電気通信大学情報理工学研究科
情報・通信工専攻情報数理工学コース

平成27年度
修士論文

GPUおよび領域分割を用いた粒子法による流体シミュレーションの高速化

平成28年3月15日

情報・通信工学専攻
学籍番号 1431051
佐々木卓雅

指導教員 龍野智哉 准教授
副指導教員 山本有作 教授

目 次

1　背景

流体シミュレーションの手法として、計算対象の空間を格子によって分割し、格子点上の固定された計算点を用いて計算を行う格子法が用いられてきた。代表的な方法として差分法、有限要素法などが挙げられる。しかし、この手法には、流体の自由表面の追跡の為に特別なスキームが必要なことや複雑形状に対する格子生成の手間がかかるなどの課題があった。この課題の解決に対して、粒子法は有望な解決法の一つと言える。粒子法は、連続体を有限個の粒子によって表し、連続体の挙動を粒子の運動によって計算する方法である (図 1)。各粒子は速度や圧力といった変数を保持しながら移動するので格子は使われない。

流体シミュレーションを行うために、最初の粒子法として、Particle-and-Force Method（PAF 法）[1] が提案された。しかし、この方法には発展性がなく現在では研究が途絶えている。その後、Particle in Cell Method（PIC 法）[2] と Marker and Cell Method（MAC 法）[3] という格子と粒子を組み合わせた方法が提案された。PIC 法は対流項を粒子で、その他の項を格子で計算を行う。MAC 法は自由表面の形状をトレースするために粒子が使われている。その後、1977 年に宇宙物理学の問題を解くため、Smoothed Particle Hydrodynamics Method（SPH 法）[4] が提案された。宇宙物理学の圧縮性流体を扱うための方法であるが、自由表面をともなう流れにも適用されている。密度のような空間分布をカーネル関数の重ね合わせで表現する。

しかし、工学に関する問題では、非圧縮性流体を扱う場合が多い。そこで、1995 年に非圧縮性流体を扱うための粒子法として、越塚らが Moving Particle Semi-implicit Method（MPS 法）[5] という方法を提案した。MPS 法は MAC 法と同様に半陰的アルゴリズムを用いている。微分演算子に対する離散化方法として、粒子間相互作用モデルを用いて流体の支配方程式の離散化を行っている。また、他の微分方程式に対してもこの離散化方法は適用可能であり、流体解析だけでなく構造解析にも使われている。

粒子法では、近傍の粒子を効率よく探索するために領域分割が有効である。グリッドによって計算領域を分割し、探索の対象となる粒子を限定することで計算量を減らすことができる。この領域分割のプログラムへの実装方法は、Uniform Grid [6] をはじめとしていくつかの方法が提案されている。

GPU (Graphics Processing Unit) は配列構造の単純なデータを単精度程度の浮動小数点演算によって順番に処理することで 2 次元の動画像データを実時間内に生成することに特化しているアーキテクチャになっている。そのため、CPU と比べて浮動小数点演算能力が高く、GPU 専用メモリのメモリバンド幅が広いという特徴がある。この GPU を数値演算にも利用しようとするのが GPGPU (General-purpose computing on graphics processing units) または GPU コンピューティングと呼ばれる技術である。

現在、NVIDIA から統合開発環境や GPGPU に特化した GPU が登場し、スー

パーコンピュータにも使用されるなど GPGPU は活用の幅が広がっている技術である。応用例としては、医療画像の解析、流体計算、電磁波シミュレーション、天文シミュレーション、たんぱく質の挙動を解析する数値シミュレーション、バイオ・インフォマティクス、金融工学など、幅広い分野で利用されている。

2 章で MPS 法の離散化方法を説明し、3 章で非圧縮性流体の計算方法を説明する。高速化方法として、4 章で GPU のアーキテクチャや最適化方法について説明し、5 章で領域分割による粒子探索の効率化方法の Uniform Grid を説明する。そして、6 章で高速化を行ったプログラムの計算時間の計測結果を示し、7 章で結論を述べる。

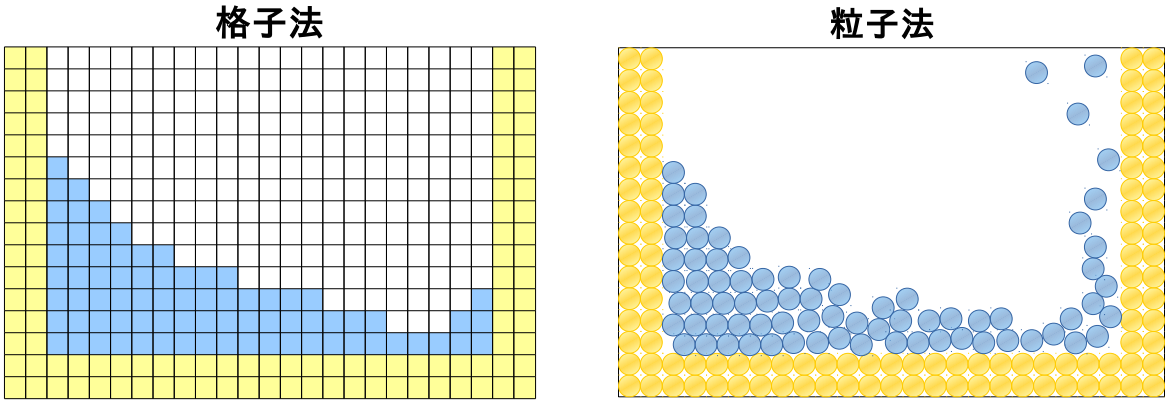


図 1: 格子法と粒子法の概念図

2　粒子法

2.1　粒子間相互作用モデル

2.1.1　重み関数

MPS 法では、勾配、発散、ラプラシアンといったベクトル解析の微分演算子に対してそれぞれ粒子間相互作用モデルを用意し、これらを用いて微分方程式を離散化する [5]。

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用いる。

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

r は粒子間距離、r_e は粒子間で相互作用する距離を表わすパラメータである (図 2)。$r = 0$ で w は無限大になるが、非圧縮性流体の解析において計算を安定させる効果がある。パラメータ r_e が小さいと相互作用する粒子の数が減るので計算時間が短

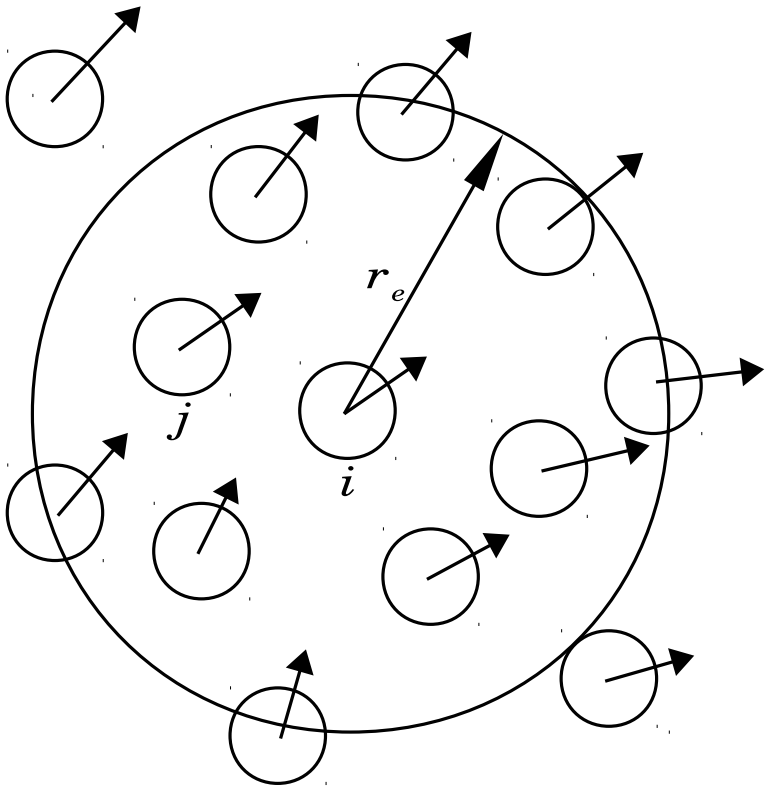


図 2: 粒子間相互作用

くなるが、小さくしすぎると計算が不安定になる。そこで計算時間と計算の安定性の兼ね合いから、r_e は微分演算子によって粒子間距離の 2〜4 倍を使い分けている。2 次元の場合、相互作用する近傍の粒子の数が 12〜46 個程度になる。式 (1) の重み関数をグラフに描くと図 3 のような形状となり、粒子間の距離がゼロに近づくと大きくなり、r_e より大きくなればゼロになる関数である。

粒子 i およびその近傍の粒子 j の位置ベクトルをそれぞれ $\boldsymbol{r}_i$、$\boldsymbol{r}_j$ とする。粒子 i の粒子数密度は粒子 i の位置における重み関数の和をとったものになる。

$$n_i = \sum_{j \neq i} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \tag{2}$$

各粒子の保持する質量が一定であるとすると、粒子数密度は流体の密度と比例する。粒子 i について、重み関数の半径 r_e 内に存在する粒子の個数 N_i は粒子数密度から次のように定義される。

$$N_i = \frac{n_i}{\int_\Omega w dv} \tag{3}$$

右辺の分母は、空間の体積を領域の積分として表したものである。シミュレーションでは重み関数として式 (1) を用いるが、ここでは議論の簡単化のため、便宜上重

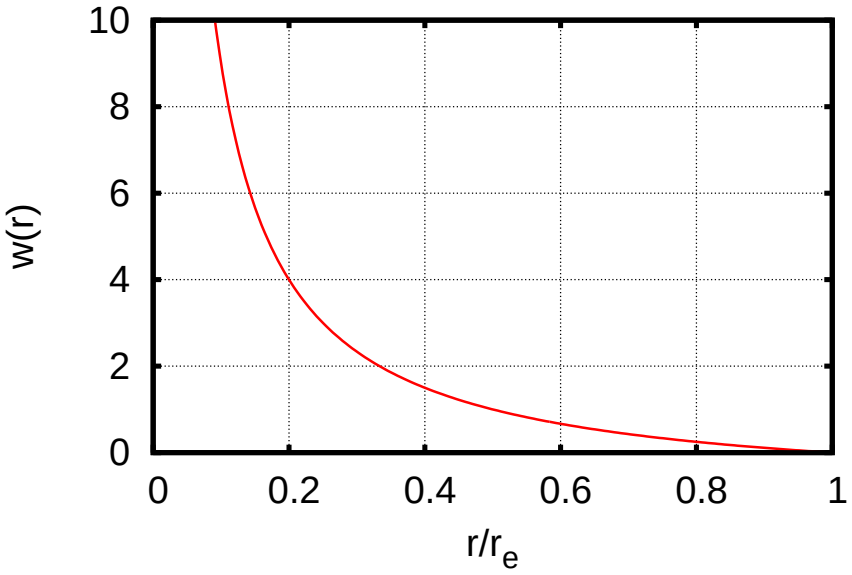


図 3: 重み関数

み関数を h として、領域 Ω の内部で 1、外部で 0 となるものとする。

$$h(r) = \begin{cases} 1 & 0 \leq r < r_e \\ 0 & r_e \leq r \end{cases} \tag{4}$$

粒子 i の重み関数の半径 r_e 内に、質量 m の粒子が N_i 個存在すると考えると、質量密度 ρ_i は次のように表される。

$$\rho_i = mN_i \tag{5}$$

この式を、式 (2) と式 (3) を使い書き直すと次の式になる。

$$\rho_i(\boldsymbol{r}_i) = \frac{m\sum_{j\neq i} h(|\boldsymbol{r}_j - \boldsymbol{i}_i|)}{\int_\Omega h dv} \tag{6}$$

式 (6) の右辺の分子は、位置ベクトル r の近傍の粒子 j に対する和である。つまり、領域の中の粒子の個数 N_i に質量 m をかけたことを表しており、式 (5) の右辺の分子と等しい。式 (6) の右辺の分母は領域の体積を表しており、式 (5) と等しい。式 (6) の重み関数を MPS 法の重み関数である式 (1) を使い、流体の密度を計算すると、

$$\rho_i(\boldsymbol{r}_i) = \frac{m\sum_{j\neq i} w(|\boldsymbol{r}_j - \boldsymbol{r}_I|)}{\int_\Omega w dv} \tag{7}$$

となり、重み関数の和を取ったものである粒子数密度で式を整理すると

$$\sum_{j\neq i} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) = \frac{\rho_i(\boldsymbol{r})\int_\Omega w dv}{m} \tag{8}$$

となる。MPS 法の重み関数では $r = 0$ で $w = \infty$ となるから、$r = 0$ を避ける為に式 (8) を $j = i$ の時に、粒子 i についての計算を除外する。

$$n_i = \sum_{j \neq i} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) = \frac{\rho_i(\boldsymbol{r}_i) \int_{\Omega'} w dv}{m} \tag{9}$$

ここで、Ω' は粒子 i を除いた領域である。式 (9) より、粒子 1 個の質量 m が一定で、領域 Ω' が一定、つまり r_e が一定である時、粒子数密度は流体の密度に比例する。

非圧縮性流体では流体の密度は一定であり、したがって粒子数密度も一定で無ければならない。この一定値を n^0 とする。n^0 の計算には重み関数の半径 r_e の中に十分に粒子がある状態で粒子数密度を計算し、シミュレーション中はこの値を使い続ける。

2.1.2 勾配モデル

互いに近接する 2 つの粒子が、それぞれ位置ベクトル $\boldsymbol{r}_i$、$\boldsymbol{r}_j$ とスカラー変数値 ϕ_i、ϕ_j を持っているとする。粒子 j の変数値 ϕ_j を、粒子 i の変数値 ϕ_i からのテイラー展開で表すと

$$\phi_j = \phi_i + \nabla\phi|_i \cdot (\boldsymbol{r}_j - \boldsymbol{r}_i) + \cdots \tag{10}$$

となる。1 次の項で考えると、

$$\nabla\phi|_i \cdot (\boldsymbol{r}_j - \boldsymbol{r}_i) \simeq \phi_j - \phi_i \tag{11}$$

左辺の勾配は、粒子 i, j に対して対称で、粒子 i と粒子 j の間の値と考えると

$$\nabla\phi|_{ij} \cdot (\boldsymbol{r}_j - \boldsymbol{r}_i) = \phi_j - \phi_i \tag{12}$$

左辺は、粒子 i,j 間における勾配ベクトル $\nabla\phi|_{ij}$ と、相対位置ベクトル $\boldsymbol{r}_j - \boldsymbol{r}_i$ の内積である。両辺を相対位置ベクトルの絶対値で割ると、右辺は差分式になる。

$$\nabla\phi|_{ij} \cdot \frac{(\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} = \frac{\phi_j - \phi_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} \tag{13}$$

さらに両辺に相対位置ベクトル $\boldsymbol{r}_j - \boldsymbol{r}_i$ 方向の単位ベクトルを掛けると、

$$\langle \nabla\phi \rangle_{ij} = \left[\nabla\phi|_{ij} \cdot \frac{(\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} \right] \frac{(\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} = \frac{(\phi_j - \phi_i)(\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} \tag{14}$$

となり、これが粒子 i,j 間で定義される勾配モデルである。$\langle\rangle$ は粒子間相互作用モデルであることを表す記号である。

勾配はスカラー変数に作用してベクトルが得られる演算子である。ある粒子に対して、その近傍にある粒子との間の物理量の受け渡しにより表される。式 (14) の勾配ベクトルに、近傍の粒子を考慮するように重み関数を利用した重み平均を

取るようにする。よって、MPS 法では粒子 i の位置における勾配ベクトルに対して、次の計算モデルを用いる。

$$\langle \nabla\phi \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \left[\frac{\phi_j - \phi_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} (\boldsymbol{r}_j - \boldsymbol{r}_i) w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right] \tag{15}$$

d は空間次元数、重み関数 w は式 (1) の重み関数、n^0 は粒子数密度である。式 (15) は、式 (14) のベクトルを重み付き平均したものである (図 4)。n^0 は重み付き平均の正規化の為に使われ、粒子 i の位置における粒子数密度 n_i ではなく、計算の簡単化の為に n^0 を用いた。しかし、重み平均を単純に取ると傾きが正しい値に対して、$1/d$ 倍になってしまう。これは、勾配がベクトル量にも関わらず、相対位置ベクトル方向の成分しか考慮していないためである。そのため、正しい値を求めるためには、加えて直交方向の傾きの成分も考慮する必要がある。2 次元の勾配計算の場合では、勾配ベクトルを表すために直交する 2 つの軸方向の勾配を足して求めなければならない。d を使わずに 2 次元の勾配計算を行う式は

$$\langle \nabla\phi \rangle_i = \frac{1}{n^0} \sum_{j \neq i} \left\{ \left[\frac{(\phi_j - \phi_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} + \frac{(\phi_{j_\perp} - \phi_i)}{|\boldsymbol{r}_{j_\perp} - \boldsymbol{r}_i|} \frac{\boldsymbol{r}_{j_\perp} - \boldsymbol{r}_i}{|\boldsymbol{r}_{j_\perp} - \boldsymbol{r}_i|} \right] w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right\} \tag{16}$$

となる。式 (16) は、図 5 に示すように、実線の矢印の相対位置ベクトル $\boldsymbol{r}_j - \boldsymbol{r}_i$ の傾きに加えて、それに破線の矢印の直交する方向のベクトル $\boldsymbol{r}_{j_\perp} - \boldsymbol{r}_i$ の傾きを足して計算している。$\phi_{j_\perp}$ は粒子 i と粒子 j の粒子間距離と同じ距離にある粒子 $j_\perp$ が持つスカラーで、相対位置ベクトルと直交する位置 $r_{j_\perp}$ にあることを表している。MPS 法では、各粒子に対して近傍の粒子がどの方向にも均等に存在すると仮定しているので、直交する位置 $r_{j_\perp}$ にも粒子が存在すると仮定して計算している。粒子 j について和を取ると、2 次元の場合 ϕ_j が 2 度出てくることになるので、$\phi_{j_\perp}$ を計算する代わりに、ϕ_j を 2 回足し合わせている。そこで、式 (15) では d 次元の時に、重み平均を d 倍にして計算している [7][8]。

2.1.3 ラプラシアンモデル

ラプラシアンは物理的には拡散を意味している。ここで、拡散方程式を考える。

$$\frac{\partial f}{\partial t} = \kappa \nabla^2 f \tag{17}$$

κ は拡散係数である。1 次元の時、この式の解析解は

$$f(x,t) = \frac{1}{\sqrt{4\pi\kappa t}} e^{-\frac{x^2}{4\kappa t}} \tag{18}$$

となる。すなわち、初期分布を原点におけるデルタ関数とした場合の解析解は $t > 0$ でガウス分布になる。加えて、d 次元におけるフーリエ変換とフーリエ逆変換は以下の式で表される。

$$F(\boldsymbol{k}) = \int_{-\infty}^{\infty} f(\boldsymbol{r}) e^{-i\boldsymbol{k}\cdot\boldsymbol{r}} d\boldsymbol{r} \tag{19}$$

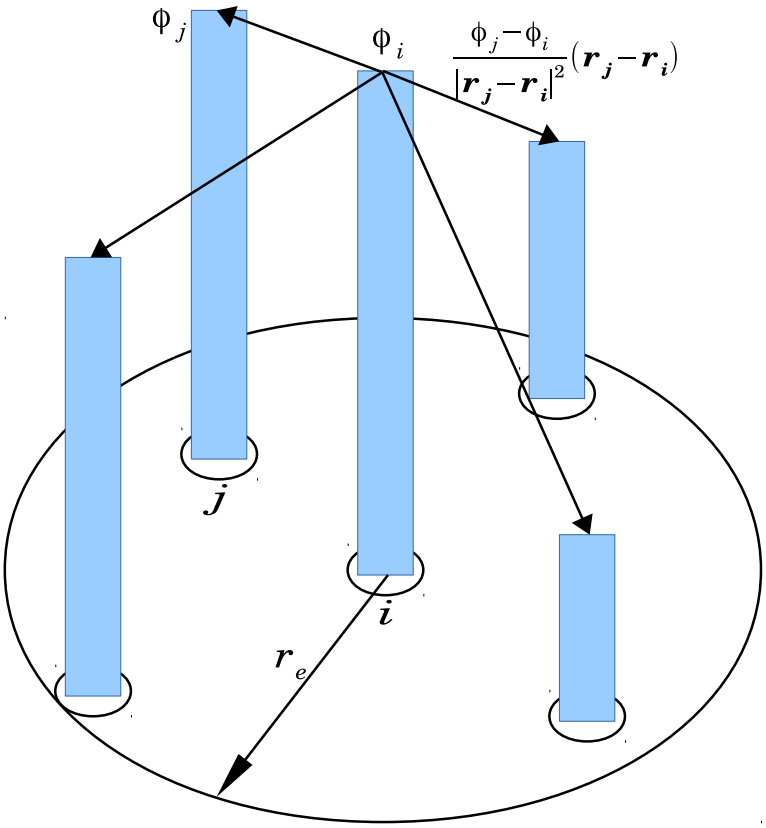


図 4: 勾配モデルの概念図

$$f(\boldsymbol{r}) = \frac{1}{(2\pi)^d}\int_{-\infty}^{\infty} F(\boldsymbol{k})e^{i\boldsymbol{k}\cdot\boldsymbol{r}}d\boldsymbol{k} \tag{20}$$

