GPUコンピューティング紹介編

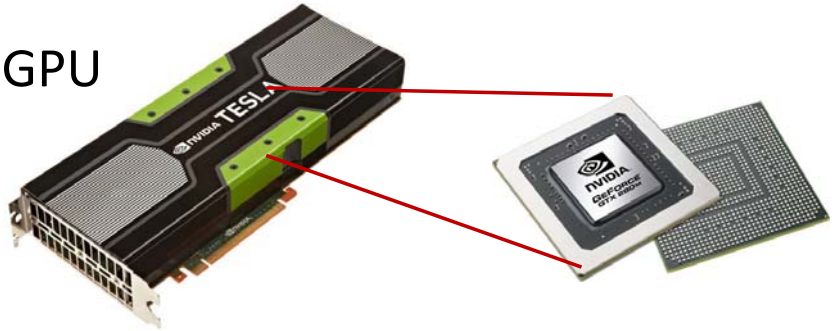
東京工業大学学術国際情報センター

GPUとは

- グラフィックプロセッサ (GPU)とは、高精細なグラフィック(画像)を高速に表示するためのプロセッサ
  - よりきれいなゲームの画像表示などのために、CPUとは独自の進化を続けてきた
  - 現在、CPUの中には2～16個のコア(小さい頭脳)があるのに対し、GPU中には数百～数千ものコアがある

CPU

GPU

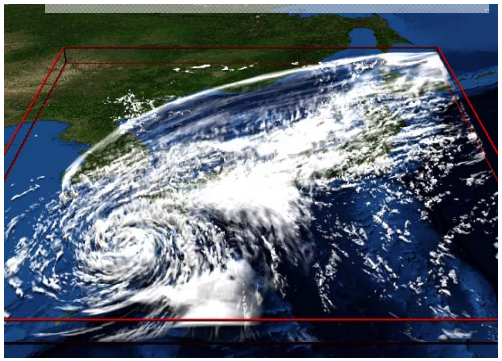


GPUコンピューティングとは

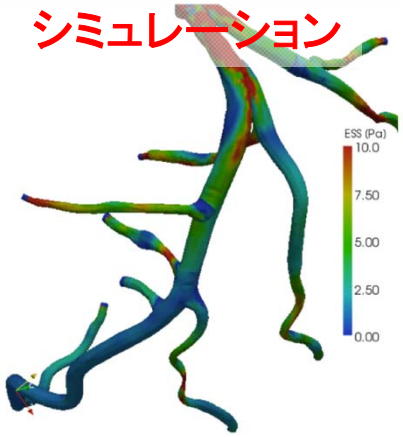
- GPUコンピューティングとは、GPUを一般のプログラムの高速化のために使うこと
  - 画像に関係したプログラムでなくてもよい！
  - GPGPU (General-Purpose computing on GPU) とも言われる
  - 2007年にNVIDIA社のCUDA言語がリリースされてから大きな注目

