Journal of Applied Mathematics & Bioinformatics, vol.3, no.4, 2013, 91-105
ISSN: 1792-6602 (print), 1792-6939 (online)
Scienpress Ltd, 2013

Hash addressing of the quasi-permanent key arrays in multilevel memory

Nikolaos G. Bardis[1], Nikolaos Doukas [1] and Oleksandr P. Markovskyi[2]

Abstract

A hash search organization is proposed that uses quasi – permanent keys in conjunction with perfect hash addressing and probing. The particular characteristic of the proposed method is the fact that the model is oriented to a multilevel organization of the memory of modern computational systems. A mathematical model of the hash search in multilevel memory has been developed that allows the optimization of the hash memory parameters during the design.

Keywords: Hash Memories, Search, Addressing, Multilevel Memory

[1] Informatics and Computer Engineering Lab, Faculty of Mathematics and Engineering Science, Department of Military Sciences, Univ. Military Sciences - Hellenic Army Academy, Vari - 16673, Greece. E-mails: nikolaos@doukas.net.gr , bardis@ieee.org
[2] Department of Computer Engineering, National Technical University of Ukraine, (Polytechnic Inst. of Kiev), Peremohy pr., Kiev 252056, KPI 2003, Ukraine. E-mail: markovskyy@i.ua

Article Info: *Received* : August 1, 2013. *Revised* : October 20, 2013.
Published online : December 10, 2013.

1 Introduction

The operation of a key based search is considered as a fundamental one in information technologies. In a significant number of widely used practical applications, the relative computational load of search operations may be as large as 80% of the total [1]. The progressive development of information systems and of information integration determines a dynamic increase of the volume of the key indices upon which searches are performed [2].

The constant expansion of the scope of use for information systems has as a consequence imposed stricter requirements for the efficiency of search procedure results. More specifically, a large proportion of pattern recognition systems, in which key search is actively used, operate in real time conditions. Based on these facts, the development of key based search technologies is an important practical target, applicable to a wide range of information processing systems.

2 Analysis of Existing Search Technologies

An important factor determining the efficiency of key based search is the incorporation of the multilevel memory organization of modern computational systems into search algorithms. In current conditions, where the volume of indices is constantly increasing and the efficiency requirements upon the search are becoming ever more demanding, the applicability of binary trees and B trees is significantly reduced, given the dependence of the search time on the volume of the key array. Apart from that, such search procedures are anyway less effective in the case of multilevel memory organization.

The fastest current key – based search method available is the associative memory that may be implemented in either hardware (content addressable memory) or in software (hash memory). The basic advantage of associative search is considered to be the fact that the search time is independent of the volume of the

key array. The main disadvantage consist is that hash searching technology requires the excess memory. But in modern condition of reducing memory cost this disadvantage loosing importance.

The second impotent disadvantage of hash searching consists of existing of the collisions. The effectiveness resolution of this problem can be finding by using of different hash techniques for different key arrays.

There exist practical applications of key – based search, where the key array is considered permanent or quasi-permanent (the computational load required for the key based search procedures exceeds by several orders of magnitude the computational load of the procedures for changing the search key array). Such applications include principally pattern recognition systems, electronic translation systems, user identification and authentication in systems supporting remote access and a large proportion of database applications.

During the hash search in permanent key arrays, it is possible to determine a one-way hash transformation that eliminates collisions. In bibliography, this class of methods is referred to as perfect hash addressing [1]. The fundamental advantage of perfect hash addressing is the absence of collisions, i.e. the key search time is determined by the time required for a single memory access. This permits the search schemes to attain maximum search speed, independent of the volume of the search key array [2].

The exercise of determining a hash transformation for perfect hash addressing exhibits exponential complexity. For the completion of this exercise, a series of methods have been proposed in bibliography [1], [2], [3], [5], [6]. The disadvantages of these methods are that they do not take into account the multilevel organization of memories and that they do not allow changing of the keys during operation. The purpose of this research is the modification of the organization of hash searches so that it becomes oriented to the quasi-permanent nature of the key array. Additionally, developments are sought in the

mathematical model of such hash searches for the optimization of its characteristics.

