A Collection of Research Processes for Genealogy and Proofs

VOLUME FOUR, SECTION 23

Technical Report, "Four Hash Algorithms and Analysis."
Written Since June 28, 1989

by

Dr. Dong-Keun Shin

On June 13, 1996

Submitted to the Chair of
Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley
Berkeley, CA 94720

FOUR HASH ALGORITHMS AND ANALYSIS

< ABSTRACT >

In this paper, four different hash algorithms will be described, and they will be discussed based on the criteria given by Donald Knuth. Especially, this report focuses on the algorithm which will fit well into database hardware filter which millions of data might be passed through. Any selected hash algorithm for the database hardware filter should distribute keys into buckets as uniformly as possible. And it is also emphasized that the algorithm should compute hash address very fast using any necessary hardware components. One of the main suggestions is to avoid time-consuming serial and/or iterative computation to generate each hash address. It is ideal to have a data independent hash function which calculates a hash address within single machine cycle with reasonably good distributions. Character to Prime Numbers Mapping hashing function with an aid of hardware components produces every hash address within single machine cycle, And the statistical analysis shows that the minimizing collision occurrences of the algorithm is better than that of the famous division method which is data dependent hash algorithm.

< INTRODUCTION >

According to what Knuth requires for good hash function in his book (The Art of Computer Programming, Vol. 3 "Sorting and Searching", Page 512), it must satisfy two requirements such as the followings:

1) Its computation should be very fast.

2) It should minimize collisions.

He added that even though many hash methods have been suggested, none of them proved to be superior to the simple division and multiplication methods. However, regarding the time consumed to add up the characters in a key and divide and/or multiply the sum, the required cycle time might be considerably large. Although this issue is quite machine dependent, this argument becomes extremely important when series of millions and millions data are waiting to be hashed out in a database computer.

The proposed hash algorithm might be called as "Characters to Prime Numbers Mapping Hash Function." This algorithm requires hardware aids such as Random Access Memorys and a bunch of exclusive-or gates to calculate a hash address with single machine cycle. So it might be more than 100 times faster than the division method. Moreover, since this algorithm is not data dependent at all, it has less chance of collision occurrences when keys are similar. For example, by the division method algorithm, the keys like 'XY' and and 'YX' will be inserted into the same bucket. And the keys like 'generator1', 'generator2', and 'generator3' probably will be put into the buckets next to each other. When the key attributes are somehow similar in any form, the data dependent hash function including the division method perform poorly showing some sorts of data clusterings.

In this report, the algorithms of four hash functions will be described and their main programs used in the modules for the performance analysis will be depicted in each section. The result of statistical analysis of those hash functions using three different data sets will will be followed. In the first section, the well-known division method will be shown. And in the second section, "Character Based Hash Function" which is very similar to Digit Analysis Hash Method will be described. Professor Maurer has recommended one hash function which is basically shift and exclusive or hashing scheme. This data independent hash function will be shown in the third section. In the fourth section, the proposed "Characters to Prime Numbers Mapping Hash Function" (or just called "Mapping Hash Function") will be illustrated.

1. Division Method Hash Function

This hash algorithm simply adds up the ordinal number of ASCII characters in a key, then it gets the remainder after dividing the sum (K) by bucket size number (M). So the resulting remainder (h(k)) could represent any bucket number from 0 through M-1.

$$h(K) = K \bmod M$$

It is said that if a prime number is chosen for M, it might provide better distributions. But this is not necessarily a true statement. As it was mentioned in the introduction, this hash function is quite data dependent function. The performance is largely dependent on the loading factor (the ratio of the number of records to the bucket size). If the bucket size gets substantially large, this function might show more data clustering when the keys are similar in any forms. When this function was tested with 16 ASCII characters random numbers, the distribution was very poor since the sum of the 16 ASCII characters might be within some number range. When Lum, Yuen, and Dodd tested this function in 1971, they only used 1, 2, 5, 10, 20, and 50 bucket sizes. This experiment was not sufficient to support their and Knuth's conclusion. The main program of the division method testing module will be shown in the next page.

