Some Aspects of LFU-RBH: A Replacement Algorithm for Database Disk Buffering

I. Atanassov

Key Words: Cache; replacement policy; disk caches least; recently used; least frequently used.

Abstract. *Disk buffering is an important aspect of every computer system. In modern computer systems, the performance gap between the volatile memory and the non-volatile storage systems is in powers of ten. Therefore, the disk block reads/writes can be the bottleneck of the system. A major factor for increasing the overall performance is improving the cache management. Operating systems offer caching to improve I/O performance. Database systems also have a disk cache. Most of the cases it is implemented as a LRU buffer or a variant of it. The present paper offers an algorithm, which tries to take advantage of the frequency aspect and also to reduce the disadvantages of the pure LFU policy. Experiments are conducted to estimate the behavior of the algorithm and the various parameters.*

Introduction

Disk caching is an important aspect of every computer system[10]. It is essential for database servers in order to achieve good performance. There exist many algorithms for disk caching — LRU, LRU-K, FBR, 2Q, LRFU, LIRS, LRD, ARC etc. [2,3,4,5,6,7,8]. The present paper offers an improvement of the previously presented algorithm [1] from the same author. This algorithm is called **LRFU-RBH** (**L**east **R**ecently/**F**requently **U**sed —**R**eferences **B**uffering and **H**ashing), but from now on it is renamed to **LFU-RBH** (**L**east **F**requently **U**sed —**R**eferences **B**uffering and **H**ashing). The experiments are conducted to estimate the hit ratio of the algorithm and to illustrate the influence of the proposed improvement. The algorithm is a part of a larger experimental setup — an implementation of database server.

Most of the existing algorithms are LRU-based. These are LRU itself, LRU-K, 2Q, LRFU. Others are LFU-based — LRD, FBR, LRFU. The proposed algorithm is most like *frequency based*, but some parameters can introduce a measure of the *regency* factor, as mentioned in the next paragraphs. According to [2], the pure LFU replacement policy should never be used in database buffer managers, because of the disadvantages of the policy. The goal of the presented **LFU-RBH** method is to take advantage of the frequency aspect, but in the same time to reduce the disadvantages of the pure LFU policy.

The LFU-RBH Algorithm

The basic topics of **LFU-RBH** are as follows:

— the main modules in LFU-RBH are the **hash table, references buffer, MRU section, data pool;**

— every block in the cache has associated **reference counter;**

— the **References Buffer** (RB) is a FIFO buffer, holding the numbers of the last **r** referenced disk blocks;

— the **RB** is divided in **s** sections;

— when a block enters the **RB** his corresponding *block reference counter* is incremented with **s;**

— when a block leaves a section of **RB**, his corresponding counter is decremented with one;

— if a *block's reference counter* drops to 0, its disk block is not freed;

— the cached blocks are organized through a *hash table;*

— to resolve collisions a chaining technique is used;

— the **collision slots** have an upper limit of 8;

— the hash table holds only the metadata for the structure — *block number, reference counter, pointer to actual data;*

— the data blocks itself are organized in a **data pool** — no one is explicitly associated with a *collision slot;*

— initially all blocks are considered free — they are associated with reference only in demand;

— the granularity of the *data pool* is a parameter of the system;

— optionally the so called **MRU section** can be activated;

— the function of the *MRU section* is to filter single-occurred references, i.e. the table scans.

Figure 1 depicts the data structures, implemented in **LFU-RBH**.

The References Buffer

The **References Buffer** (RB) is the buffer containing the last **r** references to the disk. It can be divided in **s** sections, where 0 < s <= r. When a block number enters the buffer, its *reference counter* is incremented. When a block number leaves a section, the *reference counter* in the corresponding block is decremented with one. The *reference counter* is used to determine which block is not heavily used (the cold blocks) and eventually to replace it. The detailed structure of the **references buffer** is in *figure 2.*

The parameters of the **LFU-RBH** are as follows:

1) Hash table bits — the length of the hash table, i.e. the number of the hash slots.

2) Collision slots count — the number of the collision slots per hash slot. Maximum value of 8.

3) References buffer bits — the length of the RB.

4) RB sections count — the number of sections in the RB.