これらの式と 1 次元の拡散方程式の解析解である式 (18) を用いると、MPS 法における多次元の場合の解析解を得る。

$$f(r,t) = \left(\frac{1}{\sqrt{4\pi\kappa t}}\right)^d \exp\left(-\frac{r^2}{4\kappa t}\right) \tag{21}$$

ここで、$\boldsymbol{r}$ は原点からの距離、d は空間次元数である。統計的な分散 σ^2 は

$$\sigma^2(t) = \int r^2 f(r)dv = 2d\kappa t \tag{22}$$

で表されるように時間とともに線形に増加する。任意の初期分布をデルタ関数の重ね合わせと考えれば、その解析解はガウス分布の重ね合わせになる。粒子法において、各粒子が変数を保持していることをデルタ関数の重ね合わせとみなせば、ラプラシアンモデルとしてガウス分布に従った分配を採用すればよいことになる。しかしながら、ガウス分布は、中心付近で急峻であるために粒子への離散的な分配を用いると誤差が大きくなることと、分布が無限遠方まで達するので計算時間

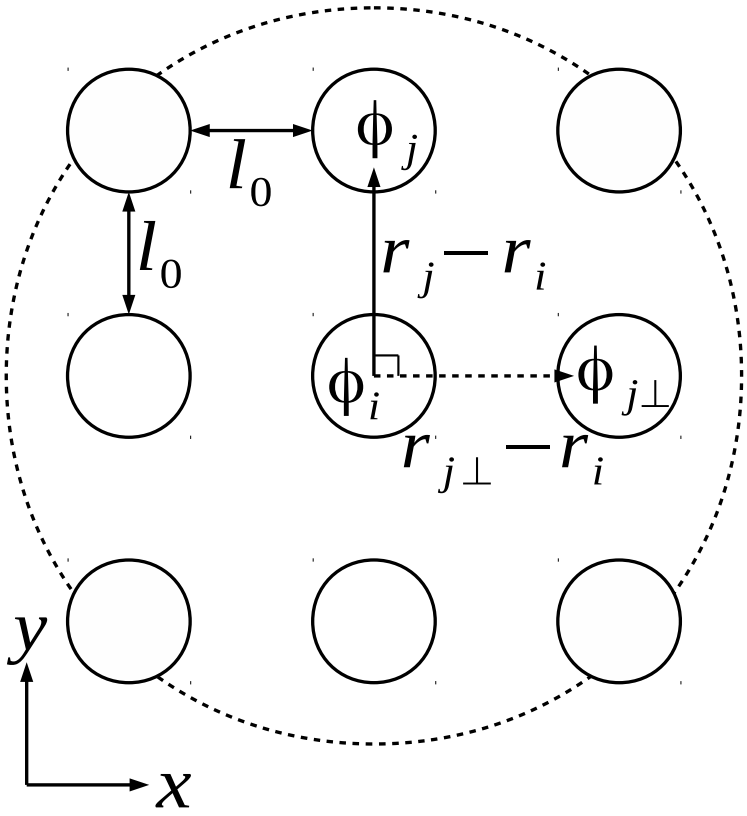


図 5: 勾配モデルの計算方法

が膨大になるという問題がある。中心極限定理によれば、ガウス分布とは異なる分布であっても、分散の増分を一致させておけば、その分布に分配を繰り返していくことでガウス分布に収束する [7]。そこで、誤差を減らすためにあまり中心付近で急峻ではなく、計算時間を節約するために有限の距離までしか達しない分布、重み関数で分配する。

以上のことから、ラプラシアンモデルを次の式で与える。

$$\left\langle \nabla^2 \phi \right\rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} [(\phi_j - \phi_i) w(|\boldsymbol{r}_j - \boldsymbol{r}_i|)] \tag{23}$$

これは図 6 に示すように、粒子 i の ϕ の一部を近傍の粒子 j に重み関数の分布で分配することを意味している (図 6)。ここで統計的な分散の増加を解析解と一致させる為に係数 λ を導入する。

$$\lambda = \frac{\sum_{j \neq i} |\boldsymbol{r}_j - \boldsymbol{r}_i|^2 w(|\boldsymbol{r}_j - \boldsymbol{r}_i|)}{\sum_{j \neq i} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|)} \tag{24}$$

係数 λ を求めるには、重み関数の半径 r_e 内に十分な粒子がある状態で計算したものを用いる。シミュレーションでは初期粒子配置で計算した、λ^0 を用いた次の式

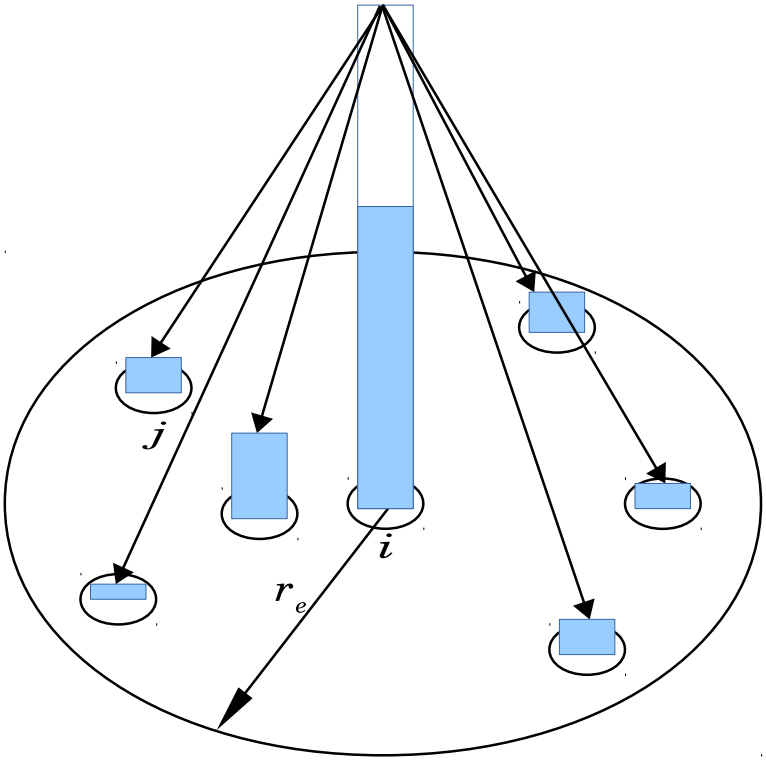


図 6: ラプラシアンモデルの概念図

で計算を行う。

$$\left\langle \nabla^2 \phi \right\rangle_i = \frac{2d}{\lambda^0 n^0} \sum_{j \neq i} \left[(\phi_j - \phi_i) w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right] \tag{25}$$

式 (17) の右辺のラプラシアンに式 (23) を適用し、左辺の時間微分にオイラー法を適用すると、次のように離散化される、

$$f_i^{k+1} = f_i^k + \Delta t \kappa \frac{2d}{\lambda n^0} \sum_{j \neq i} \left[(f_j^k - f_i^k) w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right] \tag{26}$$

上付き添字は時間ステップを表し、Δt は時間ステップ幅である。いま、粒子 p のみが変数値 f_p^k を保持し、その他の粒子の変数値はゼロとする。式 (26) で $i = p$ として

$$f_p^{k+1} = f_p^k + \Delta t \kappa \frac{2d}{\lambda n^0} \sum_{j \neq p} \left[(-f_p^k) w(|\boldsymbol{r}_j - \boldsymbol{r}_p|) \right] \tag{27}$$

次に、粒子 p の近傍の粒子 q における計算を考える。式 (26) で $i = q$ とすると、右辺における和の記号の中には、$j = p$ の時だけ値をもつので

$$f_q^{k+1} = \Delta t \kappa \frac{2d}{\lambda n^0} (f_p^k) w(|\boldsymbol{r}_p - \boldsymbol{r}_q|) \tag{28}$$

となる。式 (27) と式 (28) より、粒子 q が粒子 p より受け取る値と、粒子 p が粒子 q に対して与える値は等しく、保存性がある。MPS 法はラプラシアンモデルを用いて非定常拡散の計算をする場合に、変数値の和は保たれ、増減しない。

2.2 非圧縮性流体の計算方法

2.2.1 非圧縮性流体の支配方程式

非圧縮性流体の支配方程式を以下に示す。

$$\frac{D\rho}{Dt} + \rho\nabla\cdot\boldsymbol{u} = 0 \tag{29}$$

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\boldsymbol{u} + \boldsymbol{g} \tag{30}$$

式 (29) は連続の式、式 (30) はナビエストークス方程式と呼ばれ、ν は動粘性係数、$\boldsymbol{g}$ は重力加速度である。

非圧縮性流体において、連続の式は密度が時間変化せず、速度の発散もゼロになる。式 (29) の右辺の第 2 項を残している理由は、圧力のポアソン方程式を解く際に使うためである。差分法では、速度の発散がゼロであるという式を用いるが、MPS 法では密度が一定であるという式を用いる。式 (30) の左辺は速度ベクトルに対する時間微分で、右辺の第 1 項は圧力項、第 2 項は粘性項、第 3 項は重力項である。

左辺の D/Dt は、ラグランジュ微分で次の式で表せる。

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \boldsymbol{u}\cdot\nabla \tag{31}$$

$\boldsymbol{u}$ は粒子の速度である。これは、微分される物理量が流れに乗って観測される場合に使われる時間の微分演算子である。MPS 法では、ラグランジュ的に粒子は物理量を持ち、流れとともに移動するので、普通の時間微分と見なすことができる。

2.2.2 計算アルゴリズム

非圧縮性流体の計算アルゴリズムとして半陰的アルゴリズムを適用する。各時間ステップで、時刻 t^k における各粒子の位置 $\boldsymbol{r}_i^k$、速度 $\boldsymbol{u}_i^k$、圧力 P_i^k を元に、次の時刻 t^{k+1} の値である $\boldsymbol{r}_i^{k+1}$、$\boldsymbol{u}_i^{k+1}$、P_i^{k+1} を計算する。各時間ステップは前半の陽的な部分と後半の陰的な部分に分けて段階的に計算する。重力項と粘性項を時刻 t^k の値で陽的に、圧力項を時刻 t^{k+1} の値で陰的に計算する。MPS 法で、解くべきナビエ-ストークス方程式は次の式になる。

$$\frac{\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^k}{\Delta t} = -\frac{1}{\rho^0}\left\langle\nabla P\right\rangle_i^{k+1} + \nu\left\langle\nabla^2 u\right\rangle_i^k + \boldsymbol{g}^k \tag{32}$$

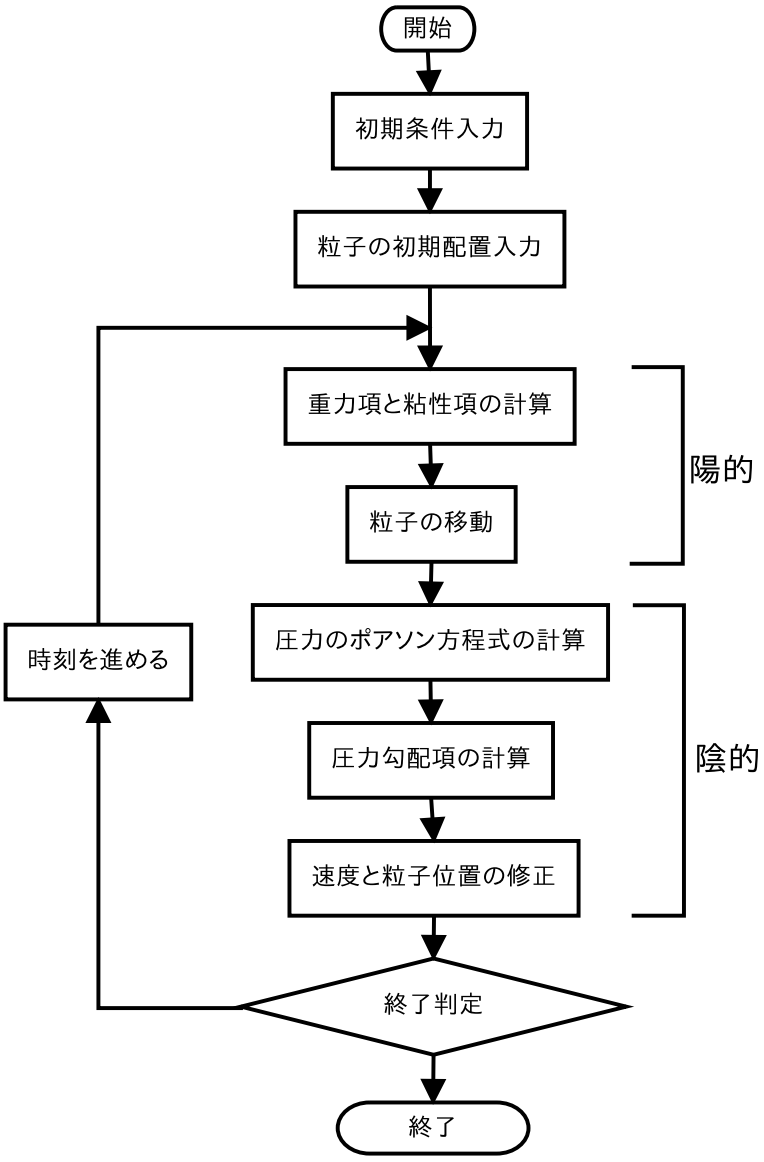


図 7: MPS 法の計算アルゴリズム

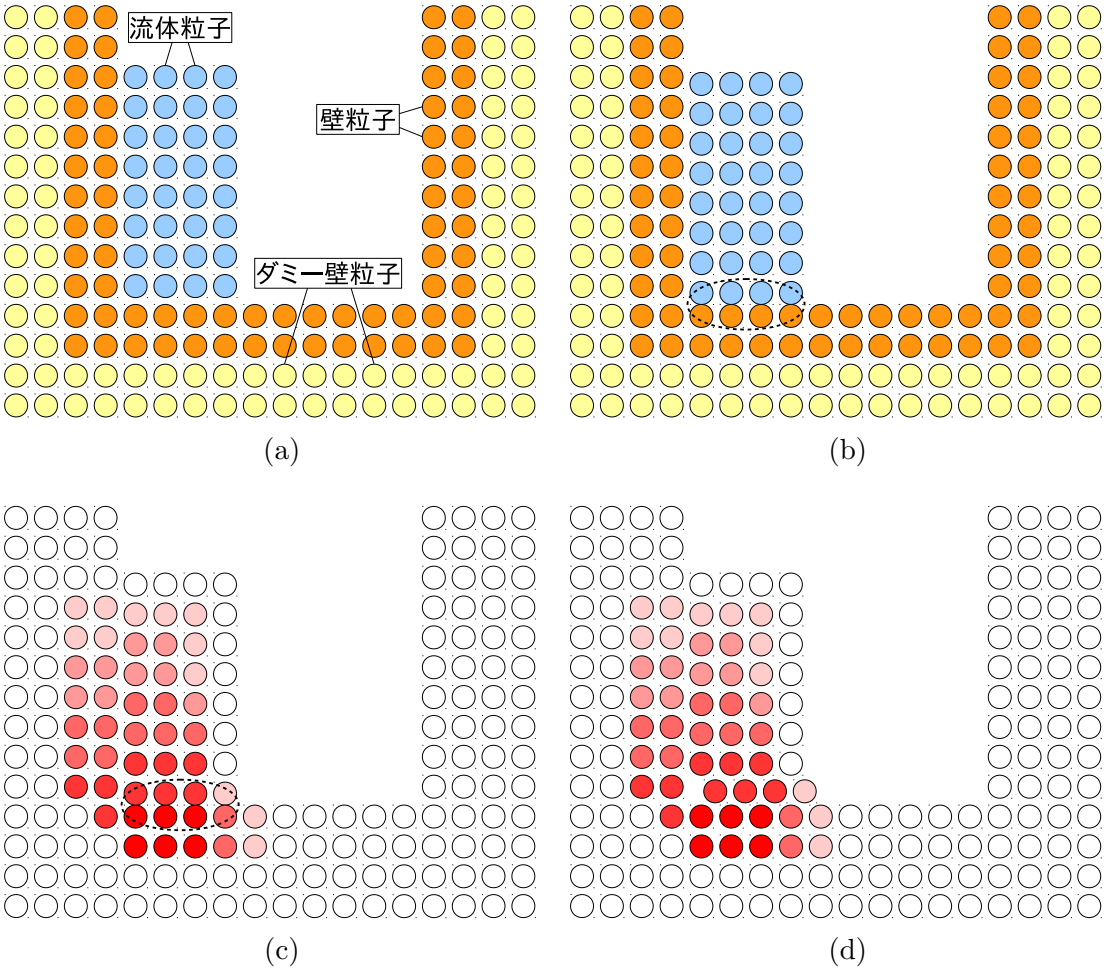


図 8: MPS 法を用いた粒子の移動方法。(a)、(b) の粒子の色は粒子の種類を、(c)、(d) の粒子の色は圧力を表している。(a) 重力項と粘性項の加速度で粒子の仮の速度を計算する。(b) 仮の速度で粒子を仮の位置へ動かす。この時、破線で囲んだ部分では、非圧縮条件を満たしておらず密度が高い。(c) 非圧縮条件を満たすように圧力を計算する。(d) 圧力勾配により加速度を求めて、粒子の速度と位置を修正する。

ρ^0 は密度を表し、非圧縮性流体なので常に一定である。式 (32) を半陰的アルゴリズムに沿って計算する (図 7)。具体的には、次の 2 つの式を 2 段階に分けて解く必要がある。

$$\frac{\boldsymbol{u}_i^* - \boldsymbol{u}_i^k}{\Delta t} = \nu \left\langle \nabla^2 \boldsymbol{u} \right\rangle_i^k + \boldsymbol{g}^k \tag{33}$$

$$\frac{\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^*}{\Delta t} = -\frac{1}{\rho^0} \left\langle \nabla P \right\rangle_i^{k+1} \tag{34}$$

最初に、式 (33) の重力項と粘性項を陽的に計算し、粒子の仮の速度 $\boldsymbol{u}_i^*$ を求める (図 8a)。

$$\boldsymbol{u}_i^* = \boldsymbol{u}_i^k + \Delta t \left[\nu \left\langle \nabla^2 \boldsymbol{u} \right\rangle_i^k + \boldsymbol{g}^k \right] \tag{35}$$

式 (35) の右辺の計算には、時刻 t^k の値しか使われていないため、代入するだけで仮の速度が求められる。ここで、粘性項の計算にラプラシアンが含まれているので、式 (25) のラプラシアンモデルを適用して計算を行う。

$$\left\langle \nabla^2 \boldsymbol{u} \right\rangle_i^k = \frac{2d}{\lambda^0 n^0} \sum_{j \neq i} \left[(\boldsymbol{u}_j^k - \boldsymbol{u}_i^k) w(|\boldsymbol{r}_j^k - \boldsymbol{r}_i^k|) \right] \tag{36}$$

流体の密度は粒子数密度に比例しているという事が言えるため、連続の式 (29) の密度が一定であるという条件を、粒子数密度が一定である条件に置き換えて計算を行っている。この一定にしなければならない粒子数密度を n^0 とする。

式 (35) で計算した仮の速度 $\boldsymbol{u}_i^*$ を代入して、仮の位置 $\boldsymbol{r}_i^*$ を求める (図 8b)。

$$\boldsymbol{r}_i^* = \boldsymbol{r}_i^k + \boldsymbol{u}_i^* \Delta t \tag{37}$$

陽的な部分の計算が終了した時点の $\boldsymbol{r}_i^*$ での粒子数密度 n_i^* は、非圧縮性を考慮した計算を行っていないので、一定にすべき粒子数密度 n^0 になっていない。よって、次の陰的な部分で粒子数密度を n^0 に修正しなければならない。これによって、粒子の位置、速度、圧力も修正され、次の時刻 t^{k+1} の値が確定する事となる。

$$n_i^{k+1} = n^0 = n_i^* + n_i' \tag{38}$$

n_i' は、非圧縮性を満たしていない n_i^* を n^0 にするための粒子数密度の修正量である。最初に、速度の修正について考える。

$$\boldsymbol{u}_i^{k+1} = \boldsymbol{u}_i^* + \boldsymbol{u}_i' \tag{39}$$

$\boldsymbol{u}'$ は速度の修正量である。仮の速度 $\boldsymbol{u}_i^*$ は非圧縮性を満たしていないので、$\boldsymbol{u}'$ による速度の修正を行う。

