

Short Paper

A New Class of Cache Memory Controllers

ADNAN SHAOUT AND LARRY COLAGIOVANNI
*The Electrical and Computer Engineering Department
The University of Michigan-Dearborn
Dearborn, MI 48128, U.S.A.*

There is a lack of flexibility in a fixed page location system because every location within a page has a determined mapping within a page buffer. This causes the buffer to contain locations that are unlikely to be used. A more flexible system introduced in this paper, called variable page location, allows a page to begin at the first location needed so that the page buffer will be filled with sequential locations that are likely to be used.

This is not easily accomplished or practical if a conventional replacement policy, such as FIFO or LRU, is used in a variable page location system; therefore, a new class of replacement policies, called location policies, is introduced. To see how well a location policy system compares to a conventional system, a performance model for each system is developed. Both models are a function of cache size, program size, and program behavior. As a preliminary step for performance modeling, a stochastic model of program behavior based on the difference between successive word references is presented.

Keywords: memory replacement policies, cache, performance analysis, program behavior, models, hit ratio, LRU, FIFO, OPT, FCRP

1. INTRODUCTION

A replacement policy tries to anticipate the needs of the CPU which is controlled by program behavior; therefore, when a cache system is designed, program behavior should be taken into consideration [1-4]. There are many different types of programs. Fortunately, most of them exhibit the temporal and spatial characteristics of locality of reference [5-7].

1.1 Existing Approaches to Cache Design

The techniques for implementing a cache memory system in multi-user and multi-processor environments have been well understood for some time [8-10]. Techniques such as fully associative, set associative, and direct mapping schemes are the most popu-

Received May 18, 1999; revised January 18, 2000; accepted May 22, 2000.
Communicated by Shing-Tsaan Huang.

lar ones. Each technique usually involves a trade-off between implementation and hit ratio and, to evaluate the three techniques, two key areas of implementation that add to the cost and complexity are the first is hit/miss determination and the second is replacement determination.

Table 1 is a qualitative summary of the existing cache techniques [11, 12] without taking cache size into consideration.

Table 1. Qualitative summary of existing cache techniques.

	Hit/ Miss Determination	Replacement Determination	Hit Ratio
Fully Associative	4	LRU = 4 FIFO = 1	LRU = 1 FIFO = 2
2-Way Set Associative	3	LRU = 3 FIFO = 1	LRU = 2 FIFO = 3
4-Way Set Associative	2	LRU = 2 FIFO = 1	LRU = 3 FIFO = 4
Direct Mapping	1	1	4

Rated on a scale from 1 to 4. (Best = 1)

1.2 Variable Page Location Design

The motivation behind this study was to develop a technique that would enhance cache performance for microcomputer applications and architectures. However, the models derived may be utilized elsewhere. To achieve higher performance, variable page locations in main memory are utilized. The idea behind variable page locations is simple. Since locality of reference says that sequential locations have greater probability of being accessed, every time a miss occurs, it should be beneficial to cache performance if a page is buffered starting from the first location needed [13, 14].

Each page of main memory has a starting location (L). Since there are m words in a page, a page ends at location $L+m-1$. For n pages in cache, there is a set of n starting locations $\{L_1, \dots, L_n\}$.

A diagram showing how main memory locations are mapped to cache is shown in Fig. 1. Fig. 1 can be considered a snapshot in time of the memory hierarchy and is meant to show which locations of main memory, the bottom half of the diagram, exists in each cache page frame, the top half of the diagram. We can also see which locations do not exist in cache, i.e., between the frames, which gives us a sense of distance from frame to frame.

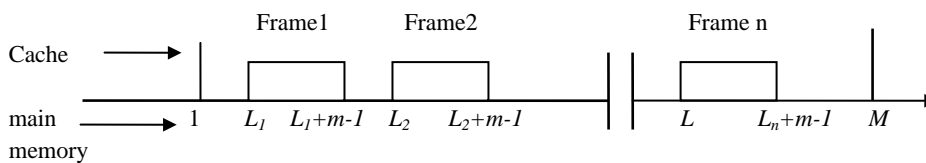


Fig. 1. Horizontal representation of a memory hierarchy.

The advantage here is that both fixed page and variable page configurations can be shown using the same diagram. In a fixed page configuration, the page frames in Fig. 1 can overlay main memory only at multiples of the page size, while a variable page configuration allows the page frames to overlay main memory at any number of locations. Using the parameters for a typical memory configuration found in some microcomputers, given in Table 2, an example snapshot in time can be given for both configurations.

Table 2. A sample memory configuration.

