Analysis of Parallel Hashing Algorithms with Key Deletion

TETSUO IDA* and EIICHI GOTO*·**

Three parallel hashing schemes are presented together with the analysis of their efficiencies based on a statistical theory. As assessment parameters, probe numbers for parallel hash algorithms in these schemes such as for pure search and for key insertion are calculated in the worst statistical equilibrium. The results show improvement of the efficiencies of parallel hash algorithms over those of conventional sequential hashing. When parallel hash algorithms are implemented in hardware with multi-bank memory, they can be executed in a time comparable to single indirect addressing.

Key words and Phrases. Hashing, Parallel Hashing Algorithms, Key Deletion, Hashing Hardware

1. Introduction

In this paper, we describe an analysis of three parallel hashing schemes which can handle key deletion without key relocation. Our motivation, in developing these schemes, is for hardware implementation of hashing in view of accumulation and recent development of many algorithms which rely heavily on hashing. These algorithms include traditional application of hashing in associative processing [1] as well as address mapping [2] and symbolic and algebraic manipulation [3, 4, 5]. In the previous paper [6], we proposed a parallel hash hardware which can handle key deletion with three hashing schemes to be implemented on it. Performance analysis of the three schemes based on a statistical theory is treated in this paper; it serves for a preliminary performance evaluation in a planned implementation of the hardware proposed in [6].

The three parallel hashing schemes are based on the parallelism of hash table accesses and on open addressing for resolution of key collisions. The hash table is realized by multi-bank (J banks, $J \geq 1$) memory, and the J keys stored in different banks are read out simultaneously. The schemes differ from each other in the generation of hash addresses and in the way the read-out keys are scanned in search for key match or emptiness. Analysis of parallel hashing schemes without key deletion is made in [7]. However, key deletion is indispensable for advanced applications, such as those mentioned above, to be really viable.

In section 2, a problem of key deletion is discussed and in section 3, the three schemes and associated algorithms are presented. As an application of the analysis technique presented here, algorithms "Insert a key if it is not present in the hash table" in the three schemes are described and the performance is analyzed in section 5.

Although part of the material in sections 3 and 4 are covered in [6], they are rewritten here in a form convenient for mathematical analysis.

2. Key deletion

Key deletion in the open addressing scheme is a cumbersome problem for these reasons;

(i) A cell where a key is to be removed can not simply be emptied since there may possibly be keys that are in collision with the key.

(ii) When data structures subjected to hashing are intricately referenced by pointers, efficient relocation of keys would be very difficult.

A deletion algorithm by key relocation is described in Knuth [8, Algorithm R]. It works only for the algorithms of linear hashing, which is one of the slowest hash algorithms described in Knuth, due to 'clustering' of keys.

A solution to this problem is to introduce collision counters into the hash table. With collision counters, we can determine whether a particular hash sequence is terminating or not. This implies that the collision counters serve two purposes; of preventing erroneous termination of a hash probe sequence even if an empty cell is encountered, and of decreasing the probe number for an unsuccessful search. An obvious penalty for this solution is extra storage requirement, i.e. $M \log_2 (MJ-J)$ bits for Scheme 1 and $MJ \log_2 MJ$ bits for Schemes 2 and 3, as is later explained.

An alternative to this method is to allow only a garbage collector (GBC) to perform key deletion: by this garbage collector scheme, one-bit collision tags may be used in place of collision counters since collision counting can be done by GBC (The collision tag table in this scheme is set up temporarily as opposed to 'per-

*Institute of Physical and Chemical Research 2-1, Hirosawa, Wako-shi, Saitama 351, Japan.

**Department of Information Science University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113, Japan.

manently' in the hardware counter scheme). GBC is invoked when the load factor α (i.e. $\alpha = N/MJ$ where N is the total number of keys in the hash table) of the hash table exceeds the permissible maximum load factor β. The garbage collection proceeds as follows: first, all the collision tags of the emptry cells are reset; then the hash probe sequence of each key is traced by setting the collision tags of the empty cells probed. If α still exceeds β after the garbage collection, the system terminates the job immediately. If the GBC scheme is not used, the system terminates the job when α exceeds β.

3. The Parallel Hasing Schemes with Deletion

For each scheme below, three basic hashing algorithms are discussed; 'S', search for a key; 'D', deletion of a key that is known to be present in the hash table (we call it an 'existing key' hereafter); and 'I', insertion of a new key. Composite algorithms (such as "I*, insert if search is unsuccessful" and "D*, delete if search is successful") will be discussed in section 5.

Fig. 1 shows, schematically, a data structure representing the hash table. This data structure is common to all the schemes.

3.1 Scheme 1

The same hash sequence $h_i(k)$, $i=1, 2, \ldots$ for a given key k is used for all J banks. Thus, in one hash probing, $K[h_i(k), 1]$, $K[h_i(k), 2], \ldots,$ $K[h_i(k), J]$ are read out simultaneously. Collision counters $C[1: M]$ are used in Scheme 1, where each element $C[i]$ is common to a row $K[i, j], j=1, 2, \ldots, J$. Note Scheme 1 is the same as an open addressing hashing scheme using a bucket of size J, except that all the cells of the bucket are accessed in parallel, where a row $K[i, j]$ $j=1, 2, \ldots, J$ is identified as a bucket i. We use in the subsequent analysis the term 'bucket' in place of 'a row of the hash table.'

Algorithm 1S: This algorithm searches for a given key k and returns to R the address of the cell where key k is stored. If the search is unsuccessful, the value of R is the reserved word for emptiness ϕ.

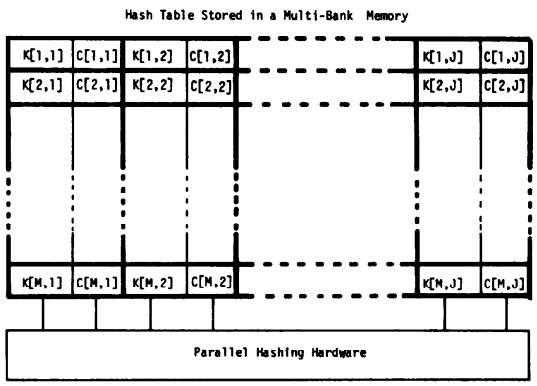


Fig. 1 Organization of a hash table for parallel hash schemes.

