

データの分布に着目した並列ソーティングアルゴリズムの性能評価

山 岸 秀 規[†] 黒 田 久 泰^{††} 金 田 康 正^{††}

並列ソートを実行する際に、PE(Processing Element)間に負荷の偏りが生じると、最も多くのデータを担当するPEにおける処理がボトルネックとなり、性能を低下させる原因となる。本研究では並列計算機上でよく用いられ汎用性の高いことで知られるレギュラーサンプリングソートアルゴリズムと相性のよい、データの分布に着目した負荷分散手法を提案する。提案する手法は、通常のレギュラーサンプリングソートアルゴリズムにおいてPE間の負荷分散がうまくいかない場合に機能し、より均等な負荷バランスを与えることによって並列ソートアルゴリズム全体としての潜在的な性能劣化の危険性を軽減する。日立の分散メモリ型並列計算機SR2201上において、提案する手法を適用したレギュラーサンプリングソートアルゴリズムと、この手法を適用しない単純なレギュラーサンプリングソートとの性能比較を行った。

Performance evaluation of parallel sorting algorithm based on data distribution

HIDENORI YAMAGISHI,[†] HISAYASU KURODA^{††}
and YASUMASA KANADA^{††}

The largest number of the elements which is processed by a certain PE increases Unbalanced load among PE(Processing Element)s and leads to poor performance of parallel sorting. We propose a load balancing method based on data distribution. This method suites regular sampling sort algorithm, which is versatile and widely used in parallel machines. Our method works and brings more equal load balance when the plain regular sampling sort algorithm failed load balancing among PEs. This reduces potential hazard of losing performance. We also compare the performances of the plain regular sampling sort algorithm and one which is applied our load balancing method on Hitachi parallel machine SR2201.

1. はじめに

順序が定義された集合の整列、すなわちソートは非常に基本的な処理であり、データベース処理をはじめとして適用される範囲も広い。また近年ではデータベースの巨大化などを背景に、並列計算機を用いて大規模なデータを高速にソートする必要が生じる場合も少なくない。このため既に多くの並列ソートアルゴリズムが研究されているが¹⁾²⁾³⁾⁵⁾⁶⁾⁷⁾、いかなるアルゴリズムにおいてもPE間の負荷分散が重要な問題となる。PE間に負荷の偏りがあると、最も多くのデータを担当するPEにおける処理がボトルネックとなり、性能が損なわれる。

しかしながら、データの分布形状を予測し、その形状に応じて適当なしきい値を選ぶことができれば、均

等な負荷分散を行うことによって潜在的な性能劣化の危険性を軽減することができる。本論文では、並列計算機上でよく用いられ汎用性の高いことで知られるレギュラーサンプリングソートアルゴリズム¹⁾²⁾において、プロセッサ間の負荷バランスを向上させることにより、アルゴリズム全体としての性能を向上させる手法について述べる。

日立の分散メモリ型並列計算機SR2201上において、提案する手法を適用したレギュラーサンプリングソートアルゴリズムと、本手法を適用しないレギュラーサンプリングソートアルゴリズムとの性能比較を行った。

まず提案する手法を適用する対象として採用したレギュラーサンプリングソートアルゴリズムと、性能比較の対象に用いたランダムサンプリングソートアルゴリズムの概要および提案する負荷分散手法を第2章で説明し、第3章においてその性能評価を行う。第4章で考察を述べ、最後に第5章でまとめとする。

[†] 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Graduate School
of Science, the University of Tokyo

^{††} 東京大学情報基盤センター
Information Technology Center, the University of
Tokyo

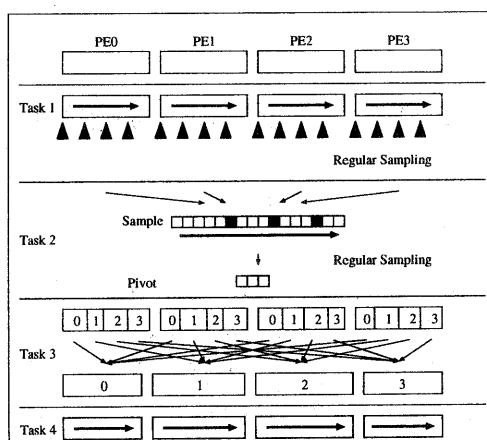


図1 レギュラーサンプリングソートアルゴリズム ($p=4$ の場合)

