ベクトル計算機上でのソーティング†

石浦 菜岐佐†† 高木 直史†† 矢島 脩三††

ソーティングをベクトル計算機上で高速に実行する方法について考察する．ベクトル計算機上での実行を考える場合には，プログラムのどれだけの部分がベクトル処理できるか，処理ベクトル長をどれだけ長くとれるか，といったベクトル処理との整合性が問題となる．本論文では，このような点を考慮し，ベクトル計算機の持つベクトル命令を生かした，いくつかのソーティング手法を提案する．ベクトル計算機 FACOM VP-200 上で性能評価を行った結果，1) スカラ計算機上で高速なアルゴリズムである頻度法，S整列法は，部分的にベクトル処理可能で，ベクトル計算機上でも高速となり，クイックソート等に比べて格段に高速である，2) 基底法，および本論文で新たに提案する番地計算基底法は，スカラ計算機上では頻度法やS整列法に劣るが，ほぼ全過程がベクトル処理できるためベクトル計算機上では逆に高速となりうる，3) S整列法等，番地計算を行うアルゴリズムで必要となる局所ソートには，スカラ計算機上では単純挿入法が有効であるが，ベクトル計算機上では奇偶置換法が有利である，等の興味深い結論を得た．

1. まえがき

ソーティングは計算機におけるデータ処理の基本演算として広く用いられており，高速アルゴリズムの開発は従来から計算機科学における重要な研究課題の一つとなっている．一方，近年の大規模計算の処理要求を満たすべく，パイプライン処理方式のスーパコンピュータ，ベクトル計算機が開発され[1),2)]，広く利用されるようになりつつある．ベクトル計算機は，数値計算を主目的に開発されたものであるが，浮動小数点演算以外の演算能力も強化されており，数値計算以外の計算への応用にも有効であることが報告されている[3)]．ベクトル計算機は，今後ますます広い分野で利用されるようになると予想されるが，その際には，基本演算の一つであるソーティングのベクトル処理の良否が，全体の処理効率を大きく左右することも考えられる．本論文ではこのような背景をふまえ，ソーティングをベクトル計算機上で高速に実行する方法について考察する．

ソーティングの計算時間は与えられるデータの性質，すなわち，ソートすべき要素の数，データ長，キー長，キー値の範囲，キー値の分布等に大きく依存する．このため，古くから様々なソーティング・アルゴリズムが開発され，データの性質による計算時間のふるまいが研究されている[4),5)]．また，並列計算機や専用ハードウェア上でのソーティングの研究も盛んであり，多くの並列ソーティング・アルゴリズムが提案されている[6)]．

ベクトル計算機上でのソーティングに関しても文献7），9）に見られるような研究がある．これらの研究は，比較と交換によるソーティングを考察の対象としたものである．比較と交換によるソーティングの最良の逐次アルゴリズムとしては，要素数 n に対する平均計算時間が $O(n \log n)$ のものが知られているが，いずれもベクトル処理には適合しにくいと考えられる[7),8)]．このため，ベクトル計算機上では Batcher の並列マージソート[4)]，ダイアモンドソート[9)] 等，平均計算時間が $O(n \log^2 n)$ ではあるが，データの流れが事前に決定できるものが有効であると報告されている[7),9)]．

これに対し本論文では，比較と交換だけではなく，間接参照による書き込み，ビット演算，番地計算等を利用し，平均計算時間が $O(n \log n)$ 以下のソーティング手法をベクトル計算機上で実現することを考える．本論文ではいくつかのソーティング・アルゴリズムについてそのベクトル処理手法を考察したが，そのなかでも，

1）頻度法[4)]，およびこれに番地計算と局所ソートを組み合わせたS整列法[5)]，

2）基底法[4)]，およびこれに番地計算と局所ソートを組み合わせた番地計算基底法，

が特に有効であるという結論が得られた．頻度法，S整列法はスカラ計算機上で最も高速なアルゴリズムの一つである．これらは部分的にではあるがベクトル処理可能であり，ベクトル計算機上でも高速なアルゴリズムとなる．また，基底法，および本論文で新たに提案する番地計算基底法は，スカラ計算機上では頻度

† Sorting on a Vector Processor by NAGISA ISHIURA, NAOFUMI TAKAGI and SHUZO YAJIMA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室

法やS整列法に劣るが，ほぼ100%ベクトル処理できるため，ベクトル計算機上では逆に高速となりうる．また，S整列法や番地計算基底法で必要となる局所ソートには，スカラ計算機上では単純挿入法が高速であるが，効率の良いベクトル処理は難しい．本論文では局所ソートにバブルソートの並列版として知られる奇偶置換法を用いることを提案する．

