情報処理学会論文誌　コンピューティングシステム　Vol.6　No.3　82–93 (Sep. 2013)

GPUスパコンにおける1億個のスカラー粒子計算の強スケーリングと動的負荷分散

都築 怜理[1,a)]　青木 尊之[1,b)]　下川辺 隆史[1,c)]

受付日 2012年12月21日, 採録日 2013年5月7日

概要：近接相互作用に基づく粒子法の大規模計算では，時間的に粒子の空間分布が大きく偏ることによるノード間の計算負荷の不均一や，ノード間の移動にともなうメモリの断片化が並列化実行性能を大きく低下させる．本論文では，メモリが階層的に分散するGPUスパコンにおいて，与えられた速度場に基づいて移動するパッシブ・スカラー粒子の計算を例題とし，分割した領域間の粒子数の不均一を定期的に解消する方法と，粒子の再整列によりメモリの断片化を解消する方法を提案し，両者を合わせて実行性能の大幅な向上を図る．隣接領域の境界を横切って移動する粒子をGPU上で探索する際，GPU計算に特化したアルゴリズムの提案と実装を行う．粒子の再整列は大きなオーバヘッドとなるため，再整列の最適回数を求めるためのモデルを提案する．GPU間の動的負荷分散と粒子の再整列を導入した実装を構築し，ベンチマーク問題に対する検証を64 GPUを用いて行い負荷分散の効果を確認した．また，GPUスパコンTSUBAME 2.0において1.6億個のパッシブ・スカラー粒子計算に対して良好な強スケーリングを得ることができ，負荷分散を用いない場合と比較して6倍以上の高速化を実現した．

キーワード：GPUコンピューティング，複数GPUスケーリング，動的負荷分散，粒子計算

Dynamic Load Balance and Strong Scaling of a Passive Scalar Computation for 160 Million Particles on a GPU Supercomputer

Satori Tsuzuki[1,a)]　Takayuki Aoki[1,b)]　Takashi Shimokawabe[1,c)]

Received: December 21, 2012, Accepted: May 7, 2013

Abstract: In large-scale particle simulations based on short-range interactions, non-uniform particle distribution makes parallel efficiency poor. In particular, it becomes a serious problem in GPU supercomputers. In this paper, we study passive-scalar particles moving with given velocities and our research focuses on the method of dynamic load balance for GPUs on multiple domains. By applying dynamic domain decomposition, we keep the number of particles constant in each domain. There are several overheads of the data communication among GPUs and inefficient memory usage due to particles across the domain boundary. In order to improve the data communication between CPU and GPU, efficient data packing of those particles to buffer memory and defragmentation of the device memory have been introduced. It is confirmed that the dynamic load balance based on the 2-dimensional slice-grid method works well for a particle computation on 64 GPUs of TSUBAME 2.0. A passive-scalar-particle computation with 160 million particles shows good scalabilities and more than 6 times speedup to compare with that without dynamic load balance.

Keywords: GPU computing, multi-GPU scaling, dynamic load balance, particle simulation

[1] 東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[a)] tsuzuki@sim.gsic.titech.ac.jp
[b)] taoki@gsic.titech.ac.jp
[c)] shimokawabe@sim.gsic.titech.ac.jp

1. 緒言

1.1 背景

パソコンのグラフィクスボードなどに搭載されるGPU（Graphics Processing Unit）は高速な画像表示を主目的と

し，ピーク性能が 1 TFlops を超える高い浮動小数点演算能力を持っている．近年では GPU をアクセラレータとして汎用計算に利用する GPU コンピューティングがさかんに研究されている．特に，NVIDIA 社による GPU コンピューティング統合開発環境の CUDA [1] のリリースにより，GPU の利用は飛躍的に広まった．単一 GPU による計算では，様々なアプリケーションに対して高速化が成功しているが [2], [3], [4], [5], [6]，グラフィクスボード上のメモリ量が限られているため，適用できる対象が小さな問題に限られている．また，複数 GPU を用いることによる高速化の要求も強い．

1.2　複数 GPU による粒子法計算

近接相互作用に基づく粒子法の計算には，SPH（Smoothed Particle Hydrodynamics）法 [7], [8] や，MPS（Moving Particle Semi-Implicit）法 [9], [10] や個別要素法である DEM（Discrete Element Method）[11] などがある．**図 1** に示す DEM 計算は粒子が接するときにのみ反発力や摩擦力が働く物理モデルであり，ある粒子の時間積分の際に接触する可能性がある粒子がたかだか数個であるような局所性の高い相互作用計算である．分子動力学計算や重力多体計算と異なりメモリ参照が計算速度を律速する．

GPU を用いて計算する場合，各粒子の位置，速度などの従属変数はグラフィクスボード上のメモリに置かれる [12], [13], [14]．複数 GPU を用いる場合，移動が激しい粒子運動に対する初期粒子番号によるデータ分割は，全 GPU への通信を考慮する必要があり．通信量が多く演算に対する通信のオーバヘッドが著しく増大し計算効率が大きく低下する．計算領域を小領域に空間分割し，分割された各小領域に GPU を割り当てる必要がある．

複数 GPU 計算では，粒子が運動することにより小領域間の境界を横切る粒子が発生すると，隣接 GPU 間でのデータ通信が必要となる．粒子の移動によるデータ通信の繰返しにより，メモリに不参照領域が断片的に発生し，メモリ使用効率が悪化する．さらに，初期に負荷が均一な領域分割を行った場合も，時間とともに粒子分布の偏りにより負荷バランスが悪化する．これらが原因となり，近接相互作用に基づく粒子法の複数 GPU を用いた計算では，並列化にともない実行性能が頭打ちとなる．

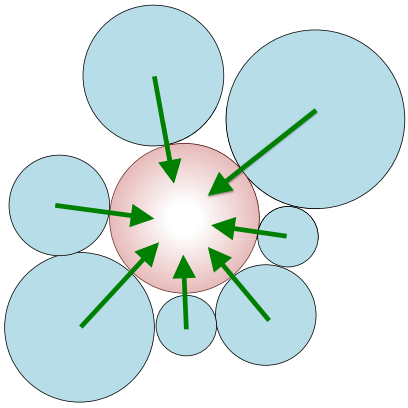

図 1　周囲の粒子の接触により相互作用する粒子
Fig. 1　Conceptual illustration of near-distance interaction among particles.

1.3　研究目的

パッシブ・スカラー粒子計算は，与えられた速度場のもとで粒子を移動させる計算で，粒子法の計算の中で粒子移動の計算に相当する．複数 GPU によるパッシブ・スカラー計算で動的負荷分散が可能になれば，これらを GPU による近接相互作用に基づく SPH，MPS，DEM へと発展させることができる．

本論文では，東京工業大学学術国際情報センターの TSUBAME 2.0 の 500 台を超える GPU を用いた数千〜1.6 億個のパッシブ・スカラー粒子計算を高い実行効率で行う．GPU 間の通信の繰返しにより発生するメモリの断片化を粒子の再整列により解消し，不均一な粒子分布に対して各 GPU の計算負荷を均一にするためのアルゴリズムを提案する．これらのアルゴリズムを実装し性能評価を行う．

1.4　既往研究との比較

複数 GPU による近接相互作用の粒子法計算で発生するメモリ管理の問題や GPU 間の動的負荷分散に取り組む研究は前例が見当たらない．たとえば，参考文献 [15] は，大規模 DEM 計算に複数 GPU を利用した先行研究であるが動的負荷分散の問題には取り組んでいない．参考文献 [16], [17] などはクラスタリング手法や CG（Computer Graphics）のレンダリングにおける複数 GPU 間の動的負荷分散に取り組んでいるが，いずれも数 GPU 程度の小規模を想定して設計されており，またアルゴリズムとしても本研究が対象とする粒子計算に応用できるものではない．複数 GPU を用いた大規模な粒子計算における GPU 間の動的負荷分散は本研究が先駆的役割を担う．

近接相互作用に基づく粒子法計算における CPU 間の動的負荷分散についてはいくつかの研究例があるが [18], [19], [20]，逐次処理を前提としたこれらの手法をそのまま GPU で利用した場合，GPU では十分な実行性能が得られないことが予想される．たとえば，CPU と同様に逐次処理で粒子探索を行うことは GPU では大きなオーバヘッドになる．GPU に特化した効率的なアルゴリズムを提案する必要がある．

