Predictive Array Access Cache-PAAC71

Alejandro Villar Briones [1], Oscar Camacho Nieto [1],
Luis Alfonso Villa Vargas [2]

[1] Centro de Investigación en Computación-IPN, Laboratorio de Sistemas Digitales,
Av. Juan de Dios Batiz s/n, Col Nueva Industrial Vallejo, 07738, México DF.
albriones77@hotmail.com, osarc@cic.ipn.mx
http://www.cic.ipn.mx
[2] Instituto Mexicano del Petróleo IMP, Programa de Matemáticas Aplicadas y Computación,
Eje Central Lázaro Cárdenas 152, Col. San Bartolo Atepehuacan, 07738, México DF.
lvilla@imp.mx
http://www.imp.mx

Abstract. In this article a memory model appears cache associative of two routes with sequential access and a predicting one on the basis of stride. The development of this model is based on the observation of behavior of several models of memories cache of data, and removing the greater benefit to this behavior. The proposed model improves the behavior of the memory system (memory cache of data) diminishing the latency of access at levels similar to those of a memory of direct access or mapped, and the percentage of error (miss_rate) at levels similar to those of an associative memory. This is significant from the point of view of yield, since several machine cycles for an operation of access to memory are diminished, predicting its effective direction, with sufficient anticipation to its calculation, besides to reduce to the consumption of energy when avoiding unfruitful accesses to other blocks of the memory.

Keywords: memory cache, direct access, associative memory, prediction, stride, latency, number of cycles, hit_rate, miss_rate.

1 Introduction

The innovations in micro architectures have been conceived generally with the observation of the behavior of the programs. The designers frequently introduce new characteristics to micro architecture to operate the patterns of behavior of the program with the purpose of improving the yield of the processor.

Nevertheless the advance in the improvement of the yield of the memories so is not accelerated as in the case of the processor. Reason why it is necessary to explode characteristic of behavior of the memories with the purpose of reducing to the access time to them and the number of errors as far as data none found.

One of the main stages of the process of execution of a program is the compilation of values of the memory and its writing (instructions of load and storage load/store). In this process, it is possible to be taken advantage of the characteristic the space locality and the temporary locality [1]. Considering these concepts he is as the memory hierarchy is developed, which consists of different levels from memory with different sizes and speeds. Next to the processor it is the denominated cache memory.

At the moment several models of this cache exist that try to diminish the percentage of failure rate (*miss_rate*) by means of having one better policy available [2], handling of replaced data (sweepings or remainder of breaks it), and policy search of data compares or sequential. If to this we added to him that the percentage of failure rate by means of dividing the cache in several blocks can be diminished (asociatividad level), diminishes the time of delay of the processor by a data that it has to be extracted from the following levels of memory when diminishing the latency (cycles by instruction). The problem that presents/displays these types of schemes, when dividing the cache memory in several blocks, is that the time search of the data is increased, since is duplicated in comparison with a memory of a single block or direct mapped.

When an associative memory is referenced, the blocks that conform it is examined in parallel form. When the labels of the blocks have been read, a label comparator selects the block that contains the wished data, and passes the data of the block to the processor. The selector of labels introduces an extra retard between the rating of the block that contains the wished data and the moment in that the data by the processor can be used. This among other factors increases the time search of a data in this type of memories. The figure 1 shows the graph in which it is compared as the memory of direct access has smaller latency of access in almost 45% with respect to the memory of associative type level two.

But on the other hand the associative memory always has demonstrated to have *miss_rate* lower than the one of a memory of direct access, in figure 2 is the difference of *miss_rate* between these two models.