我々はベクトル計算機 FACOM VP-200 上に上記ソーティング手法を含む9種類のソーティング・プログラムを実現し，性能評価を行った．その結果，上記のソーティング手法はベクトル処理に適したものであり，比較と交換によるソーティング手法に比べ格段に高速であることが確認された．

以下，2章でベクトル計算機に関する基本的事項について述べた後，3章ではベクトル計算機向きのソーティング手法について述べる．4章では実験結果に基づき，ベクトル計算機上でのソーティング手法の性能評価および考察を行う．

2. ベクトル計算機

ベクトル計算機は演算パイプライン方式のスーパコンピュータであり，ベクトル・データ（配列データ）に対する同一の繰り返し処理を演算パイプラインで実行することにより，高速計算を実現する．近年開発されたベクトル計算機の最大性能は数百 MFLOPS に及び，超大型汎用機の数十倍にも達する[1),2)]．しかし，ベクトル計算機の計算力を十分に引き出すためには，プログラムの大部分がベクトル処理可能で，かつ処理ベクトル長が十分長いことが不可欠である．さらに命令の種類，メモリ・アクセスの種類，同時に実行できる演算の数等の様々な要因も実効性能に大きく影響する．このようなベクトル計算機の特性をコーディングに反映させることはむろん重要であるが，さらにプログラムの基礎となるアルゴリズムに対しても，ベクトル処理への適合性を考慮することが重要である．

以下では，ある演算や操作をベクトル計算機のベクトル命令で実行することをベクトル処理と言う．また，ベクトル計算機で実行することを意識したコーディングやアルゴリズムをベクトル版（V版）と呼び，これを意識しない通常のコーディングやアルゴリズムをスカラ版（S版）と呼ぶことにする．プログラムをベクトル命令を用いて実行することをベクトル実行（V実行），ベクトル命令を用いないで実行することをスカラ実行（S実行）と呼ぶ．S版（あるいはV版）をS実行したときの処理時間とV実行したときの処理時間の比を加速率と呼ぶ．また，混同の恐れがなければ，S版をS実行したときの処理時間と，V版をV実行したときの処理時間の比をも加速率と呼ぶことにする．

今回の実験で使用したベクトル計算機は，FACOM VP-200（京都大学大型計算機センター）である[1)]．VP-200 は，広範な応用を指向してベクトル処理の範囲を広げるために，様々な処理機能を備えている．データ型に関しては，浮動小数点数ばかりでなく，整数，論理データもベクトル演算の対象となっており，32ビット1ワードのデータに対する加減乗算，ビットごとの論理演算が高速に実行される．主記憶は最大256 MB であり，1）連続アクセス，2）等間隔アクセス，3）リスト・ベクトル・アクセスの3種のベクトル・アクセスが可能である．リスト・ベクトル・アクセスは，

```
    DO 10 I=1, N
10  A(I)=B(L(I))
```

のように，整数配列の要素をインデクスとする間接アクセスである．また，ベクトルの要素を並べかえる有効なベクトル命令としては，ベクトル収集操作がある．これは，

```
    DO 10 I=1, N
      IF(条件式 (I)) THEN
        K=K+1
        B(K)=A(I)
      ENDIF
10  CONTINUE
```

という Fortran 文で示されるように，配列要素のうち特定の条件を満たすものだけを集めて新たな配列を構成する操作である．さらに，1）配列要素のうち特定の条件を満たすものだけを処理の対象とする条件付きベクトル演算，2）配列要素の値の合計を求める総和演算，3）配列中の最大，最小の値を持つ要素の検索等を実行するベクトル命令も備えられており，多岐にわたるベクトル処理が可能になっている．

上記のベクトル処理機能は VP-200 に限られたものではなく，HITAC S-810[1)]，NEC SX-2[2)]等のベクトル計算機にも備えられているものであり，以下の議論はこれらの計算機にも共通するものである．

3. ベクトル計算機向き高速ソーティング手法

ソーティングは，キーを持つ要素の列を入力とし，その要素をキー値の昇順あるいは降順に並べ換えたものを出力とする．以下，要素の数n，要素列のi番目の要素を a[i]（ただし，i=1〜n），a[i] のキーを a[i]. key と表すことにする．また，本章では簡単のため，キーは0から key_{max} までの整数値をとるものとし，キー値の昇順に並べ換える場合を考える．

