大規模並列計算およびGPU計算

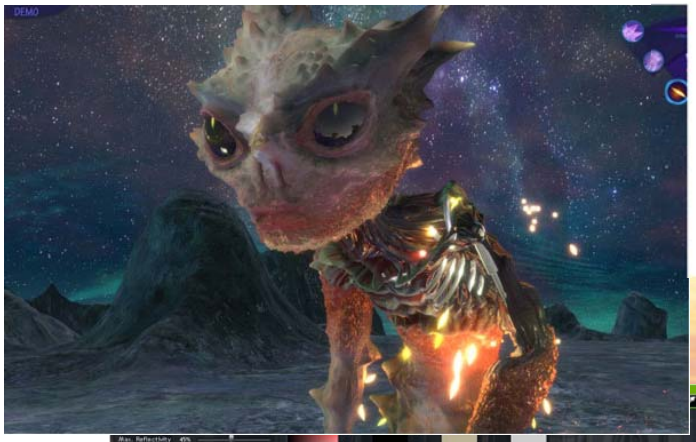
東京工業大学　学術国際情報センター

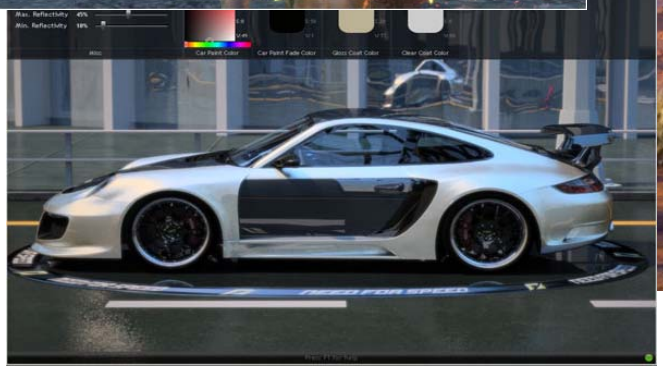
青木　尊之

taoki@gsic.titech.ac.jp

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

GPUとは





Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

GPUコンピューティング

GPU を画像処理以外の目的の
一般的な計算に使う

= GPGPU

(General-Purpose computing on GPUs)

Copyright © Global Scientific Information and Computing Center, Tokyo Institute of Technology

GPGPU

(General-purpose computing on graphics processing units)

GPU を画像処理以外の一般的計算に使う

GPUコンピューティング(GPGPU)の魅力

- 高性能：ハイエンド GPU はピーク 3 TFlops 超
- 低価格：ハイエンドでもコンシューマタイプは 数万円
- 手軽さ：普通のPCにも装着できる
- プログラミング開発：無償の開発環境

CPUと比較して単一
GPUは高消費電力 → 低消費電力: FIOPS／W

Copyright © Global Scientific Information and Computing Center, Tokyo Institute of Technology

Supercomputer in the world

GP GPU

2011 November

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024	10510.00	11280.38	12659.9
2	National Supercomputing Center in Tianjin China	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0
3	DOE/SC/Oak Ridge National Laboratory United States	Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.0
4	National Supercomputing Centre in Shenzhen (NSCS) China	Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120640	1271.00	2984.30	2580.0
5	GSIC Center, Tokyo Institute of Technology Japan	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.6

Copyright © Global Scientific Information and Computing Center, Tokyo Institute of Technology

CPU/GPU Spec Sheet

GP GPU

		Intel Xeon X5670	Tesla C2050 /M2050	GeForce GTX 680 Kepler
GPU	Peak Performance [GFlops]	76.8*,153.6	515*,1030	???*,3090
	Number of Processor	6	448	1536
	Core Clock [GHz]	2.93	1.15	1.058
Memory	Bandwidth[GB/s]	32.0	148.8	192.2
	Memory Interface [bit]	64	384	256
	Memory Clock [GHz]	1.333 (DDR3)	1.50 (GDDR5)	2.00 (GDDR5)
B_{peak}/F_{peak}	Bandwidth/Performance	0.416	0.289	???

Tesla M2050
Peak Power : 225W

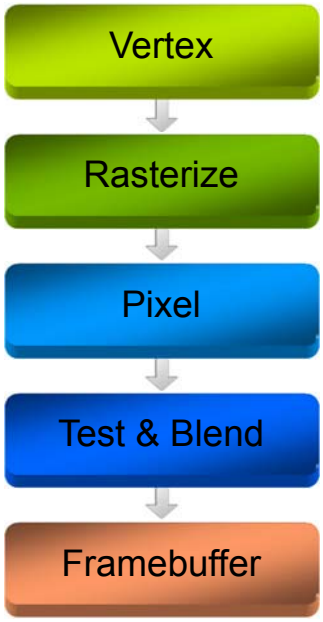
Peak Power : 195W

Copyright © Global Scientific Information and Computing Center, Tokyo Institute of Technology

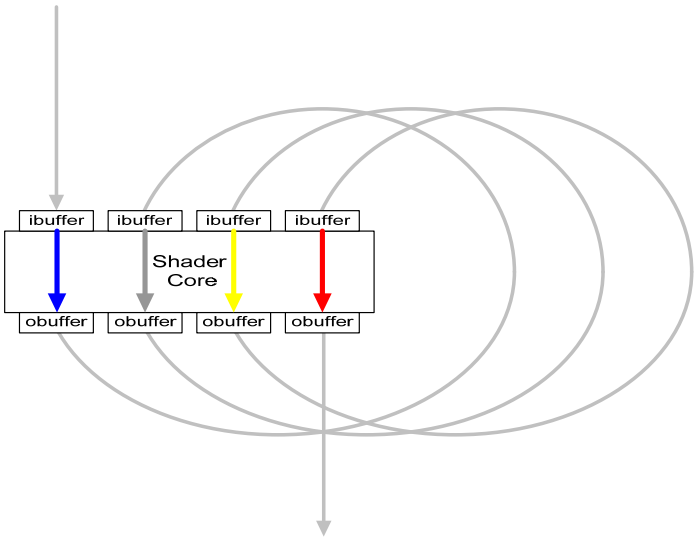
GPUアーキテクチャーの進化

GP GPU

Graphics Pipeline



Unified Shader



Shader 言語

GP GPU

Unified Shader: プログラマブル・シェーダー

OpenGLやDirectX などのAPIに専用のプログラマブルな
シェーディング機能

Open GL では version 1.5, DirectX では version 8 から

Shader プログラミング言語

OpenGL: DLSL 言語
DirectX: HLSL 言語

NVIDIA 独自の Cg (C for Graphics) 言語（HLSL 似）

汎用計算を Graphics の機能に置き換えてプログラミング

CUDA（2006年）

GP GPU

■ 標準C言語（FORTRAN）＋GPGPU用拡張機能
CUDA Runtime (Driver) API
GPU kernel 関数

■ SPMD的（一部SIMD）プログラミング・モデル
(Single Program, Multiple Data)

■ 2011年9月 現在 v4.0 が配布
Windows, Linux, Mac OS X と NVIDIA CUDA enable
GPUの組み合わせで利用可能 (GeForce 8000以降）

■ 現状のGPGPUサポート言語で最も普及
Cf. Open CL, Brook+(AMD), etc.

GeForce 8800 (2006)

GP GPU

GPU computing 世代の発の GPU

「シェーダ言語でアクセスできる高性能グラフィックプロセッサ」

◎Unified-Shader型アーキテクチャ
◎DirectX 10(Direct3D 10) Shader Model 4.0準拠
◎128ストリームプロセッサ
◎96 ROP
◎384bitメモリインターフェイス

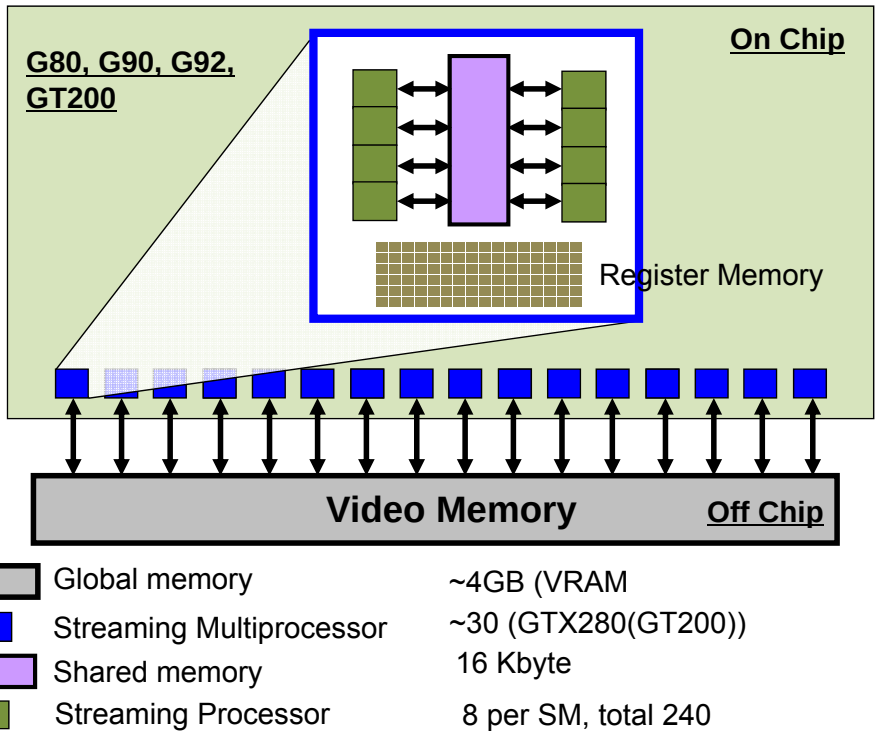
Jen-Hsun Huang

David B. Kirk

「C言語で利用できる膨大な浮動小数点並列プロセッサ」

◎極めて粒度の小さなマルチスレッディング
◎ライトバック制御が可能なキャッシュ
◎CUDA でプログラミング可能
◎スカラ型のIEEE 754“準拠”ストリームプロセッサ

NVIDIA G80 Architecture



NVIDIA Fermi GPU (2010)

