TSUBAME 共同利用　令和 4 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 LR n LA Algorithm

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/

邦文抄録（300 字程度）
計算の中で CPU と GPU を同時に使い並列処理を行うヘテロな計算を行えば、利用可能なすべての処理能力を活用することができる．もちろん CPU のピーク性能とメモリ帯域幅が GPU と比べ 20 倍程度の低いので、シミュレーションに CPU と GPU の両方を同時に使用することは非効率的な場合が多い．そこで CPU の並列化効率を向上させ、GPU と同等の性能を発揮できる新しい時間ブロックアルゴリズムを考案した．空間と時間にまたがるサブタスク間でデータを交換するようにデータ構造を工夫し、利用することとした．我々がこれまで開発してきた LRnLA 型の 1 つの有用なアルゴリズムとなる．

英文抄録（100 words 程度）
A heterogeneous computation can engage both CPU and GPU parallelism at the same time, and take advantage of all available processing power. But CPU peak performance and memory bandwidth are at least 20 times lower than that of GPU, so simultaneous use of both CPU and GPU for simulation may be inefficient. We propose a new temporal blocking algorithm of the LRnLA family that can increase the efficiency of CPU parallelization so that the performance is on par with GPU. The algorithm uses the data structure designed specifically to exchange data between subtasks spanning space and time.

Keywords: LRnLA ・temporal blocking・LBM・hwloc・many-core

背景と目的

GPU を搭載したマシンにおいて、GPU ばかりでなく CPU の計算処理能力も利用したヘテロな計算が行えれば、すべての計算処理能力を使うことができる．しかし奈良が、GPU と CPU のピーク性能やメモリーバンド幅には大きな差がある．大規模な数値計算を GPU を用いて計算を行うときでも、CPU も当然利用している．この CPU をどれほど効率的に行うのが、今回の研究テーマであった．

前回までの報告で、CPU 制御下の RAM に保存されているデータを GPU で処理しても計算速度が落ちない計算手法の開発の報告を行った[1,2]. その手法を使うと、GPU グロバールメモリに入れるほどの計算ばかりではなく、CPU 制御下の RAM や SSD に入れなければならないほどの数 TB のサイズの計算までできるようになった．さらに計算性能を向上させるてめには、CPU の処理能力まで利用するが望ましい．さて TSUBAME3.0 で設置されている CPU の性能は、GPU に比べるとかなり低く、その性能は約 20 分の 1 程度である．この環境で CPU 性能を利用し、CPU でも GPU 並みのパフォーマンスで同じ計算できるような新しいアルゴリズムを開発することが目的である．

概要

Finite Difference と Lattice Boltzmann などの数値計算スキームを CPU で効率的に実行するために、実装したアルゴリズムは ConeTorre (CT)という LRnLA (Locally Recursive non Locally Asynchronous) である[1, 3]．並列ではない計算でも、メモリバンド幅が性能ボトルネックになる場合、CT は L1, L2, L3 キャッシュでデータを局在化させることができ、データアクセス時間を減少させることができるという特徴を持っている．CT はNとN_Tという引数がある関数かサブルーチンとして理解するとわかりやすい．図 1 に示したように、時空間次元Dの計算で、$C(N)$を計算領域とする．つまり各辺にN個のセルを持つ立方体とする．ConeTorre $CT(N, N_T)$ という関数で、$C(N)$の中のスキーム計算実行していく．時間軸が1からN_Tまで分割されている．各軸で 1 セルずつずれてN_T回繰り返し、$C(N)$の計算を実行する．

さらに$CT(N, N_T)$は、$CT(n, n_T)$ $(n < N, n_T < N_T)$というサブタスクに分解できる(図 1)．$CT(N, N_T)$で計算すべき時空間を覆い、全計算を実行する．NとN_Tというパレメターがあるため、$CT(N, N_T)$というサブタスクのデータ量を調整できる．分解方法を上

手く選択すれば、L2 キャッシュに入れる $CT(nL_2, n_T L_2)$が得られる．これをさらに分解すれば L1 に入れるタスクが得られる．これによりキャッシュでのデータの局在性を十分活用することができ、計算のパーフォマンスを向上させることができる．

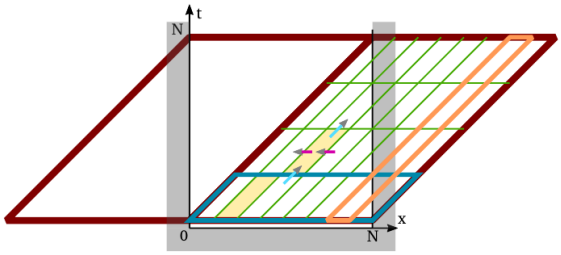

図 1. $CT(N,N)$の数値計算領域と更に分解したCTの様子．矢印はデータ依存を示す．

現在の CPU ではマルチコアが一般的である．従って、マルチコアによる並列処理を行う場合、コアあたりの L2, L3 キャッシュ量が減少するため、データの局在化によるパーフォマンス向上が期待できない．そこで、この課題を解決するため、新しい FArShFold という並列法を開発した．

FArShFold というアルゴリズムを導入するため、FArSh (Functionally Arranged Shadow) というデータ配置が必須である[4]． FArSh データは、隣にある CT の間の交換のために考案されたデータ配列である．CT 計算を行うための元の計算領域の$C(N)$のデータは、主メモリストレージ（RAM か SSD)からロードし、処理後同じのメモリストレージに保存される．CT 計算が行われる時空間での斜めのスロープのデータは、FArSh データ配列からロードし CT 計算の後、同じ FArSh データ配列に上書きされる (図 2).