速度の修正量が陰的な圧力項の計算によって生じるとすると、式 (34) に式 (39) を代入して

$$\boldsymbol{u}_i' = -\frac{\Delta t}{\rho^0} \left\langle \nabla P \right\rangle_i^{k+1} \tag{40}$$

流体の質量保存則の式において

$$\frac{D\rho_i}{Dt} + \rho_i \nabla \cdot \boldsymbol{u}_i' = 0 \tag{41}$$

左辺第 2 項の ρ_i を一定値 ρ^0 で近似すると

$$\frac{D\rho_i}{Dt} + \rho^0 \nabla \cdot \boldsymbol{u}_i' = 0 \tag{42}$$

となる。この式の両辺を ρ^0 で割る。加えて、流体の密度は粒子数密度に比例しているので、$\rho_i = n_i$、$\rho^0 = n^0$ と書き換えられる。

$$\frac{1}{n^0}\frac{Dn_i}{Dt} + \nabla \cdot \boldsymbol{u}_i' = 0 \tag{43}$$

時間に対して離散化すると

$$\frac{1}{n^0}\frac{n_i^* - n^0}{\Delta t} + \nabla \cdot \boldsymbol{u}_i' = 0 \tag{44}$$

となり、式 (40) の両辺に ∇ を掛けて、式 (44) を代入し、式 (38) を用いて書き換えると、圧力のポアソン方程式を得ることができる。

$$\left\langle \nabla^2 P \right\rangle_i^{k+1} = -\frac{\rho^0}{\Delta t^2}\frac{n_i^* - n^0}{n^0} \tag{45}$$

この式 (45) の左辺に、式 (25) の各粒子におけるラプラシアンモデルを適用すると

$$\left\langle \nabla^2 P \right\rangle_i^{k+1} = \frac{2d}{\lambda^0 n^0} \sum_{j \neq i} \left[(P_j^{k+1} - P_i^{k+1}) w(|\boldsymbol{r}_j^* - \boldsymbol{r}_i^*|) \right] \tag{46}$$

となり、P_i^{k+1} に対する連立 1 次方程式を得ることができる。この時, 重み関数の計算に使う粒子の座標は、陽的に計算した仮の粒子の座標 $\boldsymbol{r}_i^*$ を用いる。この連立方程式を解くと、次の時刻 $k+1$ の圧力が求まる (図 8c)。連立 1 次方程式の解法には、CG 法 (Conjugate Gradient Method) や ICCG 法 (Incomplete Cholesky Conjugate Gradient method) を使うことができる。この圧力を式 (40) に代入すると速度の修正量を求められ、更にその値を式 (39) に代入すると次の時刻の粒子の速度が求められる。最後に

$$\boldsymbol{r}_i^{k+1} = \boldsymbol{r}_i^* + (\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^*)\Delta t \tag{47}$$

に代入すると次の時刻の粒子の位置が求められる (図 8d)。式 (47) は次のように導出した。

$$\begin{aligned} \boldsymbol{r}_i^{k+1} &= \boldsymbol{r}_i^k + \boldsymbol{u}_i^{k+1}\Delta t \\ &= \boldsymbol{r}_i^k + \left[\boldsymbol{u}_i^* + (\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^*)\right]\Delta t \\ &= (\boldsymbol{r}_i^k + \boldsymbol{u}_i^*\Delta t) + (\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^*)\Delta t \\ &= \boldsymbol{r}_i^* + (\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^*)\Delta t \end{aligned} \tag{48}$$

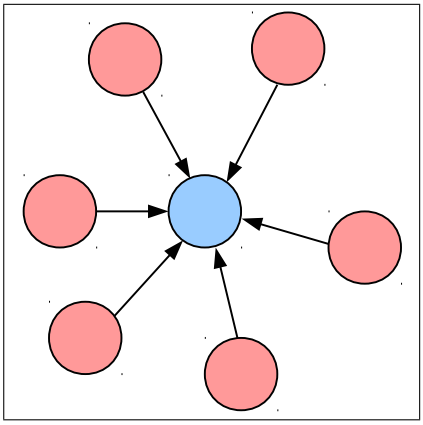


図 9: 局所的に粒子が集中する例

つまり、粒子位置は時間ステップ $k+1$ における速度 $\boldsymbol{u}^{k+1}$ を用いて陰的に計算していることになる。

MPS 法において、近傍の粒子に圧力が低い粒子が存在すると粒子間に引力が働き、圧力の低い粒子に向かって他の粒子が局所的に集まり全体の粒子の配置に偏りが生じる (図 9)。斥力が働くと、粒子同士で互いに反発しあうので全体の粒子の配置が均等になりやすい。よって、圧力の高い粒子の圧力勾配を計算するときに、圧力の低い粒子に対して引力が働かないようにするため、粒子 i における圧力が近傍の粒子 j より常に低ければ良い。そこで、式 (40) の圧力項の計算では、勾配演算子があるため式 (15) の勾配モデルを用いるが、数値安定性の為に勾配モデルに修正を加え、P_i の代わりに $\hat{P}_i$ を用いる次の式を使う。

$$\langle \nabla P \rangle_i^{k+1} = \frac{d}{n^0} \sum_{j \neq i} \left[\frac{(P_j^{k+1} - \hat{P}_i^{k+1})}{|\boldsymbol{r}_j^* - \boldsymbol{r}_i^*|^2} (\boldsymbol{r}_j^* - \boldsymbol{r}_i^*) w(|\boldsymbol{r}_j^* - \boldsymbol{r}_i^*|) \right] \tag{49}$$

$\hat{P}_i^k$ は粒子 i の圧力 P_i^k と、半径 r_e の中にある近傍の粒子 j の圧力 P_j^k の中での最低値である。

$$\hat{P}_i^k = \min_{j \in J}(P_i^k, P_j^k) \tag{50}$$

$$J = \{j : w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \neq 0\} \tag{51}$$

図 10 では、粒子 i の圧力を近傍での圧力の最低値、粒子 j の値で置き換えて圧力勾配を計算するため、粒子 i と粒子 j の間に引力が働かなくなる。図 11 は、粒子 i の圧力が近傍の粒子の中で最も低い場合で、圧力が置き換えられることなく計算され、粒子 i と粒子 j の間に引力が働く。

また、以下の式が満たされていれば、式 (49) は式 (15) と等しくなる。

$$0 = \sum_{j \neq i} \left[\frac{1}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} (\boldsymbol{r}_j - \boldsymbol{r}_i) w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \right] \tag{52}$$

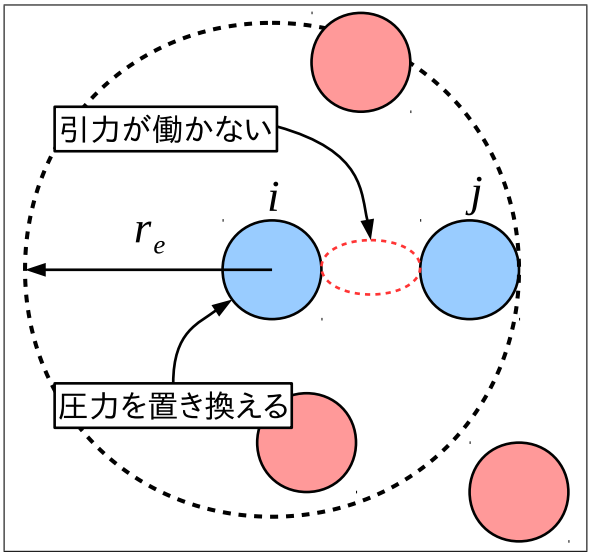


図 10: 粒子 i の圧力が高い時の圧力勾配の計算

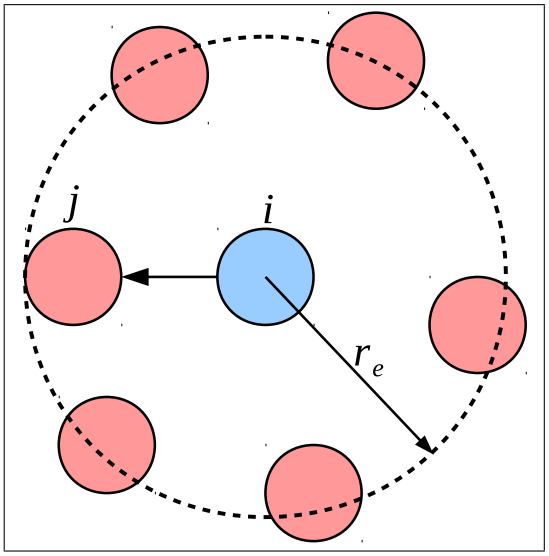


図 11: 粒子 i の圧力が低い時の圧力勾配の計算

式 (1) の重み関数は $r = 0$ で無限大を取るようになっている。このような重み関数にすると、２つの粒子が互いに接近した時、粒子間の距離が短いほど粒子数密度が増加する。すると、粒子数密度が n^0 より大きくなり、圧力が上昇する。そして、粒子間に斥力が働くことになり、粒子同士が重なりあうことが防げる。加えて、全体の粒子の配置に偏りが生じることを防ぐ効果があり、計算の安定化につながる [5]。

2.2.3 時間刻み幅

MPS 法では、数値的な不安定性が生じる対流項の計算を行わずに、粒子の移動を計算しているが、クーラン数によって時間刻み幅が制限される。MPS 法におけるクーラン数は次のように定義される。

$$C_i = \frac{\Delta t u_i}{l_0} \tag{53}$$

C_i は粒子 i における、クーラン数であることを示している。また、u_i は i における速度の絶対値、l_0 は粒子の初期粒子配置における粒子間距離である。

クーラン数の最大値に制限を設けて、数値安定性が維持されるようにする。

$$\max(C_i) = \frac{\Delta t u_i}{l_0} \leq C_{max} \tag{54}$$

時間刻み幅 Δt、粒子間距離 l_0 は一定である。よって、全粒子の中で速度の絶対値が最大の粒子のクーラン数が最大になる。

効率的な計算のために、時間刻み幅が大きい方が良い。したがって、数値安定性を維持しながら、できるだけ大きな時間刻み幅を使うことが求められる。そこで、式 (54) から C_{max} の時の時間刻み幅を考える。

$$\Delta t = \frac{l_0 C_{max}}{u_{max}} \tag{55}$$

時間ステップごとに式 (55) から時間刻み幅を計算して適応するのが望ましい。しかし、他に様々な要因が関係して数値安定性に影響を与えているので、クーラン数の最大値 C_{max} には余裕を持って 0.2 を使っている [7]。

MPS 法では、粘性項の計算を陽的に行っており数値的な不安定性が生じてくる。拡散数を次のように定義する。

$$d_i = \frac{\nu \Delta t}{l_0^2} \tag{56}$$

長さのスケールが大きいシミュレーションでは、クーラン数による数値安定性の影響が大きく、式 (55) を用いて時間刻み幅を決める必要がある。しかし、長さのスケールが小さいシミュレーションでは、拡散数による数値安定性の影響が大きく、拡散数 d_i の最大値を決めて、そこから時間刻み幅を決めると良い。なぜなら、クーラン数では粒子間距離 l_0 の -1 乗に比例しているが、拡散数では -2 乗に比例しているからである。

2.2.4 自由表面

MPS 法では、粒子数密度を用いて自由表面の判定を行う。自由表面の粒子では重み関数の半径 r_e の中に他の粒子が十分に存在せず、粒子数密度が低くなる (図 12)。そこで、MPS 法の陽的な部分の計算が終了した時点で、設定された粒子数密度より小さい粒子が、自由表面上に存在すると判定される。

$$n_i^* < \beta n^0 \tag{57}$$

ここで、β には 1.0 未満の値を用いる。ここで、自由表面上に存在すると判定された粒子には、ディリクレ境界条件を設定する。つまり、圧力のポアソン方程式の計算の際に、粒子 i の圧力 P_i^{k+1} の値をゼロに固定する。この自由表面の判定に使われるパラメータ β には、越塚らの研究によって安定的に計算できるとされる $\beta = 0.97$ の値を用いた [5]。

この自由表面の判定は、粒子数密度を用いた単純なものである。飛沫があがるような初期条件によるシミュレーションでは、他の粒子から孤立する粒子が発生する。孤立した粒子は粒子数密度が低下し、自由表面上に存在すると判定され、圧力がゼロに固定される。そして、孤立した粒子は重力によって自由落下することになる。孤立した粒子が落下し、再び液体内部に取り込まれると、自由表面上にあると判定されなくなり、通常の圧力のポアソン方程式による計算が再開される。

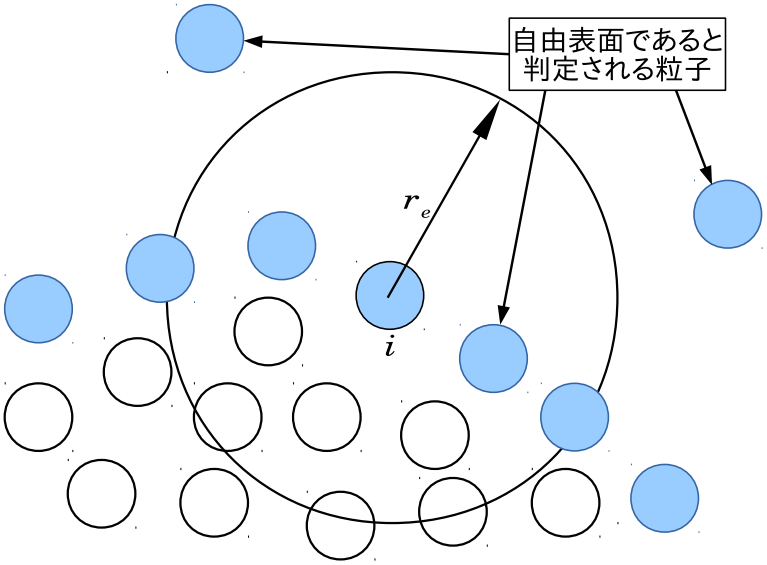


図 12: 自由表面

しかし、単純な判定条件であるために、自由表面の判定が正しく行われない場合がある。例として、自由表面上にある 2 つの粒子が非常に接近して、粒子数密度が大きくなり、双方の粒子が自由表面上に存在しないと判定される。加えて、液体内部の局所的な圧力差によって気泡が発生するキャビテーションが発生するようなシミュレーションの場合は、液体内部で発生した気泡が自由表面であると判定される。

2.2.5 壁境界

壁境界には壁に相当するものとして、座標を固定した動かない粒子を配置する (図 13)。壁となる粒子には 2 種類あり、流体粒子と接する内側の壁粒子では圧力を計算し、外側のダミー壁粒子では圧力を計算しない。内側の壁粒子においても、粒子数密度を計算して圧力を計算するので、外側に粒子が存在しないと式 (57) により、自由表面に存在すると判定される。そこで、ダミー壁粒子を壁粒子の外側に、重み関数の半径 r_e の範囲内に配置することで自由表面と判定される事を防いでいる。MPS 法の陽的な部分の計算が終了した時点で、自由表面の判定がされるため、粘性項の勾配の計算に使われる重み関数 r_e の半径の値によって、ダミー壁粒子が何層必要であるかが決まる。例として、重み関数の半径を粒子間距離 l_0 の 2.1 倍を使っている場合、外側のダミー壁粒子は 2 層必要となる。

圧力のポアソン方程式の計算には、ラプラシアンモデルを使用し圧力に対する係数行列を作成する。この時に、圧力を計算する粒子では、圧力を計算しないダミー壁粒子に対する係数をゼロにする。こうすることで、壁面に対する圧力の境界条件として、圧力勾配がゼロのノイマン境界条件を設定することができる。こ

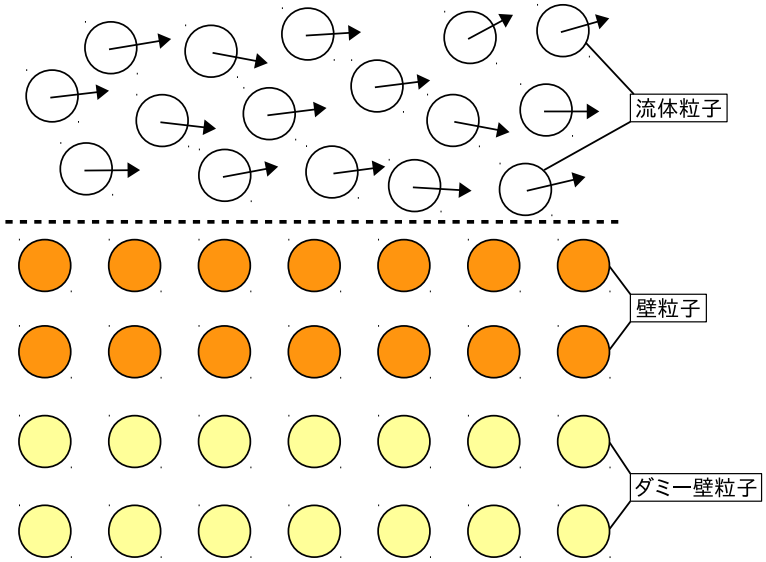


図 13: 壁境界

の条件で、流体粒子の圧力勾配の計算に、壁粒子での圧力を使用することにより、壁に近づいた流体粒子が壁で跳ね返るようになる。この際に、壁粒子の圧力勾配を計算する必要はない。

壁面に対する速度の境界条件として、粘着境界条件と自由境界条件がある。粘着境界条件とは、壁面表面において速度がゼロになるように流体粒子とは逆向きの速度を計算して壁粒子に与えて、粘性項の計算を行うことである。ただし、厳密に粘着境界条件を与える必要がない時には、壁粒子の速度を常にゼロに設定し、粘性項の計算を行う方法も使われる。今回のシミュレーションでは、この方法を用いた。

自由境界条件は、壁面表面で壁と平行な速度成分に対して壁方向の勾配をゼロとし、壁と垂直な速度成分をゼロとしたものである。これを MPS 法で表すには、粘性項の計算の時に、壁粒子またはダミー壁粒子に対する粘性の相互作用を計算しないようゼロにすればよい。

2.3 領域分割

2.3.1 Uniform Grid の概要

MPS 法には重み関数 (1) が使われており、近傍の粒子を探索するために粒子間距離の計算を行う必要がある。図 14 のように単純な方法だと、粒子 i の重み関数を求めるため、赤の破線で囲まれた全ての計算領域において、他の粒子との粒子間距離の計算を行うことになる。つまり、全粒子に対して距離の計算を行うことになる。粒子数を N とすると、$O(N^2)$ で計算量が増えていく。粒子数が少ないシ

ミュレーションであれば現実的な時間で計算することができるが、粒子数を増やしていくと計算量が爆発的に増えて計算が不可能になる。そこで、計算を行う粒子を限定できる方法を取り入れる。

この研究では、Uniform Grid という方法を用いた (図 15)[6]。計算領域をグリッドで分割し、重み関数を求めたい粒子が存在するグリッドとその周辺のグリッドに存在する粒子との粒子間距離の計算に限定する。2 次元空間の場合、図 15 の赤の破線で囲まれた領域、すなわち、9 個のグリッドに存在する粒子と粒子間距離を計算するだけで済む。これにより、遠く離れた粒子との粒子間距離を計算せずに済み、$O(N)$ の計算量に抑えることができる。図のような 2 次元空間だけでなく、同様に 3 次元空間にも適用可能である。

計算領域を分割する時のグリッド幅は、重み関数のパラメータ半径 r_e によって最小値が決まる。シミュレーションを行う際、勾配モデルとラプラシアンモデルで半径 r_e を変えるのが一般的である。例として、勾配モデルで $r_e = 2.1$、ラプラシアンモデルで $r_e = 3.1$ の半径を使用する場合を考える。半径 r_e は粒子間距離 l_0 で規格化されており、実際の計算には、これに粒子間距離 l_0 をかけた値が使われる。つまり、グリッドは粒子間距離 l_0 の 3.1 倍以上の幅を持つ必要がある。なぜなら、3.1 倍以下のグリッド幅では粒子がグリッドの境界付近に存在する時、重み関数の半径内に隣のグリッドにだけでなく、更に隣のグリッドまで入るようになり、隣のグリッドにある粒子だけでは正しい重み関数の値を求められないからである (図 16)。図 16 の赤の破線の計算領域に、重み関数の半径内にあり計算されるべき粒子 3 と粒子 4 が含まれていない。そこで、粒子間相互作用モデルの中で使っている、1 番大きな半径 r_e を基準にグリッド幅を設定する。