◎「Streaming Multiprocessor(SM)」の改革

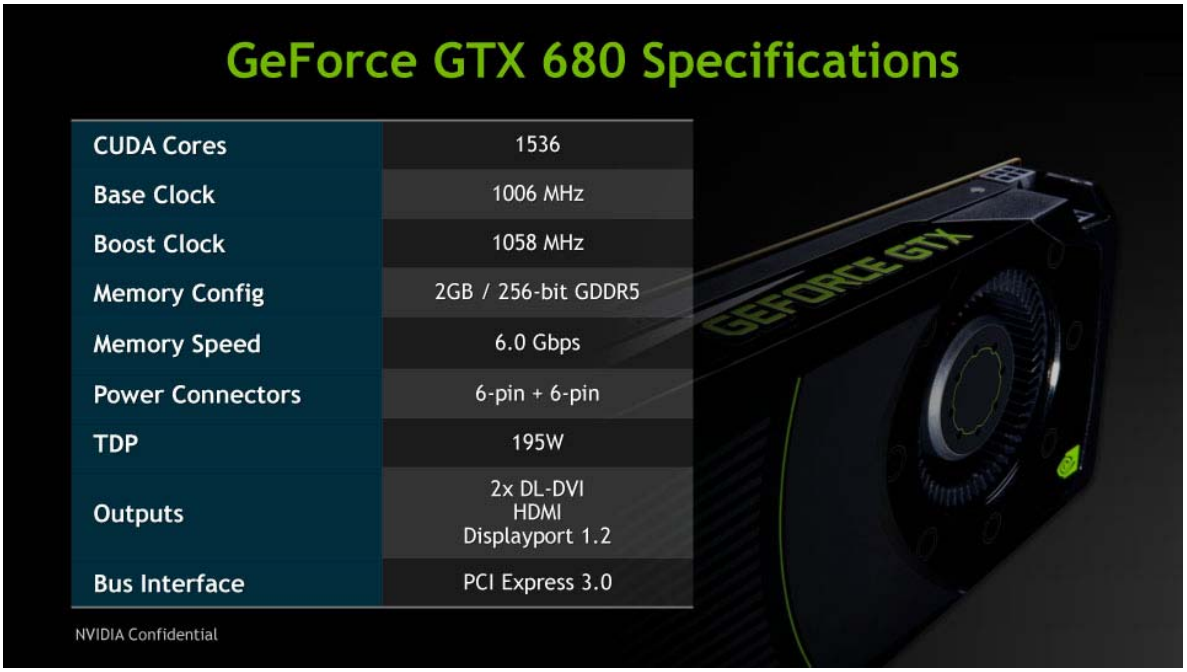
- プロセッサコア「CUDA Core(SP)」の数をSM当たり8個から32個に
- 倍精度浮動小数点演算の性能を単精度の1/2に拡張（従来は1/8）
- SM内部のライタブルメモリを16KBから64KBに拡張しコンフィギュラブルに（L1キャッシュ/Shard Memory）

◎メモリサブシステムとメモリ階層の拡張

- L1/L2キャッシュ階層をリードオンリーからライタブルに
- オンチップも含めて全てのメモリでECCをサポート

- IEEE 754-2008スタンダードに単精度/倍精度とも対応
- 複数カーネルプログラムの同時実行が可能

Kepler GPU



並列処理

GPUの内部には400以上のプロセッサ（CUDAコア）がある。これをうまく使うことによりGPU本来の性能を引き出すことができる。

並列処理（計算）を制する（マスターする）者は、GPUコンピューティングを制す。

並列計算は、

タスク並列　と　データ並列

に分けられる。

例2：算数のテスト

GP GPU

14 + 34 =	10 + 23 =	29 - 31 =
8 + 50 =	12 + 55 =	- 2 - 20 =
22 + 92 =	42 - 23 =	51 - 44 =
33 - 23 =	13 - 7 =	- 39 + 28 =
- 5 + 40 =	31 + 12 =	- 27 + 10 =
- 1 + 70 =	15 + 77 =	- 9 - 62 =
17 + 3 =	3 + 12 =	30 - 38 =
100 - 40 =	- 7 + 80 =	78 + 22 =
6 + 25 =	- 23 - 2 =	- 54 + 70 =

数字だけが異なるが、計算の手続きは同じ。

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

テスト実施の手続き（1）

GP GPU

生徒48人のクラスで算数のテストを実施する。48題の問題の中から、生徒がそれぞれ違う問題を解いて、先生に提出する。

先生が行うこと:

- 事前に算数ドリルを生徒に配布。
- クラスの生徒を班に分ける。
- 回答「始め」と「終わり」の合図。
- 答案の回収と採点および平均点の算出。

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

テスト実施の手続き（2）

GP GPU

生徒が行うこと:

・自分はどの問題をやればよいか指示を読む。
・問題を解いて計算する。
・何ページの問題を解いたかと、その答えを解答用紙に記入し、名前を書いて提出。

※ 先生はクラス全員に同じ指示しか出せないとする。

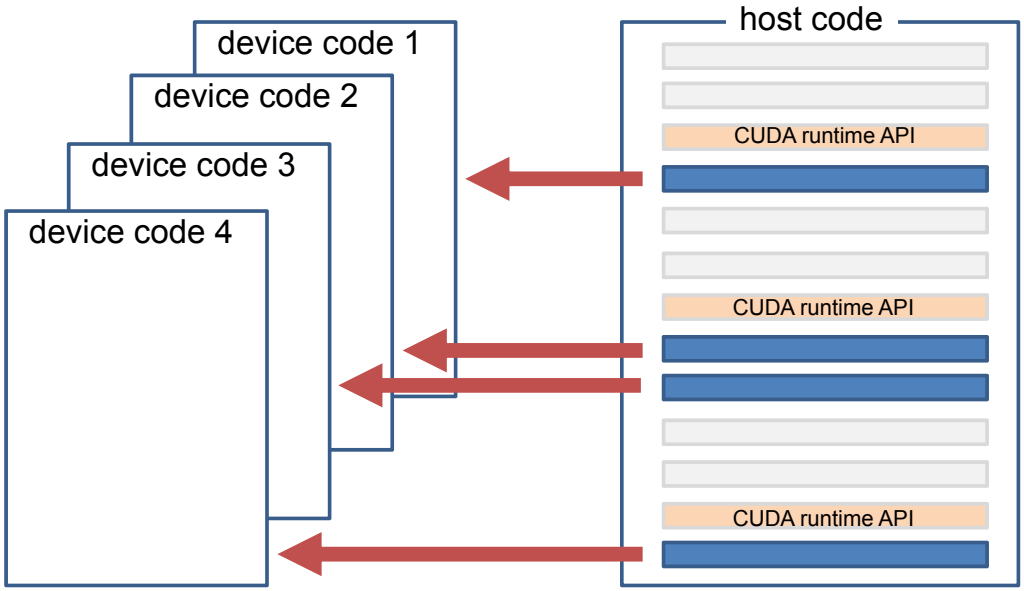
CUDAのプログラム構成

GP GPU

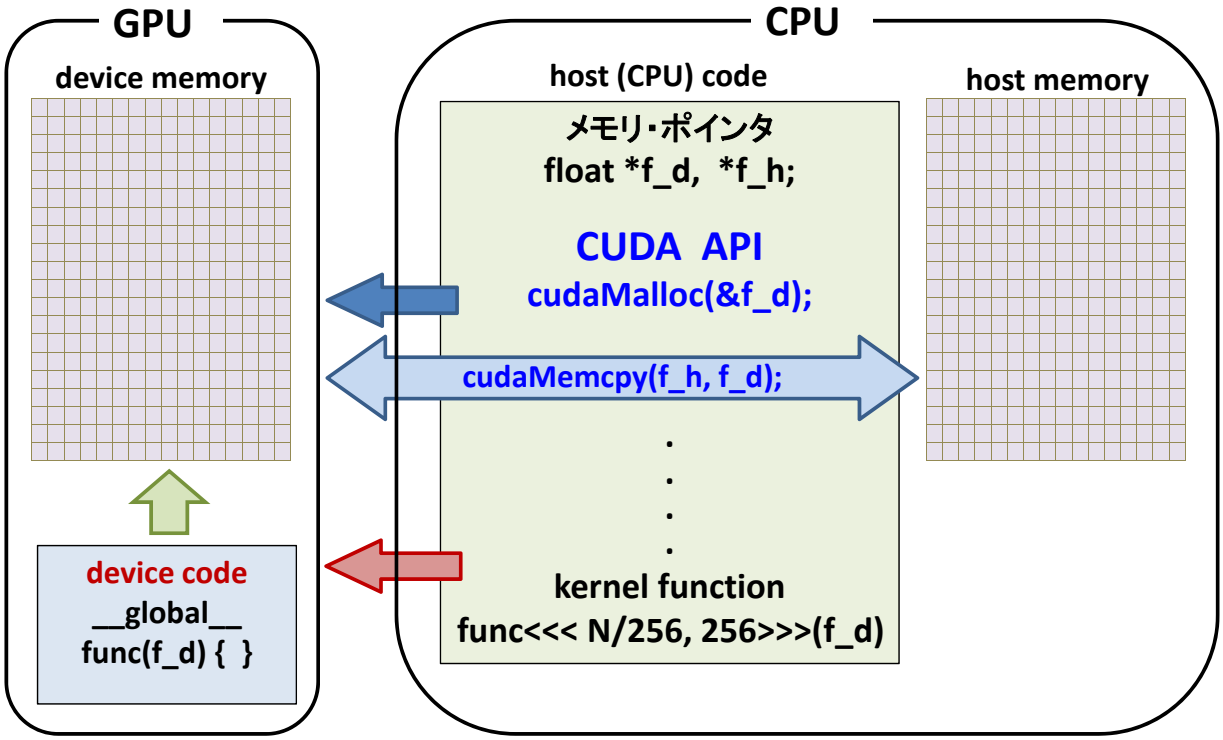
- host (CPU) code と GPU kernel 関数を記述する device code から構成される
- host code には、CUDA API（それに付随した変数）と GPU kernel 関数 call が含まれる。
- device code（GPU kernel 関数）は、thread の実行内容を記述。thread ID 等のビルトイン変数、準備された特別な関数等は含まれる以外は通常のC言語が使える。return 値は不可。

host code と device code

GPUは単独では動かない。host を CPU で実行させ、
その中からの CUDA API と GPU kernel 関数を call



CUDAのプログラム実行の概念図



GPU カーネル関数 call

host code の中で次のように call する。

function<<< Dg, Db, Ns, S>>>(a, b, c, ...);

