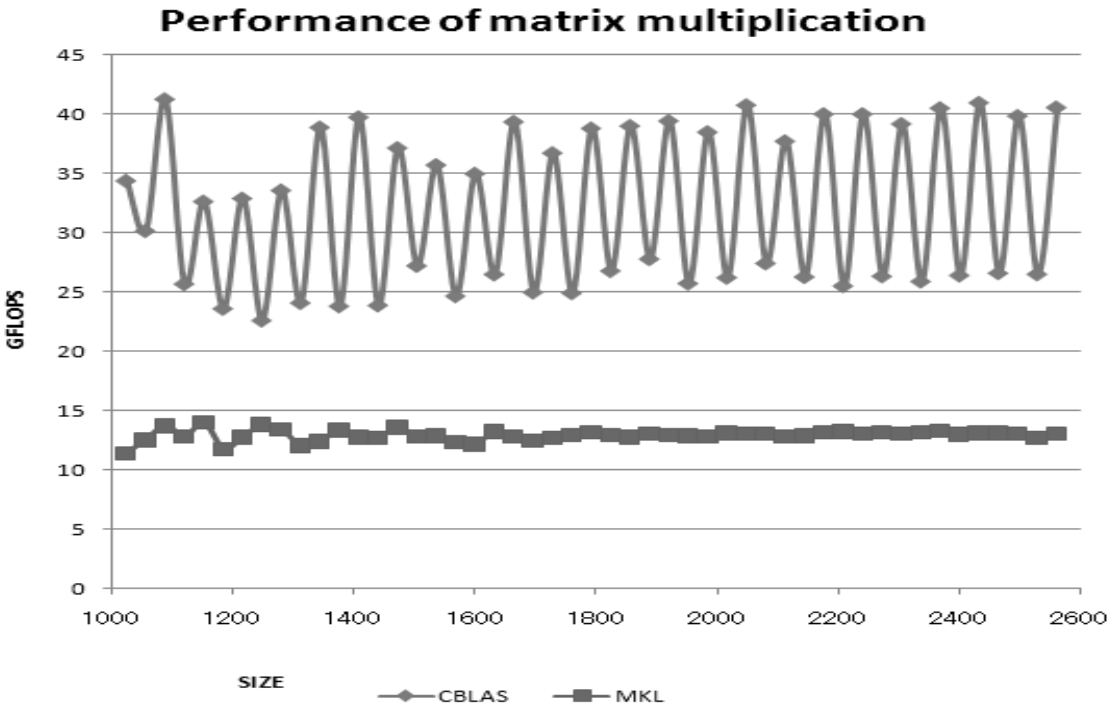


図４－２　行列積のサイズ別実効性能

5. バンド幅の測定

科学技術計算における GPU 利用の利点の１つに，高いメモリバンド幅があげられる．一般に，流体解析プログラムのパフォーマンスはメモリバンド幅によるところが大きいとされており，GPU 利用の利点が大きい．

CUDA SDK コードサンプルに BandWithTest というホスト－デバイス間及びデバイス－デバイス間のバンド幅を測定するプログラムが用意されている．これを使って 100KB～100MB までのデータ転送を行った．命令は以下の通り cudaMemcpy を用いた．

```
cudaMemcpy( d_idata, h_idata, memSize,cudaMemcpyHostToDevice)
```

測定結果を図５－１に示す．グラフで分かるようにテスト環境では 10MB 程度以降からバンド幅の上昇が鈍っている．

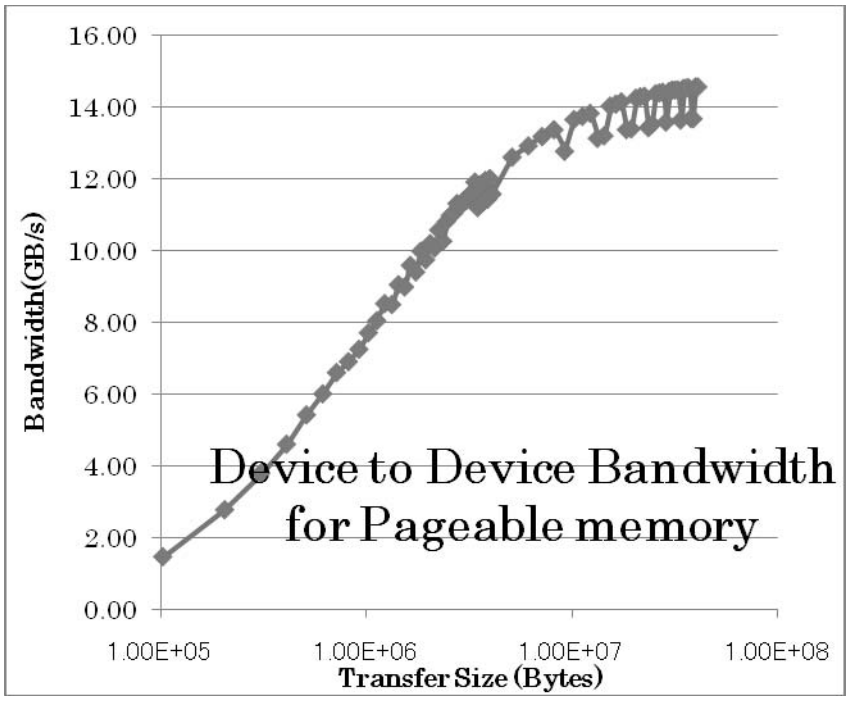
表５－１にメモリバンド幅の比較を示す．テスト環境に用いた GeForce 8600GT では 22.4GB/s のバンド幅であり，最新の GeForce GTX280 では 141.7GB/S となっている．一方スーパーコンピュータ SGI Altix3700Bx2 システムのメモリバンド幅は NUMAlink で 6.4GB/s である．