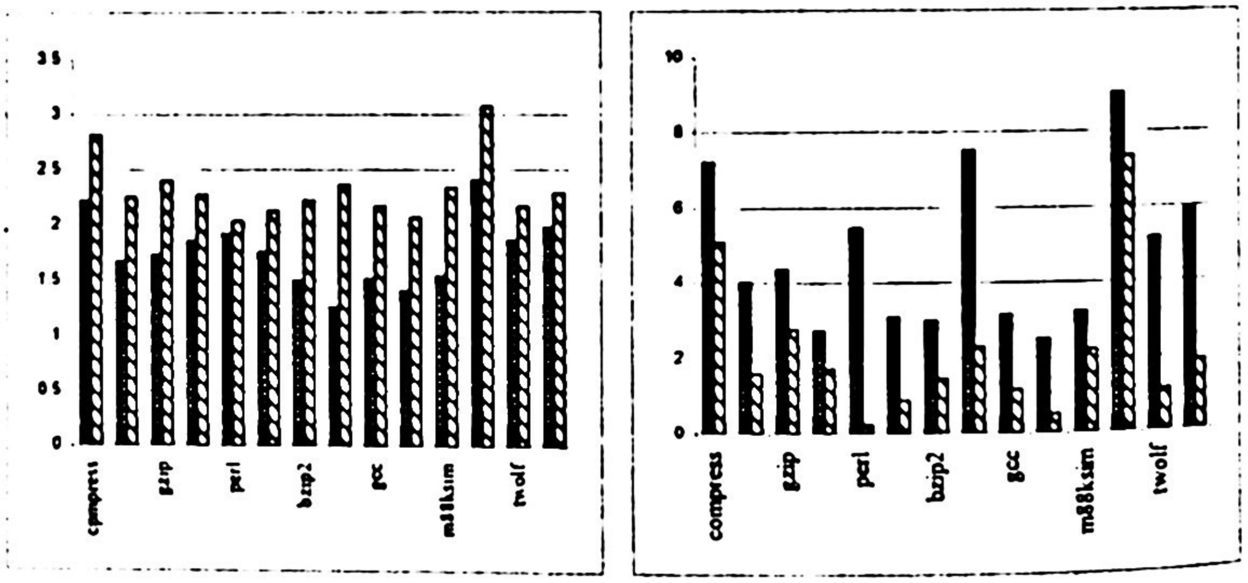

Fig. 1,2 Latency in cycles and miss_rate (*Direct Mapped* ■ vs *Associative cache level two* ▨)

The design of a memory that combines the advantages of both models, associative direct mapped and, has been the objective of many investigators. This memory must have so much *miss_rate* as a latency low, is to say not to try to obtain efficiency to sacrifice the speed of operation of the memory. In order to obtain this objective different models from access have considered, policies of error handling and replaced information (sweepings) and models of prediction of location of the data.

In this articulate, we propose a data cache memory of associative type level two, with sequential access to its blocks, but with a predictor of data's location in some of the blocks. The model which we propose uses a predictor on the basis of stride, with a saturated accountant of two bits (4 states) who allows a greater percentage of guessed right predictions, implementing itself in an associative memory of level two with sequential access. This technique offers *miss_rate* of an associative memory of level two, but with an access time even smaller than a memory of direct access.

2 Antecedents

The implementations of cache memories of sequential type or serial they can be divided in two categories. First, they can be divided in the form in which they are examined at the time of making a data search, since this can be in a static and rigid order or of a dynamic form, making the first search not necessarily in the first block, is to say not always begins the search in the first block. Second, this type of memories can be divided by the type of policy that uses to write their data in each one of the blocks, to read them, or to retire them of such.

The cache memory *Hash-Rehash* type (HR-Cache) propose by Agarwal [3] reduces *miss_rate* of a memory of direct mapped, and although this model can implement the associative level that the designer proposes, available has a performance below an associative memory of level two with policy LRU as far as *miss_rate*. This model consists of the use of a memory of direct access that is acceded by means of two Hash functions that determine a first block of access and subsequent blocks according to the number of Hash functions that are implemented.

The cache memory model *Column-Associative* type (CA-Cache) of Agarwal and Pudar [4], improves *miss_rate* and the access time of HR-Cache; being based on an operation similar to HR-Cache. CA-Cache, it makes use of two blocks, if when making a search in the first block does not locate goes to the second block, and later changes the entrance of this second block towards first, so that later searches locate to the data in the first block diminishing the access time.

Nevertheless the use of this technique implies an extra consumption of energy to exchange the input data of both or more blocks than are implemented. As far as *miss_rate* it continues being higher than the one of an associative memory of level two.

Model that combines the advantages of the techniques before mentioned is proposed by Calder and Elmer [5], Predictive Sequential Associative Cache (PSA-Cache), which instead of handling a policy of relocation of data it leans of a bit that the possible location of the data identifies, which diminishes the consumption of energy of the CA-Cache' memory, and in which use is made available of a policy MRU and a table of a denominated bit table of steering or direction, with which it determines the dynamic block to which the first search must accede when making.