Word size	=	1 byte
M	=	1024 (1K) words
n	=	64 page frames
cache size	=	64K words
main memory	=	2^{20} (1M) words

In this example, the set of page frame starting locations $\{L_1, \dots, L_{64}\}$ for a fixed page configuration can have values that are a factor of some multiple of 1024, e.g., $L_1=1$, $L_2=2049$. This is not true for a variable page configuration where L_1 could be 50 while L_{64} could be 1086. How do you use conventional replacement policies, such as FIFO, in a variable page configuration without getting overlap of information in cache? The answer is you can't do it easily or practically, therefore, a new class of replacement policies, called Location Policies, will be introduced in this paper. Location Policies are heuristic algorithms based on the location of the page needed in main memory with respect to the locations of the pages already in cache [14]. Because Location Policies are heuristic, they have the advantage of using less overhead by not maintaining statistics of program behavior. Location Policies are easily understood by visualizing the memory hierarchy as in Fig. 1. Just looking at Fig. 1 can develop many heuristic rules. For example, one rule may be to replace a page frame that contains main memory locations directly below the page needed. By viewing Fig. 1 each time a miss occurs, a visual sense of cache performance can be achieved. Location Policies are not exclusive to a variable page location configuration, i.e., they can easily be implemented in a fixed page configuration without any coherence problem.

The objective of this paper is to show that a location policy, implemented in a variable page location configuration, can improve the overall cache performance while decreasing, in some cases, implementation overhead. A preliminary step in achieving this goal is adopting a model for program behavior. A new model for program behavior, called the next reference distance, will be introduced and explained in section 2.

In section 3, a location policy called forward circular replacement policy (FCRP) will be introduced. In section 4 the next reference distance model will be used as a part of another model that predicts the miss ratio for both fixed and variable page location configurations. The last part of cache analysis is simulation, which is covered in section 5.

2. PROGRAM BEHAVIOR

Probably the most widely used model for program behavior is the working set model [2, 8, 9, 15]. Although its not the choice for this paper it is still worth mentioning and explaining. A working set is the set of pages that are most recently used by a program. As the phrase most recently used indicates, a working set is defined over some period of time. Specifically, the symbol $W(t, T)$ denotes the working set where $(t-T, t)$ is the interval of time.

2.1 Next Reference Distance Model

By recording all the relative distances of a program, i.e., for every t , both the temporal and spatial behavior of a program can be mapped onto a two dimensional plot like the one in Fig. 2. Temporal behavior will show up as negative jumps due to a looping process while the spatial nature (arrays, subprograms, etc.) will show up mostly as positive jumps. The variable P , defined as the probability that the next reference will be one above the current, will denote the sequential component.

In making the transition from a frequency distribution to a probability distribution it will be assumed that the probability distribution is stationary. Since only the sequential component of the mapping will be exploited, main memory will be viewed in a more convenient way. Fig. 3(a) shows a picture of main memory wrapped in a circle with an arrow pointing in the clockwise direction indicating increasing memory locations.

This method maps all negative jumps in Fig. 2 to positive jumps by translating the negative half of Fig. 2

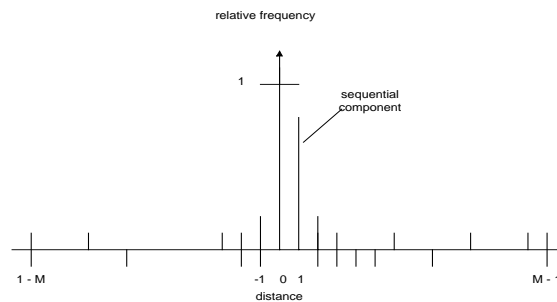
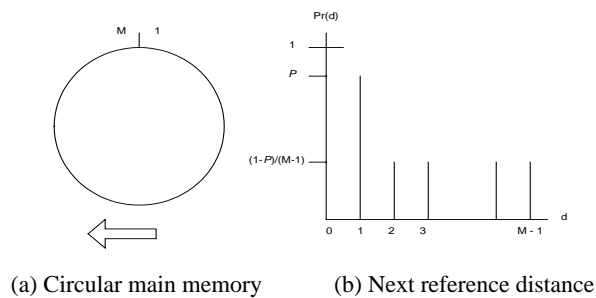


Fig. 2. Relative frequency vs. distance.



(a) Circular main memory (b) Next reference distance

Fig. 3. Forward direction mapping.

onto the positive half and adding the overlapping components.

