ポスター概要

Parallel in-place radix sort の評価

尾城 拓真[*1]　宮崎 崇史[*2]　清水 伸幸[*3]　川島 英之[*4]

Abstract – **Parallel radix sort は多くの先行研究事例があるが、その事例の多くは入力データサイズと同等かそれ以上のメモリを消費してしまう。そのため、高いパフォーマンスを得ることのできる Parallel in-palce radix sort はあまり知られていない。本研究では最新の parallel in-place radix sort である PARADIS('15) と Regions sort('19) を取り上げ実装および性能の比較をした。**

Keywords : parallel sorting, in-place algorithm

1 はじめに

配列の要素を key の順序にしたがって並び替える sorting algorithm は多くのアプリケーションで使用される基本的な処理であるため、多くの研究が行われてきた。現在、多くの種類の sorting algorithm が利用されている。例えば C++の標準ライブラリで使用されている sorting algorithm はイントロソートであり、一方で Python(2.3 以降) や Java SE 7 の標準の sorting algorithm は Tim Sort である。sorting algorithm はそれぞれに特定の場面でより優れているといった特徴やヒューリスティックスが組み込まれていることもあり、あらゆる場面において優れていると断言できるものを考案することは難しいといえる。そのため、最新の sorting algorithm として発表され、発表された性能は高いものであったとしても特定のケースに限るのではないかと考えるのは自然である。そこで本ポスターでは使用するメモリが少なくて済む in-place な algorithm であり、かつ高速に動作する parallel radix sort の近年の研究に注目し、その性能の評価を第三者の立場で行う。

2 Parallel Radix Sorting Algorithms

PARADIS[1] は 2015 年に parallel in-place radix sorting algorithm として発表された sorting algorithm である。特徴はこれまでの parallel radix sort と異なり、入力のデータ数に比例したサイズの追加の補助配列を必要としない点にある。

図 1 は PARADIS の Permutate フェーズと Repair フェーズにおいてどのような操作が配列に行われているかを示した図である。まず、ヒストグラムを求めそれによって図中の黒矢印のさす場所が決まる。これは

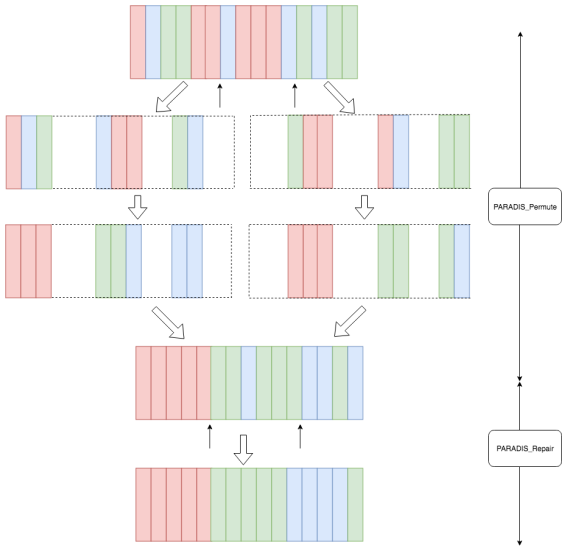

図 1　PARADIS で sort が行われる様子

各値が入るべき場所の境目を示している。そして、それに基づき配列の領域を分割し、それぞれに対して in-place に並び替え操作を並列に行う。その後 Repair フェーズにはいり、未ソート部分があれば未 sort 部分に対してそれぞれの入るべき場所の端に寄せ、再び Permutate フェーズを行う。

Regions sort[2] は PARADIS と同じ parallel in-place sorting algorithm であり、2019 年に発表された。PARADIS の並列化が配列の領域を分割して行うのに対して Regions sort の並列化は regions graph という、入れ替えるべき要素を表現するグラフを用いて行われる。これにより、PARADIS のように sort しきれなかった要素を集めて再び sorting を行う必要がない。図 2 は Regions sort において regions graph を用いて sort をする様子を図示したものである。Regions sort では各要素が入るべき領域を region と呼び、それぞれの region から正しい region へ幾つの要素を動かさな

[*1]慶應義塾大学
[*2]ヤフー株式会社
[*3]ヤフー株式会社
[*4]慶應義塾大学

ければいけないのかをコスト付き有効辺で表現する。グラフから全ての辺が取り除かれた状態が sort された状態となる。並列化はこのグラフによりおこなわれ、グラフの辺の削除方法は cycle 法と 2-path 法がある。cycle 法は図 2 における青辺のようにグラフ中で cycle になっている辺は互いに入れ替えればよいというものである。2-path 法は図 2 における赤線と緑線でおこなわれており、それぞれ 2 本の実線を消去することで新しく 1 本の辺ができている。

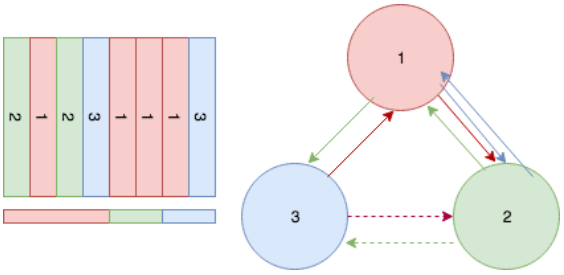

図 2　regions graph による sort の様子

二つの sorintg algorithm と in-place ではない parallel radix sort との比較を図示したのが以下である (図 3.)