Dg: dim3 タイプの grid のサイズ指定
Db: dim3 タイプの block のサイズ指定
Ns: 実行時に指定する shared メモリのサイズ
省略可： 省略した場合は、0 が設定
S: 非同期実行の stream 番号
省略可： 省略した場合は、0 が設定され、
GPUのthread間は同期実行となる

■ カーネル関数の内容がGPUで1スレッドとして実行される。
■ Dg, Db で指定される数の thread が実行される。
■ カーネル関数の実行は、CPU に対して絶えず非同期。

dim3 修飾子

function<<< Dg, Db, Ns, S>>>(a, b, c, ...);

の Dg, Db を dim3 で指定する。

dim3 a(n);	⇦ 等価 ⇨	dim3 a(n, 1, 1);
dim3 a(n, m);	⇦ 等価 ⇨	dim3 a(n, m, 1);

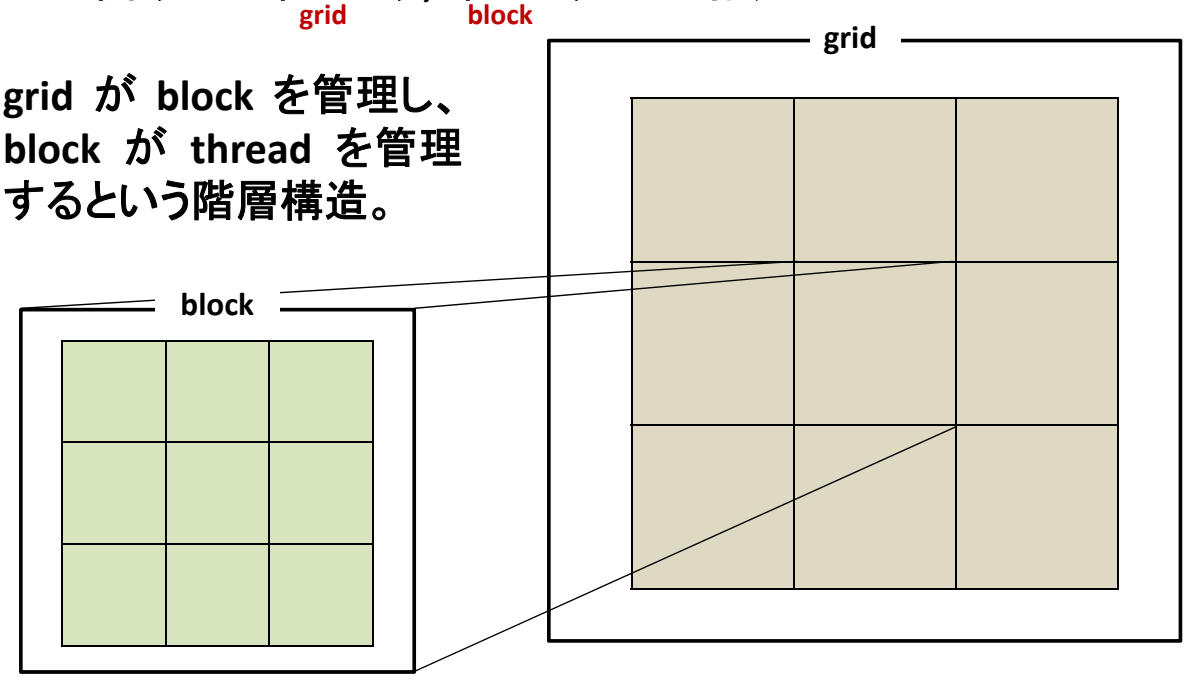
dim3 a(n, m, k);	⇦ 等価 ⇨	a.x = n;
		a.y = m;
		a.z = k;

dim3 a(n_0, m_0, k_0); は宣言と共に値の代入であり、
随時 a.x = n_1; a.y = m_1; a.z = k_1; と変更可能である。

threadの管理

カーネル関数<<< 第1引数, 第2引数>>>で指定
grid block

grid が block を管理し、
block が thread を管理
するという階層構造。



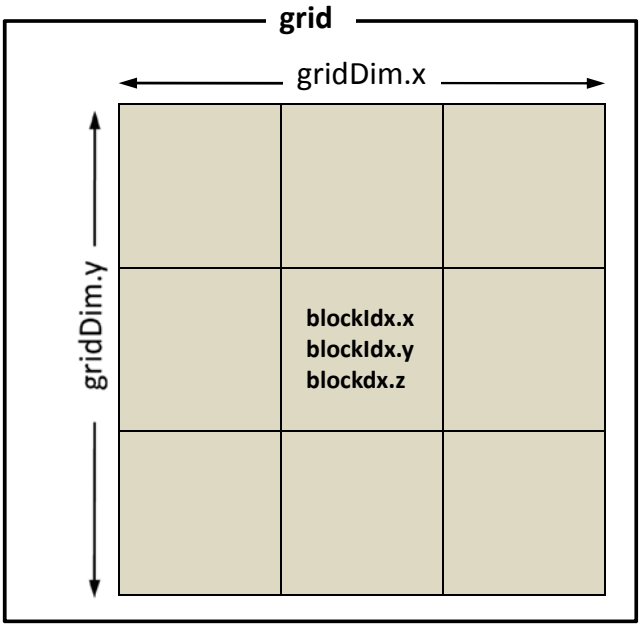
threadの管理 (grid)

kernel 関数の第1引数を以下の grid で指定すると、

```
dim3 Dg(m, n, k);
  Dg.x = m;
  Dg.y = n;
  Dg.z = k;
```

grid の中に n*m*k 個
の blockがある。

kernel 関数の中では、
m → gridDim.x
n → gridDim.y
k → gridDim.z



Threadの管理 (block)

kernel 関数の第2引数を以下の block で指定すると、

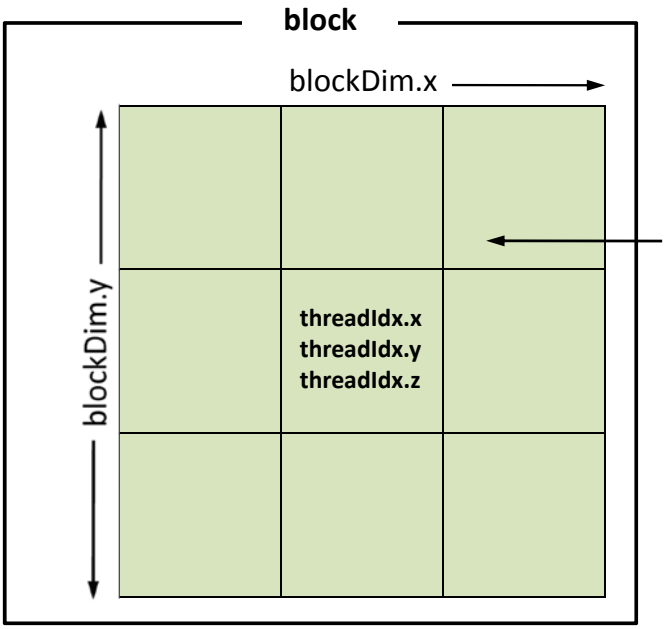
```
dim3 Db(m, n, k);
  Db.x = m;
  Db.y = n;
  Db.z = k;
```

block の中に n*m*k 個の threadが動く。

kernel 関数の中では、

blockIdx.x, blockIdx.y
blockIdx.z で指定できる

m → blockDim.x
n → blockDim.y
k → blockDim.z



配列データの加算のkernel関数

```
dim3  Dg(n/256),  Db(256);
```
← block size を 256 としている。

```
add<<< Dg, Db >>>(a_d, b_d, c_d);
```
← kernel 関数の 呼び出し実行

```
__global__  void  add
// ==========================================
(
   float    *A,       // array pointer of the global memory
   float    *B,       // array pointer of the global memory
   float    *C        // array pointer of the global memory
)
// ------------------------------------------------------------------------
{
    int    i = blockDim.x*blockIdx.x + threadIdx.x;

    C[i] = A[i] + B[i];
}
```

Built-in 変数

Device code の中で宣言せずに引用でき、
書き換え不可

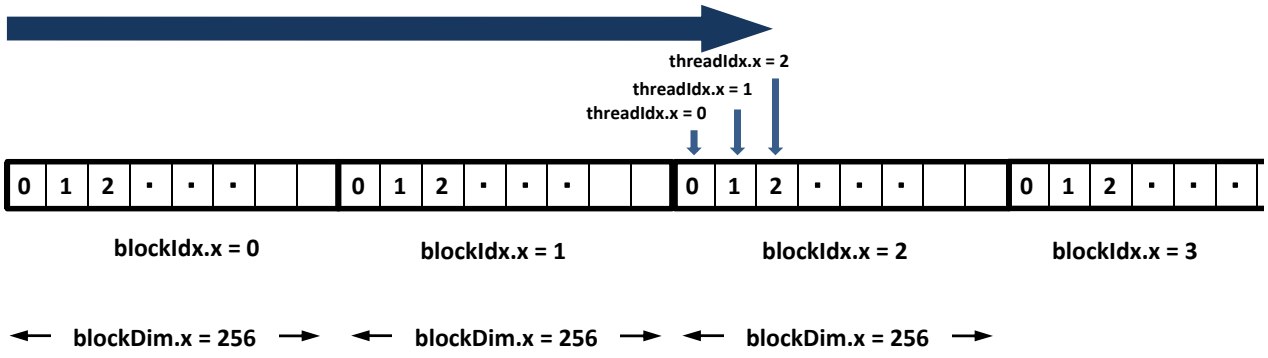
gridDim :	gridDim.x, gridDim.y, gridDim.z grid の各方向のサイズ
blockIdx :	blockIdx.x, blockIdx.y, blockIdx.z block の各方向のindex
blockDim :	blockDim.x, blockDim.y, blockDim.z block の各方向のサイズ
threadIdx :	threadIdx.x, threadIdx.y, threadIdx.z thread の各方向のindex

