# On memory contention problems in vector multiprocessors

Christine Fricker

# On memory contention problems in vector multiprocessors

C. Fricker

INRIA, Domaine de Voluceau

B.P.105, 78153 Le Chesnay Cedex, France

(1) 39 63 55 27

fricker@schuss.inria.fr.

**Abstract**

Memory interleaving considerably increases memory bandwidth in vector processor systems. The concurrent operation of the processors can produce memory bank conflicts and hence alter the memory bandwidth. Total or steady state performance for vector operations in a memory system is studied. Many methods of resolving memory bank conflicts are proposed and compared. Analytical results on the resulting effective bandwidth are presented for one of them and the others are described by exhaustive simulations. Some nonintuitive results are obtained on how conflicts depend on the size of the architecture, the number, the stride and the length of the vectors, the register length assigned by each processor to vector components.

**Index terms**

Interleaved memory, memory access in vector mode, memory bandwidth, modeling of access streams, multiprocessors, performance evaluation, registers.

## 1 Introduction

As the speed of CPUs increases, memory latency becomes the significant bottleneck of computer performance. One class of multiprocessors, the tightly-

coupled multiprocessors, are designed with a global shared memory in order to solve data transfers problems. Designers must find for them the right relationship between the memory bandwidth (number of memory accesses per cycle) and the processor bandwidth (number of requests per cycle) in order to optimize the use of memory cycles, so that a maximal memory bandwidth is balanced with a minimal latency for requests. The main factor which decreases the memory bandwidth is the memory bank contention. Thus, when designing tightly coupled MIMD computers, efficient hardware management of memory bank contention is a crucial issue. However, the problem is not easy because the memory contention depends on the rules of arbitration of conflicts and on a large number of architectural parameters: number of processors, number of banks, register length, latency between the loading of the registers, etc. The influence of these parameters is not well understood and nonintuitive. For example, it is commonly thought that increasing the size of the architecture (the number of processors and memory banks) increases the memory contention, but the specific influence of the number of processors and the number of banks is not clear.

One difficulty encountered in the analysis of the memory contention is how to choose a representation of the sequences of addresses generated by the programs. Most of the time, the accesses to the memory banks are assumed to be random, mainly because of the mathematical tractability of this model. In this case, exact or approximated models have been analyzed in [2], [1] (see also [3], [6], [9], [10], [11]). However, in the context of scientific computation, the hypothesis of independence of successive addresses is very unlikely (except perhaps in sparse computation). In this case, the programs are frequently based on the execution of loops, generating accesses to arrays referenced by a linear index . These loops generate a sequence of references to the successive components of vectors stored in contiguous memory banks (vectors of stride one) or in equally spaced memory banks (vectors of stride greater than one). Although this is a restricted model, these regular accesses are basic and represent a wide class of loops in programs. However, there has been little research on these regular memory access patterns. The rare studies on such vector operations have been done for the CRAY X-MP memory system by Cheung and Smith ([5]) and Oed and Lange ([7], [8]), but only in the case where the vector lengths are less than the register length. They show that the contention depends strongly on the vector starting addresses. Hence they consider exhaustively all the possible vector patterns. The complexity

of this combinatorial problem explodes with the number of the vectors so that their study is limited to two vectors for the analytical results and three vectors for the simulations results.

In this work we analyze regular references streams generated by operations on vectors (of the same length with different starting addresses) where the vector lengths can be greater than the register length. The target architectures are MIMD tightly coupled processors with one port per processor. The size of the architecture in this study is sufficiently small (less than 16 processors and 16 memory banks), so that a crossbar network can be used as the interconnection network, resulting in no conflicts except the memory bank conflicts. An example of such an architecture is the ALLIANT FX/8, developed by the University of Illinois (Urbana-Champaign) with eight processors and a shared cache divided into four banks. With the reference streams described above, different types of conflict management are considered. The *static priority* protocol, for which processor $i$ has priority over processor $i'$ if $i < i'$ in the case of a conflict between $i$ and $i'$, was implemented in a version of the ALLIANT FX/8 to dynamically resolve conflicts. This conflict resolution scheme always penalizes the same processor. In this paper, we propose other schemes to reduce this problem and hence to improve memory performance. These include priority policies and a queueing policy. The priority policies are variants of the static priority, with the priority depending on time and even on the memory address of the request: *cyclic priority* , as between the two processors of the CRAY X-MP, *rotation* and *conflict priorities*.

The purpose of this paper is to evaluate and to compare the performance of the different conflict resolution schemes and to study its evolution when architectural parameters are modified: the size of the architecture and the length of the registers assigned by each processor to components of each vector. The conflict rate is the proportion of the total execution time wasted in latency. This is the performance measure describing the impact of the conflicts.

The first way to have an idea of the performance is to perform simulations: we use an exhaustive technique, where all the vector patterns are simulated, as in [5]. The simulation results, obtained by a simulator called MEVAMP, are presented in graphical form. They cover the cases of practical interest and allow us to compare the different policies and to study the influence of the parameters of the architecture.

One of the policies, the *rotation priority*, is analyzed analytically. It

yields results for a larger range of parameters. Moreover, according to the simulation results mentioned above, this policy is quite efficient compared to the other policies. Its analytical study is therefore of special interest. We obtain the analytical expressions for the delay due to conflicts and the mean delay when the starting addresses and possible durations to load registers that create conflicts are randomly chosen. Upper and lower bounds are derived for the mean conflict rate, which give a good idea of the influence of the size of the architecture . This gives us some limits on the scalability of the architecture. For example, we prove that, under a certain condition on the bandwidths, it is better to have 2n processors and n banks than n processors and 2n banks.

In Section 2, the model of the architecture is described. Sections 3 and 4 deal with the analytical results. In Section 5, we discuss how the simulations were performed and present the results. Section 6 contains some extensions of the simulation results, including non unit strides. The technical proofs are given in the appendix.

# 2 Description of the model

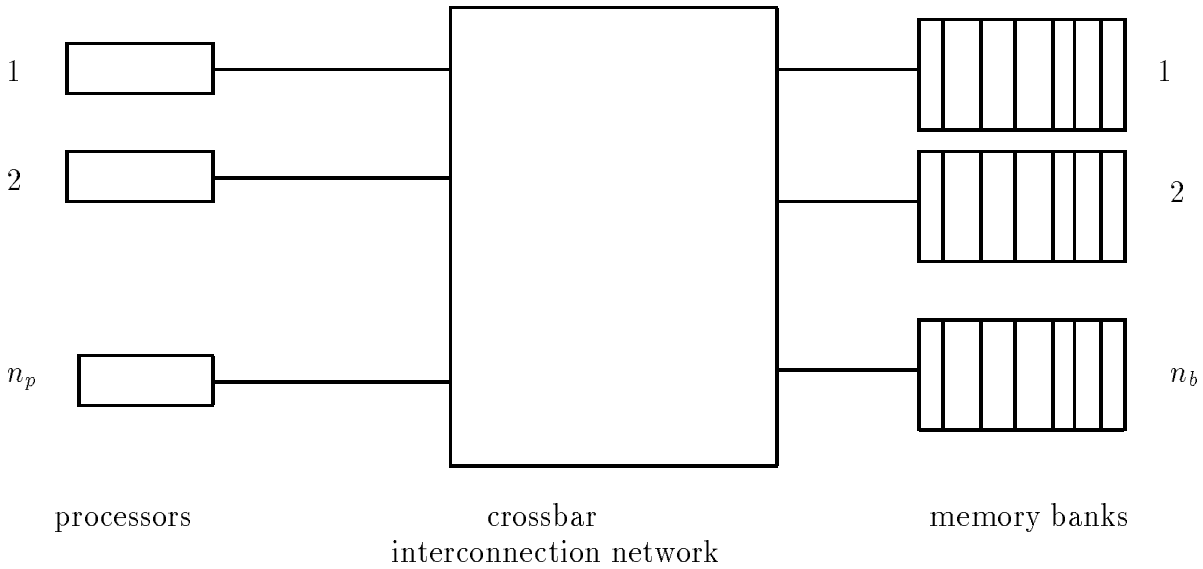


Fig. 1. The architecture.

The model analyzed in this paper is an architecture of $n_p$ processors indexed by $i\,(0 \leq i \leq n_p - 1)$ which share a memory divided into $n_b$ directly accessible banks indexed by $j\,(0 \leq j \leq n_b - 1)$ (see Fig. 1). The interconnection network is a crossbar network, so no conflicts between requests are introduced by the network. Thus all the conflicts are bank conflicts.

## 2.1 The memory system.

The memory banks are directly accessible by the processors. A successful request to a bank occupies that bank during a fixed number of cycles, denoted $t_b$, called the memory occupation time. Two or more processors may request the same bank simultaneously, resulting conflicts on the bank. When such a conflict occurs, the time required to receive the requested data will be the associated delay, denoted $t_w$, plus the memory occupation time. The delay $t_w$ depends on the conflicts with other possible requests for data stored in the same bank. A processor has the capability of delaying a request if it can not access bank because of some conflict. Conflicts are resolved dynamically, i.e. all the requests that cannot be satisfied will be delayed for one cycle by their processors, and also all the subsequent requests of that processor.

There are two types of conflicts:

- a *bank busy conflict* occurs when a bank is still busy when a processor requests it, and causing the request to fail.
- a *simultaneous bank conflict* occurs when two or more processors request the same idle bank at the same time. Arbitration is needed in this case.

Except for the *queue discipline* described later, the principle of arbitration of simultaneous bank conflicts is to assign a number, called the priority number and denoted by $P(i,t)$, between 0 and $n_p - 1$ to each processor $i$ by a one-to-one mapping $P$, possibly depending on time $t$. When a simultaneous bank conflict occurs, the processor with the lowest number has access to the bank while the other processors resubmit their requests at the next cycle. At time 0, $P$ is arbitrarily defined by $P(i,0) = i\ (0 \leq i \leq n_p - 1)$ . A variety of different priority assignments are proposed below.