3 Analysis of the Hash Search Model for Quasi-Permanent Indices

The purpose of the hash search model that will be presented is to incorporate the analytic form of the dependencies between the characteristics of the hash-memory that determine its organization into the model and enable it to solve problems of optimization of the architecture of hash memory during the design phase. By hash transformation keys are distributes of memory pages as it shown on Figure 1.

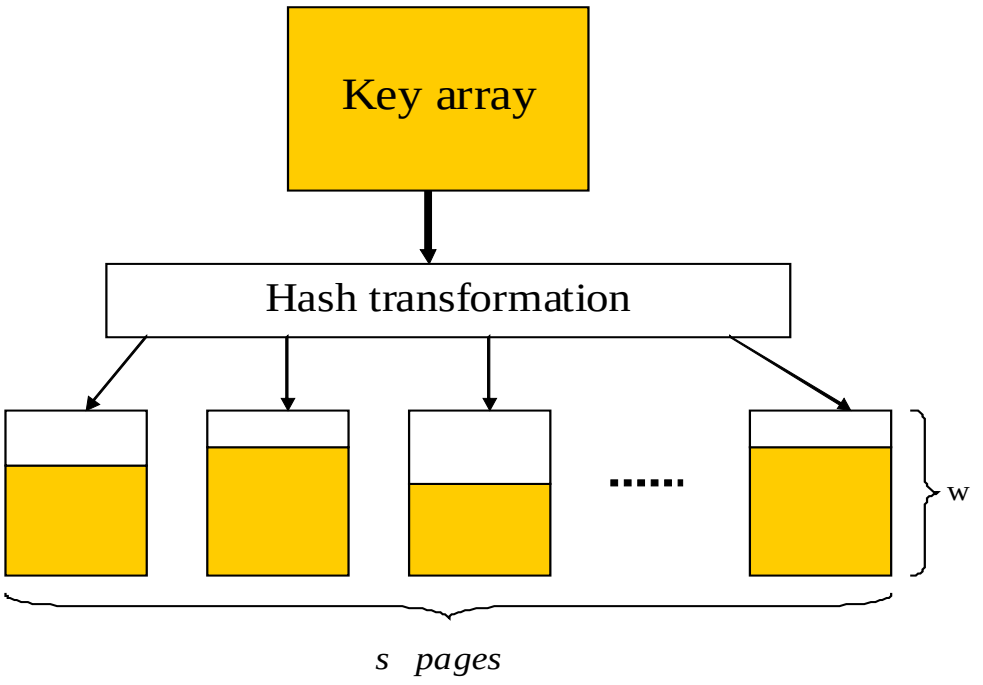

Figure 1: Structure of distribution keys of memory pages

At the basis of the model that will be presented, lies the concept of the determination of a hash transformation $H(X)$ the ensures the mapping of a given set Ω from m keys in s pages of hash memory in such a way that the entire set of keys that are classified in each of the pages do not exceed the value $(\alpha+\delta)\cdot w$, where α is the load factor of the hash memory, δ the allowed variability of the load of a hash memory page load and $\alpha+\delta\le 1$. The load factor α of the hash memory is defined by the relation between the total number m stored records to the maximum feasible record count $M=s\cdot w$ that is determined by the size of the memory:

$$\alpha=\frac{m}{M}=\frac{m}{s\cdot w} \tag{1}$$

As a record, one may consider the information taken as the key associated to a particular data item. The reference address of the position where the data is stored may be found in the record instead of the data. The determination of the hash transformation $H(X)$ that satisfies the above condition may be done by trial and error. As the test mechanism for the hash transformations, it is proposed that the prototype, block – based cryptographic algorithms (DES or Rijndael) be used, that incorporate the one-way cryptographic encoding using the key K, of the data D in the codeword $C: C=H_K(D)$.

The key for the search data X in this case is used as input data to the cipher block whence the key K of the cipher block assumes the role of synchronization code and actually appears together with the number of the hash transformation. The resulting code C of the cipher block is divided in two parts: an *h*-bit packet that serves as a hash address $A_K(X)$ of the page and the remaining bits that become the hash Sing $S_k(X)$ of the key X of the search [7]. Consequently the choice of the Hash transformation $H_K(X)$ is attained via the procedure of changing the key K of the cipher block.