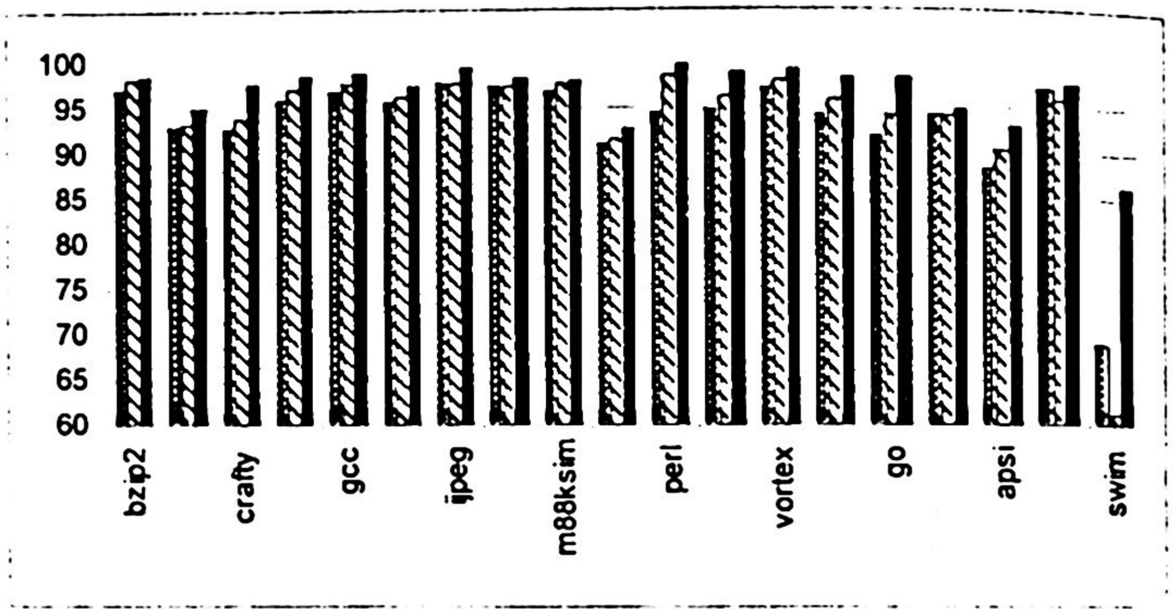

Fig. 3 Hit_rate % *(Direct Mapped* ■, *PSA-Cache* ▨ and *Associative Cache Level two* ■)

3 Predictive accesses to arrays Cache (PAAC)

Following the tendency to conform the operation of the hardware of the processor to the behavior of the programs, is necessary to detect the behavior of the service loads on the memory models cache that there is in the market. In figure 3 in the graph one of the most recent models of associative memories of sequential access is observed, an associative memory of level two and the traditional memory of direct mapped. The last has a better behavior.

The idea to develop a memory of sequential access arises to determine the internal behavior of the associative memory, since a great number of successes is located in single one of the blocks that conforms it. Figure 4 shows as more of 90% of the successes they are located in block one of the associative memory.

Considering that the access time of an associative memory of level two is two cycles, the one considers that the search of the data in the memory is made simultaneously in both blocks, which consumes a first cycle of clock and later to compare the values that throw each one of the blocks and selecting valid data by a multiplexer a second cycle of clock.

If all the accesses went to first a block one more than the ninety percent of the accesses it would be obtained in a cycle of clock instead of two, in case of not finding the data in this block the access to the second clear block would become this with a penalty of an additional cycle, nevertheless a percentage of error still below the one of a memory of direct mapped or a single block is obtained.

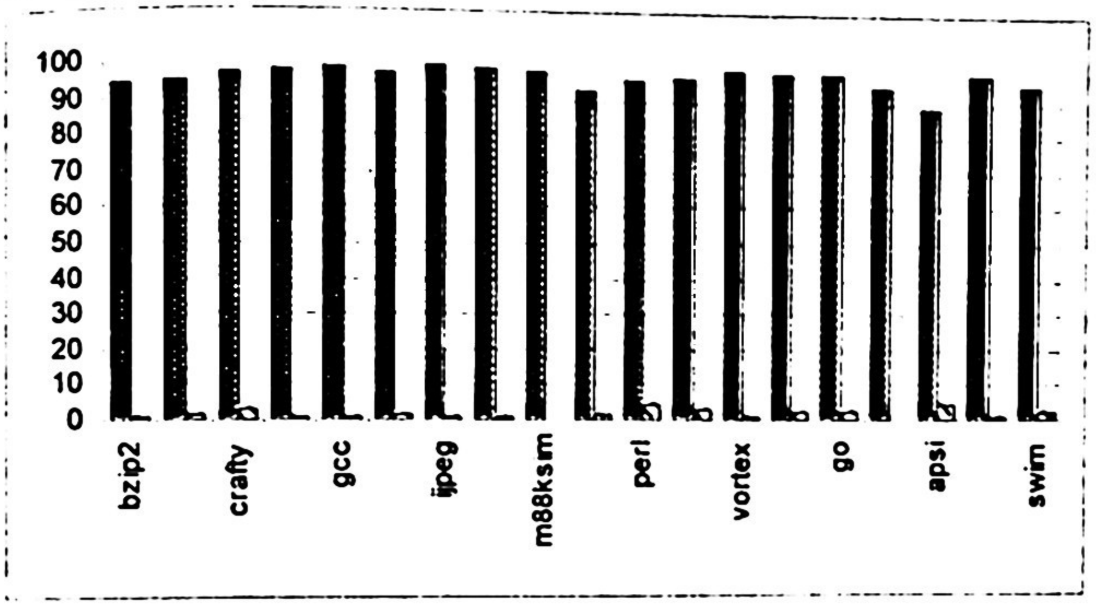

Fig. 4 Hit_rate (%) for each block in an associative cache memory of level two (*AS-N2, block one* ■ , *block two* ▨)

As this behavior is easily predictable thought about the use of a predictor on the basis of stride. The predicting one on the basis of stride, in its simpler form predicts the next value by means of determining the behavior of the accountant of the program. The basic idea is to predict future references to the memory on the basis of the passed history for the same instructions of access to memory.

