Vol.2012-HPC-133 No.37
2012/3/27

複数ノードの GPU による大規模パッシブ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]　青木尊之[†1]
下川辺隆史[†1]　王　　嫻[†1]

GPU による超大規模流体計算の速度場に対し、数千万〜数億個に及ぶ粒子を移動させてスカラー量の分布を把握するため、パッシブスカラー粒子計算を GPU で高速に行う方法を提案する。GPU 間の通信や、粒子の移動により生じるメモリ使用量の増加、負荷分散の悪化が大幅な性能の低下の原因となる。移動する粒子のみを選択的に通信することで、GPU 間のデータ転送量を最小限に抑え、粒子番号の再整列によりメモリ使用量を抑える。さらに動的負荷分散を行うことで負荷を均等に分散させて、高速に計算する実装を構築する。性能評価は、GPU を 4224 台搭載したスパコン TSUBAME 2.0 を用いて行う。

Dynamic load balance and strong scaling of Multi-GPU computation for passive scalar particles

Satori Tsuzuki,[†1] Takayuki Aoki,[†1]
Takashi Shimokawabe[†1] and Wang Xian[†1]

Particle tracing is an effective method on mesh-based applications in computational fluid dynamics (CFD). Graphics processing units (GPUs) make it possible to carry out the computation of billion passive-scalar particles on the velocity field. Data transfer among GPUs is a big overhead. Defragmentation of GPU memory in the time step and imbalance of particle distribution also reduce the performance. In this study, we present an effective method to minimize the CPU-GPU communication and optimize the frequency of the renumbering by sorting. In addition, the load balance of the particle computation should be considered by dynamically changing the domain decomposition. The performance for weak and strong scaling are examined on the TSUBAME 2.0 supercomputer and it is found that our method greatly improves the strong and weak scalability for the problems with over billion particles.

1. 緒　　言

グラフィックスボードに搭載されている GPU(Graffic Processing Unit) は、画像処理専用のプロセッサであり、ピーク演算性能が 1 TFlops に及ぶ非常に高い浮動小数点演算能力をもっている[1)]。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 GPGPU 研究が盛んに行われている。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[2)] のリリースにより、GPU の利用は飛躍的に広まった[3)4)]。

複数 GPU の計算では、単一 GPU と比較して大規模な計算が可能な反面、計算結果のデータ量も増大する。流体計算では、濃度などのスカラー分布の定量評価のために、粒子をパッシブに追跡する方法が有効である。本研究の目的は、GPU による超大規模流体のステンシル計算に対し、計算される速度場に対して数千万〜数億個の粒子を移動させスカラー量の分布を把握するために、大規模パッシブ・スカラー粒子計算を GPU で行う。GPU を用いて高速に計算する実装を行い、その性能評価を行う。パッシブ・スカラー粒子による計算は、それ自体の利用価値が高いだけでなく、粒子法による流体計算である SPH 法や、MPS 法や個別要素法 (DEM) でも利用できるため、非常に有用である[5)]。

複数 GPU の流体計算では、速度場などの計算結果は各 GPU の VRAM 上に存在するため、パッシブスカラー粒子計算も複数 GPU を用いて行う必要がある[6)]。計算領域を小領域に分割し、分割された各小領域に GPU を割り当てる。他の GPU が担当する小領域に移動する粒子が発生するため、GPU の VRAM 間でのデータ転送が必要となり、計算の大きなオーバヘッドとなる。粒子の移動によりメモリに未参照領域が離散的に発生し、メモリ使用量が増加すること、GPU 間の粒子数のばらつきによる負荷バランスの悪化も、大幅な性能低下を引き起こす。

第一段階として、各小領域の大きさを固定し、GPU 間のデータ通信、メモリ使用量を抑えた実装を構築する。第二段階として、各小領域の大きさを動的に変更し、粒子数を等しく分散することで、負荷の大きい計算に対しても、高速に計算する方法を示す。

2. 計 算 方 法

計算領域を複数の小領域に分割し、分割された各小領域に GPU を 1 台ずつ割り当てる。

†1 東京工業大学
Tokyo Institute of Technology

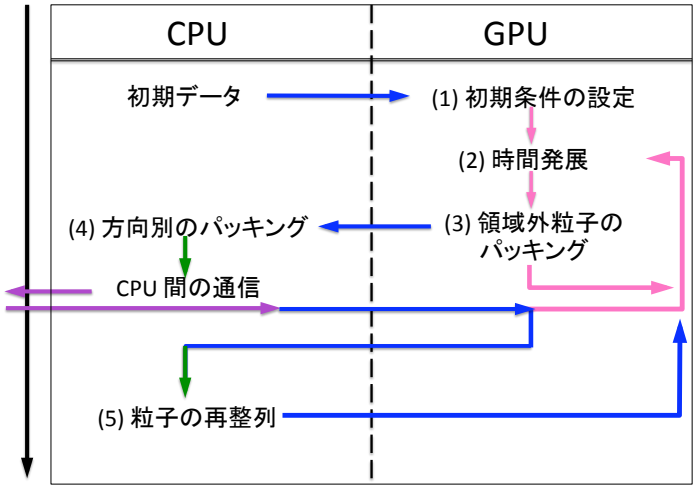

図 1 固定領域分割法の計算方法
Fig. 1 Flow diagram of the time integration for passive scalar particles on Multi-GPU platform.

