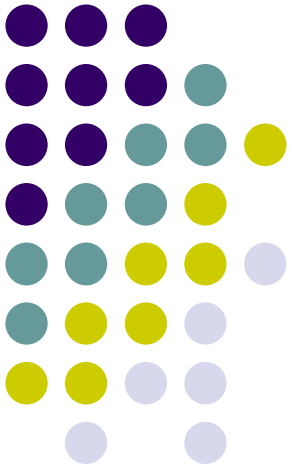
2015年度
実践的並列コンピューティング
第14回

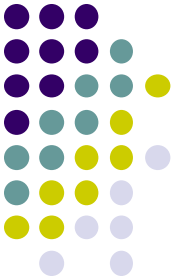
GPUプログラミング (2)



遠藤 敏夫

endo@is.titech.ac.jp

GPU上のスレッド数の考え方が、CPUと違う件



- CPU: スレッド数≦物理コア数がよい
  - TSUBAMEでは12
- GPU: 総スレッド数>>物理(CUDA)コア数がよい
  - しかも、ぎりぎりよりも、数倍以上多い方が速い傾向
  - TSUBAMEでは、GPUあたり2688CUDAコア

  ⇒ 総スレッド数は10,000以上が良く、百万などもok

- CPUとの違いの理由：GPUではコンテクストスイッチが非常に軽い
  - CPUではレジスタ・スタックの退避などをOSがソフトウェアで行う
  - GPUではハードウェアによりほぼゼロクロック
- 数倍多い方が速い理由：
  - メモリアクセスによるひまな時間(ストール)を、他のスレッドが埋めることができる
  - Intel CPUではHyperthreadに相当するが、こちらは物理コア×2まで

なぜCUDAではスレッドが二段階か

- ハードウェアの構造に合わせてある

ハードウェア (数値はK20Xの場合):

1 GPU = 14 SM

1 SM = 192 CUDA core

CUDAのモデル:

1 Grid = 複数thread block

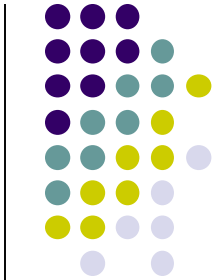
1 thread block = 複数thread

GPUの構造

1スレッドブロックは、必ず1SM上で動作
(複数スレッドブロックがSMを共有するのはあり)
1スレッドは、必ず1 CUDA coreで動作
(複数スレッドがCUDA coreを共有するのはあり)