本論文では，2章で述べたベクトル計算機のリスト・ベクトル・アクセス，ビットワイズ論理演算，ベクトル収集操作などを利用し，比較と交換という操作だけに限定されないソーティングを考える．本章で考察するアルゴリズムは，（1）番地計算法，（2）頻度法，（3）S整列法，（4）基底法，（5）番地計算基底法，の5つである．このうち，（2）と（4）は主に整数キーを対象とするものであるが，（1），（3），（5）は実数（浮動小数点数）を扱うこともできる．以下，各アルゴリズムの概要と，そのベクトル処理手法について述べ，さらに（1），（3），（5）で必要になる局所ソートのベクトル処理手法について述べる．

3.1 番地計算法 (Address Calculation Sort)[4]

データの分布が分かっているとき，その分布に基づいて各要素の順位（行き先の番地）を計算し，この順位に従って要素の並べ換えを行う（計算された番地に要素を挿入する）．データの分布が分からないときは，何らかの分布（例えば，一様分布）を仮定して番地を計算する．異なる要素に対して計算された番地が同じになることを，衝突が起こるという．衝突が起こった際の処理は種々考えられるが，ここでは挿入時に計算された番地に最も近い空き番地に要素を挿入し，最後に局所ソートを行うという方法をとることにする．衝突が頻繁に起こると処理効率が極めて悪くなるので，行き先の番地を 1〜n よりも多くとり，衝突を減らそうとすることが多い．この場合には，最後に空領域をつめる操作も必要となる．番地の計算は要素ごとに行えるので容易にベクトル処理できる．例えばキー値の分布を一様分布と仮定した場合，行き先の番地を 0〜$L-1$ として，$(L-1)\times$a[i]. key/key_{max} により行き先の番地を計算することができる．計算された番地への要素の挿入は，衝突が起こらないことが保証できればリスト・ベクトル・アクセスによりベクトル処理できるが，一般には衝突は避けられないのでベクトル処理はできない．空領域をつめる操作は，ベクトル収集操作によりベクトル処理できる．局所ソートに関しては後述するが，スカラ版には単純挿入法を，ベクトル版には奇偶置換法を用いればよい．

3.2 頻度法 (Distribution Counting Sort)[4]

キーのとりうるすべての値に対してカウンタを準備し，与えられた要素列の中にキー値の大きさがk（ただし，k=0〜key_{max}）である要素がそれぞれいくつあるかを数える．次に，このカウンタの値を累算することによって，キー値がkの要素が，ソート後何番目の位置から始まるかを計算し，この値に基づいて各要素を最終的な位置に挿入する．

アルゴリズム「頻度法」

```
for k:=0 to key_max do count[k]:=0 od;
for i:=1 to n do
  count[a[i].key]:=count[a[i].key]+1 od;
      ……(d1)
position[0]:=0;
for k:=0 to key_max-1 do
  position[k+1]:=position[k]+count[k] od;
      ……(d2)
for i:=1 to n do
  k:=a[i].key;
  position[k]:=position[k]+1;
  a'[position[k]]:=a[i] od;……(d3)
for i:=1 to n do a[i]:=a'[i] od;        □
```

頻度法では，key_{max}+1 個のカウンタが必要になるため，key_{max} があまり大きい（キー値のとりうる範囲が広い）と記憶域の制限から実行が不可能となる．さらに，key_{max} が要素数nに比べて大きいと，累算部(d2)の計算量（key_{max} に比例する）が支配的になり処理効率が悪くなる．しかし，key_{max} が小さいときにはスカラ実行で最も高速な計算法の一つである．

カウント部（d1），は現在のベクトル計算機ではベクトル処理できない．累算部(d2)は VP-200 ではベクトル処理できないが，ベクトル処理可能な計算機も存在する（詳細は4章で述べる）．挿入部(d3)はこのままではベクトル処理できないが，(d1) のカウント部でカウントと同時に，同じキー値をもつ要素の中での出現順位を記憶しておき，position とこの順位の和によって挿入位置を求め，この位置にリスト・ベクトル・アクセスで書き込みを行うという方式により，ベクトル処理可能である．

3.3 S整列法[5]

頻度法は，key_{max} が小さい（キー値の範囲が狭い）場合には非常に高速であるが，key_{max} が大きい場合には必要記憶量が増大し，処理効率も悪くなる．S整列法はこのような問題点を解決し，key_{max} が大きい場合にも（あるいはキーが浮動小数点数の場合にも）効率の良いソーティングを可能とするものである．番地計算法と同様の方法でいったんキー値を小さい値に変換した上で頻度法による並べ換えを行う．番地の衝突は最後に局所ソートを行って解決する．

