A-26

データの事前圧縮によるソートアルゴリズムの高速化に関する考察

A Study on Speed-Up of Sorting Algorithm by Precompressing Data

嶋田 強志† 上土井 陽子‡ 吉田 典可‡

Tsuyoshi SHIMADA Yoko KAMIDOI Noriyoshi YOSHIDA

1 はじめに

近年，記憶装置の大容量化に伴いデータベースシステムなど，格納される情報量も莫大なものとなっており，データの格納方法がその後のデータベース上での処理効率に影響を与える．

本研究では単一プロセッサを仮定し，データを事前にハッシュ関数を使用して，バケットへ分配する格納方法でデータを圧縮し，圧縮データを連結し，連結データを操作することで並列操作を実現する高速なソート技法の開発を目指す．

本稿では，提案アルゴリズムについて示し，提案アルゴリズムの部分列ソートについて実験及び考察する．提案アルゴリズムは1）データの事前圧縮，2）圧縮データの連結，3）圧縮データを利用したソート手続きから構成される．

2 基本となる従来アルゴリズム

本研究の基本となる従来ソートアルゴリズム Range Reduction[1] と Merging Two Words[2] について説明する．

2.1 Range Reduction アルゴリズム

Range Reduction は b ビット，n 個の要素のソート問題を $b/2^k$(k は1以上の整数) ビット，n 個の要素のソート問題に帰着させるアルゴリズムである．入力データ $X = \{x_0, x_1, ..., x_n\}$ が与えられたとき，ハッシュ関数 $H(x_i) = \lfloor x_i/2^t \rfloor$ で x_i を圧縮し，圧縮要素を $x_i \ mod 2^t$ でインデックス付けられたバケットに格納する．ソートはハッシュ関数で圧縮された要素 $\lfloor x_i/2^t \rfloor$ に対して行なわれる．Range Reduction ソートアルゴリズムにおいて，$S(n,b)$ を b ビットの要素 n 個のソートに対する計算複雑さとする．このとき，以下の不等式が成り立つ．

$$S(n,b) \leq S(n, \lceil b/2 \rceil) + O(n)$$

ここで，$O(n)$ は Range Reduction ソートのコストである．Range Reduction アルゴリズムは，要素を構成するビット数が小さいと有効なソートに対して効果的であるが，値域が大きい場合，Range Reduction を用いなかった場合とコストは同じである．

2.2 Merging Two Words アルゴリズム

Merging Two Words は 2^k 要素からなる双調列データを連結し，連結データ上で 2^{k-1} の要素対に対する k 回の比較・交換により，バイトニックソート [2] を実行するアルゴリズムである．連結データの各領域には右から左にアドレスが付けられ，各領域の最上位ビットはフラグである．

図1に Merging Two Words における比較操作について示す．比較は入力列 Z を各ループごとにアドレスビットの最上位から最下位へ，参照するビットを変えながら1と0で分割し，分割データ A，B を得る．次に A のフラグを1にセットし A と B の減算を行なう．減算の結果，フラグが1なら A に属すデータが大きい．交換は減算結果のフラグを基に作成された論理マスクを用い，シフトと論理演算により行なわれる．Merging Two Words の計算複雑さは入力要素数を m とおくと $O(\log m)$ である．Merging Two Words は単一プロセッサ上で並列操作を可能とするが，計算機上で実現する場合に，連結データサイズが大きくなり実装が困難である．例えば，int 型データ (4Byte) を8個連結した場合，連結データサイズは 256 ビットとなる．また，連結データを構成する要素のみに対応するソートしかできない．連結できる要素数は演算器が使用するレジスタの大きさに依存する．

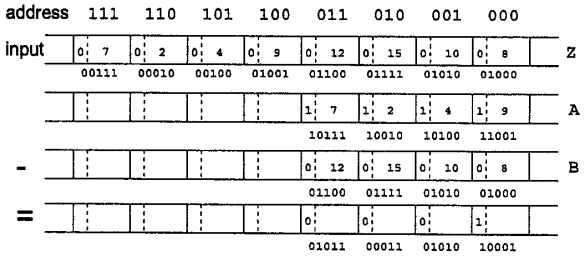

図 1: 連結データ上での要素の並列比較

3 提案アルゴリズム

Range Reduction と Merging Two Words を基礎とする理論上のソートアルゴリズムとして，Signature Sort[3] が存在する．本稿では，Range Reduction と Merging Two Words を基本とし，計算機上での実現を試みる．