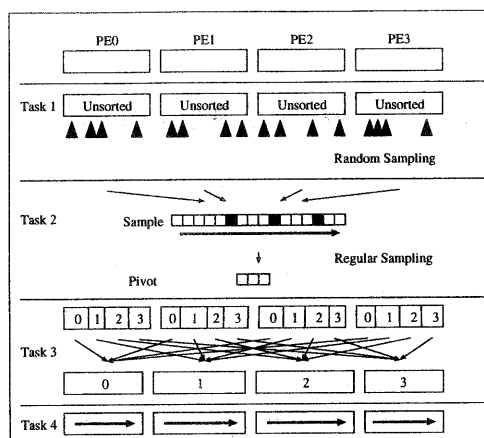


図2 ランダムサンプリングソートアルゴリズム ($p=4$ の場合)

2. データ分布に着目した並列ソートアルゴリズム

2.1 レギュラーサンプリングソートアルゴリズム

提案する手法を適用する対象として採用したレギュラーサンプリングソートアルゴリズム¹⁾¹⁾の概要を図1に示す。

以下の説明では p 台の PE で N 個の要素を扱う場合を考える。簡単のために $\omega = N/p^2$, $\rho = p/2$ とし、また N は p で割り切れるものとする。

- (1) 各 PE は、 N/p 個の要素を受け取り、各 PE 内で逐次ソートを実行する。逐次ソートの実行後、等間隔に $1, (1+\omega), (1+2\omega), \dots, (1+(p-1)\omega)$ 番目の要素をサンプルとして抽出する。
- (2) 各 PE から抽出されたサンプルを全 PE に Broadcast してひとまとまりの集合とし、各 PE 内で逐次ソートを行う。ソート済のサンプルから等間隔に $(p+\rho), (2p+\rho), \dots, ((p-1)p+\rho)$ 番目の要素を軸として抽出し、これを全 PE に Broadcast する。
- (3) 各 PE で軸をもとに要素列を p 個の部分列に分割する。より小さな要素を含む部分列から順に $0, 1, \dots, (p-1)$ の番号を割り当て、各々の部分列が同じ番号の PE へ送られるように要素を交換する。
- (4) 各 PE 内で受け取った部分列に対して逐次ソートを実行する。

Task 3 が完了した時点で、PE m (ただし $1 \leq m \leq (p-2)$) 内の任意の要素は PE $(m-1)$ 内の任意の要素よりも大きく、かつ PE $(m+1)$ 内の任意の要素よりも小さくなっているため、Task 4 の処理が完了すれば PE 0 から PE $(p-1)$ までの要素の列をつなげた全体が小さい順に並べられていることになる。

また、Task 1 および Task 4 では理論上任意の逐次ソートアルゴリズムを用いることができるが、第3章の性能評価においては Task 1 ではクイックソート、Task 4 ではマージソートを実行している。クイックソートは汎用の古典的な逐次ソートアルゴリズムとしては最も高速なアルゴリズムとして知られており、 N 個の要素のソートを平均 $O(N \log N)$ ステップで完了するが、最悪の場合には $O(N^2)$ ステップを必要とする。マージソートは既にソート済みの部分列を併せてソートする場合に効果的なアルゴリズムであり、最悪でも N 個の要素を $O(N \log N)$ ステップで完了する。これらの逐次アルゴリズムの詳細については³⁾などを参照されたい。

2.2 ランダムサンプリングによる並列ソートアルゴリズム

ランダムサンプリングによる並列ソートアルゴリズム⁷⁾の概要を図2に示す。以下の説明では p 台の PE で N 個の要素を扱う場合を考える。簡単のために $\omega = N/p^2$, $\rho = p/2$ とし、またレギュラーサンプリングの場合と同様に N は p で割り切れるものとする。