区別のため、各小領域に番号を割り振る。各小領域の大きさを固定して計算する方法 (固定領域分割法) と、動的に変更して計算する方法 (動的負荷分散法) がある。それぞれについて以下に示す。

2.1 固定領域分割法の場合

各小領域で行う計算手順を図 1 に示す。ホスト (CPU) 側で、初期データを用意する。粒子の情報は、粒子の座標と、粒子の所属する小領域の番号の 2 種類とし、各小領域でメモリを確保する。メモリの未参照領域には、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格納する。これらをデバイス (GPU) 側にコピーする。(2) 時間発展では、以下に示す 4 次のルンゲクッタ法を用いて解析解の速度場を与えて時間発展させる。

$$k_1 = hf(x_n, y_n) \tag{1}$$
$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \tag{2}$$
$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \tag{3}$$
$$k_4 = hf(x_n + h, y_n + k_3) \tag{4}$$
$$x_{n+1} = x_n + h \tag{5}$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{6}$$

時間発展により、隣接する小領域に移動する粒子 (領域外粒子) が発生する。移動する粒子の情報を、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 CPU にコピーする必要がある。GPU から CPU に全粒子の情報をコピーした場合、大きなオーバーヘッドとなる。また、隣接する小領域の数は、計算領域全体を 2 次元分割した場合で 8 方向、3 次元分割した場合で 26 方向存在する。隣接する小領域のそれぞれについて順番に GPU - CPU 間通信を繰り返すことは、性能を低下させる。そこで、(3) 領域外粒子のパッキングでは、 GPU - CPU 間の通信回数、通信量を最小限に抑えるため、GPU 上で領域外粒子をすべて選択し、1 つのバッファに集める (パッキングする)。GPU - CPU 間では、領域外粒子を集めたバッファについてのみ、1 回だけデータ通信を行う。粒子のパッキングは、CUDPP ライブラリの `cudppCompact` を利用している[7]。GPU 上で所属領域の判定用の配列を用意し、粒子が領域外の場合は 1 を、領域内の場合は 0 を格納する。判定用の配列と、粒子の配列、出力用の配列を `cudppCompact` に入力として与え、判定用の配列が 1 の粒子 (すなわち領域外粒子) のみを選出して、出力用の配列を出力する。同時に、パッキングした粒子数も出力する。粒子数をホスト側に転送し、この値と同じ大きさのバッファを をホスト側で確保した後、デバイス 側からホスト 側にパッキングされた粒子をコピーする。ホスト側で受け取ったデータは、粒子のはずれた方向に関係なくすべての領域外粒子のものであるから、(4) 粒子の方向別のパッキングで、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い、隣接する各小領域に MPI 通信を用いて送信する。受信したデータを デバイス 側にコピーして、メモリの使用領域の最後尾部に加え、領域外粒子の送受信が完了する。

領域外粒子に使用されていたメモリは、粒子の移動した後は未参照メモリとなる。計算が進むにつれて、未参照メモリの量は増加していく。図 2 はその様子を 1 番の小領域を例に模

Vol.2012-HPC-133 No.37
2012/3/27

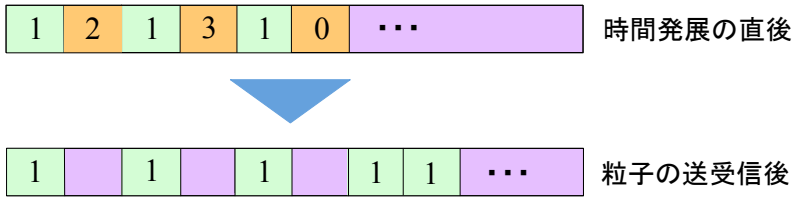

図 2 GPU メモリの断片化
Fig. 2 Defragmentation of the GPU memory as increased in the time step.

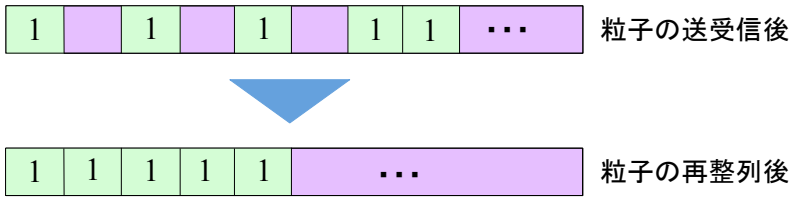

図 3 粒子番号の再整列
Fig. 3 Re-numbering by sorting

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、紫色が未参照メモリを表す。橙色が、時間発展によって他の小領域に移動する粒子が、移動前に使用しているメモリを表す。この部分は、粒子の送受信後、未参照メモリとなる。一方で、図 2 の下図の様に、新たに流入した粒子が使用する領域は、バッファの最後尾になる。GPU で計算する場合、スレッドはバッファの使用領域全体と等しい数だけ用意するので、未参照メモリによるメモリの断片化により、性能は低下する。

粒子の移動によるメモリの断片化を防ぐため、(5) で粒子の再整列をホスト側で行う。図 3 はその様子を模式的に表したものである。粒子の所属領域を表すバッファにおいて、未参照メモリの領域には小領域の最大番号より大きな数字が入っている。ソートかけることにより未参照メモリはバッファの後尾に集められる。これにより、粒子の情報をバッファの先頭からつめる事ができる。ホスト側でソートを行うため、デバイス側からホスト側にバッファをコピーする必要がある。バッファの参照領域全体の GPU - CPU 間通信を行うため、粒子の再整列を毎回行う事は大きなオーバーヘッドとなり非効率である。そのため、ソートの回数を抑えて使用する。

