1ZC-06

マーチングキューブ法のマルチ GPU 上での階層的並列処理

Hierarchical Parallelization of Marching Cubes on Multi-GPU

吉川 一輝 †　　吉田 明正 †
Kazuki Yoshikawa　　Akimasa Yoshida

1　はじめに

3D コンピュータグラフィックスにおけるボクセルデータをポリゴンデータに変換するアルゴリズムとしてマーチングキューブ法 [1] がある．マーチングキューブ法を用いた関連研究の一例として，流体シミュレーション [2] や医療画像の 3D 可視化 [3]，複数の半透明の等値面のリアルタイム高速表示 [4] 等が挙げられる．

マーチングキューブ法において，ポリゴンを貼り付けるセルの抽出とそのセル数の計算部分に，CUDA Thrust ライブラリの exclusive_scan 関数を用いる実装が容易である．しかしながら，このライブラリ関数は 1GPU 実行を前提としており，マルチ GPU 環境では，全 GPU を有効利用することができなかった．そこで本研究では，exclusive_scan ライブラリ関数を利用しつつ，独自の CUDA コードを組み合わせて，マルチ GPU 環境に対応した exclusive_scan 関数を実装した．2 セットの NVIDIA Tesla K80（GK210 を 4 台）を用いて性能評価を行った結果，データ量の増加に応じて，提案手法の有効性が確認された．

2　マーチングキューブ法の適用

本章では，マーチングキューブ法を用いたポリゴンデータ生成方法の一例として，メタボール法 [5] による濃度場のデータであるボクセルデータをマーチングキューブ法に適用する例を述べる．

マーチングキューブ法では，まず各セル内の濃度場データからポリゴン面の有無を算出し，それを 1 と 0 で表しデータ配列に格納する．次に，その配列からポリゴン面を含むセルを収集してデータを集約し，ポリゴン化された際の頂点数を計算する．集約されたセルは対称性が考慮された 14 種類のポリゴンのパターン分類を行う．そのようにして得られた結果を OpenGL 等を用いて出力している．

2.1　マーチングキューブ法の処理手順

マーチングキューブ法のプログラムを 4.1 節の性能評価に用いるマシン上で実行し，GPU 上の配列に対しマーチングキューブ法を適用しカーネル実行のみを計測した場合の処理時間の内訳を図 1 に示す．実行時間としては内部ポリゴンへの貼り付け処理である (5) の launchGeneratePolys が最も大きいが，ここではそれに匹敵する exclusive_scan 処理 (図 1 の (2) と (4) の 2 回分) に着目する．(2) の exclusive_scan ではポリゴン面の貼り付けを判定したセルのスキャンを行い，ポリゴン面を有するセルの抽出及びそのセルの総和を求めている．また，(4) の exclusive_scan 処理ではポリゴンの頂点数

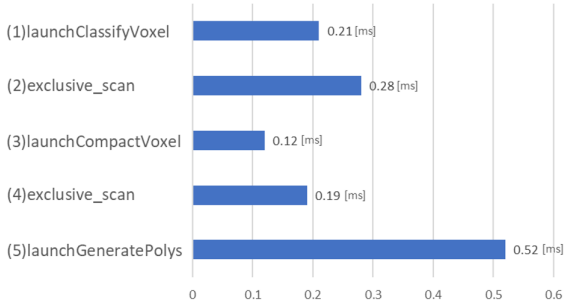

図 1　マーチングキューブ法の各処理時間．

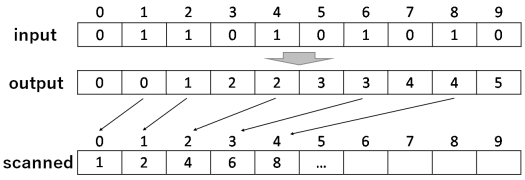

図 2　exclusive_scan とセルの集約．

の総和を求めるために用いられており，この処理の重要性がわかる．

2.2　exclusive_scan アルゴリズム

本節では，thrustScan 処理，即ち exclusive_scan の計算方法とその役割について述べる．exclusive_scan で行われる計算を式 (1) に示す．