(S1.1) [Initialization] Set $i \leftarrow 1$.
(S1.2) [Hash address generation] Set $h \leftarrow h_i(k)$.
(S1.3) [Parallel hash table access]
Read $K[h, 1], \ldots, K[h, j]$ simultaneously.
(S1.4) [Check for match] If $K[h, j]=k$ for some j
set $R \leftarrow$address of $K[h, j]$, and terminate the algorithm.
If $C[h]>0$, set $i \leftarrow i+1$ and go to (S1.2); otherwise, set $R \leftarrow \phi$ and terminate the algorithm.

Algorithm I1: This algorithm inserts a new key k and returns to R the address of the cell where key k is stored.

(I1.1) . . (I1.3) are the same as (S1.1) . . (S1.3) of Algorithm 1S.
(I1.4) [Check for emptiness] If $K[h, j]=\phi$ for some j, set $K[h, j] \leftarrow k$, $R \leftarrow$address of $K[h, j]$ and terminate the algorithm.
Otherwise, set $C[h] \leftarrow C[h]+1$ and $i \leftarrow i+1$, and go to (I1.2).

Algorithm 1D: This algorithm deletes an existing key in the hash table.

(D1.1) . . (D1.3) are the same as (S1.1) . . (S1.3) of Algorithm 1S.
(D1.4) [Check for match] If $K[h, j]=k$ for some j, set $K[h, j] \leftarrow \phi$ and terminate the algorithm.
Otherwise, set $C[h] \leftarrow C[h]-1$ and $i \leftarrow i+1$, and go to (D1.2).

3.2 Scheme 2

J independent hash sequences $h_i^{(J)}$, $i=1, 2, \ldots$ are used one for each bank j, $1 \leq j \leq J$. Thus, in one hash probing $K[h_i^{(j_1)}(k), j_1]$, $K[h_i^{(j_2)}(k), j_2], \ldots,$ $K[h_i^{(j_J)}(k), j_J]$ are read out simultaneously; and this ordering is used for inspection of the key, where $(j_1, \ldots, j_J)$ is a permutation of $(1, \ldots, J)$ chosen at random depending on key k and index i.

In Algorithm 2I, 2S, 2D, 3I, 3S and 3D below, $C[1: M, 1: J]$ are collision counters associated with each cell.

Algorithm 2S: This algorithm performs the same function as Algorithm 1S.

(S2.1) [Initialization] Set $i \leftarrow 1$.
(S2.2) [Hash address generation] Set $h[n] \leftarrow h_i^{(n)}(k)$ for $n=1, \ldots, J$ in parallel.
(S2.3) [Parallel hash table access] Read $K[h[1], 1], \ldots, K[h[J], J]$ simultaneously.
(S2.4) [Check for match] If $K[h[j], j]=k$ for some j, set $R \leftarrow$address of $K[h[j], j]$, and terminate the algorithm.
If $C[h[j_1], j_1]>0 \bigwedge \ldots \bigwedge C[h[j_J], j_J]>0$,
set $i \leftarrow i+1$ and go to (S2.2); otherwise,
set $R \leftarrow \phi$ and terminate the algorithm.

Algorithm 2I: This algorithm performs the same function as Algorithm 1I.

(I2.1) . . (I2.3) are the same as (S2.1) . . (S2.3) of Algorithm 2S.

(I2.4) [Check for emptiness]
For $n=j_1, \ldots, j_J$ do, in this order;
if $K[h[n], n]=\phi$, set $K[h[n], n] \leftarrow k$, $R \leftarrow$address of $K[h[n], n]$ and terminate the algorithm, otherwise, set $C[h[n], n] \leftarrow C[h[n], n]+1$.
Set $i \leftarrow i+1$ and go to (I2.2).

Algorithm 2D: This algorithm performs the same function as Algorithm 1D.

(D2.1) . . (D2.3) are the same as (S2.1) . . (S2.3) of Algorithm 2S

(D2.4) [Check for match]
For $n=j_1, \ldots, j_J$ do, in this order;
if $K[h[n], n]=k$, set $K[h[n], n] \leftarrow \phi$
and terminate the algorithm,
otherwise, set $C[h[n], n] \leftarrow C[h[n], n]-1$.
Set $i \leftarrow i+1$ and go to (D2.2).

3.3 Scheme 3

J independent hash sequences $h_i^{(J)}(k)$, $i=1, 2, \ldots$ are used for J banks as in Scheme 2. In one hash probing, $K[h_i^{(1)}(k), 1]$, $K[h_i^{(2)}(k), 2], \ldots, K[h_i^{(J)}(k), J]$ are read out simultaneously; and a fixed ordering is used for insertion of the key. Therefore, Algorithm 3I, 3S and 3D are the same as corresponding ones of Scheme 2 except that the scanning order in steps (I2.4), (S2.4) and (D2.4) is replaced by one specified by $j_1=1, j_2=2, \ldots, j_J=J$ regardless of key k, and hash sequence index i.

4. Analysis of the Hashing Schemes

We shall assume in the following analysis (of Scheme 1, 2 and 3) that hash sequences are random and that keys in the hash table are delected at random.

