ISSN 2186-5647

2-8

CUDA/GPUによる粒子法シミュレーションの高速化

日大生産工（学部）　○永島　侃　　日大生産工　　角田　和彦　　都立産技高専　三浦　慎一郎
日大生産工（院）小原　俊介　　日大生産工　　豊谷　純

1　はじめに

大変形する構造や流体の解析の手法として粒子法が挙げられる．この粒子法では粒子同士の影響を調べる近傍探索や，圧力ポアソン方程式の計算において膨大な演算量を要し，モデルによっては演算に数十時間，数日かかることもある．

また，演算を行うCPUは，排熱の問題や高集積化により，クロック周波数の向上が難しく，CPU単体の性能向上が困難になりつつある．そんな中，近年注目されているのが，多くのコアを用いて動画像の描画を行うGPUを，グラフィックスの用途ではなく，汎用的な計算に用いるGPGPU(General Purpose Computation on Graphics Processing Unit)である．本研究は，粒子法の1つであるMPS法[1]の解析にGPGPUを用いて[2]解析の高速化を図ることを目的としている．

GPGPUには，AMD社によるATI Stream，NVIDIA社によるCUDA(Compute Unified Device Architecture)，Apple社によるOpen CLなどの統合開発環境がある[3]が，本研究は高速化を重視し，NVIDIA社によるCUDAを用いている．数値計算例としては，MPS法を用いた水柱崩壊シミュレーションの解析を行い[1]，CPUでの逐次計算の結果，OpenMPを用いた並列計算の結果及び各種GPU（シングルGPU，マルチGPU）での計算時間について様々な粒子数で考察している．

2　MPS法

MPS法は流体解析や構造解析に用いられる粒子法の一つである．勾配や発散といった微分演算子に対応する粒子間モデルを用意し，これらの微分演算子に適用することによって離散化する．重み関数wを導入し，粒子間相互作用モデルにはこの重み関数を利用する[1]．本研究では解の安定化を図るために重み関数にlog関数を用いている．

$$w(r) = \begin{cases} \log \dfrac{r_e}{r} & (0 \le r < r_e) \\ 0 & (r_e \le r) \end{cases} \tag{1}$$

図1におけるrは粒子間距離，r_eは影響半径である．

式(2)は勾配モデル，式(3)はラプラシアンモデルである．式(4)はラプラシアンモデルにおいて，統計的な分散の増加を解析解と一致させるための係数である．

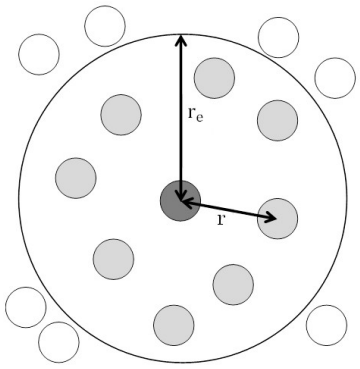

図1. 粒子間相互作用

$$\langle \nabla \phi \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \left[\frac{\phi_j - \phi_i}{|r_j - r_i|^2} (r_j - r_i)\, w\left(|r_j - r_i|\right) \right] \tag{2}$$

$$\langle \nabla^2 \phi \rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} \left[(\phi_j - \phi_i) w\left(|r_j - r_i|\right) \right] \tag{3}$$

$$\lambda = \frac{\sum_{j \neq i} |r_j - r_i|^2 w(|r_j - r_i|)}{\sum_{j \neq i} w(|r_j - r_i|)} \tag{4}$$

ただし，dとn^0はそれぞれの次元数，重み関数で表される粒子密度の初期値で，一定値である．

3　NVIDIA CUDA/GPU

現在のNVIDIA社GPUはFermiまたはKeplerとよばれるアーキテクチャが採用され，図2に示すようにGPU内にはストリーミング・マルチプロセッサー(SM)，CUDAコア，デバイスメモリ，シェアードメモリが搭載されている[3]．また，次世代のアーキテクチャであるKeplerも基本的にはFermiと同様であるが，Fermiのストリーミング・マルチプロセッサーがストリーミング・マルチプロセッサー・エクストリーム（SMX)となり，1つのストリーミング・マルチプロセッサー内に多くのCUDAコアを搭載できるようになっている．