2.3.2 プログラムへの実装方法

Uniform Grid のプログラムへの実装には、ソートを使う方法を採用した。GPU への実装を考えた際、ソートを行う GPU ライブラリが存在し、データ構造が単純で、GPU による並列処理に向いている実装方法だからである。

図 15 を例に、最初、粒子の座標からグリッド番号の解決を全粒子について行い、グリッド番号と粒子番号をペアにした粒子リスト配列を作成する (表 1)。次に、粒子リスト配列をグリッド番号で昇順にソートする (表 2)。最後に、ソートされた粒子リスト配列からグリッド配列の Start と End に対応する粒子リスト配列のインデックスを作成する。図 17 を用いて説明する。右の粒子リスト配列の表を上から順にグリッド番号が変化した時の配列のインデックスを見ていく。例えば、グリッド番号が 5 から 6 に変化する時の配列のインデックスが 5 と 6 なので、左の表のグリッド配列のインデックス (グリッド番号) が 5 の End に粒子リスト配列のインデックスの 5、6 の Start に 6 をセットする。再び、右の表を見ていくと、グリッド番号が 6 から 9 に変化する時のインデックスが 8 と 9 なので、左の表のインデックス (グリッド番号) が 6 の End に粒子リスト配列のインデックスの 8、9 の

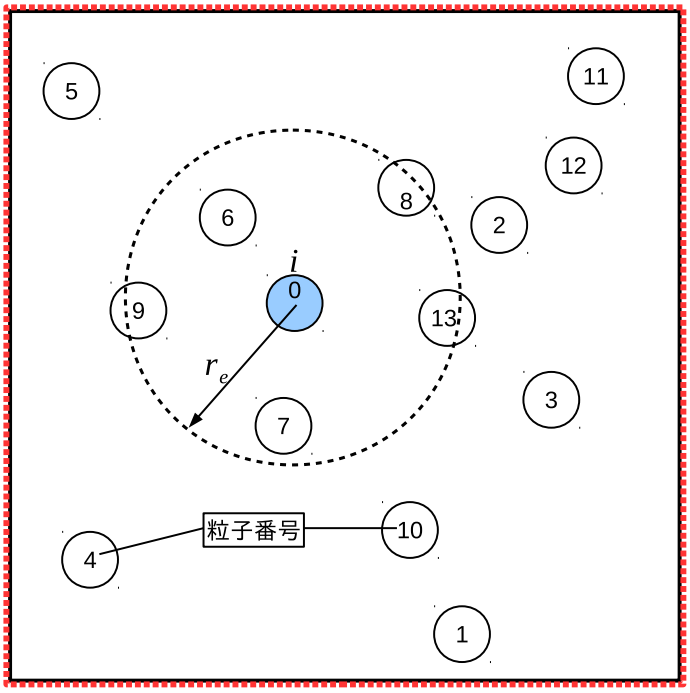


図 14: Uniform Grid を使わない場合の重み関数の計算

Start に 9 をセットする。また、グリッドに粒子が 1 つしかない場合は、左の表のインデックス (グリッド番号) が 12 のように、Start と End に同じ粒子リスト配列のインデックスをセットする。加えて、粒子がないグリッドがある場合を考えて、グリッド配列は、値をセットする前に毎回全て −1 で初期化をしておく。すると、Start が −1 かどうかを判定することにより、そのグリッドに対して粒子の存在を判定できる。

3 CUDA

初期の GPU コンピューティングは、GPU とやり取りをするために OpenGL や DirectX といった標準グラフィックス API を使う必要があり、グラフィックス API によるプログラミング上の制約を受けていたため、GPU に対して画像のレンダリングに見せかけることで汎用計算を行っていた。入力として色、座標の値を計算させたい任意の値に置き換えることで、レンダリングではない計算をレンダリングに見せかけて、GPU に計算させるという非常に手間がかかる方法であった。開発者に本格的に利用してもらうことを考えたときに、たくさんのプログラミング上の制約があり、実際に使うために大きなハードルがあった。

しかし、2006 年に NVIDIA から GeForce 8800GTX が発表されたことにより GPU コンピューティングの環境は一変した。 GeForce 8800GTX は NVIDIA の

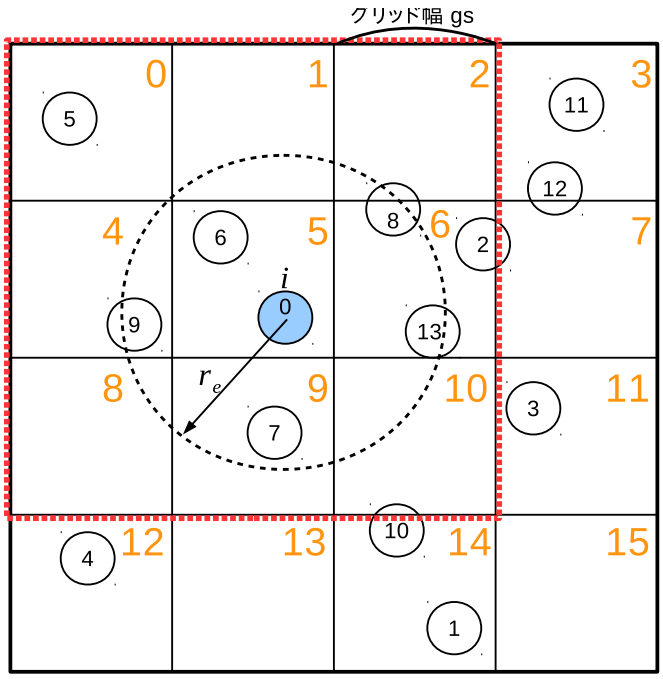


図 15: Uniform Grid を使う場合の重み関数の計算

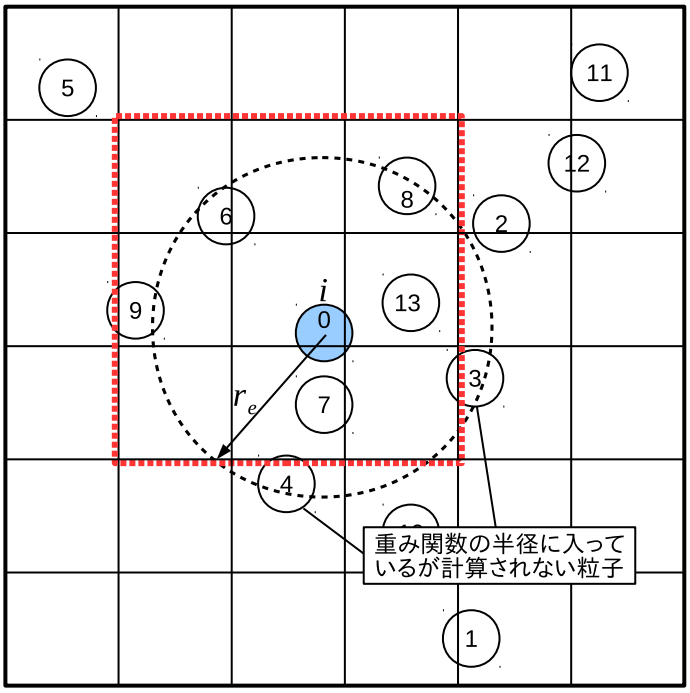


図 16: グリッド幅が半径 r_e より小さい場合の計算例

表 1: 粒子リスト配列

インデックス	グリッド番号	粒子リスト配列
0	5	0
1	14	1
2	6	2
3	11	3
4	12	4
5	0	5
6	5	6
7	9	7
8	6	8
9	4	9
10	14	10
11	3	11
12	3	12
13	6	13

表 2: グリッド番号で昇順にソートされた粒子リスト配列

インデックス	グリッド番号	粒子リスト配列
0	0	5
1	3	11
2	3	12
3	4	9
4	5	0
5	5	6
6	6	2
7	6	8
8	6	13
9	9	7
10	11	3
11	12	4
12	14	1
13	14	10

インデックス	Start	End
0	0	0
1	-1	-1
2	-1	-1
3	1	2
4	3	3
5	4	5
6	6	8
7	-1	-1
8	-1	-1
9	9	9
10	-1	-1
11	10	10
12	11	11
13	-1	-1
14	12	13
15	-1	-1

(a) グリッド配列

インデックス	グリッド番号	粒子番号
0	0	5
1	3	11
2	3	12
3	4	9
4	5	0
5	5	6
6	**6**	2
7	**6**	8
8	**6**	13
9	9	7
10	11	3
11	**12**	4
12	14	1
13	14	10

(b) 粒子リスト配列（表3）

図 17: 粒子リスト配列からグリッド配列の計算

CUDA アーキテクチャで設計された初の GPU で、GPU コンピューティングのため新しいコンポーネントがいくつか含まれ、汎用計算の多くの制限がなくなり、利便性が大きく向上した。また、OpenGL や DirectX を使用したプログラミングでは計算をグラフィックスタスクに装う必要が有りグラフィックの知識が必要だったが、NVIDIA はハードウェアの生産だけでなく、統合開発環境も提供したことにより、その知識も不要となった。これらの Compute Unified Device Architecture（CUDA）と呼ばれる、NVIDIA の GPU において汎用的なプログラムを動かすためのプラットフォーム・統合開発環境が整ったことで GPGPU は広く使われるようになった。 CUDA の特徴として、習得が容易なC言語ベースの拡張言語、自由度の高いメモリアクセス、スレッドスケジューラによる動的スケジューリングによる高い並列性、クロスプラットフォーム（Windows、Mac OS X、Linux)、様々な言語からの呼び出しが可能になったことなどがあげられる。

3.1 GPU アーキテクチャ

NVIDIA の GPU アーキテクチャは世代によって違いがあり、GPU プロセッサの構成に大きな変更が加えられることがある。GPU アーキテクチャは Tesla、Fermi、Kepler と刷新されてきており、現時点での最新 GPU アーキテクチャは Maxwell である。特に、Fermi から Kepler への GPU アーキテクチャの変更は非常に大きなもので、設計思想の根本が変わった。それまでの、パフォーマンス最適化からパフォーマンス効率最適化へと変わった [10]。この変更は、GPU コンピューティングのパフォーマンス向上にも寄与している。このようなアーキテクチャの違いは CUDA のライブラリーが隠蔽してくれるため、GPU アーキテクチャの世代を意識してプログラミングする必要はない。ここでは、研究室が所有する最も新しい GPU の Kepler をベースに、GPU のハードウェアの基本的なユニットの構成について説明する。

図 18 は GPU におけるプロセッサとメモリの構成である。基本的な構成としては、Streaming Multiprocessor（SM）と呼ばれる演算ユニットが搭載されており、Kepler では Streaming Multiprocessor X（SMX）という名称で呼ばれている [11]。このユニットの数は GPU によって異なる。 SMX は CUDA コアと呼ばれるシンプルなプロセッサが 192 個集まって構成され、CUDA コアは計算を行う最小単位と言える。 SMX に入っている CUDA コアは、Fermi と同様に 16 ユニットを 1 グループとしてまとめられている。これは、NVIDIA の GPU がワープという 32 スレッドを最小単位として命令を実行するからである (3.2 節参照)。16 ユニットのグループは、2 クロックサイクルをかけて、32 スレッド (ワープ) の命令を実行する。 この他に SMX には、図 18 に記載してないが、超越関数を計算する Special Function Unit（SFU)、メモリからデータを読み書きするロード/ストアユニット、テクスチャユニット、L1 キャッシュ、L2 キャッシュなどで構成されている。

CUDA コアごとにレジスタがあるが、さらに SMX ごとに SMX 内の CUDA コ

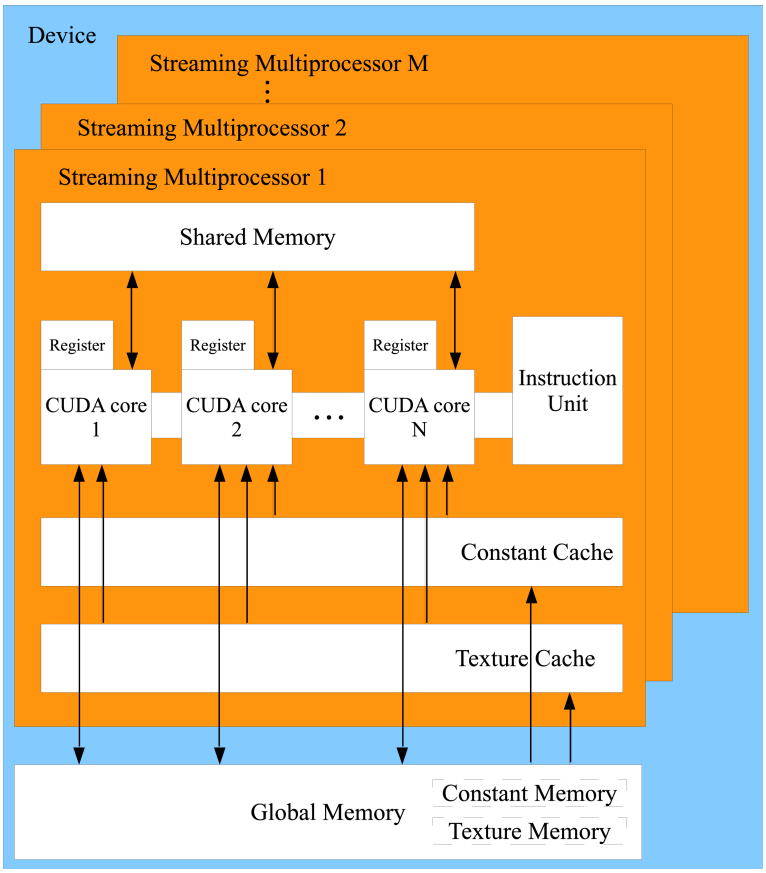


図 18: NVIDIA GPU のハードウェア構成 [9]

アで値が共有されるシェアードメモリが搭載されている。また、SMX 外には大容量のグローバルメモリ、特定の用途に使うことができるコンスタントメモリ、テクスチャメモリが存在する。このように、演算ユニットとメモリが階層性を持っているのが GPU の特徴である (3.3 節参照)。

例として、Kepler の K20X に何個の CUDA コアが搭載されているか計算してみると、14 個の SMX が搭載されているため、CUDA コアの数は 2688 個に達する。更に、NVIDIA の他の世代の GPU の詳細なスペックについて表 3 に示した。GPU 名の括弧内は GPU アーキテクチャの世代を示している。前述したとおり、Kepler 以降、計算の実効効率向上を目指して GPU アーキテクチャの改良が行われているため、演算性能の理論値が同じであっても、新しい世代の方が高い実効性能を発揮できる。また、Fermi の GPU は単精度 (32 ビット) の演算ユニットで構成されており、倍精度 (64 ビット) の値を計算するときに 2 クロックサイクル必要なため、表 3 の倍精度の演算性能が単精度の半分となっている。Kepler では、倍精度演算ユニットを独立させており、CUDA コア 192 個のうちの 64 個が倍精度専用のため、倍精度の演算性能が単精度の 1/3 となる [12]。注意する点として、コンシューマー向け GPU である GeForce は、ハイパフォーマンスコンピューティング向けの

表 3: NVIDIA GPU アーキテクチャの比較

GPU 名	C2075(Fermi)[13]	K20X(Kepler)[12][14]	GTX TITAN X (Maxwell)[15][16]
CUDA コア数	448 cores	2688 cores	3072 cores
プロセッサクロック	1150 MHz	732 MHz	1000 MHz
Streaming Multiprocessor(SM)	14 SMs	14 SMXs	24 SMMs
CUDA コア数 (SP)/SM	16 cores x 2 group	16 cores x 12 group	32 cores x 4 group
特殊関数 UNIT/SM	4 units	32 units	32 units
WARP スケジューラ/SM	2 units	4 units	4 units
共有メモリ/SM	16 KB or 48 KB[1)]	16 KB or 48 KB[1)]	96 KB
L1 キャッシュ/SM	48 KB or 16 KB[1)]	48 KB or 16 KB[1)]	48 KB KB[2)]
L2 キャッシュ/SM	768 KB	1536 KB	3072 KB
ECC メモリ機能	○	○	×
メモリインタフェース	384 bits	384 bits	384 bits
メモリテクノロジ	GDDR5	GDDR5	GDDR5
Load/Store アドレス幅	64 bit	64 bit	64 bit
メモリバンド幅	144 GB/s	250 GB/s	336.5 GB/s
GFLOPS（単精度）	1030 GFLOPS	3935 GFLOPS	6144 GFLOPS
GFLOPS（倍精度）	515 GFOPS	1312 GFLOPS	192 GFOPS

1) Fermi、Kepler 世代では共有メモリ 16KB/L1 キャッシュ48KB または共有メモリ 48KB/L1 キャッシュ16KB に構成を変更できる
2) Maxwell 世代では L1 キャッシュとテクスチャキャッシュは共通

Tesla との差別化のため、倍精度の演算性能が単精度の半分以下となる。

GPU は CPU と異なり物理コア数よりも高い並列度での処理に適した設計となっている。しかし、CUDA コアはマルチコア CPU の CPU コアのように独立して演算を行うことができない。 CPU で例えると、CUDA コアは SIMD コア、SMX は CPU コアに対応している。

複数の CUDA コアがそれぞれ異なるデータに対して同じ演算を行うデータの並列処理が GPU の並列処理である。また、CUDA コアは分岐予測器を持たないシンプルなコアなので、SMX は同じクロックの CPU コアと比べると演算性能が低く、複雑な分岐処理を行わず、並列度の高い問題でないと高い性能を発揮できない。SMX 単位での並列処理を考えた場合、SMX ごとに異なる演算を同時に行うことができる。しかし、異なる SMX 間の CUDA コア同士で同期をとるために、計算結果をグローバルメモリに書き出し GPU カーネルを終了して CPU に制御を戻さなければならない。一方、同じ SMX 内の CUDA コア同士ではシェアードメモ

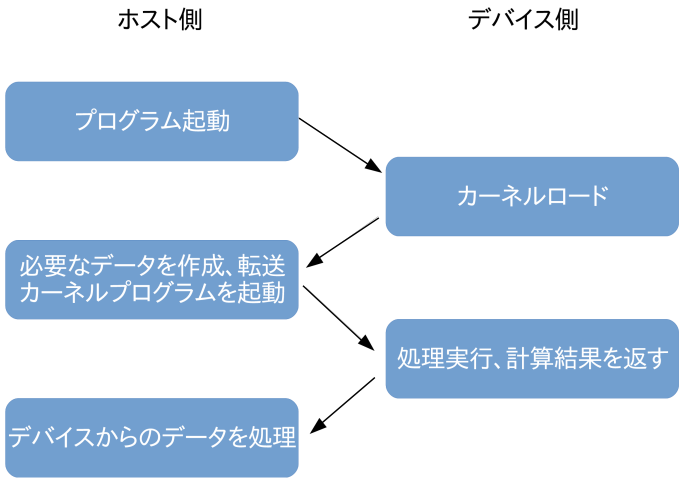


図 19: ホストとデバイスの処理の流れ

リを使い同期を取ることができるため、各種メモリを場合によって使い分けることが効率の良いプログラムを書く上で必要になる (3.3 節参照)。

3.2 プログラミングモデルと実行モデル

GPU コンピューティングでは CPU、GPU の両方で計算を行う。 CPU やメインメモリ側を「ホスト」、GPU やビデオメモリ側を「デバイス」と呼ぶ。 CUDA においてホスト側で実行させるプログラムを「ホストプログラム」、デバイス側で実行させるプログラムを「カーネル」と呼ぶ。プログラムの処理の流れは図 19 に示した。

ここで重要なのは、ホストとデバイスのメモリ転送に時間がかかるということである。メインメモリとデバイス側のメモリの間の転送速度は GPU 内のメモリの転送速度と比較して遅く、可能な限り GPU 内だけで処理完結するようにプログラミングする必要がある。

CPU が CPU コア数以上のプロセスを割り当てられると時分割実行を行うのと同様に、GPU も CUDA コア数、SMX 数以上のプロセスを割り当てられると時分割処理を行う。 CPU がコア数より多くのプロセスを実行しようとした場合、一般的にコア数以下のプロセス数の場合と比べて性能が低下する。一方、GPU はある処理からある処理に移行する場合、プロセッサの状態（コンテキスト）を保存して、さらには復元するコンテキストスイッチのコストが非常に低く、GPU のスケジューラはメモリを読み書きするときに CUDA コアが処理を待つ必要がある場合に積極的にコンテキストスイッチを行う。そのため CUDA コア数や SMX 数より

