GPUによる圧縮性・非圧縮性流体計算の驚異的な高速化

東京工業大学　学術国際情報センター

青木　尊之

GPGPU

(General-Purpose Graphics Processing Unit)

GPU を画像処理以外の目的で使う

GPUの魅力

- 高性能：ハイエンド GPU はピーク 2 TFlops 超
- 低価格：ハイエンドでも 数万円（4万円）
- 手軽さ：事務所のPCにも装着できる

単一GPU
高消費電力 ➡ 低消費電力: Flops／W

NVIDIA GPU

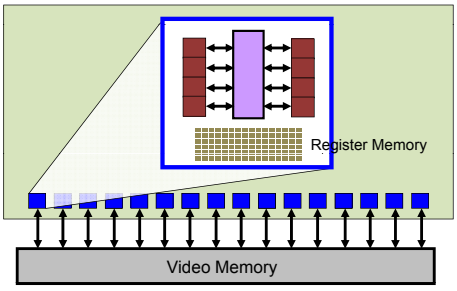
* 2 instruction issue

		GeForce GTX 285	Tesla S1070
GPU	Peak Performance [GFlops]	708*, 1063	691* x4, 1036 x4
	# of SP	240	240 x4
	SP Clock [MHz]	1476	1440
Video Memory	Transfer Rate[GB/s]	159	102
	Memory Interface [bit]	512	512
	Memory Data Clock [MHz]	2484 (GDDR3)	1613 (GDDR3)
	Capacity [GB]	1024	4096 x4

Peak Power : 183W

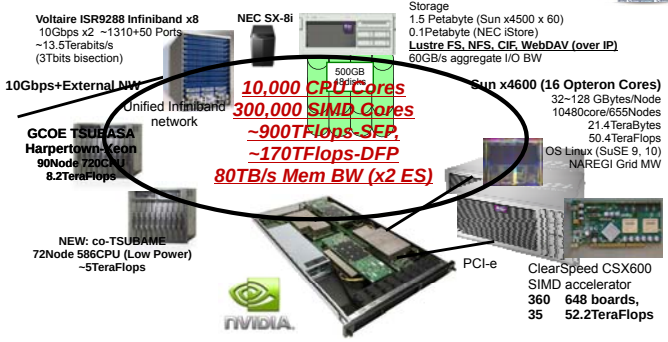
Peak Power : 700W

GPU アーキテクチャ



Global memory	~4GB (VRAM
Multiprocessor	~30 (GTX280(GT200))
Shared memory	16 Kbyte
Streaming Processor	1つのMultiprocessorに8個で合計240個

東工大　TSUBAME 1.2



Nvidia Tesla S1070: 170台, 総計 680カード
High Performance in Many BW-Intensive Apps
10% power increase over TSUBAME 1.0 (130TF SFP / 80TF DFP)

HPCにおけるGPU Computing

粒子計算: 重力多体問題（天体物理）
分子動力学計算
個別要素法計算・SPH

メモリ・アクセスに対して計算負荷が高い

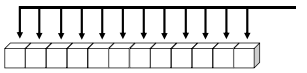
FFT: CUFFT
Nukada FFT（東工大）

メモリ・アクセスが多い

格子計算: 最近、成功例が報告され始めている

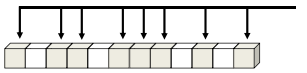
メモリ・アクセスに対して計算負荷が低い
メモリ・アクセスがスパース

HPCアプリのメモリアクセス



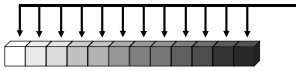
連続データアクセス
（差分系格子計算）

○



不連続（ランダム）アクセス
（有限要素法）

△



隣接データ依存

A[i] = A[i-1] + A[i-2]*C;

×

GPUの得意・不得意

× シリアルなプロセス， 色々なことをたくさんさせる

○ 超並列計算 （同じことをたくさんさせる）

- スレッド計算 （数1000～数10000スレッド）
- 限られた共有メモリを介して以外、異なるスレッドのデータは見えない
- プログラミング・テクニックが必要 （簡単に最大値も計算できない）

通常のCPUによるプログラム

```
float *a = malloc(…);
for(i = 0; i < n; i++){
    a[i] = . . .
}
```