Acceleration of Particle-based Simulations by GPU Using CUDA

Tsuyoki NAGASHIMA , Kazuhiko KAKUDA ,
Shinichiro MIURA , Shunsuke OBARA and Jun TOYOTANI

演算は図 2 のストリーミング・マルチプロセッサー内の CUDA コアで行われる．また，基本的に GPU で用いるデータは GPU 内のデバイスメモリに格納しておく必要がある．しかし，CPU と GPU 間の転送速度は GPU 内での演算に比べると非常に低速であるため，CPU，GPU 間の通信量を少なくすることが全体の高速化につながる．

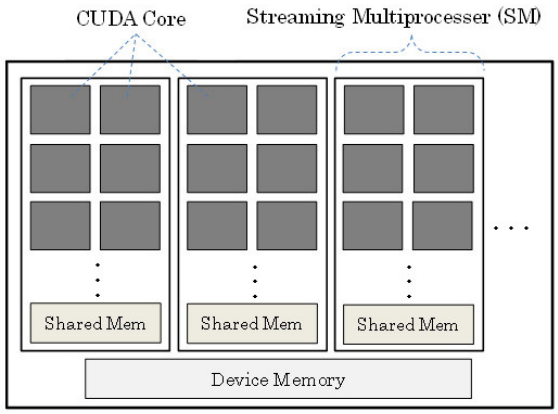

図 2.Fermi アーキテクチャ

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，ブロック毎に GPU 内のストリーミング・マルチプロセッサーに読み込まれ，ストリーミング・マルチプロセッサー内の CUDA コアがスレッドを処理する．

デバイスメモリの確保に関しては CUDA ランタイム API である cudaMalloc を用いて，動的に割り当てる方法と，__device__修飾子をつけた変数を宣言し，静的に割り当てる方法の二つの方法がある．動的に割り当てた場合は，cudaFree を用いて，メモリ領域を開放する．

CUDA には，C 言語を拡張した，CUDA C/C++ と，Fortran を拡張した CUDA Fortran があるが，本研究では CUDA C/C++を用いている．コンパイルの際には NVIDIA 社が提供している nvcc コンパイラを用いて，コンパイルが行われる．その際，ソースコード内の GPU の処理部分と，CPU の処理部分をコンパイラが自動的に分け，GPU の処理部分を nvcc コンパイラがコンパイルし，CPU の処理部分は C 言語のコンパイラに引き渡し，コンパイルが行われる．

4 マルチ GPU プログラミング

1 台のコンピュータに複数の GPU を装着することで，マルチ GPU プログラミングを実現することができる．これにより，演算性能の向上はもちろん，多くのメモリを利用することができ，より大規模なモデルの解析にも対応できる．

マルチ GPU における問題は，別の GPU のデバイスメモリ上にあるデータを使用することができない点である．そのため，一度，別の GPU のデータをメインメモリにコピーした後，メインメモリから，使用する GPU のデバイスメモリにコピーするという手続きが必要である．しかし，現在は CUDA 4.0 でサポートされた NVIDIA GPUDirect v2.0 により，主記憶を介さずに，直接別の GPU のデバイスメモリにアクセスすることができるようになり，マルチ GPU プログラミングの実現が容易になっている．

5 MPS 法の CUDA 化

図 3 に MPS 法による粒子法シミュレーションのフローチャートを示す．網掛けの部分を GPU で処理し，その他は CPU で処理している．ファイル書き出しは GPU で行うことができず，また，時間刻み幅計算や，速度，座標計算は並列性が低いため，CPU で処理を行っている．灰色の矢印は CPU から GPU へのデータのコピー，黒色の矢印が GPU から CPU へのデータのコピーである．

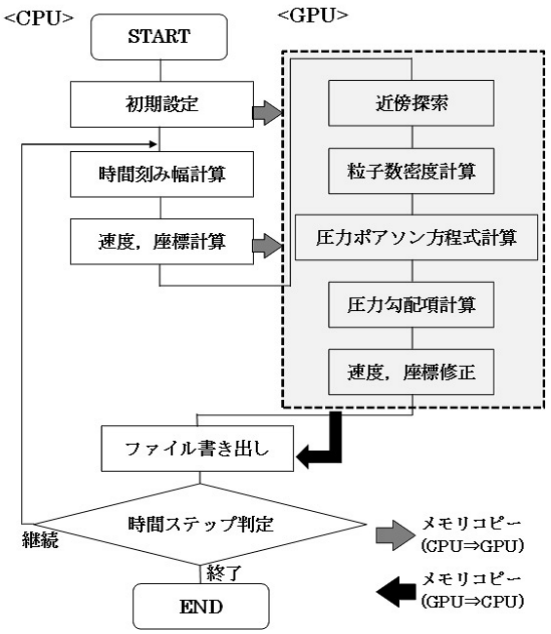

図 3. フローチャート (シングル GPU による解析)