A further assumption is that for a large program, i.e., many references, all the other components will be very small and uniform with a probability $(1-P)/(M-1)$. Fig. 3(b) shows the final model of probabilistic program behavior (next reference distance) that will be used in the performance evaluation of a cache memory. As shown in the figure, $P = \text{Pr}(1)$ and $\text{Pr}(d) = (1-P)/(M-1)$ (for $d \neq 1$) where $\text{Pr}(d)$ is a probability function of distance.

3. LOCATION POLICIES

Since programs are sequential by nature, then it would be advantages to load sequential addresses when a miss is encountered. In a fixed page configuration this is not always possible. For instance, the CPU may reference the last word in page x of main memory, therefore, there is a good chance the program will need page x for that reference only. If the page is large, then the CPU will waste a lot of time loading page x for only one reference. In a variable configuration the page can start with any reference thereby taking advantage of the programs sequential nature.

3.1 FCRP Explained

As the algorithm is described its adaptive behavior will be seen. First take another look at the memory hierarchy, in Fig. 4, to see where misses can take place.

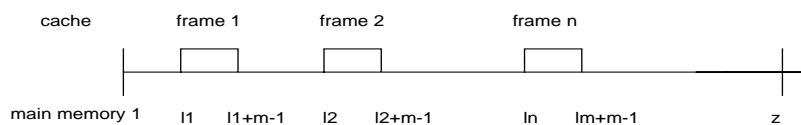


Fig. 4. Contiguous application with z location.

Notice that z (active memory) has replaced M because, as discussed earlier the total memory is not a concern. Only the portion that is used by the contiguous application, which has been re-indexed for convenience, is of concern.

There are two types of misses of interest. The first deals with a miss between any two page frames or below the first page frame. Fig. 5 shows the location of a miss between frame i and frame j . The rule for replacement is to take the frame above the miss as shown in Fig. 5.



Fig. 5. A miss between two frames.

The second type of miss deals with misses above the highest frame in cache. The rule here is to take the lowest frame for replacement. This situation is shown in Fig. 6.

Notice that a circular configuration memory hierarchy really has only one rule for replacement, and that is to move in the forward direction from the miss location and replace the first frame encountered. Thus, the term “forward” in FCRP. The replaced frame then is loaded with the miss location plus $m-1$ contiguous locations. Notice that FCRP insures no overlap of data in cache, i.e., no coherency problem, which is an advantage in implementation.

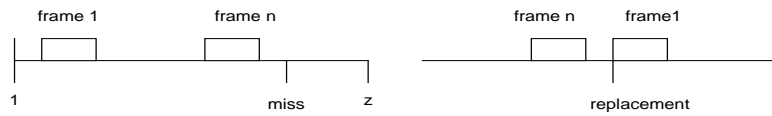


Fig. 6. A miss above the highest frame.

3.2 FCRP Implemented

There are three important considerations when implementing a cache algorithm. Two of them, hit/miss and replacement determination, have already been mentioned in section 1.1. The third, initialization, is not as important but is addressed here for the sake of understanding the software simulation presented in section 5.

First, let us consider initialization. This can be done in many ways. One way is to initialize the page frames so that they are contiguous starting from the first location requested by the CPU. This is the method used in section 5. Even though the cache does not contain any information, neither does the main memory. Initially, write operations have to be performed, e.g., loading the operating system, so that the frames will be positioned accordingly.

As discussed in section 1.1, a tag register is needed for each page frame to identify which pages are in cache (hit/miss determination). Also needed are registers and logic for replacement determination as required by the replacement policy. Each tag register will contain the starting address of a page of main memory. Aside from the tag registers, another register is needed for the next memory reference address. This register will be denoted by R and the tag registers by R_1, R_2, \dots, R_n . Fig. 7 is a block diagram for an FCRP hardware cache controller.

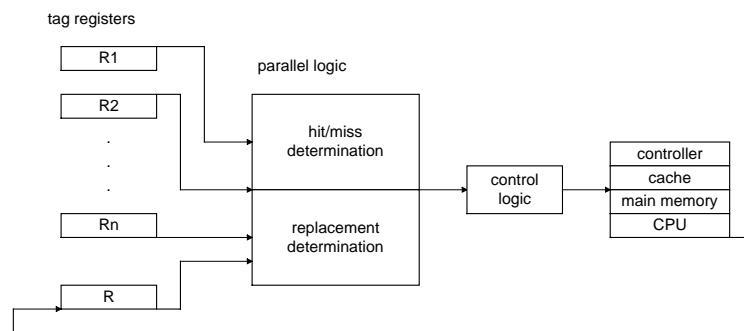


Fig. 7. Block diagram for an FCRP hardware cache controller.