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

並列データ・アクセス

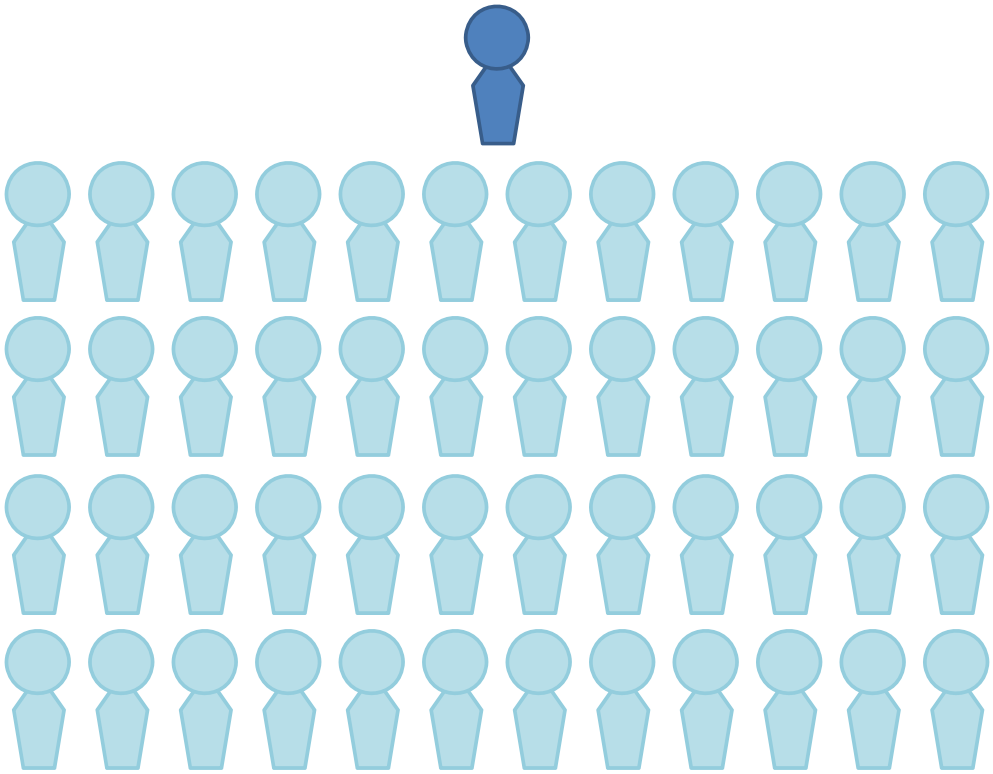
- 1次元配列データへのthreadからのアクセス
- N個のthread を発生させ、1 thread が配列の1要素にアクセス

i = blockDim.x*blockIdx.x + threadIdx.x

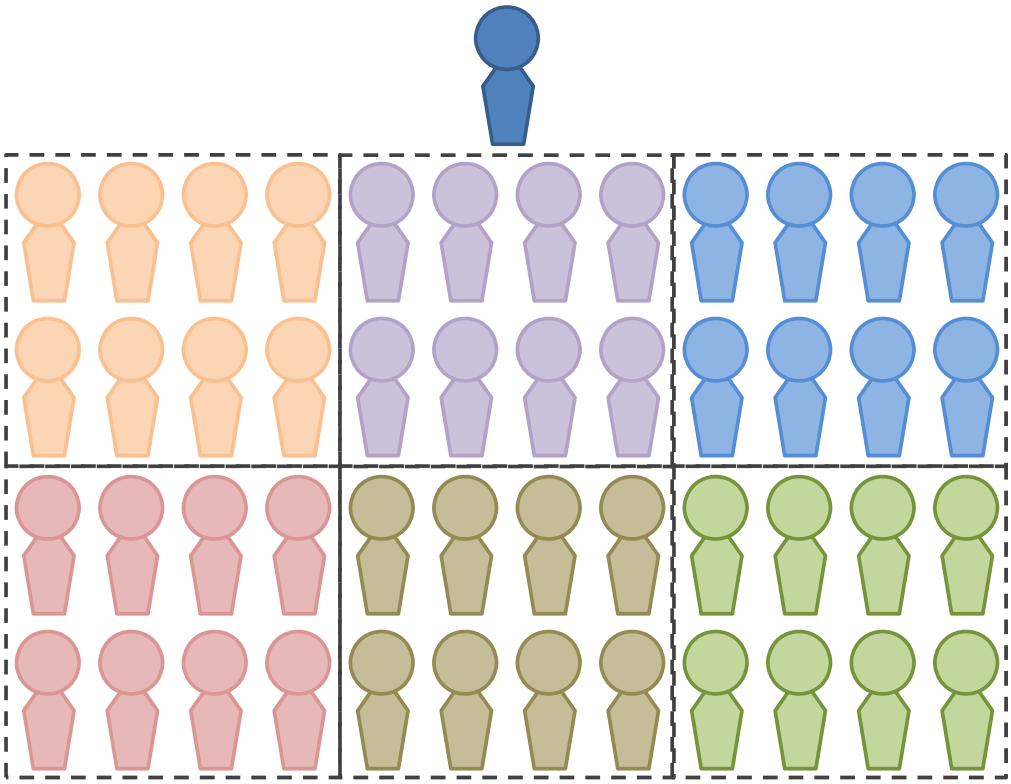


Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

クラスの班分け



クラスの班分け



アナロジー

GP GPU

host code: 先生の立場になって、問題を多くの生徒にならせる指令を出す。班分けを決める。

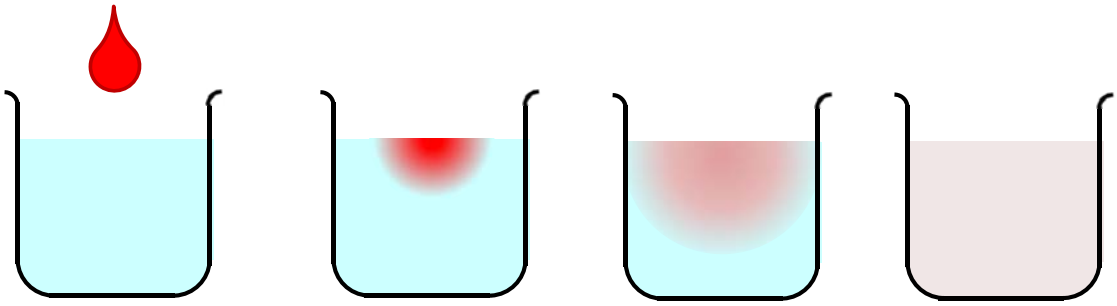
device code: 個々の生徒の立場になって何を実行するかを記述する。

クラスに何班あるか、自分は何班に所属しているか、班にはメンバーが何人いるか、自分はその班の何番目か。

拡散現象

GP GPU

コップの中の水に赤インクを落す

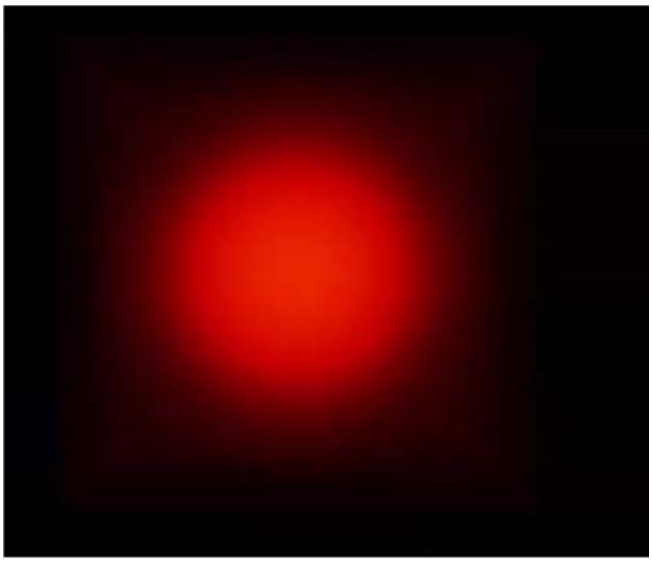


次第に拡散して赤インクは拡がって行き、最後は均一な色になる

3次元拡散方程式

$$\frac{\partial f}{\partial t} = \Delta \cdot \kappa \nabla f$$
$$= \kappa \nabla^2 f$$
$$= \kappa \left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \right)$$

κ : 拡散係数（物理定数）



典型的なステンシル計算

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

差分法による離散化

四則演算による偏微分方程式の近似計算を可能にする

$$\frac{\partial f}{\partial t} \rightarrow \frac{f_{i,j,k}^{n+1} - f_{i,j,k}^{n}}{\Delta t}$$ （時間方向の1次精度前進差分）

$$\frac{\partial^2 f}{\partial x^2} \rightarrow \frac{f_{i+1,j,k}^{n} - 2 f_{i,j,k}^{n} + f_{i-1,j,k}^{n}}{\Delta x^2}$$ （x方向の2次精度中心差分）

$$\frac{\partial^2 f}{\partial y^2} \rightarrow \frac{f_{i,j+1,k}^{n} - 2 f_{i,j,k}^{n} + f_{i,j-1,k}^{n}}{\Delta y^2}$$ （y方向の2次精度中心差分）

$$\frac{\partial^2 f}{\partial z^2} \rightarrow \frac{f_{i,j,k+1}^{n} - 2 f_{i,j,k}^{n} + f_{i,j,k-1}^{n}}{\Delta z^2}$$ （z方向の2次精度中心差分）

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

離散化式

$$f_{i,j,k}^{n+1} = f_{i,j,k}^{n} + \kappa\Delta t\left(\frac{f_{i+1,j,k}^{n} - 2f_{i,j,k}^{n} + f_{i-1,j,k}^{n}}{\Delta x^2} + \frac{f_{i,j+1,k}^{n} - 2f_{i,j,k}^{n} + f_{i,j-1,k}^{n}}{\Delta y^2} + \frac{f_{i,j,k+1}^{n} - 2f_{i,j,k}^{n} + f_{i,j,k-1}^{n}}{\Delta z^2}\right)$$

$$= c_0 f_{i,j,k}^{n} + c_1 f_{i+1,j,k}^{n} + c_2 f_{i-1,j,k}^{n} + c_3 f_{i,j+1,k}^{n} + c_4 f_{i,j-1,k}^{n} + c_5 f_{i,j,k+1}^{n} + c_6 f_{i,j,k-1}^{n}$$