CUDAプログラム

```
cudaMalloc((void**) a,… );
i = blockDim.x*blockIdx.y
  + threadIdx.x;
    a[i] = . . .
```


HPCアプリケーションにおける GPUの利用

FULL GPU計算 **加速率 10倍～100倍**

※ 適用できる計算が限られる
※ カード上のメモリ量に制限

一部 GPU計算 **加速率 数10 %～3倍**

※ Hot spot のみ GPU処理
※ CPU側のメモリとのデータ交換に時間がかかる

アプリケーション事例：CFD

圧縮性流体解析

超音速流れなど、限られた分野でのみ必要
陽解法による高精度数値計算手法

T. Aoki, Comp. Phys. Comm., Vol.102, No.1-3, 132-146 (1997)
Y. Imai, T. Aoki and K. Takizawa, J. Comp. Phys., Vol. 227, Issue 4, 2263-2285 (2008)
K. Kato, T. Aoki, et. al., Int. J. Numerical Methods in Fluids, Vol.51, 1335-1353 (2006)
Y. Imai, T. Aoki, J. Comp. Phys., Vol.217, 453-472 (2006)
Y. Imai, T. Aoki, J. Comp. Phys., Vol.215, 81-97 (2006)

非圧縮性流体解析

日常生活レベルで起こる殆どの現象は非圧縮性流体で非常によく近似できる

Semi-implicit 時間積分が必要→ Poisson Solver

・乱流， 多相流， 化学反応， 音響

Rayleigh-Taylor Instability

Y. Imai, T. Aoki and K. Takizawa, J. Comp. Phys., Vol. 227, Issue 4, 2263-2285 (2008)

IDO-CF Scheme

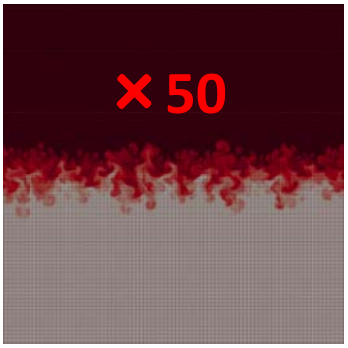
Euler equation:

$$\frac{\partial \boldsymbol{Q}}{\partial t}+\frac{\partial \boldsymbol{E}}{\partial x}+\frac{\partial \boldsymbol{F}}{\partial y}=0$$

$$\boldsymbol{Q}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ e\end{bmatrix}\quad \boldsymbol{E}=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ eu+pu\end{bmatrix}\quad \boldsymbol{F}=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ ev+pv\end{bmatrix}$$

Heavy fluid lays on light fluid and unstable.

512 x 512



CFDアプリ（非圧縮性流体解析）

非圧縮性ナヴィエ・ストークス方程式

$$\nabla\cdot\boldsymbol{u}=0$$

$$\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\nabla\boldsymbol{u}=-\frac{1}{\rho}\nabla p+\nu\Delta\boldsymbol{u}$$

Poisson 方程式

$$\Delta p^{n+1}=\frac{\nabla\cdot\boldsymbol{u}^{n+1}}{\Delta t}$$

- ■ 移流項計算： 高次精度差分法（GPU向）
- ■ 拡散項計算： 2次中心差分法（易）
- ■ 速度発散計算： スタッガード格子差分（易）
- ■ Poisson 方程式計算： Red & Black MG法（難）
- ■ 速度勾配計算： スタッガード格子差分（易）

Poisson Equation solved by MG(Multi Grid), Red & Black method

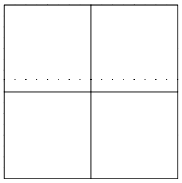
■ Algorithm Acceleration

Point Jacobi ×4～5→ SOR ×100→ MG-SOR

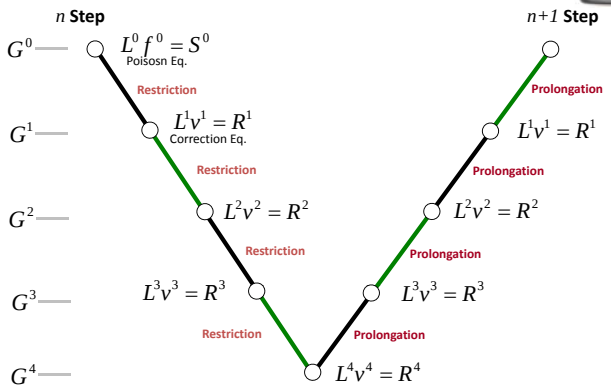
■ Hardware Acceleration :

GPU (CUDA) ×50?