2.2 動的負荷分散法の場合

固定領域分割法では、計算が進むにつれて GPU 間での粒子数にばらつきが生じる。並列計算では、計算時間は最も遅いプロセッサの計算時間に依存するため、負荷分散の悪い計算では著しく性能が低下する。各 GPU で粒子数が一定となる様、各小領域の大きさを動的に変更することで、負荷を分散する方法が考えられる。

動的負荷分散を行う計算方法を図 4 に示す。図 1 の方法で領域外粒子の通信を終えた後、各 GPU 間の粒子数が等しくなる様、再び GPU 間で通信を行い、全計算領域を再分割する方法である。以下この過程について、全計算領域を 1 次元分割した場合を例に説明する。全粒子数を N_{total} とし、GPU の数を p とすると、 1 GPU 当たりの平均粒子数は N_{total}/p である。(6) では、小領域の番号を i としたとき、式 (7)、(8) の漸化式で定義される粒子数 ΔN_i を導入する。ΔN_i は、$\Delta N_i > 0$ ならば $i+1$ 番目の小領域に送信する粒子数、$\Delta N_i < 0$ ならば $i+1$ 番目の小領域から受信する粒子数である。

$$\Delta N_0 = N_0 - N_{total}/p \tag{7}$$
$$\Delta N_i = \Delta N_{i-1} + N_i - N_{total}/p \qquad (i \geq 1) \tag{8}$$

式 (7)、(8) の漸化式に従い、小領域の 0 番から逐次的に ΔN_i を決定する。受信する ΔN_{i-1} は、$\Delta N_{i-1} > 0$ ならば、$i-1$ 番目の小領域から受信する粒子数、$\Delta N_{i-1} < 0$ ならば、$i-1$ 番目の小領域に送信する粒子数である。ΔN_{i-1}、ΔN_i により、隣接する小領域に送信する粒子数が決まる。この方法が成立する条件は、i 番目の小領域で、$i-1$、$i+1$ 番目の小領域に送信する粒子数の和が、自身の小領域のもつ全粒子数よりも小さいことである。

$$(-\Delta N_{i-1}) + \Delta N_i \leq N_i \tag{9}$$

初期条件を設定する際、各小領域の粒子数が等しくなる様に全計算領域を分割し、さらに Δt を小さくして移動量を抑えることでこの条件を満たす事ができる。 (7) では、 隣接する

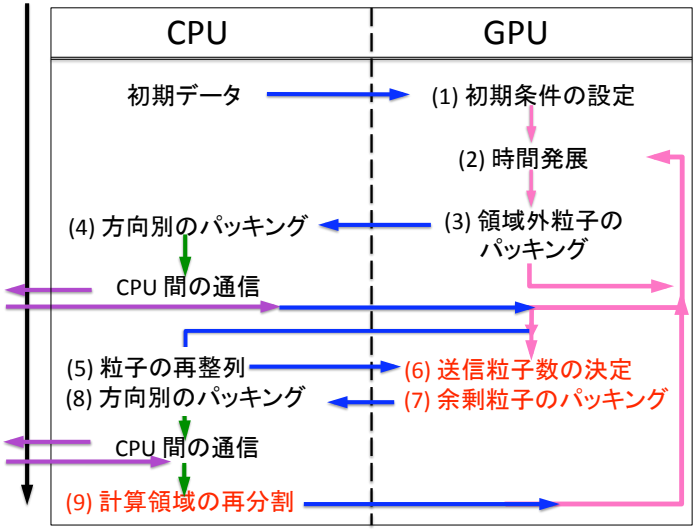

図 4 動的負荷分散法を行う場合の計算方法.
Fig. 4 Flow diagram of the time integration for passive scalar particles on Multi-GPU platform.

小領域に送信する粒子を (3) 領域外粒子のパッキングと同様の方法で、GPU 上から取得する。隣接する小領域に送信する粒子は、隣接する小領域に最近傍の粒子から順に必要数分送信する。この様子を図 5 に示す。隣接する小領域との境界線から l だけ離れた微小領域を境界領域とし、境界領域に含まれる全ての粒子をホスト側にコピーする。コピーしたバッファに対して、計算領域の分割方向に粒子の座標によるソートをかけ、昇順に並べ替える。上位の小領域は自身の計算領域の上部と隣接しているため、バッファの上部から、同様に下位の小領域にはバッファの下部から必要数を送信する。粒子を受信後、受信した粒子の座標をもとに、(9) で小領域の境界線を変更し、全計算領域を再分割する。この計算方法で逐次処理は (6) 送信粒子数の決定のみであり、それ以外はすべて各小領域が並列に行う。

3. 性 能 評 価

3.1 Weak Scaling による性能評価

固定領域分割法により計算する実装の性能評価を Weak Scaling により行う。Weak Scaling は、GPU 1 台あたりの粒子数を固定し、GPU 数を変化させて計算時間を測定する方法である。全粒子数は GPU 数に比例して増加する。