1) **The static priority**

The first one, the *static priority*, does not depend on time. Processor $i$ has priority over processor $i'$ if $i < i'$. Note that $P$ is defined by

$$P(i,t) = i, \;\; 0 \leq i \leq n_p - 1, \; t \geq 0.$$

This implies that the priority numbers of all the other processors can be defined from the priority number of a single one by the relation

$$P(i,t) = (P(i-1,t)+1) \; mod \, n_p \quad (t \geq 0, \; 0 \leq i \leq n_p - 1). \tag{1}$$

This relation will be assumed to be true for the *cyclic, rotation, conflict priorities* defined later. It is sufficient to know the processor having the highest priority at any given time, in order to know the priority of all the processors.

Priorities of this type were used in an earlier version of the ALLIANT FX/8 and manage memory contention between the ports of a CPU in the CRAY X-MP.

Note that with the static priority assignment, processor 0 always ends first.

2) **The cyclic priority**

To promote fairness between processors, another policy is to change the priority numbers cyclically. Every c cycles, if processor $i$ had the highest priority, processor $i+1$ takes it. For the *cyclic priority of period c*, $P$ is defined by

$$P(i,t) = i - [t/c] \quad (t \geq 0, \; 0 \leq i \leq n_p - 1)$$

which, as before, implies the relation ( 1).

Alternating priority of this type is used in the CRAY X-MP ([5]) between the two CPU's. With this priority assignment, $P(i,t)$ does not depend on conflicts before $t$.

3) **The rotation priority**

In order to avoid giving priority to a processor which has just completed an access, another type of arbitration is interesting, in which the map $P$ depends on the banks. This is also a priority assignment respecting the relation ( 1). Moreover, at a given bank, after an access corresponding to a request of processor $i$ has completed, the priority moves and processor $i+1$ becomes the processor of the highest priority. Hence, for this scheme, called

the *rotation priority*, the priority number of processor $i$ at bank $j$ at time $t$ is given by

$$P(i,j,t) = P(i,j,T_{n_j}^-) - 1 \quad (T_{n_j} \leq t < T_{n_j+1}, 0 \leq i \leq n_p - 1, 0 \leq j \leq n_b - 1)$$

where $T_{n_j}$ is the end of the $n_j$-th access at bank $j$ and $P(i,j,T_{n_j}^-)$ denote the value $P(i,j,t)$ just before $T_{n_j}$.

4) **The conflict priority**

The *conflict priority* is similar to the *rotation priority* except that the priority changes at the end of an access only if there was a simultaneous bank conflict for the access ($t_b$ cycles earlier). Thus, $P$ is defined by

$$P(i,j,t) = P(i,j,T_{n_j}^{'-}) - 1 \quad (T_{n_j}^{'} \leq t < T_{n_j+1}^{'}, 0 \leq i \leq n_p - 1, 0 \leq j \leq n_b - 1)$$

where $T_{n_j}^{'}$ is the end of the $n_j$-th access at bank $j$ after a simultaneous conflict and $P(i,j,T_{n_j}^{'-})$ denote the value $P(i,j,t)$ just before $T_{n_j}^{'}$.

5) **The queue discipline**

The last policy we consider, called *queue discipline*, is quite different in the sense that the priority of the request depends on its request time. Each bank has a queue and if a request is not accepted at once, the processor does not delay it until the next cycle but the request waits in the queue of the bank. The queueing discipline at each queue is assumed to be first-in-first-out, but requests which arrive simultaneously to the same queue are served according to the static priority defined above.

## 2.2 The processors

Each processor sends its requests to the bank at the following rate: when a request is about to be served, the processor which sent it waits $t_p$ cycles before sending the next request. This delay is called the inter-request time. The time required for requests to cross the network is included in $t_p$. Let the processor bandwidth be the number of requests sent by the multiprocessor per cycle; if there is no conflict, it is $n_p/t_p$. Let the memory bandwidth be the number of memory accesses per cycle; if there are no conflicts, it is $n_b/t_b$. We assume here the following condition on the bandwidths

$$n_p/t_p = n_b/t_b. \tag{2}$$

It means that the bandwidths are perfectly balanced: Without conflicts, the requests for the memory do not wait and the rate of occupation of the memory is one.

Each processor has vector registers of finite length $r$ to store the vector elements during an operation.

## 2.3 The load

We will consider a vector multiprocessor executing a vector operation. The load is shared between the processors. It can be assumed without loss of generality that the vector length $L$ (the number of the elements in the vector) is a multiple of $n_p r$. If this were not the case, the operations on the remaining elements would be executed at the end with an execution time small enough compared to the total execution time to be negligible. Each vector argument of the operation being performed is divided into $n_p$ subvectors having the same number of consecutive elements. The $i$-th processor executes the operation on the $i$-th subvector of each of the arguments. The vector elements are loaded from the memory into the vector registers of the processor. The register length $r$ is limited so that the access to a subvector is made by slices, also called blocks, of $r$ consecutive elements. The $i$-th processor requests the $n$-th slice of its subvector of each of the arguments before moving on to request the $(n+1)$-th slices. For example, suppose an operation executed by a 8-processor computer involving two vectors $V$ and $V'$ of length 96 distributed as 8 subvectors of length 12. The register length is assumed to be 4. The $i$-th subvectors $V^{(i)}$ and $V'^{(i)}$ are divided into 3 slices of 4 consecutive vector elements $S_1^{(i)}, S_2^{(i)}, S_3^{(i)}$ (respectively $S_1'^{(i)}, S_2'^{(i)}, S_3'^{(i)}$). The sequence of blocks of the $i$-th processor is: $S_1^{(i)}, S_1'^{(i)}, S_2^{(i)}, S_2'^{(i)}, S_3^{(i)}, S_3'^{(i)}$.

Unless stated otherwise, the stride of the vectors is assumed to be one.

Between the requests of two blocks, the processor waits a fixed number of cycles, called the inter-block time. It will be greater than the inter-request time $t_p$, more exactly a multiple of $t_p$, denoted $\delta t_p$ where $\delta$ is an integer.

The processors are assumed to be synchronized at the beginning of the execution.

# 3 General analytical results

The following results are independent of the chosen policy. They explain the choice of the range of two random parameters: the vector patterns and the inter-block time. Before them, let us introduce the main time metrics which will be used later.

The multiprocessor system executes vector operations as described in Section 2. Let $T_0$ be the total execution time for a vector operation without conflicts. This is the minimum value that the total execution time can attain. Given $T_0$, the total execution time $T$ is characterized by the total delay $R$ which must be added to $T_0$ to obtain $T$.

**Definition** Let the conflict rate, denoted $t_c$, be

$$t_c = \frac{R}{T}$$

where $T$ denotes the total execution time, $R = T - T_0$ the total delay due to conflicts, $T_0$ the total execution time without conflict.

It is easy to obtain an analytical expression for $T_0$.

**Proposition 1** *The total execution time without conflict can be expressed as*

$$T_0 = (r - 1 + \delta)t_p v\frac{L}{n_p r} + t_b - \delta t_p, \tag{3}$$

*where $v$ is the number of vectors of length $L$ , $r$ is the register length, $\delta t_p$ is the inter-block time, $n_p$ is the number of processors, $t_p$ is the inter-request time and $t_b$ is the memory occupation time.*

**Proof**

To execute the vector operation on $v$ vectors of length $L$, each processor has to request $v\frac{L}{n_p}$ vector components, i.e. to perform $v\frac{L}{n_p r}$ vector register stores. The time required for one such store, without conflict, is $(\delta + r - 1)t_p$, except for the first one which is $t_b + (r-1)t_p$. This gives the result.

In the analysis, the time evolution of the system can be described by the following time process: $(W(t) = (W_i(t), 0 \leq i \leq n_p - 1), t \in IN)$ where $W_i(t) = (w_j, 0 \leq j \leq n_b - 1)$ where $w_j$ is the non negative memory occupation time which remains at time $t$ for processor $i$ at bank $j$. This is called the

residual occupation time process. The process $W(t)$ is periodic from a certain instant to the completion of the first processor because $(W(t), t \in I\!N)$ is a Markov chain on a finite set with deterministic transitions. Thus, the evolution of the execution is divided into three distinct phases: a startup phase, a periodic phase called stationary and a completion phase. In this context, a desirable solution would be a conflict resolution scheme with a small startup phase and a stationary phase without conflicts. We will prove in Section 4 that this is not the case using the rotation priority, and counter examples can easily be found using the static priority for some possible starting addresses and inter-block times.

**The starting memory addresses.**

However, for a special vector pattern, the total delay is minimum for all conflict resolution schemes.

**Remark 1** *When the first component of each vector is stored in the same bank, for all conflict resolution schemes, for any register length which is a multiple of $n_b$, the total delay is*

$$R = (n_p - 1)t_b$$

*independent of inter-block time. Note that $(n_p - 1)t_b$ is the time necessary for processor $n_p - 1$ to have access to the first bank because all the processors request the same bank at time 0, due to the fact that $n_p r$ divides $L$ and $n_b$ divides $r$.*

**Proof.**

The key fact is that, if the vectors have the same starting addresses, for example 0, each processor has the same sequence of addresses to access:

$$0, 1, \ldots, (r-1) \ mod \ n_b, \ldots, 0, 1, \ldots, (r-1) \ mod \ n_b, r \ mod \ n_b, \ldots$$

If $n_b$ divides $r$ then $(r-1) mod \ n_b$ is $n_b - 1$ and each processor has access to the banks cyclically. After an initial delay of $it_b$ for processor $i$ $(0 \leq i \leq n_p)$ to access bank 0, the processors request data in each bank cyclically without any conflict. It is true, for any value of the inter-block time and for any conflict resolution scheme.