デバイスメモリの確保の方法に関して，本研究では，動的割り当てを行った際，静的割り当ての場合と比べ，1.2 倍程度計算時間が多くなってしまうという結果が得られたため，シングル GPU による解析には，静的割り当てを行っている．

計算量が多い近傍探索では，マルチ GPU による解析も行っている．ここでは，OpenMP を用いて CPU で複数のスレッドを持たせ，それぞれのスレッドに GPU を受け持たせることにより，マルチ GPU を実現している[4]．また，GPUDirect を利用して，各 GPU が直接，別の GPU のデバイスメモリにアクセスするようにプログラミングしたところ，GPU1 台よりも解析時間が多くかかってしまった．そのため，本研究では，GPUDirect は用いずに，メインメモリを介して各 GPU 同士が互いのデータにアクセスしている．更に，デバイスメモリの確保に関しては，マルチ GPU では静的割り当てが使用できないため，動的割り当てによってデバイスメモリを確保した．

圧力ポアソン方程式計算では，大規模連立一次方程式 $\boldsymbol{Ax}=\boldsymbol{b}$ を解いている．この解法には，様々な解法が提案されているが，本研究では，GPU で並列計算を行うことを踏まえ，反復法を用いている．反復法のうち，単純で並列化がしやすい解法としてヤコビ法がある．しかし，ヤコビ法は収束が遅く，本研究でも，最大反復回数（500 回）の間では収束条件である 1.0E-5 まで収束せず，最悪の場合，時間ステップあたり 13000 回程度の反復で収束するという結果を得た．また，ヤコビ法の収束性を高める解法として，ガウス・ザイデル法，SOR 法等があるがこれらの解法は並列化が難しい．そこで，本研究では共役勾配法（Conjugate Gradient Method，CG 法）に着目した．

CG 法では収束性を更に高めるために，$\boldsymbol{Ax}=\boldsymbol{b}$ の行列 $\boldsymbol{A}$ に前処理を行うことが多い．前処理を用いた CG 法としては，行列 $\boldsymbol{A}$ に対し，不完全コレスキー分解を行う ICCG 法がよく用いられるが，この解法は並列化が難しい．そのため，本研究では，単なる CG 法ではなく，行列 $\boldsymbol{A}$ のそれぞれの要素を対角項で割り，対角項がすべて 1 になるように前処理を施した対角項スケーリング共役勾配法（Scaled Conjugate Gadient Method，SCG 法）[5)6)] を用いた．SCG 法により，様々な粒子数における水中崩壊シミュレーションの時間ステップ毎の反復回数の総和は，CG 法に比べ，3000 回〜10000 回少なくなった．

6　数値計算環境

本研究は表 1〜表 4 の 4 台のマシンを用いて計算を行った．

表 1. 計算環境（マシン 1）

CPU	Intel Core i7, 2700K, 3.50GHz 8M L3 cache , 4core , 8thread	
Memory	DDR3 PC3-10600 16GB	
OS	Cent OS 6.0 64bit	
gcc	version 4.4.6	
GPU	Geforce GTX 580	
	Bus interface	PCI-e 2.0 x16
	Memory Bandwidth	192.4 GB/sec
	Global Memory	1.5GB
	SM , CUDA Core	16 , 512

表 2. 計算環境（マシン 2）

CPU	Intel Xeon E5, 2670, 2.60GHz 20M L3 cache , 8core , 16thread	
Memory	DDR3 PC3-12800 24GB	
OS	Cent OS 6.2 64bit	
gcc	version 4.4.6	
GPU	Tesla C2075	
	Bus interface	PCI-e 2.0 x16
	Memory Bandwidth	144 GB/sec
	Global Memory	6GB
	SM , CUDA Core	14 , 448