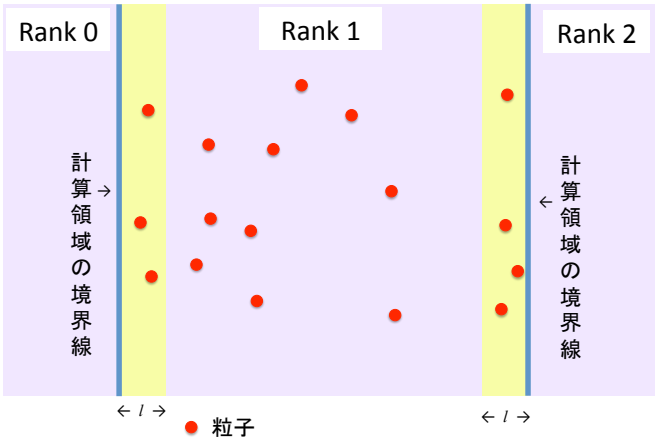

図 5 境界領域の粒子のパッキング.
Fig. 5 Process of packing particles on voundary field.

(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。TSUBAME 2.0 には、NVIDIA 社製 GPU である Tesla M2050 が 4224 基搭載されている。CPU は、Intel 社製の Xeon X5670 が搭載されている。TSUBAME 2.0 の 1 ノードに搭載された Xeon X5670 6CPU-core 2 基、Tesla M2050 3 基のうち、1 ノードあたり、2 CPU - core、2 GPU を利用し、複数 GPU を利用する際は、ノード数を増やして実行する。TSUBAME 2.0 では、CUDA は、最新の 4.0 まで対応している。本研究では CUDA 3.2 を用いる。

(2) **計算条件**

2 次元のスカラー粒子を、一定速度場 $u = v = 1.0$ で 4 次ルンゲクッタ法により時間発展させる。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。また、格子数 $nx = ny = 128$ として、$\Delta x = Lx/nx$ 、$\Delta y = Ln/ny$、とする。1 GPU あたりの粒子数は 4194304 ($= 2^{22}$) 個とし、全計算領域に均一に分布させる。全計算領域の分割方法は、x 方向の分割数と y 方向の分割数の等しい 2 次元領域分割とする。

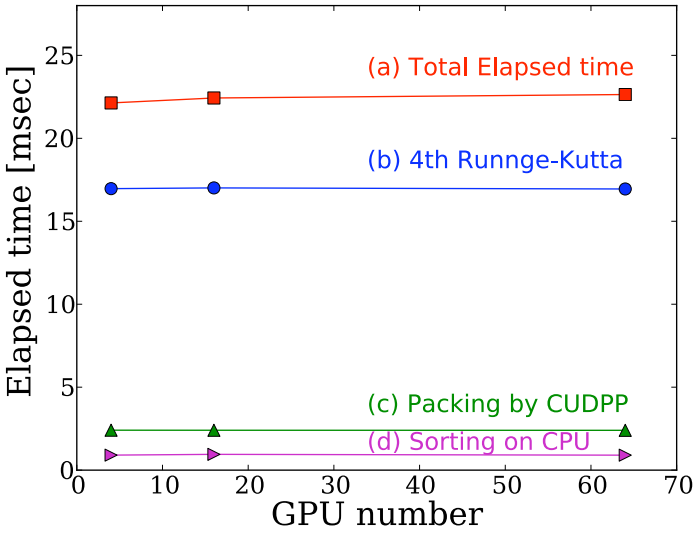

図 6 TSUBAME 2.0 を用いた Weak Scaling による性能評価
Fig. 6 The result of weak scaling on TSUBAME 2.0

時間ステップは、$\Delta t = 0.001$ とする。境界条件は周期境界条件としている。GPU 数を 4 台、16 台、64 台と変化させて、 500 ステップの計算時間をそれぞれ測定する。測定時間を全ステップ数 (500 ステップ) で割り、1 ステップあたりの計算時間を算出する。ソートは 500 ステップのうち、250 ステップ目に 1 回行う。(1) 初期条件の設定は測定対象に含めず、計算方法の図 1 における (2) 以降にかかる時間を測定する。

(3) 性能

測定された性能を図 6 に示す。GPU が 64 台の場合、粒子数は GPU が 4 台の場合の 16 倍になる。1 GPU あたりの粒子数が 4194304 ($= 2^{22}$) 個であるから、4 GPU の場合で粒子数は、約 1650 万個以上、64 GPU の場合で粒子数は、2 億 5000 万個以上に達する。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 台用いた場合にも 97% 以上を達成し、良いスケーリングを示しているとわかる。

3.2 Strong Scaling による性能評価

固定領域分割法により計算する場合の Strong Scaling による性能評価を行う。Strong Scaling では、全体の粒子数を一定にし、GPU 数を変化させて計算時間を測定する。GPU 1 台あたりの粒子数は GPU 数が増えるにつれて減少する。100 回に 1 回ソートをかけた場合と、ソートをかけなかった場合の 2 通りの方法で計算時間を測定し、計算結果を比較する。具体的な計算条件について次に示す。

(1) 計算条件

Weak Scaling の場合と同様、2 次元のスカラー粒子を、一定速度場 $u = v = 1.0$ で 4 次ルンゲクッタ法により時間発展させる。時間ステップは、$\Delta t = 0.001$ とする。このとき 1 ステップ、1 GPU あたりの粒子の移動量は 16000 個程度となる。格子数 $nx = ny = 128$ として、$\Delta x = Lx/nx$ 、$\Delta y = Ln/ny$、とする。粒子数は総粒子数を 16777216 ($= 2^{44}$) 個で固定し、GPU を 1 台、4 台、16 台、64 台と変化させ、1500 ステップの計算に要する時間を測定する。全計算領域の分割方法は、x 方向の分割数と 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、その逆数の値を実行性能 (Performance) として算出する。実行環境は Weak Scaling の場合と同様である。Weak Scaling 同様、測定対象には (1) 初期条件の設定は含めず、計算方法の図 1 における (2) 以降にかかる時間を測定する。

