ベクトル計算機上でのソーティング手法

7C-4

石浦 菜岐佐 葉山 悟 高木 直史 安浦 寛人 矢島 脩三

京都大学工学部情報工学科

1. はじめに

ソーティングは種々のデータ処理における基本操作として頻繁に用いられている。一方、近年の大規模計算の処理要求を満たすべく開発されたベクトル計算機は、数値計算のみならず広範な応用を可能としており[2][3]、今後様々な分野で益々盛んに利用されるものと思われる。このような状況において、ソーティングの効率良いベクトル処理手法の開発は、重要であると考えられる。我々は、いくつかの内部ソーティグ・アルゴリズムに対するベクトル化手法を考え、実際にFACOM VP-200上でその性能評価を行った。これより、1)いくつかのアルゴリズムについては、ベクトル処理により10倍から50倍近い高速化が達成される、2)与えられた要素数、キー長に対する最も効率の良いアルゴリズムは、ベクトル処理の導入により交代しうる、等の結果を得た。

2. ソーティング・アルゴリズムのベクトル化

本稿では10種類のソーティング・アルゴリズムについて、そのベクトル化を考察した。以下、各アルゴリズムのベクトル化手法について述べる。尚、アルゴリズムの詳細については文献[4]を参照されたい。

(1) 単純選択法(selection sort)

キー最小の要素の探索を、ベクトル・マクロ命令を用いてベクトル化する。

(2) 単純挿入法(insertion sort)

挿入位置を、ベクトル比較演算、総和演算により求める。

(3) 計数法(counting sort)

カウント部はベクトル比較演算、総和演算により、並べ換えはリスト・ベクトル・アクセスによりベクトル化する。

(4) バブル・ソート(bubble sort)

ベクトル化は困難であるため、その並列版である奇遇置換法(odd-even transposition sort)をベクトル化した。

(5) クイック・ソート(quick sort)

与えられた要素のリストを、適当に選んだ要素よりキーが大きいもの、小さいもの、等しいものの3つに分割するという操作を3回のベクトル収集命令により処理する。

(6) 番地計算法(address calculation sort)

番地計算を行う部分はベクトル化できるが、挿入と衝突の処理はベクトル化できない。局所ソーティングには奇遇置換法をベクトル化したものを用いている。

(7) 頻度法(distribution counting sort)

カウント部はベクトル化できない。並べ換えの部分のみリスト・ベクトル・アクセスでベクトル化できる。

(8) S整列法[5]

番地計算により求めた番地をキーとして頻度法で大局的な並べ換えを行った後、局所ソーティングを行う。番地計算と並べ換えの部分のみベクトル化できる。局所ソーティングには奇遇置換法をベクトル化したものを用いている。

(9) 基底法(radix sort)

基底を2とすれば、ベクトル論理演算命令と収集命令によりベクトル化できる。

(10) 番地計算基底法

番地計算により求めた番地をキーとして基底法で大局的な並べ換えを行った後、局所ソーティングを行う。局所ソーティングには奇遇置換法を用いている。処理はすべてベクトル化可能である。

この他にも、2分選択法、ヒープ・ソート等のアルゴリズムも知られているが、ベクトル化が困難と考えられたため、今回は扱っていない。

3. 性能評価

前章のソーティング手法を、京都大学大型計算機センターのFACOM VP-200上に実現し、処理速度の評価を行った。キーは4バイトの整数とし、その昇順に、キーと4バイトのデータ・ポインタを並べ換えるものとした。また、キーの分布は一様分布を仮定している。

結果を表1に示す。実行時間は比較のため、ベクトル化を意識しないコーディング(S版と呼ぶ)をスカラ命令だけで実行した場合(S実行と呼ぶ)と、ベクトル化コーディング(V版と呼ぶ)をベクトル命令を用いて実行した場合(V実行と呼ぶ)の二通りについて測定した。キーは$0\sim2^{31}-1$までの一様乱数を合同乗算法により発生させたものを用いている。

処理効率をあげるためには、キーやポインタを格納する以外に、余分な記憶領域が必要となることがある。「記憶量」の欄は全体の使用記憶量が、キーとポインタの占める記憶量の何倍になっているかを示している。前章のアルゴリズムのうち頻度法は、この大きさのキーに対しては記憶量の制約から実行できなかったので、記載していない。

ベクトル処理の効果の大きいアルゴリズムは、単純選択法、単純挿入法、計数法、バブル・ソート、

Sorting on a Vector Processor

Nagisa Ishiura, Satoru Hayama, Naofumi Takagi,
Hiroto Yasuura and Shuzo Yajima
Kyoto University

基底法、番地計算基底法である。大きな要素数に対して効率の良いアルゴリズムは、スカラ実行の場合はS整列法、番地計算法であるが、ベクトル実行の場合には加速率の違いから、S整列法、番地計算基底法が高速となる。