The more intuitive scheme of prediction is the one than it has several iterations of the same search with previous events. This is, when the Program Counter (PC) decoded an instruction of load/store, reviews the entrances of a reference table to see if a file of that instruction exists. If it does not exist, an entrance of this table is used to store the file of this instruction. In case of existing the entrance and the reference for the following iteration he is predictable, is made a pre-search. This basic scheme surrounds the use of the *PC* and has a table to store the file of instructions of load/store that have acceded to the cache memory of data. Nevertheless for our case this table will be used to determine to that block was made the search of the asked for data.

3.1 Table of prediction reference

The table of reference of prediction, organized like a memory Cache of instructions, maintains the file of previous references or accesses to the memory of data and associate *stride* to them (distance between instructions according to the *PC*) for each instruction of load or storage. The entrances of the table are conformed by the following elements:

- *Tag.*- corresponds to the direction of the instruction of load/store.
- *Last direction (operation).*- that was referenced when the PC looked for the instruction
- *Stride.*- the difference between the two you complete directions that were generated
- *State.*- Two bits that serve like accountant to establish a file of three states of the accesses of the same instruction. To the beginning, the first access accountant in one indicates an initial condition, the second time the accountant is increased and situation is considered in which it is possible to make prediction of the location (block) of the following asked for value, and in the third state a condition of a prediction with high percentage of certainty in which occurs if the prediction is made.
- *Block.*- in this field the location of the data is stored, being first or the second block of the memory. This field is updated after collecting the data of the memory and to corroborate its location

The size of the prediction table originally considered of a size of 32 entrances, nevertheless, when increasing the size of this table has an elevated percentage of prediction more.

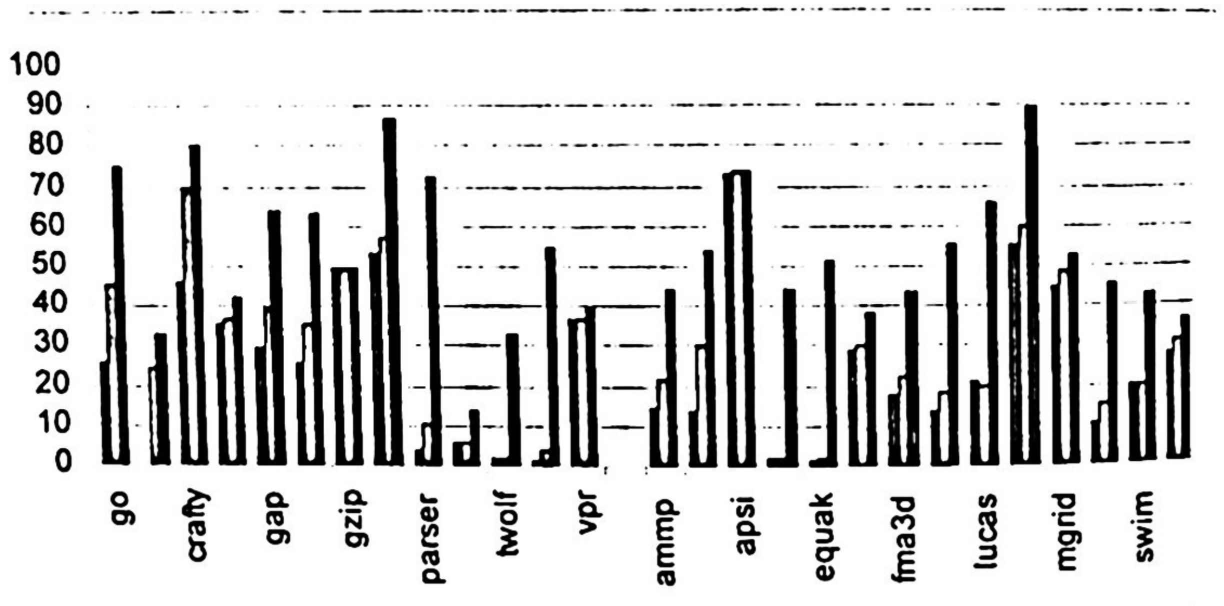

Fig. 5 Hit_rate of prediction (*32 entries* ■, *128 entries* ▨, *1024 entries* ■ , SPEC2000)

Obviously the percentage of erroneous prediction increases, but it is not in the same proportion that the guessed right prediction does. In figure 5 the behavior of the predicting one for different sizes is appraised from the table of prediction with the SPEC2000.