In the previous case of vector pattern, the stationary conflict rate is zero. Let us present cases where the conflict rate is high even for relatively small

architectures. We take this example because it is commonly thought that the degradation due to conflicts increases with the size of the architecture. We consider a 4-processor, 4-bank computer using two different conflict resolution schemes. For a register length of 4 and an inter-block time of $2t_p$, the conflict rate using the *static priority* can reach 33% for some vector starting addresses (see Figure 2 ). The *rotation priority* is also bad for small values of register lengths and inter-block times: When the register length is 4 and the inter-block time is $t_p$, the conflict rate reaches 33% for some vector starting addresses (see Figure 2). It shows that the performance of the memory is highly dependent on the starting memory addresses of the vectors and it is necessary to consider exhaustively all the vector patterns in a study of the conflicts.

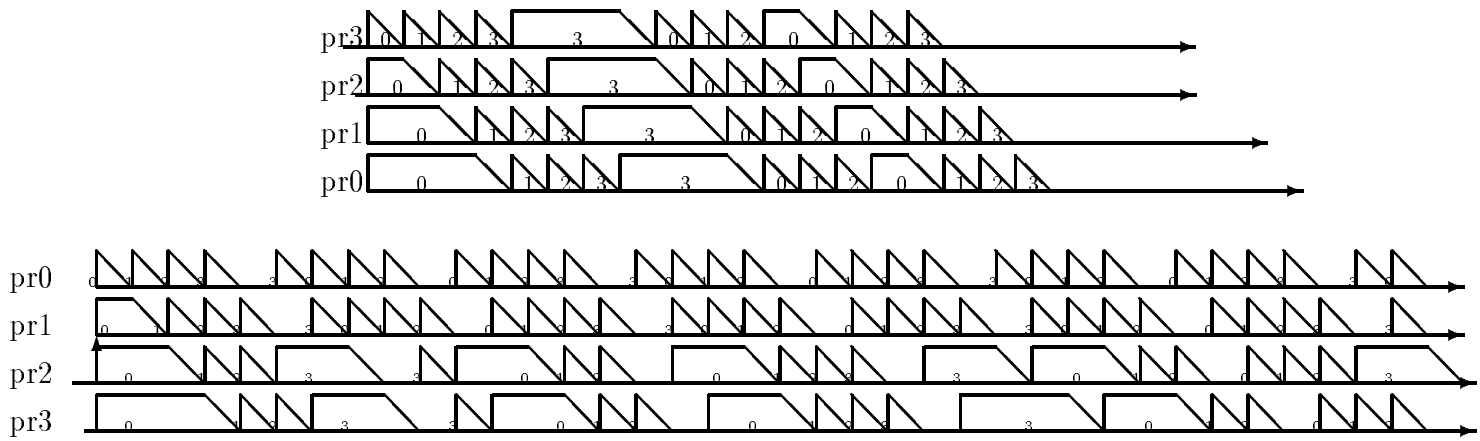


**Fig. 2. State process $W(t)$ for a 4-processor, 4-bank computer**

**. register length: 4,**

**. inter-block time: a) $2t_p$, b) $t_p$**

**. starting memory addresses: 0, 3**

**using a) static, b) rotation priorities.**

**The inter-block time.**

The latency between the load of two blocks is random. The influence of its value on conflicts must be taken into account. First, remark that

**Remark 2** *For all conflict resolution schemes, if $\delta \geq n_b$, for every $r$ which is a multiple of $n_b$ and all possible vector starting addresses, the total delay is*

$$R = (n_p - 1)t_b.$$

**Proof.**

As $L$ is evenly divisible by $n_p r$ and $r$ is evenly divisible by $n_b$, each processor requests the same bank at time 0. As mentioned above in remark 1, for every policy, the delay experienced by processor $n_p - 1$ when it first accesses this bank is $(n_p - 1)t_b$. However, the processors experience no delays for the remaining addresses of the first block. By the time that processor 0 is ready to make its first request in the second block, all other processors will have finished their first block of requests. To see this, remember that processor 0 makes its first request in the second block at time $(r-1)t_p + \delta t_p$, whereas all processors finish their first block requests at time $n_p t_b + (r - 1)t_p$. Since $\delta \geq n_b$ and $n_b t_p = n_p t_b$, $\delta t_p \geq n_p t_b$. Hence, at the time of the first request of the second block, no other processors are occupying the memory banks, so no waiting occurs. Thereafter, processors access the banks successively.

Remark 2 proves that a solution to avoid conflicts would be to force the inter-block time to be greater than $n_b t_p$. However, this inter-block time could be larger than the delay due to conflict, and the solution would not be attractive. We will see in Section 4 that, for the rotation priority, the mean delay due to conflicts, taking an inter-block time value at random between $t_p$ and $(n_b - 1)t_p$, is, for large vector lengths, $vst_p(n_b + 1)/6$ while the delay due to an inter-block time $n_b t_p$ is $vst_p n_b$, which is near six times larger. For this reason, we will study the memory performance when the inter-block time value is between $t_p$ and $(n_b - 1)t_p$, which are all the values creating conflicts.

# 4 Analysis of the rotation priority

An analysis of the *rotation priority* is presented in this section. The aim is to calculate the total delay due to conflicts as a function of the characteristics

of both the architecture and the reference streams and to use it to study the influence of the numerous parameters.

It is easy to see (cf. the appendix) that the startup phase has $(n_p - 1)t_b$ cycles, the stationary phase has at least $s - 1 = L/n_p r - 1$ full periods of $v(r - 1 + \delta)t_p + R_{stat}$ cycles, where $R_{stat}$ is the delay due to conflicts for a period of the periodic phase. Note that a period is the time necessary for each processor to store all the vector registers once. The completion phase has $(n_p - 1)t_b$ cycles. The total delay due to conflicts $R(\delta, p)$ is then given as a sum in the following proposition. The first term is the delay during the startup phase, the following terms are the delays during a single period of the stationary phase, the last term is the delay during the completion phase.

**Proposition 2** *Let us denote with $V_1, \ldots, V_v$, $v$ vectors with starting addresses $a_1, \ldots, a_v$. Without loss of generality, $a_1$ is assumed to be 0. Let $p_i = a_{i+1} - a_i$ denote the difference of the banks where the starting addresses of $V_{i+1}$ and $V_i$ $(1 \leq i \leq v - 1)$ are stored. We call $p_i$ the relative starting bank of vector $i$. For an architecture with $n_p$ processors of register length $r$ (a multiple of $n_b$), $n_b$ banks, the total delay $R$ is a function of $\delta$ and the $(v - 1)$-tuple of the relative starting banks $p = (p_i, 1 \leq i \leq v - 1)$, and is given by*

$$
\begin{aligned}
R\ (\delta, p) &= (n_p - 1)t_b \\
&+ (s - 1)t_p[\sum_{i=1}^{v-1}((p_i - r)\ mod\ n_b - \delta + 1)^+ + ((-\sum_{i=1}^{v-1} p_i - r)\ mod\ n_b - \delta + 1)^+] \\
&+ t_p \sum_{i=1}^{v-1}((p_i - r)\ mod\ n_b - \delta + 1)^+, \qquad (4)
\end{aligned}
$$

*where $max(a, 0)$ is denoted by $(a)^+$ and $s = \frac{L}{n_p r}$ is the number of slices per vector.*

The proof is given in the Appendix. In fact, we have given this delay as a function of the two random parameters: the banks of the first component of the vectors and the inter-block time. On one hand, the addresses of the first component of the vectors satisfy very simple assumptions: they are independent and uniformly distributed across the banks. On the other hand, the latency $\delta t_p$ between the loading of the registers is not a constant in the system. In order to study conflicts, we assume it is distributed randomly

over all possible values which cause conflicts. Remark 2 in Section 3 proved that conflicts occur only when $\delta$ is between 1 and $n_b - 1$.

The aim of the two following results is to give statistics on the total delay: average, best and worst cases.

**Corollary 3** *If $\delta \geq n_b$, then*

$$R(\delta, n) = (n_p - 1)t_b.$$

*Moreover if $\delta \leq n_b$, the mean $ER(\delta)$ of $R(\delta, n)$, when the starting addresses are independent and uniformly distributed on the banks, is*

$$ER(\delta) = (n_p - 1)t_b + (vs - 1)t_p(n_b - \delta)(n_b - \delta + 1)/2n_b.$$

*The mean ER of $ER(\delta)$, if $\delta$ is uniformly distributed on the set of values $\{1, \ldots, n_b - 1\}$ corresponding to the cases of conflicts, is given by*

$$ER = (n_p - 1)t_b + (vs - 1)t_p(n_b + 1)/6. \tag{5}$$

**Proof.**

The first assertion, which is contained in remark 2, follows immediately from Proposition 2.

As $a_0, a_1, \ldots, a_v$ are independent and uniformly distributed random variables on the set of the banks, it is easy to see that the relative addresses $p_1, \ldots, p_{v-1}$ are uniformly distributed on the set $\{0, \ldots, n_b - 1\}$. Finally, $(p_i - r)\ mod\ n_b$ and $(-\sum_{i=1}^{v-1} p_i - r)\ mod\ n_b$ are uniformly distributed on the set $\{0, \ldots, n_b - 1\}$. Their common mean $\frac{n_b - 1}{2}$ is used to calculate $ER(\delta)$. Moreover, if $\delta$ is uniformly distributed on the set $\{1, \ldots, n_b - 1\}$, then

$$E(\delta) = n_b/2$$

and

$$E(\delta^2) = n_b(2n_b - 1)/6.$$

This gives the result.

The following proposition gives the relative vector starting banks, defined in Proposition 2, corresponding to the minimum and the maximum of the total delay.

**Proposition 4** *Under the assumptions of the previous proposition, if $n_b$ divides $r$, the $(v-1)$-tuple of the relative starting banks giving the minimum (respectively the maximum) value of $R(\delta,p)$ is $(0,\ldots,0)$ (respectively $(n_b-1,\ldots,n_b-1)$) and*

$$\begin{aligned}\min_{p\in\{0,\ldots,n_b-1\}^{v-1}} R(\delta,p) &= R(\delta,0,\ldots,0)\\ &= (n_p-1)t_b,\end{aligned}$$

$$\begin{aligned}\max_{p\in\{0,\ldots,n_b-1\}^{v-1}} R(\delta,p) &= R(\delta,n_b-1,\ldots,n_b-1)\\ &= (n_p-1)t_b+s(v-1)t_p(n_b-\delta)^+\\ &\quad +(s-1)t_p((v-1)\ mod\ n_b-(\delta-1))^+.\end{aligned}$$

**Proof.** The minimum follows readily from the expression for $R(\delta,p)$. For the maximum of $R(\delta,p)$, we can prove the result using an argument of convexity.