$$output[i] = \sum_{j=0}^{i-1} input[j] \tag{1}$$

式 (1) を図 2 の input 配列に適用した結果は図 2 の output 配列として出力される．なお，input 配列において，1 はポリゴン面の元となる濃度面が存在しているセル，0 は存在していないセルを表す．出力された output 配列の末尾と input 配列の末尾を参照することでポリゴン面を有するセルの総和を求めることが可能となる．また，output 配列を参照することでポリゴン面を有するセルの抽出を行うことが可能になる．

3　マルチ GPU 上での exclusive_scan の CUDA 並列処理

本章では，exclusive_scan 処理をマルチ GPU 上で実現するために，CUDA による並列処理プログラムを作成した．関数内では OpenMP を用いて分割された対象データに対し exclusive_scan を行った後，各 GPU の総和データを用いて全体に対応した exclusive_scan を求める．以下の節では，具体的なアルゴリズム及び複数 GPU 上での実装について述べる．

†明治大学大学院 先端数理科学研究科 ネットワークデザイン専攻
Graduate School of Advanced Mathematical Sciences, Meiji University

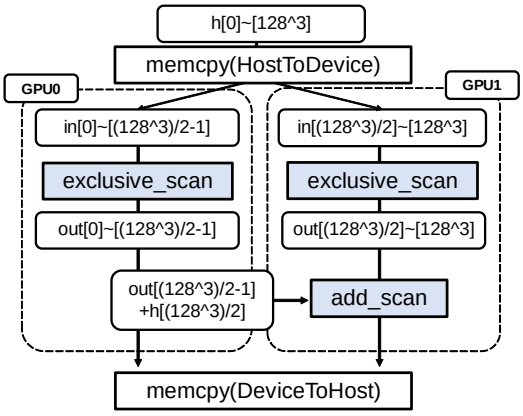

図 3　2GPU における exclusive_scan 処理.

3.1　exclusive_scan の複数 GPU 上での実装

実装する exclusive_scan 処理の処理手順を図 3 に示す．この処理では，まず各 GPU 対象領域の exclusive_scan 処理を行う．ここで，1GPU 内での exclusive_scan 処理には，CUDA Thrust ライブラリを利用する．exclusive_scan の計算では，output 配列にその配列番号よりも小さい番号の input 配列の合計が入力される．そのため，input 配列の末尾のデータは output 配列データに適用できない．そこで，入力データ配列の末尾と出力データ配列の末尾を各 GPU 内でコピーし，その和をホスト上のメモリ配列に格納しておく．

次に，各 GPU において，前述の処理でホスト上に格納した小計データを用いてカーネル関数を起動し，その小計を GPU 内で実行したデータの全ての要素に加算する．最後に出力された配列をホスト上にコピーすることで exclusive_scan の全体処理が完了する．

マーチングキューブ法における exclusive_scan の input 配列は N_x, N_y, N_z 空間を用いる場合，$N_x * N_y * N_z$ 個の要素を持つ一次元配列となる．GPU 間の処理を行う際，GPU の 1 ブロックの計算を 1024 要素とし，各スレッドで並列処理を行う．この時，GPU の各スレッドが配列 4 要素分の加算計算を行うことで，実行時のブロック数を削減し速度向上を図っている．

4　マルチ GPU 上での exclusive_scan の性能評価

本章では，マルチ GPU 上で行った exclusive_scan 処理の性能評価について述べる．

4.1　性能評価環境

性能評価に用いるマルチ GPU サーバーは，GPU：NVIDIA Tesla K80，CPU：Intel Xeon E5-2680 v3 2.5GHz 12core*2，メモリ：64GB，OS：CentOS6.9, CUDA の処理系：GCC4.4.7，CUDA Toolkit 9.1 となっている．

