共有部分がある複数データのベクトル処理方法

4M−3

金田　泰、菅谷 正弘　(日立製作所中央研究所)

1. まえがき

金田ら [1] は、共有部分がない複数のリストなどの可変長データを、それらへのポインタやインデクスからなるベクトル(インデクス・ベクトル)をつかってベクトル処理する方法をしめした。この方法では、インデクス・ベクトルの各要素からさされるデータに共有部分があっても、データの更新をおこなわないかぎりはただしく処理できる。しかし、複数の要素が同一のデータを共有しそれを更新する場合には、本来は逐次実行しなければならない処理を並列実行することになるので結果が不正になり、したがってベクトル処理できなかった。

この報告では、共有部分がある複数のデータを、共有部分だけ逐次的に処理し他の部分をベクトル処理する方法をしめすとともに、この方法の適用例をしめす。

2. 共有部分があるデータのベクトル処理の問題点

共有部分があるデータをベクトル処理しようとする場合の問題点を、ハッシングを例として図1にしめす。

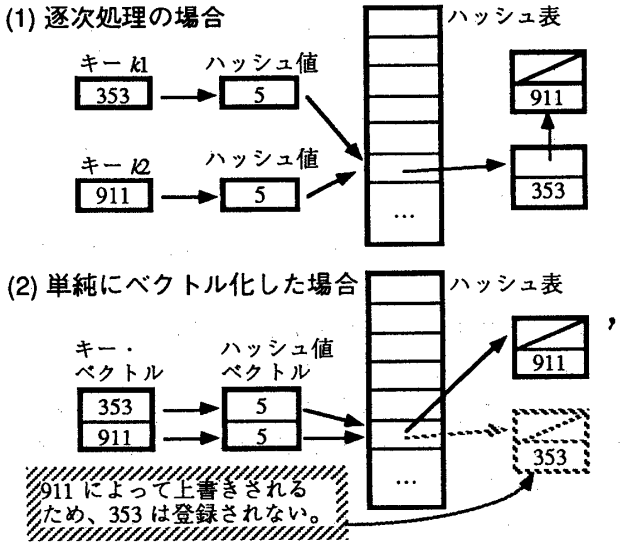


図1 ハッシュ表登録のベクトル処理の問題点

3. 共有部分があるデータのベクトル処理の原理

上記の問題点を解決するには、処理すべきデータを図2に例示したように並列処理可能な集合に分割して、それぞれをベクトル処理すればよい。並列処理可能かどうかを判定するには、つぎのようにすればよい。図3にしめすように、各データにあらかじめ作業領域を付加しておく。ベクトルのインデクス(またはベクトルのどの要素に対応するかをしめす他の識別子)を作業領域にかきこんだのちただちによみだし、かきこんだ値がそのままよみだされたかどうかを判定する。その結果をマスク・ベクトルとして表現し、それ以降の処理をこのマスクにしたがって実行する。インデクスがかきこまれていなかった要素については、そのあとで処理すればよい。

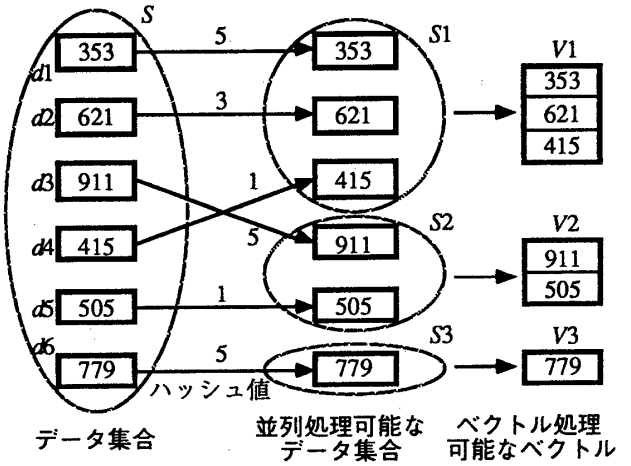


図2 並列処理可能集合への分割 (ハッシングの場合)

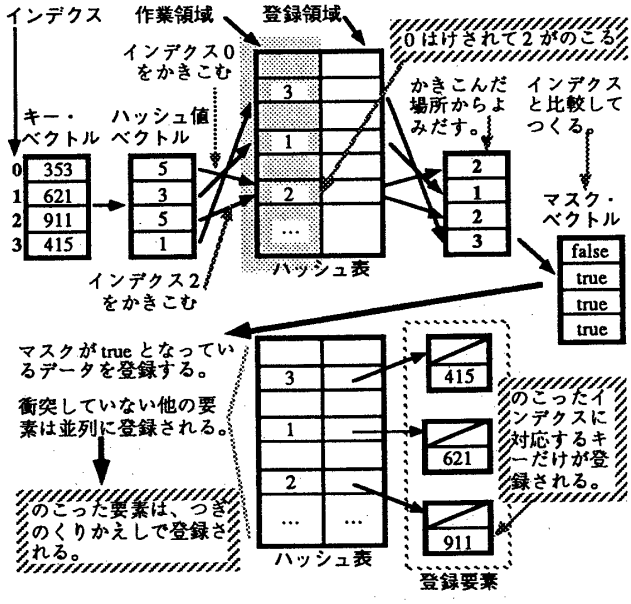


図3 具体的なベクトル処理方法 (ハッシングの場合)

A Vector Processing Method of Multiple Data with Sharing
Yasusi KANADA
HITACHI, Ltd.