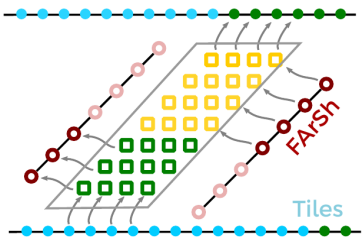

図 2. FArSh データ配列の使い方

マルチコアの並列化で L2, L3 上のデータ局在性を保つため、次のような並列化法が考えられる．まず、$CT(N,N)$で計算領域を確保する(図 1)．次に、$CT(N,N)$を$CT(N,n')$に分解する$(n' < N)$．$CT(N,n')$をさらに$CT(N,n_s)$ に分解する $(n_s < n')$ (図 3)．ここで$CT(N,n_s)$それぞれ処理はスレッド並列により実施する．その結果、空間並列化ではなく、時間軸で並列化を行うことが可能になる．その時 FArSh データ配列はスレッド数に合わせて分解し、1 コアの L2 キャッシュ分を各コアに割り付ける．$CT(N,n_s)$を更に$CT(2,n_s)$に分解する．これに対応する元の計算領域（$C(2)$という立方体）にあるデータを L1 キャッシュに入れるという方法である．

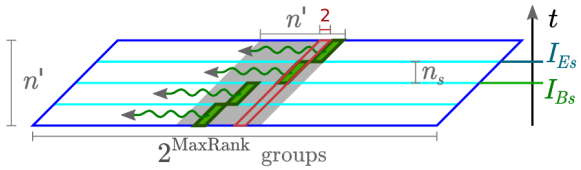

図 3: FArShFold の並列化

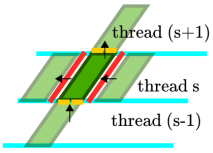

図 4. FArShFold の並列化とデータ依存の様子

その結果、CTの間のデータ交換では、CTスロープのデータは各コアの L2 および L3 キャッシュに残る．コア間の交換には、$C(2)$ のデータしか送らないことになるので、並列化の効率は飛躍的に高まる．

TSUBAME3.0 ノードの場合は Intel Xeon E5-2680 V4 が 2 台ある． 各 CPU の L2 キャッシュは 256KB であって、L3 は L2 より 10 倍大きい．コア数は 14 で$(5+5+4)$の 3 コアクラスター構成である．これらのコアは L3 キャッシュを共有している．そこで$n' = 3n_s$として$CT(N,n_s)$を5(4)スレッドで処理すれば、その5(4)スレッドが L3 キャッシュに局在化することになるので、FArSh を共有できるはずである(図 4).

結果および考察

考案した FArShFold アルゴリズムを実際の FDTD コードに実装した．Hwloc ライブラリを使用してコアあたりの処理を分ける.AVX ベクトル演算を使用も実装した．数値スキームとして、LBM を選択した．LBM で、セルあたりのデータとセル更新あたりの浮動小数点演算（FLOP)を詳しく調べることができる．具体的には LBM 法のステンシル（Q)と衝突演算子を変更する．それによりセルデータ量によってパーフォマンスの変化を調べることができる．

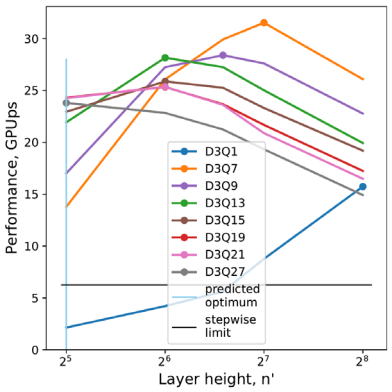

図 5. パフォーマンスベンチマーク

基本的なベンチマークは Ryzen R9 5950X で行った(図 5)．測定単位は Pups (LBM population update per second)を用いた．これは 1 秒あたりのセル更新数 (LUps - Lattice cell Update per second) にQを掛けたものである．

時空間ブロッキングを使用していない LRnLA の場合、パフォーマンスピークは~6 Pups であるが、CT を用いた場合、その 5 倍約 30Pups となりかなりの性能向上が期待できる(図 5)．またこのベンチマークでn'の最適値を得ることもできる．最適パーフォマンスは 1 GLUps を超えることも分かった．

まとめ、今後の課題

新しいアルゴリズム ConeTorre を考案した．このアルゴリズムを FDTD コードに実装して、1 個のマルチコアの CPU において GPU と同等のパーフォマンスを得た．これはマルチコアの並列化において各コアに対応する L2, L3 にデータを局在化させることができたためである．

今回は基礎検討の段階であるが、GPU でも計算処理を実行する場合、同じ CT の時空間データ領域を分解できる．今回考案したアルゴリズムを用いれば GPU と CPU 組み合わせたヘテロ計算を実行できるはずであり、GPU だけの計算のパーフォマンスより更に向上させるものと期待できる．

参考文献

[1] Zakirov, A, V Levchenko, A Perepelkina, and Yasunari Zempo. "High performance FDTD algorithm for GPGPU supercomputers." In Journal of Physics: Conference Series, vol. 759, no. 1, p. 012100. IOP Publishing, 2016.

[2] Shimokawabe, T., Endo, T., Onodera, N., Aoki, T.: A stencil framework to realize large-scale computations beyond device memory capacity on GPU supercomputers. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 525–529 (2017). IEEE

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

[4] Perepelkina, A., Levchenko, V.D. (2021). Functionally Arranged Data for Algorithms with Space-Time Wavefront. In: Parallel Computational Technologies. PCT 2021. CCIS, vol 1437.