When handling themselves a robot of prediction based on a behavior of stride, and having a so great table, as is it of 1024, no longer takes single the accesses with a very repetitive behavior, but which it also considers other accesses with no longer so repetitive behaviors, which gives foot to have prediction errors, therefore was decided to handle a table of a size of 128 entrances, with which according to the obtained results it is possible to be covered most of the accesses with repetitive behavior

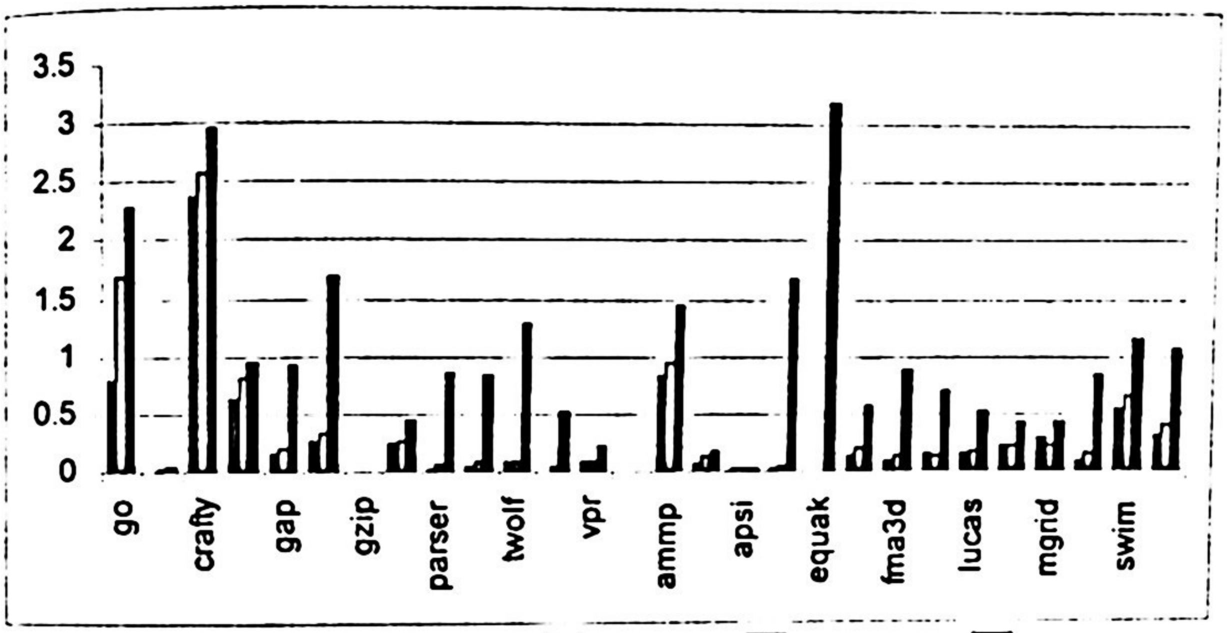

Fig. 6 Miss_rate of prediction (*32 entries* ■, *128 entries* ▨, *1024 entries* ■, SPECINT)

3.2 Predictor on the basis of stride

The basic mechanism in the prediction on the basis of stride is to keep the effective direction from the operands of memory, it calculates stride for the access, and activates the field of State to verify if the following reference by means of comparing previous stride can be predicted kept with which it finishes calculating. Stride is obtained by means of taking the difference between the effective directions from the two but recent accesses by the same instruction.

The prediction says that she is correct when the direction that comes by the normal flow is equal to the sum of stride calculated but previous stored address, of to be certain this condition is not predicted and the condition of the field is modified state. When appearing a true condition is increased the accountant of the field state, but if the condition is false decrements this accountant.

On the basis of the value of this accountant the predicted value is used to accede to the correct block of the memory breaks in where is the stored data.

In accountant who handles itself for the *State* field, the first state corresponds with a condition of boot of the prediction table activating an entrance of the table. The accountant changes to the following state whenever the result of stride is similar to the kept one in the first state; in this single state the values are updated and the value of the prediction table is not taken into account. The third state occurs at the time of repeating stride again and now if the value of the field is taken into account block to accede to the predicted block. The fourth state considers a state of very high probability of guessing right to the block that contains the data, since the behavior pattern has repeated at least three times previously (three previous states).

Nevertheless when making tests and comparisons between simulations have been that when diminishing the value of the accountant of this robot is increased the percentage of prediction in each one of the simulations. Instead of which the prediction is made in state three of the accountant now is made in the second state. Figure 7 shows the operation of the accountant.