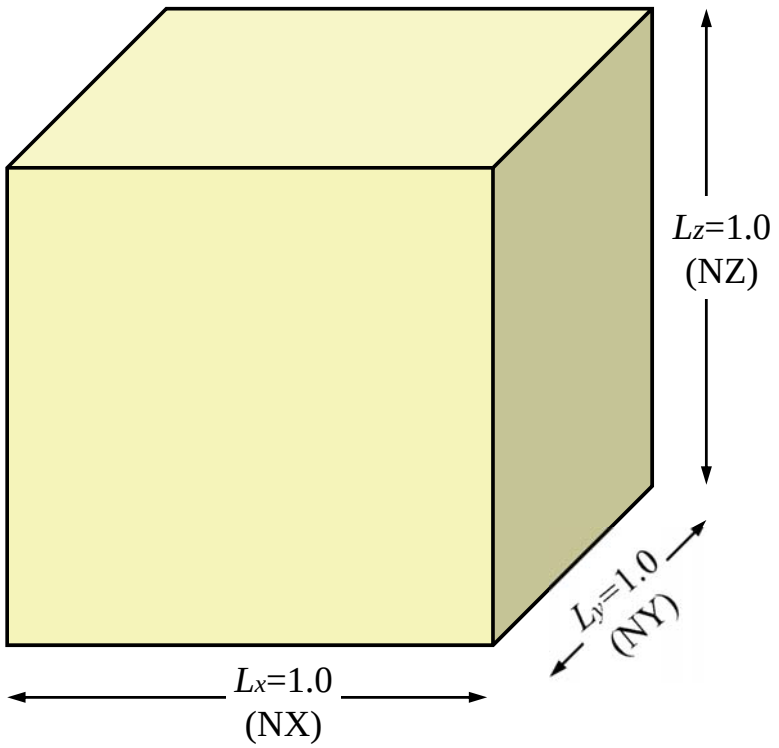
$$c_0 = 1 - \kappa\Delta 2\left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} + \frac{2}{\Delta z^2}\right) \qquad c_1 = c_2 = \frac{\kappa\Delta t}{\Delta x^2} \qquad c_3 = c_4 = \frac{\kappa\Delta t}{\Delta y^2} \qquad c_5 = c_6 = \frac{\kappa\Delta t}{\Delta z^2}$$

平均操作＝拡散方程式

$$c_0 = c_1 = c_2 = c_3 = c_4 = c_5 = c_6 = \frac{1}{7}$$

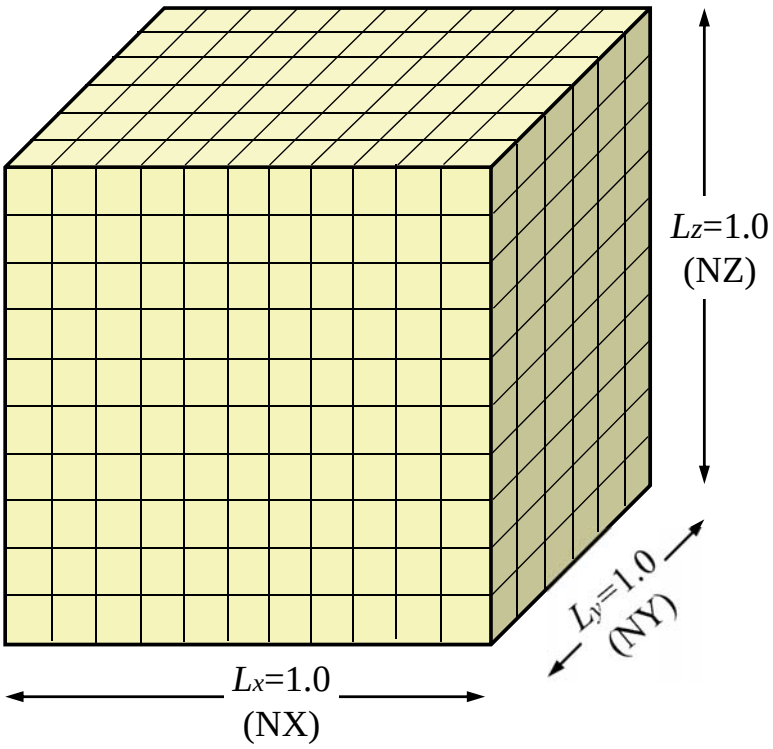
Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

計算領域



Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

計算領域



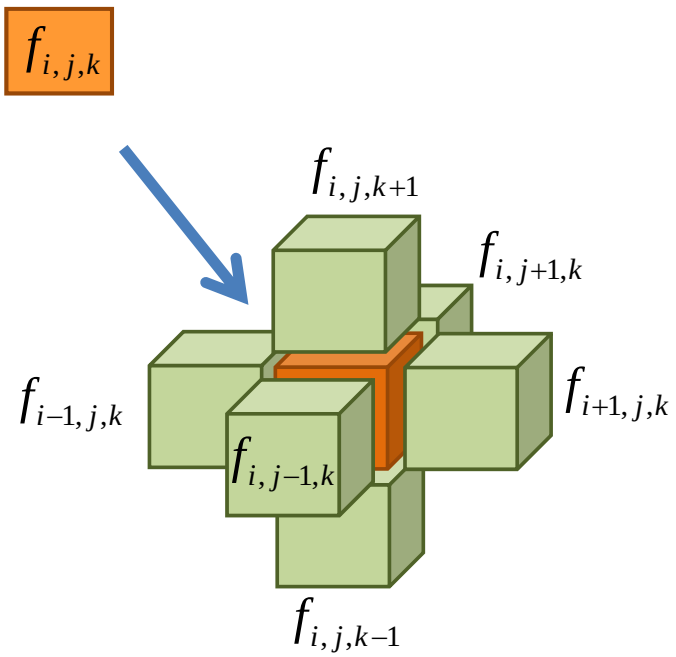
$$\Delta x = \frac{L_x}{NX}$$

$$\Delta y = \frac{L_y}{NY}$$

$$\Delta z = \frac{L_z}{NZ}$$

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

７点ステンシル



１次元配列

$$f_{i,j,k} = f[j]$$

$$f_{i+1,j,k} = f[j+1]$$

$$f_{i-1,j,k} = f[j-1]$$

$$f_{i,j+1,k} = f[j+nx]$$

$$f_{i,j-1,k} = f[j-nx]$$

$$f_{i,j,k+1} = f[j+nx*ny]$$

$$f_{i,j,k-1} = f[j-nx*ny]$$

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

時間積分

初期条件の設定

$f^{n}_{i,j,k} \rightarrow f^{n+1}_{i,j,k}$ の拡散方程式の計算

$f^{n}_{i,j,k} \leftrightarrow f^{n+1}_{i,j,k}$ のPointer 交換

$time \mathrel{+}= \Delta t$

出　力

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

初期条件と境界条件

初期条件： Cosine Bell プロファイル

$$f(x,y,z) = (1-\cos(k_x x))(1-\cos(k_y y))(1-\cos(k_z z))$$

境界条件： ノイマン境界

$$x = 0,\quad L_x \qquad \frac{\partial f}{\partial x} = 0$$ （x 方向の壁から物質が漏れない条件）

$$y = 0,\quad L_y \qquad \frac{\partial f}{\partial y} = 0$$ （y 方向の壁から物質が漏れない条件）

$$z = 0,\quad L_z \qquad \frac{\partial f}{\partial z} = 0$$ （z 方向の壁から物質が漏れない条件）

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

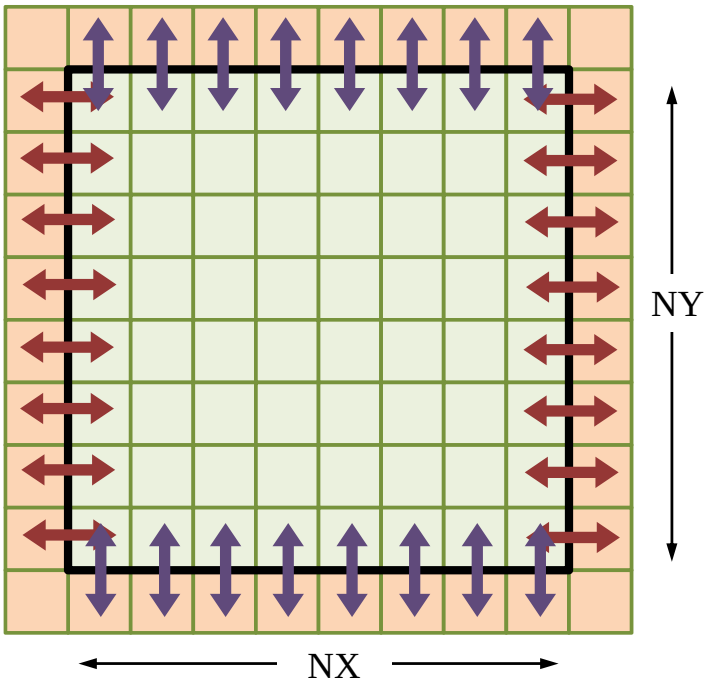
数値計算における境界条件

境界の外側の格子点を同じ値とする： ノイマン境界条件

$$x = 0: \quad \frac{f_{0,j,k} - f_{-1,j,k}}{\Delta x} = 0$$

$$y = 0: \quad \frac{f_{i,0,k} - f_{i,-1,k}}{\Delta y} = 0$$

$$z = 0: \quad \frac{f_{i,j,0} - f_{i,j,-1}}{\Delta z} = 0$$

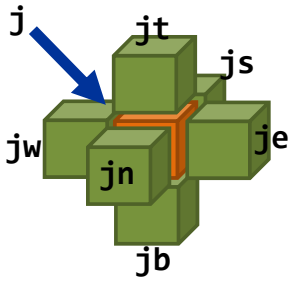


Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

CPUの拡散方程式プログラム

```
for(jz=0; jz < nz; jz++) {
    for(jy=0; jy < ny; jy++) {
        for(jx=0; jx < nx; jx++) {
            j = nx*ny*jz + nx*jy + jx;
            je = j+1; jw = j-1;  jn = j+nx; js =j-nx;
            jt = j + nx*ny;  jb = j - nx*ny;

            if (jx == nx-1) je = j;
            if (jx == 0   ) jw = j;
            if (jy == ny-1) jn = j;
            if (jy == 0   ) js = j;
            if (jz == nz-1) jt = j;
            if (jz == 0   ) jb = j;

            fn[j] = c0*f[j]
                  + c1*f[je] + c2*f[jw]
                  + c3*f[jn] + c4*f[js]
                  + c5*f[jt] + c6*f[jb];
        }
    }
}
```