(2) 性能

性能の測定結果を図 7 に示す。赤線がソートを行う場合の性能を、青線がソートを行わない場合の性能を示す。ソートを行った方がいずれも性能が向上している。図 7 と同じ計算条件で計算した場合の、各ステップの計算時間の変化を、図 8 に示す。CPU 上でのソートに必要となる GPU - CPU 通信や、ソート自体により、本来図 8 の縦軸にはソートをかけた事で生じる 270 msec 前後のピークがあるが、見やすさのため間引いてある。図 8 から、計算が進むにつれ増加していた 1 ステップの計算時間が、ソートをかける毎に元に戻っていると分かる。

ソートの頻度を変えて、ソートの効果が最も良く現れる場合について調べる。図 9 に、同様の計算条件のもと、64 GPU を用いて 6000 ステップを計算する場合に、ソートを行わない場合、20 回に 1 回行う場合、100 回に 1 回行う場合、1000 回に 1 回行う場合、2000 回に 1 回行う場合の 5 通りについて、各ステップの計算時間の積算値を測定した結果を示す。黒が ソートを行わない場合、青が 2000 回に 1 回ソートを行う場合、桃色 が 1000 回に 1 回行う場合、赤が 100 回に一回行う場合を表す。ステップ数が 6000 の段階で、1 ステップ

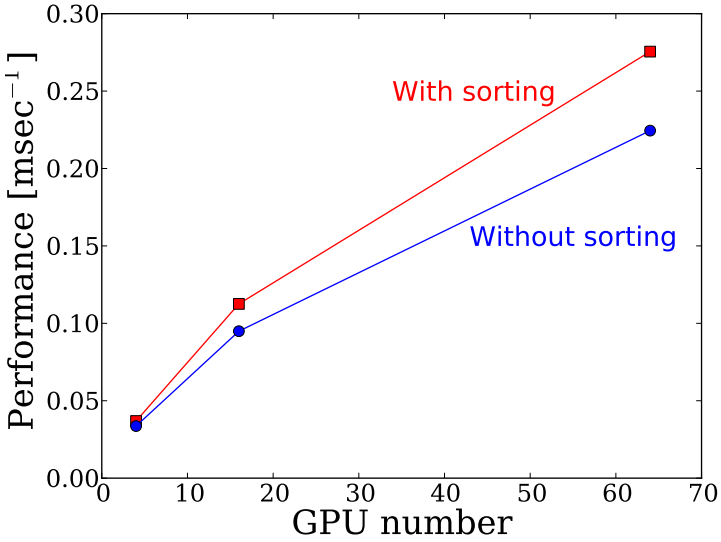

図 7 TSUBAME 2.0 を用いた Strong Scalig による性能評価
Fig. 7 The result of strong scaling on TSUBAME 2.0

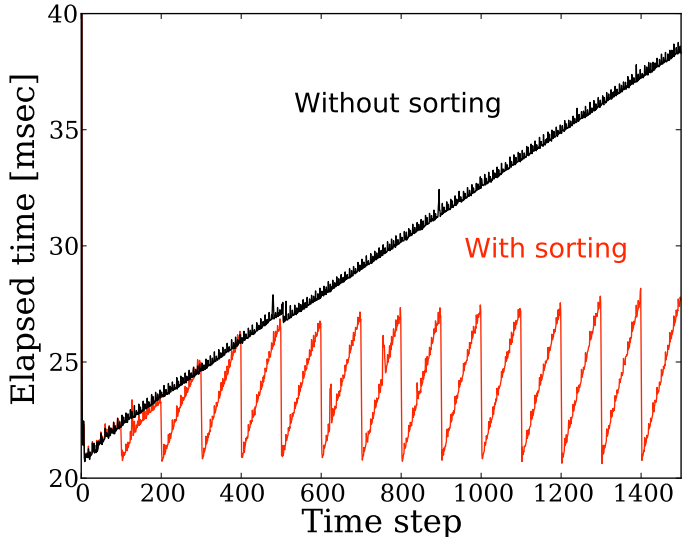

図 8 100 回に 1 回ソートを行った場合の 1 ステップあたりの計算時間の変化
Fig. 8 Time variation by sorting every 100 time steps

あたりの計算時間の積算値は、100 ステップに 1 回ソートを行う場合に、行わない場合の 1/3 以下となることが分かる。20 回に 1 回に 1 度ソートを行う場合、100 回に 1 度ソートを行う場合同様、計算時間の積算値はほぼ一定の割合で増加してるものの、100 ステップに 1 回ソートを行う場合よりも計算時間がかかり、100 ステップに 1 回ソートを行う場合が 1 番よいことが分かる。

3.3 ソートと計算時間の関係のモデル化

ソートの回数と計算時間の関係を明らかにするため、ソートを行う場合と行わない場合の二つの場合について、モデル化を行う。得られた関係式から、計算時間に対するソートの回数の最適解を決定する。