As shown in the Fig. 7, both hit/miss and replacement determination are in parallel and use the same set of registers to generate control signals. A hardware design for the FCRP cache controller was presented in [16]. A software approach to hit/miss and replacement determination will now be explained.

When the CPU initiates a reference, the cache buffers the address in R . Assume for the moment that R_1 has the lowest starting address, that R_2 has the next lowest and so on. Using the fact that addresses in page frame i range from l_i to l_i+m-1 both hit/miss determination and replacement determination can be executed by the algorithm as shown in Fig. 8. Since R_1 does not necessarily have the lowest starting address, a pointer is needed to mark the tag register containing the lowest address. This works because of the circular nature of FCRP. Recall that a miss above frame n will cause frame 1 to be replaced; therefore, R_2 now has the lowest starting address. Fig. 8 will then start with R_2 and end with R_1 . If a miss occurs above the new frame 1, then frame 2 is replaced, and R_3 has the lowest starting address.

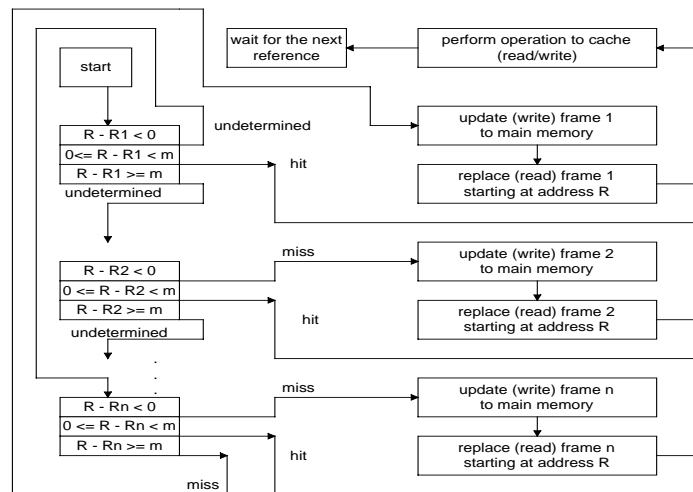


Fig. 8. Algorithm for hit/miss & replacement determination.

4. PERFORMANCE ANALYSIS

The objective of this performance analysis is to see how the cache size (nm) affects the miss ratio with regard to program behavior, program size, and memory configuration (fixed or variable). Clearly, the performance will depend on n , m , z , and P ; therefore, a formula for fixed page location and a formula for variable page location will be derived to show this. In the following subsections, anything that does not contribute to a hit or miss, e.g., data path or access times, will not be considered.

4.1 Markov Model

Because of the way cache systems are defined, a location referenced at some time t will be in cache before reference $t+1$ regardless of a hit or miss. Letting each page frame location equal a state, prior to reference $t+1$, the cache will be in the state that the reference at time t occupies. For instance, if reference t results in a hit to word location j of any page frame, then before reference $t+1$, the cache is in state j . Since there are m words per page frame, the cache can be in one of m states only.

Clearly, every reference results in a state transition. To calculate the state transition matrix for a fixed page location and variable page location configuration, some definitions will be given for convenience. Recall Fig. 3(b), the next reference distance model, where P equals the probability that the next reference will be one above the current, i.e., $\Pr(1)$. For $M = z$, all other distances have probability

$$\hat{P} = (1-P) / (z-1) = \Pr(d), \quad \text{for } d \neq 1.$$

At any time, there are nm locations in cache that could possibly be referenced and $z-nm$ locations outside of cache that also may be referenced; therefore, let the locations not in cache be denoted by

$$\hat{k} = z - nm.$$

For a fixed memory hierarchy, main memory is divided into equal pages. Since there are z locations of interest, z is divided by m to get the total number of pages in main memory. Each page of main memory has m locations, which correspond to the m states of a page frame; therefore, a miss to word j of any page will map only to word j (state j) of the replaced page frame. Let $P_{i,j}$ equal the probability of going from state i to state j ; i.e., if the cache is in state 1 at time t , then the probability that it will be in state 2 at time $t+1$ is $P_{1,2}$ [17]. Fig. 9 shows the m by m state transition matrix for a fixed location hierarchy.

$$\begin{aligned} P_{i,i+1} &= P + (z/m - 1) \hat{P}, \text{ for } i= 1 \text{ to } m-1 \\ P_{m,1} &= P + (z/m - 1) \hat{P} \\ P_{i,j} &= (z/m) \hat{P}, \text{ all other} \end{aligned}$$

Fig. 9. State transition matrix for fixed page locations.