MAIN PROGRAM OF THE DIVISION METHOD HASHING SCHEME IN PASCAL

```
{**************** MAIN PROGRAM STARTS HERE ******************}

begin

     Initialization;

     {Read Keys one by one converting them to hashed addresses}

     for Key_No := 1 to MAX_NUM_KEYS do

       begin

         Init_While_Do_Loop;

         {Read character by character for a key looking up the
          corresponding ASCII value}

         while More_Chars_Left_For_Key do

           begin
              Read_A_Char;
              Add_It_To_Sum(sum);
              Increment_Char_Pointer;
           end;


         {Using the resulting hashed address, store the input key in
          the corresponding bucket.}

         Store_Key_In_Addressed_Bucket(sum);

       end; {outer for}

     {Print out all the keys in every hashed buckets.}

     Print_Keys_In_Each_Bucket;

     {Print out all the necessary statistics for an analysis.}

     Print_Statistics;

end.
```

2. Character Based Hash Function

This algorithm uses the first two characters of each key to generate hash address for the key. The first character determines which group of buckets the key will be inserted into. Each group has 10 buckets. There are 26 alphabets in English and there are 256 buckets. Assumingly each key for this function must have alphabets in it's first two character. So bucket number 0 through 9 will contain keys which have 'a' or 'A' in their first character. The bucket number 10 through 19 are for keys which have 'b' or 'B' in their first character and so on. Accordingly, the bucket number 250 through 255 for keys which have 'z' or 'Z' in their first character. Depends on the second character of the key, the exact bucket number that the key belongs to will be determined. The least significant digit in the bucket number is as followings.

BUCKET NO.	THE SECOND CHARACTER
_ _ 0 :	'a'/'A', 'b'/'B', 'c'/'C'
_ _ 1 :	'd'/'D', 'e'/'E'
_ _ 2 :	'f'/'F', 'g'/'G', 'h'/'H'
_ _ 3 :	'i'/'I', 'j'/'J'
_ _ 4 :	'k'/'K', 'l'/'L', 'm'/'M'
_ _ 5 :	'n'/'N', 'o'/'O'
_ _ 6 :	'p'/'P', 'q'/'Q', 'r'/'R'
_ _ 7 :	's'/'S', 't'/'T'
_ _ 8 :	'u'/'U', 'v'/'V', 'w'/'W'
_ _ 9 :	'x'/'X', 'y'/'Y', 'z'/'Z'

Then it will check if the resulted bucket number resides between 0 through 255, since ther is a chance to have addresses like 256, 257, 258, and 259. In this case, it divides the number by maximum bucket size (256) to get the remainder. And this remainder becomes the final bucket address. The main program of the character based hash testing module will be shown in the next page.

The basic idea of this hash function is similar to that of the Digit Analysis hash function. The performance evaluation of this hash function shows the worst since it is extremely data dependent.

MAIN PROGRAM OF CHARACTER BASED HASHING SCHEME IN PASCAL

```
{**************** MAIN PROGRAM STARTS HERE ******************}

begin

     Initialization;

     {Read Keys one by one converting them to hashed addresses}

     for Key_No := 1 to MAX_NUM_KEYS do

       begin

         Init_While_Do_Loop;

         {Read first two characters of a key to get a hashed address}

         Read_First_Two_Chars(addr);

         {Read character by character to form a key}

         while More_Chars_Left_For_Key do
           begin
              Read_A_Char;
              Increment_Char_Pointer;
           end;


         {Using the resulting hashed address, store the input key in
          the corresponding bucket.}

         Store_Key_In_Addressed_Bucket(addr);

       end; {outer for}

     {Print out all the keys in every hashed buckets.}

     Print_Keys_In_Each_Bucket;

     {Print out all the necessary statistics for an analysis.}

     Print_Statistics;

end.
```

3. Professor Maurer's Hash Function

This algorithm requires 2 registers for a key to be stored in. One register must have fast shift operation functions to reduce the hash address calculation time. The algorithm is the followings.