(1) **ソートを行わない場合**

i ステップ目 1 回にかかる計算時間を 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に N_0 に等しいので、比例定数を α として T_0 は、

$$T_0 = \alpha N_0 \tag{10}$$

と書ける。 1 ステップ目の粒子数は、流入した粒子数を ΔN_{in}、流出した粒子数を ΔN_{out} とすると、

$$N_1 = N_0 + \Delta N_{in} - \Delta N_{out} \tag{11}$$

である。スレッド数は、$N_0 + \Delta N_{in}$ 用意され、このうち流出した粒子数 ΔN_{out} 分については、スレッドだけを起動して計算は行わない。N_1 は、計算時間に占めるスレッドの起動時間の割合を γ として、

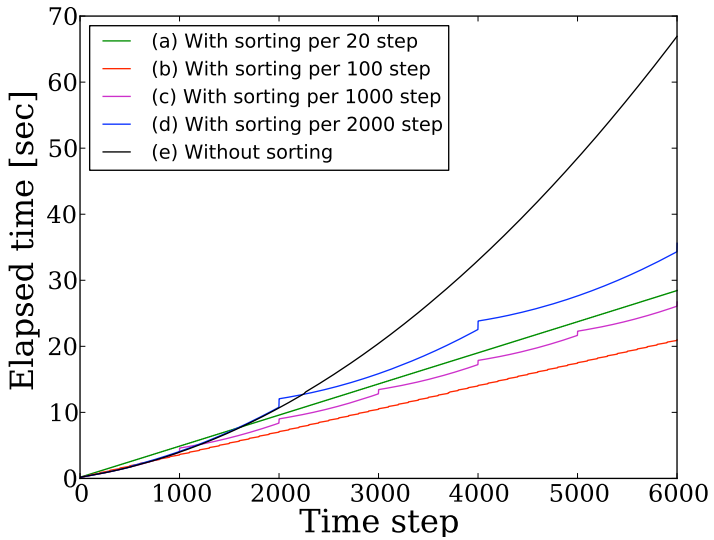

図 9 ソートの回数の違いによる積算計算時間の変化の比較
Fig. 9 Elapsed time per a step on 64 GPU by sorting every 100, 1000, and 2000 steps.

$$T_1 = \alpha(N_0 + \Delta N_{in}) - (1-\gamma)\Delta N_{out} \tag{12}$$

と書ける。一定速度場であるので、$\Delta N_{in} = \Delta N_{out} = \Delta N$ として

$$T_1 = \alpha(N_0 + \gamma\Delta N) \tag{13}$$

2 ステップ目の粒子数は、式 (12) で、ΔN_{in}、ΔN_{out} がそれぞれ 2 倍になるので、

$$T_2 = \alpha(N_0 + (2\Delta N_{in})) - (1-\gamma)(2\Delta N_{out}) \tag{14}$$

である。一定速度場であるので、$\Delta N_{in} = \Delta N_{out} = \Delta N$ として、

$$T_2 = \alpha(N_0 + 2\gamma\Delta N) \tag{15}$$

3 ステップ目以降についても同様に考えられるので、i 番目のステップ 1 回の計算時間は、

$$T_i = \alpha(N_0 + i\gamma\Delta N) \tag{16}$$

となる。m 番目のステップにおける計算時間の積算値は、式 (16) の m 番目までの和であるから、

$$T_{sum}(m) = \sum_{j=0}^{m} T_j = \sum_{j=0}^{m} \alpha(N_0 + j\gamma\Delta N) \tag{17}$$

となる。式 (17) は等差数列であり、m 項までの和を求めると、

$$T_{sum}(m) = \alpha(N - \frac{\gamma\Delta N}{2})m + (\frac{\alpha\gamma\Delta N}{2})m^2 \tag{18}$$

となる。

(2) **ソートを行う場合**

ソートを S 回に 1 回 行う場合を考える。ソートをかけた際に発生する GPU - CPU 間通信とソート 1 回にかかる計算時間の合計時間を、T_{sort} とする。ソートをかける度に計算時間は 0 ステップの計算時間に戻るので、m ステップ目における積算の計算時間は、式 (17) 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、

$$T_{sort}(m) = (\frac{m}{S})\sum_{j=0}^{m} T_j + (\frac{m}{S})T_{sort} \tag{19}$$

$$= (\frac{m}{S})\sum_{j=0}^{S} \alpha(N_0 + j\gamma\Delta N) + (\frac{m}{S})T_{sort} \tag{20}$$

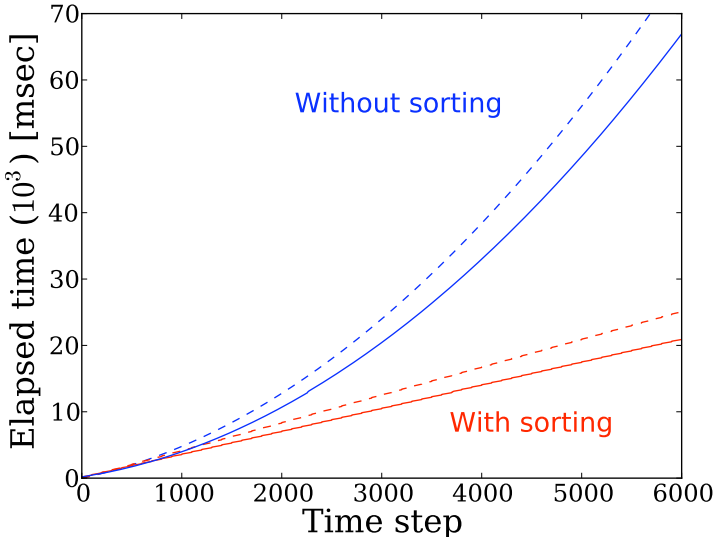

図 10 図 9 をスケーリングした結果.
Fig. 10 Scaling for the result as the same Fig.9.