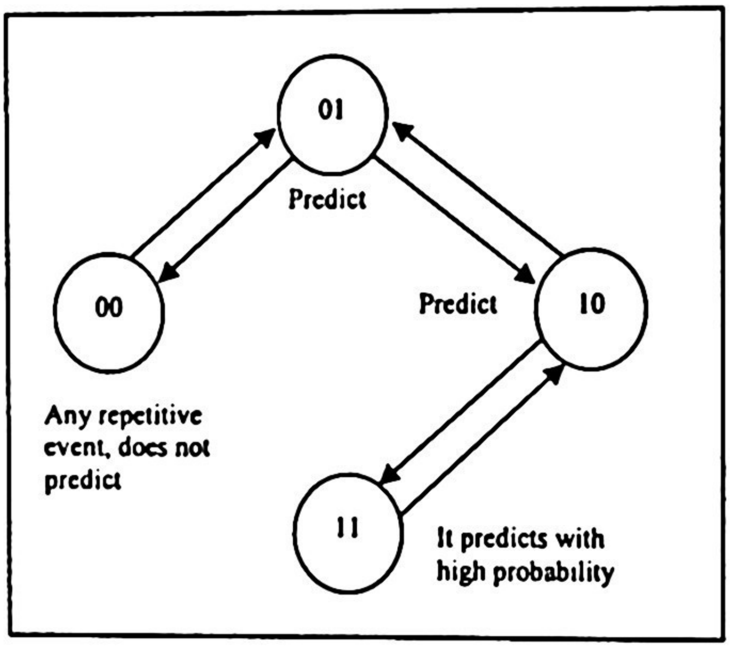
01
Predict
00
Predict
10
Any repetitive event, does not predict
11
It predicts with high probability

Fig. 7 Counter for state

SPEC	Pred. 2° state. (%)	Pred 3° state. (%)
Go	45.55	18.22
Bzip2	24.6	14.76
crafty	69.44	34.72
Eon	36.76	29.408
Gap	39.32	23.592
Gcc	35.51	10.653
gzip	49.24	44.316
mcf	57.46	34.476
parser	10.69	5.8795
vortex	3.89	1.167
Vpr	37.12	14.848
ammp	21.57	6.471
applu	30	15
apsi	73.59	51.513
facerc	30.38	18.228
fma3d	22.11	13.266
galgel	18.54	10.197
lucas	19.88	11.928
mesa	59.67	38.7855
mgrid	48.65	14.595
sixtrack	15.31	9.186
swim	19.65	11.79
wupwise	30.95	18.57

Table 1. - Percentage of hit_rate of prediction

The reason of using the value of the field block in the second state of the accountant is because many of the accesses made to a same direction of memory are repeated of constant form, if the robot allowed to predict single as of the third time that appears the same pattern of behavior was let go an event that can very possibly be predicted with a high level of certainty. If the prediction is allowed from the second state of the accountant, this characteristic takes advantage of the behavior of the hardware and an advantage of 40% in possible correct predictions over the previous model is obtained. Table 1 presents the data in percentage of *hit_rate* of prediction as of the second and third state of the accountant.

The PAAC altogether this conformed by a table of prediction, a form of sequential access and a politic of replace LRU (Fig. 8). The access to the PAAC is made of the following way: if the prediction cannot be always made access to the first block of the memory, since it demonstrated (Fig. 4) that most of the data they lodge in this; if the data is found in this block had a latency of a cycle of access, which is similar to a memory of direct mapped, but of not finding the data is acceded to the second block, which brings like consequence a latency similar to the one of an associative memory.

The way that is followed to predict the block in which the looked for data is located initiates with the use of the *Program Counter* (PC) of the instruction of access to the memory, while by a side the effective direction of the memory by means of a sum calculates (it consumes at least a cycle), we indexed our table of prediction of 128 entrances, of which the value of the accountant for that entrance is verified. If the state of the accountant allows the prediction block in that same entrance of the table accedes to the block indicated by the field. In case of success an access in a time of a cycle is obtained, similar to the one of a memory of direct mapped, without mattering if the data were in the primary or secondary block. In case of a prediction error it is acceded to the block in opposition to the one of the prediction, which tares like consequence a penalty of an additional cycle to which already uses to review the predicted block. The filling of the prediction table is made by means of comparing stride present with stride stored in the prediction table.