5) Data pool enabled — indicates whether data pooling is enabled.

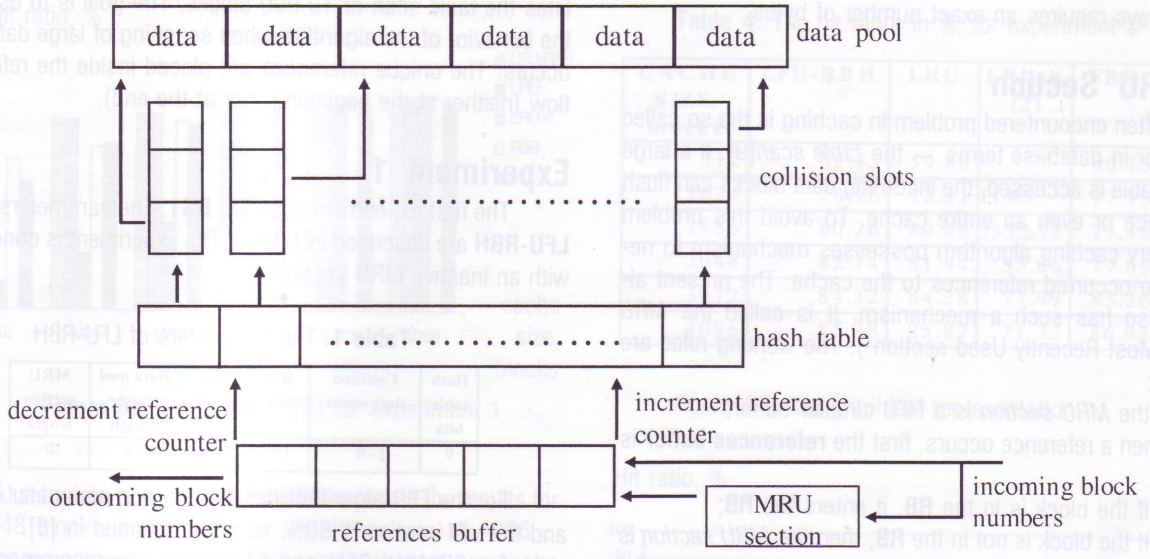


Figure 1. The data structures of the algorithm

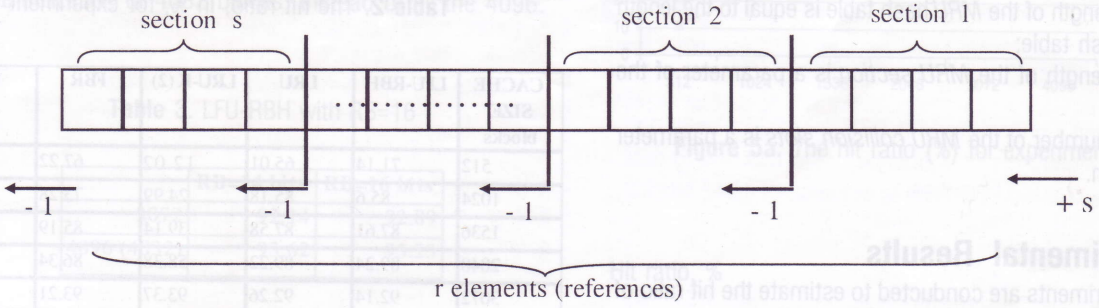


Figure 2. The structure of the references buffer with s sections

6) Data pool cluster length – the length of one data pool cluster. Must be multiple of 8.

7) MRU sections enabled – indicates whether MRU section is active.

8) MRU section length – the number of blocks in MRU section.

9) MRU collision slots – the number of the collision slots for MRU section.

Comments for the Parameters

The number of the *hash table bits* must be greater than 3, i.e. the hash table must have at least 8 slots.

The number of the *collision* slots is limited due to performance considerations. The collision slots are not ordered. Therefore, to resolve collision a straightforward searching must be performed. Thus, the resolving is with linear complexity and the number of collision slots should not be very high.

The length of the **RB** is always a power of 2. In that manner it can be very easily organized as a circular buffer, applying the appropriate mask to the operating index.

