TSUBAME 共同利用　令和２年度　学術利用　成果報告書

利用課題名　LRnLA アルゴリズムを用いた物理シミュレーション
英文：Simulation of Physical Processes with LRnLA Algorithms

善甫 康成
Yasunari Zempo

法政大学 情報科学部
Computer and Information Sciences
http://cis.k.hosei.ac.jp/

メモリバウンドの問題に関して、局所的にステンシル計算が可能な場合、GPU は CPU 処理と比較してシミュレーション速度を向上させる。 ただし、データサイズが GPU デバイスメモリに適合しない場合、処理を CPU-GPU 間でデータの交換を行うと、大幅なオーバーヘッドが生じパフォーマンスが低下する可能性がある。temporal blocking を使用すると、データ交換の影響を隠すことができる。 我々は wavefront 型 temporal blocking アルゴリズムでのデータ同期用に設計された新しいデータ構造を開発中である。この新しいデータ構造作成に向け、格子ボルツマン法の計算のための LRnLA アルゴリズム ConeFold についてテストしているところである。

For memory-bound problems with local stencil, the GPU increases the simulation speed compared to CPU processing. But if the data size does not fit the GPU device memory the CPU-GPU exchange can become a significant overhead, and decrease the performance. With temporal blocking, the exchange can be concealed. We develop a new data structure that is designed for the data synchronization in the wavefront-type temporal blocking algorithms. The data structure is tested for LRnLA algorithms ConeFold for Lattice Boltzmann Method computations.

Keywords: LRnLA algorithms, temporal blocking, data structure

背景と目的

物理問題の数値シミュレーション、特に波動現象や流体力学の分野では、矩形メッシュ上のステンシルスキームが非常によく使われている。しかし、シミュレーションのコードは、しばしば低い性能効率を持っている。

そこで、一般的なネストされたループではなく、wavefront 型のアルゴリズムを用いることで、データの再利用による効率化を図っている。Wavefront 型アルゴリズムを用いると計算ウィンドウも可能になる [1] 。

LRnLA (Locally Recursive non-Locally Asynchronous) アルゴリズム [2,3] の中には、wavefront に似ているアルゴリズムもある。3D では、LRnLA アルゴリズムは、時間と空間の 3D1T タイリングを実現している。dD1T の ConeTorre という LRnLA アルゴリズムは、1D1T の wavefront 型のアルゴリズムの直積として表せる [1]。DiamondTorre LRnLA アルゴリズムも wavefront 型であり、十字型のステンシルに対してより良い局所性がある。

多次元 temporal blocking のもう一つの問題は、データアクセスの局所性、ベクトル化のためのデータ・アライメント、並列アクセスのための coalescing の原則を満たすデータレイアウトを探すことである。

そのため、コンパイラの自動的なループの最適化では、データレイアウトの制約を超えること不可能である。従って、手動によりデータレイアウト扱う方法が必要である。

複雑なデータアクセスに関する問題は、多くの LRnLA コードにも存在していた。データストレージのインデックス計算の複雑性が上がると、その影響がデータアクセスの coalescing が不十分になることなどに現れる。

本プロジェクトでは、波面型 LRnLA アルゴリズムにおけるデータ格納問題の解決策を開発することである。これにより、CPU の RAM を主なデータ保存場所として使用し、GPU での計算を行えば、CPU と GPU の間のデータ交換を行っても、計算性能影響を与えない。場合によっては SSD ストレージまで同じように使用できる。

本プロジェクトでは、wavefront 型および、その他の temporal blocking に使用できる方法を研究しています。さらに、GPU global メモリと GPU shared メモリで効率的なデータ交換する方法を探すものである。