- (1) 各 PE は、 N/p 個の要素を受け取り、無作為にサンプリングを行う。
- (2) 各 PE で抽出されたサンプルを全 PE に Broadcast してひとまとまりの集合とし、各 PE 内で逐次ソートを行う。ソート済のサンプルから等間隔に $(p+\rho), (2p+\rho), \dots, ((p-1)p+\rho)$ 番目の要素を軸として抽出し、これを全 PE に Broadcast する。

★ サンプリング数は扱う要素数によって変わるが、文献⁷⁾では暫定的に 100 としており、同規模の問題を扱う本論文でもこれに習うこととする。

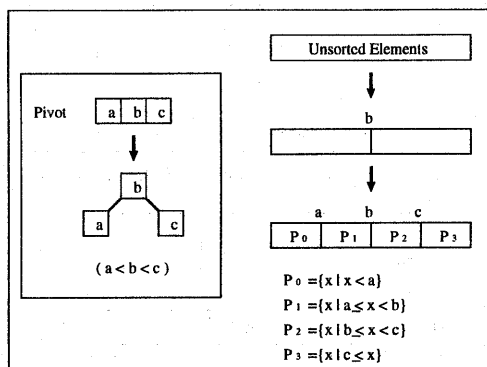


図3 ランダムサンプリングによる並列ソートアルゴリズムの Task 3 における軸に対する操作

- (3) 各 PE において軸に基づき要素列を p 個の部分列に分割する。より小さな要素を含む部分列から順に $0, 1, \dots, (p-1)$ の番号を割り当て、各々の部分列が同じ番号の PE へ送られるように要素を交換する。
- (4) 各 PE 内で受け取った部分列に対して逐次ソートを実行する。

ランダムサンプリングソートはサンプルを無作為に抽出するため、偏ったサンプリングをしてしまった場合には 1PE あたりの負荷を限定することができず、性能が低下してしまう。しかし逐次ソートを 1 回しか実行する必要がなく、平均的にはレギュラーサンプリングソートアルゴリズムよりも性能が向上することが期待される^{*}。

Task 3 における PE 内の要素分割では $(p-1)$ 個の軸をソートして中央の軸を根とする 2 分木を構成し (図 3 を参照)、クイックソートの分割手続きと類似の方法で必要に応じて配列内の要素を交換する。

Task 4 では理論上任意の逐次ソートアルゴリズムを用いることができるが⁸⁾、後述の性能評価においてはクイックソートを実行している。

2.3 負荷分散の手法

レギュラーサンプリングソートアルゴリズムは、比較的良好な負荷バランスを保証するといわれているが、それでも最悪のケースにおいては Task 4 において $\frac{2N}{p}$ 個の要素を処理しなければならない PE が出てくる可能性があり¹⁾、その場合にはこの PE における処理がボトルネックとなって、全体としての性能を損なうことになる。このような事態を避けるための負荷分散の方法を検討する。

^{*} N 個の要素に対して必要とされるステップ数は、クイックソートが $O(N \log N)$ であるのに対して、Task 3 における分割手続きは $O(\frac{N}{p} \log p)$ である。

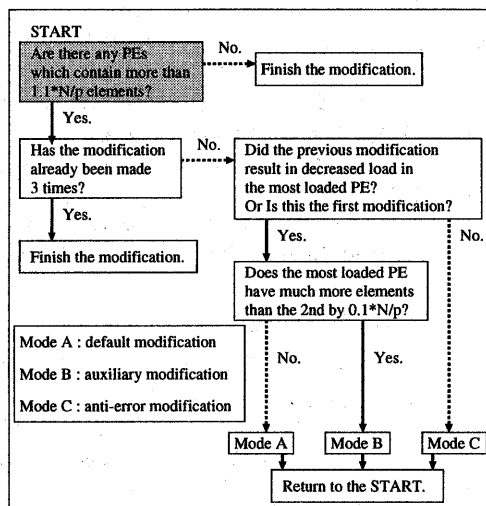


図4 軸補正プログラムのフローチャート

2.3.1 サンプル数の追加

サンプルの標本空間をより大きくとることができれば、その中から抽出した第 n 分位点は真の第 n 分位点に近づくことが期待される。したがって選び出される軸は要素全体をより均等に分けることができるようになる。