様々な分野で活躍するGPUコンピューティング

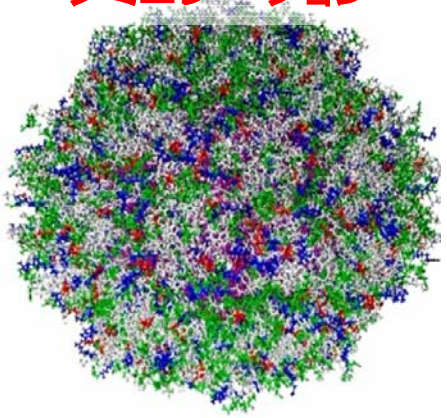
気象シミュレーション



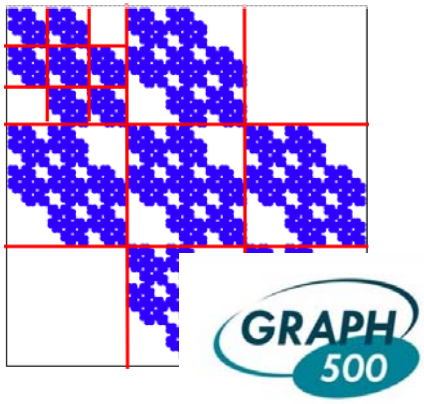
血流
シミュレーション



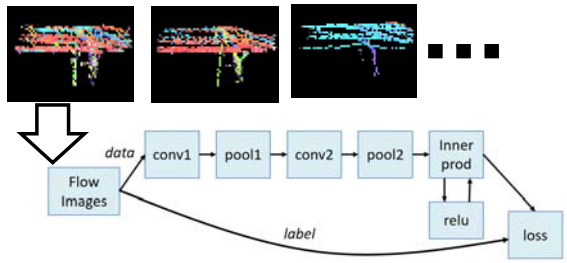
ウィルス分子
シミュレーション



ビッグデータ・グラフ解析



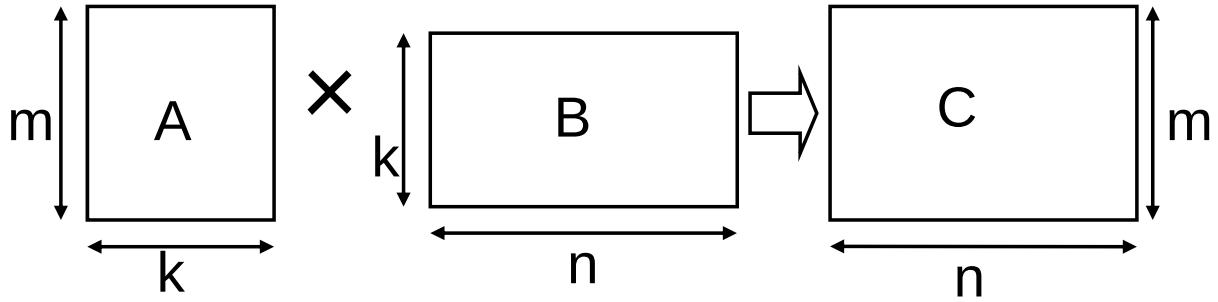
ディープラーニング



AlphaGoもGPUを
多数利用

GPUの計算速度の威力
行列積計算を例題に

- 「行列 = 行列 × 行列」の計算
- 科学技術計算で重要な役割



Cij要素は、Aの第i行とBの第j列の内積

計算の大まかな流れ:

```
for (i = 0; i < m; i++) {  // Cの第i行に注目
  for (j = 0; j < n; j++) {  // Cijに注目
    for (l = 0; l < k; l++) {  // 内積のためのループ
      c[j][i] += a[l][i]*b[l][j];
    } } }
```

GPUの計算速度の威力

m=n=k=2048の場合の計算時間をTSUBAME2.5上で測ってみた

- CPU版 (C言語で書かれたmm.c)

  → 10.1秒かかった

  ※これは1CPUコアだけ使う場合。12CPUコア使って並列化した場合は1.5秒

- GPU版 (「CUDA」で書かれたmm-gpu.cu)