The *number of sections* **s** in the **RB** buffer is an arbitrary number in the limits of 0<s<= r. If **RB** does not hold an exact number of sections, the last section is enlarged to the end of the buffer. For example, if the buffer is with length 1024(10 bits) and the number of sections is 10, then the buffer consists of 9 sections with length 102 (the integer part of 1024/10) and the last section is with length 1024 – (102*9) =106 elements (references).

The Data Pool

If *data pooling* is enabled, the parameter *data pool cluster length* determines the chunks a memory is allocated for the needs of the algorithm. For example, if *data pool cluster* is 64, then the first caching demand will allocate 64 * *disk block size* bytes and the corresponding metadata will be initialized. After 64 caching operations the cluster is depleted and the next caching demand will allocate another 64 * *disk block size* bytes and so on. As shown in the experimental results, the memory utilization can drop due to pooling mechanism. *The data pool cluster length* should be multiple of 8, because data pool metadata are organized through an bit-map and for every block is associated a bit for allocated/free status. In that manner the

cluster always requires an exact number of bytes.

The MRU Section

An often encountered problem in caching is the so called *full scan*, or in database terms – the *table scan*[9]. If a large database table is accessed, the incoming data blocks can flush a large piece or even an entire cache. To avoid this problem almost every caching algorithm possesses mechanism to neglect single-occurred references to the cache. The present algorithm also has such a mechanism. It is called the *MRU section* (Most Recently Used section). The working rules are as follows:

– the *MRU section* is a FIFO circular buffer;

- when a reference occurs, first the **references buffer** is checked;

– if the block is in the **RB**, it enters the **RB**;

– if the block is not in the **RB**, then the *MRU section* is checked;

– if the block is in the *MRU*, it is removed from the *MRU* and passed to the **RB**;

– if the block is not in the *MRU*, it enters the *MRU*;

– the *MRU section* has an associated hash table with up to 4 collision slots;

– the length of the *MRU* hash table is equal to the length of the main hash table;

– the length of the *MRU section* is a parameter of the algorithm;

– the number of the *MRU collision slots* is a parameter of the algorithm.

The Experimental Results

The experiments are conducted to estimate the hit ratio of the algorithm and the influence of the various parameters (including *MRU section*). The results are compared with four other caching algorithms – LRU, LRU-K, FBR and 2Q. The used *reference string* (the sequence of the disk block numbers) consists of random numbers. It is generated with the MatLab and has the following characteristics:

1) Reference string 1 (RS1)

10 000 normally distributed references with mean 1000 and variance 30

10 000 normally distributed references with mean 1100 and variance 30

10 000 normally distributed references with mean 1200 and variance 30

10 000 normally distributed references with mean 1300 and variance 30

10 000 normally distributed references with mean 1400 and variance 30

10 000 uniformly distributed references from 1 to 5000

The above mentioned 60 000 references are mixed and added again to the original 60 000 for a total of 120 000 references.

2) Reference string 2 (RS2)

The 120 000 references from the **RS1** plus 10 000 unique (never referenced before and never referenced in the future) numbers for a total of 130 000 references. This simulates the table scan of 10 000 blocks. The goal is to estimate the behavior of the algorithm when scanning of large data area occurs. The unique references are placed inside the reference flow (neither at the beginning, nor at the end).

Experiment 1

The first experiment uses the **RS1**. The parameters of the **LFU-RBH** are described in *table 1*. The experiment is conducted with an inactive MRU section.

Table 1. The parameters of LFU-RBH

Hash table bits	Collision slots count	RB bits	RB sections	Data pool cluster length	MRU section length
9	1-8	14	10	64	0

For the FBR algorithm the *new section* parameter is 25% and the *old section* is 60%, as recommended in [8].

For 2Q A1in=25% and A1out=50%, as recommended in [5].

Table 2 shows the results of the experiment. The columns are the **hit ratio**, in %, for the different algorithms. The first column is the number of blocks for the cache.

Table 2. The hit ratio, in %, for experiment 1

CACHE SIZE, blocks	LFU-RBH	LRU	LRU-K (2)	FBR	2Q
512	71.14	65.01	12.02	67.22	57.32
1024	85.6	85.18	24.99	73.28	74.6
1536	87.61	87.58	39.14	85.19	83.14
2048	89.24	89.22	58.38	86.34	87.65
3072	92.14	92.26	93.37	93.21	91.75
4096	93.62	94.74	95.8	95.54	94.09

Figure 3 is the graphical representation.

