

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the original text directly from the copy submitted. Thus, some dissertation copies are in typewriter face, while others may be from a computer printer.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyrighted material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is available as one exposure on a standard 35 mm slide or as a 17" × 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. 35 mm slides or 6" × 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.



Accessing the World's Information since 1938

300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

Order Number 8900252

**The effects of cache coherence on the performance of parallel
PDE algorithms in multiprocessor systems**

Johnson, Sandra Kay, Ph.D.

Rice University, 1988

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy ✓
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages _____
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages ✓
15. Dissertation contains pages with print at a slant, filmed as received _____
16. Other _____

U·M·I

RICE UNIVERSITY

The Effects of Cache Coherence on the Performance
of Parallel PDE Algorithms in Multiprocessor Systems

by

SANDRA KAY JOHNSON

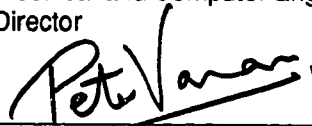
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

DOCTOR OF PHILOSOPHY

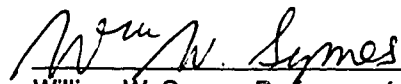
APPROVED, THESIS COMMITTEE:



Faye A. Briggs, Associate Professor of
Electrical and Computer Engineering,
Director



Peter J. Varman, Assistant Professor of
Electrical and Computer Engineering



William W. Symes, Professor of
Mathematical Sciences

Houston, Texas
May, 1988

The Effects of Cache Coherence on the Performance of Parallel PDE Algorithms in Multiprocessor Systems

by

Sandra Kay Johnson

ABSTRACT

The advent of parallel processing systems has resulted in the potential for increased performance over traditional uniprocessor systems. However, while there has been significant advances in developing these systems, designing parallel algorithms to run on them has not kept up with the pace. Although parallel algorithms have been studied in the literature, very little has been done in studying how various architectural features effect the performance of these algorithms. This thesis presents the results of a study conducted to determine how one particular design feature of a parallel processing architecture, cache coherence maintenance, affects the performance of parallel partial differential equations' (PDE) algorithms.

A high performance shared-memory multiprocessor architecture with private caches and a single bus or full crossbar interconnection network is assumed. The performance degradation as a result of using a directory based cache coherence protocol is evaluated on specific implementations of three synchronous parallel PDE algorithms (Jacobi's algorithm, red-black successive over-relaxation or SOR and the preconditioned conjugate gradient algorithm or PCG). A trace driven cache simula-

tor determines this degradation. The trace is obtained by symbolically executing the algorithm on the multiprocessor system. Parameters derived to evaluate the performance degradation are used as input into an execution time model used to calculate the time needed to execute one iteration of each algorithm. This facilitates parallel algorithm speedup calculations over the sequential algorithm as well as over the parallel algorithm without cache coherence.

The results show that implementing cache coherence degrades the overall performance of the parallel PDE algorithms considered by 10 to 30 percent. Various cache design features such as the cache blocksize, the mapping function and the cache size and the algorithm design feature considered, the PDE grid decomposition strategy, have no appreciable effects on the algorithm performance degradation. In fact, the major factors affecting this degradation are the cache miss ratio, the size of the PDE grid relative to the cache size and the write probability of the parallel algorithm. Finally, for the coherence protocol used in this study, SOR has the best speedup performance, followed by Jacobi's algorithm and PCG, respectively.

Acknowledgements

I would like to express my sincere appreciation to my research advisor, Dr. Faye' A. Briggs. Thank you for your support, encouraging words, and technical counsel. I would also like to thank Dr. William W. Symes for your technical advice on PDEs and for your membership on my research committee. Furthermore, many thanks to Dr. Peter J. Varman for your counsel and membership on my research committee.

Moreover, it is with gratitude that I thank Dr. C. S. Burrus. Your inspiring words of advice will always be remembered. To Mr. Donald Schroeder, your helpful hints on *truff* have not been forgotten. Thank you for taking the time to help.

To my family, especially my beloved mother. Words cannot express my thankfulness to you. Your constant advice to study, study, study has truly paid off for me. Finally, to my dear Alfred, thank you for being there for me through thick and thin. I'm looking forward to sharing a life with you.