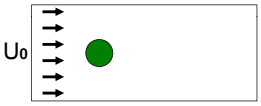
$$\frac{f_{i+1,j}-2f_{i,j}+f_{i-1,j}}{\Delta x^2}+\frac{f_{i,j+1}-2f_{i,j}+f_{i,j-1}}{\Delta y^2}=\rho_{i,j}$$



V-Cycle MG



Aerodynamics Study



円柱後方にカルマン渦列：
レイノルズ数に対する流れの変化：

$Re = 2000$

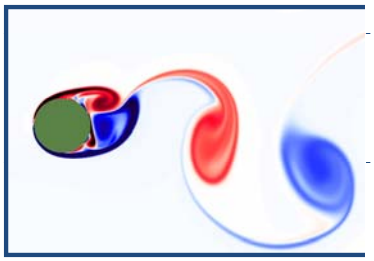
Navior Stokes Eq.

Poisson Equation

$$\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}=\frac{1}{\Delta t}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)$$

Correction

$$\frac{\partial u}{\partial t}=-\frac{1}{\rho}\frac{\partial p}{\partial x}\qquad \frac{\partial v}{\partial t}=-\frac{1}{\rho}\frac{\partial p}{\partial y}$$



CUDA 概要

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

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

■ 2006年11月発表, 現在v2.3
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. Brook+(AMD), OpenCL, RapidMind, etc.

CUDA Library

CUBLAS Library

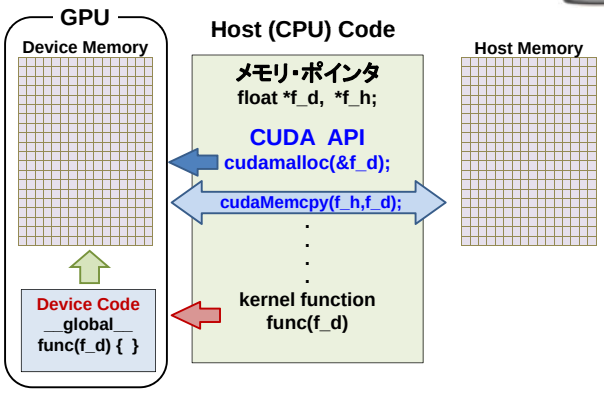
CULA Library (Lapack)

CUFFT Library

CUDPP (Data Parallel Primitives)
http://www.gpgpu.org/developer/cudpp/
Reduction, Scan, Sort, . . .

MATLAB

CUDAのプログラム概念図



GPU kernel-function call

Host Code の中で call する。

kernel_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

Dg, Db で指定される数の Thread が実行される。

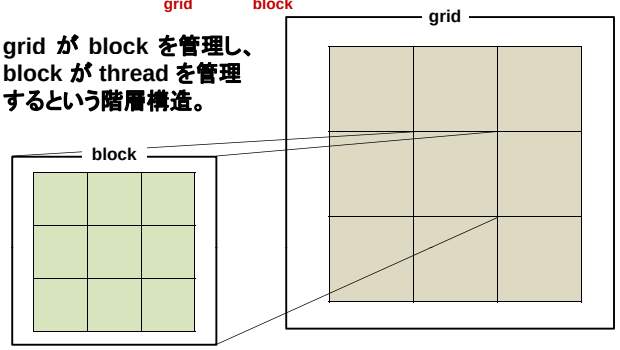
引数にHost memory のpointerの指定は不可。

kernel function の実行は、CPU に対して絶えず非同期。

Threadの管理

kernel 関数<<< 第1引数, 第2引数>>>で指定

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

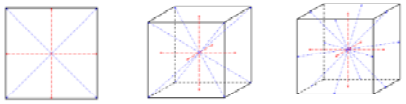


マルチGPU 計算

Lattice Boltzmann Method

Discretizing Boltzmann-BGK equation in the momentum space using a finite set of velocities $\{\mathbf{e}_i\}$