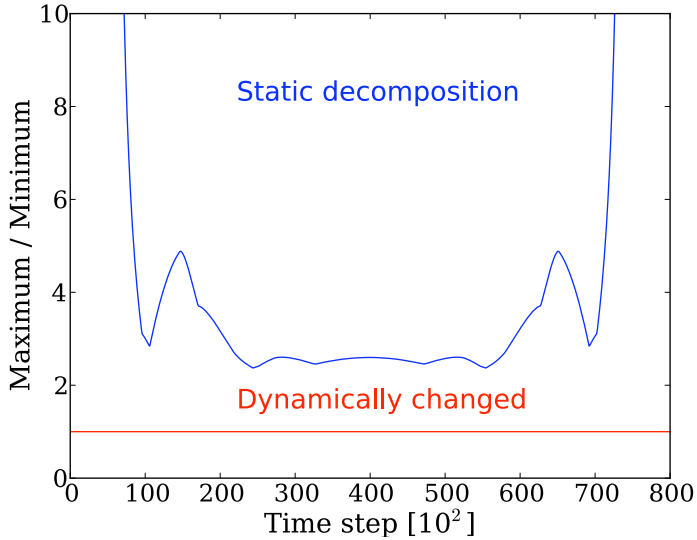

図 11 4 GPU を用いた場合の小領域間の最大粒子数と最小粒子数の比率の変化
Fig. 11 vatiarion of ratio of maximum : minimum number of particle on 4 GPU in the time step

$$= \alpha Nm + (\frac{\alpha\gamma\Delta Nm}{2})(S-1) + (\frac{m}{S})T_{sort} \tag{21}$$

となる。未定係数 α は、測定開始直後の 2 点の粒子数と計算時間から、式 (18) を用いて、$\alpha = 1.2 \times 10^{-5}$ msec とした。γ は、64 GPU を用いた際の 1 台当たりの粒子数を、単一 GPU で計算するのに必要な時間と、同じ粒子数で、スレッドだけを立てて計算は行わない場合の計算時間 (カーネル関数の起動時間) の比を測定した値 $\gamma = 6.32 \times 10^{-2}$ を用いた。

式 (18)、式 (21) と上記 α、γ の値を用いて、図 9 における、ソートを行わない場合、100 回に 1 回ソートを行った場合のそれぞれの測定結果と、今回のモデルによる計算値の比較を図 10 に示す。実線は、青が図 9 のソートを行わない場合を、赤がソートを 100 回に 1 回行う場合の測定値である。点線が、式 (18)、式 (21) によるスケーリングの結果である。図 10 の通り、測定結果をよく表現出来ている。

ソートの回数が最適値となる条件は、式 (21) が最小値をとる事である。式 (21) が極小値をとる必要条件として、

$$\frac{\partial T_{sum}(m)}{\partial S} = \frac{\alpha\gamma\Delta Nm}{2} + T_{sort}(\frac{-m}{S^2}) = 0 \tag{22}$$

がある。これを S について解くと、最適値 S_{opt} は、

$$\therefore \qquad S_{opt} = \sqrt{\frac{2T_{sort}}{\alpha\gamma\Delta N}} \tag{23}$$

と求められる。

式 (23) に、α、γ の値、別途測定した $T_{sort} = 85.4$ msec 、$\Delta N = 4194$ 個を代入すると、$S_{opt} = 231.2$ 回となる。231 回、あるいは 232 回に 1 回ソートを行った場合の計算時間の変化は、図 9 において、ソートを 1000 回に 1 回行う場合と、100 回に 1 回行う場合の間の値をとり、線形増加に近い変化になると予想できる。

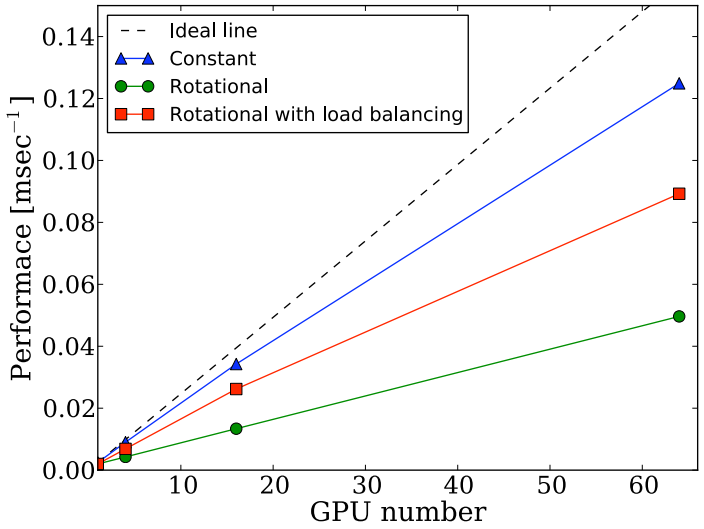

図 12 渦速度場におけるパッシブスカラー粒子計算の Strong Scaling による性能評価
Fig. 12 Strong scaling for rotational velocity field with dynamically changing decomposition on TSUBAME 2.0