⇒ TSUBAMEではグリッドサイズが14以上、かつスレッドブロックサイズが192以上(1024以下)の場合に効率的

結局スレッド数はどう決めればよい？



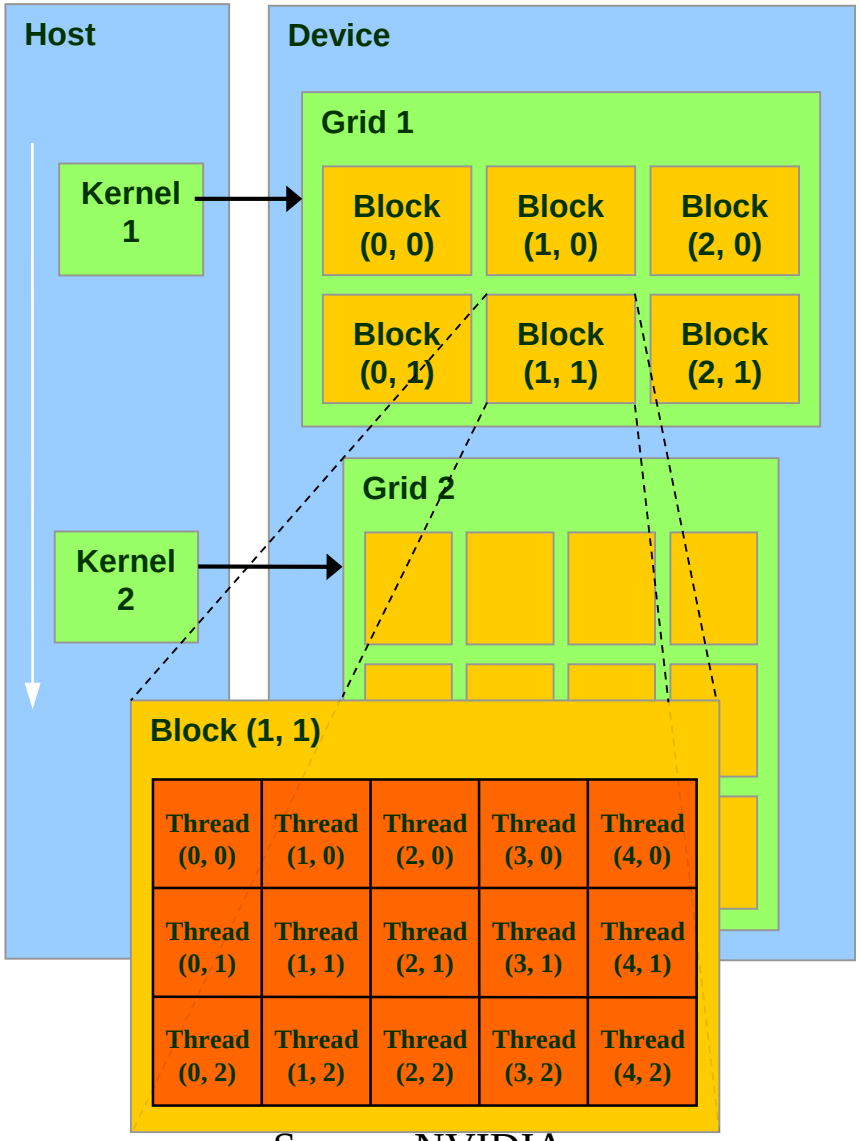
原則的に多いほうが有利

→「計算対象の配列サイズ」=合計スレッドサイズとしてしまうという、極端な方法もok

ただし、

- スレッドブロック数×スレッド数にうまく割り当てる必要 (例: inc_par サンプル)
- CUDAの定める限界値あり (後述)
- 多すぎて不利になるケースはやっぱりある
  - SM内共有メモリの利用が非効率になるなど(次回)
- プログラムが多次元配列を持つ場合、どう割り振るか考える必要
  - たとえば、三次元のステンシル計算のとき、割り振り方は複数考えられるので、決める必要
    - 1スレッド = 1点
    - X, Y方向は並列化して、Z方向は各スレッドに行わる
    - ・・・

CUDAの機能：多次元スレッド指定(1)



Source: NVIDIA

- スレッドブロック数およびスレッド数はそれぞれが
  - int型整数
  - 三次元のdim3型 (CUDA特有)

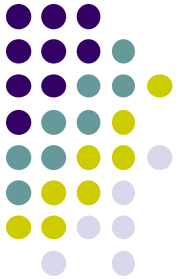
  のどちらか

→ 合わせて最大6次元で指定

- 指定例
  - <<<100, 30>>>
  - <<<dim3(100,20,5), dim3(4, 8, 4)>>>
  - <<<4, dim3(20, 9)>>>

なお、dim3(100,1,1)と100は同じ意味

CUDAの機能：多次元スレッド指定(2)



dim3はx, y, zというメンバを持つ構造体

- 自分のIdを見るには
  - blockIdx.x, ~.y, ~.z: 自分が何番目のブロックにいるか(0以上)
  - threadIdx.x , ~.y, ~.z : 自分がブロック内で何番目のスレッドか(0以上)
- スレッド数などを見るには
  - gridDim.x , ~.y, ~.z : 全体でいくつブロックがあるか
  - blockDim.x , ~.y, ~.z : 各ブロックにいくつのスレッドがあるか

- 多次元指定は、純粋にプログラムのしやすさのためにある
  - 性能には原則関係しない。性能に関係するのは、あくまでブロック数とスレッド数

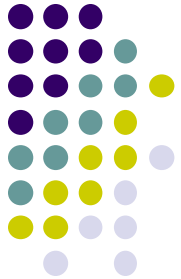
ブロック数・スレッド数の制限

ブロック数・スレッド数に指定可能な最大値にも注意

- TSUBAMEのK20Xでは
  - スレッドブロック数：xは2^{31}-1まで、y・zは65535まで
  - ブロック中スレッド数: x, yは1024まで、zは64まで。さらに総数1024まで

    結構ひっかかりやすい。結局、ブロック中スレッド数は固定にして、ブロック数を大きくすることが多い
  - GPUによって違う。CUDA C Programming GuideのAppendix G参照のこと
    - http://docs.nvidia.com/cuda/cuda-c-programming-guide
    - K20Xの「Compute capability」は3.5