The values shown in Fig. 9 were calculated using deductive reasoning and the defined variables. For instance, calculating a transition from state 1 to state 3 ($P_{1,3}$) entails adding up all the possibilities of getting to state 3 out of all the possible references. As stated earlier, in a fixed page configuration, word 3 of any page will map only to word buffer 3 (state 3) of any page frame; therefore, to get to state 3, a reference to word 3 of any of the z/m pages is needed. Since the cache is in state 1 at time t , none of the references are a distance 1 apart, therefore, each of the z/m references has probability \hat{P} so $P_{1,3} = (z/m) \hat{P}$.

Fig. 10 shows the state transition matrix for a variable page configuration.

$$\begin{aligned} P_{i,i+1} &= P + (n - 1) \hat{P}, \text{ for } i= 1 \text{ to } m-1 \\ P_{m,i} &= (n + \hat{k}) \hat{P}, \text{ for } i= 2 \text{ to } m-1 \\ P_{m,1} &= P + (n + \hat{k} - 1) \hat{P} \\ P_{i,j} &= n \hat{P}, \text{ all other} \end{aligned}$$

Fig. 10. State transition matrix for variable page locations.

4.2 Miss Ratio

Most reports in the literature give performance measures in terms of a limiting (average) fault rate (miss ratio) [9]. In this section the average fault rate denoted by P_{miss} , the probability of a miss for any reference, will be derived for fixed and variable page location configurations. The models for P_{miss} will be a function of n , m , z , and P and not any particular replacement policy. They will serve only to show the average behavior of each type of configuration. Since prior to any reference, the cache must be in one of the m states, it follows that $P_1 + P_2 + \dots + P_m = 1$. If the cache is in state 1 before any reference, then the probability of a miss is

$$P_{\hat{k}} = ((1-P)/(z-1)) (z-nm).$$

This result is arrived at because there are only z possible distances (jumps), $z - nm$ of which are outside of the cache. Recall that only a distance of 1 has a probability of P , and that the rest have a probability of \hat{P} ; therefore, state 1 (for $m > 1$) implies that a distance of 1 is in the cache. The same is true for the probability of a miss for states 2 through $m-1$, i.e., as long as there is a location in cache after the current state. The above probability is not necessarily true if the state is m and the next location is not in cache; i.e., the two adjacent page frames are not contiguous as shown in Fig. 11.

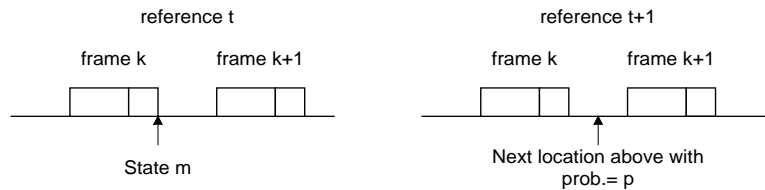


Fig. 11. Not contiguous page frames.

Let P_c equal the probability that there is a contiguous page frame and P_{nc} the probability that there is not. Clearly, $P_c = 1 - P_{nc}$. To find the probability of a miss if the state is m , we must consider both possibilities. If there is a contiguous page frame, the probability is as discussed above ($\hat{k} \hat{P}$). When there is no contiguous page frame, one location is removed from k , and P is added to get a miss probability of $P + \hat{P}(\hat{k} - 1)$.

All the possibilities are now covered; therefore, to find the probability of a miss, Bayes' theorem is used [18].

$$P_{miss} = \sum_{j=1}^m P(\text{miss}, j) = \sum_{j=1}^m P(\text{miss} | j) P_j.$$

Let $k = P_{\hat{k}}$ for convenience, then

$$P_{miss} = kP_1 + kP_2 + \dots + kP_{m-1} + P_m [P_{nc} (P + \hat{P}(\hat{k} - 1)) + P_c k].$$

Substitute $P_c = 1 - P_{nc}$; then,

$$P_{\text{miss}} = k \sum_{j=1}^m P_j + P_m P_{\text{nc}} (P - \hat{P}) = k + P_m P_{\text{nc}} ((Pz - 1) / (z - 1))$$

$$= ((1 - P) / (z - 1)) (z - nm) + P_m P_{\text{nc}} ((Pz - 1) / (z - 1)).$$

The above expression for P_{miss} is near completion. All that remains is to find suitable models for P_m and P_{nc} . This may seem like a difficult task, but they can be closely modeled using intuitive methods or, for P_m , using the Chapman-Kolmogorov equations and the state transition matrices shown in Figs. 9 and 10 [17].