粒子が境界を横切り隣接領域へ移動することで，メモリの断片化が起こる．これを解消するため，定期的にメモリ上で粒子の再整列を行う必要がある．複数 CPU による計算ではそれほどのオーバヘッドとならず，再整列の頻度を制御する必要がない場合が多い．一方，GPU 上で粒子デー

タを再整列する場合，GPU は高い効率が期待できない．GPU からホスト側のメモリへデータ転送して CPU で再整列する場合，再整列自体は高速に実行できるが，CPU–GPU 間の通信をともなうこととなり，その通信時間が大きなオーバヘッドとなる．本研究では，再整列の適切な頻度を予測する方法について検討する．

2. 複数 GPU を用いたパッシブ・スカラー粒子の時間積分

本研究ではパッシブ・スカラー粒子の時間積分を次の手順で行う．(1) 現在の領域分割のもと，粒子を速度場を用いて時間発展させる．(2) 領域外に移動した粒子について GPU 間で通信を行う．(3) 定期的に粒子の再整列を行い，1 ステップ分の時間積分を行う．(4) 粒子分布の悪化が設定したしきい値以上である場合，次の時間ステップに進む前に GPU 間の負荷分散を行う．

本論文では，2 章で基礎となる GPU 実装について，2.1 節で (1)，(2) を扱い，2.2 節で (3) を説明する．3 章で実装の性能評価を行う．(4) は 5 章以降で扱う．

2.1 時間発展と GPU 間通信

すでに計算領域が小領域に分割されていて，領域分割が時間的に変更されない場合の各小領域で行う計算手順を**図 2** に示す．ホスト（CPU）側で，1. **初期条件の設定**を行う．粒子の情報として座標と粒子の所属する小領域の番号を保持するために，各小領域でメモリを確保する．メモリの不参照領域には，参照領域との区別のため，小領域の最大番号より大きな番号をフラグとして格納する．これらをデバイス（GPU）側のメモリにコピーする．2. **時間発展**では，以下に示す 4 段ルンゲクッタ法を用いて時間発展させる．本研究では検証のため，再現が容易な解析解の速度場を用いる．

$$k_1 = hf\left(t_n, y_n\right) \tag{1}$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \tag{2}$$

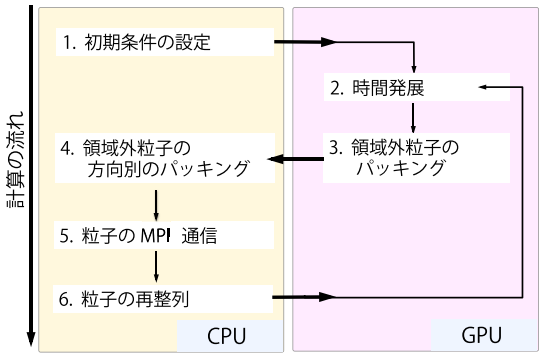

図 2　時間積分の手順

Fig. 2　Flow diagram of the time integration for passive scalar particles on static-decomposed domains.

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \tag{3}$$

$$k_4 = hf(t_n + h, y_n + k_3) \tag{4}$$

$$t_{n+1} = t_n + h \tag{5}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{6}$$

式 (1)～式 (6) は従属変数 y についての 4 段ルンゲクッタ法の各段を表す．h が時間刻み，t_n が n ステップ目における時刻である．時間発展により，小領域境界を横切り外に移動する粒子（領域外粒子）が発生すると，移動先の GPU に転送する必要がある．GPU 間のデータ転送は CPU を介して行うため，粒子の情報をホスト（CPU）のメモリにコピーする必要がある．全粒子の情報を GPU のメモリから CPU のメモリにコピーし CPU で領域外粒子の選別を行うと，GPU と CPU 間の通信が大きなオーバヘッドとなる．3. **領域外粒子のパッキング**では，GPU–CPU 間の通信回数，通信量を可能な限り抑えるため，GPU 上で粒子の移動する方向に関係なくすべての領域外粒子を 1 つのバッファに集める．このバッファを CPU へ 1 度に転送する．ホスト側で受け取ったデータは，粒子の横切った方向に関係なく領域外粒子すべてを含むので，4. **粒子の方向別のパッキング**により隣接する小領域の各方向に送信する粒子の選別をホスト側で行う．隣接する各小領域に 5. **粒子の MPI 通信**を行う．受信したデータをデバイス側にコピーし，メモリの使用領域の最後尾部に加えて通信が完了する．

2.2 粒子の再整列

小領域から外に移動した粒子が使用していたメモリは，不参照メモリとなる．計算ステップが進行するにつれて，不参照メモリは増加していく．**図 3** は 1 番の小領域の断片化を模式的に表したものである．図中の数字は小領域の番号を表す．緑色が自分の小領域内の粒子が使用しているメモリを，紫色が不参照メモリを表す．橙色が次のステップで他の小領域に移動する粒子を表し，粒子の送受信後に不参照メモリとなる．また，新たに流入した粒子の情報は，メモリの使用領域の最後尾に追加される．

GPU を用いる場合，効率的に計算するためスレッドは連続したメモリを参照することになる．このとき，粒子データが有効か無効かは関係なくメモリ参照は行われるため，不参照メモリの部分には不要なデータの読み込みとそれに

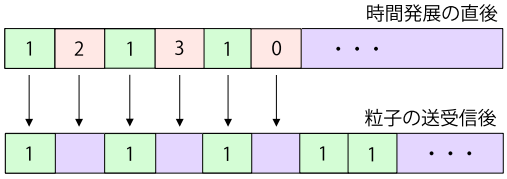

図 3　GPU メモリの断片化

Fig. 3　Fragmentation of the GPU memory as increased in the time step.

ともなう条件分岐が入る．このため，使用メモリの断片化は GPU 計算の性能を低下させる．

メモリの断片化を防ぐため，6. **粒子の再整列**をホスト側で行う．**図 4** はそれを模式的に表したものである．粒子の属する小領域の番号を表すバッファにおいて，不参照メモリには小領域の最大番号より大きな数字が書き込まれていて，再整列を行うことにより不参照メモリはバッファの最後尾に集められる．

再整列は逐次処理を多く含むため，GPU よりも CPU の方が適している場合が多い．**図 5** は粒子を粒子構造体のメンバの 1 つである所属領域の番号でソートする場合の計算時間を，CPU と GPU とで比較したものである．(A) は STL（Standard Template Library）による CPU でのソートの時間を，(B) は (A) に CPU–GPU 間通信を合わせた時間を，(C) は THRUST ライブラリ [21] による GPU でのソートの時間を表す．CUDA は Version 4.0 を，CPU によるソートに対しては Intel コンパイラ（Version 11.1）を用いている．GPU のソートは，CPU 上でのソートと比較して 15 倍近く計算時間がかかっており，CPU でのソート自体にかかる時間と CPU–GPU 間通信の時間を合わせても，GPU でのソートにかかる時間より短いことが分かる．本論文では粒子の再整列を CPU で行う．GPU–CPU 間で全粒子の情報をコピーする必要があり，ソートの回数を制御する必要がある．

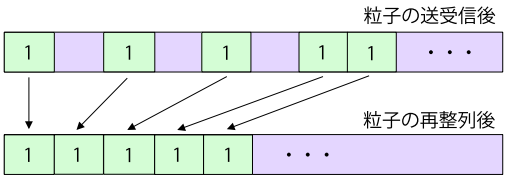

図 4　粒子番号の再整列
Fig. 4　Re-numbering by sorting.

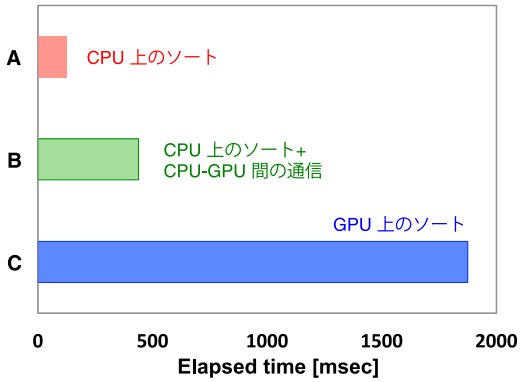

図 5　4,194,304 個の粒子を粒子構造体のメンバ（所属領域の番号）でソートする場合の CPU と GPU の計算時間の比較
Fig. 5　Comparison of computational time of sorting on CPU with that on GPU.