行列積演算 (1)



- 行列積演算サンプルプログラム

  行列A, B, Cがあるとき、C=A×Bを計算する

  全行列とも1024x1024のとき、いくつかのバージョンを比較:

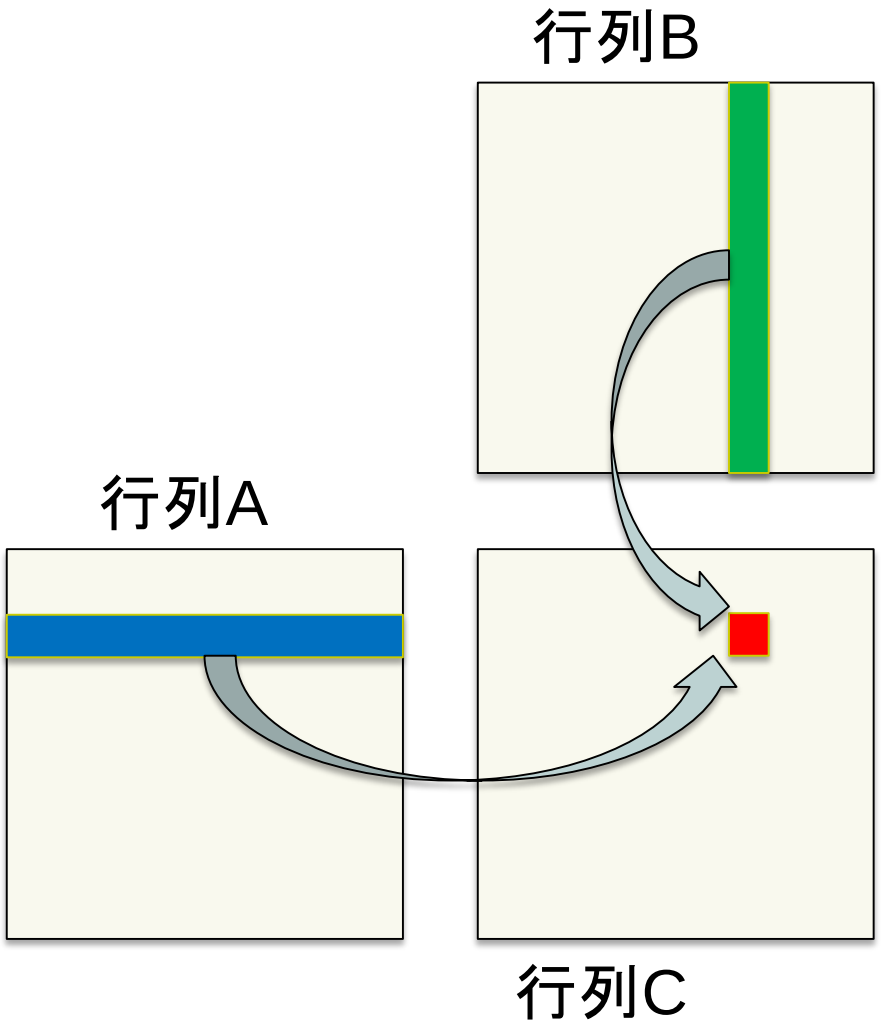
  - mm
    - CPU上の1スレッドで計算 → **約1.1秒**
  - mm-omp
    - CPU上の複数スレッドで計算 → **約0.14秒** (12スレッド)
  - mm-cuda1t
    - GPUの1スレッドで計算 → **約400秒**