  → 0.32秒。CPU版より約31倍も速い！！

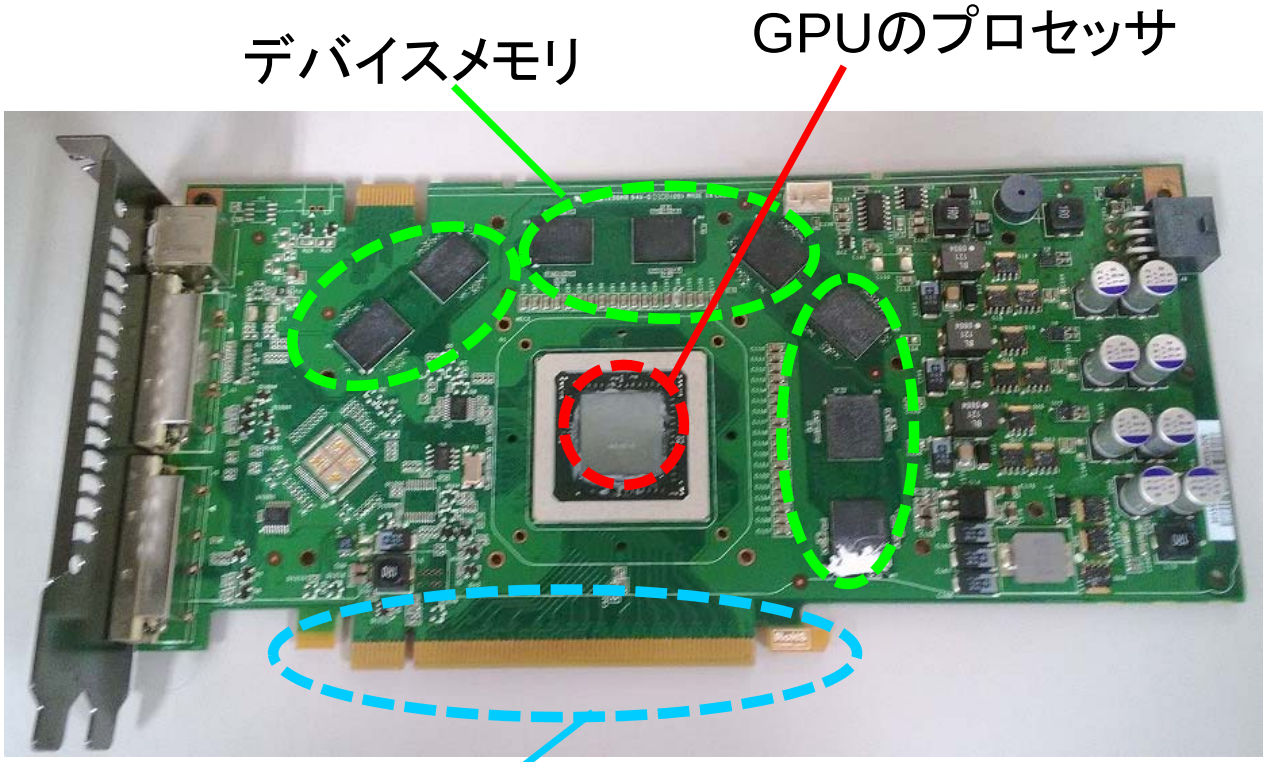
GPUの特徴

- コンピュータにとりつける増設ボード
  ⇒単体では動作できず、CPUから指示を出してもらう
- 多数コアを用いて計算
  ⇒多数のコアを活用するために、多数のスレッドが協力して高速計算
  ⇒うまくいけばCPUより10倍以上の高速計算が可能だが、プログラムの種類によってはCPUより遅い・動かない場合も
- 普通のCPU側のメモリとは別に、GPUがメモリを持つ。そのサイズは製品によって異なり、約1～12GB
  ⇒CPU側のメモリと別なので、「データの移動」もプログラミングする必要

コア数・メモリサイズは、製品によって違う

GPUボードの構成

外側を外した様子

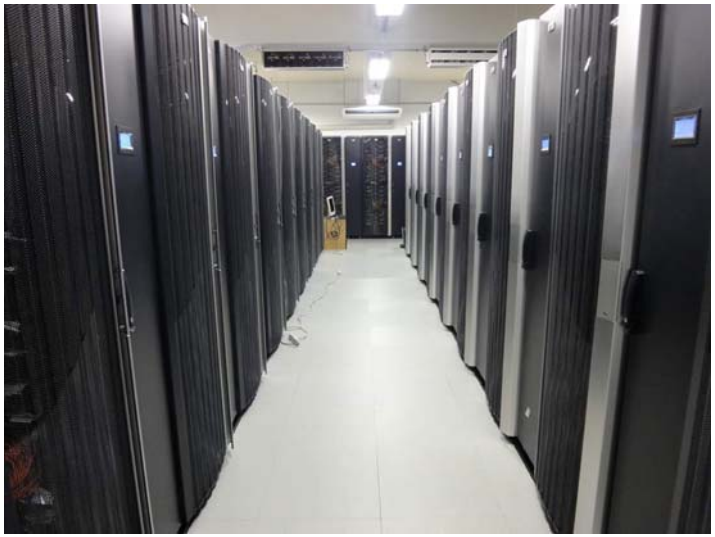


今回使うK20X GPUの性能

- 計算速度: 1.31 TFlops (倍精度)、3.95 TFlops (単精度)
  - 近年のCPUは20～100GFlops程度なので、10倍以上高速
- コア数:
  - 14SMX x 192CUDAコア = 2688CUDAコア
- メモリ容量: 6GB
  - 2688コアが、デバイスメモリと呼ばれる、6GBのメモリを共有している
  - CPU側のメインメモリとは別
- メモリバンド幅: 約250 GB/s
  - CPUは10～50GB/s程度
- その他の特徴
  - キャッシュメモリ (L1, L2)
  - ECC
  - CUDA, OpenAcc, OpenCLなどでプログラミング