Among various probe numbers pertinent to a specific application, the probe number for successful search, PS, and the probe number for unsuccessful search, PU, are important measures of the efficiency since other probe numbers for composite algorithms (such as those mentioned in section 3) are determined or bounded by these two quantities. In this section, we shall calculate PS and PU of the worst statistical equilibrium; that is, deletions and insertions take place alternately keeping the load factor α of the hash table very close to the permissible maximum β. In mathematical terms, probe numbers are functionals of loading history function $\alpha(t)$, where t (for time) is an integer which is incremented by one each time either a deletion or an insertion takes place. We conjecture that in random hashing

$$PS[\alpha(t)] \leq PS[\beta]=PS, \quad \alpha(t) \leq \beta < 1 \tag{1.1}$$

$$PU[\alpha(t)] \leq PU[\beta]=PU \tag{1.2}$$

$$PI^*[\alpha(t)] \leq PI^*[\beta]=PI^* \tag{1.3}$$

The conjecture has been confirmed by simulations; although in a simple case, the relation (1.1) in uniform hashing is proved mathematically [9]. That is, $PS[\alpha(t)] \leq PS[\beta'(t)]$ where $\beta'(t)$ is defined as follows: $\beta'(t)=t$ for $0 \leq t \leq N$, $\beta'(N+t)=N$ for even $t>0$ and $\beta'(N+t)=N+1$ for odd $t>0$; and N denotes a sufficiently long time period so that the statistical equilibrium is achieved. Hence, we call the worst statistical equilibrium 'the worst cycle' hereafter. Note that in the worst cycle, PS is equal to PI, i.e. a probe number for insertion.

4.1 Analysis of Scheme 1

Let B_i, b_i and p_i, respectively, denote a state of a bucket consisting of i occupied cells, the number of buckets B_i and the fraction of buckets B_i *i.e.* $p_i=b_i/M$. Under the assumption of the randomness, we can calculate the distribution of B_i's in the worst cycle analytically. In the following derivations of equations, we assume that $M \gg 1$ and regard α as a continuous variable (continuum approximation to a discrete problem). It is observed (and indeed proved for the probe number without key deletion) that the continuum approximation gives pessimistic (larger) values of probe numbers (than those of the exact discrete treatments).

When a new key is inserted in the hash table of load factor α,

$$d^+p_0/d\alpha = -p_0J/(1-p_J) \tag{1.4}$$

$$d^+p_j/d\alpha = (p_{j-1}-p_j)J/(1-p_J) \quad 1 \leq j < J \tag{1.5}$$

$$d^+p_J/d\alpha = p_{J-1}J/(1-p_J) \tag{1.6}$$

and when a key is deleted,

$$d^-p_0/d\alpha = p_1/\alpha \tag{1.7}$$

$$d^-p_j/d\alpha = ((j+1)p_{j+1}-jp_j)/\alpha \quad 1 \leq j < J \tag{1.8}$$

$$d^-p_J/d\alpha = -p_JJ/\alpha \tag{1.9}$$

where we use the notation d^+ and d^- to distinguish between the increment caused by insertion and that by deletion.

Equation (1.4) is derived from the relationship; the decrement of b_0 due to insertion of a key (*i.e.* $-d^+b_0/d(MJ\alpha)$) is equal to the ratio of b_0 to the total number of buckets having one or more empty cells (*i.e.* $b_0/M(1-p_J)$). The increment of b_j $(1 \leq j < J)$ is caused by the insertion in buckets B_{j-1}, and the decrement by the insertion in buckets B_j. Hence, eq. (1.5) follows. Since no state transition of a bucket from B_J in case of the insertion is possible, eq. (1.6) follows.

In case of the deletion of a key, the decrement of b_J (*i.e.* $-d^-b_J/d(MJ\alpha)$) is equal to the ratio of keys in buckets B_J (*i.e.* Jb_J) to the total number of keys in the hash table. Hence, eq. (1.9) follows. The increment of b_j $(1 \leq j < J)$ is caused by the deletion of a key that is one of the $(j+1)b_{j+1}$ keys in buckets B_{j+1}, and the decrement by the deletion of a key that is one of the jb_j keys in buckets B_j. Hence, eq. (1.8) follows. Equation (1.7) is similarly derived.

For the statistical equilibrium in the worst cycle,

$$d^+p_j/d\alpha + d^-p_j/d\alpha = 0 \quad \text{at } \alpha=\beta \quad \text{and for } 0 \leq j \leq J. \tag{1.10}$$

Using the condition $\sum_{i=0}^{J} p_i = 1$ and the relations (1.10) we obtain:

$$p_j = q^j/(j!\ texp(J, q)) \quad 0 \le j < J \tag{1.11}$$

$$\beta = (1/J)\left(\sum_{i=1}^{J} q^i/((i-1)!\ texp(J, q))\right) \tag{1.12}$$

where $q = J\beta/(1-p_J)$, and $texp\,(J, q)$ is a truncated exponential function defined by $texp\,(J, q) = \sum_{i=0}^{J} q^i/i!$.

Given β, we can calculate p_i's. The value of P_J thus obtained is enough to obtain PS, which is given as:

$$PS = 1/(1-p_J) \tag{1.13}$$

Note term $(1-p_J)$, which is the probability of a bucket having at least one empty cell, is equal to the probability of a given key being probed in the worst cycle.

Probe number PU depends on collisions rather than on the states B_j of buckets, since an unsuccessful search for key k terminates only if a hash probe sequence $h_0(k)$, $h_1(k)$, ... terminates at a certain point $h_j(k)$. To determine PU numerically, we introduce a definition of a further concept and some related notations.

A collision number of a bucket is a number of keys in the hash table that have collided at the bucket.

Let $B_j^{(k)}$ $(j=0, 1, \ldots, J$ and $k=0, 1, \ldots, MJ-J)$ denote a substate of a bucket in which the bucket holds j empty cells and is with collision number k, and let $p_j^{(k)}$ be the probability of a bucket being in substate $B_j^{(k)}$.