A model for P_{nc} will now be developed by inspection. We will begin by looking at the case with only one page frame ($n = 1$). Clearly, with one page frame and $m < z$, there can not be any contiguous page frames; therefore, $P_{\text{nc}} = 1$ for $n = 1$ and $m < z$. If $m = z$, then the entire page frame is contiguous because main memory wraps around as shown in Fig. 3(a); therefore, $P_{\text{nc}} = 0$ for $n = 1$ and $m = z$. The one page frame situation is exactly modeled through the addition of two step functions, as shown in Fig. 12.

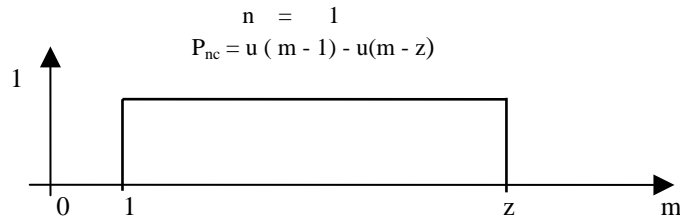


Fig. 12. P_{nc} for $n = 1$.

For the case where $n = z/m$, it should be obvious that all n page frames must be contiguous; therefore, $P_{\text{nc}} = 0$. From the above two extreme cases, i.e., $n = 1$ and $n = z/m$, it is seen that P_{nc} is also extreme, i.e., P_{nc} equals 1 or 0; therefore, to model the case in which $1 < n < (z/m)$, a linear approximation will be used as shown in Fig. 13.

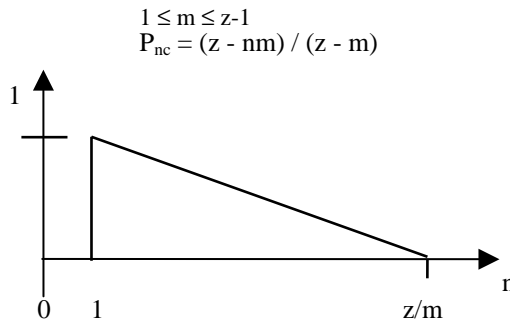


Fig. 13. P_{nc} for $z > nm$.

Combining both equations for P_{nc} produces the model

$$P_{\text{nc}} = ((z - nm) / (z - m)) [u(m-1) - u(m-z)].$$

Let us now look at P_m . As stated earlier, fixed and variable configurations must be

looked at individually. To distinguish between the two, P_m^f will be used for fixed P_m and, P_m^v for variable P_m . For a fixed configuration, it can be surmised that

$$P_m^f = 1/m$$

because of the direct mapping of word locations from page to page frame and the fact that no state is favored, i.e., a uniform distribution ($P_1^f = P_2^f = \dots = P_m^f = 1/m$).

For a variable configuration, P_m^v can be computed numerically by using Fig. 4.2 and the Chapman-Kolmogorov equations. The computation entails matrix multiplication until a steady state is reached, which can require a lot of computer time; therefore, a heuristic model will be used, i.e.,

$$P_m^v \approx 1/m (1 - ((m-1) f(n))/m^c), \text{ where } f(n) = (z - n)/(z - 1), \text{ and } c = \text{constant.}$$

A first look at P_m^v shows that it differs from P_m^f by the factor inside the brackets. Note that this factor has values between 0 and 1, and approaches 1 as n approach z . This is due to the fact that $P_{i,m} \propto n$. As m approaches z , P_m^v should approach P_m^f ; therefore, the constant c is used for convergence and is determined by the size of z . For instance, by letting the value in the brackets be equal to u for $m = z$, a value for c can be found by the formula

$$c = (\log_{10}(z-1) - \log_{10}(1-u)) / (\log_{10}(z)).$$

The value for u used in this paper is $u = 0.99$; therefore, $\log_{10}(1-u) = -2$. Values of u closer to 1 cause c to increase. The larger c is, the faster P_m^v converges to P_m^f . This fact may not be desired; therefore, it should be remembered.

4.3 Comparison

The basis for P_{miss} for each type of configuration has now been developed. As stated earlier, P_{miss} is considered stationary for all t ; therefore, P_{miss} is not indexed by t . In the following subsection, some numbers will be used to see how well a variable page location configuration does in comparison with a fixed page location.

4.3.1 Theoretical

To get a feeling for how the two different configurations compare, a practical example with different page sizes will be given. Suppose that a very large application, $z = 1M$, running continuously, e.g., on a LAN operating system, has a measured p value of 0.8. The goal is to design a cache system with an overall 20% miss ratio ($P_{\text{miss}} = 0.2$). Based on using the P_{miss} formulas derived above for both a fixed and variable page location configurations, the values listed in Table 3 was calculated so as satisfy to meet the criteria.