次に処理時間のキー長への依存を調べるため、キーをそれぞれ$0\sim2^{15}-1$、$0\sim2^{7}-1$、$0\sim2^{3}-1$の一様乱数で与えて同様の実験を行った。図1は、与えられた要素数、キー長に対して高速なアルゴリズムを示している。図1(S)はS実行のみの場合、図1(V)はV実行も含めた場合を示している。単純挿入法、頻度法、S整列法はS実行においてもV実行においても同様の領域で高速となっている。これに対し、S実行において、要素数の多い領域で高速であった番地計算法は、ベクトル処理による加速率の差から、V実行では基底法、番地計算基底法にその座を譲っている。

尚、同様の実験を東京大学大型計算機センターのHITAC S-810/20上でも行ったが、ほぼ同様の結果が得られている。

4. 考察

単純選択法を始めとする、処理時間$O(n^2)$のアルゴリズム(但しnは要素数)は、ベクトル化による加速率という点では良い結果をもたらす。しかし、1)要素数が小さいときには、ベクトル処理の効果は期待できず、2)要素数が大きいときには、処理量が$O(n\log n)$やの$O(n)$のアルゴリズムのほうが有利となる等の理由により、実際にベクトル処理が有効になるのは、奇遇置換法のように局所ソーティングに用いる場合に限られる。処理時間が$O(n\log n)$のアルゴリズムについては、ベクトル計算機との整合性が良いものがみつかっていない。処理時間$O(n)$のアルゴリズムでは、基底法、番地計算基底法、頻度法、S整列法等、ベクトル処理に向いたものがあり、要素数が大きいソーティングで威力を発揮すると考えられる。特に基底法、番地計算基底法は、全処理がベクトル化できるため、パイプラインの処理能力向上により、更に高速化が期待できる。

5. むすび

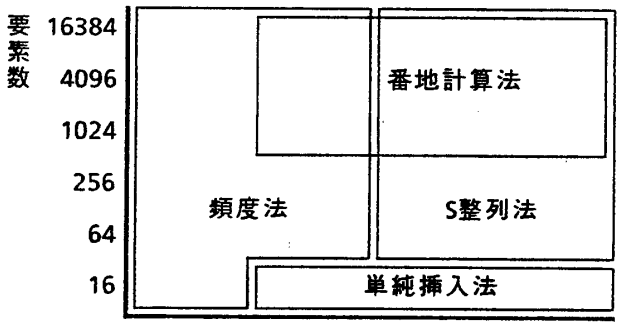
いくつかのソーティグ･アルゴリズムについてベクトル化手法を考え、その性能評価を行った。本稿で述べた以外のアルゴリズムについても検討を行ってゆきたい。最後に、御討論頂いた矢島研究室、津田研究室の諸氏に感謝いたします。

参考文献

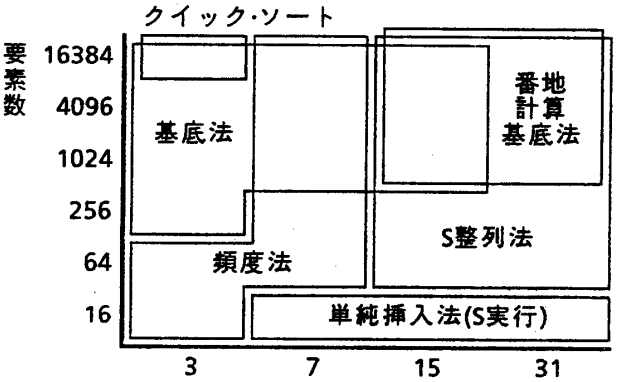
[1] 日経エレクトロニクス, 1983 4-11, pp.105~184.
[2] 石浦他: 情処論文27-5, pp. 510~517, (1986).
[3] 加賀谷他: 第33回情処全大6X-2, (1986).
[4] D. E. Knuth: "The Art of Computer Programming: volume 3", Addison-Wesley, (1973).
[5] 大野他: 第28回情処全大5Q-8, (1984).

表1 実行時間の比較

	記憶量 [倍]		実行時間 [μsec] 上段:要素数 64 中段:要素数 1024 下段:要素数 16384		
	S版	V版	S実行	V実行	S/V
単純選択法	1.0	1.0	656 143,000 36,506,000	327 7,100 784,000	2.0 20.1 46.6
単純挿入法	1.0	1.0	494 99,200 26,292,000	480 14,500 1,932,000	1.0 6.8 13.6
計数法	1.5	2.0	748 167,000 48,723,000	449 16,700 3,152,000	15.5 10.0 1.7
バブルソート	1.0	1.5	1,410 367,000 98,753,000	687 34,600 7,786,000	2.1 10.6 12.7
クイックソート	1.0	2.0	559 13,600 284,000	814 15,200 247,000	0.7 0.9 1.1
番地計算法	2.5	2.5	397 6,350 114,000	256 2,560 52,800	1.6 2.5 2.2
S整列法	2.5	2.5	303 5,460 114,000	118 992 17,000	2.6 5.5 6.7
基底法	1.5	1.5	1,530 23,000 431,000	440 2,300 36,400	3.5 10.0 11.8
番地計算基底法	2.0	2.0	552 12,900 311,000	189 1,020 20,200	2.9 12.6 15.4



(S) S実行のみ考慮した場合　キー長



(V) V実行も考慮した場合　キー長

図1 要素数、キー長とアルゴリズム