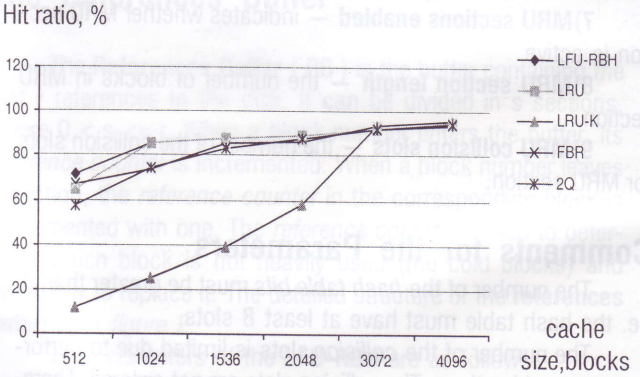


Figure 3a. The hit ratio (%) for experiment 1

As the results show, the **LFU-RBH** outperforms the other algorithms in most of the cases. For the first four cache sizes it shows best results. While it is not true for the last two cache sizes, it is important to denote that for a cache size of 4096

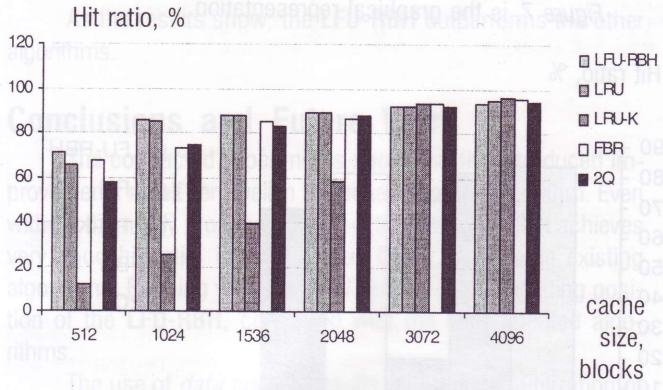


Figure 3b. The hit ratio (%) for experiment 1

blocks, due to data pooling the **LFU-RBH** achieves the results for only 3648 blocks, while the other algorithms use all 4096.

The experiment is conducted again for the last two cache sizes (3072 and 4096) for **RB**=16 bits, i.e. the length of the **References Buffer** is 65 536 references.

For **RB**=16 bits the **LFU-RBH** shows an improvement of performance. *Table 3* provides the results. *Figure 4* is the graphic for the last two cache sizes and **RB**=16. Again, the data pooling gives the utilization of 4032 blocks, instead of all the 4096.

Table 3. LFU-RBH with RB=16

	RB=14 bits	RB=16 bits
3072	92.14	92.89
4096 (4032)	93.62	95.23

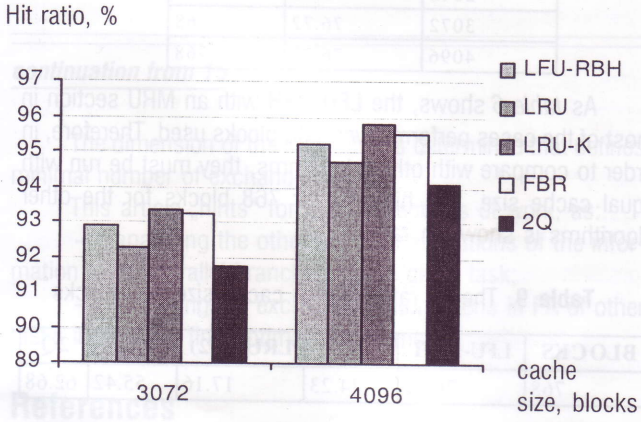


Figure 4. LFU-RBH with RB=16 bits

As the results show, the **LFU-RBH** behaves better with larger **RB**.

Experiment 2

The second experiment uses **RS2**. The parameters of the **LFU-RBH** are the same as for experiment 1. *Table 4* shows the results.