境界条件



Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

GPUカーネル関数

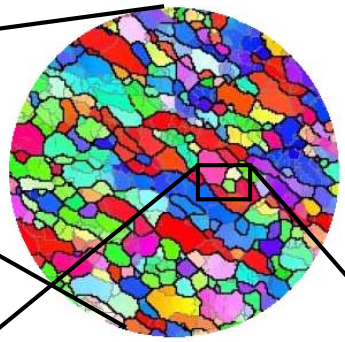
```
__global__  void  gpu_diffusion3d(double *f, double *fn,
int nx, int ny, int nz, double c0, double c1, double c2, double
c3, double c4, double c5, double c6)
{
    int jx = ........ ;  /* x方向のindex計算をする */
    int jy = ........ ;  /* y方向のindex計算をする */
    int jz = ........ ;  /* z方向のindex計算をする */

    int j,  je,  jw,  jn,  js,  jt,  jb;

                /* この部分を完成させる */

                                  レポート課題
}
```

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

Development New Materials

Mechanical Structure

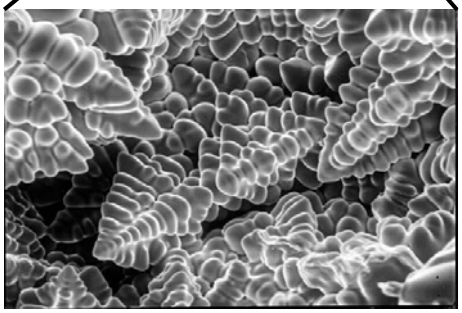
Material Microstructure



Low-carbon society

Improvement of fuel efficiency by reducing the weight of transportation

Developing lightweight strengthening material by controlling microstructure

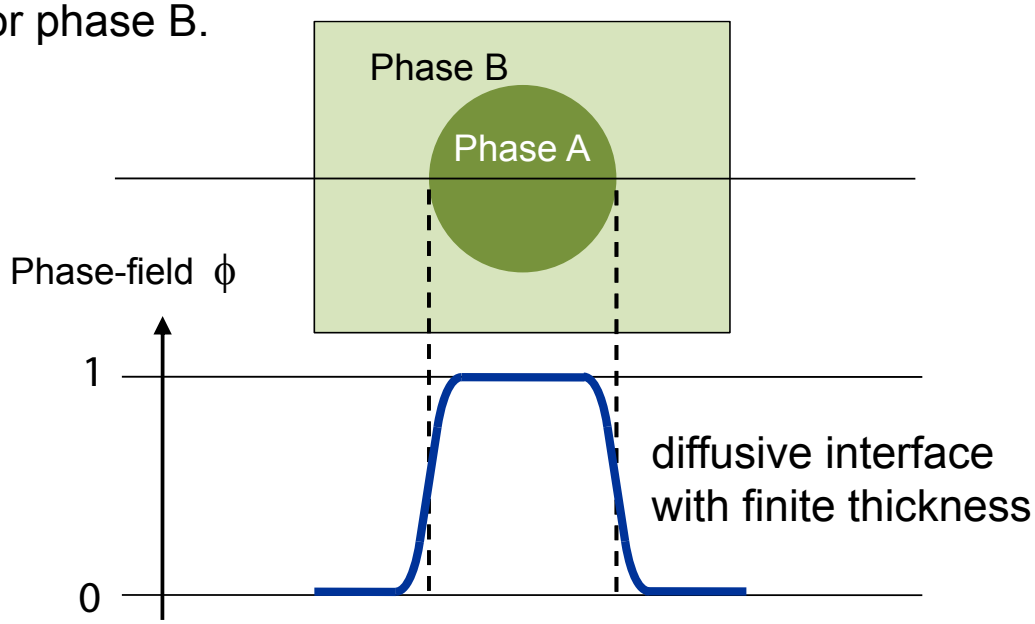


Dendritic Growth

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

Phase-Field Model

The phase-field model is derived from non-equilibrium statistical physics and f = 0 represents the phase A and f = 1 for phase B.

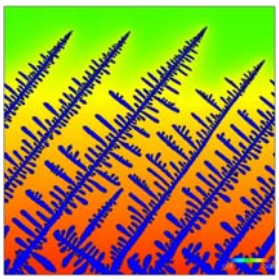


Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

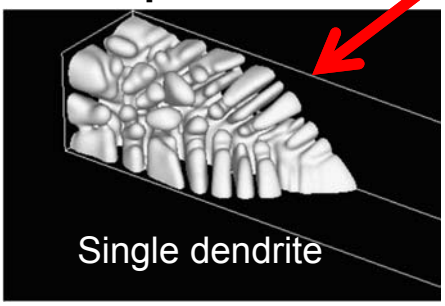
Requirement for Peta-scale Computing

Previous works

2D computation

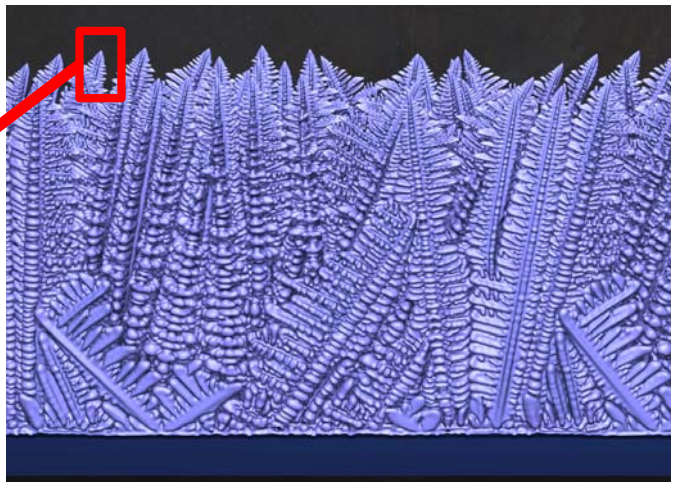


3D computation



×1000 large-scale computation

on TSUBAME 2.0



~ mm scale

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

Al-Si: Binary Alloy

Time evolution of the phase-field ϕ
（Allen-Cahn equation)

$$\frac{\partial \phi}{\partial t} = M_\phi \left[\nabla \cdot (a^2 \nabla \phi) + \frac{\partial}{\partial x}\left(a \frac{\partial a}{\partial \phi_x} |\nabla \phi|^2 \right) + \frac{\partial}{\partial y}\left(a \frac{\partial a}{\partial \phi_y} |\nabla \phi|^2 \right) + \frac{\partial}{\partial z}\left(a \frac{\partial a}{\partial \phi_z} |\nabla \phi|^2 \right) - \Delta S \Delta T \frac{dp(\phi))}{d\phi} - W \frac{dq(\phi)}{d\phi} \right]$$

Time evolution of the condensation: c

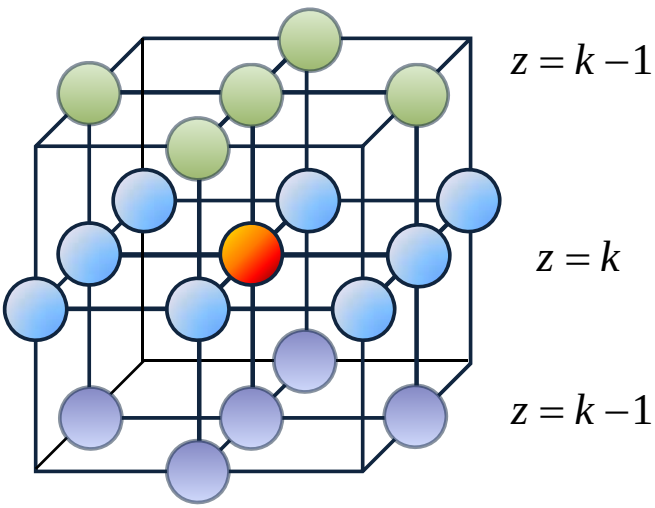
$$\frac{\partial c}{\partial t} = \nabla \cdot [D_S \phi \nabla c_S + D_L (1-\phi) \nabla c_L]$$

