CUDA/GPU による水柱崩壊シミュレーションの高速化

著者　井上　智寛　　指導教員　下條　雅史

1. はじめに

物理のシミュレーションの１つに流体のシミュレーションがある。中でも粒子法による流体のシミュレーションでは形状の大幅な変形、分裂などの現象が再現できる。しかし、よりリアルなシミュレーションを求めて粒子数を増やすと膨大な量の演算が必要となり、多大な時間がかかることになる。そこで本研究では **CUDA(Compute Unified Device Architecture)**という C 言語の統合環境を用い、水柱の崩壊のシミュレーションの並列処理化を行い、処理性能の向上を目的とする。

2. 概要

2.1 CUDA(Compute Unified Device Architecture)

CUDA は NVIDIA 社製 GPU 向けのプログラミング環境であり、C/C++言語を元に独自の拡張を行った専用の言語および対応するコンパイラといくつかの数値計算ライブラリから構成されている。

2.1.1 グリッド、ブロック、スレッド

CUDA ではスレッドを主体として並列計算が行われる。複数のスレッドの集まりをブロック、複数のブロックの集まりをグリッドとして扱い、ブロックごとに GPU 内の演算部が搭載されており、その内部の CUDA コアがスレッドを処理する。

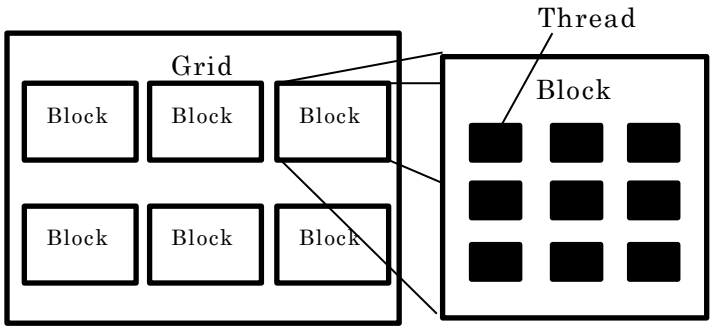

図 1　グリッド、ブロック、スレッドの構造

2.1.2 CUDA プログラミングモデル

CUDA では、CPU 側をホストとよび、GPU 側をデバイスと呼ぶ。ホスト側でのプログラムは、ホスト上で動作し、ホスト上のメモリを利用する。デバイス側でのプログラムをカーネルプログラムと言う。デバイス上で動作し、GPU が処理を行い、デバイスメモリを利用する。

CUDA を用いてプログラミングをしたときのプログラム全体の流れ図を図 2 に示す。

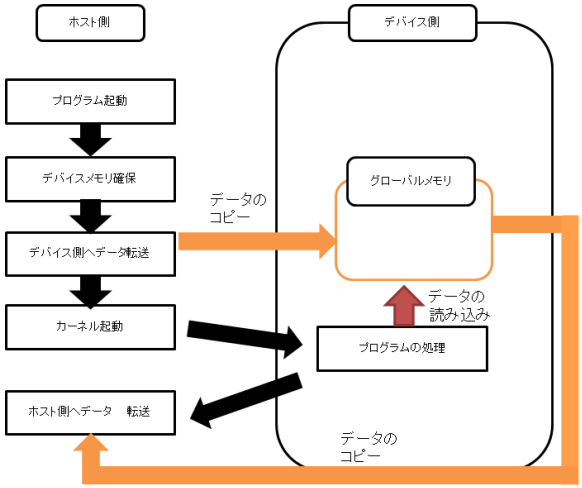

図 2 プログラムの流れ

2.1.3 CUDA スレッドの同期

GPU コンピューティングでは CUDA コアの数よりも多くのスレッドが動く。呼び出された関数は並列に動作しても、それは非同期になっており実行の順序がずれる。そこで、CUDA では同一ブロック内のスレッドを同期させる関数があり、この関数を用いることでスレッド同士の同期をとる。

2.2 粒子法

連続体を有限個の粒子の集合として表し、この粒子１つ１つに速度、圧力といった変数を保持させ、連続体の挙動を粒子の運動によって計算する方法を粒子法という。

2.3 MPS(Moving Particle Semi-implicit)法

今回取り扱う MPS 法は粒子法の１つであり、連続体の挙動を表す微分方程式の微分演算子に粒子間相互作用モデルを適用して離散化を行う。

2.3.1　粒子間相互作用モデル

勾配モデル

$$\langle \nabla \varphi \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \left[\frac{\varphi_j - \varphi_i}{\left| \vec{r}_j - \vec{r}_i \right|^2} \left(\vec{r}_j - \vec{r}_i \right) w\left(\left| \vec{r}_j - \vec{r}_i \right| \right) \right]$$

発散モデル

$$\langle \nabla \cdot \vec{u} \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \frac{\left(\vec{u}_j - \vec{u}_i \right) \cdot \left(\vec{r}_j - \vec{r}_i \right)}{\left| \vec{r}_j - \vec{r}_i \right|^2} w\left(\left| \vec{r}_j - \vec{r}_i \right| \right)$$

ラプラシアンモデル

$$\langle \nabla^2 \varphi \rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} \left[\left(\varphi_j - \varphi_i \right) w\left(\left| \vec{r}_j - \vec{r}_i \right| \right) \right]$$