Table 4. The hit ratio, in %, for experiment 2

CACHE SIZE, blocks	LFU-RBH	LRU	LRU-K (2)	FBR	2Q
512	65.43	59.88		60.75	52.86
1024	78.96	78.31	11.09	77.73	68.77
1536	80.76	80.46	23302.12	77.48	76.65
2048	82.14	81.94	53.66	79.66	80.74
3072	84.32	84.28	77.99	85.26	84.06
4096	85.28	85.82	71.33	86.92	85.67

Figure 5 is the graphical representation.

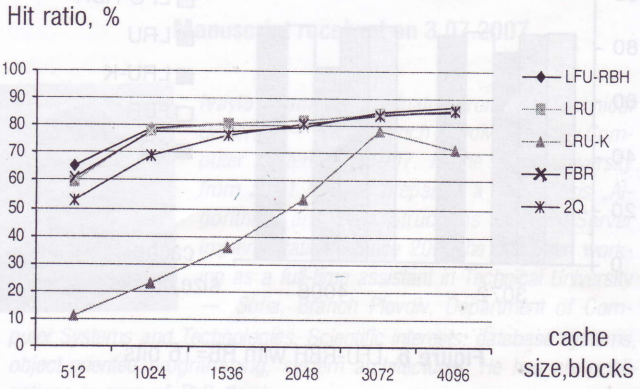


Figure 5a. The hit ratio (%) for experiment 2

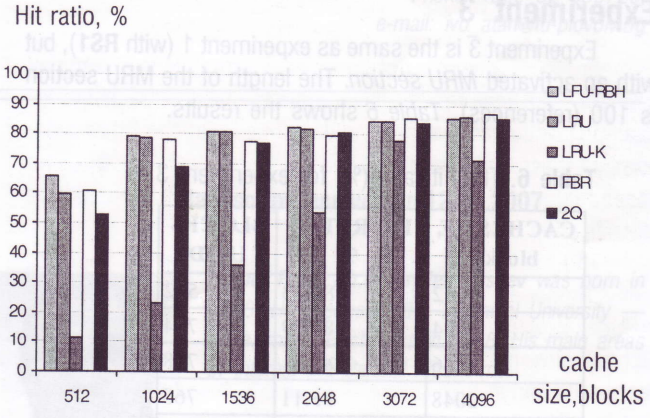


Figure 5b. The hit ratio (%) for experiment 2

As the results show, the **LFU-RBH** outperforms the other algorithms in most of the cases. Only for a cache size 4096 the **LFU-RBH** is outperformed. It is important to denote that for a cache size 4096 blocks, the **LFU-RBH** achieves the results for 3648 blocks, due to data pooling.

The experiment is conducted again for the last two cache sizes (3072 and 4096) for **RB**=16 bits, i.e. the length of the **References Buffer** is 65 536 references.

For **RB**=16 bits the **LFU-RBH** shows an improvement of the

performance. *Table 5* shows the results. *Figure 6* is the graphic for the last two cache sizes and **RB**=16. It is important to denote that for a cache size 4096 the data pooling results in 4032 used blocks.

Table 5. LFU-RBH with RB=16

	RB=14 bits	RB=16 bits
3072	84.32	85.25
4096 (4032)	85.28	86.99

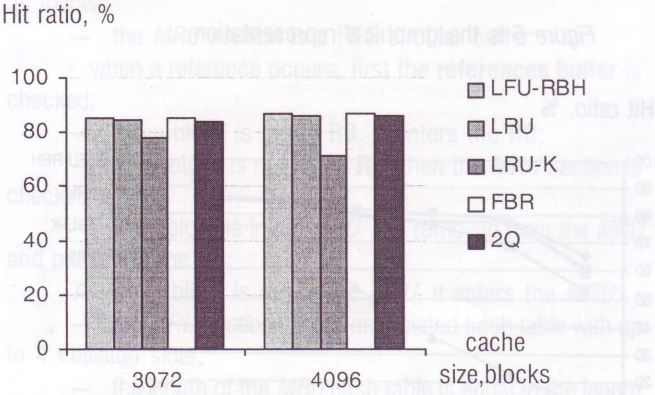


Figure 6. LFU-RBH with RB=16 bits

As the results show, the **LFU-RBH** for 4096 blocks outperforms other algorithms. It should be noted that due to data pooling the result for the **LFU-RBH** is actually for 4032 blocks.