表 3. 計算環境（マシン 3）

CPU	Intel Core i7, 3770K, 3.50GHz 8M L3 cache , 4core , 8thread	
Memory	DDR3 PC3-12800 16GB	
OS	Ubuntu 12.04 64bit	
gcc	version 4.4.7	
GPU	Gefoce GTX 690 （GTX 680 × 2）	
	Bus interface	PCI-e 3.0 x16
	Memory Bandwidth	192.3 GB/sec
	Global Memory	2GB
	SM , CUDA Core	8 , 1536

表 4. 計算環境（マシン 4）

CPU	Intel Core i7, 3820, 3.60GHz 10M L3 cache , 4core , 8thread	
Memory	DDR3 PC3-10600 16GB	
OS	Ubuntu 12.04 64bit	
gcc	version 4.4.7	
GPU	Gefoce GTX 580 × 4	
	Bus interface	PCI-e 2.0 x16
	Memory Bandwidth	192.4 GB/sec
	Global Memory	3.0GB
	SM , CUDA Core	16 , 512

CUDA Driver，CUDA Tool kit，CUDA SDK はすべて Version 4.2 である．マシン 3 には Geforce GTX690 が搭載されているが，この GPU は 1 台の GPU の中に 2 台の GPU が搭載されているという特殊な GPU である．マシン 4 に関しては Geforce GTX580 が 4 台装着されている．しかし，これらの GPU はマルチ GPU 用にプログラミングしない限り，1 台の GPU でのみ演算が行われる．アーキテクチャは GTX580，Tesla C2075 が Fermi，GTX690 が Kepler である．

CPU 側のコンパイラ（C コンパイラ）に関しては，Linux 系 OS の標準的 C コンパイラである GNU コンパイラコレクション（GCC）を用いている．また，GCC の最適化オプションとして-O2 を設定している．GPU での処理部分への最適化のオプションに関しては設定していない．また，数値演算ライブラリの CUBLAS や CUSPARSE は使用していない．

7　数値計算例

本研究では，様々な計算量で解析するために，1458，16221，31465，60875，120788 の 5 種類の粒子数の水柱モデルを用いて，流体の粒子が重力の影響を受けて崩壊する様子を解析した．図 4 は粒子数 16221 の水中崩壊シミュレーションを CPU で計算し，OpenGL を用いて可視化した結果である．また，図 5 は同じシミュレーションを GPU で行った結果である．ここでは，水の粘性や表面張力等については考慮していない．同一の時間における水中崩壊の様子は CPU，GPU どちらも同じ結果となることを確認できる．

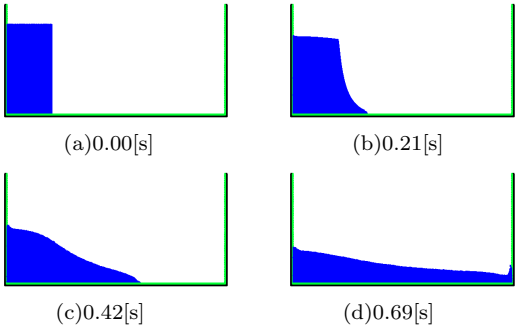

図 4. 水柱崩壊の解析結果（CPU）

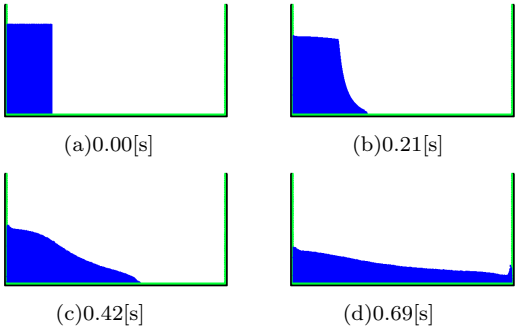

図 5. 水柱崩壊の解析結果（GPU）

図 6 はシングル GPU による高速化倍率のグラフである．マシン 3 の CPU(Core i7 3770K）での逐次計算よりもどれだけ高速化できたかを粒子数別に示している．このグラフから，GeForce GTX580 が一番高速であることがわかる．Tesla C2075 に関しては，汎用計算向けの GPU であるが，メモリバンド幅が小さく，データ通信に時間を要したため，速度が出なかったことが考えられる．また，GeForce GTX690 は，グラフィックに非常に特化しており，倍精度演算性能が弱いことが原因と考えられる．