3.　均一分布・一様速度場を用いた検証

計算領域全体にわたり粒子が均一に分布し，一様な速度場で移動する場合は，時間が経過しても粒子分布が変化せず，各小領域の計算負荷は均一に保たれる．そのような計算では．小領域の境界を横切り隣接小領域へ移動する粒子についての GPU 間データ転送とメモリの断片化が計算の顕著なオーバヘッドとなる．

本章ではそのような動的負荷分散を必要としない計算条件のパッシブ・スカラー粒子計算に対して，弱スケーリングと強スケーリングによる性能評価を行う．強スケーリングでは，粒子の再整列を行う場合と行わない場合のそれぞれについて計算時間を測定する．さらに再整列の頻度を変えて計算時間を測定し両者の関係を調べる．

3.1　弱スケーリング

1 プロセスあたりの粒子数を固定し，プロセス数に比例して粒子数を増加させる．全粒子数はプロセス数に比例して増加する．計算規模が拡大しても並列化効率が保たれることを確認する．

(1) **実行環境**　東京工業大学学術国際情報センターの TSUBAME 2.0 を用いる．1 ノードあたり，GPU は NVIDIA 社の Tesla M2050 が 3 基, CPU は Intel 社の Xeon X5670 が 2 個搭載されている．1 ノードにつき 2 core, 2 GPU を利用する．CUDA の Version は 3.2 である．

(2) **計算条件**　2 次元の計算領域に均一に分布させた粒子を一定速度場（$u = v = 1.0$）のもと 4 次ルンゲクッタ法により時間発展させる．周期境界条件を用いて計算負荷を絶えず均等にする．1 GPU の担当する小領域は，x 方向の長さを $Lx = 1.0$，y 方向の長さを $Ly = 1.0$ とする．1 GPU あたりの粒子数は 4,194,304（$= 2^{22}$）個とする．初期条件では全計算領域に均一に粒子を分布させる．計算領域の分割方法は，x 方向，y 方向の分割数の等しい 2 次元領域分割とする．時間ステップは，$\Delta t = 0.001$ とする．500 ステップの計算時間を測定する．再整列は 500 ステップのうち，250 ステップ目に 1 回行う．初期条件の設定は測定に含めない．

(3) **計算時間**　4 GPU から 64 GPU まで変化させたときの 1 ステップあたりの計算時間を**図 6** に示す．赤が 4 段ルンゲクッタ法による時間発展を，緑が GPU での領域外粒子のパッキングを，橙色が領域外粒子の方向別のパッキングを，青が粒子の再整列の計算時間を表す．領域外粒子のパッキングは全体の計算時間の 1 割程度であり，複数 GPU 化により必要となる処理のオーバヘッドは小さい．

64 GPU の場合で粒子数は，2 億 5,000 万個以上になる．並列計算の最小規模である 4 GPU の計算時間に対する，複数 GPU を用いた場合の計算時間の割合を効率と定義すると，64 GPU を用いた場合に並列化効率 97%以上となり，

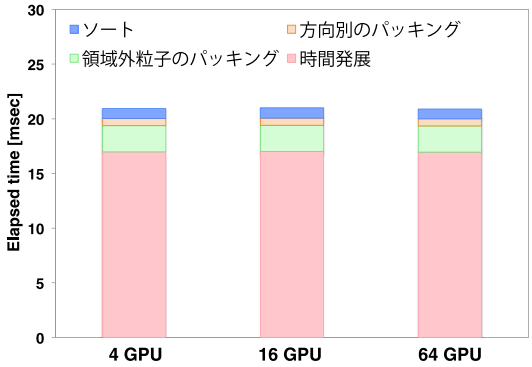

図 6　均一分布・一定速度場を用いた弱スケーリング

Fig. 6　The result of weak scaling under constant velocity field with uniform particle distribution.

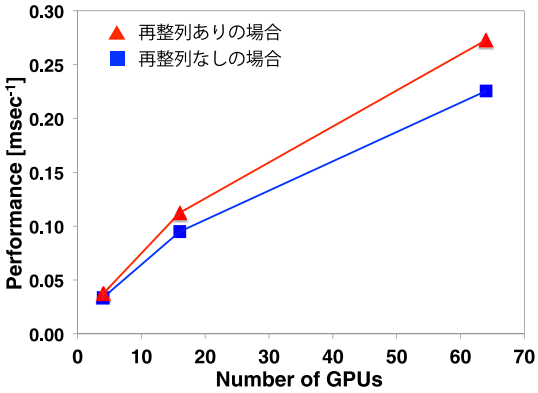

図 7　均一分布・一定速度場を用いた強スケーリング

Fig. 7　The result of strong scaling under constant velocity field with uniform particle distribution.

良好な弱スケーリングが達成されている．

3.2　強スケーリング

全体の粒子数を一定にし，GPU 数を変化させてそれぞれの場合の計算時間を測定する．1 台あたり GPU が計算する粒子数は GPU 数が増えるにつれて減少する．

(1) **計算条件**　2 次元の計算領域に均一に分布させた粒子を一定速度場（$u = v = 1.0$）のもと 4 次ルンゲクッタ法により時間発展させる．周期境界条件を用いて計算負荷を絶えず均等にする．1 ステップ間に，1 GPU あたりの粒子の移動量は 16,000 個程度となるように Δt を設定している．粒子数は総粒子数を 16,777,216（$= 2^{44}$）個で固定する．1,500 ステップにかかる計算時間を測定する．計算領域の分割方法は，x 方向，y 方向の分割数の等しい 2 次元領域分割とする．測定時間を全ステップ数で割り 1 ステップあたりの計算時間を算出し，逆数の値を実行性能とする．測定対象には初期条件の設定は含めない．

(2) **性能**　測定結果を**図 7** に示す．赤線が 100 回に 1 度再整列を行う場合の性能を，青線が再整列を行わない場合の性能を表す．再整列を行う場合に性能が向上することが確認できる．1 GPU に対して，4 GPU で 2.0 倍，16 GPU で 6.2 倍，64 GPU で 15.6 倍の性能向上を達成している．

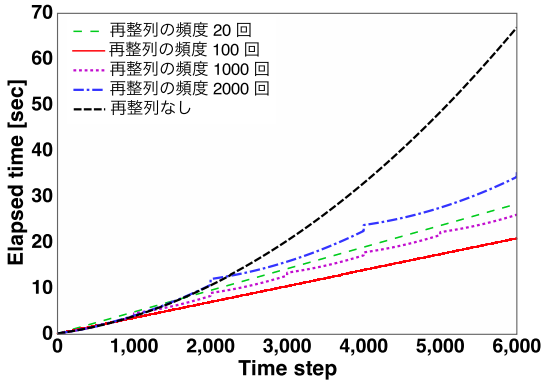

図 8　64 GPU を用いた場合の再整列の頻度の違いによる積算計算時間の変化の比較

Fig. 8　Elapsed time per a step on 64 GPU by sorting every 100, 1,000, and 2,000 steps in the case using 64 GPUs.