2.3 非圧縮性流れ

非圧縮性流れは圧力や温度による密度の変化が無視でき、密度がほぼ一定である性質をもつ。この研究では水を非圧縮性流れと考え水柱の崩壊を扱う。

3. 計算アルゴリズム

① 時間ステップ
② 陽的計算
③ 陰的計算

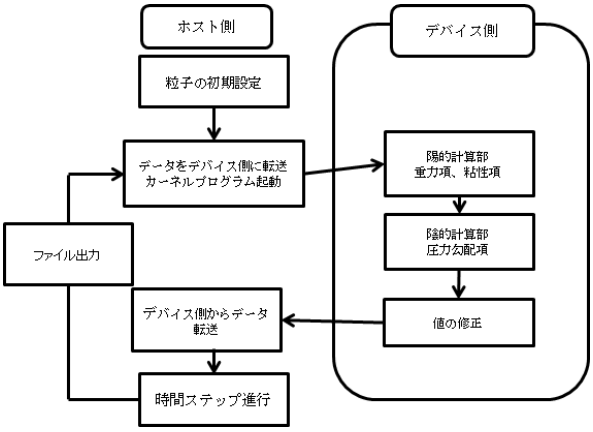

図 3.GPU による計算アルゴリズム

図 3 は、CUDA を用いた MPS 法による水柱崩壊シミュレーションのアルゴリズムを示す。

① 時間ステップ処理は数値が安定するように刻み幅を計算し、時間を進行させる。

②時刻 t での値を用いて陽解法で計算できる粒子の速度、位置を求める。この時の速度、位置を$\vec{u}^*,\vec{r}^*$とする。これは、陽的計算後に粒子数密度が一定量n^0になってないためである。このときの粒子数密度をn^*とすると、③で粒子数密度を修正する。

③ここでは粒子数密度をn^0に修正し、粒子の速度と位置が修正される。粒子数密度を修正する際に時刻 t+1 の圧力が必要となる。

$$n^0 = n^{t+1} = n^* + n'$$
$$\vec{u}^{t+1} = \vec{u}^* + \vec{u}' \qquad (1)$$
$$\vec{r}^{t+1} = \vec{r}^* + \Delta t\vec{u}'$$

n'は粒子数密度の修正量、$\vec{u}'$は速度の修正量である。$\vec{u}'$は圧力勾配項によって生じるとする。

$$\vec{u}' = -\frac{\Delta t}{\rho_0}\nabla P^{k+1} \qquad (2)$$

速度の修正量と粒子数密度の修正量は、流体の密度が粒子数密度に比例していることより圧縮性流れの質量保存則は

$$\frac{1}{n^0}\frac{dn}{dt} + \nabla\cdot\vec{u} = 0$$

と書き換えられる。これを離散化し、(2)式の発散をとったものを代入することで次の圧力のポアソン方程式が得られる。

$$\nabla^2 P^{t+1} = -\frac{\rho_0}{\Delta t^2}\frac{n^* - n^0}{n^0}$$

このポアソン方程式の左辺を各粒子位置においてラプラシアンモデルで離散化すると、

$$\langle\nabla^2 P\rangle_i^{t+1} = \frac{2d}{\lambda n^0}\sum_{j\neq i}\left[\left(P_j^{t+1} - P_i^{t+1}\right)w\left(\left|\vec{r}_j^* - \vec{r}_i^*\right|\right)\right]$$

となり、未知数P_i^{t+1}に対する連立 1 次方程式が得られ、この連立 1 次方程式を解くことで新しい時刻 t+1 の圧力が求まる。

3.1 CG(Conjugate Gradient)法

CG 法は対称正定値行列とする連立一次方程式を解くためのアルゴリズムである。大規模な行列を解く際によく利用されている方法で、理論上では行列の次元数の反復で収束することが可能とされている。

4. 結果

今回は、様々な計算量で解析するために、949, 2445, 3023, 4035, 5305, 6417, 7939, 10002, 20000 の 9 種類の粒子数の水柱モデルを用いて崩壊する様子を解析した。図 4 はそれぞれの粒子数での CPU での逐次処理と GP を用いた並列処理の処理時間を比較したグラフである。時間の測定は time 関数を用いた。

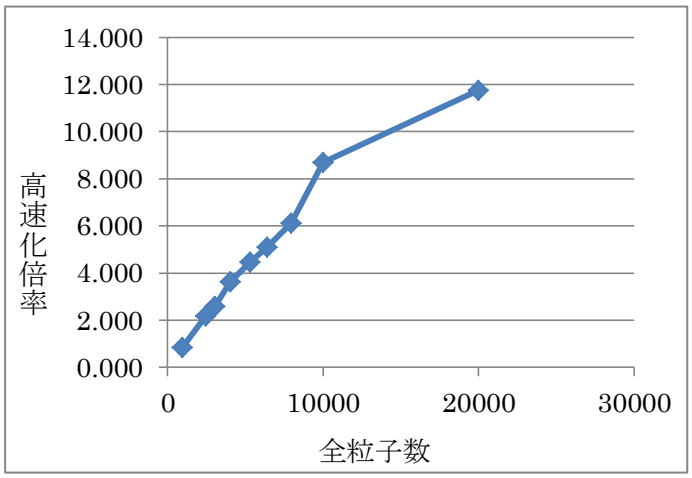

図 4 処理時間の比較

表 1 数値計算環境

CPU	Intel Core i3 , 3220K , 3.30GHz	
OS	Windows 7 Enterprise 64 ビット	
GPU	Geforce GTX 550Ti	
	Global Memory	1024 MB
	SM , CUDA core	4 , 192

5. 今後の目標

・粒子数を増やした状態での安定した処理性能の実現。
・効率のよいプログラミング
・3 次元への拡張

参考文献

・Jason Sanders , Edward Kandrot 著
「CUDA BY EXAMPLE 汎用 GPU プログラミング入門」
・越塚誠一著　「粒子法」