1. Store A key both in the shift register and key register.
2. The shift register will rotate one bit right, so the right most bit will be stored in the left most bit in the shift register.
3. Then from the rightmost bit of both the key register and shift register, get two bit from the two registers and exclusive or them and store the resulted bit into the key register. Repeat this process until the leftmost bit of the key register and and the shift register are completed.
4. In the second time, shift three bits right, and do the same exclusive oring as described in the process 3.
5. Then shift 7 bits right, and do the same exclusive oring as described in the process 3.
6. Then shift 15 bits right, and do the same exclusive oring as described in the process 3.
7. Then shift 31 bits right, and do the same exclusive oring as described in the process 3.
8. Then shift 63 bits right, and do the same exclusive oring as described in the process 3.

The main program of the Professor Maurer's hashing scheme will be shown in the next page.

MAIN PROGRAM OF CHARACTER BASED HASHING SCHEME IN PASCAL

```
{**************** MAIN PROGRAM STARTS HERE ******************}

begin

     Initialization;

     {Read Keys one by one converting them to hashed addresses}

     for Key_No := 1 to MAX_NUM_KEYS do

       begin

         Init_While_Do_Loop;

         {Read character by character for a key looking up the
          corresponding ASCII value and convert the value to binary}

         while More_Chars_Left_For_Key do

           begin
              Read_A_Char;
              Convert_ASCII_To_Binary;
              Increment_Char_Pointer;
           end;

         Shift_And_EX_OR(1);
         Shift_And_EX_OR(3);
         Shift_And_EX_OR(7);
         Shift_And_EX_OR(15);
         Shift_And_EX_OR(31);
         Shift_And_EX_OR(63);

         Get_Hashed_Addr (Bucket_Addr);

         {Using the resulting hashed address, store the input key in
          the corresponding bucket.}

         Store_Key_In_Addressed_Bucket(Bucket_Addr);

       end; {outer for}

     {Print out all the keys in every hashed buckets.}

     Print_Keys_In_Each_Bucket;

     {Print out all the necessary statistics for an analysis.}

     Print_Statistics;

end
```

4. Characters to Prime Number Mapping Hashing Function

4.A) Hardware Descriptions and Hash Address Computation

This hashing scheme requires hardware components such as random access memorys for each character of a key and 128 exclusive or gates for this specific design. In a RAM, 2 to the power of 6 (64) prime numbers are stored to be mapped out using ASCII character bits as input address. So the output from a RAM is selected bits of prime number which is chosen from the input character in the key attribute register. In order to reduce the contents of a RAM from 128 words to 64 words, only significant 6 bits out of 7 ASCII character bits are used for RAM input address. The number of identifiable characters in a key is 16 in this scheme, so there are 16 prime numbers coming out of 16 respective RAMs.

These 16 prime numbers are exclusive-ored together to produce a final hash address in a parallel way. First bits of the 16 prime numbers are exclusived-ored together to generate the first bit of the resulting hash address. Simultaneously, the second bits of the 16 prime numbers are exclusive-ored together to generate the second bit of the resulting hash address and so on. All of the 16 bits of final hash address are created at the same time.

To generate the first bit of the hash address, the first bit of the first selected prime number and that of the second selected prime number are exclusive-or together to generate the resulting bit to the next level of exclusive-or. By the same way, the first bit of the third prime number and that of the fourth prime number will be exclusive-ored to generate a resulting bit for the next level of exclusive-or. So these two generated bits are exclusive-ored together in the second level to send a resulting bit to the third level exclusive-or with the resulting bit generated from the fifth, sixth, seventh, and eighth bits.