3.4 動的負荷分散法の性能評価

動的負荷分散法の Strong Scaling による性能評価を行う。ベンチマーク問題として、解析解の渦速度場を与えて時間発展させ、動的負荷分散を行う場合と行わない場合を比較する。

(1) **計算条件**

2 次元のスカラー粒子を、4 次ルンゲクッタ法により時間発展させる。速度場は以下に示す回転速度場とする。

$$u(x,y,t) = -2\cos(\pi t/T)\sin(\pi x)\sin(\pi x)\cos(\pi y)\sin(\pi y) \quad (24)$$
$$v(x,y,t) = 2\cos(\pi t/T)\cos(\pi x)\sin(\pi x)\sin(\pi y)\sin(\pi y) \quad (25)$$

時間ステップは、$\Delta t = 0.0001$ とする。T は周期であり、$T = 8.0$ とする。これにより一周期に要する計算は 80000 ステップとなる。格子数 $nx = ny = 128$ として、$\Delta x = Lx/nx$ 、$\Delta y = Ln/ny$、とする。全計算領域の分割方法は、y 方向の 1 次元分割とする。初期条件は、全計算領域の上部に円形に集中して配置する。粒子数は総粒子数を 16777216 ($= 2^{44}$) 個で固定する。初めに、固定領域分割法により場合に、GPU 4 台、16 台、64 台と変化させ、一周期 (80000 ステップ) を計算し、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) を測定する。

(2) **計算結果**

固定領域分割法により、4 GPU を用いて計算した場合の、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 11 に示す。これにより、計算開始直後と終了直後に特に負荷バランスが取れていないことがわかる。そこで、計算条件は変更せず、計算開始直後 2000 ステップ目までにかかる計算時間を測定し、動的負荷分散を行う場合と行わない場合とで性能を比較する。性能結果を、図 12 に示す。緑が動的負荷分散を行わない場合の性能を、赤が動的負荷分散を行った場合の計算結果を表す。青は同じ粒子数で、粒子を全計算領域に均等に配置し、一定速度場を用いて計算した場合の性能を表す。渦速度場中におけるソートの回数の最適値は、現段階では正確には分からないため、ソートの回数を 10 ステップごとに変化させ、最も性能の高かった値 (測定的に得られた最適値) としている。動的負荷分散を行う場合、行わない場合よりも 2 倍程度性能が向上していることが分かる。

4. ま と め

大規模パッシブスカラー粒子計算を複数 GPU を用いて計算する方法を、NVIDIA 社の GPU コンピューテイング統合開発環境 CUDA を用いて開発し、Weak Scalling、Strong Scaling による性能評価を東京工業大学 TSUBAME 2.0 を用いて行った。GPU 上でのパッキングを利用することで GPU - CPU 間の通信量を最小限に抑え、粒子の再整列を行うことで、粒子数の移動により生じるメモリアクセスの増加を効果的に抑えた結果、Weak Scaling、Strong Scaling 共に良いスケーリングを示し、Weak Scaling では、効率 97% 以上を達成した。Strong Scaling では、計算するステップ数の増加とともに指数的に増加する計算時間を、線形変化に抑え、1500 ステップの測定では、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。ソートの回数と計算時間の関係のモデル化を行い、ソート回数の最適値の理論式を得た。負荷バランスの悪い計算に対しても、動的に負荷分散を行うことで、固定領域分割法により計算する場合の約 2 倍性能を向上させた。

Vol.2012-HPC-133 No.37
2012/3/27

謝辞 本研究の一部は、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」から支援を頂いた。記して感謝を表す。

参 考 文 献

1) 杉原健太，青木尊之：マルチノード GPU クラスタによる高次精度移流スキームの演算性能，第 23 回数値流体力学シンポジウム講演予稿集 (2009).
2) CUDAProgrammingGuide3.2：http://developer.download.nvidia.com/compute/cuda/3_2/oolkit/docs/CUDA_C_Programming_Guide.pdf (2010).
3) Shimokawabe, T., Aoki, T., Ishida, J. Kawano, K., Muroi, C.：145 TFlops Performance on 3990 GPUs of TSUBAME 2.0 Supercomputer for an Operational Weather Prediction *Procedia Computer Science, Volume 4, Proceedings of the International Conference on Computational Science, ICCS 2011 , 2011, Pages 1535-1544.*(2011).
4) Shimokawabe, T., Aoki, T., Takaki, T., Yamanaka, A., Nukada, A., Endo, T., Maruyama, N., and Matsuoka, S.：Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer *Proceedings of the 2011 ACM/IEEE conference on Supercomputing (SC'11)*(2011).
5) 原田隆宏，越塚誠一，河口洋一郎：GPU を用いた SPH シュミレーション，計算工学会講演予稿論文集 Vol.12 (2007).
6) 都築怜理，青木尊之，王 嫻：複数 GPU を用いた大規模パッシブスカラー粒子計算の高速化，第 25 回数値流体力学シンポジウム講演予稿集 (2011).
7) Debprakash,P., and Sean,P., and Yong,C., and Naren,R.：Accelerator-Oriented Algorithm Transformation for Temporal Mining *Network and Parallel Computing, 2009. NPC '09. Sixth IFIP International Conference* (2009).