Note that the expression of $R(\delta,0,\ldots,0)$ is a consequence of remark 1. If $n_b$ does not divide $r$, this is no longer true.

This analysis is now used to study the influence of the different architectural parameters, in order to guide the design of the vector machines. We will consider the register length and the size of the architecture.

**Influence of the register length**

First we discuss the influence of the register length, which appears to be a key parameter. We present a monotonicity result, which suggests that large register lengths are preferable.

Let us write, from formula ( 3), the mean value of $T_0$, when $\delta$ is randomly chosen in $\{1,\ldots,n_b-1\}$, and rewrite the formula ( 5) so as to emphasize the dependence on $r$.

$$ET_0=\frac{vLt_p}{n_p}+\frac{vLt_p}{n_p}(\frac{n_b}{2}-1)\frac{1}{r}+t_b-\frac{n_bt_p}{2},$$

$$ER=(\frac{5}{6}-\frac{1}{n_p})t_pn_b-\frac{t_p}{6}+\frac{vLt_p}{n_p}\frac{n_b+1}{6}\frac{1}{r}.$$

In this form, it is easy to see that $ET_0$ and $ER$ are functions of the form $\alpha\frac{1}{r}+\beta$ given by

$$ET_0 = \frac{a}{r} + b,$$
$$ER = \frac{a'}{r} + b'$$

where

$$a = \frac{vLt_p}{n_p}(\frac{n_b}{2} - 1), \quad b = \frac{vLt_p}{n_p} + t_b - \frac{n_b t_p}{2},$$
$$a' = \frac{vLt_p}{n_p}\frac{n_b+1}{6}, \quad b' = (\frac{5}{6} - \frac{1}{n_p})t_p n_b - \frac{t_p}{6}.$$

The monotonicity result follows.

**Proposition 5** *$ET_0$, $ER$, $ET$ and $Et_c$ are non increasing functions of $r$.*

**Proof.** The previous formulas give the result for $ET_0$, $ER$ and their sum, $ET$, because $a$ and $a'$ are non-negative. Unfortunately, $Et_c$ can not be expressed in terms of $ET_0$ and $ET$. However, $Et_c$, which is the mean of $t_c(\delta, p)$ over $\delta$ and $p$, is monotonic because each $t_c(\delta, p)$ is monotonic, which is not difficult to prove.

We just have noted that the mean conflict rate $Et_c$ can be expressed as a summation $\sum_{\delta,p} t_c(\delta, p)$ which is a complicated expression. To derive further results on the influence of the parameters, it is desirable to have more explicit formulas for $Et_c$, even if they are asymptotic, for example for large L. The next proposition gives such results in the case where $v = 2$.

**Proposition 6** *As $L \to \infty$,*

$$ET \sim vL\frac{t_p}{n_p}(1 + \frac{4n_b - 5}{6r}). \tag{6}$$

*Let us assume that $v = 2$ and recall that $n_b$ divides $r$. The mean of the stationary conflict rate $(Et_c)_s = \lim_{L\to\infty} Et_c$ is given by*

$$(Et_c)_s \sim \frac{n_b}{6r} \tag{7}$$

*as* $r \to \infty$ *and* $n_b \to \infty$. *More precisely* $(Et_c)_s$ *has an upper bound* $M$ *and a lower bound* $m$ *given by their expansions where* $1/r$ *is the independent variable*

$$M = \frac{1}{2rn_b(n_b-1)}(\frac{n_b^3}{3} + \frac{3n_b^2}{4} - \frac{n_b}{2} + \frac{2}{3}) + O(\frac{1}{r^2}) \tag{8}$$

$$m = \frac{1}{2rn_b(n_b-1)}(\frac{n_b^3}{3} - \frac{3n_b^2}{4} + \frac{n_b}{2} - \frac{1}{3}) + O(\frac{1}{r^2}) \tag{9}$$

*which yield asymptotic relationships as* $n_b \to \infty$*:*

$$M = \frac{n_b}{6r}(1 + \frac{13}{4n_b} + \frac{7}{4n_b^2} + \frac{15}{4n_b^3} + o(\frac{1}{n_b^3})), \tag{10}$$

$$m = \frac{n_b}{6r}(1 - \frac{5}{4n_b} + \frac{1}{4n_b^2} - \frac{3}{4n_b^3} + o(\frac{1}{n_b^3})). \tag{11}$$

*The precision of these bounds is given by*

$$\frac{M-m}{2} \sim \frac{3}{8r} \quad as \quad n_b \to \infty.$$

See the proof in the Appendix. The main result of the analysis is that the conflict rate increases linearly with the number of memory banks and decreases with the register length as $1/r$, in the limit as L approaches $+\infty$. The bounds are useful to estimate the accuracy of the first term of the expansion.

**Remark.**

The quantity $\frac{M-m}{2}$ is, in the first order, a decreasing function of $r$, independent of $n_b$ and is less than 1% when $r \geq 33$. Hence these asymptotic functions are interesting if $r$ is sufficiently large because $\frac{M-m}{2}$ is not too big (see Fig. 3).

| r | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|
| M-m/2(in %) | 9.37 | 4.69 | 2.34 | 1.17 | 0.58 | 0.29 |

**Fig. 3. Error on the approximation of the conflict rate.**

Nevertheless, for small values of $r$, it is perhaps not sufficient to replace $M$ and $m$ by their asymptotic forms when $r \to \infty$. However, for the cases of practical interest developed in Section 5, except for $n_b = 8, r = 8$ and $n_b = 16, r = 16$, it can be verified that $(Et_c)_s$ is between the expansions of $M$ and $m$ given by ( 10) and ( 11).

The threshold $r_\alpha$, the smallest $r$ such that $Et_c$ is less than a given value $\alpha$, is interesting from a practical point of view. It can be estimated by $\overline{r_\alpha} = \frac{n_b}{6\alpha}$, the smallest $r$ such that $\frac{n_b}{6\alpha}$ is less than $\alpha$, where $\frac{n_b}{6r}$ is the first term of the expansion of $\lim_{L\to\infty} Et_c$ as $r \to \infty$.

**Influence of the size of the architecture**

Using Proposition 6, let us examine the influence of the architecture size. L is assumed to be large so that $ET$ and $Et_c$ are reduced to their asymptotic forms given by relations ( 6) and ( 7). The register length is fixed. Hence $Et_c$ depends only on the number of banks and is a linear increasing function. $ET$ is inversely proportional to the bandwidth $n_p/t_p$ and a function of the number of banks which has the form $\alpha n_b + \beta$. These results should be compared to those of Bailey [1], which give an decreasing performance as the square of the number of memory banks . Therefore with an interconnection network of the same size $2n^2$, a 2n-processor, n-bank computer has better performance than an n-processor, 2n-bank computer with the same memory bandwidth.

# 5 The other priorities

The other conflict resolution schemes appear to be more difficult to study analytically than the rotation priority, because the delay is no more the same for each processor. Hence, simulations are used to study exhaustively all vector patterns. The simulator, which is called MEVAMP and written in C, is based on an explicit description of the behavior of the system as a function of time. It determines the total execution time and the conflict rate at the end for every set of vector starting addresses. The principle is to construct a time dependent process describing the state of the system big enough to be a Markov chain: the state of the process at cycle $t$ depends only on the state of the process at cycle $t-1$. The process used here is $W(t)$, which was introduced in Section 3. Moreover this evolution process has the following attractive property: given the vector starting addresses, the

evolution process is deterministic, that is, the transitions are deterministic. Hence it is easy to simulate it exactly.

In this section, we present the results obtained by the simulator for all conflict resolution schemes, except the rotation priority where the analytical expressions are used. These results are presented in graphical form.

In order to obtain useful simulation results, it is crucial to control the range of the different parameters. We present them below.

## 5.1 Framework of the simulations.

The parameters of the vectors are their number, their stride, their length, their starting addresses. The number of vectors is fixed at two, in order to minimize the combinatorial complexity of the system. The stride is one. The vector length is fixed at 8192, which seems large enough to be significant and will be discussed later. All possible starting addresses are explored.

The parameters of an architecture are the number of processors, the number of banks, the memory occupation time, which determines the inter-request time (by relation ( 2)), the inter-block time and the register length. Small numbers ( powers of 2 up to 16) of processors and banks have been simulated in so-called *symmetrical* architectures i.e. where $n_b = n_p/2$, $n_b = n_p$, $n_b = 2n_p$, and for all values of $\delta$ between 1 and $n_b - 1$ (see the discussion of Remark 2 in Section 3, register lengths of 4, 8, 16, 32, 64, 128.

The simulations give the exact determination of the total execution time for these architectures for every vector starting addresses and every inter-block time involving conflicts. We derive the mean conflict rate, denoted $Et_c$, (called also conflict rate if it is not ambiguous), when the starting addresses and the inter-bloc time are randomly chosen, and study it as a function of the register length for a given architecture and a given conflict resolution scheme. The results are presented in Figure 5.

## 5.2 Conclusion of the analysis.

For a given architecture, the conflict resolution schemes are compared. For a given conflict resolution scheme, the influence of the architectural parameters is described. The following insights can be derived from a study of Figure 5.