さらに共有メモリとストリーミングプロセッサのバンド幅はレジスタ並の速度と言われ正確な値は不明であるが，グローバルメモリよりも数百倍高速とされている [3]．しかしこの共有メモリは 16KB 程度の大きさであり，多数のスレッドによるメモリアクセスのために 16 のバンクに分割されている．これにより広いバンド幅を実現している．しかし，バンクへの複数の同時アクセスではアクセスがシリアル化されるためにバンクの競合が発生する．図５－２及び図５－３にバンクアドレス指定の例を示す．バンクの競合のチェックには warp_serialize 信号を使用し，できるだけバンク競合をなくす作業が必要となる．

成瀬らは，安定して高いメモリバンド幅を実現するために，試行錯誤を繰り返し，プログラミングを工夫することで，実効値をピーク値の 88%まであげることに成功した．スレッド進行の同期化，アクセスパターンの局所化，スレッド数の最適化が重要であるとまとめている [8]. このことによって GPU のメモリバンド幅の実効値の高さも明らかになった．

CPU と GPU のデータ転送は PCI-E インターフェイスを介して行われる．pinned memory calling と呼ばれる高速化手法も存在する [7]が十分ではなく，そのため，GPU の高いバンド幅を最大限に利用するためには，できるだけ CPU-GPU 間の転送を少なくする工夫が必要となる．

GPU の並列化や PC クラスタによる GPU の分散利用も可能であるが，データ転送がボトルネックになることを考慮して設計する必要がある．



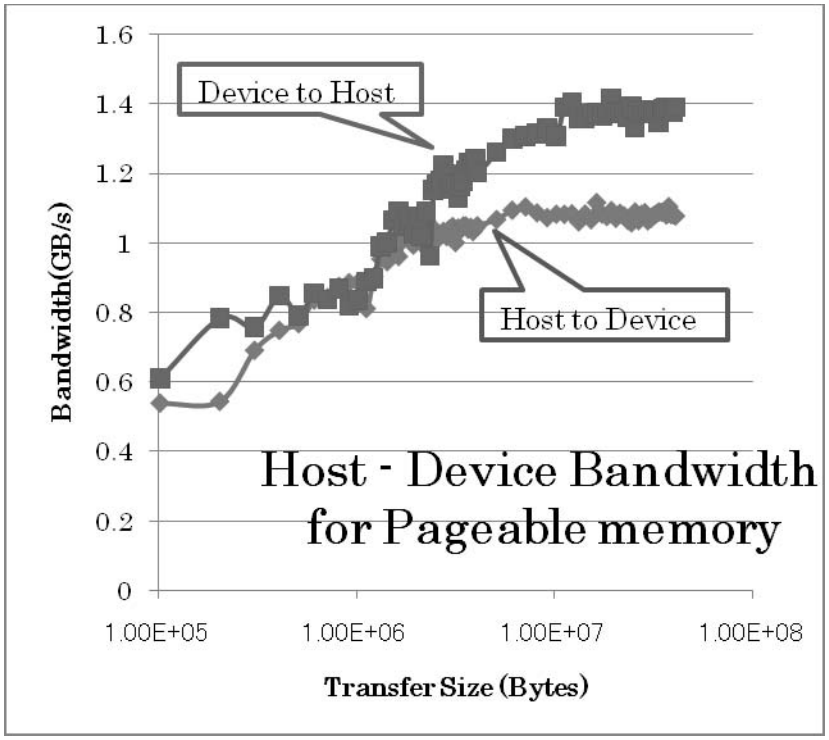


図５－１　テスト環境におけるメモリバンド幅の測定

表５－１　メモリバンド幅の比較

NVIDIA　GeForce 8600GT	22.4GB/s
NVIDIA　GeForce GTX280	141.7GB/s
Altix3700Bx2 システム(NUMAlink)	6.4GB/s
DDR2-800 SDRAM	6.4GB/s
DDR3-2133 SDRAM	17.6GB/s

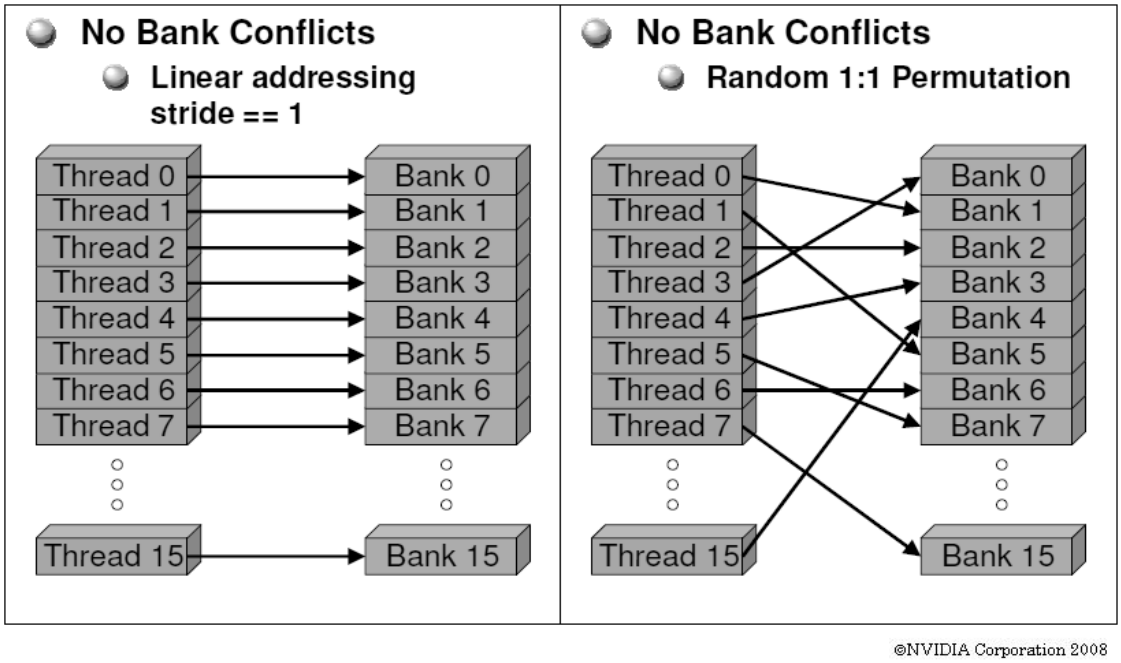


図５－２　バンクアドレス指定の例（競合なし）[3)]