以前のGPUにはキャッシュメモリが無かったので、高速なプログラム作成がより大変だった

TSUBAME2スーパーコンピュータ



Tokyo-Tech
Supercomputer and
UBiquitously
Accessible
Mass-storage
Environment

「ツバメ」は東京工業大学のシンボルマークでもある

- TSUBAME1: 2006年～2010年に稼働したスパコン
- TSUBAME2.0: 2010年に稼働開始したスパコン
  - 2010年には、世界4位、日本1位の計算速度性能
- TSUBAME2.5: 2013年にGPUをK20Xへ入れ替え
  - 合計計算性能(理論値)は5.7PFlops。京コンピュータの11PFlopsと良い勝負
  - 現在、世界31位、日本3位

高性能の秘訣が
GPUコンピューティング

TSUBAME2.5の計算ノード

- TSUBAME2.0は、約1400台の計算ノード(コンピュータ)を持つ
- 各計算ノードは、CPUとGPUの両方を持つ
  - CPU: Intel Xeon 2.93GHz 6コア x 2CPU=12 コア
  - GPU: NVIDIA Tesla K20X x 3GPU

    0.07TFlops x 2 (CPU) + 1.31TFlops x 3 (GPU) = 4.08TFlops

96%の性能がGPUのおかげ!

  - メインメモリ(CPU側メモリ): 54GB
    - デバイスメモリ(GPU側メモリ): 6GB x 3GPU
  - SSD: 120GB
  - ネットワーク: QDR InfiniBand x 2 = 80Gbps
  - OS: SUSE Linux 11 (Linuxの一種)

本選では，こんな GPU を使いこなしてもらいます
お楽しみに！

以下は，GPU プログラミングの
さわりをちょっとだけ

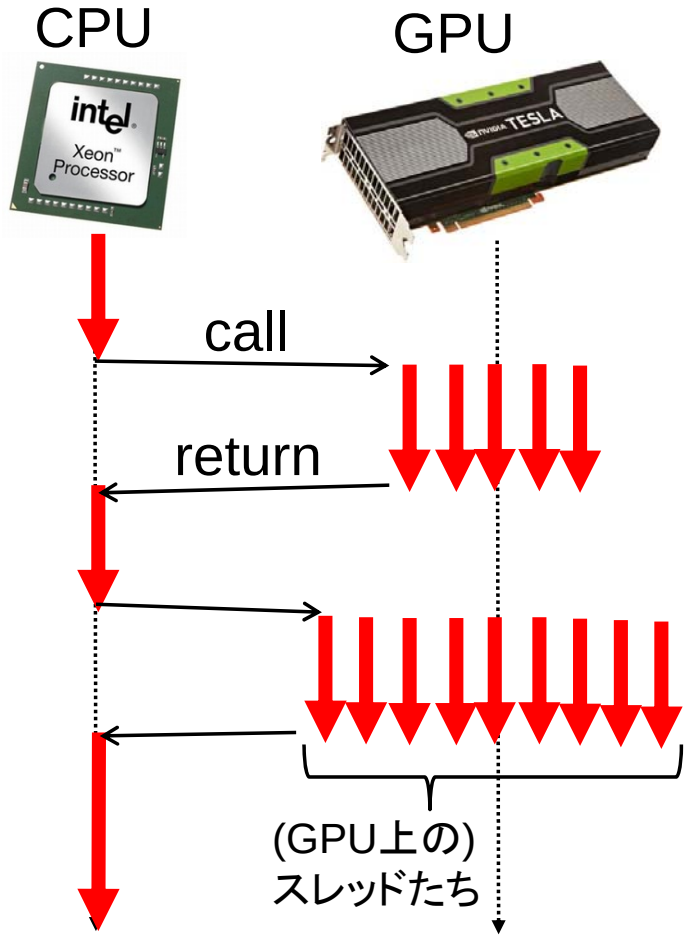
GPU プログラミング入門（そのまた入り口）

プログラミング言語CUDA

- GPU プログラミングのための言語
  正確には NVIDIA GPU向けのプログラミング言語
- 基本は標準C言語　＋　GPGPU用拡張機能
  C言語の基本的な知識(特にポインタ)は必要となります
- nvcc コマンドを用いてコンパイル
  ソースコードの拡張子は.cu