概要

時空間の依存グラフには、(x, y, z, t)の点が、この

座標に対応するフィールド値を得るための演算を示す。LRnLA アルゴリズムは、その空間におけるポリトープとそのポリトープの細分で表せる。

すべてのメッシュ値を時刻 N_T までに更新するタスクは、依存グラフ全体を覆うボックスで表現される。ネストされたループでは、同じ t 座標を持つすべての点を覆うボックス（サブタスク）に細分される。そのあと、並列実行のためにさらにサブタスクに分解されることもある。再帰的な細分化は、メッシュノードの更新で終了となる。LRnLA アルゴリズムと temporal blocking アルゴリズムには、そのかわりに、t 軸に伸びるパイラミッドかプリズムなどの形に細分がおこなわれる。

依存関係グラフのdD座標空間への投影はデータ空間であり、ポリトープの投影はその実行時に保存しなければならないデータを表す。ステンシルコードでは、ロードしなければならないデータは、その投影の近辺にある。

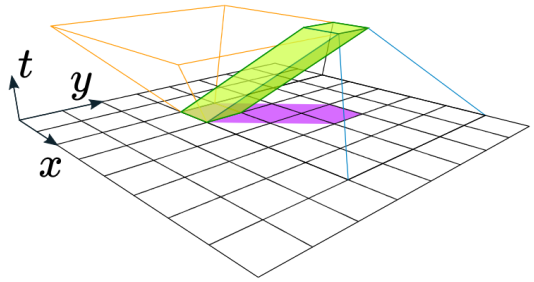

Fig. 1. 同期平面上の 2 つのキューブの影響領域と依存領域の交点としての ConeTorre と、その投影でのセルに格納されているデータ(ピンク)。2D1T で二次元のシミュレーションをあらわすが、実際のコードは 3D1T となっている。

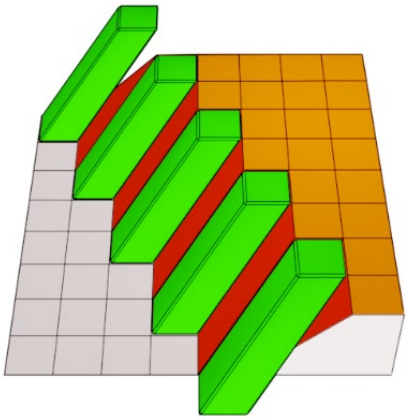

Fig. 2. 非同期 ConeTorre の wavefront。オレンジ色のタイル上では、$t = N_T$；赤色のスロープ上では、$0 < t < N_T$ 。

ConeTorre という LRnLA アルゴリズム(図１、２）は、Wavefront 型アルゴリズムの一般化である。その形は、2 つのハイパーキューブの間にある 3D1T プリズムと言える。この構造は、多次元で可能であり、座標軸で分割することができる。

データレイアウト

立方体のシミュレーション領域を例にとって説明する。図３は、ConeTorre をデータ空間に投影した時の様子を表している。

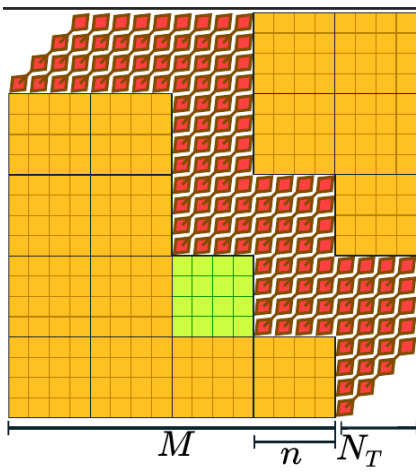

Fig. 3. データレイアウト。図中で一番小さい四角形は ConeTorre のベースの$8 \times 8 \times 8$ セルと考える。緑色：Tile、黄色：BaseTile、赤：FArSh。

基本的な ConeTorre 計算では、$t = 0$ の$8 \times 8 \times 8$ セルの立方体が GPU レジスタ（またはターゲットとなるコンピュータシステムの上位メモリレベル）にロードされ、更新される。その後、その立方体の左側のセルがメインメモリストレージに保存され、立方体の右側のセルがそこからロードされる。この後、ConeTorre での計算はさらに１ステップ進む。