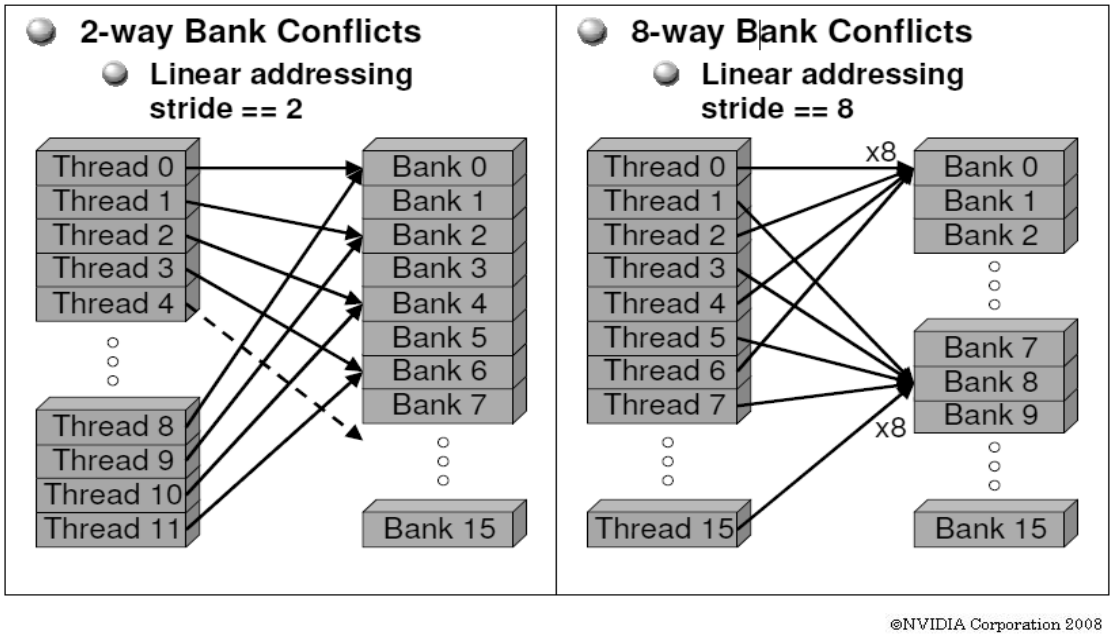


図５－３　バンクアドレス指定の例（競合あり）[3)]

6. リアルタイムシミュレーション

科学技術計算分野においてシミュレーション結果を理解するために可視化は不可欠な技術であり，原子力機構でも可視化技術開発を進めてきた．特に原子力分野では，見えないものを扱うため重要な技術である．また，リアルタイムシミュレーションは，緊急時の対応や原子力教育の用途として，要望は高いと思われる．

一方，GPU ではユニファイドシェーダー方式が採用され，ゲーム等では，よりリアルな描写を行うために，物理演算等を GPU 内で行うリアルタイムシミュレーションが可能となった．科学技術計算分野でも流体シミュレーションなど GPU を利用したリアルタイムシミュレーションの事例は多く報告されている [9]．

ここでは，CUDA SDK コードサンプル内にある Fast N-Body simulation をテスト環境にて実行し調査した．このコードは多体問題を扱うシミュレーションであり，粒子運動など適用範囲が広い．

ここで扱うプログラムは，宇宙空間上の N 個の物体の動きをシミュレーションするものである．１つの物体は，他の N-1 個の物体の引力の影響を受けて，その位置，質量及び速度が決定する．計算式を図６－１に示す．この計算について，すべての物体の質点に及ぶ力を正確に行うためには $O(N^2)$ という膨大な計算量がかかる．通常は，この計算量を減らすために，ある値以上離れた物体の計算を行わないカットオフを行うことで計算量を減らすような対応が行われるが，GPU の高い演算能力によりすべての物体における計算が可能になった．

図６－２に N-body simulation の実行時の画面イメージを示す．

テスト環境にてコンパイル，実行したところ，4,096 個の物体が非常に滑らかに可視化された．本プログラムでは，性能を数値化した瞬間値がウインドウ上部に表示されるが，表示性能は約 80fps，実効効率は約 27GFlops と表示された．一方，CPU 版(nbody_gold.cpp)をコンパイル実行したところ，画像が各フレームでストップするような表示がなされ，正常な計測ができなかった．GPU 内での演算は，実際には一定時間ステップごとにまとめて計算を行っているようだが，GPU によるリアルタイムシミュレーションの有用性は明らかである．

今後，原子力機構における可視化・リアルタイムシミュレーションの要望への適用が期待できる．

$$D_i \approx Gm_i \cdot \sum_{1 \le j \le N} \frac{m_j r_{ij}}{\left(\left\| (r_{ij}) \right\|^2 + \varepsilon^2 \right)^{3/2}}$$

図６－１　N-Body Simulation で用いられる計算式 [10)]