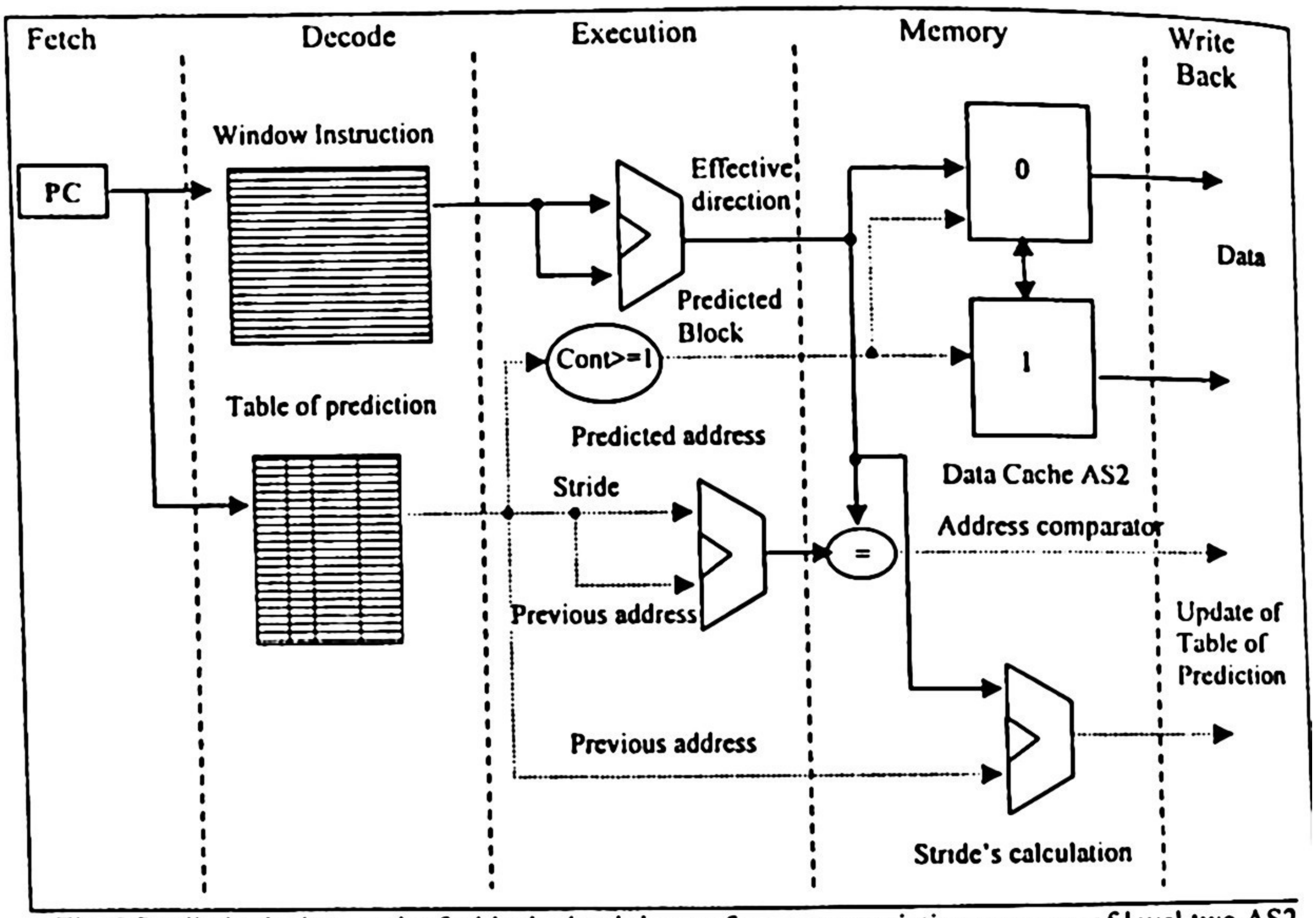

Fig. 8 Prediction's data_path of a block, that it is part from an associative memory of level two AS2 (normal flow $\longrightarrow$, prediction's flow $\dashrightarrow$).

Stride is obtained from the difference of the previous direction with the present direction, operation that can be conducted single until the calculation of the effective direction finishes.

In addition a comparison between the predicted direction (extreme of previous direction with stride stored) and the effective direction is made. If both comparisons guessed right, when comparing both stride is increased the accountant of the entrance the table, besides to verify in that block was the data within the cache memory, to updat the field block of the table.

In the next access to the memory the prediction table is verified and on the basis of accountant of events of stride and of the field block it is possible to predict in which from the memory is stored our data.

If the direction of the data is predictable accedes to the block indicated by the prediction table, if the data is located in this block, conserves the latency of direct mapped, but if not finding the data it is acceded to the other block in search of the data. If the prediction were successful and the data was located in block two represent that is eliminated the additional cycle to have to look for the data in the first block of sequential form and soon to accede to block two. When we have a huge prediction's percentage it can be used to reduce the access time to the memory.

4 Methodology of Evaluation

We compared the behavior of different models from cache memories using simulations of service loads, for it was used the Simplescalar [6] We obtained measures from 20 programs, from SPEC95 and SPEC2000. The models simulated memories were: memory of direct mapped (MP), memory Column-Associative type provided with a predictor on the basis of stride (CA-Pred), an associative memory level (two AS-2) and our model (PAAC); all the models with sizes of 8K, 16K, 32K and 64K. To determine performance of memory it is used the formula presented.

$$\text{Performance} = \frac{K + L}{M}$$

K = (hits in block one) + (hits in block two) + (hits by prediction)
L = (miss in any block) + (miss in prediction)
M = Total number of cycles for execution

The latency of success for a direct mapped memory is 1 cycle, for an associative memory level two is of 2 cycles, for our model it is one cycle for block one and predicted hits, two cycles for block two (sequential access). The error latency is 18 cycles for the three models. At last all the access (hits and misses) are added and divided by total number of cycles of benchmark execution.

5 Results

The index of errors for our model (PAAC) is similar to the one of an associative memory of type, since it leaves from the same principle of positioning of data, but the advantage that presents/displays is the access time, table 3 shows the obtained access times for each one of the models. In table 3 the smaller index of latency for the PAAC, even lower is emphasized than the one of a memory of direct mapped of the same size. Even making comparisons between the number of cycles of latency between PAAC and one of direct mapped, a PAAC of 16K has a smaller latency that a memory of direct mapped of 32K, this point is excellent considering that like the size of our storage device is increased the index of miss rate is reduced.