Obviously,

$$p_j = \sum_{k=0}^{MJ-J} p_j^{(k)}. \tag{1.14}$$

For each k, change of $p_j^{(k)}$'s due to insertion of a new key is given by:

$$d^+ p_0^{(k)}/d\alpha = -p_0^{(k)} J/(1-p_J) \tag{1.15}$$

$$d^+ p_j^{(k)}/d\alpha = (p_{j-1}^{(k)} - p_j^{(k)})J/(1-p_J) \quad 1 \le j < J \tag{1.16}$$

$$d^+ p_J^{(k)}/d\alpha = (p_{J-1}^{(k)} + p_J^{(k-1)} - p_J^{(k)})J/(1-p_J) \tag{1.17}$$

and due to deletion of a key is by:

$$d^- p_j^{(k)}/d\alpha = ((j+1)p_{j+1}^{(k)} - jp_j^{(k)} + (k+1)p_j^{(k+1)} - kp_j^{(k)})/\alpha \quad 0 \le j \le J \tag{1.18}$$

with the convention $p_{J+1}^{(k)} \equiv 0$ and $p_J^{(-1)} \equiv 0$.

Equations (1.15) and (1.16) are obvious. In eq. (1.17), term $p_{J-1}^{(k)}$ represents the effect of insertion of a key into a bucket $B_{J-1}^{(k)}$; and the terms $p_J^{(k-1)}$ and $p_J^{(k)}$ represent the effects of collisions. Likewise, terms $(k+1)p_j^{(k+1)}$ and $kp_j^{(k)}$ in eq. (1.18) are introduced because of collisions. Since in the worst cycle $d^+ p_j^{(k)}/d\alpha + d^- p_j^{(k)}/d\alpha = 0$ at $\alpha = \beta$ for $0 \le j \le J$, we obtain

$$q(p_{j-1}^{(k)} - p_j^{(k)} + \delta_{kJ} p_J^{(k-1)}) + (j+1)p_{j+1}^{(k)} - jp_j^{(k)} + (k+1)p_j^{(k+1)} - kp_j^{(k)} = 0 \quad 0 \le j \le J \tag{1.19}$$

with the convention $p_{-1}^{(k)} \equiv 0$ and $q = J\beta/(1-p_J)$ as before.

Let $Q_j(\lambda) = \sum_{k=0}^{\infty} \lambda^k p_j^{(k)}$ $(j=0, 1, 2, \ldots, J)$ be the generating functions for these probability distributions.

Equations (1.19) are equivalent to

$$q(Q_{j-1}(\lambda) - Q_j(\lambda) - \delta_{jJ}\lambda Q_J(\lambda)) + (j+1)Q_{j+1}(\lambda) - jQ_j(\lambda) - (\lambda-1)\,dQ_j(\lambda)/d\lambda = 0 \tag{1.20}$$

Let $z = q(\lambda - 1)$ and we have

$$Q_j(\lambda) = p_j F_j(q; z) \tag{1.21}$$

and

$$F_j(q; z) = \sum_{n=0}^{\infty} f_j^{(n)} z^n \quad 0 \le j \le J$$

where $f_j^{(0)} = 1$, $0 \le j \le J$ because of condition (1.14).

Substituting eq. (1.21) into eq. (1.20), we obtain the following recurrence relations for the coefficients $f_j^{(n)}$ $(n=0, 1, \ldots)$:

$$\begin{aligned} f_1^{(n)} &= (q+n)f_0^{(n)}/q \\ qf_{j+1}^{(n)} &= (q+j+n)f_j^{(n)} - jf_{j-1}^{(n)} \quad 1 \le j \le J-1 \\ f_J^{(n)} &= (Jf_{J-1}^{(n)} + f_J^{(n-1)})/(J+n) \end{aligned} \tag{1.22}$$

Let y be the probability that a bucket being with collision number $k=0$. Then

$$y = \sum_{j=0}^{J} p_j^{(0)} = \sum_{j=0}^{J} p_j F_j(q; -q) \tag{1.23}$$

Since an unsuccessful search terminates only if a bucket being probed is with zero collision number

$$PU = 1/y. \tag{1.24}$$

It would be difficult to find a closed form of $Q_j(\lambda)$ except for $J=1$. In this case, F_0 and F_1 are the solutions of confluent hyper-geometric differential equations:

$$F_0(q; z) = {}_1F_1(q, q+2; z) \tag{1.25}$$

and

$$F_1(q; z) = {}_1F_1(q+1, q+2; z)$$

where

$${}_1F_1(x, y; z) = \sum_{i=0}^{\infty} \Gamma(x+i)\Gamma(y)z^i/(\Gamma(x)\Gamma(y+i)i!) \tag{1.26}$$

This result for $J=1$ agrees with the result obtained in [9] through a different approach based on the fact that the sequence $\{p_0^{(k)}; k=0, 1, \ldots.\}$ and $\{p_1^{(k)}; k=0, 1. \ldots.\}$ each form a Markov chain over the discrete time instances.

4.2 Analysis of Schemes 2 and 3

We first consider Scheme 3 since Scheme 2 is the special case of Scheme 3 (all α_i's are equal). Let α_j be the load factor of the j-th bank. Since increment of number of keys in the first bank due to insertion of a key (*i.e.* $dM\alpha_1/d(MJ\alpha)$) is equal to $(1-\alpha_1)/(1-\alpha_1 \ldots \alpha_J)$,

$$d^+\alpha_1/d\alpha = J(1-\alpha_1)/(1-\alpha_1 \ldots \alpha_J) \tag{2.1}$$

Similarly

$$d^+\alpha_j/d\alpha = J\alpha_1 \ldots \alpha_{j-1}(1-\alpha_j)/(1-\alpha_1 \ldots \alpha_J), \quad 2 \le j \le J \tag{2.2}$$

As for deletion, we have:

$$d^-\alpha_j/d\alpha = -\alpha_j/\alpha, \quad 1 \le j \le J \tag{2.3}$$

In the worst cycle,

$$d^+\alpha_j/d\alpha + d^-\alpha_j/d\alpha = 0 \quad \text{at } \alpha=\beta, \text{ and for } 1 \le j \le J. \tag{2.4}$$

Letting $\beta_i = \alpha_i$ for $\alpha = \beta$ we obtain the following;

$$\begin{aligned} u_1 &= J\beta/(1-\beta_1 \ldots \beta_J) \\ \beta_j &= u_j/(u_j+1) \quad 1 \le j \le J \\ u_j &= \beta_{j-1}u_{j-1} \quad 2 \le j \le J \end{aligned} \tag{2.5}$$

Similar to eq. (1.13), we have

$$PS = 1/(1-\beta_1 \ldots \beta_J). \tag{2.6}$$

$\beta_1, \ldots, \beta_j$ can be computed using the relation (2.5) and the relation $\beta = \sum_{i=1}^{J} \beta_i/J$. To obtain PU, we must find the fraction of cells in collision.