アルゴリズム「S整列法」

番地計算を行い，各要素に 0〜$L-1$ のラベルを付ける；……(s 1)

頻度法により，要素列をラベルの大きさに従って並べ換える；……(s 2)

局所ソートを行い，同じラベルを持つ要素を並べ換える；……(s 3)

□

(s 1)の番地計算部は番地計算法と同様，ベクトル処理可能である．頻度法によるソーティング(s 2)は前節に述べたとおり，カウント部(d 1)と累算部(d 2)以外はベクトル処理できる．局所ソートは奇偶置換法によればベクトル処理できる．

S整列法は key_{max} が大きい場合にはスカラ実行で最も高速なソーティング手法の一つである．計算の全過程をベクトル処理することはできないものの，ベクトル処理できる部分の比率が大きいため，ベクトル実行においても高速な計算手法であると考えられる．

3.4 基底法 (Radix Sort)[4]

各要素のキー値を r 進数で表現したときの，各桁の値によって要素を分類する，という操作を下位桁から順次行うことによりソーティングを行う．計算機上では通常データが2進数で表現されていることを利用し，$r=2$として計算する．

アルゴリズム「基底法」

```
keylength := truncate (log2 (key_max)) + 1 ;
for k := 1 to keylength do
  m := 0 ;
  for i := 1 to n do
    if a[i].key の下位から k ビット目が 0
      ……(r 1)
    then m := m + 1 ; a'[m] := a[i] ……(r 2)
    fi od ;
  for i := 1 to n do
    if a[i].key の下位から k ビット目が 1
      ……(r 3)
    then m := m + 1 ; a'[m] := a[i] ……(r 4)
    fi od ;
  for i := 1 to n do a[i] := a'[i] od ;
od ;                                          □
```

ただし，truncate (x) は x の小数部切り捨てを表す．基底法によるソーティングの計算時間は $n \times$ keylength に比例するので，key_{max} が小さい（キー値のとりうる範囲が狭い）ほど高速である．(r 1)(r 3)の判定はベクトル論理演算により，(r 2)(r 4)の操作はベクトル収集操作により処理できるので，本手法はほぼ全過程がベクトル処理できる．処理ベクトル長が常に n であるため，加速率は大きいと考えられる．

3.5 番地計算基底法

ベクトル計算機向きのソーティング・アルゴリズムとして，本論文で新たに提案するものである．S整列法が，番地計算，頻度法，局所ソートを組み合わせているのと同様に，番地計算，基底法，局所ソートを組み合わせた計算手法である．

アルゴリズム「番地計算基底法」

番地計算を行い，各要素に 0〜$L-1$ のラベルを付ける；……(a 1)

基底法により，要素列をラベルの大きさに従って並べ換える；……(a 2)

局所ソートを行い，同じラベルを持つ要素を並べ換える；……(a 3)

□

基底法はキー長に比例した処理時間が必要となるので，key_{max} が大きいと効率が悪くなる．これに対し番地計算基底法は，番地計算により短いキー長で大局的ソートを行うので，key_{max} が大きいときにも高速である．また，S整列法に比べると，全過程がベクトル処理できるため，加速率は大きくなると考えられる．

3.6 局所ソート

番地計算法，S整列法，番地計算基底法では最後のステップで，同じラベルを持つ要素を並べ換える局所ソートが必要になる．局所ソートでは要素がほとんどキー順に並んでいるので，従来のスカラ計算機上ではこの性質を利用して，単純挿入法[4]や単純交換法（バブルソート）[4]により処理を高速に行うことが可能である．文献5)ではS整列法の局所ソートに単純挿入法を用いており，我々が簡単な実験を行った結果で

も，単純挿入法が高速であった．しかし，単純挿入法はデータがほとんど並べ換えられているという条件を生かした上で効率良くベクトル処理するのが難しい．一般のソーティングを対象とする場合にはほぼ 100% ベクトル処理可能であるが，これを局所ソートに適用すると，無駄が多く効率が悪くなってしまう．このように単純挿入法はベクトル計算機上での局所ソートには適さないと考えられる．

一方，バブルソートも逐次的な性質を持つためベクトル処理不可能であるが，その並列版として知られる奇偶置換法 (Odd-even transposition sort)[4] は容易にベクトル処理でき，しかもデータがほとんど並んでいるという性質を利用して，高速に局所ソートを行うことができる．奇偶置換法は隣接する要素間でのキー値の比較，交換を繰り返すことにより並べ換えを行うものであり，アルゴリズムは次のようになる．