Considering that a page may contain *w* keys, hash address $A_K(X)$, the page has $h = \log_2 s$ − bit codes. The hash function distributes the m keys in *s* groups that contain $n_1, n_2 ... n_s$ keys, since $\sum_{j=1}^{s} \eta_j = m$. Given that the hash transformation arranged the keys in the hash memory page, it is mandatory that the entire set of hash addresses of each page does not exceed the maximum allowed number of $u = (\alpha + \delta) \cdot \omega$ records per page: $\forall \varphi \in \{1,..., \sigma\} : \eta_\varphi \le u$. If this is maintained then in each page of the hash memory, there is enough memory space to store $\omega \cdot (1 - \alpha - \delta)$ records.

Taking into account the fact that the hash transformation $H_K(X)$ arranges the hash address uniformly, then in each page there exist on average $\frac{m}{s}$ hash addresses. As a theoretical model for the distribution of the Hash addresses, the most accurate model is the Bernoulli probability distribution. According to this model, the distribution of the Hash addresses of the m keys within the limits of a given page may be considered as m experiences. The event of the allocation of a given address to the address space of a given page is then associated with a probability of $\frac{1}{s}$. Hence, according to the properties of the model, one may calculate that the mathematical expectation of the hash addresses that correspond to a given page is equal to $\frac{m}{s}$ with variance $c = m \cdot \frac{1}{s} \cdot (1 - \frac{1}{s})$. Hence, according to the de Moivre – Laplace theorem, all the hash addresses that correspond to the range of a given page are subject to a Gaussian distribution with mean $\frac{m}{s}$ and variance $m \cdot \frac{1}{s} \cdot (1 - \frac{1}{s})$.

The probability P_{OS} for a page overflow, i.e. the probability of the number of hash addresses that correspond to a given (constant) page of the hash memory

exceeds u, for a Normal distribution and taking into account (1), is defined by the following expression:

$$P_{os} = 0.5 - \Phi(\frac{u - m/s}{\sqrt{m/s}}) = 0.5 - \Phi(\frac{w\cdot(\alpha+\delta) - m/s}{\sqrt{m/s}}) = 0.5 - \Phi(\delta\cdot\sqrt{\frac{w}{\alpha}}) \quad (2)$$

For a permanent key array, i.e. for the case where a page does not need to have excess free memory, for $w\cdot(1-\alpha-\delta)$ records, where $\delta = 1-\alpha$, the probability that the number of hash addresses that correspond to a given page does not exceed w, is defined by the following expression:

$$P_{osw} = 0.5 - \Phi(\sqrt{w}\cdot\frac{1-\alpha}{\sqrt{\alpha}}) \quad (3)$$

From expression (3) it follows that the probability of overflow of the page at the given order depends on the value of the coefficient δ of the redundant free memory of the page, on the coefficient α of completion as well as on the volume of the page and the memory required for storing these records.

In order to eliminate the possibility of page overflow of all s pages during the population of the constant set Ω of m keys, it is necessary to choose corresponding hash transformations. The probability P_O of a certain trial of the hash transformation achieves elimination of the possibility of overflow for all the pages of the hash memory, is calculated via the calculation of the probability of each page not overflowing:

$$P_0 = (1 - P_{OS})^S = \left[0.5 + \Phi(\delta\cdot\sqrt{\frac{w}{\alpha}})\right]^S \quad (4)$$

The mean g of number of trials that are necessary before choosing the hash transformation, so as to eliminate the possibility of overflow of the pages of the hash memory is calculated by the following expression:

$$g = \sum_{j=1}^{\infty} j\cdot P_0\cdot(1-P_0)^{j-1} = \frac{1}{P_0}. \quad (5)$$

The experiments that were carried out based on the above statistical modeling have shown the sufficiency of the proposed mathematical model of the hash addresses in quasi-permanent key indices in hash memories that are divided in pages. The proposed model may be used for the optimization of the parameters of the hash memory.