A collision number k of a cell is defined as it is defined for the bucket and k ranges from 0 to $MJ-1$.

Let $a_j^{(k)}$ be the probability, for each bank j, of a cell being occupied with collision number k; and let $b_j^{(k)}$ be the probability, for each bank j, of a cell being empty with collision number k. Then, we have for $j = 1, 2, \ldots, J$:

$$d^+a_j^{(k)}/d\alpha_j = (a_j^{(k-1)} - a_j^{(k)} + b_j^{(k)})/(1-\alpha_j) \tag{2.7}$$

$$d^+b_j^{(k)}/d\alpha_j = -b_j^{(k)}/(1-\alpha_j) \tag{2.8}$$

$$d^-a_j^{(k)}/d\alpha = (-a_j^{(k)}/\alpha) + ((k+1)a_j^{(k+1)}/\alpha) - (ka_j^{(k)}/\alpha) \tag{2.9}$$

$$d^-b_j^{(k)}/d\alpha = (a_j^{(k)}/\alpha) + ((k+1)b_j^{(k+1)}/\alpha) - (kb_j^{(k)}/\alpha) \tag{2.10}$$

The coefficient $1/(1-\alpha_j)$ in equations (2.7) and (2.8) is the average number of probes to insert a new key in bank j. The increment of the number of cells in bank j with collision number k caused by the insertion is the product of the average number of probes for insertion and the total number of cells that may be turned into the cells with collision count k by the insertion. Hence, equations (2.7) and (2.8) follow.

Similarly, $d^-Ma_j^{(k)}/d(MJ\alpha)$ is the increment of the number of cells with collision number k caused by deletion of a key. Each term of the right hand side of equation (2.9) is accounted for from the consideration:

(i) $a_j^{(k)}/(J\alpha_j)$ is the probability of the cell, in bank j, with collision number k being deleted.

(ii) Term $ka_j^{(k)}/J\alpha$ is the contribution of the deletion of keys that are in collisions of $Ma_j^{(k)}$ cells in bank j (collision term). Let T be the total number of collisions and N be the total number of occupied cells in the hash table, then

$$T = \sum_{j,k} Mk(a_j^{(k)} + b_j^{(k)}) \quad \text{and} \quad N = \sum_{j,k} Ma_j^{(k)}.$$

Number of candidate keys is $Mka_j^{(k)}$. Since the decrement of the collision number due to one key deletion is (on the average) T/N, the number of collisions concerned is $(T/N)Mka_j^{(k)}$. Therefore, ratio of this quantity to T gives the desired contribution.

(iii) $(k+1)a_j^{(k+1)}/J\alpha$ is also a collision term and is derived from the same consideration as (ii).

Equation (2.10) is similarly derived.

We now introduce generating functions for the probability distributions $a_j^{(k)}$'s and $b_j^{(k)}$'s,

$$f_j(\lambda) = \sum_{k=0}^{\infty} a_j^{(k)}\lambda^k \quad \text{and} \quad g_j(\lambda) = \sum_{k=0}^{\infty} b_j^{(k)}\lambda^k \quad j = 1, \ldots, J.$$

It can be shown that equations (2.7)–(2.10) under the conditions

$$d^+a_j^{(k)}/d\alpha + d^-a_j^{(k)}/d\alpha = 0 \text{ and } d^+b_j^{(k)}/d\alpha + d^-b_j^{(k)}/d\alpha = 0 \quad \text{at } \alpha=\beta,$$

(i.e. in the worst cycle) reduce to confluent hypergeometric differential equations, whose solutions are given by

$$f_j(\lambda) = \beta_j\, {}_1F_1(1+s_j, s_j+2; s_j(\lambda-1)) \tag{2.11}$$

$$g_j(\lambda) = (1-\beta_j)\, {}_1F_1(s_j, s_j+2; s_j(\lambda-1)) \tag{2.12}$$

where $s_j = J\beta\cdot\beta_1\cdots\beta_{j-1}/(1-\beta_1\cdots\beta_J)$ and ${}_1F_1$ is defined in (1.26). Since an unsuccessful search terminates when at least one of the J cells that are simultaneously read out is with zero collision number,

$$\begin{aligned} PU &= 1/\left(1 - \prod_{i=1}^{J}(1-a_i^{(0)}-b_i^{(0)})\right) \\ &= 1/\left(1 - \prod_{i=1}^{J}(1-f_i(0)-g_i(0))\right) \end{aligned} \tag{2.13}$$

PU and PS in Scheme 2 can be obtained if we let $\beta = \beta_1 = \beta_2 = \cdots = \beta_J$. instead of (2.5). Clearly, in Scheme 2

$$f_1(\lambda) = f_2(\lambda) = \cdots = f_n(\lambda) = \beta\, {}_1F_1(q+1, q+2; q(\lambda-1))$$

$$g_1(\lambda) = g_2(\lambda) = \cdots = g_n(\lambda) = (1-\beta)\, {}_1F_1(q, q+2; q(\lambda-1))$$

where $q = \beta/(1-\beta)$.

5. Composite Algorithms and Their Probe Numbers

In this section, we give the algorithm for "Insert a new key if search is unsuccessful" and its probe number PI^* in the three schemes. Note that this is the algorithm to be used in the construction of associative tables. A probe number PD^* for D^* discussed in section 3 is related to the relation $PD^* \le 2PS$.