多くのプロセスを生成したほうがメモリアクセスのレイテンシが隠蔽されて高い性能を得ることができる。

CUDA プログラムにはスレッド・ブロック（スレッドブロック）・グリッドという概念がある (図 20)。

スレッドは、カーネルを動作させた時の多数のプログラムの最小単位を指す。CPU でも使われる単語であるが、CPU ではコア数とほぼ同数のスレッドが動作するのに対して、GPU はコアに対しはるかに多いスレッドが並列に動作することで、性能を引き出している。 CUDA コア単位のジョブがスレッドである。スレッドはホスト側から起動されて各 CUDA コアで同じ処理が行われるが、カーネルの呼び出しのタイミングが 1 つあたり 1 クロックずれるため実行のタイミングはそれぞれ異なる。処理は非同期動作であるが、「`__syncthreads()`」 という関数を使うことにより同じ SMX 内 CUDA コアに関してはスレッドの同期を取ることが可能である。

ブロックはスレッドをまとめたもので、1 つのブロックあたり最大 1024 スレッドが設定できる。「1 ブロックあたり $8 \times 8 \times 8$ スレッド」と x 方向、y 方向、z 方向に三次元的表現で管理することができる。また、「1 ブロックに 512 スレッド」「1 ブロックに 16×16 スレッド」といったように 1 次元的、2 次元的にも表現できる。SMX 単位のジョブがブロックである。

グリッドはブロックをまとめたもので、ブロックと同じく 3 次元的表現ができる。　x 方向に設定できる最大ブロック数は 2147483647 個、y,z 方向に設定できる最大ブロック数は 65535 個である。カーネルの実行単位がグリッドである。

同一グリッドの 1 ブロックあたりのスレッド数はすべて同じであり、ブロックごとにスレッド数の変更はできない。同一ブロックのスレッドはすべて同じ SMX に割り当てられ、各ブロックはコンテキストスイッチが行われても異なる SMX に割り当てられることはない。

すべてのスレッドは並列に動作すると書いたが、厳密には 32 スレッドごとに動作する。これは、Kepler において 16 個の CUDA コアが 1 クロックサイクルで 1 スレッド動き、32 スレッドごとに処理を行う。これが 1 つの単位となり、ワープと呼ぶ。つまり、1 ブロックあたりのスレッドの処理数は 32 の倍数が望ましい。例えば、1 ブロックあたり 8 スレッドだと残りの 24 個の CUDA コアが余り、50 スレッドだと 32 スレッドを処理した後残り 18 スレッドを次のサイクルで処理を行う際に、14 個の CUDA コアが余り処理に無駄ができるからである。

スレッドの実行は 1 クロックサイクルで 1 スレッド動くため非同期であり、あるスレッドの処理が終わっても別のスレッドは処理中ということがある。そこであるスレッドの結果を参照して、別のスレッドの計算を行いたい場合は「`__syncthreads()`」という同一ブロック内のスレッドに対して動作する関数を使用する。この関数はブロック内のすべてのスレッド処理がその関数に到達するまでスレッドの処理を一時停止させることで、同期を取ることができる。しかし、すべてのスレッドがこの関数のところまで処理を進むまで待つ時間がかかるため、必要に応じて処理

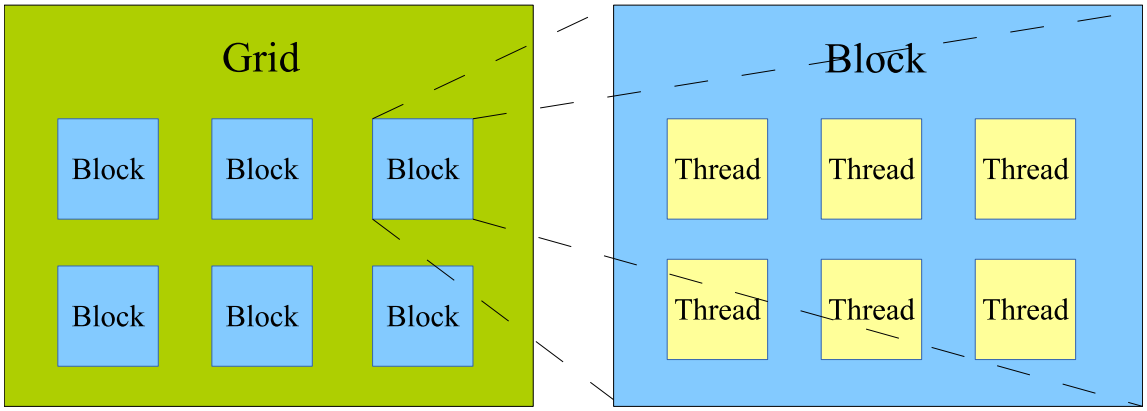


図 20: CUDA のグリッド、ブロック、スレッドの概念図

のどの位置で同期をとるか考えて使う必要がある。

一般的にスレッド数は 128 以上割り当てると良いとされるが、各スレッドのどれくらいの演算量、メモリを使うかによって最適な値は変わるため、性能を引き出すためにはカーネル毎にスレッド数を調整する必要がある。

3.3 メモリの種類

GPU には複数種類のメモリが搭載されており、それぞれ異なる特性を持っている。メモリの搭載位置、転送速度の比較は図 21 に示した。

レジスタ

各 CUDA コアごとに独立して一番近い所に搭載されている。低レイテンシで最も高速にアクセスできるメモリ。 4Byte 単位で構成されており、カーネル関数で宣言された通常変数は基本的にレジスタに割り当てられる。スレッド同士での共有はできない。

シェアードメモリ

各 SMX ごとに独立して搭載されているメモリで、デフォルトでは 16KB の容量がある。 Fermi、Kepler では、L1 キャッシュとメモリを共用しているため、プログラム上の設定から容量の比率をシェアードメモリ 16KB、L1 キャッシュ48KB かシェアードメモリ 48KB、L1 キャッシュ16KB に切り替えることができる。同一 SMX 内の各スレッドから共有メモリとして利用できる。レジスタほどではないが高速で低レイテンシ。 CPU から直接読み書きすることはできない。メモリがバンクに分かれており、同時に複数のスレッドが同一のバンクにアクセスすると性能が低下するバンクコンフリクトが起きるため、最適化の際には注意が必要である (3.4.2 節参照)。

グローバルメモリ

GPU 全体で共有される大容量メモリ。全ブロック、全スレッドから共有メモリとして利用できる。 GPU 上ではなく DRAM に置かれており、ビデオメモリの容量分利用可能だが、テクスチャメモリと共用。連続アクセスに対しては高速、ランダムアクセスに対しては低速、いずれも 400〜600 クロックサイクルのレイテンシがある。 CPU から API を介して値を読み書きすることができ、カーネルをまたいでデータを保持できる。

ローカルメモリ

グローバルメモリと同じく DRAM 上に置かれている。スレッド毎のレジスタ数が多すぎる場合にデータを一時的に格納しておくためのメモリ。 Fermi 以降のアーキテクチャではキャッシュが効くため、以前より高速にアクセスできる。

テクスチャメモリ

グローバルメモリをバインドすることで使用する、GPU 全体で共有されるメモリで専用のキャッシュを持っている。全ブロック、全スレッドから共有メモリとして利用可能だが、GPU からは読み込みしかできない。 CPU から API を介して値を書き込むことができ、カーネルをまたいでデータを保持する事が可能。テクスチャユニットを備えており、近傍要素の線形補間などができる。

コンスタントメモリ

GPU 全体で共有されるメモリで、64KB の容量で読み込み専用。 SMX 毎に専用のキャッシュが搭載されている。グローバルメモリよりも高速にアクセスできる。 CPU から API を介して値を書き込むことができ、カーネルをまたいでデータを保持する事が可能。カーネルプログラムの引数や定数を格納するために使用する。

3.4 最適化の手法

3.4.1 コアレッシング

各スレッドがグローバルメモリにランダムアクセスするとアクセスの効率が悪く、400〜600 クロックサイクルのレイテンシがかかる。しかし、グローバルメモリのコアレッシング（Coalescing）を利用することで、メモリへのアクセスを高速化することができる。コアレッシングとは連続したメモリ領域をまとめて転送するという機能である。図 22 のように、ハーフワープの各スレッドが同一のデータサイズにアクセスし、それぞれのアクセスする先が一定サイズのセグメント内に収まる場合にコアレッシングが発生する。具体的には、グローバルメモリへのア

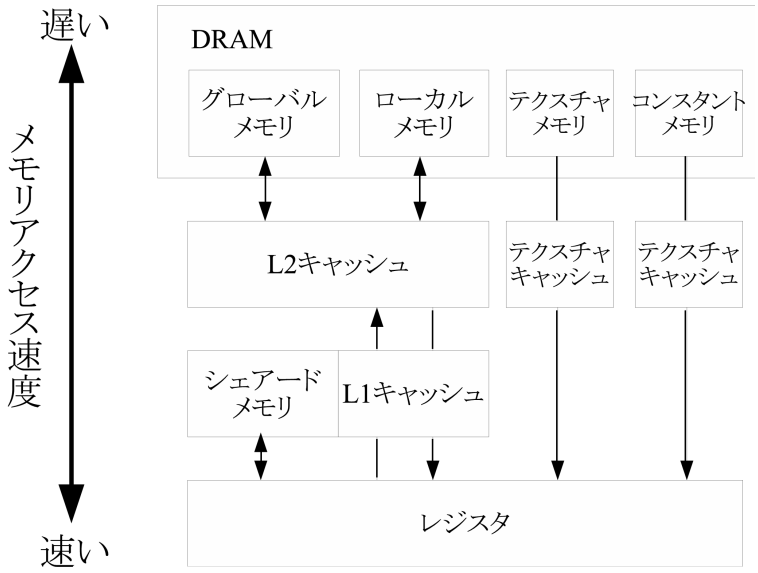


図 21: メモリのアクセス速度とアクセス経路 [18]

クセスは 32 バイト、64 バイト、128 バイト単位で、更に先頭のスレッド（スレッド 0）がそれぞれ 32、64、128 の倍数のアドレスを先頭にしてアクセスする必要がある。領域のデータサイズは 1、2、4、8、16 バイトに対応している。他にも、一部のスレッドがメモリにアクセスしない場合でも、条件が揃えばコアレッシングが発生する（図 23）。

図 24 のようなアクセスでは、アクセス先が一定サイズのセグメントに収まっていてもメモリのアライメントを跨ぎ、別セグメントにアクセスが起きコアレッシングが発生しない。また、図 25 の順番に沿っていないメモリアクセスもコアレッシングが発生しない。

これらのコアレッシングされる条件の他に、古い世代の GPU ではさらに制限がある。新しい世代の GPU になると、コアレッシングを利用しない場合のアクセスの速度が、L1 キャッシュ、L2 キャッシュによって改善されている。

3.4.2 バンクコンフリクト

シェアードメモリは高速にアクセスが可能であり、このメモリを効率的に利用すると性能が飛躍的に向上するが、逆に速度が低下してしまうアクセスパターンが存在する。シェアードメモリは 16 個のバンクによって構成されている。アクセスはハーフワープ単位で行われ、各バンクはシェアードメモリを 4 バイト単位で管理している。バンク 0 では、シェアードメモリのアドレスの 0、64、128 … を管理していることになる。ハーフワープの 16 スレッドがそれぞれ別個のバンクにアクセスすると、16 個のバンクをすべて使い並列処理が可能になる（図 26、図 27）。

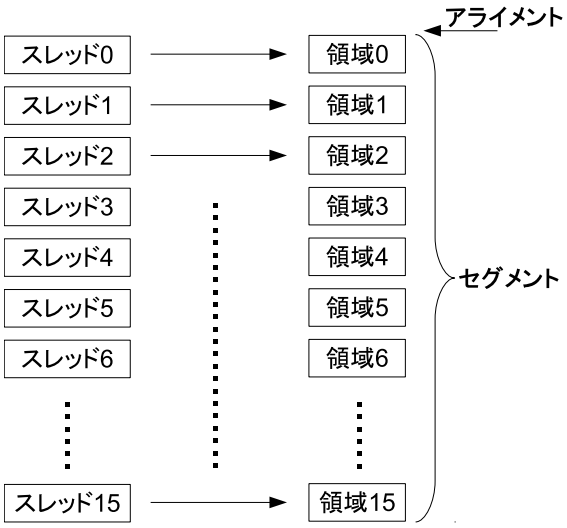


図 22: コアレッシングが発生するメモリアクセス

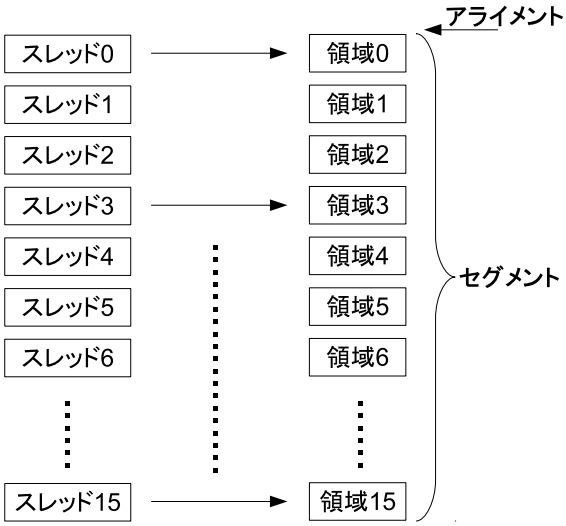


図 23: アクセスを行わない領域があるが、コアレッシングが発生するメモリアクセス

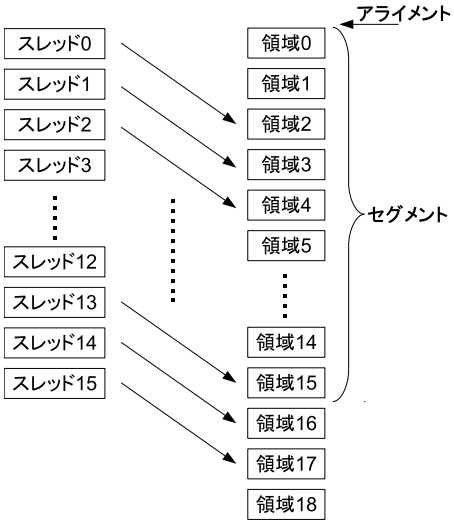


図 24: 別セグメントにアクセスがあり、コアレッシングが発生しないメモリアクセス

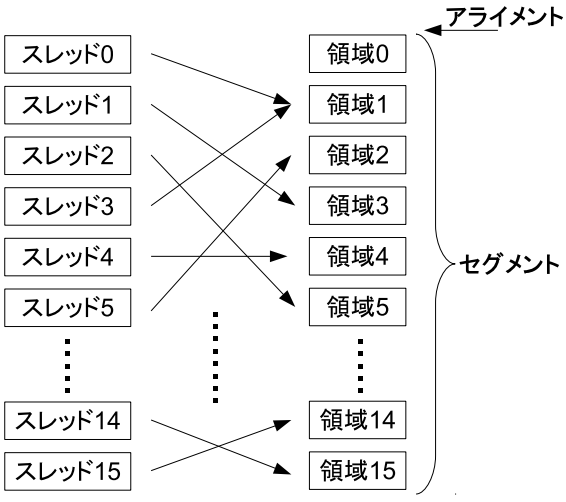


図 25: 順番に沿っていないので、コアレッシングが発生しないメモリアクセス

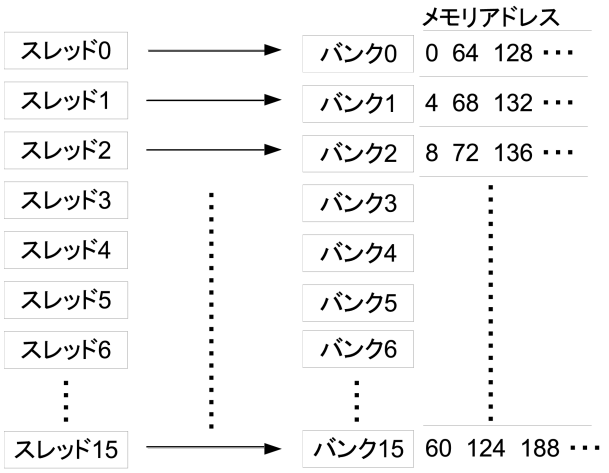


図 26: バンクコンフリクトが発生しないメモリアクセス

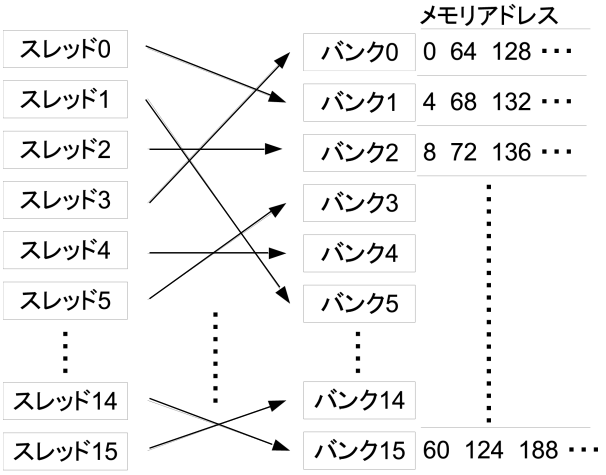


図 27: 順番に沿っていないが、バンクコンフリクトが発生しないメモリアクセス

しかし、複数スレッドが同一のアドレスにアクセスすると、アクセス先のバンクに衝突が発生する (図 28)。これをバンクコンフリクトという。バンクコンフリクトが発生したバンクでは 1 度に 1 つのアクセスしか処理できす、逐次処理を行うことになりプログラム上のボトルネックとなる。また、同一アドレスでなくても４バイトのデータを 1 つおきにアクセスした場合も、偶数番目のバンクに 2 回のアクセスが発生し、奇数番目のバンクにはアクセスがない状況になり、すべてのバンクを使ったアクセスに比べて倍の時間がかかることになる (図 29)。

例外として、すべてのスレッドが同じデータにアクセスした場合はバンクコンフリクトが発生しない。

3.4.3 Divergent 分岐

GPU の分岐命令は 1 ワープ単位で実行される。よって、ワープ内のスレッドの分岐先が一致するか、しないかで動作が変わる。すべてのスレッドが同じ分岐ならば、分岐先の処理のみ実行される。しかし、スレッドによって分岐が異なる場合は、最初に then のスレッドの処理が実行され、その間 else のスレッドは命令を実行せずに休止する。次に else のスレッドの処理が実行され、then のスレッドは休止する。これを Divergent 分岐と呼ぶ (図 30)。 CPU では、if 文は片方の分岐先の処理しか実行しないが、Divergent 分岐がある GPU では、そのワープが then と else の両方の処理をするため時間がかかってしまう。できるだけ、ワープ内で異なる分岐が起きないようにプログラミングする必要がある。