From expression (5) it follows that for a given number g_Z of trials for the choice of a hash transformation, that guarantees the allocation of records in the pages with completion of less than $(\alpha+\delta)\cdot 100\%$, the values α, δ and w must satisfy the condition:

$$\sqrt{\frac{w}{\alpha}}\cdot\delta = \Phi^{-1}(S\sqrt{\frac{1}{q_Z}} - 0.5) \tag{6}$$

If it is necessary to ensure the fast selection of a hash transformation, then this may be obtained using the relation $\sigma\cdot\Pi_{o\sigma}\approx 1$. In this case for large values of S, the following approximation holds:

$$P_0 = (1-P_{OS})^S = 1-\sum_{i=1}^{S} \tag{7}$$

The analysis of the mathematical model demonstrates that, the compromise involved in the hash search, exists in the selection of the number of the pages among which exchanges take place between the main and the cache memory. The analysis leads to the conclusion that the search speed essentially depends on the time for transferring the arranged hash page addresses from the main hash memory to the cache memory, which in turn depends on the size of the pages. Consequently, from the point of view of attaining high speed hash searches, the page size needs to be reduced. At the same time, reducing the time required for selecting the hash transformation requires according to (5) an increase of the page size. A resolution of the above compromise may be found by the defined frequency of the key array reconstruction; the more frequently new keys are assigned, the smaller the required time for selecting the hash transformation and

the larger the page size δ may be. The derived by experimental way plots of number trials for hash transformation selection and time searching dependences on page size for number keys 4000, $\alpha = 0.75$ and $\delta = 0.15$ is presented on Figure 2 below.