Algorithm 1I*: This algorithm inserts key k in the hash table if key k is not stored in it, and Y is set to *true*. Otherwise, Y is set to *false*. In either case, R is set to the address of the cell where key k is stored.

(I*1–1) [Initialization] Set $i \leftarrow 1$, del$\leftarrow$*false*, ocu$\leftarrow$*false*, $Y \leftarrow$ *true*.

(I*1–2) [Hash address generation] }
(I*1–3) [Parallel hash table access] } the same as (I1–2) and (I1–3), respectively.

(I*1–4) [Check for match]
If $K[h,j] = k$ for some j, set $R \leftarrow$ address of $K[h,j]$ and $Y \leftarrow$ *false*, otherwise go to (I*1–5).
Decrement the collision counters that have been incremented by the application of this algorithm using an algorithm similar to Algorithm 1D, and

terminate the algorithm.

(I*1–5) [Generate boolean conditions]

Set $E \leftarrow \bigvee_{j=1}^{J} (K[h,j]=\phi) \bigwedge C[h]=0$,

$O \leftarrow \bigwedge_{j=1}^{J} (K[h,j] \neq \phi) \bigwedge C[h]=0$,

$C \leftarrow \bigwedge_{j=1}^{J} (K[h,j] \neq \phi) \bigwedge C[h]>0$,

$D \leftarrow \bigvee_{j=1}^{J} (K[h,j]=\phi) \bigwedge C[h]>0$,

and jump to (I*1–6, I*1–7, I*1–8, I*1–9) according to the boolean conditions that have been set in this step.

(I*1–6) [In case E is *true*]
If del = *true*, set $K[x,y] \leftarrow k$, $R \leftarrow$address of $K[x,y]$ and terminate the algorithm.
Otherwise, set $K[h,n] \leftarrow k$ where n is the index such that $K[h,n]=\phi$, set $R \leftarrow$address of $K[h,n]$ and terminate the algorithm.

(I*1–7) [In case D is *true*]
Let n be the index such that $K[h,n]=\phi$.
If ocu = *true*, set $K[h,n] \leftarrow k$, $R \leftarrow$address of $K[h,n]$ and terminate the algorithm.
If del = *true*, set $i \leftarrow i+1$ and go to (I*1–2).
Otherwise, set $x \leftarrow h$, $y \leftarrow n$, del $\leftarrow$ *true*, $i \leftarrow i+1$ and go to (I*1–2).

(I*1–8) [In case O is *true*]
If del = *true*, set $K[x,y] \leftarrow k$, $R \leftarrow$address of $K[x,y]$ and terminate the algorithm.
Set ocu $\leftarrow$ *true*, $C[h] \leftarrow C[h]+1$, $i \leftarrow i+1$, and go to (I*1–2).

(I*1–9) [In case C is *true*]
If del = *false*, set $C[h] \leftarrow C[h]+1$
Set $i \leftarrow i+1$ and go to (I*1–2).

Algorithm 2I*: This algorithm performs the same function as Algorithm 1I*.

(I*2–1) [Initialization] Set $i \leftarrow 1$, del $\leftarrow$ *false*, ocu $\leftarrow$ *false*, $Y \leftarrow$ *true*.

(I*2–2) [Hash address generation] }
(I*2–3) [Parallel hash table access] } the same as (I2–2) and (I2–3), respectively.

(I*2–4) [Check for match]
If $K[h[j],j]=k$ for some j, set $R \leftarrow$address of $K[h[j],j]$ and $Y \leftarrow$ *false*, otherwise go to (I*2–5).
Decrement the collision counters that have been incremented by this algorithm using an algorithm similar to Algorithm 2D, and terminate the algorithm.

(I*2–5) [Generate boolean conditions]

Set $C \leftarrow \bigwedge_{j=1}^{J} (K[h[j],j] \neq \phi \bigwedge C[h[j],j]>0)$,

$O \leftarrow \bigwedge_{j=1}^{J} (K[h[j],j] \neq \phi) \bigwedge \bar{C}$,

$D \leftarrow \bigwedge_{j=1}^{J} (C[h[j],j]>0) \bigwedge (K[h[n],n]=\phi$ for some n),

$E \leftarrow (\overline{O \bigvee C \bigvee D})$

and jump to (I*2–6, I*2–7, I*2–8, I*2–9) according to the boolean condition that have been set in this step.

(I*2–6) [In case E is *true*]
If del = *true*, set $K[x,y] \leftarrow k$, $R \leftarrow$address of $K[x,y]$ and terminate the algorithm.
Otherwise, for $m=j_1, \ldots, j_J$ do, in this order;
if $K[h[m],m]=\phi$ then set $K[h[m],m] \leftarrow k$, $R \leftarrow$ address of $K[h[m],m]$ and terminate the algorithm, otherwise set $C[h[m],m] \leftarrow C[h[m],m]+1$.

(I*2–7) [In case D is *true*]
If ocu = *true*, for $m=j_1, \ldots, j_J$ do, in this order; if $K[h[m],m]=\phi$ then set $K[h[m],m] \leftarrow k$, $R \leftarrow$ address of $K[h[m],m]$ and terminate the algorithm, otherwise set $C[h[m],m] \leftarrow C[h[m],m]+1$.
If del = *true*, set $i \leftarrow i+1$ and go to (I*2–2)
Otherwise, for $m=j_1, \ldots, j_J$ do, in this order; if $K[h[m],m]=\phi$ then set $x \leftarrow h[m]$, $y \leftarrow m$, del $\leftarrow$ *true*, $i \leftarrow i+1$, and go to (I*2–2)

(I*2–8) [In case O is *true*]
If del = *true*, set $K[x,y] \leftarrow k$, $R \leftarrow$address of $K[x,y]$ and terminate the algorithm.
Set ocu $\leftarrow$ *true*, for $j=j_1, j_2, \ldots, j_J$ set $C[h[j],j] \leftarrow C[h[j],j]+1$.
Set $i \leftarrow i+1$, and go to (I*2–2).