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

Finite Difference Method

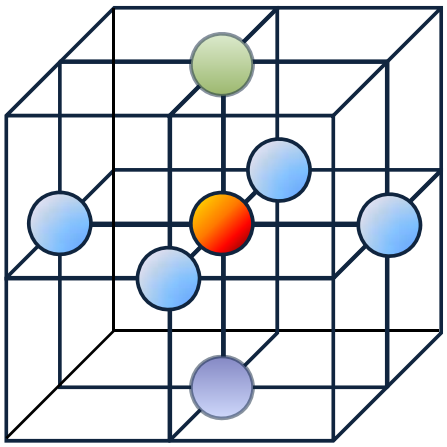
Phase Field : ϕ

19 points to solve $\phi_{i,j,k}$



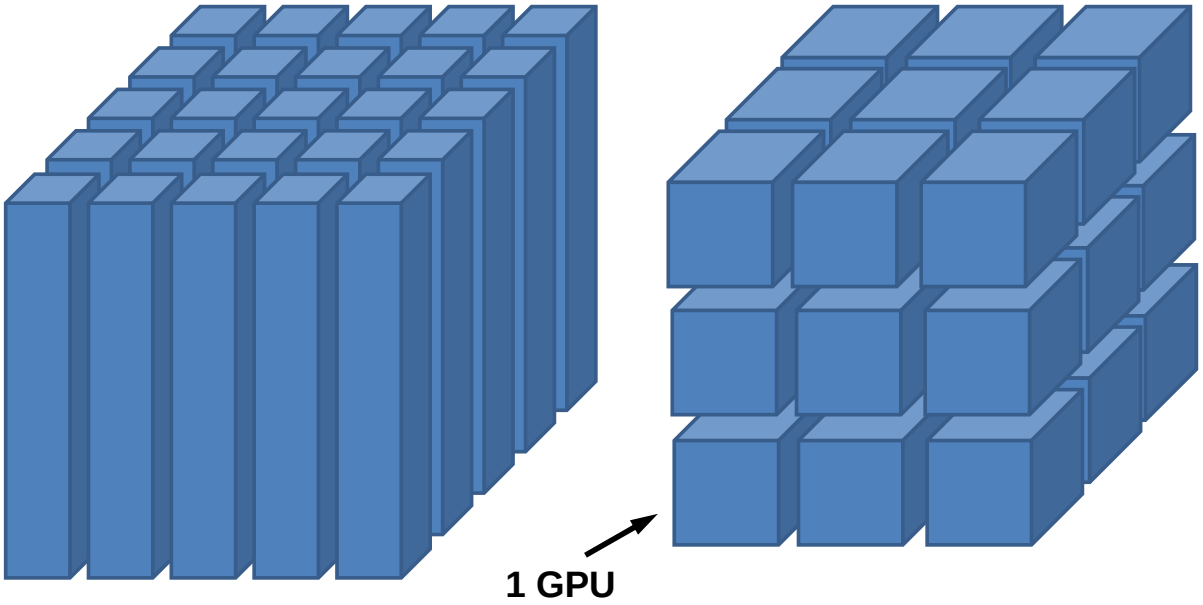
Condensation : c

7 points to solve $c_{i,j,k}$

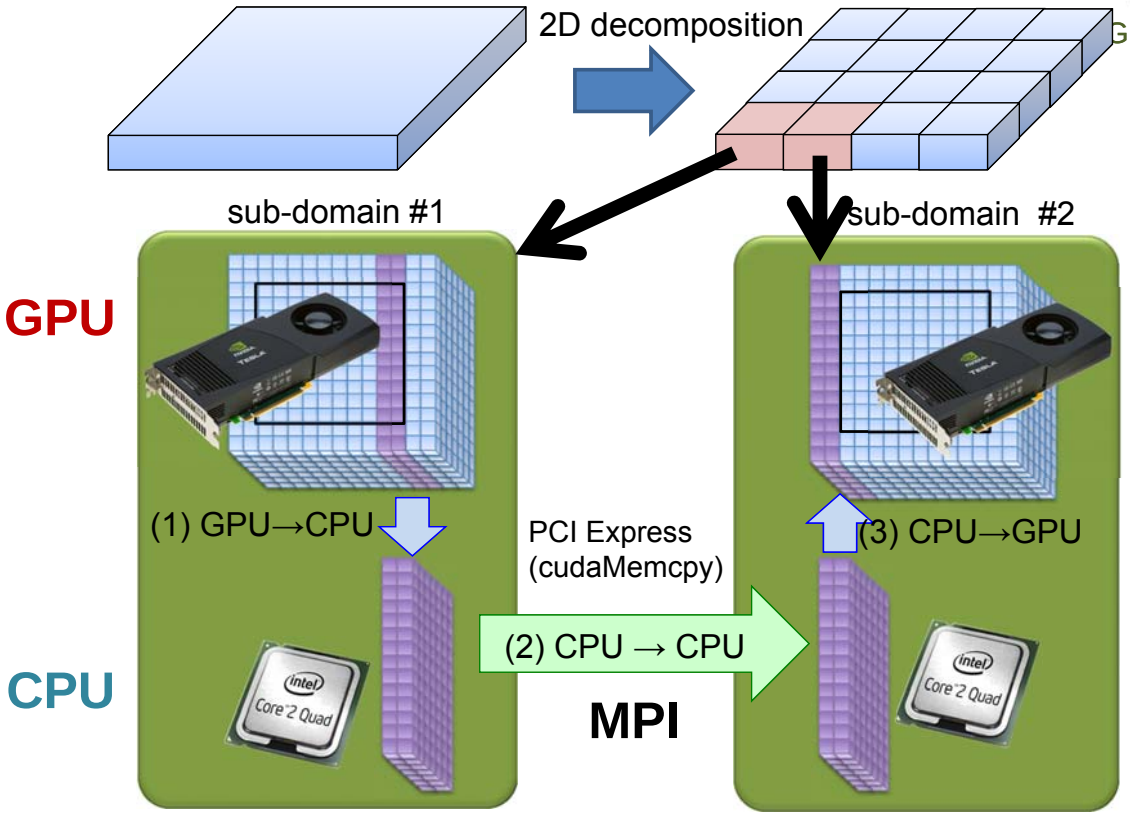


Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

Multi-dimensional Domain Decomposition

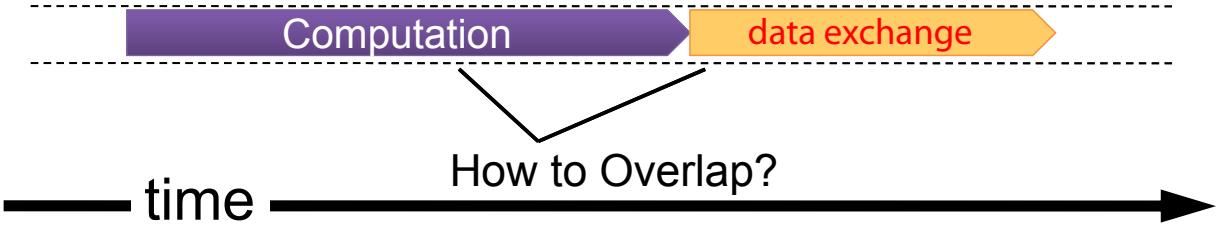


Multi-GPU : Domain decomposition



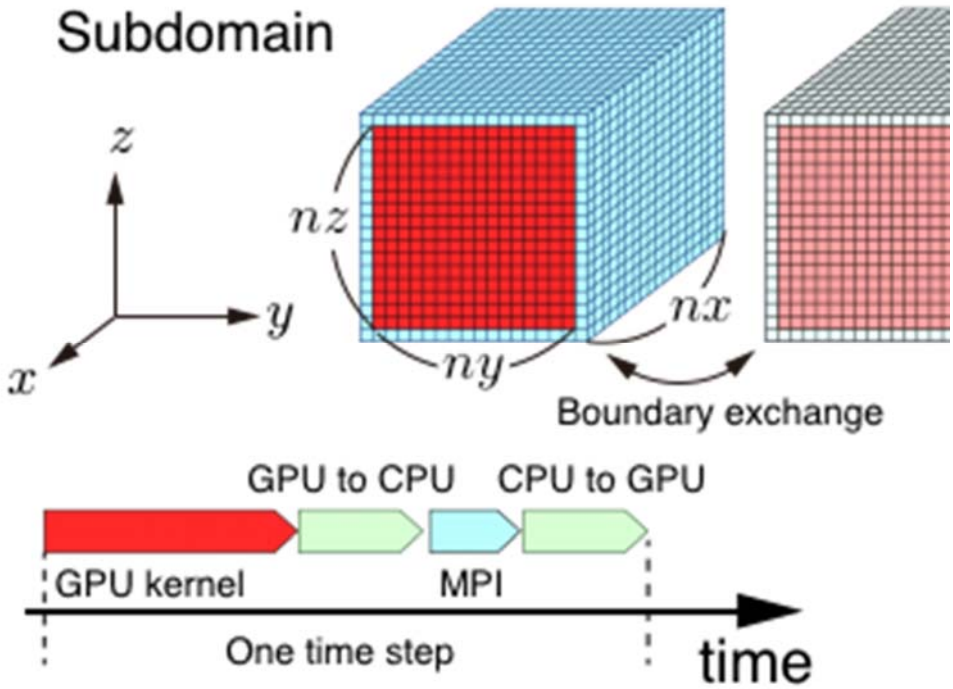
Multi-GPU Optimization

Typical explicit time integration

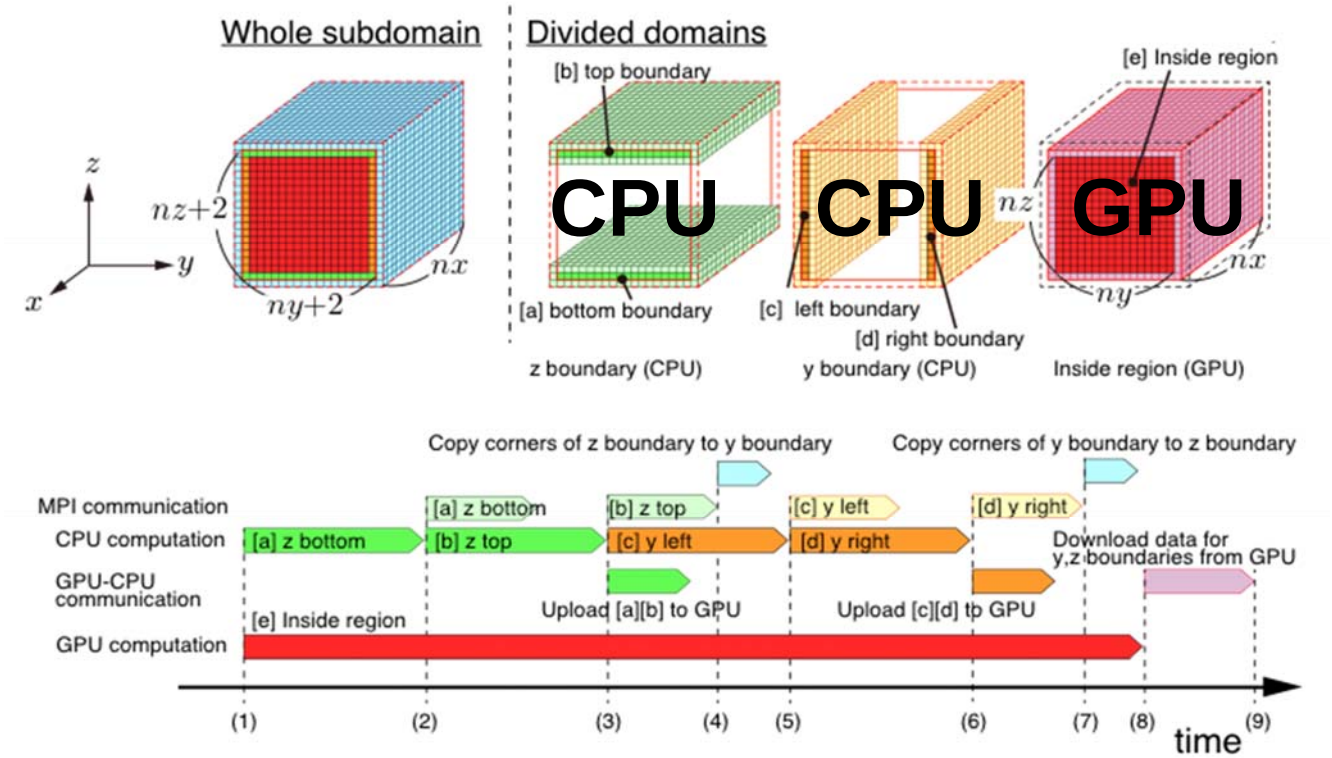


1. **GPU-only Method (without overlapping)**
2. **Hybrid-YZ Method**
3. **Hybrid-Y Method**