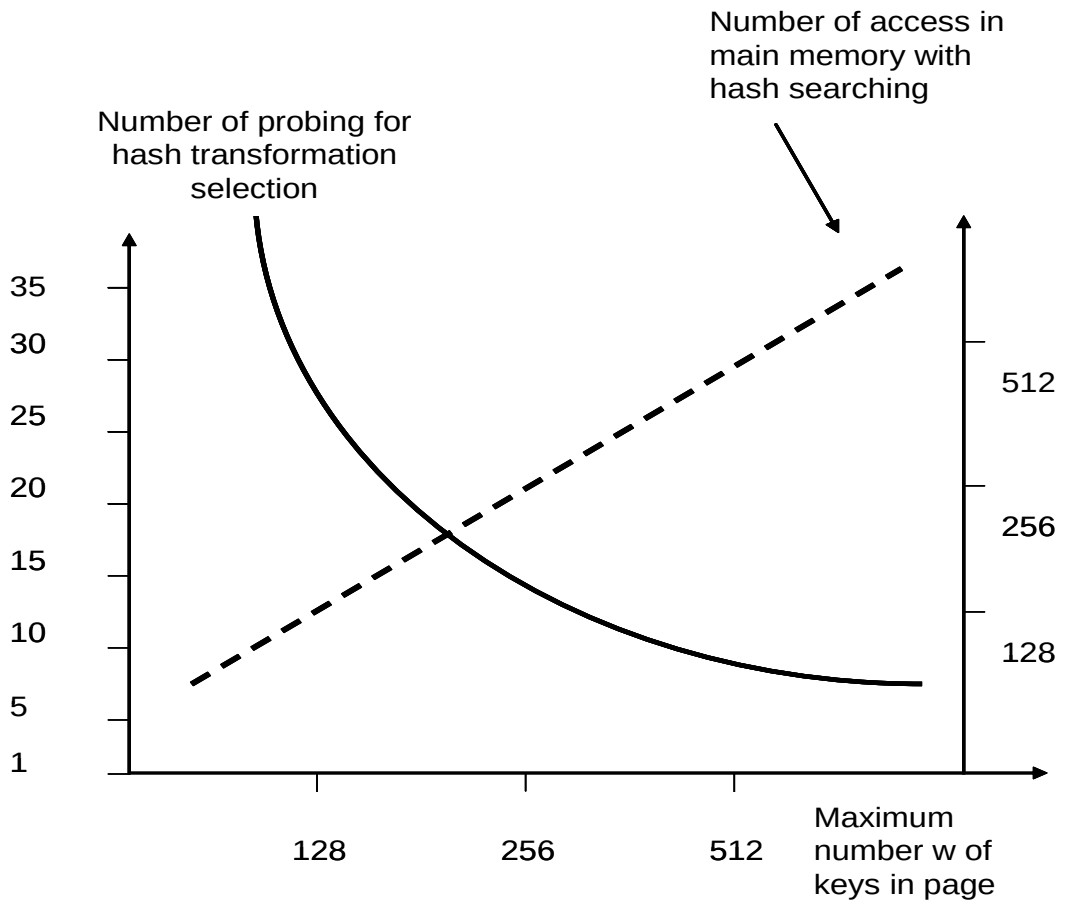

Figure 2: Graph of the dependence of the number of probing for the selection of Hash transformation and the time of hashing search in the dependence on the size of page for 4000 keys, $\alpha = 0.75$ and $\delta = 0.15$

The resolution of these contradicting requirements may be attained either by increasing the size of the pages or by reducing the proportion of the hash memory that is occupied.

4 Organization of Hash Searches

In this section, the principles of the organization of a hash search are outlined. The basic key array exists in the form of Hash Signs and associates the information stored in main memory with the hash addresses. The selection of the hash transformation for nearly constant memories guarantees the assignment of records in the main memory, in accordance with the hash addresses. Structure of proposed hash searching organization on two-level memory is shown in Figure 3.

The cache memory is set as the active page of the main memory. Additionally, an area is assigned for new records for which there is insufficient space in the main memory pages corresponding to their keys. Apart from that, the storing of the most frequently used codes of the hash transformation may also be organized in the cache memory. The hash transformation $H_K(X)$ is determined by selection for the set Ω of the m keys, either before system boot or at a specially determined time during the system setup. The transformation distributes the user records in the s pages of the hash memory so that each page has enough space for storing more than $w \cdot (1 - \alpha - \delta)$ records.

The transformation $H_K(X)$ is determined by selection in the form of the code K of the key and the hash search to be used is hence defined. During this process the set Ω of the keys is divided into s subsets, each one containing less than $w \cdot (\alpha + \delta)$ elements. This means that each page stores less than $w \cdot (\alpha + \delta)$ records. Beside this, the cipher block forms $\log_2 w$ bit hash addresses $U_K(X)$, that define all the records of the corresponding keys X within the pages. The Hash Sign code $S_K(X)$ together with the Hash address $A_K(X)$ of the page and the hash address $U_K(X)$ within the page, using the one way transformation that implements the cipher block, uniquely defines the key X. Within the page, records are inserted using their own hash address $U_K(X)$, Possible conflicts are resolved using known technologies [4], more frequently by trials.