(I*2–9) [In case C is *true*]
If del = *false*, for $j=j_1, j_2, \ldots, j_J$ set $C[h[j],j] \leftarrow C[h[j],j]+1$.
Set $i \leftarrow i+1$ and go to (I*2–2).

Algorithm 3I* is the same as Algorithm 2I* except that $j_1, j_2, \ldots, j_J$ are fixed regardless of key k and i; *i.e.*, $j_1=1, j_2=2, \ldots, j_J=J$.

*PI** of the three schemes can be determined by the probabilities of O, E, C and D being set in step 5: That is, when insertion is successful, I* is rephrased as;

if O, apply Alogrithm I, else
if E, terminate the algorithm after inserting a key, else
if C, apply Algorithm I*, else (necessarily D)
perform an unsuccessful search and then insert a key.
Thus $PI^* = Pr[O](1+PI)+Pr[E]+Pr[C](1+PI^*)+Pr[D](PU+1)$ or $PI^* = (Pr[O]PS+Pr[D]PU+1)/(Pr[O]+Pr[E]+Pr[D])$

where we use the notation $Pr[x]$ which denotes the probability of x being set, and the relation $PI=PS$. Otherwise, when key duplication is detected $PI^*=2PS$

Table 1 lists formulae giving $Pr[O]$, $Pr[E]$, $Pr[C]$ and $Pr[D]$ at a given load factor β in the three schemes without counting a memory access for actual key insertion. We note $PI^* \leq PI+PU=PS+PU$ since some of the hash addresses generated in algorithms S and I are shared in algorithm I*.

6. Results of Analysis

Figs. 2, 3, 4 and 5 show the graphs of PU and PS with load factor β as a parameter.

Table 1 Formulae of $Pr[O]$, $Pr[E]$, $Pr[C]$ and $Pr[D]$.

Scheme 1

$$Pr[E]=\sum_{j=0}^{J-1} p_j F_j(q;-q)$$
$$Pr[O]=p_J F_J(q;-q)$$
$$Pr[C]=p_J F_J(q;0)-Pr[O]$$
$$Pr[D]=1-Pr[E]-Pr[O]-Pr[C]$$

Scheme 2

$$Pr[E]=1-Pr[O]-Pr[C]-Pr[D]$$
$$Pr[O]=\beta^J-Pr[C]$$
$$Pr[C]=\beta^J(1-F_1(q;-q))^J$$
$$Pr[D]=[1-F_0(q;-q)+\beta(F_0(q;-q)-F_1(q;-q))]^J-Pr[C]$$

Scheme 3

$$Pr[E]=1-Pr[O]-Pr[C]-Pr[D]$$
$$Pr[O]=\prod_{i=1}^{J}\beta_i-Pr[C]$$
$$Pr[C]=\prod_{i=1}^{J}(\beta_i-f_i(0))$$
$$Pr[D]=\prod_{i=1}^{J}(1-f_i(0)-g_i(0))-Pr[C]$$

We observe the following relations for $J\leq 64$ and $\beta\leq 0.90$.

$$PS_3<PS_2, PS_1$$
$$PU_3<PU_2, PU_1$$
$$PI_3^*<PI_2^*, PI_1^*$$

(The subscripts denote Scheme numbers.)
Figures 2 and 3 show clearly the improvements of the performance of the multi-bank schemes. (Although we only show PU and PS in Scheme 3, similar improvements are observed in Schemes 1 and 2.) Figures 4 and 5 are comparisons of the three schemes at a fixed number of banks $J=4$. PU's increase rapidly as β approaches 1. Figure 5 shows that PU_1 is slightly larger than PU_2 and PU_3 for $\beta\leq 0.85$. However, we find the following relations for $\beta=0.95$ and $J=2, 4, \ldots, 64$:

$$PU_2>PU_1>PU_3.$$

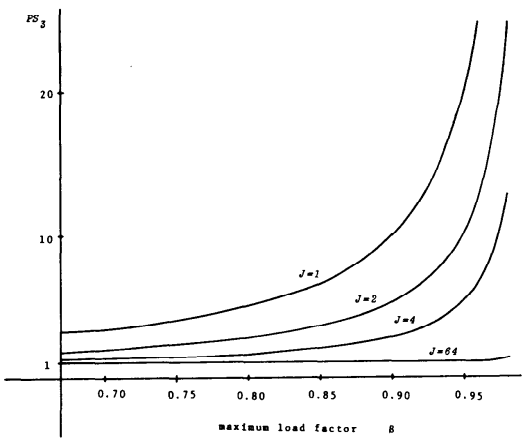


Fig. 2 Dependence of PS on number of banks J in Scheme 3.

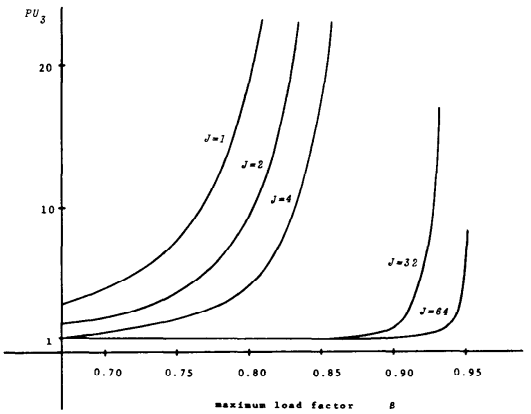
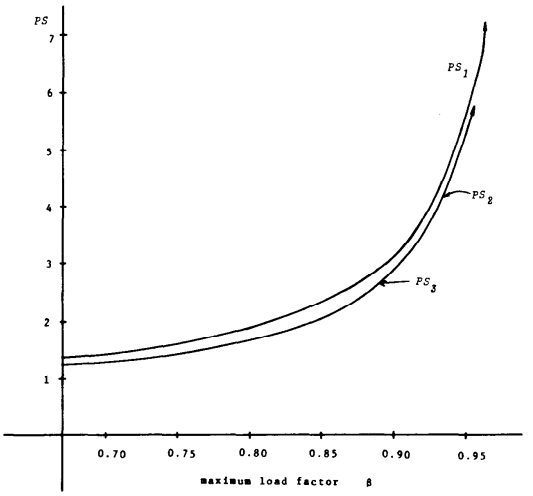


Fig. 3 Dependence of PU on number of banks J in Scheme 3.