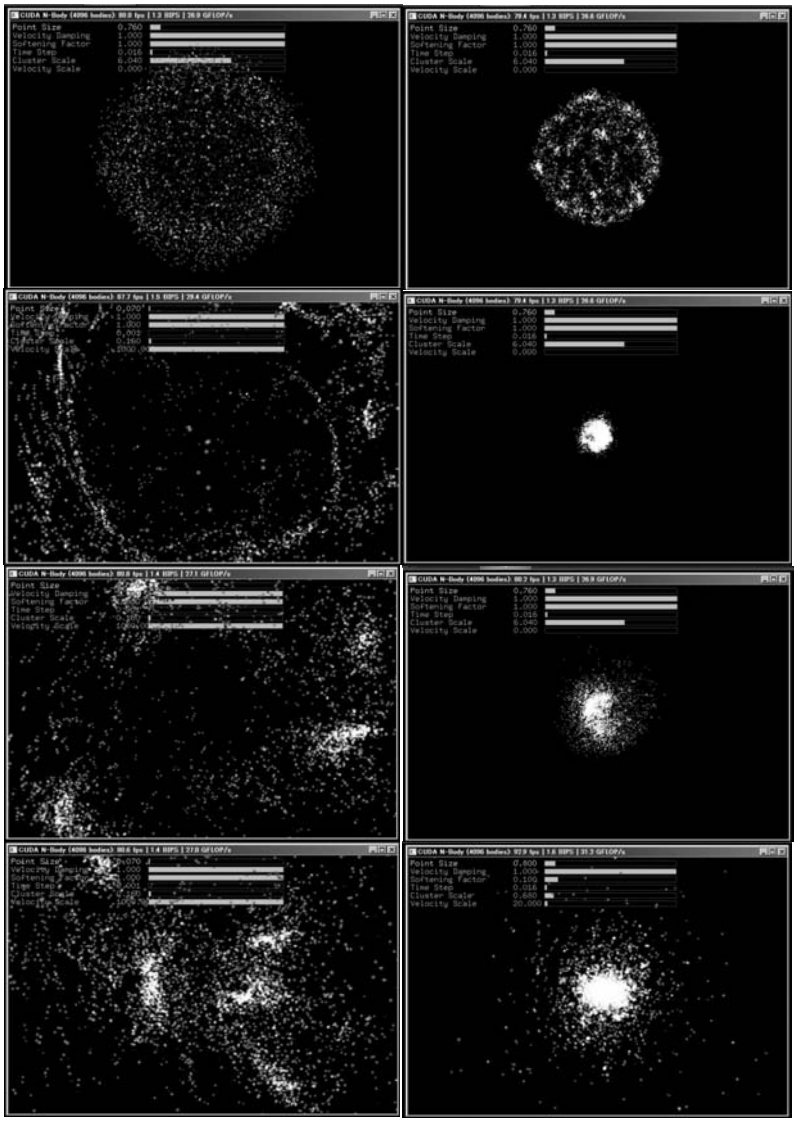


図６－２　N-body Simulation の実行イメージ

7. 今後の課題と展望

本来，GPU はグラフィックス処理を目的として開発されてきたものであり，科学技術計算に用いるにはいくつか問題点がある．

(1)倍精度浮動小数点演算をサポートしていない又は単精度に対し性能が低い．
(2)メモリのエラーコレクション(ECC)機能がなく信頼性に劣る．
(3)高性能を引き出すためのプログラミングの難易度が高い．
(4)IEEE754 に完全に準拠していない．
(5)GPU を並列に用いる場合通信性能がボトルネックになる．

(1)については，メーカにより科学技術計算に特化した製品が販売され始めた．たとえば，Tesla S1070 は単精度ピーク性能 4.14 TFlops，倍精度ピーク性能 345 GFlops とあり [11]，単精度に対し性能が 1/10 以下であるが，倍精度浮動小数点演算をサポートしている．(2)については，現状は不明であるが，丸山らによる GPU 向けソフトウエア ECC の研究等，取り組み [12]が進んでいる．(3)については，教育とサポート及び利用環境の充実が必要であると考える．また，PGI 社から CUDA をサポートした製品がリリースされる予定である [13]．(4)については表 7－1 に示すようにテストに使用した GeForce 8600GT は準拠していないが，NVIDIA 社のサイトによると Tesla 等の製品では，「準拠」となっている [11]．(5)については，すべての並列計算環境にいえることである．ただし，CUDA では fast PCI-e transfer に対応したプログラミングが可能となっている．

GPUをはじめとしたアクセラレータ技術は，科学技術計算分野において，ますます利用価値の高いものとなると思われる．図 7－1 及び図 7－2 に実効性能とメモリバンド幅の向上のグラフを示すとおり，CPUに比してGPUの性能向上は目覚しい．また，性能当たりの価格や消費電力は，CPUに比べて低い．一方，CPUメーカでは，「AMD Fusion」[14]構想など，今後CPUチップ内に複数のCPUと共にGPUなどのアクセラレータ回路も統合して，総合的な処理性能の向上を図る計画がある．

本調査で用いたCUDAのプログラミングモデルは実用性が高く，グリッド・ブロック構造による階層的スレッドの制御方法は，OpenCL[15]の策定に大きく影響を及ぼし，今後メニーコア時代を迎える計算機において，広く利用されていく可能性がある．

表７－１　IEEE754 対応状況(2008 年)[3)]

Floating Point Characteristics

	G8x	SSE	IBM Altivec	Cell SPE
Format	IEEE 754	IEEE 754	IEEE 754	IEEE 754
Rounding modes for FADD and FMUL	Round to nearest and round to zero	All 4 IEEE, round to nearest, zero, inf, -inf	Round to nearest only	Round to zero/truncate only
Denormal handling	Flush to zero	Supported, 1000's of cycles	Supported, 1000's of cycles	Flush to zero
NaN support	Yes	Yes	Yes	No
Overflow and Infinity support	Yes, only clamps to max norm	Yes	Yes	No, infinity
Flags	No	Yes	Yes	Some
Square root	Software only	Hardware	Software only	Software only
Division	Software only	Hardware	Software only	Software only
Reciprocal estimate accuracy	24 bit	12 bit	12 bit	12 bit
Reciprocal sqrt estimate accuracy	23 bit	12 bit	12 bit	12 bit
log2(x) and 2^x estimates accuracy	23 bit	No	12 bit	No