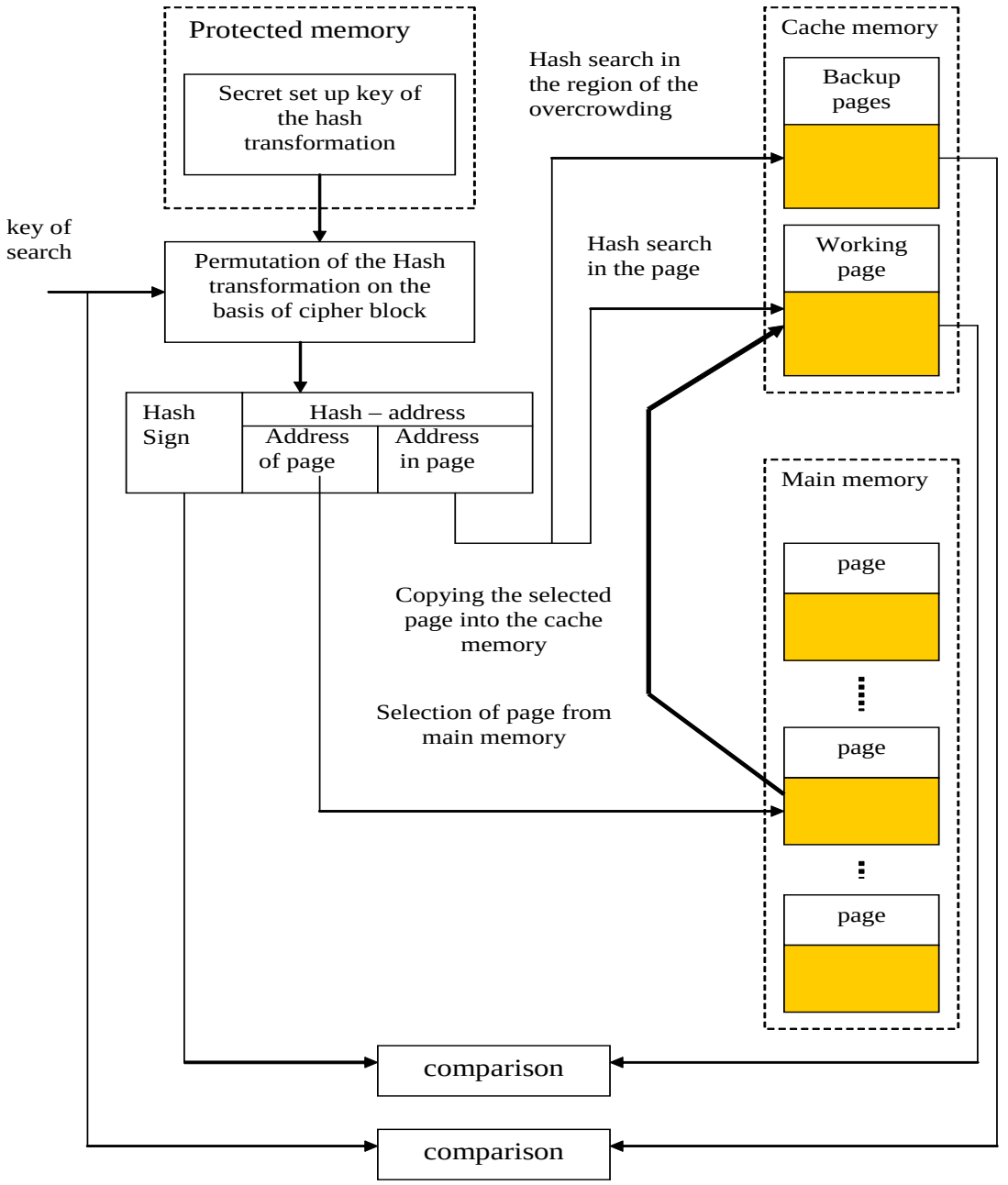

Figure 3: Organization of hash ~~searching in quasi~~-permanent key arrays for two-level memory

Each one of the records includes the hash Sign $S_K(X)$ of the keys associated with the data in each case. For the reduction of the size (length) of the records and the subsequent increase of *w* the reference code of the address may be stored in the place of all codes apart from $S_K(X)$.

For the deletion of a record X, the record is removed from the page by initialization of the memory it occupies and taking into account the displacement of the chain of conflicts to which it formed part.

For the introduction of a new record X, a relocation of the addresses $A_K(X)$ of the pages from main memory to cache memory is generated. If the page is not full, then an attempt is made for the new record X to be inserted at the address $U_K(X)$. If the corresponding page is occupied, then step by step trials are made until a free area is located. If the page that is arranged as $A_K(X)$ appears a completely occupied, then the record X, that is associated with a subset of the bits of the hash address $A_K(X)$ fits in the special sector of the cache memory that is essentially an overflow area. If the corresponding location in this page is also occupied, then an attempt to resolve the conflict by a step by step trial search is made. In the case where the record is inserted in the overflow area, this does not contain the Hash Sign X, but the entire code. If the predetermined overflow area appears to be completely occupied, then a refresh cycle needs to be initiated.

The process of inserting new records in the hash memory described above may be implemented in two ways, each one giving priority to inserting the record in one of the two available areas. The insertion of a new record between cycles of refreshing the system may occur:

- In the overflow area, which offers a small capacity and is located in the cache memory
- In the main hash memory, where it occupies part of an unoccupied zone in the page that is associated with the hash address key of the new record.

From a theoretical point of view, two cases of record organization may be distinguished:

1. Initially the record is inserted in the free space of the associated page of the main memory and in the absence of that space then it is inserted in the overflow area of the cache memory.
2. The record is initially inserted in the overflow area of the cache memory and when this becomes full, in the free space of the associated page of the main hash memory.