アルゴリズム「奇偶置換法」

repeat

a [1]. key と a [2]. key, a [3]. key と a [4]. key, …をそれぞれ比較し後者が小さければ要素を交換する, ……(e 1)

a [2]. key と a [3]. key, a [4]. key と a [5]. key, …をそれぞれ比較し後者が小さければ要素を交換する ……(e 2)

until ((e 1) (e 2)ともに交換が行われなくなる)；

□

(e 1) (e 2)の操作は，ベクトル比較演算，条件付きベクトル演算によりベクトル処理できる．本手法は全過程がベクトル処理可能であり，また，データがほとんどキー順に並んでいるという性質を利用できるので，ベクトル計算機上での局所ソートには最も適していると考えられる．

4. 性能評価と考察

4.1 各アルゴリズムの性能

前章で述べたソーティング手法の性能評価を行うため，ベクトル計算機上での処理時間を測定する実験を行った．プログラミング言語は Fortran 77，計算機は FACOM VP-200 (京都大学大型計算機センター) である．比較のため，3章で述べたアルゴリズムも含め，次の9つのソーティング・アルゴリズムについて，プログラムを作成した．ベクトル処理の効果を評価するため，各アルゴリズムに対してベクトル版，スカラ版の両方を作成した（ただし，両者が一致するものもある）．それぞれのベクトル処理手法を簡単に示す．

（1）番地計算法，（2）頻度法，（3）S整列法，（4）基底法，（5）番地計算基底法は3章で述べたとおりである．

（6）単純選択法[4]：最大値/最小値検索命令を用いてほぼ全過程がベクトル処理できる．

（7）単純挿入法[4]：3章で述べたとおり局所ソートには適さないが，一般の場合にはベクトル比較演算，総和演算等を用いてほぼ全過程がベクトル処理できる．

（8）単純交換法[4]：スカラ版はバブルソート，ベクトル版は3章で述べた奇偶置換法である．

（9）クイックソート[4]：分割操作をベクトル収集操作を用いてベクトル処理した．分割が進むにつれ，ベクトル長が短くなるという欠点があるので，分割がある程度進んだ段階で，スカラ版は単純挿入法に，ベクトル版は奇偶置換法に切り換えるバージョン（「クイックソート2」と呼ぶ）も作成した．

ソーティングの対象となる要素は4バイトのキーと4バイトのポインタから成るものとした．キー値には乱数発生ルーチン（合同乗算法による）で発生させた一様乱数を用いた．また，番地のとる範囲は，番地計算法では $0 \sim 2n-1$，S整列法および番地計算基底法では $0 \sim n-1$ とした．このため，番地計算基底法の平均処理時間は $O(n \log n)$ となっている．

表1は，各アルゴリズムについて，要素数 n を 2^6, 2^{10}, 2^{14} としたときの処理時間を測定した結果である．key_{max} の値は $2^{15}-1$ である．この表より，各アルゴリズムのベクトル計算機との整合性に関して以下が観察される．

1）単純選択法，単純挿入法，単純交換法は，要素数 2^{14} において，10倍から50倍近い加速率を達成しており，ベクトル計算機との整合性が良いといえる．しかし，これらのアルゴリズムは，要素数 n に対し平均処理時間が $O(n^2)$ であるため，平均処理時間が $O(n \log n)$ や $O(n)$ のアルゴリズムに比べると，n が大きくなった場合の処理時間がベクトル処理の効果以上に増大し，効率が悪くなる．逆に n が小さい場合にはベクトル長が十分とれないため，加速率は小さい．したがって，これらのアルゴリズムは単独で用いる場合，ベクトル計算機で実行する利点はないと考えられる．

2）クイックソートはそのままではほとんど加速されない．また，分割を途中で打ち切り単純法に切り換

えた場合でも，あまり大きな加速率は得られない．

3）番地計算法はほとんど加速されない．

4）頻度法は要素数が多い場合スカラ実行において最も高速であり，S整列法はこれに次ぐ．頻度法は部分的にしかベクトル処理できないため，加速率はあまり大きくない．しかし，HITAC S-810[1]は，

```
DO I=2,N
10  A(I)=A(I-1)*B(I)+C(I)
```

で示される一次回帰演算を処理するベクトル命令を備えており，頻度法の累算部(d2)をベクトル処理することができるため，加速率は大きくなる．**表2**は頻度法，S整列法について同じプログラムを S-810/20 (東京大学大型計算機センター) 上で実行した結果であるが，VP-200 の場合よりも加速率が大きくなっている．この2つのアルゴリズムはベクトル計算機との整合性はあまり良くないものの，ベクトル計算機上においても高速なアルゴリズムであるといえる．

4）基底法および番地計算基底法は，スカラ実行では頻度法，S整列法，番地計算法などのアルゴリズムよりも遅いが，ほぼ全過程がベクトル処理されるため

表1 各アルゴリズムの処理 CPU 時間
Table 1 Processing CPU time for each sorting algorithm.