4. 適用例

4.1 ハッシング

ハッシングのベクトル処理方法の概略は図3にしめしたが、そのアルゴリズムは図4のようになる。このアルゴリズムでは、同一のキー値が存在しないことを仮定して、識別子として図3におけるインデクスのかわりにキー値じたいを使用することで、ハッシュ表へのかきこみ回数をへらしている(同一のキー値が存在する場合は、図3の方法をそのまま適用すればよい)。金田[2]ではこのアルゴリズムをよりくわしく説明している。

```
入力: ハッシュ表 table (すでにキーが登録されていてもよい)。
      ハッシュすべきキー集合 key[1 : n]。
出力: 入力したキー集合が登録ずみのハッシュ表 table。

/* ハッシュ値の計算と仮登録 */
hashed_value[1 : n] := hash(key[1 : n]);    /* ハッシュ値を計算する */
where table[hashed_value[1 : n]] = 未登録 do
   table[hashed_value[1 : n]] := key[1 : n];    /* 未登録部分に登録 */
end where;

for i in 1 .. ハッシュ表サイズ loop    /* テーブルあふれは無視 */

/* 未登録判定と未登録要素の収集 (圧縮) */
   nrest := 未登録要素の数;
      /* key[i] が未登録かどうかは, table[hashed_value[i]]        */
      /* の値によって判定する                                     */
   hashed_value[1 : nrest] :=
      hashed_value[1 : n] の各要素のうち未登録のもの;
   key[1 : nrest] := key[1 : n] の各要素のうち未登録のもの;
      /* 未登録要素のハッシュ値とキーとを, それぞれ,             */
      /* よりみじかい配列につめる                                 */

/* 登録終了判定 */
   if nrest = 0 then exit loop;                  /* ループを脱出 */
   n := nrest;                                   /* n の値を更新する */

/* つぎの添字値の計算と仮登録 */
   hashed_value[1 : n] :=
      (hashed_value[1 : n] + 1) mod ハッシュ表サイズ;
   where table[hashed_value[1 : n]] = 未登録 do
      table[hashed_value[1 : n]] := key[1 : n];  /* 未登録部分に登録 */
   end where;

end loop;
```

図4 ベクトル・ハッシングのアルゴリズム

4.2 番地計算ソートのベクトル化

番地計算ソート (address calculation sort) [4] は、ソートすべき数をハッシングと類似の方法でテーブルに登録する処理をふくむ (ただし、すでに登録した要素をテーブルに挿入する際に既登録要素をシフトする必要が生じる点がことなる)。この処理は従来の方法ではベクトル化できなかった[3]が、4.1と同様の方法を使用することによってベクトル化可能になる。このベクトル処理アルゴリズムは金田[2]において説明している。

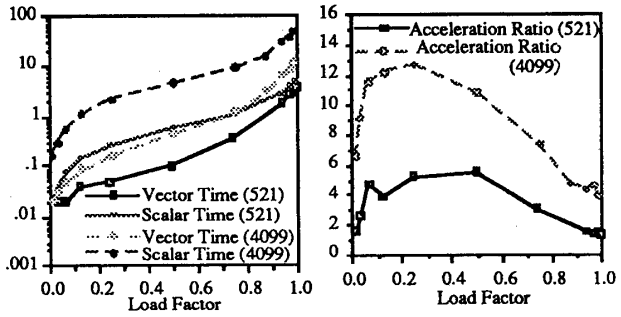
5. 結果

5.1 ハッシュ表への登録

オープン・ハッシングにおけるサイズが521と4099の空のハッシュ表への整数の登録性能例を図5にしめす。横軸は登録後のロード・ファクタ (ハッシュ表のエントリのうち登録されたものの割合) をしめす。

5.2 番地計算ソート

一様分布する $[0, 2^{16})$ の整数値の番地計算ソートの性能を測定した結果を表1にしめす。結果の詳細については金田[2]を参照されたい。金田[2]には、同様の方法でベクトル化処理できる頻度ソートのベクトル処理性能を測定した結果もしめしている。



(1) 総登録時間　　(2) 加速率

図5 空ハッシュ表への登録性能 (S-810)

表1 番地計算ソートの実行時間 (S-810, 単位 *mS*)

要素数	スカラ処理	ベクトル処理	加速率
2^6	289	110	2.89
2^{10}	4,286	560	7.65
2^{14}	66,955	5,215	12.84

6. むすび

この報告では、共有部分がある複数データをベクトル処理する方法をしめし、ハッシュ表への登録と番地計算ソートに適用して最高約13倍の加速率をえた。この報告ではしめさなかったが、この方法はリストや木などのより複雑なデータ構造に対しても適用可能であり、1つの2分木への複数登録などにおいても高い加速率がえられている。したがって、この技法によってベクトル処理可能なプログラムの範囲が拡大されることが期待できる。

参考文献

[1] 金田 泰, 菅谷 正弘: プログラム変換にもとづくリストのベクトル処理方法とそのエイト・クウィーン問題への適用, 情報処理学会論文誌, Vol. 30, No. 7, 1989.

[2] Kanada, Y.: A Vectorization Technique of Hashing and its Application to Several Sorting Algorithms, *PARBASE '90*.

[3] 石浦 菜岐佐, 高木 直史, 矢島 脩三: ベクトル計算機上でのソーティング, 情報処理学会論文誌, Vol. 29, No. 4, 1988.

[4] Knuth, D.: Sorting and Searching, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, 1973.