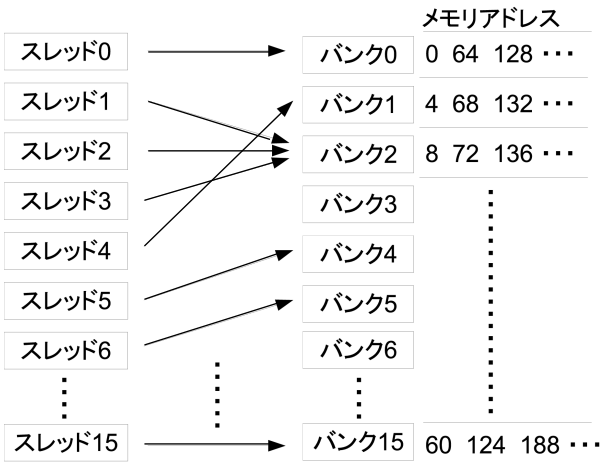


図 28: 同じバンクにアクセスがあり、バンクコンフリクトが発生するメモリアクセス

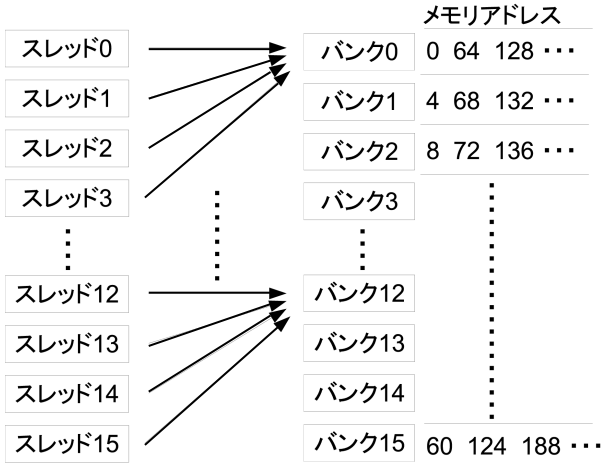


図 29: 偶数番目のバンクにのみアクセスが有り、バンクコンフリクトが発生するメモリアクセス

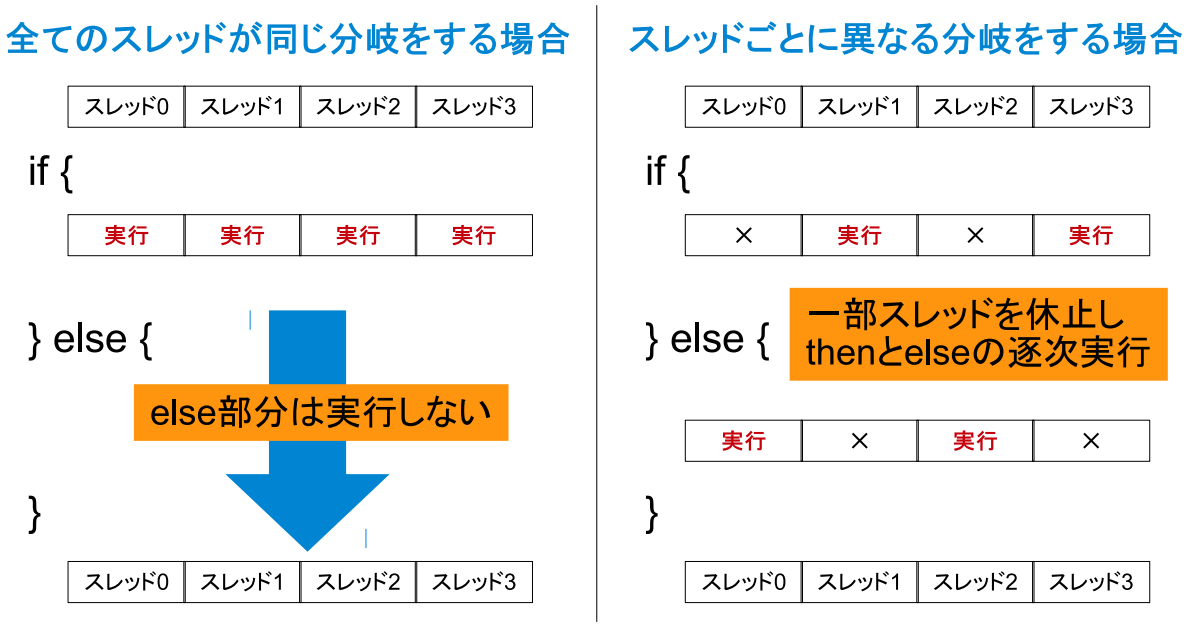


図 30: Divergent 分岐

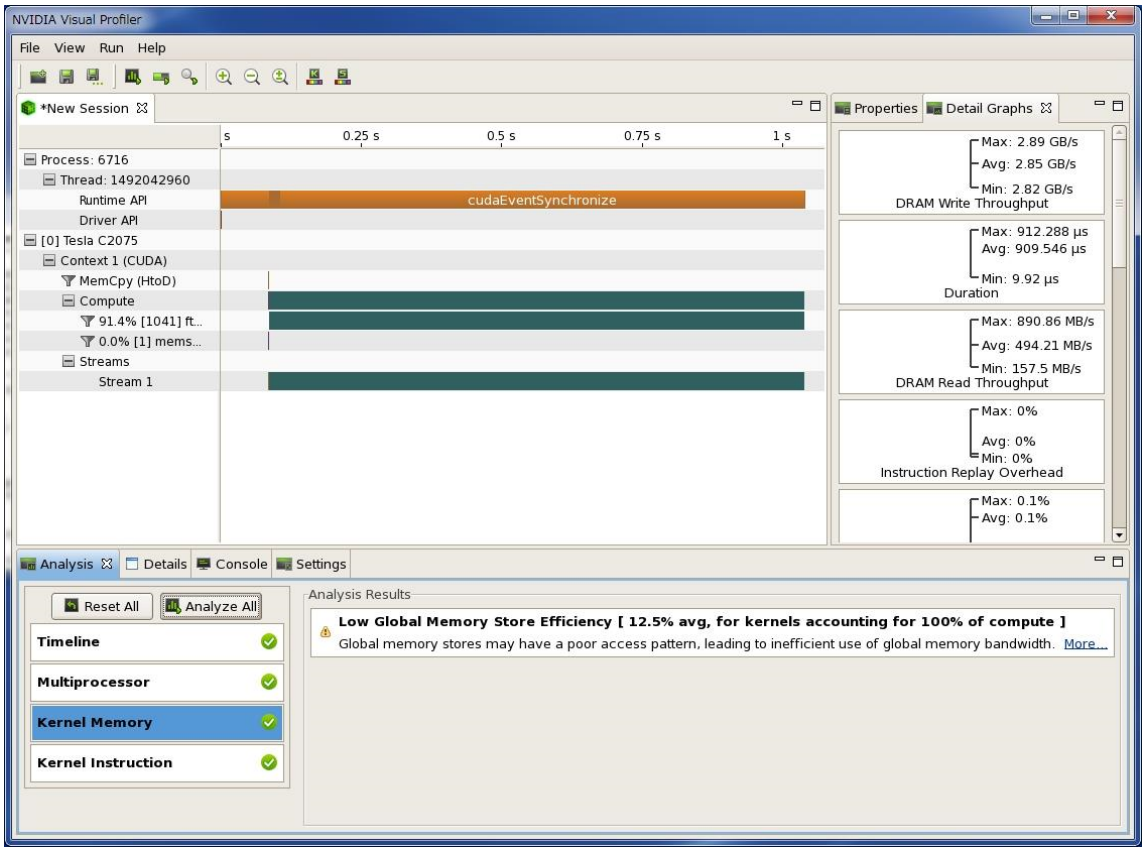


図 31: NVIDIA Visual Profiler の動作画面

3.4.4 CUDA Visual Profiler

CUDA でより速いプログラムを書くためには、プログラムを解析し、ボトルネックとなっている部分を確認してから、プログラムの最適化を行う必要がある。 自分でプログラムを解析するのは簡単なことではない。そこで使用するのが、プロファイリングツールの CUDA Visual Profiler である。 CUDA Visual Profiler は、プログラムを実行し、GPU に組み込まれているパフォーマンスカウンタを調べ、これらのカウンタに基づいてデータを集計し、そのデータに基いてレポートを表示する。カーネルを実行するのにかかった時間だけでなく、レジスタやメモリの使用状況、コアレスアクセスかどうか、バンクコンフリクトや Divergent 分岐の発生状況なども確認できる。様々な形式でグラフ化してプロファイリング結果を見られるようになっており、また最適化すべき部分も指摘してくれる。 CUDA プログラミングにおいて非常に重要なツールである。

4　流体シミュレーションの高速化

4.1　プログラムの主要な関数における計算量

表 4 に、CPU のシミュレーションプログラムの主要な関数についての説明を載せた。CPU と GPU では関数の構成が異なるが、処理の流れは基本的に図 7 の計算アルゴリズムに沿って行われる。Uniform Grid に関する処理は、最初に計算される重力項と粘性項の計算の前に入る。

MPS 法では半陰的アルゴリズムを用いているため、最も計算時間のかかる処理は圧力のポアソン方程式を解く部分となる。粒子数 N の場合、$N \times N$ の係数行列が作られる。しかし、行列の非零要素数が $N\times$(近傍の粒子数) となる疎行列で、かつ対称行列になる。例として、2 次元のシミュレーションにおいてラプラシアンモデルの重み関数の半径 r_e を 3.1 とした時、近傍の粒子数は 20 $\sim$ 30 個になり、行列の要素数は $N \times (20 \sim 30)$ となる。このことから、プログラム全体の計算量はラプラシアンモデルの重み関数の半径によって大きく変わることになる。よって、シミュレーションの際、大きすぎる値を使わないように注意が必要である。このシミュレーションプログラムでは、共役勾配法 (CG 法) を用いて方程式を解いている。この時、CG 法の 1 ステップあたりの計算量は疎行列なので $O(N)$、反復回数は $O(N^{0.5})$ となるため、CG 法全体の計算量は $O(N^{1.5})$ となる。

次に計算時間がかかる処理は、重み関数の計算で必要となる近傍の粒子の探索である。効率化する方法を適用せずに探索を行うと、計算量は $O(N^2)$ となり粒子数が増えると爆発的に計算量が増加していく。そこで、Uniform Grid を使い領域分割を行うことで、計算量は $O(N)$ となる (2.3.1 節参照)。

表 5 に主要な関数についての計算量のオーダーを載せた。重み関数をともなわない関数は、すでに $O(N)$ なので Uniform Grid の適用による計算量の改善はない。表 4 の中で、Uniform Grid による計算量の改善が期待できるのは、重み関数の計算をともなう「cal_viscosity」、「set_coefficient_matrix」、「cal_pressure_gradient」、圧力の最低値の探索を行う「set_minimum_pressure」の 4 つの関数である。以上より、プログラム全体の計算時間が Uniform Grid 適用後では、$O(N^{1.5})$ でスケーリングされると期待できる。

4.2　初期条件

このシミュレーションプログラムの計算時間を計測するために 2 次元の流体シミュレーションの計算を行った。粒子の初期配置は、流体の自由表面の状態を観察できる問題として有名なダム崩壊シミュレーションになるように配置した (図 32)。シミュレーションを行う時、流体粒子が箱から飛び出し計算から除外され、計算される粒子数が減らないようにするため蓋をつけている。

表 4: 関数リスト

関数名	説明
set_uniform_grid	粒子のグリッド番号を計算し粒子リスト配列に格納
sort_grid_index	粒子リスト配列をグリッド番号の昇順にソート
grid_start_end	ソートされた粒子リスト配列からグリッド配列を計算する
cal_gravity	重力項の計算
cal_viscosity	粘性項の計算
move_particle	重力項と粘性項の加速度による粒子の移動
set_boundary_condition	速度、圧力の境界条件の設定
set_coefficient_matrix	圧力のポアソン方程式の係数行列の計算
poisson_solver	反復解法による圧力のポアソン方程式の計算
set_minimum_pressure	近傍の粒子が持つ圧力の最低値の探索
cal_pressure_gradient	圧力勾配の計算
move_particle_pressure_gradient	圧力勾配の加速度による粒子の速度と位置の修正

表 5: Uniform Grid 適用前後の関数のオーダー (*のついてる関数のみに適用)

関数名	適用前	適用後
set_uniform_grid	$O(N)$	$O(N)$
sort_grid_index (quick sort)	$O(N \log N)$	$O(N \log N)$
grid_start_end	$O(N)$	$O(N)$
cal_gravity	$O(N)$	$O(N)$
cal_viscosity*	$O(N^2)$	$O(N)$
move_particle	$O(N)$	$O(N)$
set_boundary_condition	$O(N)$	$O(N)$
set_coefficient_matrix*	$O(N^2)$	$O(N)$
poisson_solver (CG 法)	$O(N^{1.5})$	$O(N^{1.5})$
set_minimum_pressure*	$O(N^2)$	$O(N)$
cal_pressure_gradient*	$O(N^2)$	$O(N)$
move_particle_pressure_gradient	$O(N)$	$O(N)$

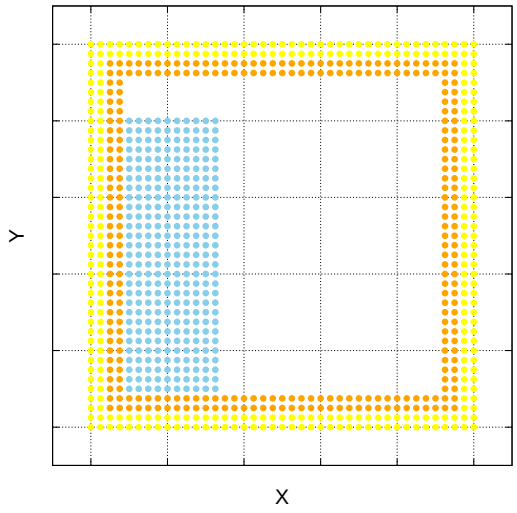


図 32: 粒子の初期配置。水色は流体粒子、橙色は壁粒子、黄色はダミー壁粒子を表す。

パラメータ名	変数名	数値
ReDensity	r_e	2.1
ReGradient	r_e	2.1
ReLaplacian	r_e	3.1
Dimension	d	2
GravityY	g	-9.8
FluidDensity	n^0	1000
GridSize	gs	4.0

表 6: シミュレーションのパラメータ

シミュレーションのパラメータは表 6 に示した。「ReDensity」、「ReGradient」、「ReLaplacian」はそれぞれ、粒子数密度、勾配モデル、ラプラシアンモデルの重み関数の半径 r_e に使われる値である。この値は、2 次元の流体シミュレーションでよく使われる値である [8]。大きな値を使うとシミュレーションの安定性が向上するが計算量も大きく増えてしまう (図 38 参照)。特に、3 次元のシミュレーションでは重み関数の半径内に存在する粒子の数が多いので計算量への影響が更に大きい。しかし、3 次元では近傍の粒子が多いので、2 次元より小さな半径でも安定的に計算できる場合が多い。この半径は、初期の粒子配置における粒子間距離 l_0 で規格化されているため、実際には $2.1l_0$ というように、l_0 をかけた値がシミュレーションでは使われている。

半径として、2.1 のように中途半端な値を用いる理由として、2.0 のような切りの良い値だと、初期の粒子配置で $2.0l_0$ 離れた距離にある粒子との重み関数がゼロになってしまうからである。つまり、半径の境界の外にちょうど粒子が存在することになり、重み関数の計算から除外されてしまう。2.0 でも計算できるが、$2.0l_0$ だけ離れた粒子も重み関数の半径に入れたいので、余裕を持たせて 2.1 のような値を使っている。

「Dimension」は空間次元、「GravityY」は重力である。流体は水を想定しているので、流体密度の「FluidDensity」は 1000kg/m にした。Uniform Grid のグリッド幅を決める「GridSize」は、ラプラシアンモデルの半径から余裕を持たせて 4.0 にした (2.3.1 節参照)。

4.3 シミュレーション例

粒子数が 882 個と 8762 個の場合のシミュレーションを行った。粒子数は、壁粒子とダミー壁粒子を含んだ数である。計算開始から 0.25 ごとの時間で、粒子の座標と速度をプロットした (図 33、図 34)。

図 33 からは、流体の大まかな動きはわかるが、粒子数が少ないため自由表面の形がはっきりと分からない。10 倍近い粒子数でシミュレーションを行うと図 34 のように、自由表面の形をはっきりと観察することができる。このシミュレーションでは、重力による下向きの力が流体の非圧縮性により右へと方向を変え、流体が加速され、壁にぶつかり波や飛沫が発生する様子がよくわかる。

このように、MPS 法において実用的なシミュレーションを行うためには何千何万という多くの粒子が必要であり、また計算量も格子法などと比べて多いため、シミュレーションの高速化は非常に重要である。

4.4 CPU による計算時間の計測

4.4.1 プログラムの種類

最初に CPU 上でのシミュレーションの高速化を行った。それぞれの高速化手法による効果を確認するため、以下に示す 4 種類のプログラムを作成した。

CPUv1 高速化は行っておらず、ガウスの消去法を使用

CPUv2 反復解法ライブラリ lis を使用

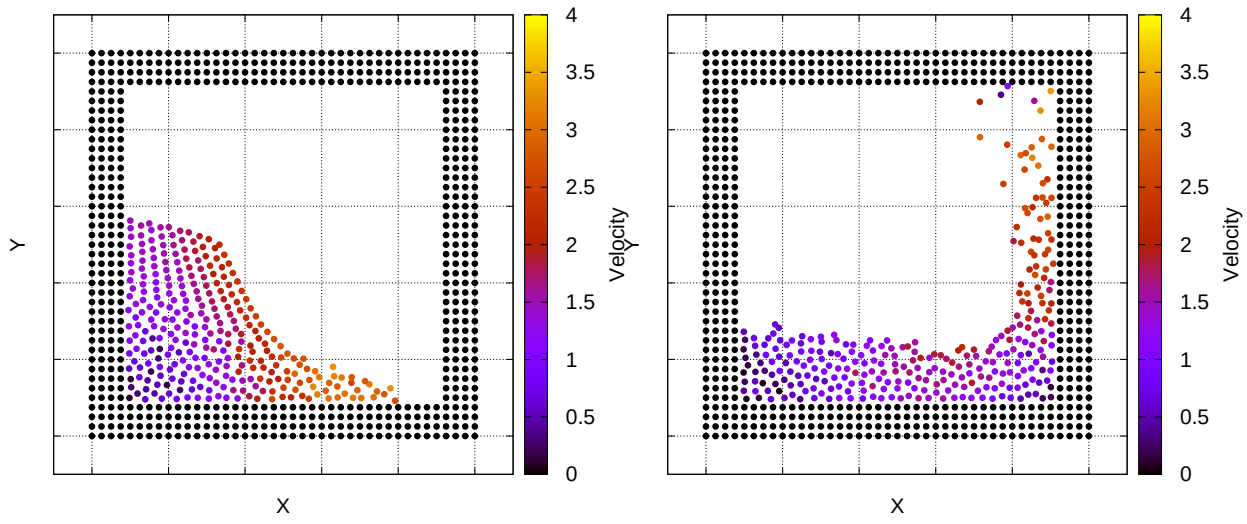
CPUv3 v2 に Uniform Grid を適用

CPUv4 v3 に OpenMP を適用

CPUv1 は高速化を行っていない。加えて、圧力のポアソン方程式を解くためにガウスの消去法を使用しており、MPS 法の動作を確認するためのプログラムである。

CPUv2 は圧力のポアソン方程式を解くために、lis (Library of Iterative Solvers for linear systems) [19] という線形方程式や固有値問題を解く反復解法ライブラリを使用した。また、OpenMP ライブラリを使用することで並列計算も可能であり、このプログラムでは CG 法を使い並列処理で計算をしている。

CPUv3 は CPUv2 に対して、近傍の粒子を効率よく探索するために Uniform Grid を適用したプログラムである。図 7 より、粒子の探索は、勾配モデルによる計算で 2 回、ラプラシアンモデルによる計算で 1 回行われる。しかし、実際のプログラムでは更に、粒子数密度の計算、境界条件のチェック、近傍での圧力の最低値のチェックで粒子の探索が行われている。そこで、それらの処理も Uniform Grid を使用した粒子の探索に書き換えた。



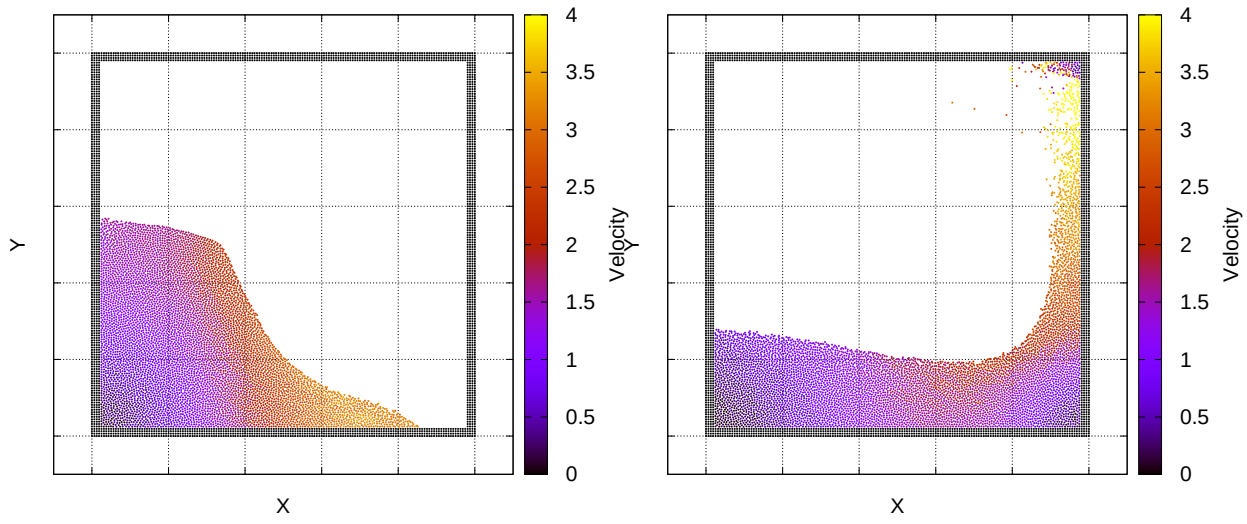
(a) t=0.25

(b) t=0.50

(c) t=0.75

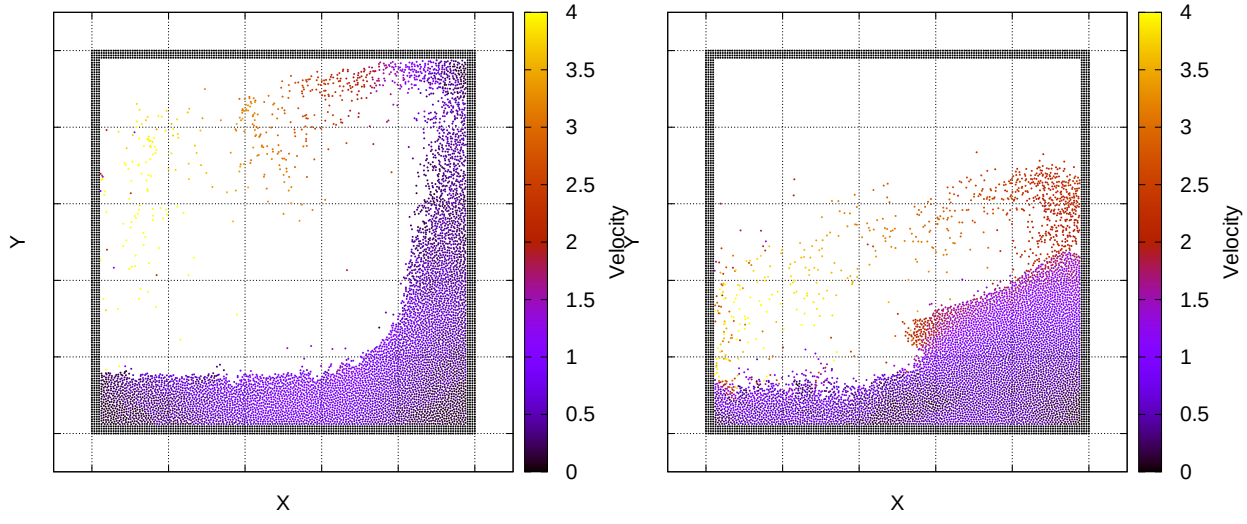
(d) t=1.00

図 33: 882 粒子のシミュレーションによる粒子の位置と速度の時間変化



(a) t=0.25

(b) t=0.50



(c) t=0.75

(d) t=1.00

図 34: 8762 粒子のシミュレーションによる粒子の位置と速度の時間変化

CPUv4 は CPUv3 に対して、OpenMP を適用したプログラムである。lis による計算部分のみが並列化されていたため、他の処理も OpenMP を使い並列化を行った。しかし、lis ライブラリを使った値の代入処理は並列化できなかったため、シングルスレッドで処理を行っている。