**Influence of the conflict resolution scheme.**

In comparing the strategies, note that the *static priority* is not the best: the conflict rate of the *static priority* is twice the conflict rate of the *rotation priority* (except for small register lengths –4, 8 and even 16– where the *rotation priority* has a high conflict rate). For the version of the ALLIANT FX/8 with vector registers of length 32 and a static priority for solving memory conflicts, the conflict rate for accessing two vectors of stride one and length 8192 is 0.035. For the same architecture, the other conflict resolution schemes exhibit smaller conflict rates. The conflict policy gives the best results as shown in Figure 4.

| policy | static | cyclic 1 | queue | conflict | rotation |
|---|---|---|---|---|---|
| conflict rate | 0.0354 | 0.0262 | 0.0326 | 0.0215 | 0.0260 |

**Fig. 4. Influence of the conflict resolution scheme for a 8-processor, 4-bank computer with register length 32 such as the ALLIANT FX/8.**

Secondly, except for the *static priority*, for large register lengths, the strategies have similar conflict rates. This leads us to examine more closely the influence of the parameters.

**Influence of the register length.**

For all policies, the conflict rate is rapidly decreasing as the register length increases for $r$ greater than some value. The reason is that, independent of the conflict resolution scheme, the number of conflicts decreases as the register length grows. For example, the analysis of the *rotation priority* shows that $ER - (n_p - 1)t_b$ is proportional to the number of conflicts $2s - 1$ (when $n_b$ divides $r$). For the ALLIANT FX/8, when the register length is multiplied by 2 (respectively 4), the conflict rate for accessing two vectors of stride one decreases from 0.035 to 0.034 (respectively 0.020) (see static priority in Figure 5). When three vectors of stride one are accessed, the conflict rate drops from 0.075 to 0.044 (respectively 0.024) when the register lengths are doubled (respectively quadrupled) (see static priority in Figure 7). A second observation concerns the conflict rate for small values of the register length. For small architectures (4-processor, 4-bank and 8-processor, 4-bank), the conflict rate decreases as $r$ increases. However for larger architectures, there is a maximum on the curve representing the conflict rate as a function of the

register length. The bigger architecture, the larger the peak. Furthermore, the position of the peak moves to the right as the size of the architecture increases. This is difficult to explain. Thirdly, note that the *queue-discipline* is good for any register length: the conflict rate is low for large register lengths and this policy has one of the lowest maxima of the conflict rate, especially for the biggest architectures: it is the best for 16-processor, 8-bank, 8-processor, 16-bank and 16-processor, 16-bank computers. It is an attractive solution for all the register lengths.

**Influence of the size of the architecture.**

Concerning the influence of the architecture, a similar behavior to that of the *rotation priority* is also observed for the other strategies: the conflict rate depends especially on the number of banks. The *cyclic priority* and the *queue-discipline* are good examples (see Figure 5). The architectures are generally such that a 2n-processor, n-bank computer performs better than an n-processor, 2n-bank computer which itself performs better than a 2n-processor, 2n-bank computer. Note the low conflict rate, given large register lengths, for the 2n-processor, n-bank computers. The ALLIANT FX/8 has this type of architecture.

# 6 Extension of the results

In this section, some extensions of the simulations with MEVAMP are presented. While the simulations presented in the last section indicated the influence of the register length, the strategies and the size of the architecture on performance, these additional simulation results allow us to study the influence of some other parameters: vector length, the number of vectors and the length of the vector stride.

**The stationary phase**

We check whether the conflict rate presented for the vector length 8192 in Section 5 is a good approximation of the stationary conflict rate. It depends on the length of the startup phase, which is calculated by simulations, as is the stationary conflict rate, using the framework presented in Section 5. The values of the length of the startup phase are not presented here, but they show that, except for the rotation priority, for large values of the register length, a vector length of 8192 is not sufficient to characterize the stationary

phase and even to attain it. Stationary conflict rates for accessing two vectors of stride one for a variety of policies and architecture are presented in Figure 6. They are generally lower than the conflict rates for a vector length of 8192 (compare Fig. 6 to Fig. 5). From Proposition 2 and Corollary 3, we have the following result for the rotation priority:

**Proposition 7** *For an architecture of $n_p$ processors with register length $r$ (a multiple of $n_b$), $n_b$ banks, the delay $R_{stat}$, due to conflicts during a period of the stationary phase, is a function of $\delta$ and $p = (p_i, 1 \leq i \leq v-1)$, and is given by*

$$R_{stat}(\delta, p) = t_p[\sum_{i=1}^{v-1}((p_i - r) \ mod \ n_b - \delta + 1)^+ + ((-\sum_{i=1}^{v-1} p_i - r) \ mod \ n_b - \delta + 1)^+].$$

*Moreover if $\delta \leq n_b$, the mean $ER_{stat}(\delta)$ of $R_{stat}(\delta, n)$, when the starting addresses are assumed to be independent and uniformly distributed on the banks, is*

$$ER_{stat}(\delta) = vt_p(n_b - \delta)(n_b - \delta + 1)/2n_b.$$

*The mean $ER_{stat}$ of $ER_{stat}(\delta)$ where $\delta$ is uniformly distributed on the set of values $\{1, \ldots, n_b - 1\}$ corresponding to the cases of conflicts, is given by*

$$ER_{stat} = vt_p(n_b + 1)/6.$$

Moreover the stationary conflict rate $\lim_{L\to\infty} t_c(\delta, p)$ can be derived and from exact expressions for its mean, it is possible to derive bounds and asymptotics for two vectors as seen in Proposition 7 .

**The case of three vectors**

To execute a vector operation, at least three vectors must be loaded or stored in the memory. It is therefore interesting to study the influence of the number of vectors accessed by the processors on the performance. Unfortunately the exhaustive simulations, for more than three vectors, become quite tedious, because the possible combinations of the starting addresses grows as $(n_b)^v$. A more limited set of parameters were simulated for three vectors ( see Fig. 7) .

For the rotation priority, the mean delay due to one period of the stationary phase is proportional to the number of vectors (see Proposition 7).

This is important because, for this policy, the stationary phase is quite representative of the total execution.

For the other policies, the conflict rate for accessing three vectors is nearly the same as the conflict rate for accessing two vectors for large register lengths or even smaller. However, for small register lengths, the conflict rate is higher for three vectors than for two. Moreover the peak phenomenon, which was particularly noticeable on large architectures when two vectors were accessed, tends to disappear. The conflict rate becomes a non-increasing function of the register length for most of the architectures studied except for the largest ones. This shows that large register lengths yield better performance. All of the policies considered, except the static one, exhibit better performance than the rotation priority when the register lengths are large.

**The case of vectors of non unit strides**

In order to study the influence of the stride of the vectors, simulations were performed to compute the conflict rate for two vectors of length 8192 and varying strides for every policy and every architecture as a function of the register length. Experiments were performed for the case when the first vector has stride one and the second has stride two; and for the case where both vectors have stride two.

The simulations show that the mean conflict rate is higher than for vectors of stride one: for the architectures studied, it can reach up to 0.50, which we consider to be high.

Notice that in this case the conflict rate is an increasing function of the register length. Moreover the influence of this parameter is quite strong. For example, for the ALLIANT FX/8, the mean conflict rate for accessing two vectors of stride two is 0.24 for a register length 32 (0.15, for a register length 4 and 0.43, for a register length 128).

The influence of the architecture is different for small and large register lengths. For small register lengths, the conflict rate is two or three times higher for the biggest architectures studied than for smaller architectures. For large register lengths, the conflict rate increases by 10 to 20 per cent increasing the size of the architecture in the range of sizes studied.

# 7 Conclusion

The purpose of this paper was the analysis of the regular reference streams that occur in the context of basic vector operations on shared memory multiprocessors with finite vector registers. This allowed us to estimate the degradation of memory bandwidth due to memory conflicts. The effects of these conflicts on memory performance were studied for a variety of conflict resolution schemes, both those that are well-known and some new-ones presented here.

Analytical results are presented for one conflict resolution scheme, the rotation strategy. For the other conflict resolution schemes, exhaustive simulations were performed for all vector patterns when two vectors are accessed. These results show the influence of a variety of parameters on the conflict rate, our primary performance measure. The two main parameters studied were the choice of conflict resolution scheme and the register length. A good choice of strategy can reduce the conflict rate by a factor of two. In particular, the static strategy exhibits poor performance compared to the other strategies.

Furthermore, we observed that when the register length is large, the conflict rate is considerably reduced for vectors of stride one, which represent the main reference streams. The rotation strategy, analytically studied, exhibits good performance for a large range of register lengths. In this case, the conflict rate is inversely proportional to the register length. As shown by analytical expansions, the conflict rate grows linearly with the number of banks, the only architectural parameter upon which it depends. Our simulations show also that the stationary conflict rate, as the vector length tends to $+\infty$, is less than the conflict rate for a fixed vector length. Owing to the explosion in the number of the vector patterns, the entire range of architectural parameters was not simulated for the case where three vectors are accessed simultaneously. However, limited experimentation suggests that the system performs better when three vectors are accessed than when two vectors are accessed simultaneously.

# Acknowledgment

I would like to thank A.V. Veidenbaum for his helpful suggestions about arbitration techniques, W. Jalby for helpful discussions and H. Bui and M. Badel for their contribution to the simulator.


# Appendix

**Proof of Proposition 2.** To prove the proposition, first we prove the following lemma.

**Lemma 8** *Assume that each processor has access to the same sequence of data divided into $t$ blocks of $r_k$ data stored cyclically in the banks, each with first address $a_k$, $(t \leq +\infty, 0 \leq k \leq t-1)$. Let $T_k$ be the time when processor 0 obtains the access to bank after requesting the first element of the k-th block. Assume that for every $k$, $r_k \geq n_b$. The sequence $(T_k, 0 \leq k \leq t-1)$ is given by*

$$T_0 = 0,\ T_k = T_{k-1} + R_k + r_{k-1}t_p \quad (1 \leq k \leq t-1)$$

*where*

$$R_k = t_p((a_k - a_{k-1} - r_{k-1})\ mod\ n_b - \delta + 1)^+ \quad (1 \leq k \leq t-1)$$

*$R_k$ is the delay each processor experiences due to the k-th conflict $(1 \leq k \leq t-1)$ when it requests the first address of the k-th block . The delay due to the 0-th conflict is $it_b$ for processor $i$ $(1 \leq i \leq n_p - 1)$. Therefore the sequence $(T_{k,i}, 0 \leq k \leq t-1)$ of the first access times of processor $i$ is given by*

$$T_{k,i} = T_k + it_b \quad (1 \leq i \leq n_p - 1).$$

**Proof of the lemma.**

Assume that $\delta = 1$.