  - mm-cuda
    - GPUの複数スレッドで計算

    → **約0.036秒 (cudaMemcpy除くと0.025秒)**

- 参考: cbtest: 速いGPU BLASであるCUBLASを使用

行列積演算(2): cpu版



行列Cの要素$C_{i,j}$を求めるには

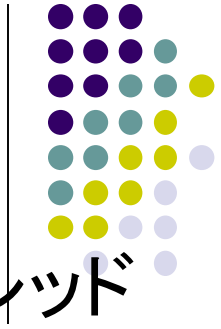
- Aの第i行全体
- Bの第j列全体

の内積計算を行う

→ このためにforループ

C全体を計算するためには、
三重のforループ

行列積演算 (3): GPU並列版



- mm-cudaでは、m x n個のスレッドを用い、1スレッドがCの1要素を計算

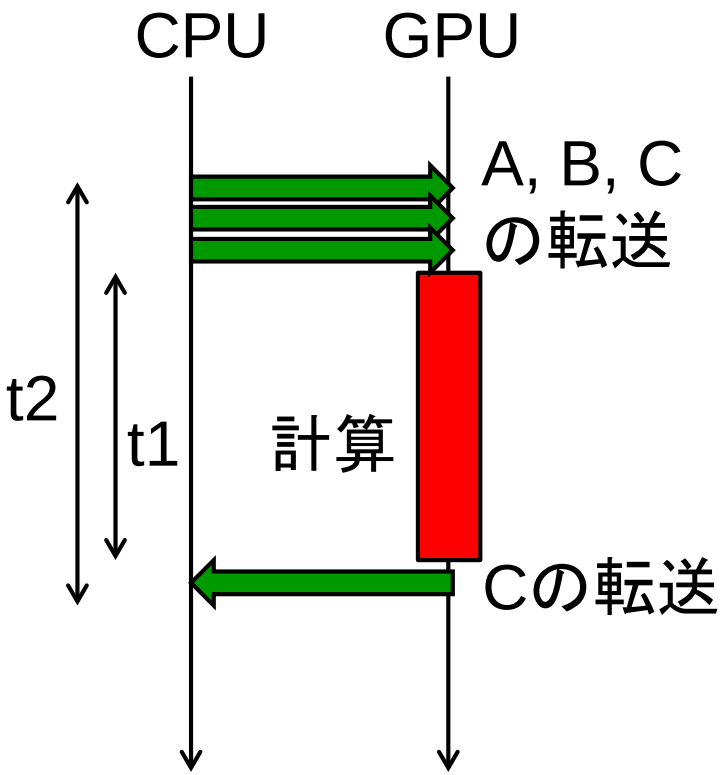
```
matmul_kernel<<<dim3(m / BS, n / BS), dim3(BS, BS)>>>
        (DA, DB, DC, m, n, k);
```

BSは前もって適当に決めた数(16)

- カーネル関数は内積のための一重forループ
- グリッドサイズ・ブロックサイズとも二次元で指定

ちなみに、更なる並列化のために、Cの1要素の計算を複数スレッドで行うのは容易ではない (内積のreduction時にスレッド間の同期が必要)

データ転送時間を性能測定に含めるべきか否か？



- 一概にはどちらが正しいとは言えない。実用的なプログラムでは、前後の文脈によるため
- サンプルプログラムでは、t1とt2両方表示

- t1 ≒ cmnk
- t2 ≒ t1 + d(mk+kn+2mn)
  - c,dはアーキテクチャから決まる定数

時間計測に関する注意

- プログラム中の各部分にかかる時間を測るために、clock(), gettimeofday()関数を使うことはよくある
- CUDAプログラムで以下を測るとき注意が必要

  (a) cudaMemcpy(ホスト→デバイス方向)

  (b) カーネル関数呼び出し
- 本当の時間よりもはるかに短く見えてしまう
  - 実際には、上記(a)(b)を実行すると、「仕事を依頼しただけ」の状態で、実行が帰ってきてしまう(非同期呼び出し)

  → 時刻測定前にcudaDeviceSynchronize()を行っておくこと
  - cudaDeviceSynchronize()の意味:「現在までにGPUに依頼した仕事が、全部終了するまで待つ」