3.3　再整列の頻度と計算時間の関係

図 7 の 64 GPU 使用時において，再整列の頻度を 5 通りに変えた場合の積算計算時間の測定結果を**図 8** に示す．それぞれ再整列を行わない場合（黒線），20 回に 1 回行う場合（緑線），100 回に 1 回行う場合（赤線），1,000 回に 1 回行う場合（紫線），2,000 回に 1 回行う場合（青線）を表す．再整列の頻度を上げるにつれて計算時間は線形変化に近づき，頻度が 100 回に 1 回の場合に，行わない場合の 1/3 以下となる．頻度が 20 回の場合，頻度が 100 回に 1 回の場合よりも計算時間がかかる．この中では頻度が 100 回に 1 回の場合が，最適解の近傍と予想できる．

4.　再整列の頻度のモデル化

3 章では，粒子の再整列を定期的に行うことにより積算計算時間が線形変化となることを示した．本章では，再整列の頻度と計算時間の関係のモデル化を行い．計算時間を最小化するための再整列の最適頻度について考察する．はじめに，再整列を行わない場合と行う場合のそれぞれについて計算時間のモデル化を行う．

4.1　再整列を行わない場合

i ステップ目の計算時間を T_i とする．各スレッドで同様の計算を行う場合，GPU の計算時間は，スレッド数に比例する．0 ステップ目では，スレッド数は粒子数 N_0 に等しいので，比例定数を α として T_0 は次のように書ける．

$$T_0 = \alpha N_0 \tag{7}$$

1 ステップ目の粒子数 N_1 は，流入した粒子数を ΔN_{in}，流出した粒子数を ΔN_{out} とすると，$N_1 = N_0 + \Delta N_{in} - \Delta N_{out}$ である．ΔN_{out} はメモリに離散的に存在する不参照粒子数であるため，スレッドは，$N_0 + \Delta N_{in}$ だけ起動する．よって第一次近似的には計算時間は $N_0 + \Delta N_{in}$ に比例する．

しかし，不参照メモリは粒子データの読み込みのメモリアクセスだけで，書き込みのメモリアクセスは行わないため，GPU にかかる全体の負荷は，すべてのスレッドが粒子の実データを扱う場合よりは軽くなる．そこで，計算時間は，不参照メモリ数 ΔN_{out} に比例する量だけ短縮されると仮定する補正項を導入する．T_1 は，β を補正項にかかる係数として次のように表せる．

$$T_1 = \alpha(N_0 + \Delta N_{in}) - \beta \Delta N_{out} \tag{8}$$

一定速度場より $\Delta N_{in} = \Delta N_{out} = \Delta N$ であるから，式 (8) は次のようになる．

$$T_1 = \alpha(N_0 + \gamma \Delta N) \qquad \left(\gamma = \frac{\alpha - \beta}{\alpha}\right) \tag{9}$$

0 ステップ目と比較して，2 ステップ目は 1 ステップ目と 2 ステップ目の合計の流入量（$= 2\Delta N_{in}$）だけ粒子の実データが増加し，1 ステップ目と 2 ステップ目の合計の流出量（$= 2\Delta N_{out}$）だけ不参照メモリが発生している．計算時間 T_2 は，式 (8) で，ΔN_{in}，ΔN_{out} がそれぞれ 2 倍になるため，次のように書ける．

$$T_2 = \alpha(N_0 + (2\Delta N_{in})) - \beta(2\Delta N_{out}) \tag{10}$$

式 (9) と同様に整理すると，式 (10) は次のようになる．

$$T_2 = \alpha(N_0 + 2\gamma \Delta N) \tag{11}$$

3 ステップ目以降についても同様であるので，i 番目のステップ 1 回の計算時間 T_i は次のようになる．

$$T_i = \alpha(N_0 + i\gamma \Delta N) \tag{12}$$

m 番目のステップまでの計算時間の積算値 $T_{sum}(m)$ は，式 (12) の m 番目までの和であり，次のようになる．

$$T_{sum}(m) = \sum_{j=0}^{m} T_j = \sum_{j=0}^{m} (A + j2B) \tag{13}$$

ここで，$\alpha N_0 = A$，$\alpha\gamma\Delta N/2 = B$ とした．式 (13) は等差数列である．m 項までの和を求めると，次のようになる．

$$T_{sum}(m) = A + (A + B)m + Bm^2 \tag{14}$$

4.2　再整列を行う場合

再整列を S 回に 1 回行う場合を考える．再整列の際に発生する GPU–CPU 間通信や再整列 1 回にかかる計算時間の合計時間を，T_{sort} とする．再整列をかけるたびに計算時間は 0 ステップの計算時間に戻るので，m ステップ目までの積算の計算時間は，式 (13) の S ステップ目までの和 m/S 回の合計と，m/S 回の T_{sort} の和であるので，以下のように表せる．

$$T_{sum}(m) = \left(\frac{m}{S}\right) \sum_{j=0}^{S} T_j + \left(\frac{m}{S}\right) T_{sort} \tag{15}$$

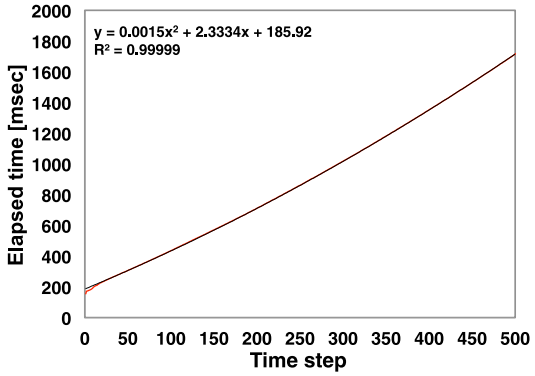

図 9　図 8 の再整列を行わない場合の測定値を，式 (14) によりフィッティングした結果

Fig. 9　Fitting of parameters to the beginning of the result by 500 step.

式 (12) を代入すると次のようになる．

$$T_{sum}(m) = \left(\frac{m}{S}\right) \sum_{j=0}^{S} \alpha(N_0 + j\gamma \Delta N) + \left(\frac{m}{S}\right) T_{sort} \tag{16}$$

$\alpha N_0 = A$，$\alpha\gamma\Delta N/2 = B$ より，式 (16) は次のようになる．

$$T_{sum}(m) = \left(A + (S+1)B + \frac{A + T_{sort}}{S}\right) m \tag{17}$$

4.3　係数 A，B の決定

α，γ は，粒子数やレジスタの占有率などの様々な要因に依存するパラメータであるため，未定係数の A，B を関係式 $A = \alpha N_0$，$B = \alpha\gamma\Delta N/2$ から一意に決定することは難しい．そこで，再整列を行わない場合の計算時間の測定値に対して，式 (14) を用いて最小二乗法によるフィッティングを行い A，B を決定する．

図 9 は，図 8 の再整列を行わない場合の測定値の初期の 500 ステップを，式 (14) によりフィッティングした結果である．赤線が測定値を，黒線がフィッティング曲線を表し，$A = 2.3319$，$B = 0.0015$ と求まる．

4.4　測定値とモデル値の比較

測定値と，式 (14)，式 (17) から決定される $T_{sum}(m)$ を比較する．再整列を行わない場合，100 回に 1 回再整列を行った場合のそれぞれの測定結果と，$T_{sum}(m)$ の比較を **図 10** に示す．黒の実線が図 8 の再整列を行わない場合の，赤の実線が 100 回に 1 回再整列を行う場合の測定値である．黒色の点線，赤色の点線はそれぞれ式 (14)，式 (17) による予測結果である．図 10 に示すとおり，モデルの曲線と測定値の曲線の傾向はよい一致を示している．再整列を行わない場合は式 (14) のようにステップ数 m の 2 乗に比例して計算時間が増加し，再整列を行う場合は式 (17) のように，ステップ数 m に比例することが分かる．

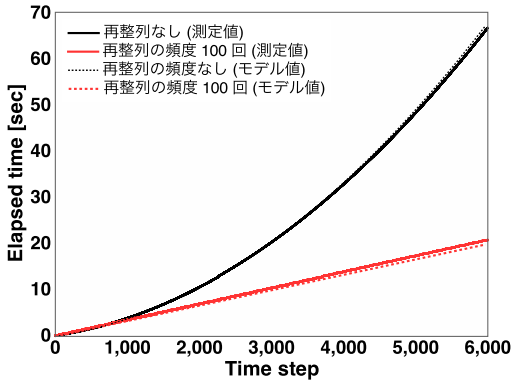

図 10　図 8 の測定値とモデル値の比較

Fig. 10　Comparison of the estimated value with the result as the same Fig. 8.

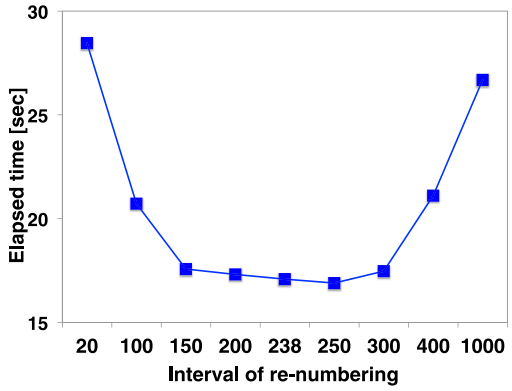

図 11　再整列の頻度の違いによる計算時間の変化

Fig. 11　Comparison of the elapsed time in case of changing the interval of re-numbering.