	処理 CPU 時間 [μsec] 上段：要素数 $n=2^8$ 中段：要素数 $n=2^{10}$ 下段：要素数 $n=2^{14}$		
	S版S実行	V版V実行	(加速率)
単純選択法	657	325	(2.0)
	143,000	7,080	(20.2)
	36,492,000	785,000	(46.5)
単純挿入法	493	480	(1.0)
	99,100	14,400	(6.9)
	26,315,000	1,935,000	(13.6)
単純交換法	1,380	579	(2.4)
	357,000	28,100	(12.7)
	96,446,000	5,908,000	(16.3)
クイックソート	549	814	(0.7)
	13,500	14,800	(0.9)
	282,000	212,000	(1.3)
クイックソート 2	365	337	(1.1)
	10,100	2,510	(4.0)
	227,000	43,400	(5.2)
番地計算法	395	248	(1.6)
	6,400	2,680	(2.4)
	114,000	51,400	(2.2)
頻度法	12,400	8,970	(1.4)
	15,300	9,750	(1.6)
	64,800	22,100	(2.9)
S整列法	319	111	(2.9)
	5,070	961	(5.3)
	91,000	15,600	(5.8)
基底法	754	248	(3.0)
	11,300	1,020	(11.1)
	214,000	14,300	(15.0)
番地計算基底法	651	189	(3.4)
	14,000	950	(14.7)
	347,000	15,900	(21.8)

($key_{max}=2^{15}-1$, FACOM VP-200)

表2 頻度法，S整列法の S-810 上での処理 CPU 時間
Table 2 Processing CPU time for distribution counting sort and S-sort on S-810.

	処理 CPU 時間 [μsec] 上段：要素数 $n=2^8$ 中段：要素数 $n=2^{10}$ 下段：要素数 $n=2^{14}$		
	S版S実行	V版V実行	(加速率)
頻度法	12,800	3,430	(3.7)
	16,100	4,500	(3.6)
	61,000	14,900	(4.4)
S整列法	397	129	(3.1)
	6,360	1,420	(4.5)
	105,000	21,200	(5.0)

($key_{max}=2^{15}-1$, HITAC S-810/20)

表3 大きな要素数に対する処理 CPU 時間
Table 3 CPU time for processing large number of elements.

key_{max}	n	処理 CPU 時間 [msec]			
		頻度法	基底法	S整列法	番地計算基底法
2^7-1	2^{14}	8.1	8.5	13.9	14.1
	2^{15}	16.1	16.7	27.9	29.7
	2^{16}	32.2	33.6	56.2	61.4
	2^{17}	64.1	67.1	117.0	129.0
$2^{15}-1$	2^{14}	22.1	14.3	15.6	15.9
	2^{15}	34.5	28.6	36.5	29.3
	2^{16}	59.9	56.9	81.1	61.6
	2^{17}	110.5	133.8	170.0	130.0
$2^{31}-1$	2^{14}	*	26.4	17.0	16.4
	2^{15}		52.8	40.8	34.1
	2^{16}		105.6	92.4	71.9
	2^{17}		211.3	193.0	155.0

(FACOM VP-200, V版V実行)

* 記憶領域の制約上実行不可能

加速率が15～20倍と大きく，ベクトル実行では逆に高速となる場合がある．

5）頻度法，基底法，S整列法，番地計算基底法は，クイックソート等に比べて格段に高速である．

表3は，頻度法，基底法，S整列法，番地計算基底法の4つのアルゴリズムについて，さらに大きな要素数に対するベクトル実行の処理時間を測定した結果である．キー値の取りうる範囲と処理時間の関係を調べるために，$key_{max}=2^7-1$, $2^{15}-1$, $2^{31}-1$ の3通りについて測定を行った．これより，以下が観察される．

1）key_{max} の値が小さいときには，頻度法，基底法が高速であるが，key_{max} の値が大きいときには，逆にS整列法，番地計算基底法が高速である．

2）頻度法と基底法では，要素数に比べ key_{max} が小さい場合には頻度法のほうが高速であるが，key_{max} が大きい場合には，基底法のほうが高速である．また，key_{max} があまり大きい場合には，頻度法は記憶域の制約から実行不可能となる．