It is apparent that the first case will offer a large number of new record entries, which may be distributed in the hash memory before there is a need to refresh the contents. The advantage of the second case is that the use of the overflow area is more efficient and this implies a significantly higher speed of the hash search.

By using the n-bits key X as a key for code K, the cipher block calculates the h-bit hash address $A_K(X)$ of the page, the hash address $U_K(X)$ internal to the page and the Hash Sign code $S_K(X)$. The hash address $Q_K(X)$ for the record X within the overflow area is also simultaneously formulated. The address $Q_K(X)$ is a subset of the bits $A_K(X)$, $U_K(X)$ and $S_K(X)$. The Hash search of the overflow area is performed for the address $Q_K(X)$. If the search is successful, then the record with code X is retrieved from the overflow area and access to the information associated to X is accessed.

Together with the search in the overflow area of the cache memory, the page that is associated with the code $A_K(X)$ is recovered from the main memory to cache memory. Following that, using the address $U_K(X)$ the corresponding record is read. The hash sign of the record is calculated and compared with the hash sign $S_K(X)$. In the event where these two coincide, the passwords are compared and the process of granting access rights is completed. In the event where the codes of the hash signs do not coincide, step by step trials are performed with the remaining record until either a matching hash sign or an empty record is

found. The empty record case implies that a record with code X is not stored in the memory.

The time t_1 of the search with the distinguishing key X is defined as the sum of:

t_H - the time for calculating the hash transformation using the cipher block.

t_s - the time for swapping the page with address $A_K(X)$ from main memory to cache memory

t_X - the time for a hash search in the selected page of the cache memory

$$t_1 = t_H + t_S + t_X$$

Considering that $t_S >> t_H >> t_X$ the search time is principally defined by the time consumed in testing the page from main memory to cache memory.

5 Results

The results obtained from conducting this research, that focus in increasing the efficiency of hash address searching in nearly constant key indices, may be summarized as follows:

A hash search model was developed that corresponds to the nearly constant key array case. The model takes into account the multilevel memory organization of modern computational systems.

The basis of the model that was developed, proposed the organization of hash searches in nearly constant key indices. It was shown that the search time is defined by access to no more than low level memory pages.

The proposed hash memory organization may be efficiently used for increasing the throughput of databases, of linguistic processors as well as for accelerating authentication of users in computer networks.

References

[1] Berman F., Bock M.E., Dittert E., O`Donnel M.J., Plank D., Collections of function of perfect hashing, *SIAM Journal Computers*, **15**(2), (1986), 604-618.

[2] Czech Z.J., Havas G., Majevski B.S., An Optimal algorithm for generating minimal perfect hash functions, *Information Processing Letters*, **43**(5), (1997), 257-264.

[3] Jagannathan R., Optimal partial-match hashing design, *ORSA Journal of Computing*, **3**(2) (1991), 86-91.

[4] Ningping Sun, Ryozo Nakamura, Nonbing Zhu, Akiro Tada, Wenling Sun, An analysis of average search cost of external hashing with separate chain, *Processing of 7-th WSEAS International Conference on Circuits, Systems, Communications and Computers* (CSCC-2003), (2003), 315-324.

[5] Ramakrishna M.V., Bannai Y., Direct perfect hashing function of external files, *Journal of Database Administration*, **2**(1), (1991), 19-28.

[6] Polymenopoulos A., Bardis E.G., Bardis N.G., Markovskaja N.A., Perfect Hashing Using Linear Boolean Functions, *WSEAS Press - Problem in Applied Mathematics and Computational Intelligence*, ISBN: 960-8052-30-0, (2001), 5-11.

[7] Bardis E.G., Bardis N.G., Markovskyy A.P., Spyropoulos A.K., High Storage Utilization of Hash Memory by Reducing of Information Redundancy for Hashing, Submitting in the special issue of *IMACS/IEEE CSCC'99 International MultiConference*, Software and Hardware Engineering for the 21th Century, ISBN: 960-8052-06-8, (1999), 272-276.