4.5　最適解の決定

再整列の回数が最適値となる条件は，式 (17) が最小値をとることである．式 (17) が極小値をとる必要条件として，

$$\frac{\partial T_{sum}(m)}{\partial S} = Bm + (A + T_{sort})\left(\frac{-m}{S^2}\right) = 0 \qquad (18)$$

を S について解くと，最適値 S_{opt} は，

$$S_{opt} = \sqrt{\frac{A + T_{sort}}{B}} \qquad (19)$$

となる．式 (19) に，A，B，別途測定した $T_{sort} = 85.4\,\mathrm{msec}$ を代入すると，$S_{opt} = 238$ 回となる．

式 (19) から求めた最適解 238 回の妥当性を検証するため，再整列の頻度を 100 から 50 回刻みで変化させて，6,000 ステップにかかる計算時間をそれぞれ測定した結果を**図 11** に示す．再整列の頻度の最適値は 250 回付近にあることが分かりモデル値の妥当性を示している．

4 章で示したモデルは一定速度場で均一な粒子分布に対するものであり，実際の粒子法の計算にはそのまま適用できないが，GPU 数が多くなると領域内の粒子分布や速度分布は緩やかになることが多く，再整列の頻度に対する 1 つの目安として有効であると考えられる．

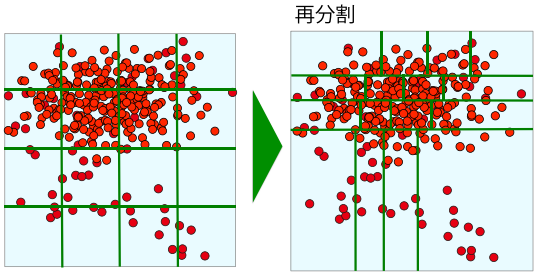

図 12　動的負荷分散の概要

Fig. 12　Outline of dynamic load balance.

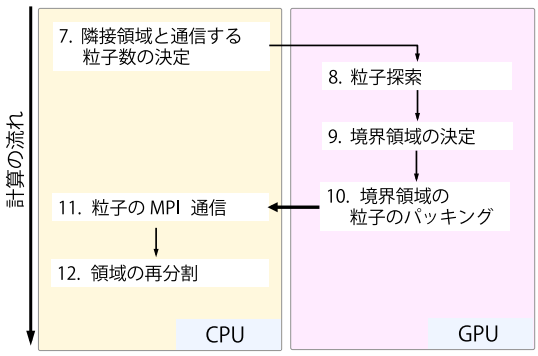

図 13　動的な再領域分割処理の流れ

Fig. 13　Flow diagram of dynamic re-domain decomposition.

5.　スライスグリッド法による動的負荷分散

4 章では均一粒子分布で一様な速度場に対する検証を行った．しかし一般的な粒子分布と非一様な速度場のもとでは，時間発展とともに粒子が移動することによる GPU 間の粒子分布の偏りが生じる．**図 12** のように各小領域境界を移動し大きさを動的に変更することで GPU 1 台あたりの粒子数を一定にすることにより計算負荷を均一に保つ．本研究はすでに提案されているスライスグリッド法による動的な 2 次元領域分割を GPU で実現する．ここでは動的な 1 次元領域分割を垂直方向と水平方向に順次適用して実現する．

5.1　隣接領域への送信粒子数の決定

動的な再領域分割処理の流れを**図 13** に示す．まず，**7. 隣接領域と通信する粒子数の決定**を行う．

全粒子数を N_{total} とし，GPU の数を p とする．1 GPU の均等粒子数は N_{total}/p である．i 番目の小領域内の粒子数を N_i としたとき，式 (20)，(21) から求められる小領域間で転送する粒子数 ΔN_i を導入する．

$$\Delta N_0 = N_0 - N_{total}/p \qquad (20)$$

$$\Delta N_i = \Delta N_{i-1} + N_i - N_{total}/p \qquad (i \geq 1) \qquad (21)$$

隣接領域への送信量，ΔN_{i-1}，ΔN_i の正負と粒子の送信方向の関係について**図 14** に示す．$\Delta N_i > 0$ のとき，i 番目の小領域から $i+1$ 番目の小領域に ΔN_i 個の粒子を送信

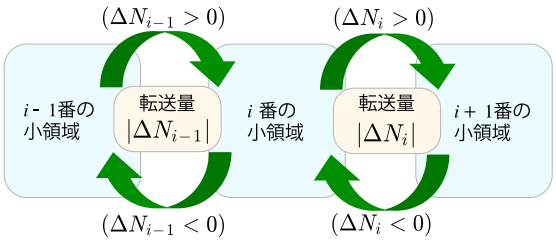

図 14　小領域の番号と粒子の通信量の関係
Fig. 14　Relationship between domain number and the amount of sent particles.

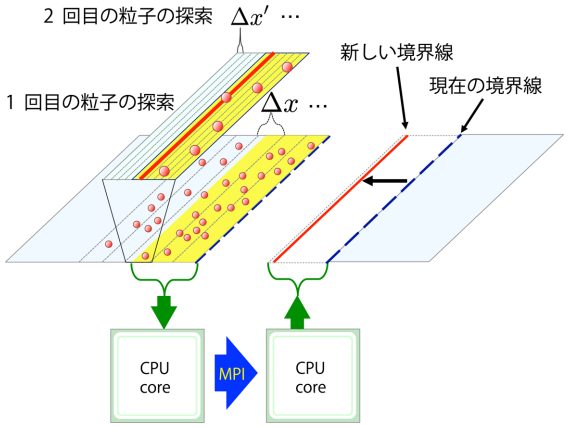

図 15　小領域の分割
Fig. 15　Domain-split to make the list of particle position.

する．$\Delta N_i < 0$ のとき，$i+1$ 番目の小領域から i 番目の小領域に $-\Delta N_i$ 個の粒子を送信する．

効率的な通信を行うため，式 (20)，(21) で小領域間で転送する粒子数を先に決定し，実際の粒子データの通信は全 GPU で並列に行う．ここで，粒子を送信しても小領域内の粒子が 0 未満にならないことが条件であるため，決定した送信粒子数の和は，小領域内にある粒子数よりも小さいことが条件となる．

$$\max(\Delta N_i, 0) + \max(-\Delta N_{i-1}, 0) \leq N_i \qquad (22)$$

5.2　粒子探索

図 13 の 8. **粒子探索**では，隣接小領域に移動する粒子を GPU 上で探索する．粒子は空間にランダムに分布しているため，**図 15** のように自身の小領域を微少幅 Δx で分割し，各分割（以下，セルという）内の粒子を数え上げる．

CUDA を用いた GPU による計算では，1 スレッドが 1 粒子を担当し，各スレッドで担当する粒子の属するセル ID を判定する．粒子に所属セルの ID を割り振り，ID を記した配列型のデータに対して，各 ID の数をそれぞれ数え上げる．GPU 上で排他処理を可能にする `atomic` 関数を利用する方法が考えられるが，GPU の実行性能を著しく低下させる．本論文では `atomic` 関数を使用せず，各 ID について `reduction` 操作のカーネル関数の 1 回の実行で数え上げる方法を提案する．GPU 上での `reduction` の効率的な方法は，Harris らによって示されており [22]，本研究ではこれを各 ID について同時に行うよう変更する．

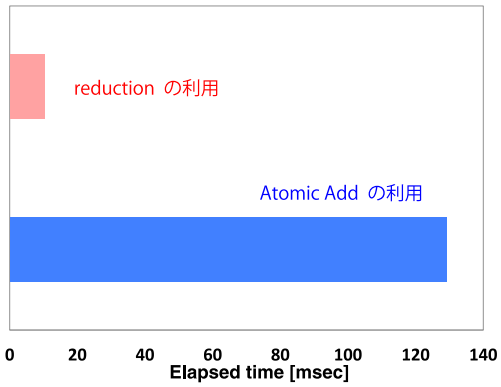

図 16　`reduction` 操作と `atomic` 関数の計算時間の比較
Fig. 16　Comparison between the computational time of `reduction` and `atomic` function.