ただし、過度にサンプル数を増加させれば軸を決定するまでに要する通信量や処理量もそれに応じて増加することになるであろう。本稿第 3 章では、本来 p 台の PE による実行時に 1PE あたり p 個の要素をサンプリングすべきところをサンプル数を $\frac{1}{2}$ 倍から 8 倍に変化させて実行時間を測定し、性能の変化を調べる。

2.3.2 局所分布に着目した軸の補正

抽出したサンプルの局所的な分布状況から全体の分布を予測し、その予測に応じて抽出済みのサンプルの中からより適している軸を選び直すというアプローチは山岸によって研究されている⁸⁾⁹⁾。

現在のところサンプルの局所的な分布状況を把握するために以下のような統計情報を計算している。

- (1) 軸と同一の値をもつ要素の頻度 (frequency)
- (2) 軸を中心とする 3 次の標準化モーメント、歪度係数 (skewness)
- (3) 軸を中心とする 4 次の標準化モーメント、尖度係数 (kurtosis)

(1) の頻度は、軸と同一の値をもつ要素が多数存在する場合に、それらの要素すべてを同一の PE へ渡すよりも隣接する 2PE 間へまたいで割り当てた方が PE 間の要素数のばらつきを抑えることができる、という考えに基づいている。軸と同一の値をもつ要素が m 個存在する場合、隣接する 2PE 間が m 要素に m 個以上の差がある場合には m 個の要素を要素数の少

ない方へ割り当てる。隣接する 2PE 間がもつ要素数の差が m 個未満の場合には、2PE 間の負荷が均等になるように分けて割り当てる。なお、この方法に従って割り当てられた軸と同一の値をもつ要素は、あらかじめ各 PE 内で最大（または最小）の要素であることが分かっているので、Task 4 において処理の対象とする必要がない。

(2) の歪度係数および (3) の尖度係数に関し、軸 a を中心とする m 次の標準化モーメント α_m は次式で求められる。

$$\alpha_m = \frac{1}{\sigma^m} \sum (X - a)^m$$

ここで、 X はモーメントを計算する対象となる要素を意味し、現在の実装では軸 a を中心に前後 $p-1$ 個のサンプルに対して計算を行っている。また、 σ は標準偏差 (standard deviation) を表し、次式で求めることができる。

$$\sigma = \sqrt{\frac{1}{n} \sum (X - a)^2}$$

ここで、 n は計算の対象となる要素 X の総数を表す。歪度係数は、各要素と軸との相対的な大小関係を 3 乗することにより、分布の偏りを符号として保存しつつ標準化（標準偏差の 3 乗で割ってある）することによって、絶対的な偏りの度合を表している。

尖度係数は、近傍に 4 次のオーダーに勝るだけの分布があるかどうか、近傍以外からの影響がどの程度あるかを標準化した量であり、直感的にはこの値が大きいほど分布は「尖っている」ことになる。「尖っている」分布の近傍では、要素のわずかな大小差で、大幅に順位が変わってしまう。

これらの計算は抽出済みのサンプルに対してのみ行われるため、 $n \ll N$ を仮定すれば、必要とされる処理時間は並列ソート全体の実行時間からみれば微小であると考えられる。現段階におけるこの軸補正プログラムのフローチャートを図 4 に示す。

軸補正プログラムの詳細な説明に入る前に、いくつかの用語を定義しておく。以下ではサンプリングソートの Task 3 において最も多くの要素を割り当てられる PE を最大負荷 PE (the most loaded PE) と定義し、その要素数を最大要素数と呼ぶ。また、Task 3 でも各 PE が N/p ずつの要素を割り当てられることが理想的であるが、これよりも多い要素を持つ PE において、その PE の持つ要素数と N/p との差を超過要素数と呼ぶ。また、以下ではサンプル数を s とおく。

図 4 において Mode A は上で説明した統計量を利用するデフォルトの負荷分散方法である。超過要素数が $0.1 \times N/p$ を越えている PE が補正対象に選ばれ、要素の範囲を規定する 2 つの軸のうちより少ない要素をもつ PE との間に挟まれた側を要素数を減らす向きへ移動する (図 5)。ここでいう移動とは、既に抽出済みのサンプルの中から軸を選び直すことであるが、