©NVIDIA Corporation 2008

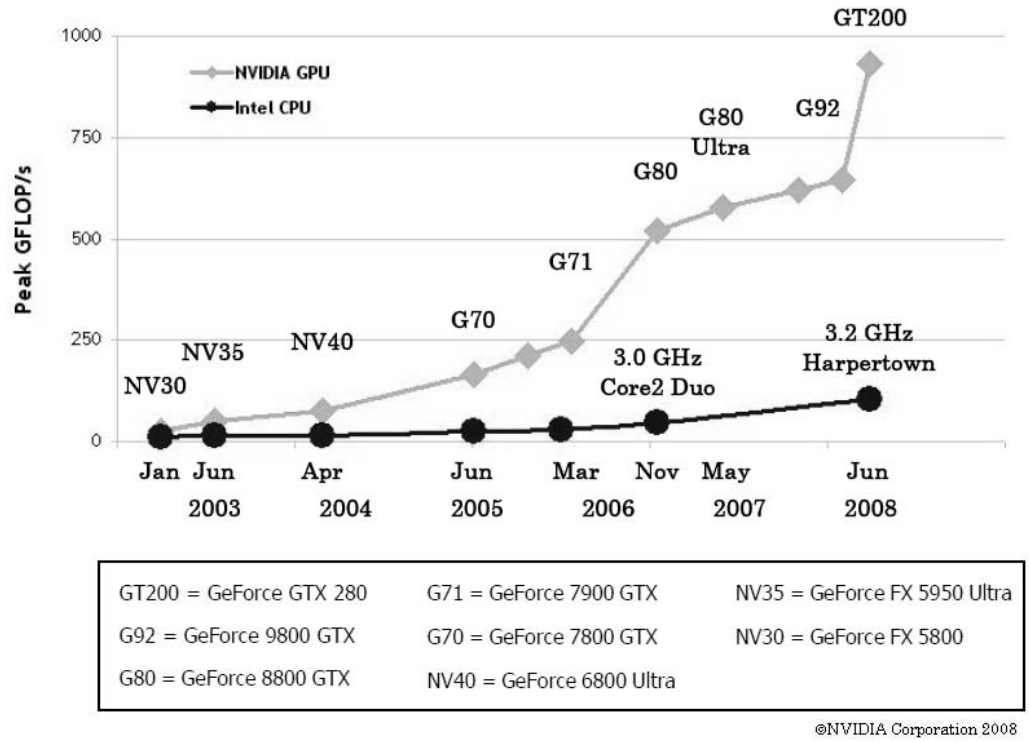


図７－１　GPU と CPU の実効性能の向上 [4)]

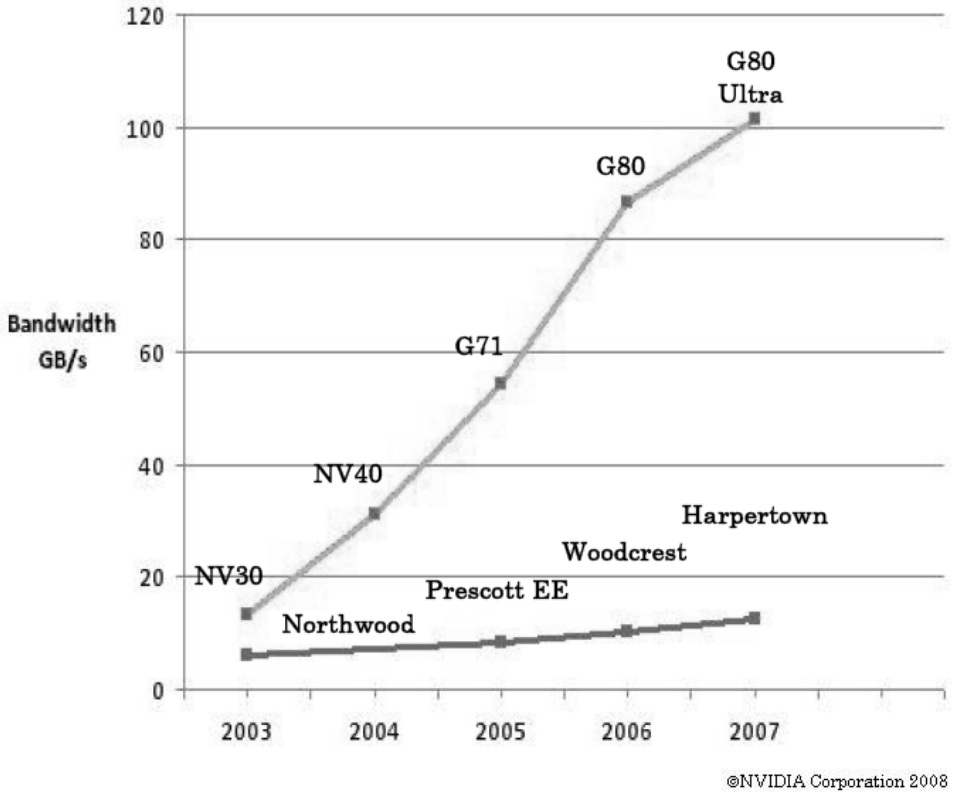


図７－２　GPU と CPU のメモリバンド幅の向上 [4)]

8. おわりに

本調査では，NVIDIA 社が提供する GPU と開発環境 CUDA を用いて，GPU の性能調査及び科学技術計算のための GPU 利用方法調査を行い，有用性と課題を示した．