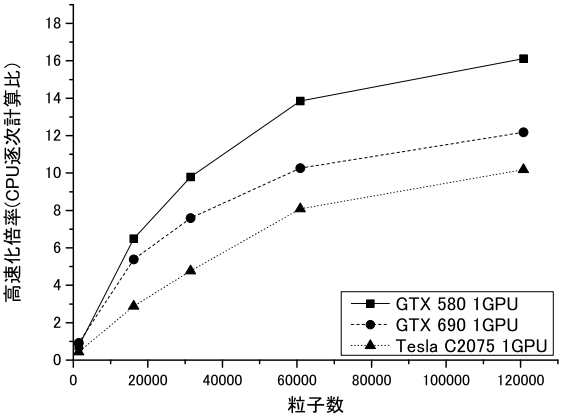

図 6.GPU 別の高速化倍率

図 7 は，OpenMP を用いた CPU における解析と GPU での解析の演算時間の比較である．図 6 と同様に，マシン 3 の CPU（逐次計算）に対する高速化倍率を示している．ここでは，3 台の GPU のうち一番高速である GeForce GTX580 のシングル GPU, マルチ GPU(2 台) での演算結果と，マシン 2 の CPU(Xeon)，マシン 3 の CPU（Core i7）で並列計算した結果を比較している．粒子数 120788 ではマルチ GPU による解析が一番高速になった．また，GPU はサーバ用 CPU である Xeon よりも高速に解析することができている．

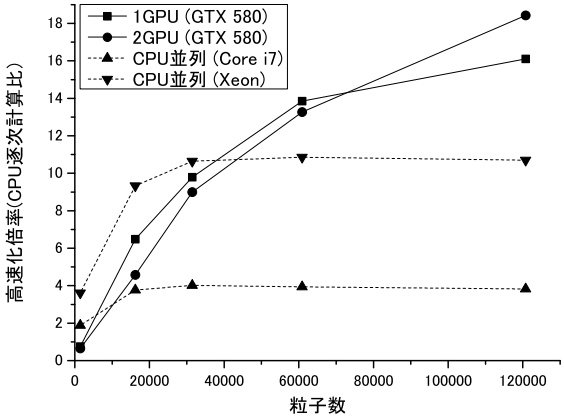

図 7.CPU と GPU の速度の比較

8　おわりに

本研究では，MPS 法を用いた水柱崩壊シミュレーションの解析を GPU を用いて行った．粒子数によって高速化倍率に違いがあるが，粒子数 120788 において，CPU(Core i7) の逐次計算に比べ，シングル GPU で約 16 倍，マルチ GPU で約 18 倍の高速化を達成することができた．また，サーバ用 GPU である Xeon を OpenMP で並列計算させた場合よりも GPU での解析が高速であるという結果を得た．また，解析結果に関しては，同一ステップ時間において，CPU での解析と同じ結果を得ることができた．今後はマルチ GPU による解析の更なる高速化とともに，CUDA だけではなく，ATI Stream や Open CL 等の開発環境で GPGPU を実現し，性能比較をしていきたい．

[参考文献]

1)S.Koshizuka and Y.Oka,”Moving-Particle Semi-implicit Method for Fragmentation of Incompressible Fluid”, NUCLEAR SCIENCE AND ENGINEERING, (1996), 123, p.421-434
2)K.Kakuda, T.Nagashima, S.Obara, ”Particle-based Fluid Flow Simulations on GPGPU Using CUDA”, CMES, submit
3) 青木尊之, 額田彰, はじめての CUDA プログラミング, 工学社, (2009), p.25
4) 河田直樹, 大久保寛, 土屋隆生, GPU を用いた電磁界数値解析の高速化に関する検討, 信学気報, A-P2009-141 , (2009)
5) 速水謙, 原田紀夫, 対角項スケーリングを施した共役勾配法のベクトル計算機における有効性について, 情報処理学会論文誌, Vol.30 No.11 , (1989)
6) 池田優介, 藤野清次, 山田知典 , ノルム前処理付き共役勾配法の性能評価 , ハイパフォーマンスコンピューティング, 91-3 , (2002)