CUDAのプログラミングモデル

~分散メモリと共有メモリの両方登場~



- main()から開始し、当初はCPUだけ動く
  - GPUは、CPUから処理を依頼された時だけ動く
  - GPU上で動く関数 = GPUカーネル関数
- CPUとGPU間は(原則)別のメモリ空間
  - ここは分散メモリ
- GPU上では多数のスレッド達が動き、スレッド達の間は共有メモリ

CUDAプログラム構成

ホスト関数 ＋ GPUカーネル関数

- 二種類の関数がcuファイル内に混ざっている
- ホスト関数
  - CPU上で実行される関数
  - ほぼ通常のC言語。main関数から処理がはじまる
  - GPUに対してデータ転送やGPUカーネル関数呼び出しを実行
- GPUカーネル関数
  - GPU上で実行される関数
  - ホストプログラムから呼び出されて実行
  - (単にカーネル関数と呼ぶ場合も)

CUDAプログラムのコンパイルと実行例

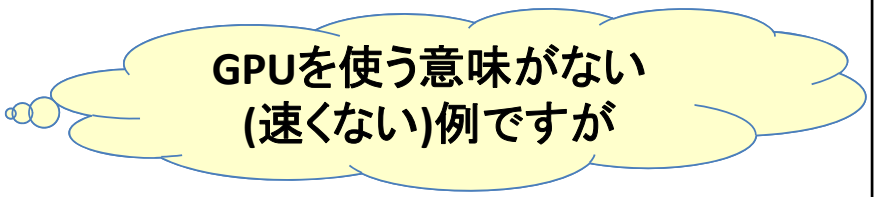
- サンプルプログラム inc_seq.cu で説明
- 以下のコマンドをターミナルから入力し、CUDAプログラムのコンパイル、実行を確認してください

```
$ nvcc inc_seq.cu -o inc_seq
$ ./inc_seq
```