As seen in Table 3, there are substantial savings in buffer size. This is especially true for practical page sizes for microcomputers ($m=4$ and $m=16$). P_{miss} is an expected value; therefore, a cache system implemented with a structured replacement policy, e.g., FCRP or LRU, may have better miss ratios. Specifically, fixed page location cache systems have a theoretical limit exhibited by the OPT (optimal) replacement policy [7].

Table 3. Theoretical cache design: $z=1M$, $P=0.8$, $P_{miss}=0.2$.

m	Fixed		Variable		Difference	
	N	words	n	words	%less	words
4	131,072	524,288	95,144	380,576	27.4	143,712
16	13,107	209,712	8,917	142,672	32.0	67,040
64	964	61,696	736	47,104	23.7	14,592
256	63	16,128	53	13,568	15.9	2,560

5. SIMULATION

Because of the time constraints, the parameters used for simulation were not necessarily the best ones for showing the merits of FCRP over conventional systems. Instead, the parameters were chosen solely to show how well the theoretical models described in section 4 predict the simulated results of fixed and variable page location cache systems.

5.1 Simulation Program (SIMCACHE)

The C program written for simulation, called SIMCACHE, has two main functions. The first function is to simulate memory traces that are used to drive the second function, which is to calculate the miss ratio of a cache system. The program is menu driven, and it is DOS based.

5.1.1 Trace simulation

A program trace is a string of numbers generated by the CPU that is driven by a program. The numbers in the string represent the sequence of memory addresses of opcodes and operands in main memory used by the CPU. SIMCACHE simulates three different traces. The first trace is a simple loop, the second is a semi-random trace, and the third is a fragmented trace. All three traces have the same storage format; i.e., they are ASCII files. The traces also share five common variables that are selected in a menu. A trace is given a DOS filename, total references, beginning address, smallest address, and largest address. The latter four variables are stored in the trace file in the order mentioned. Each variable is separated by a space; therefore, it is easy to examine a trace file with an ASCII editor to see what the variables are.

5.1.2 Miss ratio determination

Using a trace file with the format explained in the last section, the performance of four different cache systems could be simulated. They were FCRP, OPT, LRU, and FIFO. The theoretical values for both fixed and variable page location systems were also computed using the acronyms TheoF and TheoV, respectively.

5.2 Results

As stated earlier, simulating a cache system requires careful planning. Time is a big factor when simulating the OPT algorithm because every time a miss occurs, the dis-

tances for each page frame have to be calculated. This translates into many disk reads. Space is another factor since trace files with many references translate into large disk files.

5.2.1 Theoretical vs. practical

To see how well the theoretical models predict performance, one semi-random trace were used. The trace file is SEMI4K.6 the memory variables, i.e., references=81920, beginning=0, smallest=0, and largest=4095. From the given parameters $z=4096$, therefore, each location should be referenced an average of 20 times. Table 4 shows how close the predicted miss ratios match the simulated miss ratios for both fixed and variable systems.

Table 4. Miss ratios for trace file SEMI4K.6, $P=0.605037$.

n	m	FIXED			VARIABLE	
		TheoF	LRU	FIFO	TheoV	FCRP
256	4	0.410	0.408	0.409	0.373	0.366
512	4	0.273	0.273	0.271	0.250	0.251
768	4	0.137	0.136	0.135	0.126	0.130
1024	4	0.000	0.000	0.000	0.000	0.000

As the above table shows, the predicted values are exceptionally good for each type of configuration. Of special interest here is the slope of FCRP. It should be noted that FCRP has a flatter slope than TheoV predicts.

As stated earlier, the above examples were used primarily to show how well the models work. Although the examples use small numbers, there still are significant savings in computer time. For instance, using the values listed in Table 4 and comparing $\overline{FCRP} = 0.187$ (average FCRP) with $\overline{LRU} = 0.204$ gives an average of 1433 less misses for FCRP over 80K references. Obviously, the difference in misses becomes more significant as the number of references gets larger.

5.2.2 Fragmented traces

To study the performance of cache systems under non-ideal program behavior, one fragmented trace, FRAG8K.6#3, were used. The fragmented trace consisted of references=81920, beginning=0, smallest=0, largest=8191, fragments=3, and fraction=0.5. In this case, $z=8K$, but only 4k of that space was used. The FRAG8K.6#3 trace attempted to simulate non-ideal program behavior of the SEMI4K.6 file if it was divided into 3 fragments and spread out over an 8K space.

Table 5, shows the result of cache performance for FCRP and LRU using the same page sizes used for the semi-random traces.

Table 5. Miss ratios for trace file FRAG8K.6#3, $P=0.600911$.