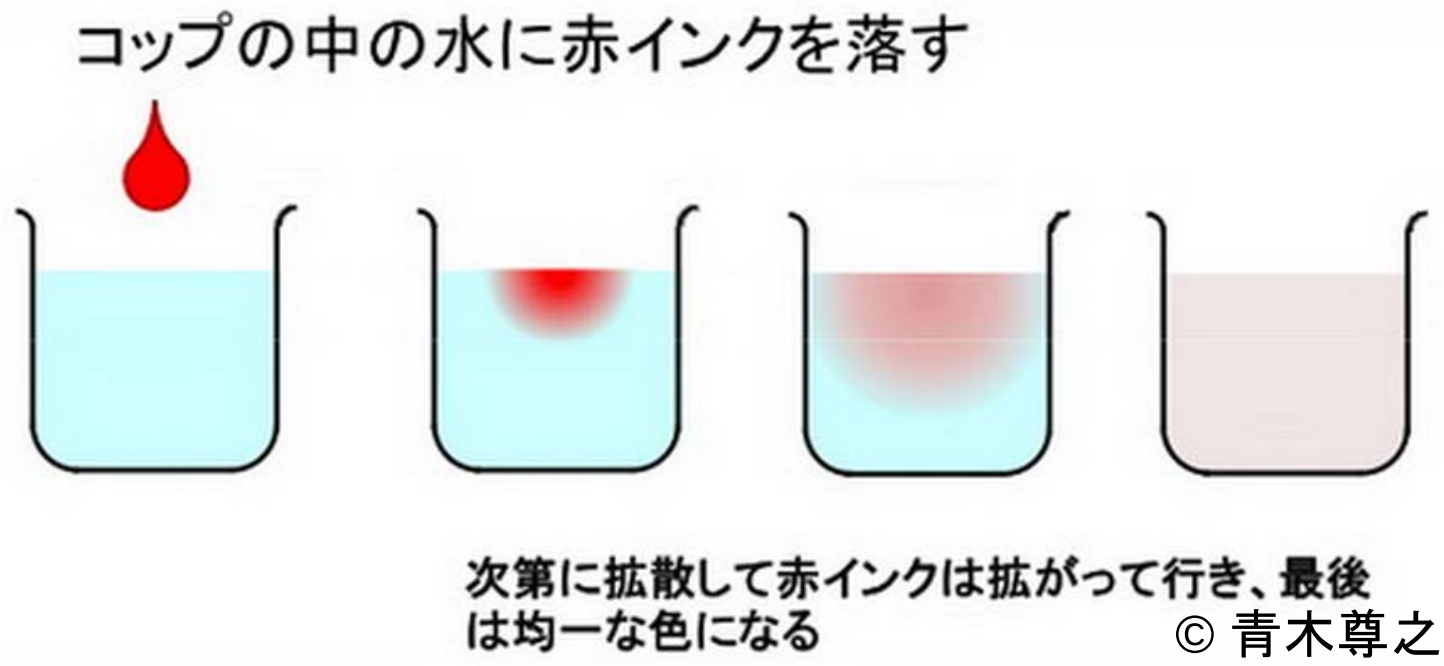
各部分ごとの時間計測を行うには

```
clock_t t1, t2, t3, t4

cudaDeviceSynchronize(); t1 = clock();
cudaMemcpy(･･･, cudaMemcpyHostToDevice);

cudaDeviceSynchronize(); t2 = clock();
my_kernel<<<･･･, ･･･>>>(･･･);

cudaDeviceSynchronize(); t3 = clock();
cudaMemcpy(･･･, cudaMemcpyDeviceToHost);

cudaDeviceSynchronize(); t4 = clock();
```

- t1とt2の差分が、cudaMemcpy (ホストからデバイス)の時間
- t2とt3の差分が、カーネル関数実行にかかった時間
- t3とt4の差分が、cudaMemcpy (デバイスからホスト)の時間

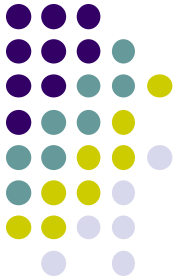
再掲
サンプルプログラム: diffusion

拡散現象

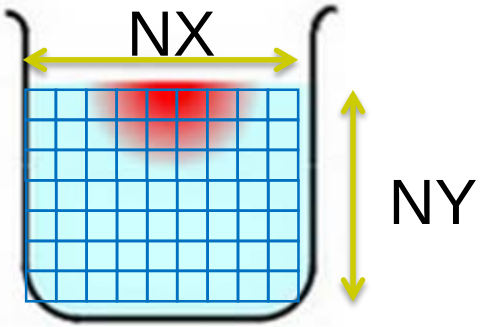


- 各点のインク濃度は、時間がたつと変わっていく
  → その様子を計算機で計算
  - 天気予報などにも含まれる計算
  - GPUで並列化するには？？

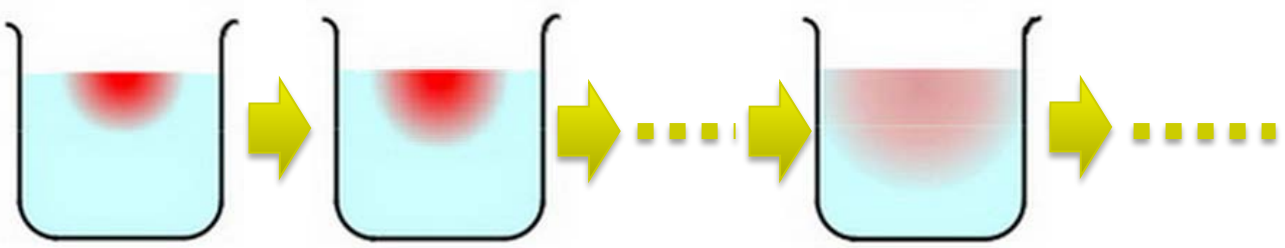
再掲:
diffusionのデータ構造



- シミュレーションしたい空間をマス目で区切り、配列で表す(本プログラムでは二次元配列)