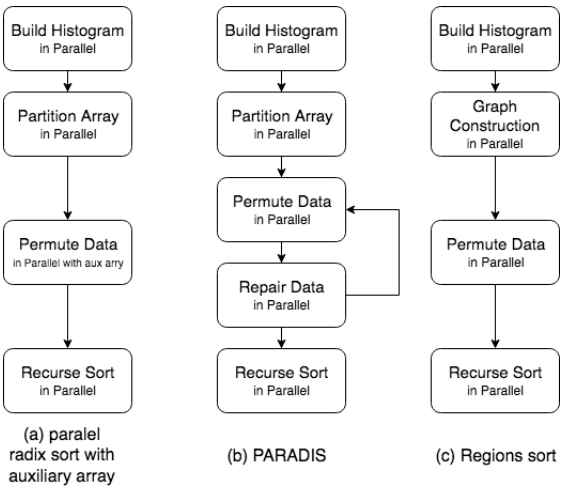

図 3　各 parallel radix sort 概要

(a) は補助配列を用いた parallel radix sort であり、(b)、(c) に比べ多くのメモリを消費する。(b) は PARADIS であり、Permutate Data と Repair Data において現在の桁の全ての要素が sort 済みになるまで同様の操作を繰り返すのが特徴的である。一方で (c) はグラフを構築する手間があるがそうしたことは起きない。

3　性能評価

上記で紹介した sorting algorithm(図 3 参照) に対して性能評価を行った。

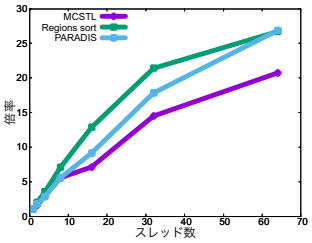

図 4　unif を sort

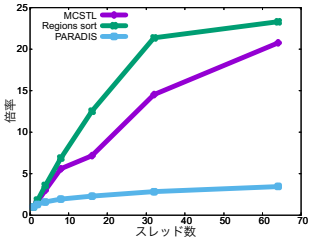

図 5　zipf(0.75) を sort

実験環境は CPU @ 2.10GHz、論理コア数 224 の環境で行い、コンパイラは g++ 7.4.0、最適化オプションは O3 を使用した。使用するデータ分布は一様分布については C++の標準ライブラリ内の uniform distribution。skew のある分布として zipfian distribution[3] を使用した。ベンチマークとしては MCSTL[4] を使用した。これは現在 libstdc++ parallel mode に統合されている parallel merge sort である。

図 4 は 10^9 件の一様分布から生成された int 型の乱数に対してどれくらいの倍率でスケールするかを示している。図 5 は 10^9 件の zipfian 分布から生成された int 型の乱数に対して同様な実験をおこなったものである。

4　おわりに

最新の sorting algorithm である PARADIS と Regions sort を比較することによって PARADIS の論文内では解決できるとされていたデータの分布が偏った場合の性能について Regions sort の優位性が確認できた。ただし、PARADIS の論文内ではロードバランスのヒューリスティクスが含まれているとされているためさらに改善される可能性はある。今後はこの実験結果を発展させ、より優れた sorting algorithm を検討していきたい。

参考文献

[1] Minsik Cho, Daniel Brand, Rajesh Bordawekar, Ulrich Finkler, Vincent Kulandaisamy, and Ruchir Puri. PARADIS: an efficient parallel algorithm for in-place radix sort. Proceedings of the VLDB Endowment , 8(12):15181529, 2015.

[2] Omar Obeya, Endrias Kahssay, Edward Fan, and Julian Shun. Theoretically-efficient and practical parallel in-place radix sorting. In Annual ACM Symposium on Parallelism in Algorithms and Architectures , pages 213224, 2019.

[3] Jim Gray, Prakash Sundaresan, Susanne Englert, Ken Baclawski, and Peter J. Weinberger. 1994. Quickly Generating Billion-record Synthetic Databases. In ACMSIGMOD International Conference on Management ofData. 243252.

[4] Johannes Singler, Peter Sanders, and Felix Putze. 2007. MCSTL: The Multi-core Standard Template Library. In Euro-Par. 682694.