本調査に用いたハードウエアはコンシューマ向けのローエンドのものであるが，ハイエンドの環境においても同様の開発環境を用いることができ，無理なく移行が可能である．また，CUDA は C 言語ベースの開発環境であるが，Fortran からの利用も可能であり，特に数値演算ライブラリの利用価値は高いことがわかった．

本調査により，GPU 利用の利点として，次があげられることがわかった．

(1)計算密度の大きい浮動小数点演算性能が高い．

(2)メモリバンド幅が高い．

(3)リアルタイム可視化シミュレーションが可能である．

(1)については，行列計算や多体問題・分子動力学等の分野への利用が，(2)については，FFT や流体力学等の分野への利用に対して有用であると思われる．また，原子力研究では可視化シミュレーションについても計算機利用割合が高く，(3)の利点は大きい．その他，利点としては，CPU に比して対性能比での価格の安さと消費電力の低さがあげられる．

一方，GPU 利用の課題も多く，主に次があげられる．

(1)倍精度浮動小数点演算をサポートしていない又は単精度に対し性能が低い．

(2)メモリのエラーコレクション(ECC)機能がなく信頼性に劣る．

(3)高性能を引き出すためのプログラミングの難易度が高い．

これらの課題に対しては，製品開発や研究が進んでいる．

原子力研究における計算分野としては核計算，熱流動，電磁気，構造，材料等，幅広い分野にわたっており，このうち GPU を利用できる分野は多いと思われる．図８－１に原子力機構の大型計算機 Altix3700Bx2 システムにおける CPU 時間使用実績を示す．これら上位に上げられるコードの多くは，行列計算，FFT，分子動力学，粒子法，流体力学等を用いるプログラムであり，GPU 利用が有効である可能性が高いと思われる．原子力分野における科学技術計算においても，GPU 利用が進みつつある [16]．

今後，科学技術計算用途に特化した NVIDIA 社製の Tesla[11]の調査及び原子力研究で利用の多いプログラムについて GPU の適用作業を進める予定である．

本報告書が，GPU を用いた科学技術計算に取り組もうする研究者の参考になれば幸いである．

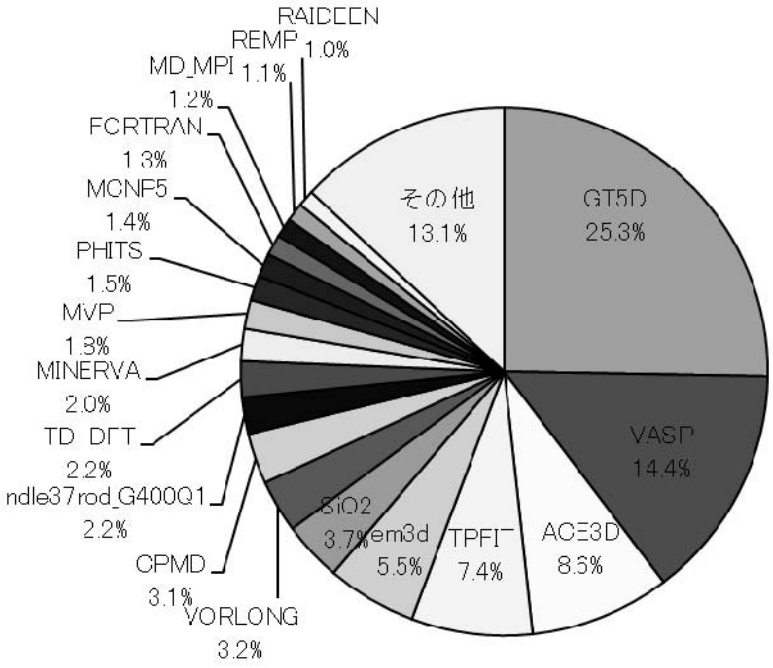


図８－１　平成 20 年度 Altix3700Bx2 システムコード別 CPU 時間使用実績

謝辞

本報告書を執筆する機会を与えてくださいましたシステム計算科学センター長　平山俊雄氏，情報システム管理室長　久米悦雄氏に深く感謝いたします．本報告書の内容及び作成にあたりご助言をいただきました情報システム利用推進室　松本潔氏，情報システム管理室　関暁之氏に深く感謝いたします．

参考文献

1)計算科学技術部会：”原子力における計算科学技術の未来”,日本原子力学会誌, 51,pp.306-309(2008)
2)http://www.nvidia.co.jp/cuda
3)NVIDIA : "CUDA Technical Training Volume I Introduction to CUDA Programming" (2008)
4)NVIDIA : "NVIDIA CUDA Compute Unified Device Architecture Programming Guide version 2.0" (2008)
5)http://www.nvidia.co.jp/object/geforce_8600_jp.html
6)http://www.nvidia.co.jp/object/cuda_get_jp.html
7)”FORTRAN using pinned memory calling CUBLAS”, http://developer.download.nvidia.com/compute/cuda/1_1/Fortran_Cuda_Blas.tgz
8)成瀬彰,住元真司,九門耕一 : "GPGPU上での流体アプリケーションの高速化手法", IPSJ SIG Technical Report,2008-HPC-117,pp.49-54 (2008)
9)http://www.nvidia.co.jp/object/cuda_showcase_jp.html
10)Lars Nyland, Mark Harris and Jan Prins:" GPU Gems 3,Fast N-Body Simulation with CUDA", Chapter 31 (2007).
11)http://www.nvidia.co.jp/object/tesla_computing_solutions_jp.html
12)丸山直也,額田彰,松岡聡 : "CPU向けソフトウエアECCの性能評価",IPSJ SIG Technical Report,2009-HPC-119,pp.25-30 (2009)
13)http://www.pgroup.com/resources/accel.htm
14)http://www.amd.com/jp/fusion/Pages/index.aspx
15)http://www.khronos.org/opencl/
16)小玉泰寛 : "GPUの汎用開発環境CUDAを用いた中性子輸送計算の高速化",日本原子力学会「2008年秋の大会」予稿集,A32 (2008)