N	m	FCRP	LRU
256	4	0.023	0.533
512	4	0.020	0.530
768	4	0.018	0.527
1024	4	0.013	0.523

As seen from the above tables, there are substantial differences in performance among all four cases. For instance, using Table 5, $\overline{FCRP} = 0.0185$ and $\overline{LRU} = 0.528$, which gives FCRP 41758 less misses over 80K of references.

6. CONCLUSIONS

In this paper, four new ideas have been introduced. The first idea, variable page locations, defines a new and more flexible way of mapping pages to page frames. The second idea, a horizontal representation of a memory hierarchy, represents a new way of viewing a cache/main memory hierarchy, by giving a sense of distance between page frames and a visual sense of cache performance. The third idea, the next reference distances model, explains program behavior in terms of space instead of time. The fourth idea, Location Policies, is used to show how a variable page location configuration can be practically implemented while increasing performance.

REFERENCES

1. O. I. Aven, L. B. Boguslavsky, and Y. A. Kogan, "Some results on distribution-free analysis of paging algorithms," *IEEE Transactions on Computers*, Vol. c-25, No. 7, pp. 737-745.
2. P. J. Denning, "The working set model for program behavior," *Communication of ACM*, Vol. 11, No. 5, 1968, pp. 323-333.
3. P. A. Franaszek and T. J. Wagner, "Some distribution-free aspects of paging algorithm performance," *Journal of ACM*, Vol. 21, No. 1, 1974, pp. 31-39.
4. J. E. Shemer and G. A. Shippey, "Statistical analysis of paged and segmented computer systems," *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 6, 1966, pp. 855-863.
5. E. G. Coffman Jr. and T. A. Ryan Jr., "A study of storage partitioning using a mathematical model of locality," *Communications of ACM*, Vol. 15, No. 3, 1972, pp. 185-190.
6. P. J. Denning and S. C. Schwartz, "Properties of the working-set model," *Communications of ACM*, Vol. 15, No. 3, 1972, pp. 191-198.
7. J. P. Hayes, *Computer Architecture and Organization*, 3rd ed., McGraw-Hill Book Company, 1998
8. W. F. King, "Analysis of paging algorithms," in *Proceedings of IFIP Cong.*, 1971, pp. 485-490.
9. G. S. Rao, "Performance analysis of cache memories," *Journal of ACM*, Vol. 25, No.

- 3, 1978, pp. 378-395.
10. A. J. Smith, "Cache memories," *ACM Computing Surveys*, Vol. 14, No. 3, 1982, pp. 473-530.
 11. C. K. Chow, "Determination of cache's capacity and its matching storage hierarchy," *IEEE Transactions on Computers*, Vol. c-25, No. 2, 1976, pp. 157-164.
 12. J. E. Smith and J. R. Goodman, "Instruction cache replacement policies and organizations," *IEEE Transactions on Computers*, Vol. C-34, No. 3, 1985, pp. 234-241.
 13. L. Colagiovanni and A. Shaout, "Cache memory replacement policy for a uniprocessor system," *IEE Electronic Letters*, 1990, Vol. 26, No. 8, pp. 509-510.
 14. A. Shaout and L. Colagiovanni, "A location policy cache memory control design," in *Proceedings of the ISMM International Conference on Parallel and Distributed Computing and Systems*, 1991, pp. 102-106.
 15. D. Ferrari, "Improving locality by critical working sets," *Communications of ACM*, Vol. 17, No. 11, 1974, pp. 614-620.
 16. A. Shaout and J. Cancchio, "Hardware design for forward circular memory replacement policy," Technical Report #UMD994, ECE Dept., the University of Michigan-Dearborn, September/94.
 17. K. S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, Inc., 1982.
 18. A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 2nd ed., McGraw-Hill Book Company, 1984.

Adnan K. Shaout is a Professor in the Electrical and Computer Engineering Department at The University of Michigan-Dearborn. His current research areas include Applications of Fuzzy Set Theory, Computer Arithmetic, Parallel Processing and Intelligence Systems. Dr. Shaout has about 17 years experience in teaching and conducting research in the electrical and computer engineering fields at Syracuse University and The University of Michigan-Dearborn. Dr. Shaout has published over 80 papers on topics related to electrical and computer engineering. Dr. Shaout obtained his B.Sc., M.Sc, and Ph.D. degrees in Computer Engineering from Syracuse University, Syracuse, NY, in 1982, 1983, and 1987, respectively. Dr. Shaout is a member of ASME and IEEE.

Larry Colagiovanni is a graduate student at the University of Michigan-Dearborn in the Electrical and Computer Engineering Department.