The resulting bit from the second level exclusive-or will be exclusive-or again with another resulting bit out of the four first bits of the fifth, sixth, seventh, and eighth prime numbers. So the third level exclusive-or resulting bit will be exclusive-or again with the resulted bit came out from the eight first bits of 9th, 10th, 11th, 12th, 13th, 14th, 15th, 16th prime numbers by the the same way. Finally, the fourth level exclusive-or with two bits from the third level will generate the first bit of the hash address.

This is a entirely concurrent process, so while the first bit of the hash address is being calculated, the other fifteen bits of the hash address is being computed through the four levels of exclusive-or gates.

This hash address is acquired within single machine cycle time including several gate delays. Then it uses only portion of this 16 bits address to get a bucket number. If the number of buckets is 256, only 8 bits out of the 16 resulting bits are necessary.

4.B) Simulation Processes.

In software simulation, this process should be very tedious serial task. It generates the first hash address bit going through the four level of exclusive-or functions. Each level has its own storage to keep the resulting boolean value of exclusive-oring. The first level of exclusive-or procedure has a loop to get the eight resulting exclusive-ored bits and store them on the array one at a time. Then it moves to the second level to exclusive-or the eight bits stored in the first level array. Then four bits will be stored at the second level array resulted from the 8 bits in the first level array. In the third level, it stores two boolean bit values into the third level array after the exclusive-oring the four bits in the second level array. The final exclusive-or is called to generate the hash address bit using the two bits stored in the third level array. The final bit is stored in the final hash address array until all bits are calculated.

After it finishes the first bit computation, it does the same process as above to get the second one, and get third one, and so on. When it finishes computation for 16 hash address bit, it uses only the portion of them to get the bucket address. The main program of the characters to prime number mapping hashing scheme testing module will be shown in the next page.

MA˘˘ PROGRAM OF CHARACTERS TO PRIME NUMBERS MAPPING HASH IN PASCAL

```
begin

      Initialization;

      {Read Keys one by one converting them to hashed addresses}

      for Key_No := 1 to MAX_NUM_KEYS do

        begin

          Init_While_Do_Loop;

          {Read character by character for a key looking up the
           corresponding prime number and convert it to binary number}

          while More_Chars_Left_For_Key do

            begin
               Read_A_Char;
               Look_Up_Char_In_Prime_Num_Table(index);
               Save_Binary_Prime_Num(index);
               Increment_Char_Pointer;
            end;


          {Get the first digit of the binary prime numbers which
           converted from a character, and do the Exclusive-Or operation
           to get the first bit for the hashed address. Repeat this
           process up to the last digit.}

          for Bit_No :=1 to MAX_BOOL_DIGIT do

            begin
               First_Level_Ex_Oring;
               Second_Level_Ex_Oring;
               Third_Level_Ex_Oring;
               Last_Ex_Oring_And_Store_An_Addr_Bit;
            end;

          Store_Key_In_Addressed_Bucket;

        end; {outer for}

      {Print out all the keys in every hashed buckets.}

      Print_Keys_In_Each_Bucket;

      {Print out all the necessary statistics for an analysis.}

      Print_Statistics;

end.
```

< MORE DISCUSSION >

There could be hundreds of hash functions, but some of them are well known such as divison method, random method, midsquare, algebraic coding, folding, digit analysis, and radix method. Lum, Yuen, and Dodd (CACM 14 1971, 228-239) mentioned that algebraic method is the second to the division method in their performance evaluations. They also said that midsquare method gives good performance. Both of them could be easily implemented using some hardware components, but the midsquare method requires time for a multiplication and the algebraic coding method takes time for iterative subtraction processes for each key. As they noticed that folding and digit analysis are erratic, both could be ignored. The random method also requires time to generate a random number using the key as the seed. The radix method would take long time to convert each digit of a key into another base number.

< CONCLUSION >

The hash function for the database filter should perform as good as the division method can do, and it must compute a hash address within single machine cycle. To satisfy these two requirements, Characters to Prime Number Mapping hash function with an aid of hardware is highly recommended for the effective database coprocessor and for other similar applicative areas.