3.1 アルゴリズムの概要

Merging Two Words を計算機上で実現する場合に，連結データサイズが大きくなり実装が困難で，大きな入力データ数に対応できない．また，Range Reduction では，データの取り得る値を減少（圧縮）させてもソートの十分な高速化が得られない．提案手法では，Range Reduction を複数回適用し，入力データを計算機上で最も効率的にデータアクセス可能だと考えられる要素単位 1Byte に圧縮し，圧縮データを連結する．次に圧縮・連結データを拡張 Merging Two Words によりソートし，Range Reduction の逆操作により元のデータを得る．

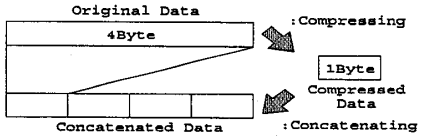

図 2: データ要素の圧縮と連結

演算器が直接操作するデータ（図2）について考える．ある要素が 4Byte の int 型であるとき，この要素を圧縮し 1Byte の圧縮要素を得る．圧縮要素を4個連結し

†広島市立大学大学院 情報科学研究科 (Graduate School of Information Sciences, Hiroshima City University)

‡広島市立大学 情報科学部 (Faculty of Information Sciences, Hiroshima City University)

た場合，圧縮・連結データはint型と同じ大きさとなる．圧縮要素に情報の損失がなければ，要素を演算器が直接操作することで4要素を並列に処理可能で，演算時間が短縮される．本稿では圧縮・連結データを(m,f) Wordと呼ぶ．mは連結した要素数，fは連結する各要素のビット数である．圧縮・連結データサイズは$m\times f$である．図2の例では(4,8)Wordである．

3.2 Merging Two Wordsの拡張

拡張Merging Two Wordsは，$(m,8)$Word（ここでmは2の巾乗）の部分列を入力要素とする．また，部分列ソートにMerging Two Wordsを使用し，分割・併合しながらバイトニックソートを行うことで要素数の大きなデータをソートする．

図3にm=16のMerging Two Wordsを部分列ソートとする拡張Merging Two Wordsを示す．図3の入力データ列はすでにRange Reductionにより1Byteに圧縮されている．最初に入力列を16要素ごとに連結する．次に1st stepでは入力列がランダムであるため，長さ2の双調列を8個，昇順，降順にソートし，長さ4の双調列を4個生成し，昇順，降順にソートする．これを繰り返し，長さ16の双調列でMerging Two Wordsをそれぞれ実行する．2nd step以降では通常のMerging Two Wordsによる部分列のソートを行なう．また，各Merging Two Wordsの出力は半分に分割・併合され新たな(16,8)Wordの双調列とみなせ，この双調列に対するMerging Two Wordsによるソートを逐次的に繰り返し，全体をソートする．

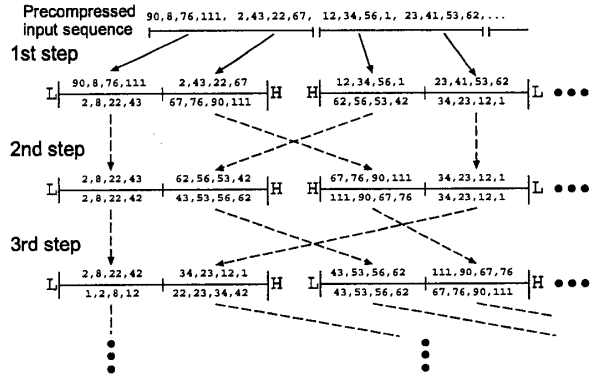

図3: 分割・併合によるソート

3.3 提案アルゴリズムの計算複雑さ

提案アルゴリズムの計算複雑さについて示す．未圧縮の全入力データ要素数をn，圧縮データ要素の連結数をm $(n > m \geq 2)$とおく．Range Reductionの計算複雑さは$O(n)$である．また，Merging Two Wordsによる交換コストは1st stepが$\log^2 m$で2nd step以降のMerging Two Wordsによる交換コストは$\log m$である．$M(m,n)$を連結数m，要素数nのときの拡張Merging Two Wordsの計算複雑さとする．このとき，以下の関係式が成り立つ．

$$M(m,n) = \log^2(\frac{2n}{m})\cdot\frac{n}{m}\cdot\log m + \frac{n}{m}\cdot\log^2 m$$
$$\leq \frac{\log m}{m}\cdot n\cdot(\log^2 n + \log m)$$