Spec	MP	CA Pred	AS2	PAAC
applu	5.74	5.77	5.39	5.39
apsi	11.88	9.6	7.11	7.11
bzip2	2.99	1.79	1.46	1.46
compress	7.197	6.7	5.1	5.1
crafty	7.497	6.05	2.29	2.29
gap	4.02	2.80	1.60	1.60
gcc	3.13	2.26	1.14	1.14
go	8.11	5.74	1.67	1.67
gzip	4.348	3.76	2.76	2.76
ijpeg	2.49	2.20	0.49	0.49
li	2.72	2.66	1.72	1.72
M88ksim	3.20	2.49	2.20	2.20
mcf	9.00	8.64	7.34	7.34
mgrid	2.96	3.97	2.36	2.36
perl	5.45	1.38	0.23	0.23
swim	31.13	38.91	13.73	13.73
twolf	5.15	3.76	1.11	1.11
vortex	3.07	1.93	0.87	0.87
vpr	5.9	4.09	1.84	1.84

Table 2.- Percentages of miss_rate (%)

SPEC	MP	CA_Pred	AS2	PAAC
applu	2.16	2.34	3.12	2.01
apsi	2.28	2.62	3.6	2.12
bzip2	1.51	1.59	2.22	1.25
compress	2.22	2.42	2.81	1.88
crafty	2.26	2.25	2.35	1.43
gap	1.67	1.77	2.25	1.29
gcc	1.52	1.68	2.17	1.21
go	1.78	2.01	2.5	1.5
gzip	1.73	1.83	2.43	1.48
ijpeg	1.41	1.66	2.07	1.15
li	1.86	1.69	2.27	1.3
M88ksim	1.54	1.72	2.34	1.42
mcf	2.41	2.87	3.07	2.16
mgrid	1.38	1.72	2.11	1.2
perl	1.92	1.7	2.03	1.1
swim	2.9	3.28	4.02	2.9
twolf	1.87	1.78	2.17	1.23
vortex	1.65	1.76	2.13	1.16
vpr	1.99	2.02	2.29	1.35

Table 3. - Index of latency (cycle)

6 Conclusions and futures works

On the basis of the predicted results and to the obtained ones, we can be made comparisons of the degree of effectiveness of proposed model. In the case of the area predictions of location of data it is possible to be continued implementing improvements as far as possible in the prediction, since in spite of having a percentage of 94% effectiveness, we diminish the *miss_rate* of prediction with the propose of increasing the performance of our model. The application of this model in associative memories with greater number of blocks is the next point of study and to evaluate its yield, also possible implementation in language VHDL for its simulation at level layout to determine its power consumption.

References

1. J-k. Peir, Y. Lee, and W. Hsu, "Capturing Dynamic Memory Reference Behaviour with Adaptive Cache Topology," Proc. 8th ASPLOS, 1998.
2. J.W. Fu, J.H. Patel and B.L. Janssens, "Stride Directed Prefetching in Scalar Processors", Proc of the 25th Int. Symp. On Microarchitecture (MICRO-25), pp. 102110, 1992
3. Anant Agarwal, John Hennesy, and Mark Horowitz. Cache performance of operating systems and multiprogramming, ACM Transactions on Computer Systems, 6:393-431, November 1988.
4. Anant Agarwal and Steven D. Pudar. Column-Associative caches: A technique for reducing the miss_rate of direct mapped caches. In 2oth Annual International Symposium on Computer Architecture, SIGARCH Newsletter, pages 179-190, IEEE, 1993
5. B. Calder, and D. Grunwald, J. Elmer, "Predictive Sequential Associative Cache", Proc. 2nd HPCA, 1996.
6. Todd Austin, Eric Larson, and Dan Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," IEEE Computer, February 2002.
7. J-L Baer and T-F Chen, "An Effective On-Chip Preloading Scheme To Reduce Data Access Penalty", ACM, ICS 1991
8. Alan Jay Smith, "Cache Memories", Computing Surveys, 14(4): 473-530, September 1082.
9. M. Hill, A. Smith, "Evaluating Associativity in CPU Caches", IEEE Trans Computers, Vol. 22(12), Dec. 1989
10. M. Jonsn, Sperscalar Microprocessor Design. Englewood Cliffs, New Jersey; Prentice Hall, 1990.
11. David A. Paterson, John L. Hennessey, Computer Organization and Design: The Hardware/Software Interface, Second Edition, Morgan Kaufmann Publishers, 1997.
12. Deszö Sima, Terrence Fountain, Peter Kacsuk, Advanced Computer Architectures: A Design Space Approach. Addison Wesley, 1997
13. Norm Jouppi. Cache writes policies and performance. In 20th Annual International Symposium on Computer Architecture, SIGARCH Newsletter, pages 191-201, IEEE, May 1993.