Experiment 3

Experiment 3 is the same as experiment 1 (with **RS1**), but with an activated *MRU section*. The length of the MRU section is 100 (references). *Table 6* shows the results.

Table 6. The hit ratio (%) for experiment 3

CACHE SIZE, blocks	HIT RATIO, %	BLOCKS USED
512	81.03	512
1024	82.99	704
1536	83.11	768
2048	83.11	768
3072	83.11	768
4096	83.11	768

As *table 6* shows, the **LFU-RBH** with an MRU section in most of the cases performs with 768 blocks used. Therefore, in order to compare with other algorithms, they must be run with equal cache size. The hit ratio for 768 blocks for the other algorithms is shown in *table 7*.

Table 7. The hit ratio (%) for cache size 768 blocks

BLOCKS	LFU-RBH	LRU	LRU-K (2)	FBR	2Q
768	83.11	80.68	18.59	80.01	68.02

Figure 7. is the graphical representation.

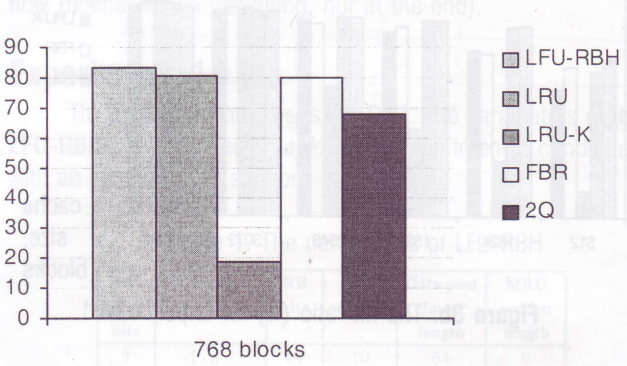


Figure 7. The hit ratio (%) for 768 blocks

As the results show, the **LFU-RBH** outperforms the other algorithms.

Experiment 4

Experiment 4 is the same as experiment 2 (with **RS2**) but with an activated *MRU section*. The length of the MRU section is 100 (references). *Table 8* shows the results.

Table 8. The hit ratio (%) for experiment 4

CACHE SIZE, blocks	HIT RATIO, %	BLOCKS USED
512	74.77	512
1024	76.60	768
1536	76.71	768
2048	76.72	768
3072	76.72	768
4096	76.72	768

As *table 6* shows, the **LFU-RBH** with an MRU section in most of the cases performs with 768 blocks used. Therefore, in order to compare with other algorithms, they must be run with equal cache size. The hit ratio for 768 blocks for the other algorithms is shown in *table 9*.

Table 9. The hit ratio (%) for cache size 768 blocks

BLOCKS	LFU-RBH	LRU	LRU-K (2)	FBR	2Q
768	76.72	74.23	17.16	55.42	62.68

Figure 8 is the graphical representation.

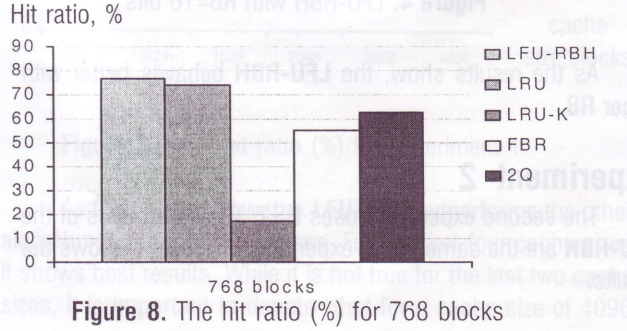


Figure 8. The hit ratio (%) for 768 blocks

As the results show, the **LFU-RBH** outperforms the other algorithms.

Conclusions and Future Work

The conducted experiments show that the introduced improvements result very well in the behavior of the algorithm. Even without the newly proposed MRU section the **LFU-RBH** achieves very good hit ratio, in most cases better than some existing algorithms. Running with the *MRU section* gives a leading position of the **LFU-RBH,** compared with the experimented algorithms.