GPU のシェアードメモリには，セル ID と粒子番号を要素に持つ整数型の 2 次元配列を用意する．2 次元配列には，粒子のセル ID に該当する要素は 1 を，それ以外は 0 を格納する．各 ID について，それぞれブロック内でデータの加算を行う．シェアードメモリの計算結果を，グローバルメモリに書き込み，CPU のメモリに転送する．シェアードメモリからグローバルメモリに書き込む際，1 次元配列で確保された領域に書き込むことで 1 回の転送で済む．CPU 上でブロックごとの計算結果を各セルについて足し合わせ，`reduction` が完了する．セル ID の種類やブロック内のスレッド数によりシェアードメモリが不足する場合は CUDA の API [1] を用いてシェアードメモリの大きさをデフォルトの 16 KB から 48 KB へ変更する．

図 16 に，16,777,216 個の粒子に 10 種類のセル ID を割り振り，各セル ID の個数について GPU 上で数え上げるのに要する計算時間を，`atomic` 関数と比較した結果を示す．本実装により 10 倍以上の高速化が確認できる．

9. **境界領域の決定**で，隣接小領域に送信する粒子を含むセルの数（境界領域）を求める．1 度の Δx 刻みでの境界領域の探査では不十分な可能性がある．境界領域の端のセルを微小幅 $\Delta x'$ で再分割し，この部分の粒子探索も行うような再帰的な探査を行う．10. **境界領域の粒子のパッキング**，11. **粒子の MPI 通信**は，時間発展後の GPU 間通信の 3.〜5. と同様である．12. **領域の再分割**では，受信した粒子と送信した粒子の座標をもとに境界領域を変更する．

5.3　2 次元領域分割への拡張

2 次元領域分割の動的負荷分散では，スライスグリッド法による 1 次元領域分割を垂直方向に行い．次に水平方向に行う．垂直方向の再領域分割では，水平方向に並んだ小領域を 1 つのグループとし，グループの粒子数の総和が一定になるよう，垂直方向の境界線を変更する．各グループの左

端の小領域を小マスタと呼ぶことにし，小マスタがグループの総粒子数を管理する．隣接する垂直方向の小マスタ間で通信を行い，隣接グループに送信する粒子数を決定する．次に，水平方向の再領域分割は，グループ内で各小領域の粒子数が一定になるように，隣接小領域の境界線を変更する．

6. 不均一粒子分布の動的負荷分散法の性能評価

本章では動的負荷分散を行うことが必要な計算条件のパッシブ・スカラー粒子計算に対し，5 章で提案した負荷分散法を用いることで高速化を図る．非一様な速度場に対して不均一な粒子分布を仮定し，強スケーリングによる性能評価を行い，提案する負荷分散法の有用性を確認する．

6.1 渦速度場での検証

(1) **計算条件**　2 次元に配置したスカラー粒子を，4 段ルンゲクッタ法により時間発展させる．以下に示すような非一様な渦速度場を用いる．

$$u(r,t) = -2\cos(\pi t/T)\sin(\pi x)\sin(\pi x)\cos(\pi y)\sin(\pi y) \tag{23}$$

$$v(r,t) = 2\cos(\pi t/T)\cos(\pi x)\sin(\pi x)\sin(\pi y)\sin(\pi y) \tag{24}$$

T は速度場が時間的に変化する周期であり，$T = 8.0$ とする．粒子の時間積分のステップは $\Delta t = 0.01$ とする．速度場の 1 周期は 800 ステップとなり，粒子分布も 800 ステップで初期分布に戻る．計算領域は，x 方向の長さを $Lx = 1.0$，y 方向の長さを $Ly = 1.0$ とする．64 台の GPU を使用し，x 方向，y 方向の分割数の等しい 2 次元領域分割とする．初期条件では，疑似乱数を用いてガウス分布に配置する．粒子数 4,096（$= 2^{12}$）個とする．各時刻での小領域間の粒子数の比（最大粒子数/最小粒子数）と，そのときの各境界線を求める．動的負荷分散は各ステップで実行することにする．

(2) **計算結果**　各時間ステップにおける小領域間の粒子数の比（最大粒子数/最小粒子数）の測定結果を**図 17** に示す．動的負荷分散の実行により，赤線で示すとおり GPU 間の粒子数の比はほぼ一定となっている．計算結果のスナップショットを**図 18** に示す．図 18 は右から順番に，400 ステップ目までの 100 ステップごとのスナップショットである．渦速度場は周期関数であり，後半の 400〜800 ステップについては，図 18 の逆過程である．変化する粒子分布を追従して計算領域を再分割していることが分かる．

6.2 回転速度場での強スケーリング

(1) **計算条件**　渦速度場は，粒子分布の空間的非対称から検証には適しているが，後半に粒子分布が均一化するため，動的負荷分散の性能評価には不向きである．ここでは，以下の回転速度場を用いる．

$$u(t) = -1.0\cos(\pi t/T) \tag{25}$$

$$v(t) = 1.0\sin(\pi t/T) \tag{26}$$

T は周期であり，$T = 1.0$ とする．DEM などの粒子法では，計算の安定性などのため，時間ステップは $\Delta t = 10^{-6}$ 程度の値であり，ここでも $\Delta t = 10^{-6}$ とする．総粒子数が 10^7 個と，1.6×10^8 個の 2 通りの場合を測定する．総粒子数が 10^7 個の場合，GPU を 4 台から 128 台まで変化させる．総粒子数が 1.6×10^8 個の場合，GPU を 64 台から

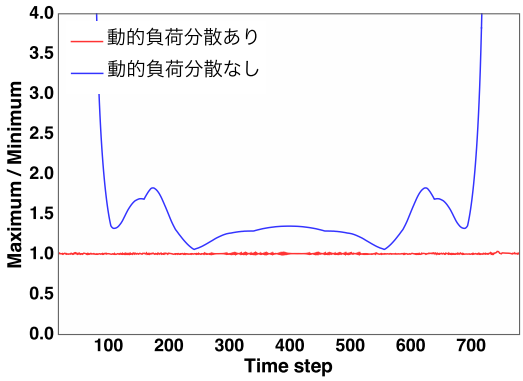

図 17　4 GPU を用いた場合の小領域間の最大粒子数と最小粒子数の比率の変化

Fig. 17　Temporal change of the maximum/minimum ratio of the particle number included in sub-domains in the case using 4 GPUs.

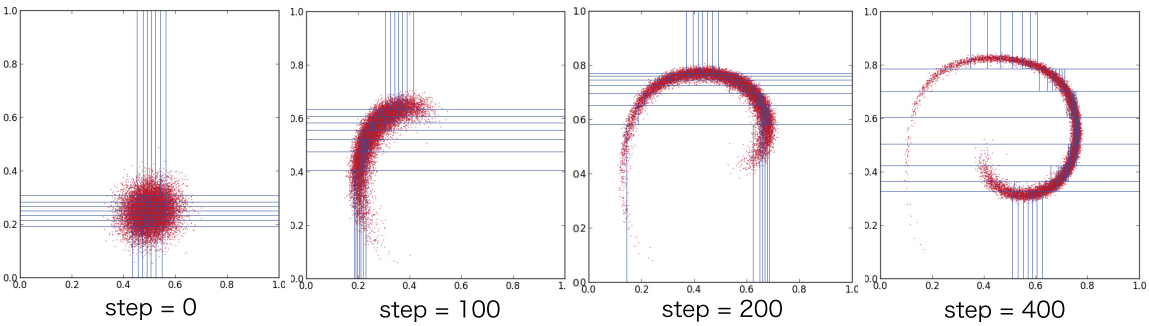

図 18　2 次元動的領域分割による渦速度場中の粒子計算のスナップショット

Fig. 18　Movement of particles with dynamically changed 2D-decomposition.