3）S整列法と番地計算基底法では，この n の範囲では全般に番地計算基底法のほうが高速である．

基底法と頻度法，あるいはS整列法と番地計算基底法の優劣は，ベクトル計算機のハードウェア構成にも依存すると考えられる．今後，単純にベクトル命令の処理速度が高速化されるならば，ベクトル命令の比率の高い基底法，番地計算基底法が高速になると考えられる．逆に，頻度法のカウント部(d1)に現れるような操作をベクトル処理するハードウェアが構成され，累算部(d2)に現れる一次回帰演算の処理が強化されれば，頻度法，S整列法が有利になるであろう．

4.2 局所ソート

本論文では，ほとんどソートされている要素列のソートをベクトル計算機上で行う場合には，単純挿入法よりも奇偶置換法を用いることを提案している．その効果を示すため，S整列法において局所ソート部を両方のアルゴリズムで実現し，処理速度を測定した．結果を**表4**に示す．表中①はS整列法のスカラ版であり，局所ソートには単純挿入法を用いている．②はS整列法のベクトル版であり，局所ソートには奇偶置換法を用いている．③，④はベクトル版において，局所ソートを置き換えたものであり，③は単純挿入法ベクトル版のベクトル実行を，④は単純挿入法スカラ版のスカラ実行を用いている．③と④を比較すると，③のほうが大幅に遅くなっている．これは3.6節でも述べたとおり，単純挿入法ベクトル版は局所ソートとして用いた場合には，計算の無駄の増加，ベクトル長の短縮をまねき，極めて効率が悪くなるためである．②と④の比較から，ベクトル実行における局所ソートには，単純

表4 局所ソートの性能比較（S整列法の処理 CPU 時間 [μsec])
Table 4 Performance comparison of local sort. (CPU time for S-sort [μsec])

		①	②	③	④
番地計算と頻度法		(S版S実行)	(V版V実行)	(V版V実行)	(V版V実行)
局所ソート		単純挿入法 (S版S実行)	奇偶置換法 (V版V実行)	単純挿入法 (V版V実行)	単純挿入法 (S版S実行)
n	64	319	111	520	201
	1024	5,070	961	10,900	2,850
	16384	91,000	15,600	326,000	46,300

($key_{max}=2^{15}-1$, FACOM VP-200)

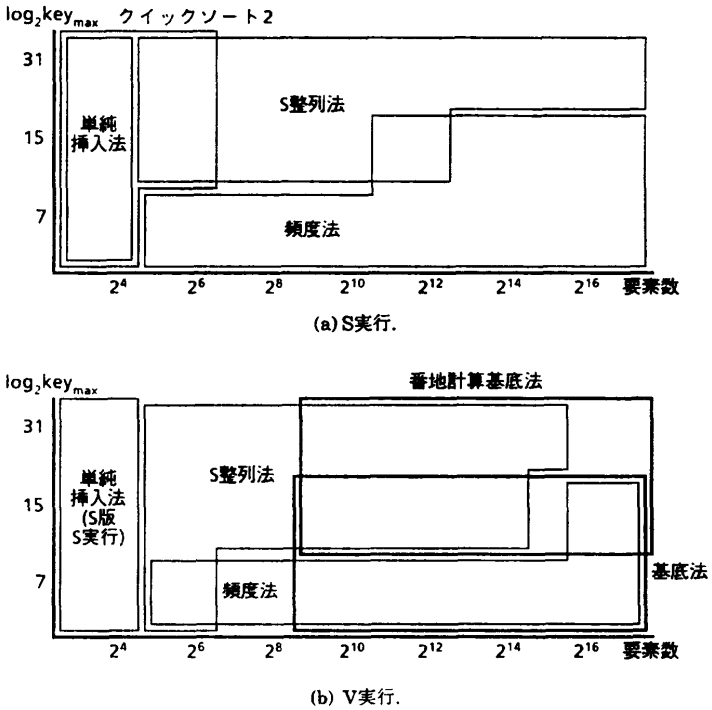


図1 キーの最大値×要素数の領域における最高速なアルゴリズム
Fig. 1 Fastest algorithm in the maximum key-value×element-number-domain.

挿入法よりも奇偶置換法のほうが優れていることが分かる.