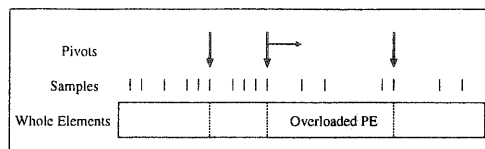


図 5 Mode A における負荷分散の補正方法の模式図

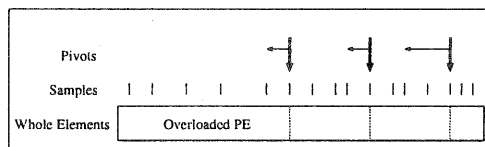


図 6 Mode B における負荷分散の補正方法の模式図

現在の実装においてその移動量は以下のように定めている。

- (1) 移動の対象となる軸の近傍で尖度係数 $\alpha_4 < 3$ の場合には 0。
- (2) 移動の対象となる軸の近傍で歪度係数 $\alpha_3 > 3$ の場合には超過要素数を l として $\frac{l \times s}{2N}$ 。
- (3) それ以外の場合には超過要素数を l として $\frac{l \times s}{N}$ 。

$\alpha_4 < 3$ の場合に補正量が 0 であるのは、あまりに急激な変位を伴う分布に対して補正を行うと逆効果となる恐れがあるためである。正規分布の中心における尖度係数の値が 3 であることから、およそ目安としてそれよりも緩やかな分布のみを補正の対象としている。

一様分布において妥当と考えられる補正量 $\frac{l \times s}{N}$ を歪度係数が 3 よりも大きい場合に半分にしてしているのは、歪みの大きな分布においては少しずつ補正をかけて PE 間の負荷バランスを確かめる方が安全と考えられるためであるが、3 という値は暫定的なもので、この最適値については今後研究の余地がある。

Mode B は特定の PE に対して極端に多くの負荷が分布した場合に Mode A があまり有効に働かないために補助的に用意された補正モードであり、最も多くの要素をもつ PE から最も少ない要素しかもたない PE の間に要素を「ばらまく」(図 6)。 $\frac{l \times s}{N}$ のうち半分以上を最も少ない要素しかもたない PE へ割り当て、残りを両 PE 間に均等に割り当てる。

Mode C は「軸の動かし過ぎ」を防ぐための補正モードである。直前に行われた補正によって別な PE が最も多くの要素をもつようになり、かつその要素数が補正前の最大要素数よりも大きくなっていった場合に適用される。Mode C は直前の補正を半分だけに戻すように作用する。

補正の必要が認められない場合のほか、補正が3回行われた場合にも軸補正プログラムの実行は打ち切りとなる。未補正の場合と3回の補正案の中で最大要素数をもっとも少なくなるような軸の選び方が採用され、処理は通常の並列ソートアルゴリズムのTask 3へ引き継がれる。このフローチャートに従った実装では、軸補正プログラムの実行後に負荷分散が悪化することはないが、必ず最適軸が選ばれるという保証はない。

3. 性能評価

これまでで説明した並列ソートアルゴリズムと負荷分散手法を分散メモリ型並列計算機日立 SR2201 上で実装し、性能比較を行った。SR2201 の各 PE の理論ピーク演算性能は 300MFlops、PE 間は三次元クロスバ網で結合されており、その最大転送性能は 300Mbyte/秒である^{*}。

3.1 問題設定

倍精度浮動小数点数から成る 4 次元ベクトルを要素にもち、下に示すような分布をした問題データに対し性能を評価する。要素間の比較は 2 乗ノルムの大小を順序として、2 乗ノルムが等しい場合は辞書式に第 1 要素から第 4 要素までの大小を順序として決定する。