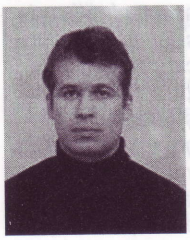
The use of *data pooling* results in a reduced utilization of the buffer memory. Also, the results for the different reference strings (**RS1** and **RS2**) state that the **LFU-RBH** carries better the simulation of a large area scan situation.

Many other experiments can be conducted to tune the algorithm and to balance between different parameters — hash bits, the length of RB, the number of sections, the length of MRU section and etc.

References

1. Atanassov, I. An Approach for Database Disk Buffering.International Conference of Young Scientists, Plovdiv, Bulgaria, June 2007.
2. Effelsberg, W., T. Haerder.Principles of Database Buffer Management. ACM Digital Library, 1984, ISSN:0362-5915.
3. Goh, C. L., Y. Shu, Z. Huang, B. C. Ooi. Dynamic Buffer Management with Extensible Replacement Policies. ACM Digital Library, 2006, ISSN:1066-8888.
4. Jiang, S., X. Zhang. LIRS: An Efficient Low Interreference Recency Set Replacement Policy to Improve Buffer Cache Performance. ACM Digital Library, 2002, ISBN:1-58113-531-9.
5. Johnson, T., D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. ACM Digital Library, 1994, ISBN:1-55860-153-8.
6. Lee et all. On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies. ACM Digital Library, 1999, ISSN:0163-5999.
7. O'Neil, E. J., P. E. O'Neill, G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. ACM Digital Library, 1993, ISSN:0163-5808.
8. Robinson, J., M. V. Devarakonda. Data Cache Management Using Frequency-based Replacement. ACM Digital Library, 1990, ISSN:0163-5999.
9. Shallahamer, C. A. All About Oracle's Touch Count Data Block Buffer Cache Algorithm. www.orapub.com, 2004.
10. Tanenbaum, A. Modern Operating Systems. Second Edition, Prentice Hall, 2001, ISBN 0-13-092641-8.

Manuscript received on 3.07.2007



Ivaylo Atanasov *graduated from the Technical University — Sofia, Branch Plovdiv, speciality Computer Systems in 1997. In the same university from 2001 he was preparing a PhD, thesis „Algorithms and Data Structures in Client-Server Implementation". Since 2003 he has been working as a full-time assistant in Technical University — Sofia, Branch Plovdiv, Department of Computer Systems and Technologies. Scientific interests: database systems, object-oriented programming, system architectures. He has six publications in area of PhD thesis.*

Contacts:
Department of Computer Systems and Technologies
Technical University-Sofia, Branch Plovdiv
Phone: +359 32 659729,
e-mail: ivo_atan@tu-plovdiv.bg

continuation from 15

The dimension of the submatrix is determined that defines minimal number of exchanged words.

This article „hints" for future directions of work, as:

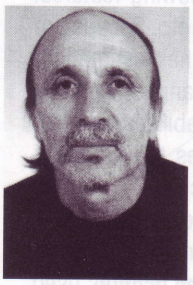
— analsying the other possible allocations of the information in the parallel branches of the given task;

— analysing the exchanging interactions in PA of other tasks in order to find optimal algorithms.

References

1. Seyed H. Roosta. Parallel Processing and Parallel Algorithms: Theory and Computation. Springer, 2000.
2. Cosnard, M., D. Trystram. Parallel Algorithms and Architectures. Thomson Computer Press, 1995.
3. Евреинов,Э. В., Ю. Г. Косарев. Однородные универсальные вычислительные системы высокой производительности. Новосибирск Наука, 1966.
4. Vasilev, N. Main Principles for Searching and Creating Parallel Algorithms, Information Technologies and Control, 2004, 2, ISSN 1312-2622.
5. Wilkinson, B., M. Allen. Parallel Programing. Prentice Hall,1999.

Manuscript received on 12.04.2007



Assoc. Prof., Ph.D Nayden Vasilev *was born in 1943. He graduated the Technical University — Sofia, major Electronics in 1968. His main areas of interest are Parallel Algorithms.*

Contacts:
Technical University of Sofia—branch Plovdiv
Plovdiv 4000,
25 Tzanko Dustabanov Str.
e-mail: mnvasilev@yahoo.com