- 時間を少しずつ、パラパラ漫画のように進めながら計算する

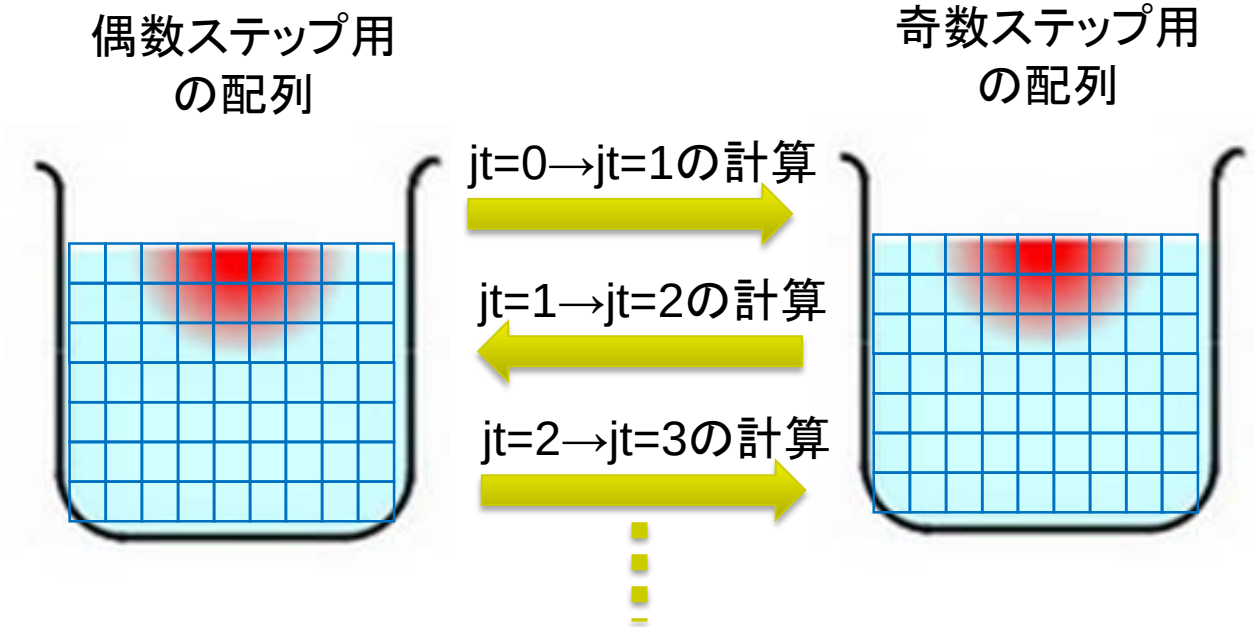


時間ステップ jt=0　　jt=1　　jt=20

再掲:
ダブルバッファリング技術

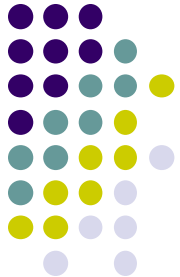
- 全時間ステップの配列を覚えておくとメモリ容量を食い過ぎる

→ 二ステップ分だけ覚えておき、二つの配列(ダブルバッファ)を使いまわす



※ サンプルプログラムでは、大域変数
float data[2][NY][NX];
で表現

想定されるGPU版diffusionの流れ



CPU上で初期条件作成

cudaMallocでGPUメモリ上の領域確保(配列二枚分)

初期条件の二次元格子データをCPUからGPUへ(cudaMemcpy)

For (jt = 0; jt < nt; jt++) //時間ループ

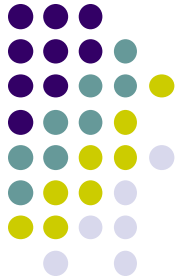
　　GPUカーネル関数を呼出し、その中で全格子点を計算

　　二つのバッファを交換

結果の二次元格子データをGPUからCPUへ(cudaMemcpy)