1. GPU-only method

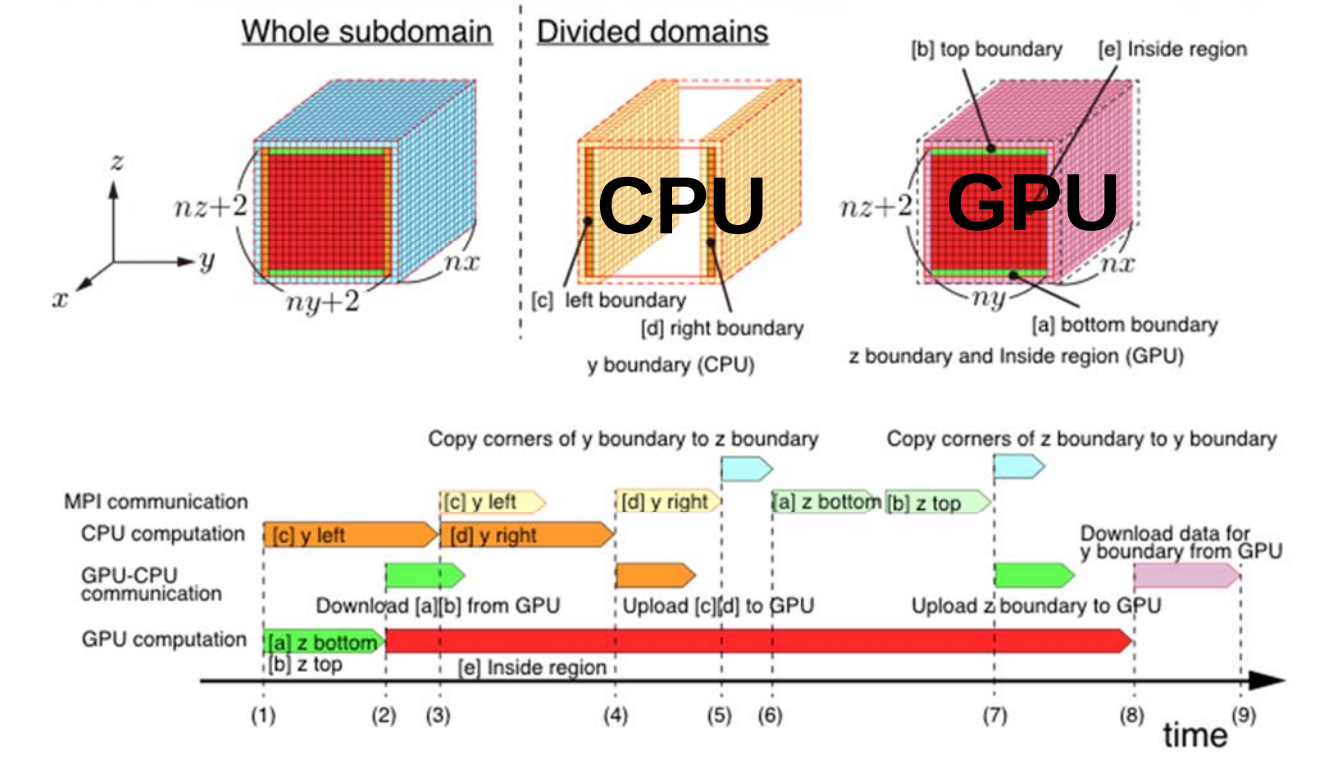


2. Hybrid-YZ method



Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

3. Hybrid-Y method

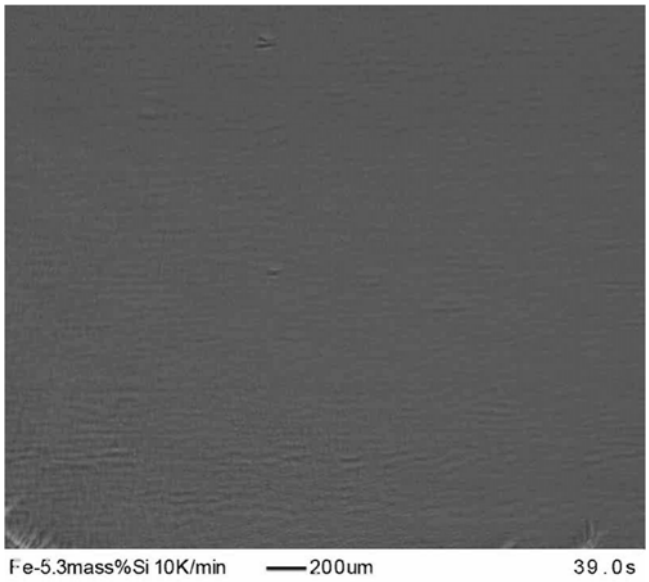


Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

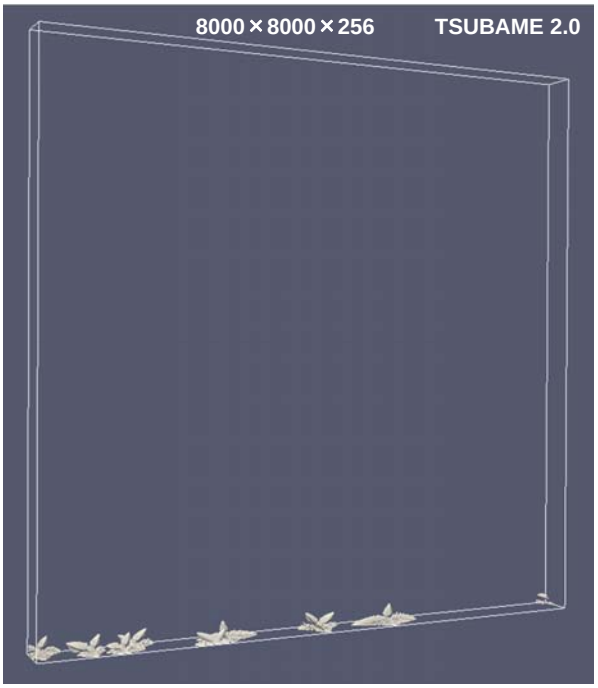
Comparison with Experiment

GP GPU

Observation:
X-ray imaging of Solidification of a binary alloy at Spring-8 in Japan by Prof. Yasuda (Osaka University in Japan)

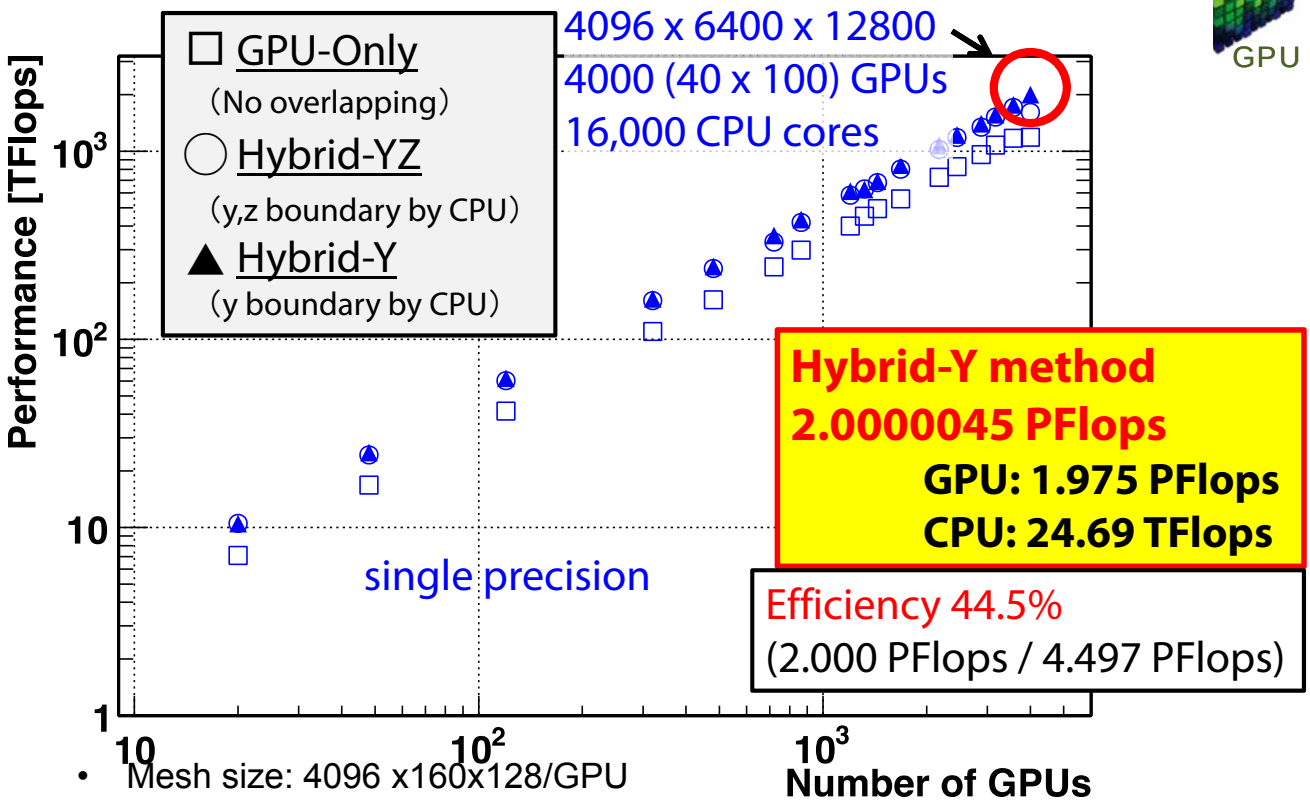


Phase-field simulation



Weak scaling

GPU



- Mesh size: 4096 x160x128/GPU
- NVIDIA Tesla M2050 card / Intel Xeon X5670 2.93 GHz on TSUBAME 2.0