512 台まで変化させる．計算領域の分割方法は，スライスグリッド法による 2 次元領域分割とする．10,000 ステップの計算時間を測定する．測定時間から 1 ステップの計算時間を算出し，その逆数の値に粒子数をかけたものを以下のような式で実行性能とする．

$$P = \frac{1}{T_{step}} \times N_{all} \tag{27}$$

1 ステップの計算時間を T_{step}，総粒子数を N_{all}，性能を P とした．動的負荷分散は以下に定義される均等粒子数からのずれ R が 20%を超えた場合に実行する．

$$R = \frac{N_i - N_{all}}{N_{all}} \times 100 \tag{28}$$

再整列の頻度の最適値は分からないため，100 ステップごとに再整列の頻度を変えて性能を測定する．

(2) **実行性能**　強スケーリングの結果を**図 19** に示す．図中赤色は粒子数 10^7 個の測定結果を，青色は粒子数 1.6×10^8 個の測定結果を表す．四角が動的負荷分散を行わない場合を，三角が行う場合を表す．ひし形は同じ粒子数の均一粒子分布，一定速度場の計算に対する性能で目標値を示す．黒の点線は，4 GPU での性能を GPU の台数に比例させて引いた，性能の理想直線である．10^7 個の粒子を 4 GPU で計算させた場合の性能と，1.6×10^8 個の粒子を 64 GPU で計算した場合の性能値は直線上の値となり，弱スケーリングが達成されている．図 19 から，動的負荷分散を行うことにより最大で 6 倍程度性能が向上していると分かる．4 章の均一粒子分布で一様速度場のもとでの計算は動的負荷分散が必要なく，この計算と比較すると，不均一粒子分布で回転速度場の場合でも，動的負荷分散を導入することにより 90%程度の性能を達成している．各ステップで 1 つの小領域の境界を横切る粒子の移動量は，512 GPU を用いて動的負荷分散を行う場合に全粒子数の 1〜2%程度である．したがって，10,000 ステップでの動的負荷分散の実行回数は 3 回となっている．

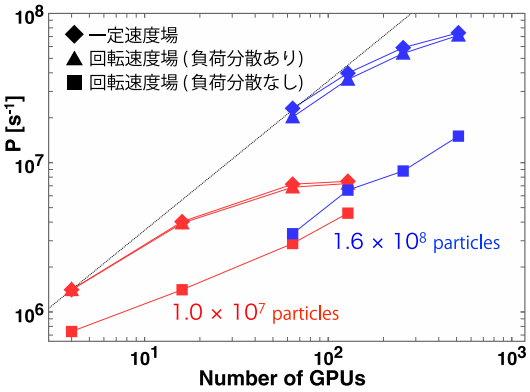

図 19　TSUBAME 2.0 を用いた強スケーリングによる性能評価
Fig. 19　Strong scaling for the computation of passive scalar particles with dynamically changed decomposition on TSUBAME 2.0.

(3) **計算時間の内訳**　512 GPU を用いた場合の主要項目の計算時間の内訳を**図 20** に示す．粒子分布の不均一により測定時間は各 GPU で異なり，全体の計算時間は最も遅い GPU の時間に律則されるため，それぞれの項目について各 GPU で 1 ステップあたりの平均計算時間を算出した中の最大値を示している．動的負荷分散 1 回の実行時間は 43 msec 程度であった．実行回数が少ないため負荷分散によるオーバヘッドは無視できる程度である．

6.3　CPU による粒子計算との比較

緒言で述べたとおり，GPU は CPU と比較して計算性能が高く粒子計算の高速化が期待できる．本論文では動的負荷分散を定量的に評価するために非常に簡単なパッシブ・スカラー粒子計算を対象としている．そこで，CPU に対して全体の計算時間の短縮化に注目するよりも，個々のプロセスの所要時間を CPU と GPU で比較した．6.2 節の強スケーリングで行った 64 GPU を用いた 1,000 万個の粒子計算に対して，同じ計算条件で 64 CPU core を用いて計算する場合の 1 ステップあたりの計算時間の内訳を**図 21** と

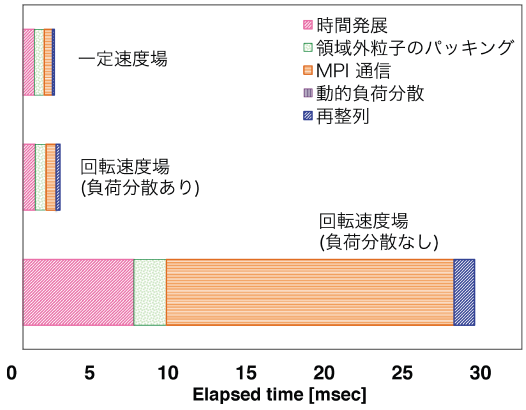

図 20　図 19 の 512 GPU 使用時の計算時間の内訳
Fig. 20　Break down of Fig. 19 with 512 GPUs.

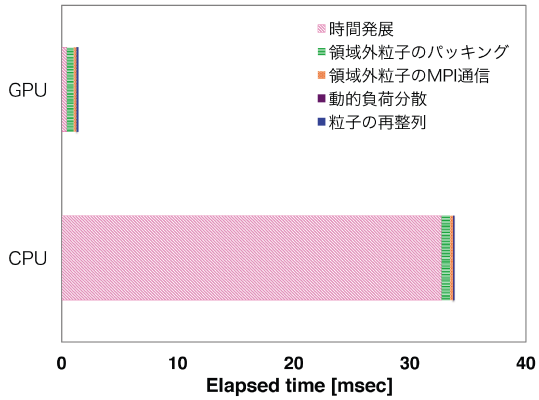

図 21　図 19 における 1,000 万個の粒子計算に対する 64 GPU 使用時の計算時間と CPU での計算時間の比較
Fig. 21　Comparison of the CPU time with the break down of Fig. 19 with 64 GPUs.

表 1　図 21 の測定値の詳細
Table 1　Measured value as the same on Fig. 21.

	時間発展	パッキング	MPI 通信	負荷分散	再整列
GPU:	0.46	0.58	0.26	0.0071	0.13
CPU:	32.75	0.73	0.26	0.0519	0.05

単位：msec

表 1 に示す．

GPU における負荷分散 1 回の実行時間が 28 msec 程度であるのに対して，CPU 間の動的負荷分散では 144 msec 程度であった．これはセル ID を割り振る際に領域内の粒子数についての繰返し文を実行するためである．複数 CPU の場合も負荷分散の実行頻度が 3,000 ステップに 1 回程度であるため，1 ステップあたりの計算時間に占める動的負荷分散のオーバヘッドは無視できる程度である．

図 21 から，本論文のような単純な粒子計算でさえも CPU は時間発展部分の計算がほとんどの時間を占めており，64 CPU core での計算では 64 GPU を用いる場合の 60 倍程度の計算時間となっている．時間発展部分の計算時間は粒子計算の内容に大きく依存するが，ほとんどの場合は演算律速であるため CPU と GPU のピーク演算性能比になり，CPU の方が実行性能は低いが高いスケーラビリティを持つといえる．

7.　結論

GPU スパコンにおいて実現が困難であった粒子法計算の動的負荷分散を，パッシブ・スカラー粒子の計算に対して実現するアルゴリズムを提案し実装を行った．隣接領域との近傍の粒子探索では，`atomic` 関数を使用する場合と比較して 10 倍近く高速化することに成功した．また，GPU 間の通信の繰返しにより生じるメモリの断片化を粒子の再整列により効果的に解消する方法を提案し，再整列の頻度と計算時間の関係をモデル化した．

性能評価を東京工業大学の GPU スパコン TSUBAME 2.0 を用いて行った．初期条件で粒子を均等配置し，一様速度場により粒子を移動させる動的負荷分散の必要のない計算をベンチマーク問題として取り上げ，GPU 間通信と粒子の再整列の検証を行うために，弱スケーリングと強スケーリングを行った．弱スケーリングでは，64 GPU 使用時に並列化効率 97%以上を達成した．強スケーリングでは，再整列の頻度を 100 回に 1 回とした場合に，1 GPU に対して，4 GPU で 2.0 倍，16 GPU で 6.2 倍，64 GPU で 15.6 倍の性能向上を達成した．さらに，再整列の頻度を変えて計算時間を測定した結果，ステップ数の 2 乗で増加する計算時間を，再整列の利用により線形増加に抑える効果があることを明らかにした．粒子の再整列の頻度と計算時間の関係のモデルを提案し，モデル値と測定値が良い一致を示した．これにより，他の計算に対しても再整列の最適頻度を予測する見通しが得られた．