問題 A：各要素が -100~100 までの一様乱数であるような 160 万個の 4 次元ベクトル

問題 B：ノルムが 5 段階に変化する 640 万個の 4 次元ベクトル

問題 C：ノルムが 5 段階に変化する 1600 万個の 4 次元ベクトル。問題 B よりも要素の重複が多い。

問題 D：問題 B と同じ分布で 1600 万個の 4 次元ベクトル

3.2 実行結果

32PE の SR2201 でレギュラーサンプリングによるアルゴリズムによって各問題を解いたときの実行時間と最大要素数を表 1 に示す。レギュラーサンプリングによるアルゴリズムでサンプル数を変化させ、問題 D を解いたときの実行時間と最大要素数を表 2 に示す。レギュラーサンプリングによるアルゴリズムと軸補正プログラムを組み合わせて各問題を解いたときの実行時間と最大要素数、および適用された補正モードを表 3 に示す。また、提案する負荷分散手法を適用するレギュラーサンプリングソートアルゴリズムと、同手法を適用しないレギュラーサンプリングソートアルゴリズム、そして第 2 章で説明したランダムサンプリングソートアルゴリズムによって問題 B を PE 台数を変えて実行したときの実行時間と PE 台数の関係を

表 1 レギュラーサンプリングによるアルゴリズムの実行結果 (SR2201 32PE)

問題の種類	問題 A	問題 B	問題 C	問題 D
Task 1 (秒)	1.362	4.408	10.04	9.866
Task 2 (秒)	0.064	0.185	0.730	0.660
Task 3 (秒)	0.168	0.210	0.540	0.300
Task 4 (秒)	0.251	1.761	4.680	3.030
合計 (秒)	1.845	6.564	15.99	13.86
最大要素数	40,828	398,782	998,489	507,533

表 2 サンプル数の変化による実行結果の違い (問題 D, SR2201 32PE)

サンプル数	8	16	32	64
Task 1 (秒)	9.924	10.01	9.866	9.897
Task 2 (秒)	0.565	0.645	0.665	0.665
Task 3 (秒)	0.707	0.106	0.302	0.406
Task 4 (秒)	13.39	6.240	3.033	3.781
合計 (秒)	23.46	17.01	13.87	14.75
最大要素数	1,974,909	549,500	507,533	749,542
サンプル数	128	256	512	1024
Task 1 (秒)	9.888	9.917	9.940	10.08
Task 2 (秒)	0.705	0.756	0.759	0.723
Task 3 (秒)	0.303	0.247	0.405	0.398
Task 4 (秒)	3.965	3.858	3.823	3.801
合計 (秒)	14.86	14.78	14.92	14.75
最大要素数	749,542	749,542	749,542	752,801

表 3 軸補正を行ったときのレギュラーサンプリングによるアルゴリズムの実行結果 (SR2201 32PE)

問題の種類	問題 A	問題 B	問題 C	問題 D
Task 1 (秒)	1.253	4.325	10.09	9.846
Task 2 (秒)	0.570	0.370	1.427	0.930
Task 3 (秒)	0.020	0.878	0.175	0.089
Task 4 (秒)	0.101	0.996	3.374	3.164
合計 (秒)	1.944	6.236	15.07	14.03
最大要素数	40,828	206,287	717,417	507,533
補正モード	none	B,A,C	B,B	none

図 7 に示す。なお、各 Task ごとの実行時間は、該当の Task が完了するまでの実行時間と直前の Task が完了するまでの実行時間の差の平均値を示している。

4. 考察

表 2 に端的に現れているように、単純にサンプル数を増やすことが必ずしもより理想的な負荷バランスをもたらすとは限らない。32 よりも多くの PE 台数で実行した場合には、全体としてのサンプル数は更に増大するため、軸を決定するまでに要する通信量や処理量も無視できなくなると考えられる。従って、レギュラーサンプリングソートにおいて過度にサンプル数を増やすことは、一般にはあまり有効な負荷分散手法とは考えにくい。

^{*} 東京大学情報基盤センターが所有している 1024PE の SR2201 のうち 32PE を使用した。PE 間通信には MPI ライブラリを用い、日立最適化 FORTRAN90 コンパイラに -W0,'opt(o(ss))' を指定して生成された実行ファイルをバッチジョブによって実行したときの性能を測定した。