国際単位系（SI）

表１．SI 基本単位

基本量	SI 基本単位	
	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd

表２．基本単位を用いて表されるSI組立単位の例

組立量	SI 基本単位	
	名称	記号
面積	平方メートル	m^2
体積	立法メートル	m^3
速さ，速度	メートル毎秒	m/s
加速度	メートル毎秒毎秒	m/s^2
波数	毎メートル	m^{-1}
密度，質量密度	キログラム毎立方メートル	kg/m^3
面積密度	キログラム毎平方メートル	kg/m^2
比体積	立方メートル毎キログラム	m^3/kg
電流密度	アンペア毎平方メートル	A/m^2
磁界の強さ	アンペア毎メートル	A/m
量濃度[a]，濃度	モル毎立方メートル	mol/m^3
質量濃度	キログラム毎立法メートル	kg/m^3
輝度	カンデラ毎平方メートル	cd/m^2
屈折率[b]	（数字の） 1	1
比透磁率[b]	（数字の） 1	1

(a) 量濃度（amount concentration）は臨床化学の分野では物質濃度（substance concentration）ともよばれる。
(b) これらは無次元量あるいは次元１をもつ量であるが、そのことを表す単位記号である数字の１は通常は表記しない。

表３．固有の名称と記号で表されるSI組立単位

組立量	SI 組立単位			
	名称	記号	他のSI単位による表し方	SI基本単位による表し方
平面角	ラジアン[b]	rad	1[b]	m/m
立体角	ステラジアン[b]	sr[c]	1[b]	m^2/m^2
周波数	ヘルツ[d]	Hz		s^{-1}
力	ニュートン	N		m kg s^{-2}
圧力，応力	パスカル	Pa	N/m^2	m^{-1} kg s^{-2}
エネルギー，仕事，熱量	ジュール	J	N m	m^2 kg s^{-2}
仕事率，工率，放射束	ワット	W	J/s	m^2 kg s^{-3}
電荷，電気量	クーロン	C		s A
電位差（電圧），起電力	ボルト	V	W/A	m^2 kg s^{-3} A^{-1}
静電容量	ファラド	F	C/V	m^{-2} kg^{-1} s^4 A^2
電気抵抗	オーム	Ω	V/A	m^2 kg s^{-3} A^{-2}
コンダクタンス	ジーメンス	S	A/V	m^{-2} kg^{-1} s^3 A^2
磁束	ウエーバ	Wb	Vs	m^2 kg s^{-2} A^{-1}
磁束密度	テスラ	T	Wb/m^2	kg s^{-2} A^{-1}
インダクタンス	ヘンリー	H	Wb/A	m^2 kg s^{-2} A^{-2}
セルシウス温度	セルシウス度[e]	℃		K
光束	ルーメン	lm	cd sr[c]	cd
照度	ルクス	lx	lm/m^2	m^{-2} cd
放射性核種の放射能[f]	ベクレル[d]	Bq		s^{-1}
吸収線量，比エネルギー分与，カーマ	グレイ	Gy	J/kg	m^2 s^{-2}
線量当量，周辺線量当量，方向性線量当量，個人線量当量	シーベルト[g]	Sv	J/kg	m^2 s^{-2}
酸素活性	カタール	kat		s^{-1} mol

(a)SI接頭語は固有の名称と記号を持つ組立単位と組み合わせても使用できる。しかし接頭語を付した単位はもはやコヒーレントではない。
(b)ラジアンとステラジアンは数字の１に対する単位の特別な名称で、量についての情報をつたえるために使われる。実際には、使用する時には記号rad及びsrが用いられるが、習慣として組立単位としての記号である数字の１は明示されない。
(c)測光学ではステラジアンという名称と記号srを単位の表し方の中に、そのまま維持している。
(d)ヘルツは周期現象についてのみ、ベクレルは放射性核種の統計的過程についてのみ使用される。
(e)セルシウス度はケルビンの特別な名称で、セルシウス温度を表すために使用される。セルシウス度とケルビンの単位の大きさは同一である。したがって、温度差や温度間隔を表す数値はどちらの単位で表しても同じである。
(f)放射性核種の放射能（activity referred to a radionuclide）は、しばしば誤った用語で"radioactivity"と記される。
(g)単位シーベルト（PV,2002,70,205）についてはCIPM勧告2（CI-2002）を参照。