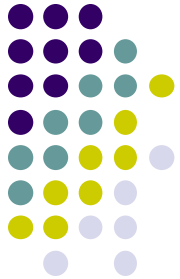
※ 時間ループの中に(格子全体の)cudaMemcpyを置くと非常に遅い (各自試してみましょう)

それ以外に気を付ける場所



- GPUカーネル内から(普通の=ホストメモリ上の)大域変数はアクセスできない
  - 配列のありかを渡すための一つの方法は、GPUメモリをさすポインタを引数として渡す
  - 一次元配列を使うのであれば、インデックスの計算が必要
    - data[from][jy][jx] → ddata[from*NY*NX+jy*NX+jx] など

本授業のレポートについて



- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする
  - OpenMPパート
    - ノード内のCPUコアを使う並列プログラミング
  - MPIパート
    - 複数ノードを使う並列プログラミング
  - GPU(CUDA)パート
    - 1GPU内の数百コアを使う並列プログラミング

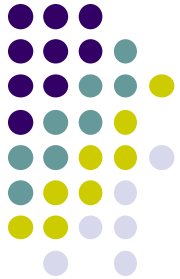
GPUパート課題説明 (1)

以下のG1, G2, G3の、いずれかについてレポートを提出してください.

[G1] 行列積プログラムの性能を、行列サイズを変化させながら性能評価してください。CPU(OpenMP)版とも比較してください。

- データ転送コストを速度計算に入れる場合・入れない場合それぞれについて測定
  - 転送コストが相対的に大きくなるのはどういう場合か。計算量オーダー、転送量オーダーにも触れて議論すること
- GPU版とCPU版の性能比についても調べること。差が大きいとき、小さいときはどういう場合か
- プログラムを改良してもok

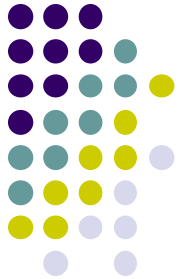
GPUパート課題説明 (2)



[G2] diffusionサンプルプログラムをGPUを用いて並列化し、性能評価してください。

- CPU1コア版は
  ~endo-t-ac/ppcomp/15/diffusion/
- 参考プログラム: …/advection-cuda/
- 改良してもok。たとえば
  - Divergent分岐の影響の削減
  - Shared memoryの利用による高速化
  - マルチGPUの利用
  - ほか

GPUパート課題説明 (3)



[G3] 自由課題：任意のプログラムを，GPU を用いて並列化し、性能評価してください

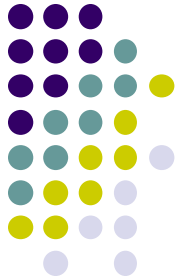
- たとえば，過去のSuperConの本選問題

  http://www.gsic.titech.ac.jp/supercon/

  たんぱく質類似度(2003)，N体問題(2001)・・・

  入力データは自分で作る必要あり
- たとえば，自分が研究している問題

課題の注意



- いずれの課題の場合も，レポートに以下を含むこと
  - 計算・データの割り当て手法の説明
  - TSUBAME2などで実行したときの性能
    - スレッド数、スレッドブロック数を様々に変化させたときの変化に触れているとなお良い
    - 問題サイズを様々に変化させたとき(可能な問題なら)
  - 高性能化のための工夫が含まれているとなお良い
    - 「XXXのためにXXXをしてみたが高速にならなかった」のような失敗でも可
  - プログラムについては，zipなどで圧縮して添付
    - 困難な場合，TSUBAME2の自分のホームディレクトリに置き，置き場所を連絡

課題の提出について

- GPUパート提出期限
  - 8/10 (月) 23:50
- OCW-i ウェブページから下記ファイルを提出のこと
- レポート形式
  - 本文：PDF, Word, テキストファイルのいずれか
  - プログラム：zip形式に圧縮するのがのぞましい

- OCW-iからの提出が困難な場合、メールでもok
  - 送り先： ppcomp@el.gsic.titech.ac.jp
  - メール題名： ppcomp report

次回/Next

- 7/27(月)
  - GPUプログラミング(3) (最終回)

  - スケジュールについてはOCW pageも参照
    - http://www.el.gsic.titech.ac.jp/~endo/

    → 2015年度前期情報(OCW) → 講義ノート