※ “$” はコマンドプロンプトを示しますので、”$”は入力しないでください

サンプルプログラム: inc_seq.cu

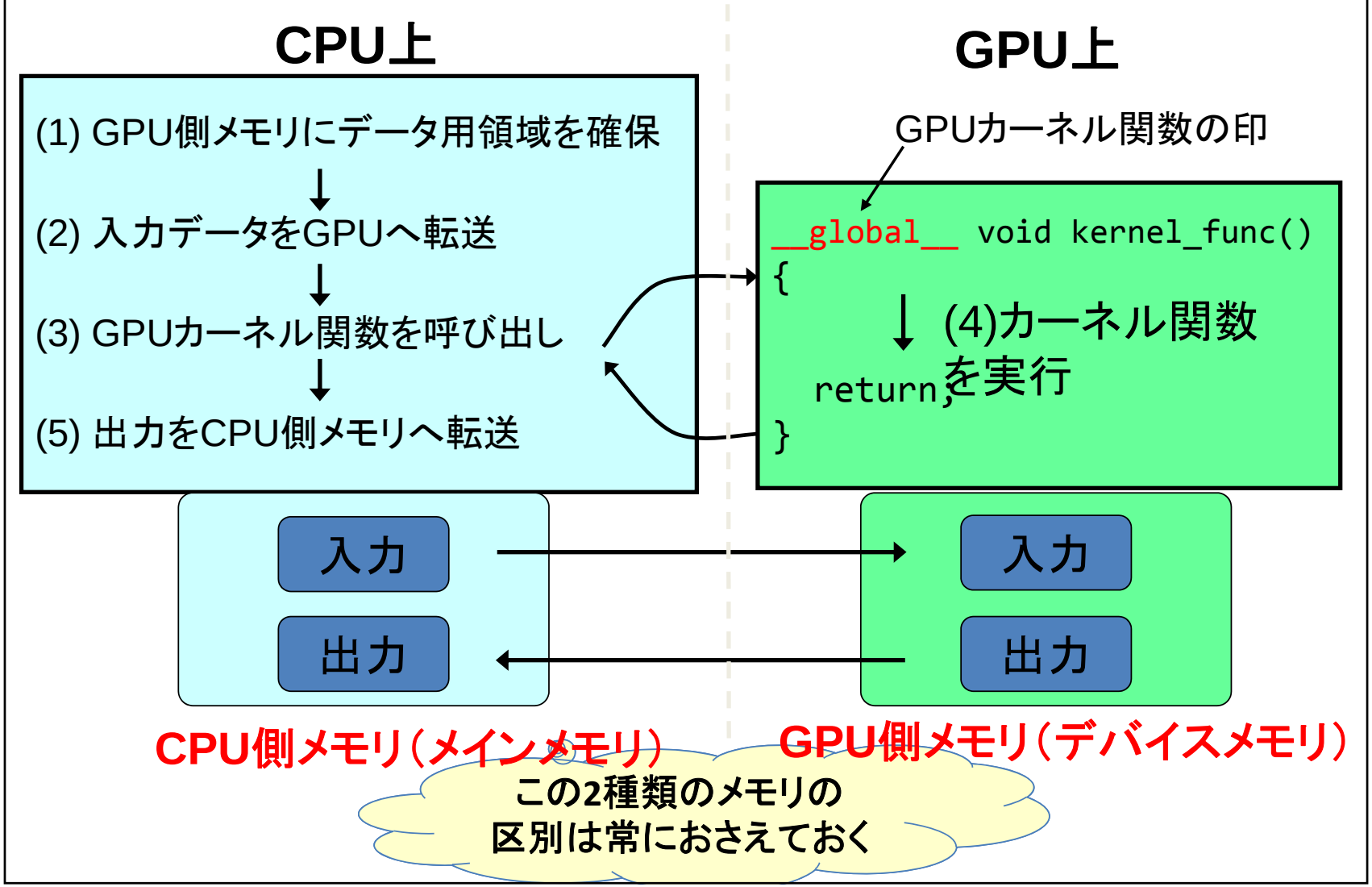
int型配列の全要素を1加算




```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <cuda_runtime.h>

#define N  (32)
__global__ void inc(int *array, int len)
{
    int i;
    for (i = 0; i < len; i++)
        array[i]++;
    return;
}
int main(int argc, char *argv[])
{
    int i;
    int arrayH[N];
    int *arrayD;
    size_t array_size;
```

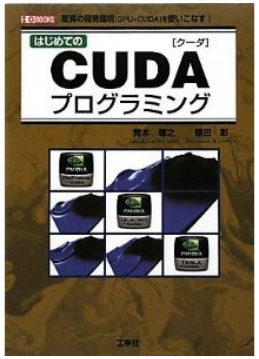
```
    for (i=0; i<N; i++) arrayH[i] = i;
    printf("input: ");
    for (i=0; i<N; i++)
        printf("%d ", arrayH[i]);
    printf("¥n");

    array_size = sizeof(int) * N;
    cudaMalloc((void **)&arrayD, array_size);
    cudaMemcpy(arrayD, arrayH, array_size,
            cudaMemcpyHostToDevice);
    inc<<<1, 1>>>(arrayD, N);
    cudaMemcpy(arrayH, arrayD, array_size,
                cudaMemcpyDeviceToHost);
    printf("output: ");
    for (i=0; i<N; i++)
        printf("%d ", arrayH[i]);
    printf("¥n");
    return 0;
}
```

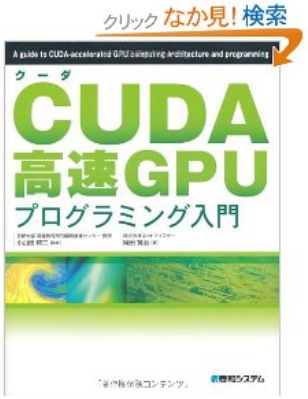
典型的なCUDAプログラムの流れ



まあ，こんなところで ;-)

CUDA関連書籍もありますが
そう心配しないでも大丈夫！
やってみると結構簡単です





著者は東工大
の先生