**Step 1:**

The i-th processor has access to $a_0$, the first bank required, at time $it_b$ $(1 \leq i \leq n_p - 1)$.

At time 0, there is a conflict because all the processors request an address in bank $a_0$. Hence it is easy to see that, at time $it_b$, the i-th processor has the highest priority for this bank which becomes available.

**Step 2:**

During the time interval $]jt_p, jt_p + t_b[$ $(1 \leq j \leq n_b - 1)$, all the banks between $a_0$ and $a_0 + j$ are busy.

This is because processor 0 requests an address at bank $a_0 + j$ at time $jt_p$ and, due to relation ( 2), a bank becomes busy again as soon as it becomes available.

**Step 3:**

During the time interval $](n_b - 1)t_p, r_0 t_p[$ $(r_0 \geq n_b)$, all the banks are busy and the processors access the banks successively. The time $n_b t_p$ when processor 0 requests the bank $a_0$ for the second time is equal to $n_p t_b$, the time when processor $n_p - 1$ finishes with bank $a_0$. Thus, all the banks are busy during the time interval $](n_b - 1)t_p, (n_b + 1)t_p[$ and they remain busy until processor 0 requests the first address $a_1$ of the second block, at time $r_0 t_p$.

**Step 4:**

The sequence $(T_k, 0 \leq k \leq t - 1)$ is given by

$$T_0 = 0,\ T_k = T_{k-1} + R_k + r_{k-1} t_p \quad (1 \leq k \leq t - 1)$$

and the first access times of processor $i$ are given by

$$T_{k,i} = T_k + i t_b \quad (1 \leq i \leq n_p - 1).$$

Moreover, during the time interval $[T_k + (n_b - 1)t_p, T_k + r_k t_p[$, all the banks are busy and the processors request the banks successively without delay.

This step can be proved by recursion on $k$. The case $k = 0$ is already proved. Assume that the proposition is true for $k - 1$ and let us prove it for $k$. At time $T_{k-1} + r_{k-1} t_p$ when processor 0 requests bank $a_k$, processor $n_p - 1$ finishes its access at bank $(a_{k-1} + r_{k-1})\ mod\ n_b$. The processor 0 waits at bank $a_k$ until processor $n_p - 1$ has visited all banks from $(a_{k-1} + r_{k-1} + 1) mod\ n_b$ to $a_k$, i.e. $a_k - a_{k-1} - r_{k-1}\ mod\ n_b$ banks. This gives

$$R_k = t_p(a_k - a_{k-1} - r_{k-1})\ mod\ n_b.$$

Then processor 0 finds the banks containing the other data of the $k$-th block available when it requests data from them. As for processor $i$, it will experience the same delay as processor 0 at bank $a_k$, $it_p$ cycles later. This proves step 4. Step 5 is then evident.

**Step 5:**

If $\delta \geq 1$,

$$R_k = t_p((a_k - a_{k-1} - r_{k-1}) \; mod \; n_b - \delta + 1)^+ \quad (1 \leq k \leq t-1).$$

**Proof of the proposition.**

Each processor has the same sequence of memory requests because $n_b$ divides $r$. Given the hypotheses of the proposition, the sequence of the first relative addresses of the blocks is periodic with period $v$ and has the following values:

$$p_1, p_2, \ldots, p_{v-1}, -\sum_{i=1}^{v-1} p_i, \ldots, p_{v-1}.$$

Moreover $t = vs - 1$. This ends the proof.

**Proof of Proposition 7.**

Relation ( 4) in the particular case where $v$ is two, assuming that $n_b$ divides $r$, gives:

$$R(\delta, p) = (n_p - 1)t_b + st_p(p \; mod \; n_b - \delta + 1)^+ + (s-1)t_p((n_b - p) \; mod \; n_b - \delta + 1)^+.$$

By defining the functions $g_1$ and $g_2$ as follows

$$g_1(p) = (p \; mod \; n_b - \delta + 1)^+ \;\; and \;\; g_2(p) = (n_b - p \; mod \; n_b) - \delta + 1)^+,$$

we will obtain more specific expressions for

$$Et_c(\delta) = \frac{1}{n_b} \sum_{p=0}^{n_b - 1} (1 - \frac{T_0(\delta)}{T_0(\delta) + R(\delta, p)})$$

for different values of $\delta$. In particular, if $\delta \geq \frac{n_b}{2}$, then

$$Et_c(\delta) = \frac{1}{n_b} \left( \sum_{p=1}^{n_b - \delta} (1 - \frac{a}{b_1 p + c_1}) + (2\delta - n_b)(1 - \frac{a}{b_0}) + \sum_{p=\delta}^{n_b - 1} (1 - \frac{a}{b_3 p + c_3}) \right)$$

where

$$\begin{aligned} a &= T_0(\delta), \\ b_1 &= -(s-1)t_p, \\ c_1 &= T_0(\delta) + (n_p - 1)t_b + (s-1)t_p(b + 1 - \delta), \\ b_0 &= T_0(\delta) + (n_p - 1)t_b, \\ b_3 &= st_p, \\ c_3 &= T_0(\delta) + (n_p - 1)t_b + st_p(1 - \delta). \end{aligned}$$

If $\delta < \frac{n_b}{2}$, then

$$Et_c(\delta) = \frac{1}{n_b}\left(1 - \frac{a}{b_0} + \sum_{p=1}^{\delta-1}(1 - \frac{a}{b_1 p + c_1}) + \sum_{p=\delta}^{n_b-\delta}(1 - \frac{a}{b_2 p + c_2}) \sum_{p=n_b-\delta+1}^{n_b-1}(1 - \frac{a}{b_3 p + c_3})\right)$$

where

$$\begin{aligned} b_2 &= t_p, \\ c_2 &= t_p(2sr + sn_b - 1). \end{aligned}$$

Using the respective signs of $b_i$ $(1 \leq i \leq 3)$, the summations are of the form $\sum_{k_1}^{k_2} f(k)$ where $f$ is monotone. Furthermore, if $f$ is non-increasing,

$$\int_{k_1}^{k_2+1} f \leq \sum_{k=k_1}^{k_2} f(k) \leq \int_{k_1-1}^{k_2} f$$

and if $f$ is non-decreasing,

$$\int_{k_1-1}^{k_2} f \leq \sum_{k=k_1}^{k_2} f(k) \leq \int_{k_1}^{k_2+1} f.$$

These two inequalities permit us to obtain bounds on $Et_c(\delta)$ of the form

$$m(\delta) \leq Et_c(\delta) \leq M(\delta).$$

The stationary values of $m(\delta)$ and $M(\delta)$ can be calculated using any symbolic computation package. The following formulas were obtained:

$$\lim_{s\to\infty} m(\delta) = \begin{cases} \frac{2}{n_b}(n_b - \delta - 2(\delta + r - 1)ln\frac{\delta+2r-2+n_b}{2(\delta+r-1)}) & if\ \delta \geq \frac{n_b}{2} \\ 1 - \frac{1}{n_b} - \frac{(\delta+r-1)(n_b-2\delta+1)}{rn_b(2r+n_b)} - \frac{4}{n_b}(\delta + r - 1)ln\frac{\delta+2r-1+n_b}{n_b+2r-1} & if\ \delta < \frac{n_b}{2} \end{cases}$$

and

$$\lim_{s\to\infty} M(\delta) = \begin{cases} \frac{2}{n_b}(n_b - \delta + 2(\delta + r - 1)ln\frac{\delta+2r-1+n_b}{2\delta+2r-1}) & if\ \delta \geq \frac{n_b}{2} \\ 1 - \frac{1}{n_b} - \frac{(\delta+r-1)(n_b-2\delta+1)}{rn_b(2r+n_b)} - \frac{4}{n_b}(\delta + r - 1)ln\frac{\delta+2r-1+n_b}{n_b+2r} & if\ \delta < \frac{n_b}{2} \end{cases}.$$

Then, using again the comparison between the summation and the integral in the second summation, we get

$$m \leq \frac{1}{n_b}\sum_{\delta=1}^{n_b-1} m(\delta) \leq Et_c \leq \frac{1}{n_b}\sum_{\delta=1}^{n_b-1} M(\delta) \leq M,$$

where the values of $m$ and $M$ are given by ( 8) and ( 9). Expansions of the bounds $m$ and $M$, when $n_b$ tends to $\infty$, are easy to derive.

# References

[1] D.H. Bailey, Vector computer memory bank contention, IEEE Trans. Comp. 36 (3), pp. 293-298, March 1987.

[2] D.P. Bhandarkar, Analysis of memory interference in multiprocessors, IEEE Trans. Comp. 24 (9), pp. 897-908, Sept. 1975.

[3] F. Baskett and A.J. Smith, Interference in multiprocessor computer systems with interleaved memory, Communications of the ACM 19 (6), pp. 327-334, June 1976.

[4] P. Budnick and D.J. Kuck, The organization and use of parallel memories, IEEE Trans. Comp. 20 (12), pp. 1566-1579, Dec. 1971.

[5] T. Chung and J.E. Smith, A simulation study of the CRAY X-MP memory system, IEEE Trans. Comput. 35 (7), pp. 613-622, July 1986.

[6] M. Crehange, J-M. Frailong, B. Plateau, Performance of a vector computer with memory contention, Proceedings, High Performance Computer Systems, 1987.

[7] W. Oed and O. Lange, Modelling, measurement, and simulation of memory interference in the CRAY X-MP, Parallel Computing 3, pp. 343-358, Oct. 1986.

[8] W. Oed and O. Lange, On the effective bandwidth of interleaved memories in vector processor systems, IEEE Trans. Comput. 34 (10), pp. 949-957, Oct. 1985.