NVIDIA Tesla K80[6] は，Kepler アーキテクチャの GPU（GK210）を 2 基搭載した構成となっており，各 GPU は 2496CUDA コア（最大周波数 824MHz），12GB のデバイスメモリから構成されている．

本性能評価では，マーチングキューブに使われる exclusive_scan の 4GPU 実行の評価を取り扱う．加算処理の GPU 実行におけるスレッド数は 1024，ブロック数は配列の長さ 128^3=2048*1024 を考慮して 2048 となる．1 スレッドで 4 計算を行うため GPU2 台を用いる場合はブロック数は 256，GPU4 台を用いる場合はブロック数は 128 となる．データサイズを 10 倍に大きくした実行においては，加算処理において GPU2 台でブロック数 2560，GPU4 台で 1280 に設定される．

表 1　exclusive_scan の実行時間.

GPU 数	データ数=128^3	データ数=$128^3 * 10$
1GPU	3.44[ms](1.00)	34.24[ms](1.00)
2GPU	3.08[ms](0.90)	20.36[ms](0.59)
4GPU	3.50[ms](1.02)	12.09[ms](0.35)

4.2　GPU 上での exclusive_scan 処理の並列実行

本性能評価では，まず 128*128*128 の空間を想定し，exclusive_scan の input 配列の長さを 128^3=2,097,152 とした．評価では実装対象の exclusive_scan 関数を抽出し新たにマルチ GPU に対応した CUDA コードにより実行時間を測定する．ホスト上の要素数 128^3 の配列に unsigned int データの数値を格納した後，1GPU，2GPU，4GPU それぞれにおいて exclusive_scan を用いた計算を繰り返し行う．ホスト上に存在する配列データがホスト上の別の配列に格納されるまでの実行時間を測定した．

実行結果は表 1 に示す．データ数=128^3 配列の場合．1GPU における実行時間は 3.44[ms]，2GPU は 3.08[ms]，4GPU は 3.50[ms] であり，2GPU において 10%の実行時間の短縮となった．

次に，より大規模な配列の場合の評価として配列のデータサイズを 10 倍にしたところ，1GPU での計算結果が 34.24[ms] であったのに対し 4GPU では 12.09[ms] であり，65%の実行時間の短縮となった．これにより，提案手法による exclusive_scan 関数はマルチ GPU 上で高い実効性能を達成できることが確認された．

5　おわりに

本稿では，3DCG の生成に用いられるマーチングキューブ法において，exclusive_scan 処理のマルチ GPU 上での並列処理手法を提案した．本手法では，部分的に CUDA Thrust ライブラリを利用しつつ，独自の加算処理を組み合わせることにより，マルチ GPU 上での exclusive_scan 処理を実現した．NVIDIA Tesla K80 上で行った性能評価の結果から，exclusive_scan の性能評価において 1GPU 比で 65%の実行時間短縮が得られており，提案手法の有効性が確認された．

参考文献

[1] William E. Lorensen,Harvey E. Cline. Marching cubes:A high resolution 3D surface construction algorithm. ACM SIGGRAPH,1987.
[2] 小原俊介, 豊谷純, 角田和彦, 古市昌一粒子法による流体シミュレーションと物理ベース CG ISSN 2186-5647,2011.
[3] Manuel Beniani,Mariagiovanna Sami,Danilo Pietro Pau. MRI Parallel Processing for Embedded Visualization. ICCC-Berlin,2013.
[4] Y.M.Xie,G.Y.Wang,T.T.Wong,P.A.Heng. Parallel visualization of multiple translucent isosurfaces. APCCAS,2008.
[5] 乾 正知. GPU 並列図形処理入門. 技術評論社,2014.
[6] NVIDIA. TESLA K80 GPU ACCELERATOR, https://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf,2015.
[7] Mark Harris. Parallel Prefix Sum (Scan) with CUDA NVIDIA,https://www.mimuw.edu.pl/~ps209291/kgkp/slides/scan.pdf,2007.