スライスグリッド法による 2 次元領域分割の動的負荷分散を GPU 間で実現する実装を構築した．渦速度場のような非対称に粒子が分布する問題をベンチマークとして取り上げ，64 GPU を用いた検証では，GPU 間で均一な粒子分布になることが確認できた．回転速度場での粒子計算に対して，最大 512 GPU を用いた強スケーリングを行った．1.6×10^8 個の粒子を用いた場合，動的負荷分散を行う場合も，計算負荷の均等な計算の 90%程度の性能を達成し，動的負荷分散を行わない場合と比較して，512 GPU を用いた計算で 6 倍以上の性能向上を達成した．

謝辞　本研究の一部は科学研究費補助金・基盤研究（B）課題番号 23360046「GPU スパコンによる気液二相流と物体の相互作用の超大規模シミュレーション」，科学技術振興機構 CREST「次世代テクノロジーのモデル化・最適化による低消費電力ハイパフォーマンス」および「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」から支援をいただいた．記して謝意を表す．

参考文献

[1] NVIDIA: CUDA Programming Guide (2012), available from ⟨http://developer.download.nvidia.com/compute/DevZone/docs/html/C/docs/CUDA_C_Programming_Guide.pdf⟩.

[2] 豊田将千，岩崎　慶：マルチ GPU による流体シミュレーションの高速な可視化（感性情報処理とマルチメディア技術および一般），映像情報メディア学会技術報告，Vol.36, No.19, pp.13–14 (2012).

[3] Shimokawabe, T., Aoki, T., Ishida, J., Kawano, K. and Muroi, C.: 145 TFlops Performance on 3990 GPUs of TSUBAME 2.0 Supercomputer for an Operational Weather Prediction, *Procedia Computer Science*, *Proc. International Conference on Computational Science, ICCS 2011*, Vol.4, No.0, pp.1535–1544 (2011).

[4] Shimokawabe, T., Aoki, T., Takaki, T., Endo, T., Yamanaka, A., Maruyama, N., Nukada, A. and Matsuoka, S.: Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer, *Proc. 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11*, pp.3:1–3:11, ACM (2011).

[5] 成瀬　彰，住元真司，久門耕一：GPGPU 上での流体アプリケーションの高速化手法：1 GPU で姫野ベンチマーク 60 GFLOPS 超（高性能計算とアクセラレータ），情報処理学会研究報告［ハイパフォーマンスコンピューティング］，Vol.2008, No.99, pp.49–54 (2008).

[6] 王　嫻，青木尊之：3 次元格子ボルツマン法による非圧縮流体ソルバーの GPU による加速（OS4.GPGPU による計算力学 (1)，オーガナイズドセッション），計算力学講演会講演論文集，Vol.2009, No.22, pp.741–742 (2009).

[7] Krog, O.E. and Elster, A.C.: Fast GPU-Based fluid simulations using SPH, *Proc. 10th International Conference on Applied Parallel and Scientific Computing, PARA '10*, Vol.2, pp.98–109, Springer-Verlag (2012).

[8] Zhang, F., Hu, L., Wu, J. and Shen, X.: A SPH-based method for interactive fluids simulation on the multi-GPU, *Proc. 10th International Conference on Virtual Reality Continuum and Its Applications in Industry,*

VRCAI '11, pp.423–426, ACM (2011).
[9] Chiemi, H., Hitoshi, G., Hiroyuki, I. and Abbas, K.: GPU-acceleration for Moving Particle Semi-Implicit method, *Computers & Fluids*, Vol.51, No.1, pp.174–183 (2011).
[10] 後藤仁志，堀智恵実，五十里洋行，Khayyer, A.：GPU による粒子法半陰解法アルゴリズムの高速化，土木学会論文集 B，Vol.66, No.2, pp.217–222 (2010).
[11] Beutel, A., Mølhave, T. and Agarwal, P.K.: Natural neighbor interpolation based grid DEM construction using a GPU, *Proc. 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, *GIS '10*, pp.172–181, ACM (2010).
[12] 原田隆宏，政家一誠，越塚誠一，河口洋一郎：GPU 上での粒子法シミュレーションの空間局所性を用いた高速化，計算工学講演会論文集，Vol.13, No.1, pp.287–288 (2008).
[13] 安部拓也，井村誠孝，池田　聖，眞鍋佳嗣，千原國宏：水滴表現のための粒子ベース液体シミュレーション，映像情報メディア学会誌，Vol.63, No.7, pp.951–956 (2009).
[14] 原田隆宏，政家一誠，越塚誠一，河口洋一郎：複数の GPU を用いた粒子法シミュレーションの並列化，情報処理学会論文誌，Vol.49, No.12, pp.4067–4079 (2008).
[15] Xu, J., Qi, H., Fang, X., Lu, L., Ge, W., Wang, X., Xu, M., Chen, F., He, X. and Li, J.: Quasi-real-time simulation of rotating drum using discrete element method with parallel GPU computing, *Particuology*, *Multiscale Modeling and Simulation of Complex Particulate Systems*, Vol.9, No.4, pp.446–450 (2011).
[16] Kijsipongse, E. and U-ruekolan, S.: Dynamic load balancing on GPU clusters for large-scale K-Means clustering, *2012 International Joint Conference on Computer Science and Software Engineering* (*JCSSE*), pp.346–350 (2012-06).
[17] Ou, Y., Chen, H. and Lai, L.: A dynamic load balance on GPU cluster for fork-join search, *2011 IEEE International Conference on Cloud Computing and Intelligence Systems* (*CCIS*), pp.592–596 (2011).
[18] Nakashima, H., Miyake, Y., Usui, H. and Omura, Y.: OhHelp: A scalable domain-decomposing dynamic load balancing for particle-in-cell simulations, *Proc. 23rd International Conference on Supercomputing*, *ICS '09*, pp.90–99, ACM (2009).
[19] 武宮　博，川崎琢冶，樋口健二：粒子モデルのための動的負荷分散アルゴリズム，情報処理学会全国大会講演論文集，Vol.55, No.1, pp.56–57 (1997).
[20] 河野　瑛，田浦健次朗：重心ボロノイ分割を用いた並列粒子法のための動的負荷分散法，情報処理学会研究報告［ハイパフォーマンスコンピューティング］，Vol.2011, No.66, pp.1–8 (2011).
[21] Thrust Quick Start Guide, available from ⟨http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/Thrust_Quick_Start_Guide.pdf⟩.
[22] Optimizing Parallel Reduction in CUDA, available from ⟨http://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf⟩.

都築 怜理　（学生会員）

1988 年生．2011 年東京工業大学理学部物理学科卒業．2013 年同大学大学院総合理工学研究科創造エネルギー専攻修士課程修了．2013 年同大学院総合理工学研究科創造エネルギー専攻博士課程進学．2012 年情報処理学会コンピュータサイエンス奨励賞受賞．粉体工学会等会員．

青木 尊之　（正会員）

1960 年生．1983 年東京工業大学理学部応用物理学科卒業．1985 年同大学大学院総合理工学研究科エネルギー科学専攻修了．1985 年富士通研究所厚木研究所入社．1986 年東京工業大学大学院総合理工学研究科助手．1997 年同大学原子炉工学研究所助教授．2001 年同大学学術国際情報センター教授．数値流体力学，計算力学，GPU コンピューティングの研究に従事．文部科学大臣表彰，日本応用数理学会業績賞，ACM ゴードンベル賞ほか．日本機械学会フェロー，日本応用数理学会等会員．

下川辺 隆史　（正会員）

1983 年生．2005 年東京工業大学理学部物理学科卒業．2007 年同大学大学院理工学研究科基礎物理学専攻修士課程修了．2012 年同大学院総合理工学研究科創造エネルギー専攻修了．博士（理学）．2012 年同大学学術国際情報センター特任助教．2013 年同大学同センター助教．ペタスケールの大規模物理計算の研究に従事．2011 年 ACM ゴードンベル賞・特別賞受賞．日本応用数理学会，IEEE，IEEE-CS，ACM 各会員．