[9] B. Ramakrishna Rau, Interleaved memory bandwidth in a model of a multiprocessor computer system, IEEE Trans. Comp. 28 (9), pp. 678-681, Sept. 1979.

[10] K.V. Sastry and R.Y. Kain, On the performance of certain multiprocessor computer organizations, IEEE Trans. Comp. 24 (11), pp. 1066-1074, Nov. 1975.

[11] D.W.L. Yen, J.H. Patel, E.S. Davidson, Memory interference in synchronous multiprocessor systems, IEEE Trans. Comp. 31 (11), pp. 1117-1121, Nov. 1982.

| time | memory banks 0 1 2 3 |
|---|---|
| 1 | 0 - - - |
| 2 | 1 0 - - |
| 3 | 2 1 0 - |
| 4 | 3 2 1 0 |
| 5 | - 3 2 1 |
| 6 | - - 3 0 |
| 7 | 0 - - 1 |
| 8 | 1 0 - 2 |
| 9 | - 1 0 3 |
| 10 | - - 1 2 |
| 11 | 0 - - 3 |
| 12 | 1 0 - - |
| 13 | 2 1 - - |
| 14 | 3 2 1 - |
| 15 | - 3 2 1 |
| 16 | - - 3 0 |
| 17 | 0 - - 1 |
| 18 | 1 0 - - |
| 19 | 2 1 0 - |
| 20 | 3 2 1 - |
| 21 | 0 3 2 - |
| 22 | 1 0 3 2 |
| 23 | - 1 0 3 |
| 24 | - - 1 0 |
| 25 | - - - 1 |
| 26 | - - - 0 |
| 27 | 0 - - 1 |
| 28 | 1 0 - 2 |
| 29 | 2 1 0 3 |
| 30 | 3 2 1 - |
| 31 | 0 3 2 - |
| 32 | 1 0 3 - |
| 33 | 2 1 0 - |
| 34 | 3 2 1 0 |

**Fig. 9. Conflicts for a 4-processor, 4-bank computer, static priority and register length 4 (inter-block time $2t_p$ and starting memory banks 0 and 3)**

| time | memory banks 0 1 2 3 |
|---|---|
| 1 | 0 - - - |
| 2 | 1 0 - - |
| 3 | 2 1 0 - |
| 4 | 3 2 1 0 |
| 5 | - 3 2 1 |
| 6 | - - 3 2 |
| 7 | - - - 3 |
| 8 | - - - 0 |
| 9 | 0 - - 1 |
| 10 | 1 0 - 2 |
| 11 | 2 1 0 3 |
| 12 | 3 2 1 - |
| 13 | 0 3 2 - |
| 14 | 1 0 3 - |
| 15 | 2 1 0 - |
| 16 | 3 2 1 0 |
| 17 | - 3 2 1 |
| 18 | - - 3 2 |
| 19 | - - - 3 |
| 20 | - - - 0 |
| 21 | 0 - - 1 |
| 22 | 1 0 - 2 |

**Fig. 10. Conflicts for an 4-processor, 4-bank computer, rotation priority and register length 4 (inter-block time $t_p$ and starting memory banks 0 and 3)**

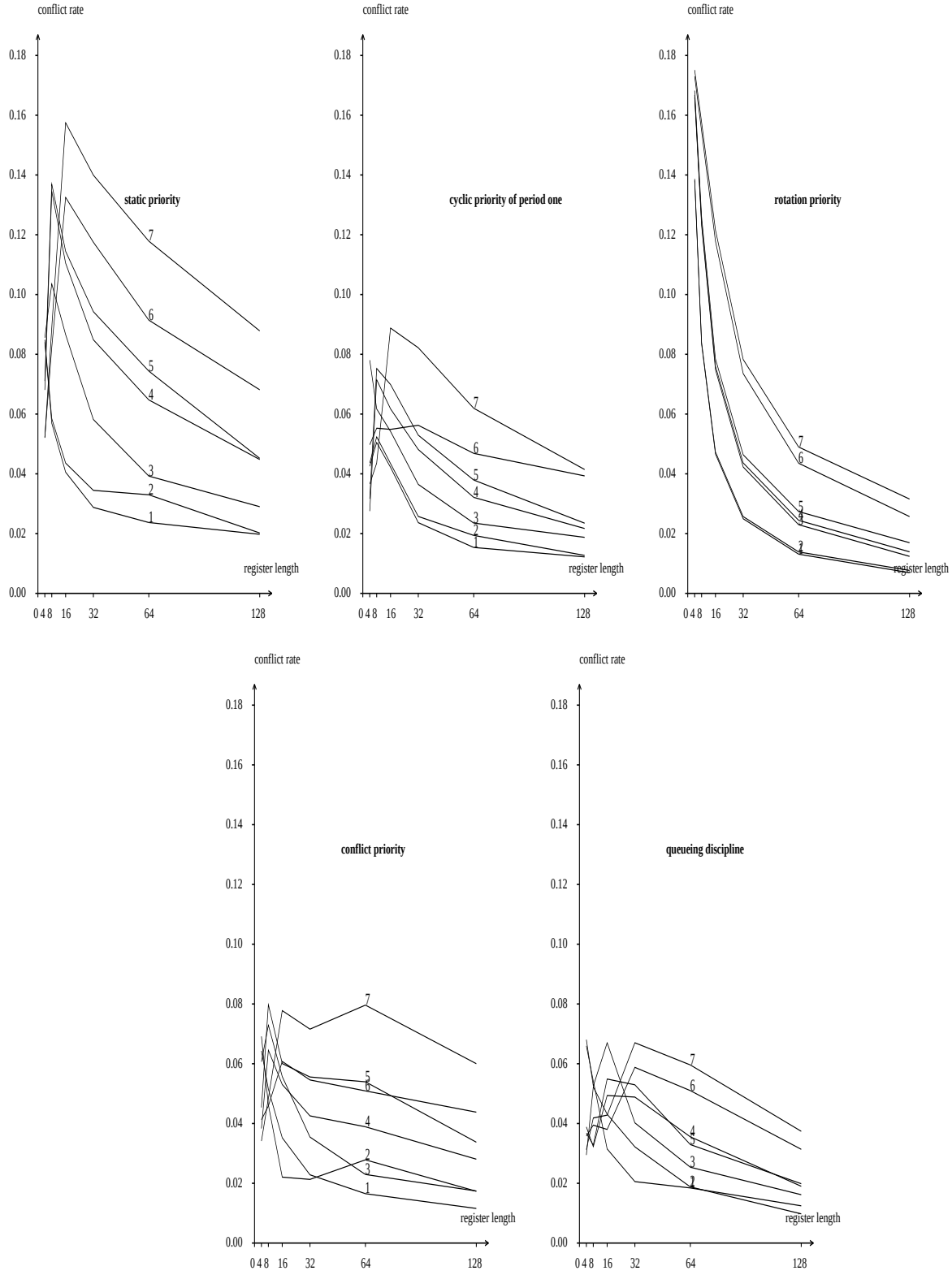


**Fig. 5. Conflict rate for accessing two vectors of stride one under varying policies. Architectures with $n_p$ processors and $n_b$ banks were considered for the following values of $n_p$ and $n_b$ : 1: 4-4, 2:8-4, 3:4-8, 4:8-8, 5:16-8, 6:8-16, 7:16-16.**