5. むすび

ベクトル計算機向きのソーティング手法を提案した. また実際にベクトル計算機上にプログラムを実現し, 性能評価を行った. ソーティング・アルゴリズムとベクトル計算機との整合性は様々であり, 与えられた要素数, キー値のとる範囲の大きさに対して最も高速なアルゴリズムは, ベクトル計算機上では交代する (**図1**参照). スカラ計算機上で最も高速である頻度法, S整列法は部分的ではあるがベクトル処理可能で, ベクトル計算機上でも高速なアルゴリズムとなる. 基底法, および本論文で新たに提案した番地計算基底法は, スカラ計算機上では頻度法やS整列法に劣るが, ほぼ全過程がベクトル処理できるため, ベクトル計算機上では逆にこれより高速となる場合がある. また, これらのソーティング手法は比較と交換のみによるものに比べて格段に高速となる. さらに, 本論文ではベクトル計算機向きの局所ソート手法として, 単純挿入法に代わって奇偶置換法を用いることを提案したが, その有効性は実験結果からも確認することができた. 本論文の結果は, ベクトル計算機上でソーティングを行う場合のアルゴリズムの選択に, 一つの指針を与えるものと考えられる.

謝辞 本論文のソーティング・アルゴリズムのプログラム作成に御協力いただいた, 葉山悟氏に感謝いたします. また御討論いただいた本学津田孝夫教授, 安浦寛人助教授, 平石裕実助教授, および津田研究室, 矢島研究室の諸氏に感謝いたします.

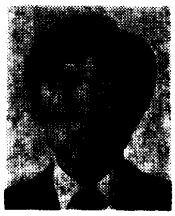
参考文献

1) 速さを競うスーパーコンピュータ, 日経エレクトロニクス, 1983年4月11日号 (No. 314), pp. 105-184 (1983).
2) スーパーコンピュータ SX システム, 日経エレクトロニクス, 1984年11月19日号 (No. 356), pp. 237-271 (1984).
3) 石浦菜岐佐, 安浦寛人, 矢島脩三: ベクトル計算機による高速論理シミュレーション, 情報処理学会論文誌, Vol. 27, No. 5, pp. 510-517 (1986).
4) Knuth, D. E.: *The Art of Computer Programming: Vol. 3/Sorting and Searching*, Addison-Wesley, Reading, MA (1973).
5) 大野義夫, 島田規人, 渋谷政昭: 「一様」に分布する $O(n)$ のソート, 第28回情報処理学会全国大会論文集, 5Q-8, pp. 15-16 (1984).
6) Bitton, D., Dewitt, D. J., Hsiao, D. K. and Mennon, J.: A Taxonomy of Parallel Sorting, *ACM Comput. Surv.*, Vol. 16, No. 3, pp. 287-318 (1984).
7) Rönsch, W. and Strauss, H.: Timing Results of Some Internal Sorting Algorithms on Vector Computers, *Parallel Computing 4*, pp. 49-61 (1987).
8) 石浦菜岐佐, 高木直史, 矢島脩三: ベクトル計算機向きソーティング手法, 信学技法, COMP 86-88, pp. 79-85 (1987).
9) Brock, H. K., Brooks, B. J. and Sullivan, F.: Diamond: A Sorting Method for Vector Machines, *BIT*, Vol. 21, pp. 142-152 (1981).

(昭和62年8月19日受付)
(昭和63年2月10日採録)

石浦菜岐佐 (正会員)

昭和36年生. 昭和59年京都大学工学部情報工学科卒業. 昭和61年同大学院修士課程修了. 昭和62年1月より京都大学工学部助手. 論理回路のCAD/DAの研究に従事. 電子情報通信学会会員.



高木 直史 (正会員)

昭和34年生. 昭和63年京都大学工学部情報工学科卒業. 昭和58年同大学院修士課程修了. 昭和59年4月より京都大学工学部助手. 工学博士. VLSI向きハードウェアアルゴリズム, 算術演算回路, 論理設計用CAD/DA等の研究に従事. IEEE, 電子情報通信学会各会員.

矢島 脩三 (正会員)

昭和8年生. 昭和31年京都大学工学部電気工学科卒業. 同大学院博士課程修了. 工学博士. 昭和36年より京大工学部に勤務, 昭和46年情報工学科教授. 昭和35年京大第一号計算機KDC-1を設計稼動, 以来, 計算機, 論理設計, オートマトン等の研究教育に従事, 著書は「電子計算機の機能と構造」(岩波, 57年) 等, 本学会元常務理事, 元会誌編集委員 (地方), 元JIP編集委員. 電子情報通信学会元評議員およびオートマトンと言語研専元委員長, North-Holland出版IPL編集委員, IEEE Senior Member.