主なデータ保存場所では、データは AoS（array of structure）方式で整理されている。配列要素には、ユニットスキームの更新で更新されるデータが含まれている。配列は、データアクセスの局所性のためにタイルで構成されており、その極端な例が Z カーブ配列である。これは自然なデータの保存方法であり、データの初期化や視覚化のための出力、その他の診断に便利である。

しかし、ConeTorre（図１）でアクセスされるデータ配列の一部の形状は、このようなパターンに適合していない。配列のインデックス計算には、通常、整数演算のオーバーヘッドが存在する。計算したデータを保存する際には，正方形に準拠したデータ構造の中で，細長い六角形をループさせる必要がある(図１，ピンク）。そして、次の ConeTorre で読み込まれる時に、再び六角形が横断されます。

本プロジェクトでは、wavefront 型アルゴリズムにおけるデータ交換に最適な新しい FArSh (Functionally Arranged Shadow)というデータ構造を提案する。

ベースとなる立方体の右側にある点(x_0, y_0, z_0, t_0)を考えてみる。点$(x_{0+i}, y_{0+i}, z_{0+i}, t_{0+i})$, $i = 0,..,N_T\text{-}1$ の線は、FArSh の要素である。下の ConeTorre では i の大きい順に書かれ、上の ConeTorre ではその順に読まれる。このように、共通の面を持つ

ConeTorre 間のデータは、このようなセルの並びによって交換することができる。FArSh のデータは、サブタスク間でデータ交換がおける所でのみ必要となる。

ConeTorre には、GPU で保存されている Tile という配列からベースキューブのデータと、FArSh からスロープのデータを読み込む。実行後、上側の立方体のデータを BaseTile に、上側の斜面のデータを FArSh に書き込み、以前に保存したデータを置き換える。

この手法は、計算流体力学のステンシル計算方式である Lattice Boltzmann Method (LBM)の実装でテストを行っている。GPU のグローバルメモリに格納されている主なデータは Tile 配列で、LBM のデータが格納されている。FArSh は、セルのラインの配列として GPU グローバルメモリに格納されている。

ConeTorre は、8×8×8 の CUDA スレッドのブロック上で起動して計算が実装されている。まず、ベースとなるセルをメモリ上の Tile からスレッドレジスタにロードし、CUDA スレッドごとに 1 セルずつロードする。BaseTile データ構造は、CPU RAM で初期化されます。BaseTile は 3 次元配列であり、BaseTile の配列要素は Tile である。GPU は Tile を CPU RAM から 1 つずつロードし、ConeFold で実行して、結果の Tile を CPU RAM にセーブする。FArSh は GPU に格納されているので、前に計算されたものから次の ConeFold に必要なデータの送信を行う。

まとめと今後の課題

現在の高性能アルゴリズムに関して、理論的に理想的な形は次のようになる。計算を局所化するために最も高いメモリレベルを使用し、プロセス間のデータ交換にはより小さなメモリレベルを使用し、ストレージには最も大きなメモリ容量を持つメモリレベルを使用することである。またシステム上でのデータの移動は、計算と同時に行うことである。

FArSh では、データ交換をより細かく制御し、この目標に限りなく近づけることができる。つまり、シミュレーションデータは CPU RAM にあるが、GPU 内のサブタスク間のデータ交換に必要なデータは GPU メモリから出ることはない。

参考文献

[1] Wolfe, Michael. "Loops skewing: The wavefront method revisited." International Journal of Parallel Programming **15**(4), 279-293 (1986)

[2] Levchenko, V.D., Perepelkina, A.Y. Locally Recursive Non-Locally Asynchronous Algorithms for Stencil Computation. Lobachevskii J Math **39**, 552–561 (2018). https://doi.org/10.1134/S1995080218040108

[3] Zakirov, Andrey, Vadim Levchenko, Anastasia Perepelkina, and Yasunari Zempo. "High performance FDTD algorithm for GPGPU supercomputers." In Journal of Physics: Conference Series, **759**(1), 012100. (2016)

[4] Endo, Toshio, "Applying Recursive Temporal Blocking for Stencil Computations to Deeper Memory Hierarchy." 2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA). IEEE, 2018.