$m=1$のとき上界，$U(1,n) = n\log^2 n$，

$m=n$のとき下界，$L(n,n) = \log^2 n$となる．

よって，以下の不等式が成り立つ．

$$n\log^2 n > M(m,n) > \log^2 n$$

Range Reductionの適用回数をCとすると，提案アルゴリズム全体の計算複雑さ$T(m,n)$は，以下の等式により表される．

$$T(m,n) = M(m,n) + C\cdot O(n)$$

以上よりmが大きい程，ソート時間は短縮される．

4 実験と考察

提案アルゴリズムの部分列ソートMerging Two WordsについてC言語を用いて実現し，2つのソート列（双調列）を併合するMerge Sort（計算複雑さは$O(n)$）及びQuick Sort（計算複雑さは$O(n\log n)$）と比較した．使用計算機はUltra SPARC-IIi, 要素数16,8,4をもつ双調列な入力データに対し，アルゴリズムを10^6回繰り返し適用し，CPU時間を計測した．連結データ数が16，8のとき使用言語の制約上，アルゴリズムに忠実な実現ができないため連結データ数4のMerging Two Wordsを再帰的に使用し実現した．入力データは双調列である．実験結果を表1に示す．

表1: 各手法の計算時間 [s]

	入力データの要素数		
	4	8	16
Merging Two Words	0.77	2.05	4.57
Merge Sort	0.57	1.21	2.33
Quick Sort	1.34	3.56	14.24

拡張Merging Two Wordsは部分列ソートMerging Two Wordsの並列操作性を利用したソートアルゴリズムである．そのため，Merging Two Wordsの実行時間がMerge Sortより遅ければ，部分列ソートにMerge Sort適用すればよいということになる．また，Merge Sortを部分列ソートとする$O(n\log n)$のソートアルゴリズム[4]が知られているため，Merging Two WordsはMerge Sortより実行が速い場合，高速化の可能性がある．Merging Two Words は$m=8,16$のとき，言語の制約により忠実な実装ができなかったため，Merge Sortより全てのデータで遅い結果となった．アルゴリズムに忠実な実装が可能であれば，理論上で実行時間は$m=8$のとき3/5倍，m= 16のとき1/3倍程度に短縮される．

5 まとめ

実験結果より提案アルゴリズムの部分列ソートMerging Two Wordsについて，連結データ数以内でアルゴリズムに忠実な実装が可能ならMerge Sortより高速であることが分かった．提案アルゴリズムでは，単一プロセッサ上の圧縮・連結データの並列操作によるソートの高速化を試みたが，単一プロセッサのソートの最適計算時間$O(n\log n)$を上回ることができなかった．今後の課題として，提案アルゴリズム全体の実現及びその実験的評価，並びに部分列ソートMerging Two Wordsの効率的な実現のためのハードウェアの構築，部分列ソートMerging Two Wordsを使用した新たな分割・併合ソートの開発が挙げられる．

参考文献

[1] D.Kirkpatric and S.Reisch,"Upper bounds for sorting integers on random access machines",Theoret.Computer Science,Vol.28, pp.263-276, 1984.

[2] A.Albers and T.Hagerup,"Improved parallel integer sorting without concurrent writing",Proc.3rd Annual ACM-SIAM Symp. on Discrete Algorithms, pp.463-472, 1992.

[3] A.Andersson,T.Hagerup,S.Nilsson and R.Raman,"Sorting in Linear Time?",Proc.27th Annual ACM Symp. on Theory of Computing, pp.427-436, 1995.

[4] R.Sedgewick,"Algorithms IN C++ THIRD EDTION",2000.