Note: PS_2 is slightly larger than PS_3 although indistinguishable in the figure.

Fig. 4 PS in the three schemes for number of banks $J=4$.

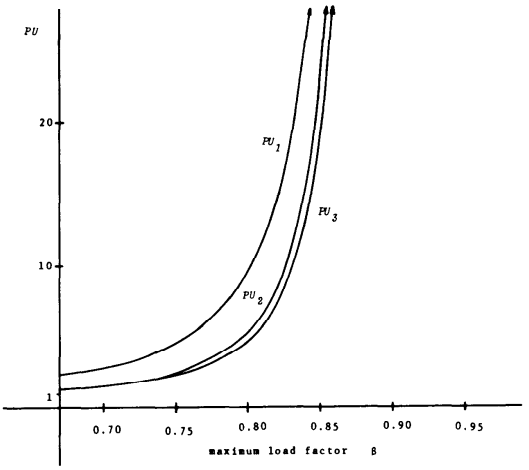


Fig. 5 PU in the three schemes for number of banks $J=4$.

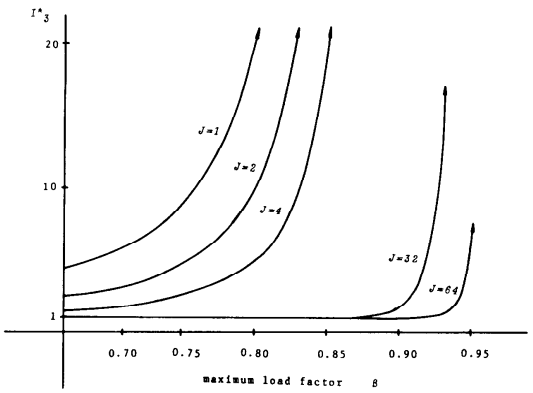


Fig. 6 Dependence of PI^* on number of banks J in Scheme 3.

Fig. 6 shows the graph of PI^* in Scheme 3. Since the values of PU's dominate those of PS's in $\beta \geq 0.9$, graphs of PI^* exhibit similar behavior to those of PU's as β asymptotically approaches 1.

7. Concluding Remarks

When parallelism is fully exploited, average probe numbers are shown to be greatly reduced over the conventional single-bank sequential hashing. In the three schemes, parallelism of hash table access is exploited. Implicit in the calculation of PS, PU and PI^* is the assumption that hash address generation (n hash addresses at one hash probing in Schemes 2 and 3) and hash table access are performed in parallel. Therefore, these probe numbers give the (worst) processing time for the algorithms S, I and I* in the memory cycle time unit. These values are cross-checked with those obtained by simulations for small J and β. Although Scheme 2 is the most complicated, our analysis revealed that its performance is the poorest when the hash table gets nearly full. Overall performance of Scheme 3 seems to be best. The schemes are particularly suited for the implementation of a hash hardware since the contemporary primary memory for a large scale computer employs banking similar to the one for the hash table proposed. In this regard, Scheme 1 requires less hardware resource than Scheme 3 and, moreover, is easiest to implement. Choice of the schemes in the implementation is up to the system's requirement (*i.e.* speed and cost of a processor and memory).

A common aspect in hashing is the degradation of efficiency when β approaches 1. From the results, it is seen that larger permissible maximum load factor β can be selected when J is taken large, say 64, than that of conventional single bank hashing. One should note that timing values obtained in the analysis are the ones in the worst cases.

Randomness of hash sequences and of key deletions is a subject for controversy in the real applications. However, this is not the scope of this paper.

Basic hashing operations in these schemes can be performed in a time comparable to single indirect addressing.

We found that calculation of the probe numbers for large J and β require special treatment of the exponent parts of floating numbers. We had to write a special floating arithmetic package ($|\text{exponent}| < 2^{35}$). Suitable automatic means of handling 'big exponent' [10] and 'variable precision' [10, 11] in future computer systems are strongly urged.

References

1. FELDMAN, J. A. AND ROVNER, P. D. An Algol-Based Associative Language. *CACM*, **12**, 8, (August 1969), 439–449.
2. FABRY, R. S. Capability-Based Addressing. *CACM*, **17**, 8, (August 1974) 403–412.
3. SCHWARTZ, J. T. On Programming an Interim Report of the SETL Project, Installment II: The SETL language and examples of its use, Courant Institute of Mathematical Sciences, New York University, New York, N.Y. 1973.
4. GOTO E. AND KANADA, Y. Hashing Lemmas on Time Complexity with Application to Formula Manipulation. Proc. ACM-SYMSAC '76, New York, 1976.
5. SASSA, M. AND GOTO, E. A Hashing Method for Fast Set Operations. *Information Processing Letters*. **5**, 31–34, 1976.
6. IDA, T. AND GOTO, E. Performance of a Parallel Hash Hardware with Key Deletion. Proc. IFIP 77 Congr. Tronto, 1977.
7. GOTO, E., IDA, T. AND GUNJI, T. Parallel Hashing Algorithms. *Information Processing Letters*. **6**, 8–13, 1977.
8. KNUTH, D. E. *The Art of Computer Programming Vol.* **3**, Addison-Wesley Publishing Company, 1973.
9. GUNGI, T. AND GOTO, E. Comparison of Hash Algorithms with Key Deletion (To be published).
10. FATEMAN, R. J. The MACSYMA 'Big-Floating-Point' Arithmetic System. Proc. ACM-SYMSAC '76, New York, 1976.
11. WYATT, W. T. ET AL. A Portable Extended Precision Arithmetic Package and Library with Fortran Precompiler, *ACM Transactions on Mathematical Software*, **2**, 3, (1976).