表４．単位の中に固有の名称と記号を含むSI組立単位の例

組立量	SI 組立単位		
	名称	記号	SI 基本単位による表し方
粘度	パスカル秒	Pa s	m^{-1} kg s^{-1}
力のモーメント	ニュートンメートル	N m	m^2 kg s^{-2}
表面張力	ニュートン毎メートル	N/m	kg s^{-2}
角速度	ラジアン毎秒	rad/s	m m^{-1} s^{-1}=s^{-1}
角加速度	ラジアン毎秒毎秒	rad/s^2	m m^{-1} s^{-2}=s^{-2}
熱流密度，放射照度	ワット毎平方メートル	W/m^2	kg s^{-3}
熱容量，エントロピー	ジュール毎ケルビン	J/K	m^2 kg s^{-2} K^{-1}
比熱容量，比エントロピー	ジュール毎キログラム毎ケルビン	J/(kg K)	m^2 s^{-2} K^{-1}
比エネルギー	ジュール毎キログラム	J/kg	m^2 s^{-2}
熱伝導率	ワット毎メートル毎ケルビン	W/(m K)	m kg s^{-3} K^{-1}
体積エネルギー	ジュール毎立方メートル	J/m^3	m^{-1} kg s^{-2}
電界の強さ	ボルト毎メートル	V/m	m kg s^{-3} A^{-1}
電荷密度	クーロン毎立方メートル	C/m^3	m^{-3} sA
表面電荷	クーロン毎平方メートル	C/m^2	m^{-2} sA
電束密度，電気変位	クーロン毎平方メートル	C/m^2	m^{-2} sA
誘電率	ファラド毎メートル	F/m	m^{-3} kg^{-1} s^4 A^2
透磁率	ヘンリー毎メートル	H/m	m kg s^{-2} A^{-2}
モルエネルギー	ジュール毎モル	J/mol	m^2 kg s^{-2} mol^{-1}
モルエントロピー，モル熱容量	ジュール毎モル毎ケルビン	J/(mol K)	m^2 kg s^{-2} K^{-1} mol^{-1}
照射線量（X線及びγ線）	クーロン毎キログラム	C/kg	kg^{-1} sA
吸収線量率	グレイ毎秒	Gy/s	m^2 s^{-3}
放射強度	ワット毎ステラジアン	W/sr	m^4 m^{-2} kg s^{-3}=m^2 kg s^{-3}
放射輝度	ワット毎平方メートル毎ステラジアン	$W/(m^2 sr)$	m^2 m^{-2} kg s^{-3}=kg s^{-3}
酵素活性濃度	カタール毎立方メートル	kat/m^3	m^{-3} s^{-1} mol

表５．SI 接頭語

乗数	接頭語	記号	乗数	接頭語	記号
10^{24}	ヨタ	Y	10^{-1}	デシ	d
10^{21}	ゼタ	Z	10^{-2}	センチ	c
10^{18}	エクサ	E	10^{-3}	ミリ	m
10^{15}	ペタ	P	10^{-6}	マイクロ	μ
10^{12}	テラ	T	10^{-9}	ナノ	n
10^9	ギガ	G	10^{-12}	ピコ	p
10^6	メガ	M	10^{-15}	フェムト	f
10^3	キロ	k	10^{-18}	アト	a
10^2	ヘクト	h	10^{-21}	ゼプト	z
10^1	デカ	da	10^{-24}	ヨクト	y

表６．SIに属さないが、SIと併用される単位

名称	記号	SI 単位による値
分	min	1 min=60s
時	h	1h =60 min=3600 s
日	d	1 d=24 h=86 400 s
度	°	1°=(π/180) rad
分	’	1’=(1/60)°=(π/10800) rad
秒	”	1”=(1/60)’=(π/648000) rad
ヘクタール	ha	$1ha=1hm^2=10^4m^2$
リットル	L, l	$1L=11=1dm^3=10^3cm^3=10^{-3}m^3$
トン	t	$1t=10^3$ kg

表７．SIに属さないが、SIと併用される単位で、SI単位で表される数値が実験的に得られるもの

名称	記号	SI 単位で表される数値
電子ボルト	eV	$1eV=1.602\ 176\ 53(14)\times10^{-19}J$
ダルトン	Da	$1Da=1.660\ 538\ 86(28)\times10^{-27}kg$
統一原子質量単位	u	1u=1 Da
天文単位	ua	$1ua=1.495\ 978\ 706\ 91(6)\times10^{11}m$

表８．SIに属さないが、SIと併用されるその他の単位

名称	記号	SI 単位で表される数値
バール	bar	1 bar=0.1MPa=100kPa=10^5Pa
水銀柱ミリメートル	mmHg	1mmHg=133.322Pa
オングストローム	Å	1 Å=0.1nm=100pm=10^{-10}m
海里	M	1 M=1852m
バーン	b	$1 b=100fm^2=(10^{-12}cm)2=10^{-28}m^2$
ノット	kn	1 kn=(1852/3600)m/s
ネーパ	Np	SI単位との数値的な関係は、対数量の定義に依存。
ベル	B	
デジベル	dB	

表９．固有の名称をもつCGS組立単位

名称	記号	SI 単位で表される数値
エルグ	erg	$1 erg=10^{-7}$ J
ダイン	dyn	$1 dyn=10^{-5}$N
ポアズ	P	1 P=1 dyn s cm^{-2}=0.1Pa s
ストークス	St	1 St =$1cm^2$ s^{-1}=$10^{-4}m^2$ s^{-1}
スチルブ	sb	1 sb =1cd cm^{-2}=10^4cd m^{-2}
フォト	ph	1 ph=1cd sr cm^{-2} 10^4lx
ガル	Gal	1 Gal =1cm s^{-2}=$10^{-2}ms^{-2}$
マクスウェル	Mx	1 Mx = 1G cm^2=10^{-8}Wb
ガウス	G	1 G =1Mx cm^{-2} =10^{-4}T
エルステッド[c]	Oe	1 Oe ≙ $(10^3/4π)$A m^{-1}

(c) 3元系のCGS単位系とSIでは直接比較できないため、等号「 ≙ 」は対応関係を示すものである。

表10．SIに属さないその他の単位の例

名称	記号	SI 単位で表される数値
キュリー	Ci	$1 Ci=3.7\times10^{10}Bq$
レントゲン	R	$1 R=2.58\times10^{-4}C/kg$
ラド	rad	$1 rad=1cGy=10^{-2}Gy$
レム	rem	$1 rem=1 cSv=10^{-2}Sv$
ガンマ	γ	1γ=1 nT=10-9T
フェルミ		1フェルミ=1 fm=10-15m
メートル系カラット		1メートル系カラット = 200 mg = 2×10-4kg
トル	Torr	1 Torr = (101 325/760) Pa
標準大気圧	atm	1 atm = 101 325 Pa
カロリー	cal	1cal=4.1858J（「15℃」カロリー），4.1868J（「IT」カロリー）4.184J（「熱化学」カロリー）
ミクロン	μ	1 μ =1μm=10^{-6}m

（第8版，2006年改訂）