D2Q9
D3Q15
D3Q19…



$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \nabla f_i = -\frac{1}{\lambda}\left(f_i - f_i^{eq}\right)$$

$$f_i^{eq} = \rho w_i \left[1 + \frac{3}{c^2}(\mathbf{e}_i \cdot \mathbf{u}) + \frac{9}{2c^4}(\mathbf{e}_i \cdot \mathbf{u})^2 - \frac{3}{2c^2}(\mathbf{u} \cdot \mathbf{u})\right]$$

i is the value in the direction of *ith* discrete velocity
$\mathbf{e}_i$ is the discrete velocity set;
w_i is the weighting factor
c is the particle velocity
$\mathbf{u}$ is the macroscopic velocity

Lattice Boltzmann Method

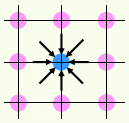
Further discretization in physical space x, and time t

$$f_i(\mathbf{x} + \mathbf{e}_i\delta t, t + \delta t) - f_i(\mathbf{x}, t) = -\frac{1}{\tau}\left(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)\right) \qquad \tau = \lambda / \delta t$$

LBM code

Collision step:

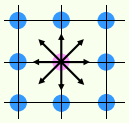
$$\bar{f}_i(\mathbf{x}, t) = f_i(\mathbf{x}, t) - \frac{1}{\tau}\left(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)\right)$$



Purely local !!

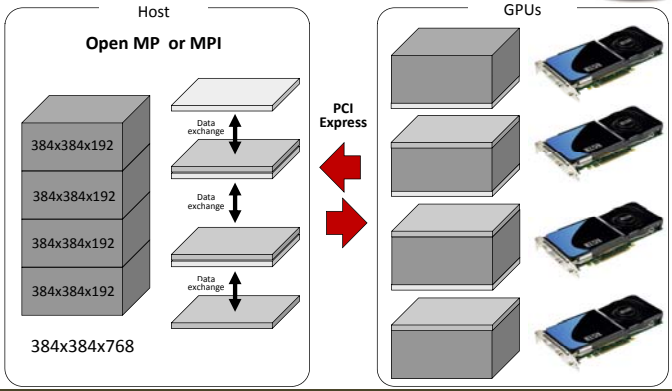
Streaming step:

$$f_i(\mathbf{x} + \mathbf{e}_i\delta t, t + \delta t) = \bar{f}_i(\mathbf{x}, t)$$

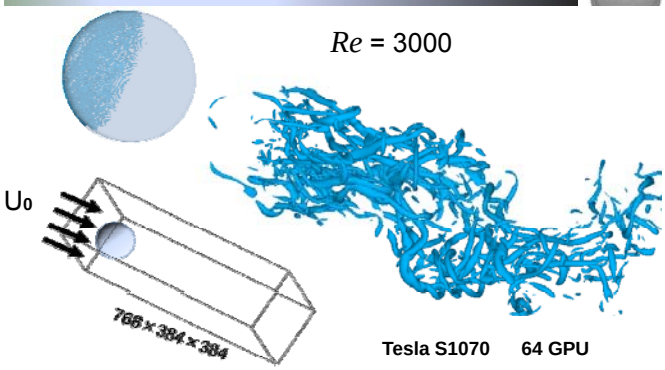


Uniform data shifting
Little computational effort !!

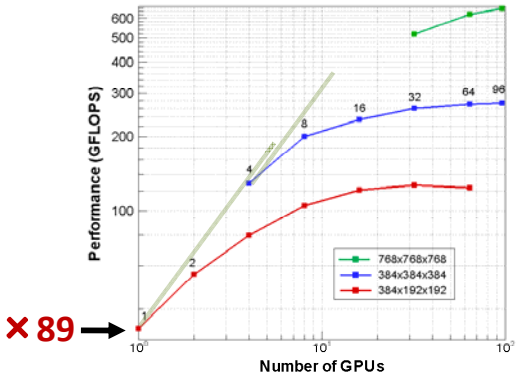
GPU間データ通信



Incompressible Flow



Multi-GPU Scalability



Real-time Tsunami Simulation

ADPC : Asian Disaster Preparedness Center

Early Warning System:

Data Base → Real-time CFD

high accuracy

Shallow-Water Eq.