CPU は Core i7-3770 を使用し、倍精度浮動小数点演算の理論性能は 217.6 GFLOPS、メモリは DDR3-1600 を使用し、転送速度の理論性能は 25.6 GB/s である。

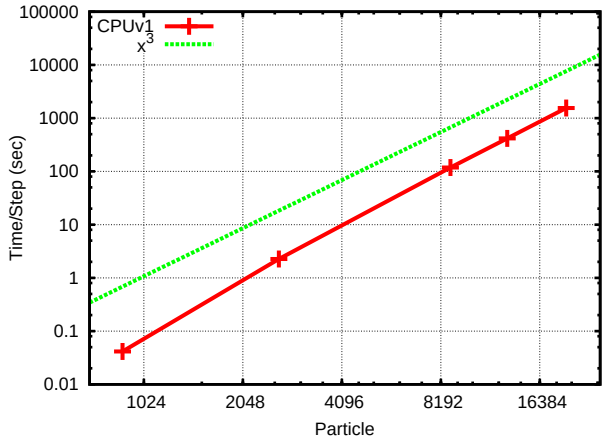
4.4.2 計測結果

図 35 に CPU の各プログラムにおける 1 ステップあたりの計算時間を示した。全てのプログラムにおいてスケーラビリティがあることが確認できた。v1 は、粒子数 N に対する計算時間は粒子の探索にかかる $O(N^2)$ ではなく、$O(N^3)$ でスケーリングされている。これは、CPUv1 で圧力のポアソン方程式を解く際に、$O(N^3)$ のガウスの消去法を使用しているためである。CPUv2 は、圧力のポアソン方程式を lis ライブラリを使い、$O(N^{1.5})$ の CG 法で解いている。よって、近傍の粒子探索にかかる $O(N^2)$ でスケーリングされている。CPUv3 は、Uniform Grid を適用し近傍の粒子探索が $O(N)$ になり、CG 法の $O(N^{1.5})$ でスケーリングされると期待したが、実際には $O(N^2)$ でスケーリングされた。CPUv4 は、Uniform Grid により $O(N^{1.5})$ でスケーリングしており、更に OpenMP で高速化されていることが確認できた。

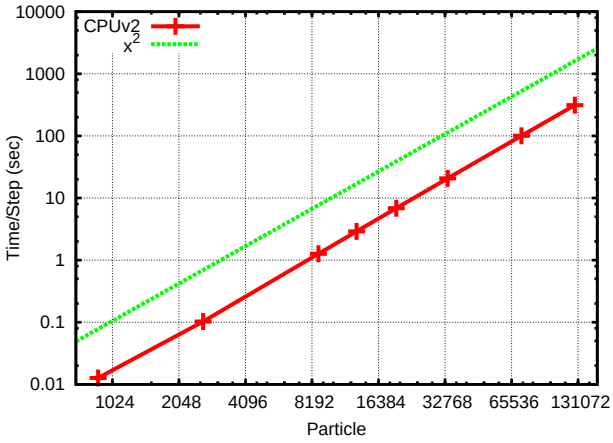
次に、各プログラムの計算時間の比較を行った (図 36)。CPUv2、CPUv3、CPUv4 の順で、各手法を適用するごとに高速化された。粒子数が少ない段階では各プログラムの計算時間の差は小さいが、粒子数が多くなるとプログラムの計算量のオーダーの違いにより、計算時間の差が大きくなることが確認できた。CPUv1 は、他のプログラムと比較して圧倒的に遅いことが分かるので、次の比較対象からは除外する。

CPUv2 を基準とした、CPUv3、CPUv4 の高速化率を図 37 に示した。CPUv3 において、粒子数が 882 個の時は約 2 倍の高速化率で、粒子が増えると高速化率が上がるが、約 3 倍で頭打ちになっている。CPUv4 において、粒子数が 882 個の時は同様に約 2 倍の高速化率だが、粒子数が 13 万個の時は約 46 倍速くなっている。また、高速化率は、CPUv3 とは違い、頭打ちになることはなく粒子数に比例して高くなっていることからも、Uniform Grid の適用による計算量の削減が行われていることがわかる。

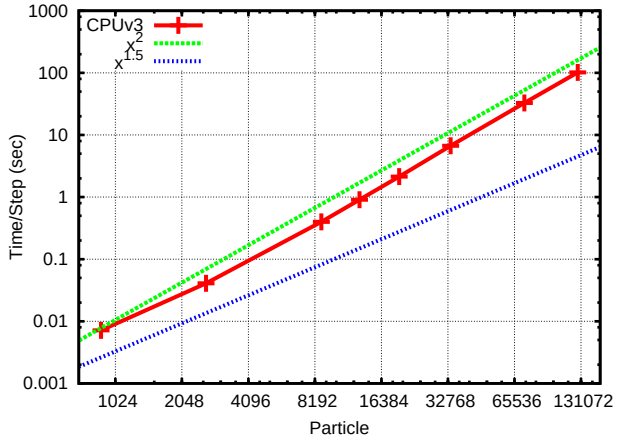
CPUv4 を使いグリッド幅を変化させて 1 ステップあたりの計算時間の測定を行った (図 38)。グリッド幅の値は、粒子間距離 l_0 で規格化されている。計算時間はグリッド幅が大きくなるにつれて増えていった。グリッド幅が大きくなることでグリッド内に存在する粒子が増え、近傍の粒子を探索する際、距離の計算の対象となる粒子が増えた影響のためだと考えられる。つまり、グリッド幅の大きさは重み関数の計算に使われる最も大きい半径を考慮に入れて、できるだけ小さい値を



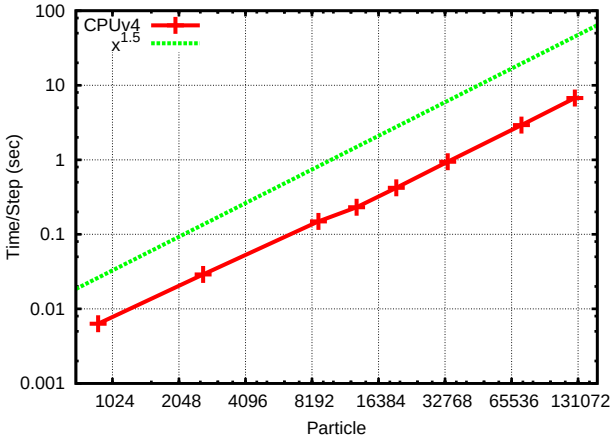
(a) CPUv1



(b) CPUv2



(c) CPUv3



(d) CPUv4

図 35: CPU の各プログラムにおける 1 ステップあたりの計算時間

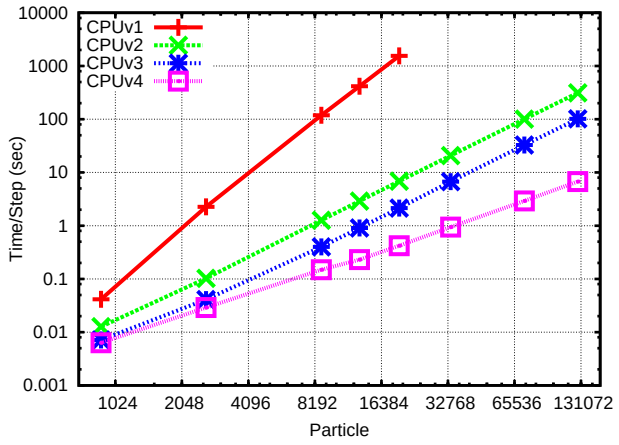


図 36: 各プログラムにおける 1 ステップあたりの計算時間の比較

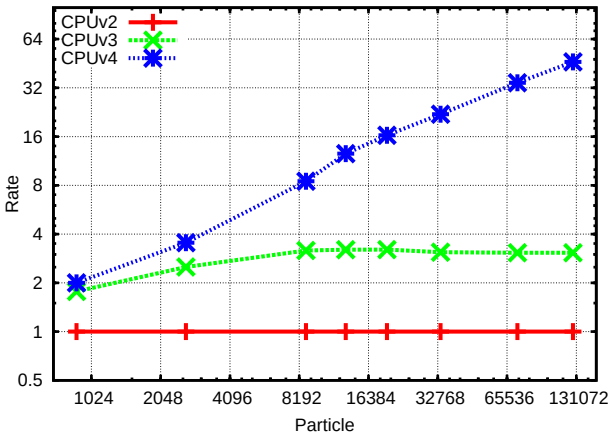


図 37: CPUv2 に対しての高速化率

使うべきだという結論になった。

4.5 GPU による計算時間の計測

4.5.1 プログラムの種類

CPU 上での計測結果を踏まえて、GPU によるシミュレーションの高速化を行った。それぞれの高速化手法による効果を確認するため、以下に示す 2 種類の倍精度で計算するプログラムを作成した。

GPUv1 Uniform Grid を適用

GPUv2 シェアードメモリに粒子の座標の値を格納

GPUv1 は、Uniform Grid を適用するとともに、CPU プログラムから関数を再構成した。CPU では、重力項や粘性項といった計算を 2 つの関数に分けて行っていたが、GPU ではカーネルを終了すると高速にアクセスできるレジスタやシェアードメモリに確保した値を破棄することになる。そこで、確保した値を再利用するために、可能な限り関数を減らし、プログラムで実行するカーネルの数を少なくした。

また、GPU に搭載されている、それぞれ異なる特性を持ったメモリを有効活用するようにプログラミングした。シミュレーションで定数として使われるパラメータは、コンスタントメモリに格納した。また、頻繁に呼び出すことになる粒子の座標や速度といった配列の値は、グローバルメモリから値を読み出すときにコアレッシングが発生するように値を並び替えた。ソートされた粒子リスト配列のイ

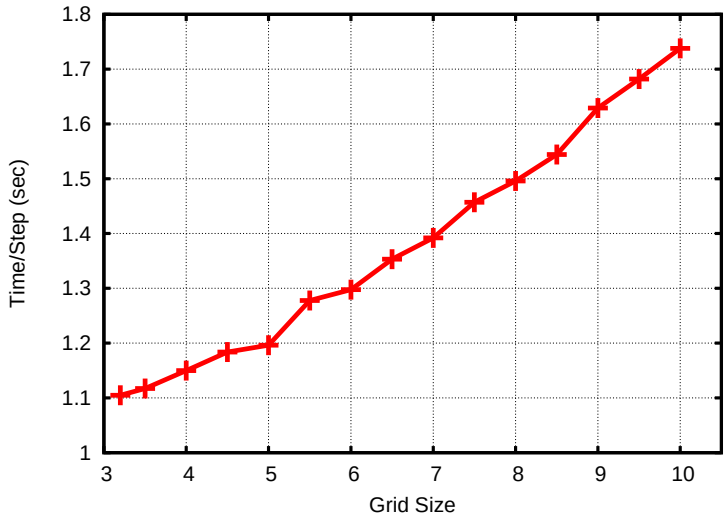


図 38: グリッド幅による 1 ステップあたり計算時間の比較

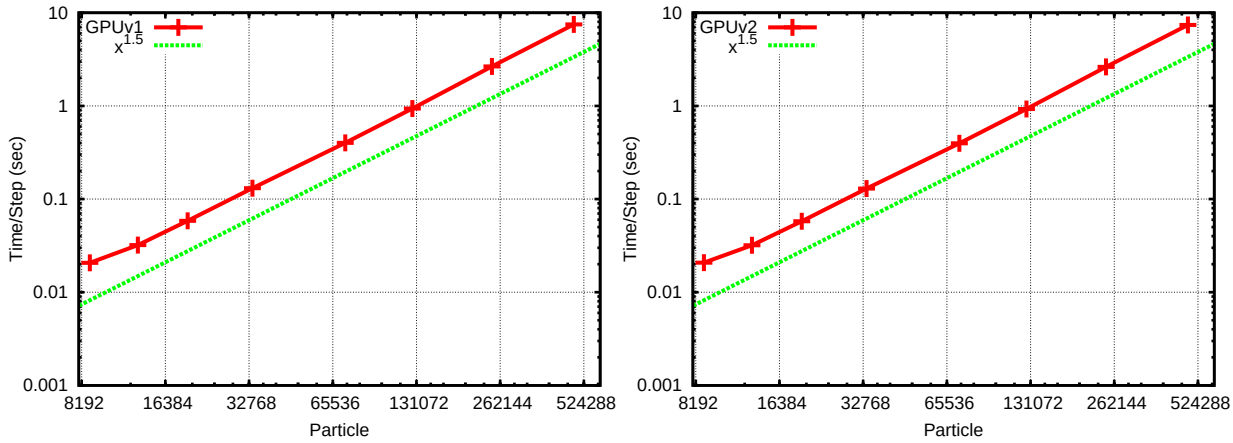
ンデックスに沿って、新しい配列に値を並び替えて格納した。この配列の値は同じグリッドの粒子であれば連続して格納されているため、近傍の粒子の探索などでグリッドごとに値を呼び出す時、コアレッシングが発生する。レジスタ、シェアードメモリについては前述したとおりである。

加えて、分岐処理についても CPU のプログラムから見なおした。GPU では、分岐によっては Divergent 分岐が発生する。しかし、分岐の書き方によって、その発生数は大きく変わってくるので、関数の再構成も踏まえて分岐の書き方や順番を見直した。

圧力のポアソン方程式を解くために、MAGMA (Matrix Algebra on GPU and Multicore Architectures) [20] というライブラリを使用した。

GPUv2 は、シェアードメモリに周辺のグリッドに存在する粒子の座標を格納して計算するプログラミングである。近傍の粒子を探索するために、周辺のグリッドの粒子の座標が頻繁に呼び出される。そこで、1 つのブロックに 1 つのグリッドを割り当て、ブロック内で周辺のグリッドの値をシェアードメモリで共有する。すでに、GPUv1 の段階でグリッドごとに連続して値が格納された配列から読み込んでおりキャッシュも効いていると思われるが、明示的に L1 キャッシュメモリと同じレイテンシのシェアードメモリの値を使うことによる高速化を期待した。

GPU は K20X を使用し、倍精度浮動小数点演算の理論性能は 1312 GFLOPS、メモリは GDDR5 で転送速度の理論性能は 250 GB/s である。



(a) GPUv1　　(b) GPUv2

図 39: GPU の各プログラムにおける 1 ステップあたりの計算時間

4.5.2　計測結果

図 39 に GPU の各プログラムにおける 1 ステップあたりの計算時間を示した。GPU には高い演算能力があり、十分な粒子数のシミュレーションでないとスケーラビリティを確認できない。そこで、粒子数が 8000 個から、計測した計算時間を載せた。GPUv1 と GPUv2 ともに、$O(N^{1.5})$ で計算時間が増加していることが確認できた。

GPUv2 がどのくらい高速化されたかを確認するため、GPUv1 に対しての GPUv2 の高速化率を図 40 に示した。粒子数が少ない場合には、高速化されず逆に遅くなった。粒子数が多くなると、高速化率は増加するが、最終的に約 1%の高速化に留まった。シェアードメモリに値を格納する処理が重かったためだと考えられる。加えて、GPUv1 でコアレッシングが起こるようなアクセスにしていたため、すでに十分高速化されていた可能性がある。つまり、グローバルメモリとのアクセス回数が減り、値の読み込み速度向上の効果に対して、シェアードメモリに粒子の座標を格納する処理のコストが大きく、全体としての高速化が僅かになってしまった。また、速度が向上した処理が全体の計算時間に占める割合が小さいことも 1 つの要因である。しかし、3 次元のシミュレーションでは探索する粒子数が大きく増加するため、シェアードメモリによる高速化の効果が大きくなり、2 次元のシミュレーションより大きな高速化率が見込まれる。

図 41 から各 GPU のプログラムは CPUv4 より計算時間が短く、CPUv4 に対して GPUv2 は約 7 倍の高速化が達成できた。この計測で使用した GPU は CPU と比較して、演算性能は約 6 倍、メモリの転送速度は約 9 倍の差がある。演算性能の差以上に高速化されていることから、CPU のプログラムではメモリの転送速度がボトルネックとなっている。しかし、高速化はメモリ転送速度の差の約 9 倍に達

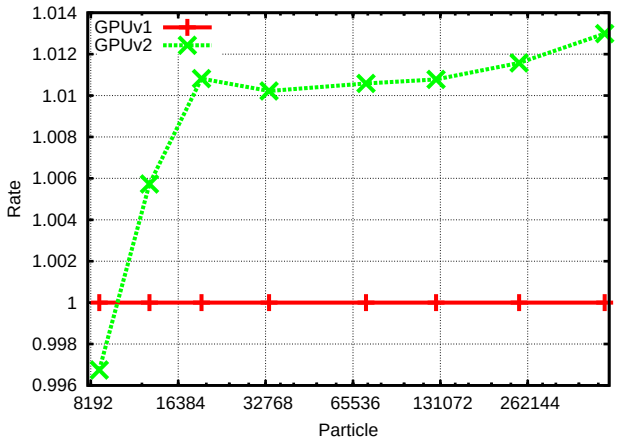


図 40: GPUv1 に対しての GPUv2 の高速化率

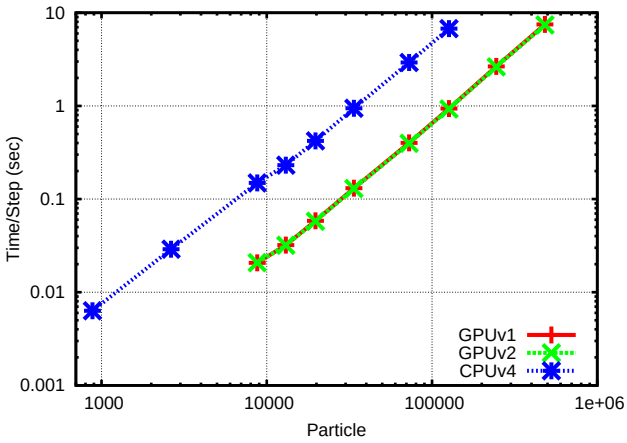


図 41: v4 と GPU の各プログラムの 1 ステップあたりの計算時間の比較

していない。この理由として、このプログラムでは分岐が多数存在することから、Divergent 分岐も多数発生し演算の実効効率が低下したためだと考えられる。

5　まとめ

本研究では、粒子法による流体シミュレーション、特に Uniform Grid を用いた領域分割による近傍にある粒子探索の高速化と CUDA を用いた GPU コンピューティングによるシミュレーションの高速化を行った。最初、CPU のシミュレーションコードで領域分割の実装を行い、その後 GPU のシミュレーションコードを作成した。圧力のポアソン方程式を解くソルバーとして、CPU では lis、GPU では MAGMA を利用した。