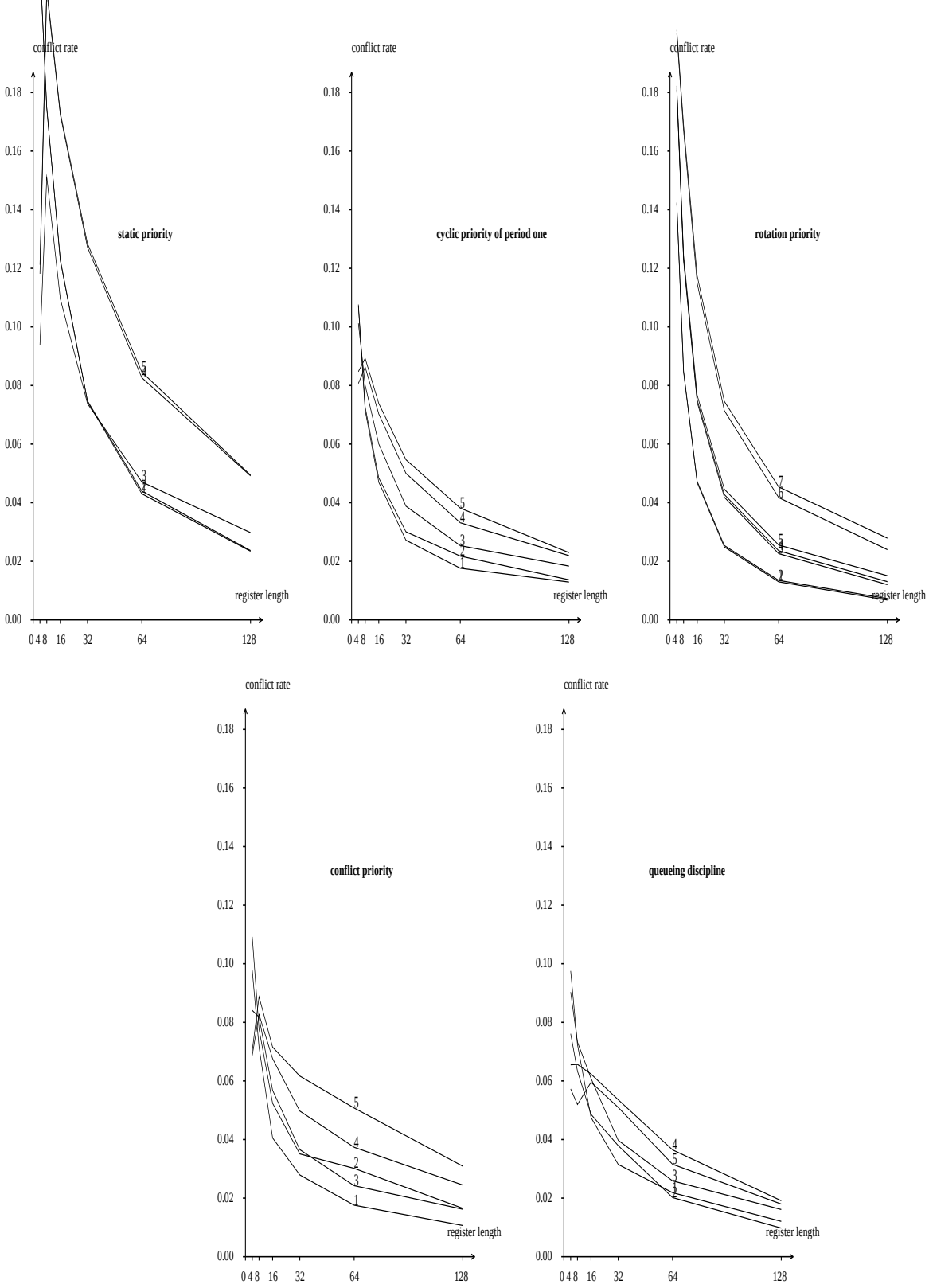


**Fig. 7. Conflict rate for accessing three vectors of stride one under varying policies Architectures with $n_p$ processors and $n_b$ banks were considered for the following values of $n_p$ and $n_b$ : 1: 4-4, 2:8-4, 3:4-8, 4:8-8, 5:16-8, 6:8-16, 7:16-16.**

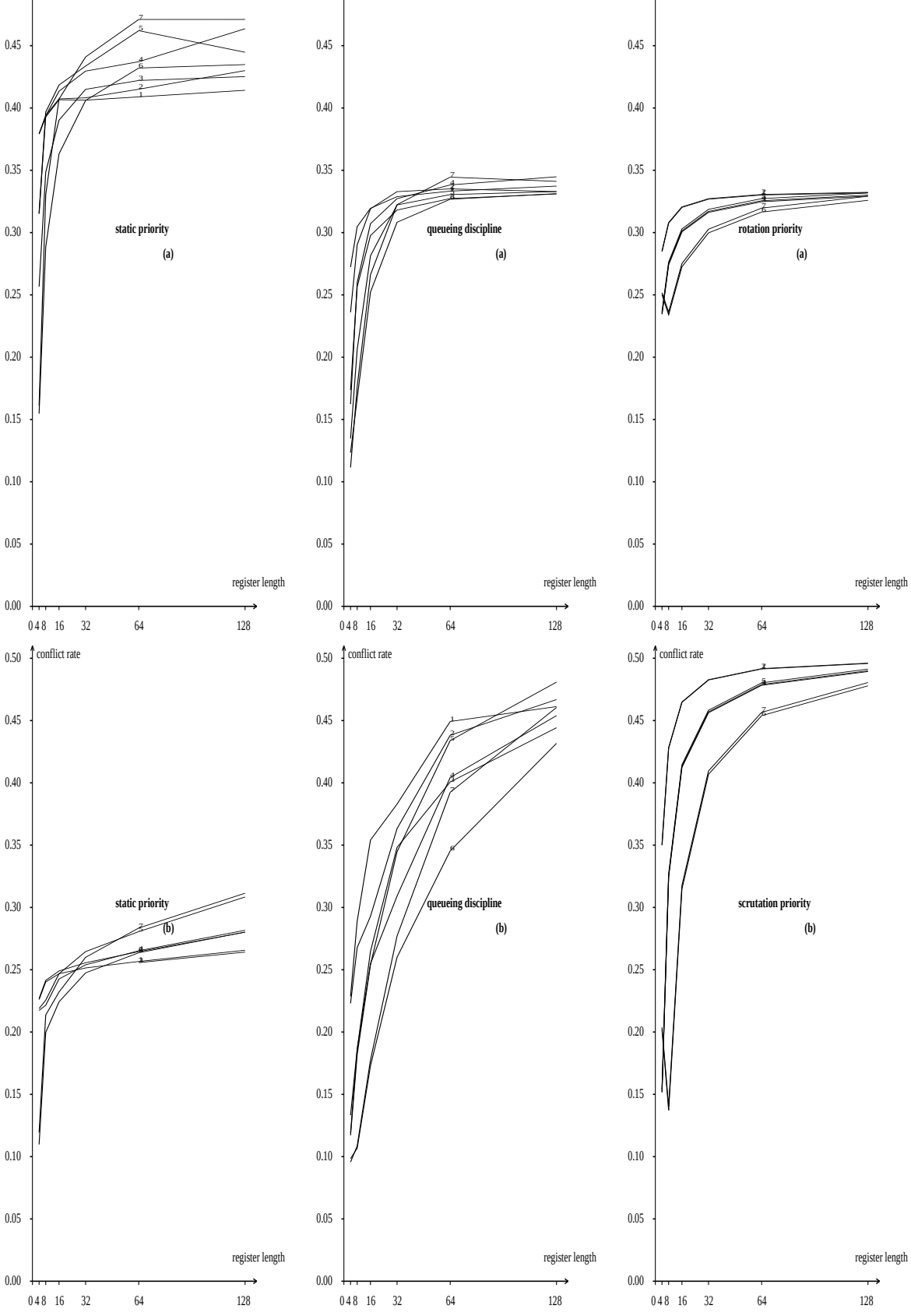


**Fig. 8. Influence of the stride on memory contention accessing two vectors of length 8192:**

**a. 2 vectors of stride 1 and 2 respectively;**

**b. 2 vectors of stride 2;**

**Architectures with $n_p$ processors and $n_b$ banks were considered for the following values of $n_p$ and $n_b$ : 1: 4-4, 2:8-4, 3:4-8, 4:8-8, 5:16-8, 6:8-16, 7:16-16.**

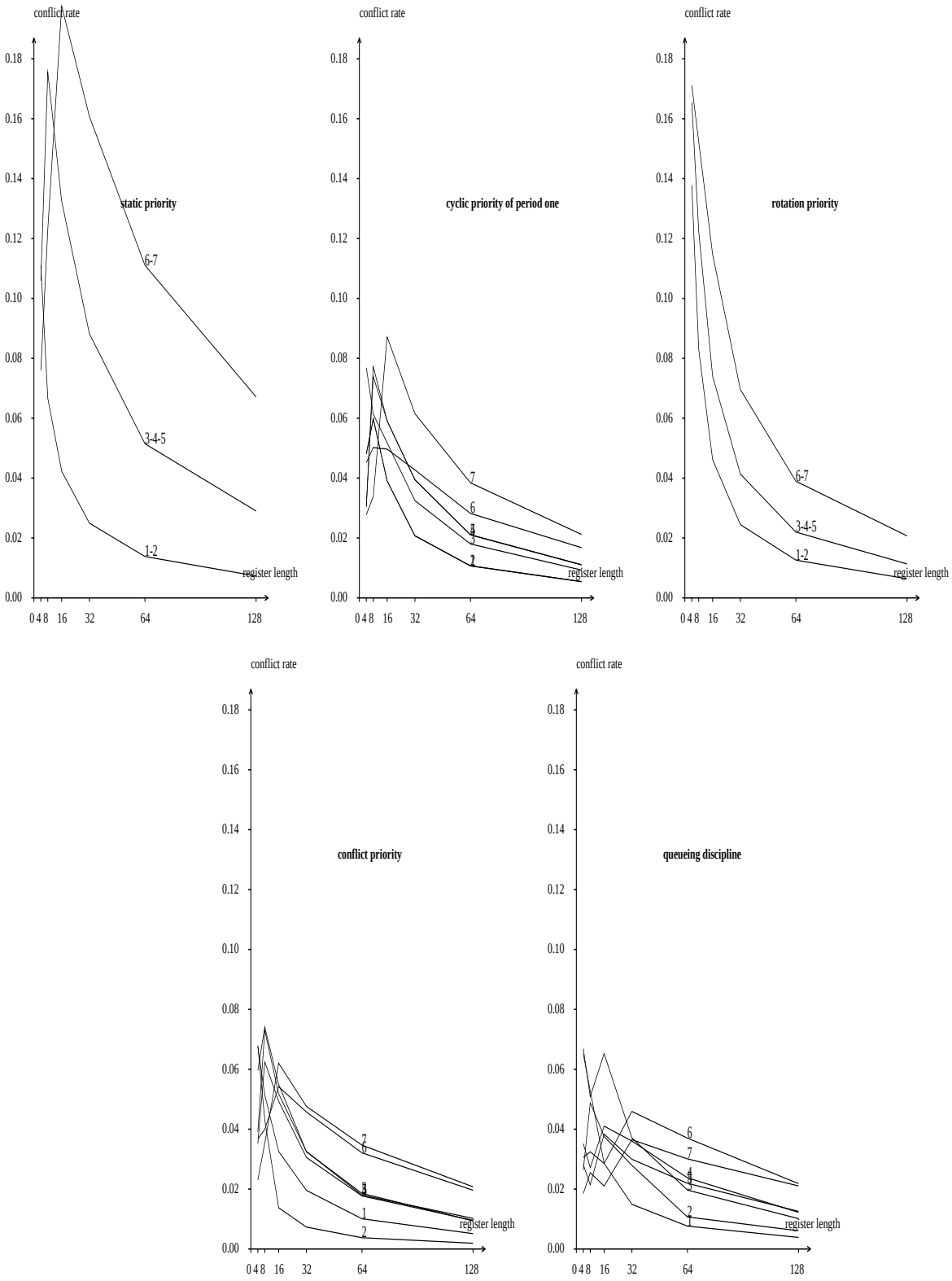


**Fig. 6. Stationary conflict rate for accessing two vectors of stride one under varying policies. Architectures with $n_p$ processors and $n_b$ banks were considered for the following values of $n_p$ and $n_b$ : 1: 4-4, 2:8-4, 3:4-8, 4:8-8, 5:16-8, 6:8-16, 7:16-16.**