Conservative Form:

$$\frac{\partial h}{\partial t}+\frac{\partial hu}{\partial x}+\frac{\partial hv}{\partial y}=0$$

Assuming hydrostatic balance in the vertical direction,

$$\frac{\partial hu}{\partial t}+\frac{\partial}{\partial x}\left(hu^2+\frac{1}{2}gh^2\right)+\frac{\partial huv}{\partial y}=-gh\frac{\partial z}{\partial x}$$

3D → 2D equation

$$\frac{\partial hv}{\partial t}+\frac{\partial huv}{\partial x}+\frac{\partial}{\partial y}\left(hv^2+\frac{1}{2}gh^2\right)=-gh\frac{\partial z}{\partial y}$$

Numerical Methods

Directional Splitting

First Step: x-directional computation

$$\frac{\partial h}{\partial t}+\frac{\partial hu}{\partial x}=0 \qquad \frac{\partial hv}{\partial t}+\frac{\partial uhv}{\partial x}=0 \qquad \frac{\partial hu}{\partial t}+\frac{\partial}{\partial x}\left(hu^2+\frac{1}{2}gh^2\right)=-gh\frac{\partial z}{\partial x}$$

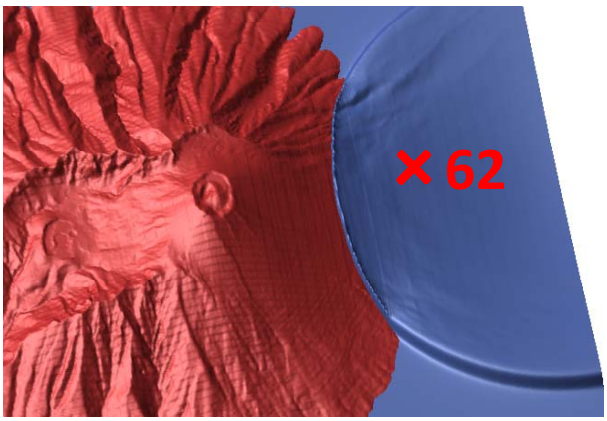
Second Step: y-directional computation

$$\frac{\partial h}{\partial t}+\frac{\partial hv}{\partial y}=0 \qquad \frac{\partial hu}{\partial t}+\frac{\partial vhu}{\partial y}=0 \qquad \frac{\partial hv}{\partial t}+\frac{\partial}{\partial y}\left(hv^2+\frac{1}{2}gh^2\right)=-gh\frac{\partial z}{\partial y}$$

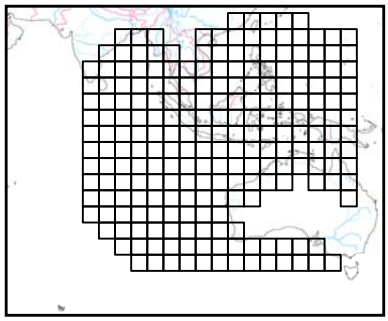
MOC (Method of Characteristics),

Conservative Semi-Lagrangian Method,

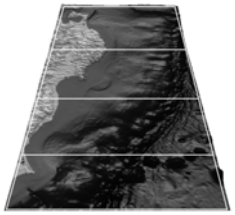
Thin Water run-up to dry area.



大規模リアルタイム津波シミュレータの構築



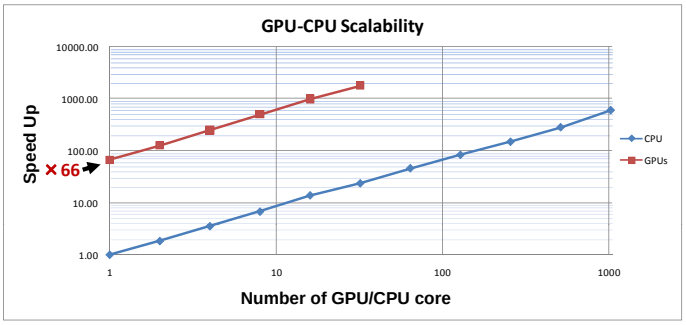
241 GPU 5000km×5000km(500m格子)