2 章で MPS 法の離散化方法を説明し、3 章で非圧縮性流体の計算方法を説明した。高速化方法として、4 章で GPU のアーキテクチャや最適化方法について、5 章で粒子探索の効率化方法の Uniform Grid について説明した。そして、6 章で高速化を行ったプログラムの計算時間の計測結果を示した。

CPU 上のシミュレーションの高速化では、Uniform Grid による領域分割により近傍の粒子探索の計算量が $O(N^2)$ から $O(N)$ になった。そして、全体の計算時間が CG 法の $O(N^{1.5})$ でスケーリングされることを確認した。また、粒子数が 13 万個の時、Uniform Grid の適用に加えて OpenMP による処理の並列化を行ったプログラムの CPUv4 は、CPUv2 に対して約 46 倍の高速化がされた。

GPU 上のシミュレーションの高速化では、GPU に搭載されている様々なメモリを活用したシミュレーションコード GPUv1 を作成した。レジスタやシェアードメモリに格納された値を再利用するために、CPU のシミュレーションコードから関数構成を変更し、できるだけ関数の数を少なくし、必要となるカーネルの数を

減らした。また、グローバルメモリへのアクセスでは、値の再配置によりコアレッシングが起きやすくなるように変更した。更に、シェアードメモリに周辺のグリッドに存在する粒子の座標を格納し、グローバルメモリとのアクセス回数を減らしたプログラム GPUv2 も作成した。このプログラムに関して、シェアードメモリに値を格納する処理のコストが大きく、GPUv1 に対して GPUv2 の高速化の割合は僅かであった。しかし、3 次元のシミュレーションでは探索する粒子数が大きく増加するため、シェアードメモリによる高速化の効果が大きくなり、2 次元のシミュレーションより大きな高速化率が見込まれる。最終的に、粒子数が 13 万個の場合、GPUv2 の計算時間を CPUv4 と比較すると約 7 倍の高速化を達成できた。

A　拡散方程式の解析解の導出

拡散方程式の初期分布の関数 $f(x,t)$ のフーリエ変換を

$$F(k,t) = \int_{-\infty}^{\infty} f(x,t)e^{-ikx}dx \tag{58}$$

とする。拡散方程式 (17) の両辺をフーリエ変換すると左辺は

$$\int_{-\infty}^{\infty} \frac{\partial f}{\partial t}e^{-ikx}dx = \kappa\frac{\partial F}{\partial t} \tag{59}$$

となる。f と $\frac{\partial f}{\partial x}$ が無限遠で 0 になると仮定すると、右辺は

$$\begin{aligned}
\int_{-\infty}^{\infty} \kappa\frac{\partial^2 f}{\partial x^2}e^{-ikx}dx &= \left[\kappa\frac{\partial f}{\partial x}e^{-ikx}\right]_{x=-\infty}^{x=\infty} - \int_{-\infty}^{\infty}\kappa\frac{\partial f}{\partial x}(ike^{-ikx})dx \\
&= -\left[\kappa f ike^{-ikx}\right]_{x=-\infty}^{x=\infty} + \int_{-\infty}^{\infty}\kappa f(-k^2e^{-ikx})dx \\
&= -\kappa k^2\int_{-\infty}^{\infty} fe^{-ikx}dx \\
&= -\kappa k^2 F
\end{aligned} \tag{60}$$

となる。初期分布としてデルタ関数を用いる。

$$f(x,0) = \delta(x) \tag{61}$$

$F(k,0) = 1$ だから、フーリエ振幅 $F(k,t)$ が満たす常微分方程式は

$$\frac{dF}{dt} = -\kappa k^2 F \tag{62}$$

であり、この解は

$$F(k,t) = e^{-\kappa k^2 t} \tag{63}$$

である。これを逆フーリエ変換する。

$$\begin{aligned}
f(x,t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-\kappa k^2 t}e^{-ikx}dk \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} \exp\left\{-\kappa t\left[k^2 + \frac{ix}{\kappa t}k - \frac{x^2}{4(\kappa t)^2}\right] - \frac{x^2}{4\kappa t}\right\}dk \\
&= \frac{1}{2\pi}e^{-\frac{x^2}{4\kappa t}}\int_{-\infty}^{\infty} \exp\left\{-\kappa t\left[k + \frac{ix}{2kt}\right]^2\right\}dk \\
&= \frac{1}{2\pi}e^{-\frac{x^2}{4kt}}\sqrt{\frac{\pi}{\kappa t}} \\
&= \frac{1}{\sqrt{4\pi\kappa t}}e^{-\frac{x^2}{4\kappa t}}
\end{aligned} \tag{64}$$

という解析解が求められる。

B　MPS 法の発散モデル

発散はベクトルに作用してスカラーが得られる演算子である。粒子 i とその近傍に粒子 j が存在し、それぞれ位置ベクトル $\boldsymbol{r}_i$、$\boldsymbol{r}_j$、変数値ベクトル $\boldsymbol{u}_i$、$\boldsymbol{u}_j$ を持っているとする。2 次元の時、$\boldsymbol{u} = (u, v)$ の発散は

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \tag{65}$$

となる。粒子 i と粒子 j の間の、相対位置ベクトル x' の微分は

$$\frac{\partial u}{\partial x'} = \frac{u_j - u_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} \tag{66}$$

となる。発散の計算には x' 成分のみ必要なので

$$\frac{\partial u_{x'}}{\partial x'} = \frac{u_j - u_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} \cdot \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|} \tag{67}$$

となる。しかし、この式は勾配モデルの式 (14) と同様に、相対位置ベクトルの方向の成分しか考慮されてないので、式 (15) のように、垂直な方向の成分の影響も考える。

$$\langle \nabla \cdot u \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \frac{(u_j - u_i) \cdot (\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \tag{68}$$

d は空間次元数である。この式は発散モデルの式とする。

変数ベクトルが粒子 i と粒子 j の間に設定されている時には、$\boldsymbol{r}_i - \boldsymbol{r}_j$ を $\boldsymbol{r}_{ij}$、$\boldsymbol{u}_i - \boldsymbol{u}_j$ を $\boldsymbol{u}_{ij}$ と置き換える。

$$\langle \nabla \cdot u \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \frac{u_{ij} \cdot (\boldsymbol{r}_{ij} - \boldsymbol{r}_i)}{|\boldsymbol{r}_{ij} - \boldsymbol{r}_i|^2} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \tag{69}$$

また、

$$\boldsymbol{r}_{ij} = \frac{\boldsymbol{r}_i + \boldsymbol{r}_j}{2} \tag{70}$$

を式 (69) に代入すると

$$\langle \nabla \cdot u \rangle_i = \frac{2d}{n^0} \sum_{j \neq i} \frac{u_{ij} \cdot (\boldsymbol{r}_j - \boldsymbol{r}_i)}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} w(|\boldsymbol{r}_j - \boldsymbol{r}_i|) \tag{71}$$

となる。

C　ラプラシアンモデルと勾配モデルに発散モデルを適用した式との比較

ラプラシアンは、勾配にさらに発散を作用させたものである。そこで、粒子間で定義される勾配モデルの式 (14) に、発散モデルの式 (71) を適用してみる

$$\begin{aligned}\left\langle \nabla\cdot\langle\nabla\phi\rangle_{ij}\right\rangle_i &= \frac{2d}{\lambda n^0}\frac{\frac{(\phi_j-\phi_i)(\boldsymbol{r}_j-\boldsymbol{r}_i)}{|\boldsymbol{r}_j-\boldsymbol{r}_i|^2}\cdot(\boldsymbol{r}_j-\boldsymbol{r}_i)}{|\boldsymbol{r}_j-\boldsymbol{r}_i|^2}w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)\\ &= \frac{2d}{\lambda n^0}\frac{(\phi_j-\phi_i)}{|\boldsymbol{r}_j-\boldsymbol{r}_i|^2}w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)\end{aligned} \tag{72}$$

これをラプラシアンモデルと比較すると、式 (23) の

$$\frac{1}{\lambda n^0}\sum_{j\neq i}\left[(\phi_j-\phi_i)w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)\right]=\frac{\sum_{j\neq i}(\phi_j-\phi_i)w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)}{\sum_{j\neq i}|\boldsymbol{r}_j-\boldsymbol{r}_i|^2w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)}$$

部分が、式 (73) では

$$\frac{\sum_{j\neq i}\frac{(\phi_j-\phi_i)}{|\boldsymbol{r}_j-\boldsymbol{r}_i|^2}w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)}{\sum_{j\neq i}w(|\boldsymbol{r}_j-\boldsymbol{r}_i|)} \tag{73}$$

になっている。どちらも分子と分母に和の記号が使われているが、$|\boldsymbol{r}_j-\boldsymbol{r}_i|^2$ が式 (23) では分母側に含まれ、式 (73) では分子側に含まれているところが異なる。式 (23) の重み関数に $|\boldsymbol{r}_j-\boldsymbol{r}_i|^2$ を掛けたものを式 (73) の重み関数として採用すると、式 (23) と式 (73) は等しくなる。

したがって、MPS 法では勾配モデルに発散モデルを作用させたものと、ラプラシアンモデルは、同じ重み関数を使用した場合に一致しない [7]。加えて、実際のシミュレーションでは、モデルによって重み関数のパラメータ r_e を変更しているので、この点からも整合性が崩れている。

D　圧力のポアソン方程式の詳細な導出と計算

非圧縮性流体は密度が変化しないので

$$\frac{D\rho}{Dt}=0 \tag{74}$$

となる。これより連続の式は

$$\nabla\cdot\boldsymbol{u}=0 \tag{75}$$

となる。この式と、ナビエ-ストークス方程式 (30) を用いて圧力を求める式を導出する。ナビエ-ストークス方程式に粒子間相互作用モデル使うと

$$\frac{D\boldsymbol{u}_i}{Dt}=\frac{1}{\rho^0}\langle\nabla P\rangle_i^{k+1}+\nu\left\langle\nabla^2\boldsymbol{u}\right\rangle_i+\boldsymbol{g} \tag{76}$$

と表せる。この式を、圧力項以外を陽的、圧力項を陰的に 2 段階に分けて解くと

$$\frac{\boldsymbol{u}_i^* - \boldsymbol{u}_i^k}{\Delta t} = \nu \left\langle \nabla^2 \boldsymbol{u} \right\rangle_i^k + \boldsymbol{g}^k \tag{77}$$

$$\frac{\boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^*}{\Delta t} = -\frac{1}{\rho^0} \left\langle \nabla P \right\rangle_i^{k+1} \tag{78}$$

となる。式 (78) の両辺に ∇ をかけて発散を取ると

$$\frac{\left\langle \nabla \cdot \boldsymbol{u} \right\rangle_i^{k+1} - \left\langle \nabla \cdot \boldsymbol{u} \right\rangle_i^*}{\Delta t} = -\frac{1}{\rho^0} \left\langle \nabla \cdot \nabla P \right\rangle_i^{k+1} \tag{79}$$

となる。非圧縮の条件を満たすためには、時刻 t^{k+1} において式 (75) より

$$\nabla \cdot \boldsymbol{u}^{k+1} = 0 \tag{80}$$

を満たさなければならない。つまり、式 (79) は

$$\frac{0 - \left\langle \nabla \cdot \boldsymbol{u} \right\rangle_i^*}{\Delta t} = -\frac{1}{\rho^0} \left\langle \nabla^2 P \right\rangle_i^{k+1} \tag{81}$$

となり、未知数の $\boldsymbol{u}^{k+1}$ が消える。よって、この式は

$$\left\langle \nabla^2 P \right\rangle_i^{k+1} = \rho^0 \frac{\left\langle \nabla \cdot \boldsymbol{u} \right\rangle_i^*}{\Delta t} \tag{82}$$

と整理され、ポアソン方程式を得る。MPS 法では、右辺を速度の発散の形で表すと圧縮を検出することができず、時間ステップを進めると共に密度の誤差が蓄積し体積が保存されない場合がある。そこで、右辺を粒子数密度で表すことを考える。仮の速度 $\boldsymbol{u}_i^*$ についての連続の式

$$\frac{D\rho_i}{Dt} + \rho^0 \nabla \cdot \boldsymbol{u}_i^* = 0 \tag{83}$$

を、流体密度のラグランジュ微分 $D\rho/Dt$ の前進差分で近似する。ここで、非圧縮性から $\rho_i^k = \rho^0$ となるので

$$\frac{\rho_i^* - \rho^0}{\Delta t} + \rho^0 \left\langle \nabla \cdot \boldsymbol{u} \right\rangle_i^* = 0 \tag{84}$$

となる。ρ_i^* は圧力項以外の計算が終了した時点での仮の速度 $\boldsymbol{u}_i^*$ で粒子を移動させた後の流体密度である。この式を、流体密度と粒子数密度の関係式

$$\begin{aligned} \frac{\rho_i^k - \rho^0}{\rho^0} &\simeq \frac{(mn_i^k/V) - (mn^0/V)}{mn^0/V} \\ &= \frac{n_i^k - n^0}{n^0} \end{aligned} \tag{85}$$

で近似すると

$$\begin{aligned}\langle \nabla \cdot \boldsymbol{u} \rangle_i^* &= -\frac{1}{\Delta t}\frac{\rho_i^* - \rho^0}{\rho^0} \\ &\simeq -\frac{1}{\Delta t}\frac{n_i^* - n^0}{n^0}\end{aligned} \tag{86}$$

となり、速度の発散が粒子数密度の時間変化率の近似で表せる。この式を、式 (82) に代入すると

$$\begin{aligned}\left\langle \nabla^2 P \right\rangle_i^{k+1} &= \rho^0 \frac{1}{\Delta t}\left(-\frac{1}{\Delta t}\frac{n_i^* - n^0}{n^0}\right) \\ &= -\frac{\rho^0}{(\Delta t)^2}\left(\frac{n_i^* - n^0}{n^0}\right)\end{aligned} \tag{87}$$

と MPS 法による圧力のポアソン方程式を得る。粒子数密度で表す形だと、重み関数が圧縮を粒子の接近として検出することができる。よって、時間ステップを進めても密度の誤差が蓄積することを抑制することができ、体積の保存が良くなる効果がある。この圧力のポアソン方程式 (87) の計算方法について考える。両辺を $-\rho^0$ で割ると

$$-\frac{1}{\rho^0}\left\langle \nabla^2 P \right\rangle_i^{k+1} = \frac{1}{(\Delta t)^2}\left(\frac{n_i^* - n^0}{n^0}\right) \tag{88}$$

となる。次に、ラプラシアンモデルの式 (25) を適用する。

$$-\frac{1}{\rho^0}\frac{2d}{\lambda^0 n^0}\sum_{j\neq i}\left[(P_j^{k+1} - P_i^{k+1})w(|\boldsymbol{r}_j^* - \boldsymbol{r}_i^*|)\right] = \frac{1}{(\Delta t)^2}\left(\frac{n_i^* - n^0}{n^0}\right) \tag{89}$$

この式は、時刻 t^{k+1} における圧力 P^{k+1} 以外の値は、重み関数と定数を用いた計算で求めることができる。

P_j^{k+1} の係数は

$$a_{ij} = \begin{cases} -\dfrac{1}{\rho^0}\dfrac{2d}{\lambda^0 n^0}w(|\boldsymbol{r}_j^* - \boldsymbol{r}_i^*|) & (j \neq i) \\ \dfrac{1}{\rho^0}\dfrac{2d}{\lambda^0 n^0}\displaystyle\sum_{j'\neq i} w(|\boldsymbol{r}_{j'}^* - \boldsymbol{r}_i^*|) & (j = i) \end{cases} \tag{90}$$

となり、右辺は

$$b_i = \frac{1}{(\Delta t)^2}\frac{n^* - n^0}{n^0} \tag{91}$$

という、式で表すことができる。また、これらの式から式 (89) は

$$a_{i1}P_1^{k+1} + a_{i2}P_2^{k+1} + \cdots + a_{ii}P_i^{k+1} + \cdots + a_{iN-1}P_{N-1}^{k+1} + a_{iN}P_N^{k+1} = b_i \tag{92}$$

という式で表せる。N は粒子数である。この式は、全ての粒子において成立するので、時刻 t^{k+1} の圧力 P_i^{k+1} を求める連立 1 次方程式を得ることができる。この式を行列を用いて書くと

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1i} & \cdots & a_{1N-1} & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2i} & \cdots & a_{2N-1} & a_{2N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{iN-1} & a_{iN} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ a_{N-11} & a_{N-12} & \cdots & a_{N-1j} & \cdots & a_{N-1N-1} & a_{N-1N} \\ a_{N1} & a_{N2} & \cdots & a_{Nj} & \cdots & a_{NN-1} & a_{NN} \end{pmatrix} \begin{pmatrix} P_1^{k+1} \\ P_2^{k+1} \\ \vdots \\ P_i^{k+1} \\ \vdots \\ P_{N-1}^{k+1} \\ P_N^{k+1} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_{N-1} \\ b_N \end{pmatrix} \tag{93}$$

となる。左辺の係数行列を $\boldsymbol{A}$、圧力のベクトルを $\boldsymbol{x}$、右辺のベクトルを $\boldsymbol{b}$ と置くと

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{94}$$

と表せる。この行列で表した連立 1 次方程式は、ガウスの消去法、ヤコビ法、SOR などで解くことができる。しかし、対称行列でゼロ要素が多い疎行列のため、共役勾配法で解くのが適切だと思われる。いずれかの解法で $\boldsymbol{x}$ を求めることで、P_i^{k+1} が求められる。

謝辞

本研究を行うにあたり、終始適切な助言を賜り、丁寧に指導をしていただいた龍野智哉先生に深謝いたします。また、多くの情報やアドバイスをいただいた龍野研究室の先輩・同期・後輩の皆様に感謝いたします。

参考文献

[1] Daly, B. J., Harlow, F. H., Welch, J. E., Wilson, E. N. and Sanmann, E. E., Numrerical Fluid Dynamics Using the Particle-and-Force Method, LA-3144, 1965

[2] Amsden, A. A., The Particle in Cell Method for the Calculation of the Dynamics of Compressible Fluids, LA-3466, 1966

[3] Harlow, F. H. and Welch, J. E., Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface, Phys. Fluids., 8, 2182-2189, 1965

[4] Lucy, L. B., A Numerical Approach to the Testing of the Fission Hypothesis, Astron. J, 82, 1013-1024, 1977

[5] Koshizuka, S. and Oka, Y., Moving-Particle Semi-Implicit Method for Fragmentation of Incompressible Fluid, Nucl. Sci. Eng., 123, 421-434, 1996

[6] Green, Simon., Particle Simulation using CUDA, http://docs.nvidia.com/cuda/samples/5_Simulations/particles/doc/particles.pdf (2015/10/22)

[7] 越塚誠一, 日本計算工学会編, 計算力学レクチャーシリーズ 5 粒子法, 丸善株式会社, 2005

[8] 越塚誠一, 柴田和也, 室谷浩平, 粒子法入門 流体シミュレーションの基礎から並列計算と可視化まで C/C++, 丸善株式会社, 2014

[9] High Performance Stochastic Simulation on the Graphics Processing Unit, http://www.cs.ucsb.edu/~cse/research_gpu.html(2013/9/27)

[10] NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210, http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf (2016/1/24)

[11] 【後藤弘茂の Weekly 海外ニュース】 NVIDIA が次世代 GPU アーキテクチャ「Kepler」のベールを剥いだ, http://pc.watch.impress.co.jp/docs/column/kaigai/20120322_520640.html (2016/1/24)

[12] 高速演算記 第 24 回「Kepler 解説その 1 〜サンプルプログラムで見るパフォーマンス〜」 G-DEP, http://www.gdep.jp/column/view/31 (2016/1/24)

[13] Tesla ワークステーションの特徴、tesla c2075, http://www.nvidia.co.jp/object/workstation-solutions-tesla-jp.html (2014/1/23)

[14] TESLA K20X GPU ACCEL ERATOR Board Specification, http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf (2016/1/24)

[15] GeForce GTX TITAN X Specifications GeForce, http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications (2016/1/24)

[16] Maxwell: The Most Advanced CUDA GPU Ever Made Parallel Forall, http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/ (2016/1/24)

[17] ARK — IntelR Core i7-3770 Processor (8M Cache, up to 3.90 GHz), http://ark.intel.com/ja/products/65719 (2013/9/23)

[18] ゼロから始める GPU Computing, http://www.gdep.jp/page/index/market (2013/9/22)

[19] SSI: Scalable Software Infrastructure for Scientific Computing, Lis: Library of Iterative Solvers for Linear Systems, http://www.ssisc.org/lis/ (2016/1/25)

[20] The Innovative Computing Laboratory, University of Tennessee, MAGMA, http://icl.cs.utk.edu/magma/ (2016/1/25)