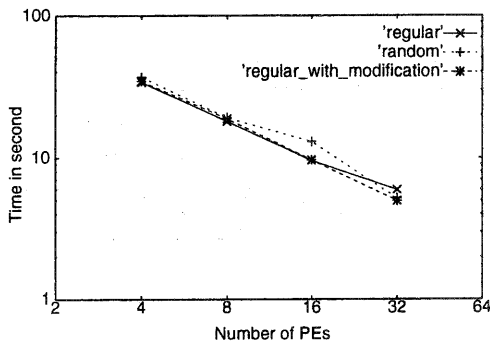


図7 台数効果

表3からは、問題Bと問題Cにおいて提案する手法が適用され、PE間の負荷バランスが向上した結果、並列ソートアルゴリズム全体としての性能が最大で5.8%向上していることが分かる。Task4でマージソートを使っているため処理時間で判断する場合には性能向上の度合は小さく見えるが、最も負荷が集中したPEにおいて処理される要素数(最大要素数)を比較すると、最大で40%近い要素が他のPEへ分散されていることが分かる。

図7では、均等な負荷を仮定した場合にレギュラーサンプリングソートアルゴリズムよりも必要な計算ステップ数が少なく、高速であることが期待されるランダムサンプリングソートアルゴリズムが、16台のPEによる実行時に他のアルゴリズムと比べて極端に悪い性能を示している。実装方法やサンプリング数の調整によって平均的なケースにおける性能を向上させる余地があるとしても、本質的にランダムサンプリングソートアルゴリズムは最悪ケースにおける1PEあたりの負荷を限定することが難しい。その一方で、提案する手法を適用したレギュラーサンプリングソートはPE台数と関係なく安定した性能を示している。

5. おわりに

データの分布に着目した負荷分散手法を提案し、この手法を適用したレギュラーサンプリングソートアルゴリズムと適用しないレギュラーサンプリングソートアルゴリズムとの性能比較を行った。提案する負荷分散手法を適用することにより最大で5.8%の性能向上が得られるケースを確認することができた。

より安定した性能を示す負荷分散手法の開発、およびレギュラーサンプリングソート以外の並列ソートアルゴリズムに本手法を適用する方法の研究が今後の課題である。

参考文献

- 1) Xiaobo Li, Paul Lu, Jonathan Schaeffer, John Shillington, Pok Sze Wong and Hanmao Shi: On the Versatility of Parallel Sorting by Regular Sampling, 1993.
- 2) Hanmao Shi and J.Schaeffer: Parallel Sorting by Regular Sampling, Journal of Parallel and Distributed Computing, 14(4), pp.361-372, 1992
- 3) Selim G.Akl: Parallel Sorting Algorithm, Academic Press, 1985.
- 4) 津田孝夫: 岩波講座ソフトウェア科学 (9) 数値処理プログラミング, 岩波書店, 1988.
- 5) Andrew Tridgell, Richard Brent and Brendan McKay: Parallel Integer Sorting, Australian National University Computer Science Technical Report TR-CS-97-10, 1997.
- 6) Nancy Amato, Ravishankar Iyer, Sharad Sundaresan and Yan Wu: A Comparison of Parallel Sorting Algorithms on Different Architectures, Department of Computer Science Texas A&M University, 1996.
- 7) 吉本芳英: データ数がプロセッサ数よりずっと大きい時の並列ソート, 富士通研究所並列処理研究センター第34回PCRセミナー, <http://www.fujitsu.co.jp/hypertext/fpcrf/j/seminar/34.html>, 1999.
- 8) 山岸秀規: レギュラーサンプリングを用いる局所分布に着目した改良並列ソート, 情報処理学会第59回全国大会講演論文集 (1), pp.193-194, 1999.
- 9) 山岸秀規: サンプリングソートの負荷分散に関する研究, 東京大学情報基盤センタースーパーコンピューティングニュース Vol.2 No.1, pp.32-44, 2000
- 10) D.E.Knuth: The Art of Computer Programming, Vol.3, Sorting and Searching, Addison-Wesley, Menlo Park, 1981.
- 11) 東京大学教養学部統計学教室: 統計学入門, 東京大学出版会, 1991.