8 GPU 400km×800km (100m格子)

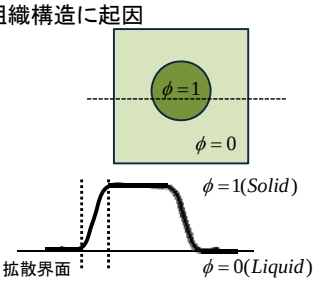
CPU-GPU Comparison

(1 CPU Core based)



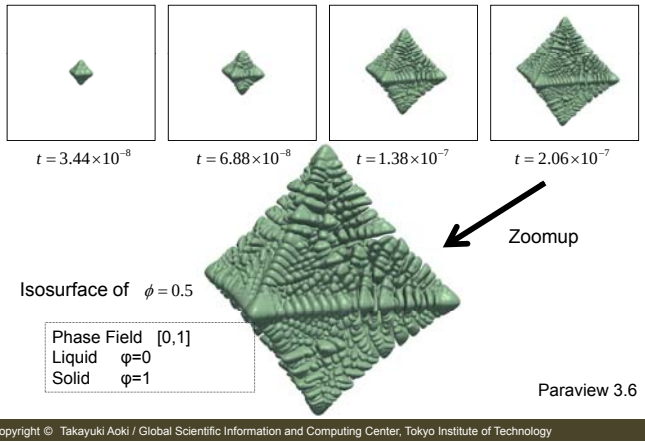
Phase Field Model による
純金属凝固 デンデライト成長

- 高性能な材料（鉄鋼材料等）の開発
  →力学的特性は材料のミクロの組織構造に起因
  →組織構造形成過程の予測
- Phase Fieldモデル
  →純金属の凝固シミュレーション
  →計算コスト（大）
  大規模計算
  新たな知見への期待

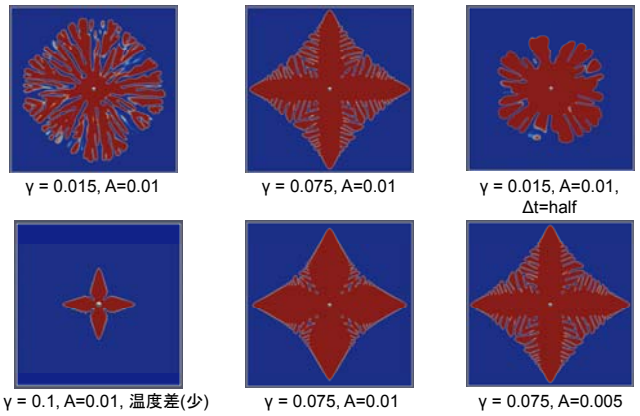


Phase Fieldシミュレーションに対する
GPUコンピューティングによる高速化

結晶凝固成長 (1024x1024x1024)



パラメータの違いによる結果



マルチGPU計算の克服 すべき問題

GPUの計算が速いため、GPU ⟷ CPU のデータ交換は非常に性能低下をもたらす可能性がある。

GPU内での計算と CPU との転送時間とのバランスが重要なポイントになる。GPU搭載の複数ノードのクラスター（スパコン）では、ノード間インターコネクションの性能にも注意する必要がある。

※ PCI Express Gen 3.0 に期待。

マルチGPU計算の克服 すべき問題

GPUの計算が速いため、GPU ⟷ CPU のデータ交換は非常に性能低下をもたらす可能性がある。

GPU内での計算と CPU との転送時間とのバランスが重要なポイントになる。GPU搭載の複数ノードのクラスター（スパコン）では、ノード間インターコネクションの性能にも注意する必要がある。

※ PCI Express Gen 3.0 に期待。

SUMMARY

■FULL GPUの流体計算が可能
■スパコンに匹敵する性能がある
■格子系の流体計算は比較的容易である
（圧縮性流体計算にはさらに適している）
■GPU の得意・不得意を理解してCUDAプログラミング
を行う必要がある
■倍精度版GPUを使う場合でも単精度計算は有効
■Shared メモリの利用が Key Point
■3次元計算、高次精度計算がGPUの演算性能を引き出す
■マルチGPU（複数カード・複数ノード）が今後の課題