Table of Contents

CHAPTER 1 INTRODUCTION	1
Motivations	2
Related Work	9
Thesis Overview	12
CHAPTER 2 BACKGROUND	14
Cache Coherence	14
Classical Solution	15
Broadcast Protocols	16
The Write-Once Protocol	17
The Illinois Protocol	18
The Ownership Protocol	18
Directory Protocols	19
Coherence Protocol Simulated	20
Performance Parameters and Penalties	28
Execution Time Model	30
Simulation Methodology and Model	34
CHAPTER 3 JACOBI'S ITERATIVE ALGORITHM	38
The Classical Algorithm	38

Simulation Results	41
Miss Ratio/Miss Ratio Degradation	42
Invalidation Ratio	53
Cross-interrogate Cast-out	70
Cross-interrogate Change State	89
Prefetching Strategies	109
Multiprocessor Speedup	140
Jacobi's Conclusions	157
CHAPTER 4 SUCCESSIVE OVER-RELAXATION	160
Red-Black SOR	160
Simulation Results	162
Miss Ratio/Miss Ratio Degradation	162
Invalidation Ratio	172
Probability of a XICO	180
Probability of a XICS	180
Prefetching Strategies	192
Multiprocessor Speedup	217
SOR Conclusions	229
CHAPTER 5 THE PRECONDITIONED CONJUGATE GRADIENT	
ALGORITHM	233
Preconditioned CG	233

Simulation Results	237
Miss Ratio/Miss Ratio Degradation	237
Invalidation Ratio	244
prXICO	244
prXICS	250
Prefetching Strategies	258
Multiprocessor Speedup	274
PCG Conclusions	281
CHAPTER 6 CONCLUSION	287
Algorithm Comparisons	287
Research Summaries	291
Future Work	293

CHAPTER 1

INTRODUCTION

The advent of parallel processing systems has resulted in the potential for increased performance over traditional uniprocessor systems. While there has been significant advances in developing these systems, designing parallel algorithms to run on them has not kept up with the pace. Parallel algorithms present innovative and efficient means of handling sophisticated problems and allow the user to investigate larger and/or more complex problems. They offer speedup over their corresponding sequential algorithms resulting in increased system throughput. This increased performance is exhibited in a wide spectrum of applications. Matrix algorithms, partial differential equations (PDE) algorithms, and sorting and searching algorithms are a few examples used in such applications as computational physics, aerodynamics, image processing and artificial intelligence.

There are various performance metrics that may be used in evaluating the performance of parallel algorithms. Four paradigms beneficial to the development and performance evaluation of these algorithms are discussed in [1]. The underlying system architecture is a major component of algorithm performance and should seriously be considered when evaluating them. The design space of parallel processors consists of many features such as general versus special purpose machines, shared memory versus message passing communication, tens of processors versus hundreds or even thousands of processors and processor-memory or processor-processor

interconnection topology. Therefore parallel algorithms may be optimized for execution on a particular class of machine architecture.

1.1. Motivations

Although parallel algorithms have been studied in the literature, much less has been done in studying how various architectural features effect the performance of these algorithms. This thesis presents the results of a study on how one particular design feature of a parallel processing architecture, cache coherence maintenance, affects the performance of parallel PDE algorithms. PDE algorithms were chosen because they are widely used in scientific and engineering communities and they are the subject of numerous research papers (see [2] for a comprehensive survey).

The PDE algorithms studied are based on the model problem [3]. A linear system of equations are obtained from the solution of two-dimensional, elliptic boundary value problems. The equations are derived using central differences to replace the derivatives of Laplace's equation describing a rectangular region R . The region is completely specified at the boundaries. This is known as the Dirichlet problem. A 5-point discretization of the region is used for all algorithms studied.

The high-performance parallel processing architecture used as a foundation in this study is a tightly-coupled multiprocessor system as shown in Figure 1.1. Here we see P processors connected to M memory modules via an interconnection network. There are also N I/O processors or channels to coordinate I/O activities. This architecture is similar to the Sequent Symmetry series [4] and SPUR [5], both with bus architectures, and the IBM 3090 Model 400 [6], a full crossbar system. All pro-

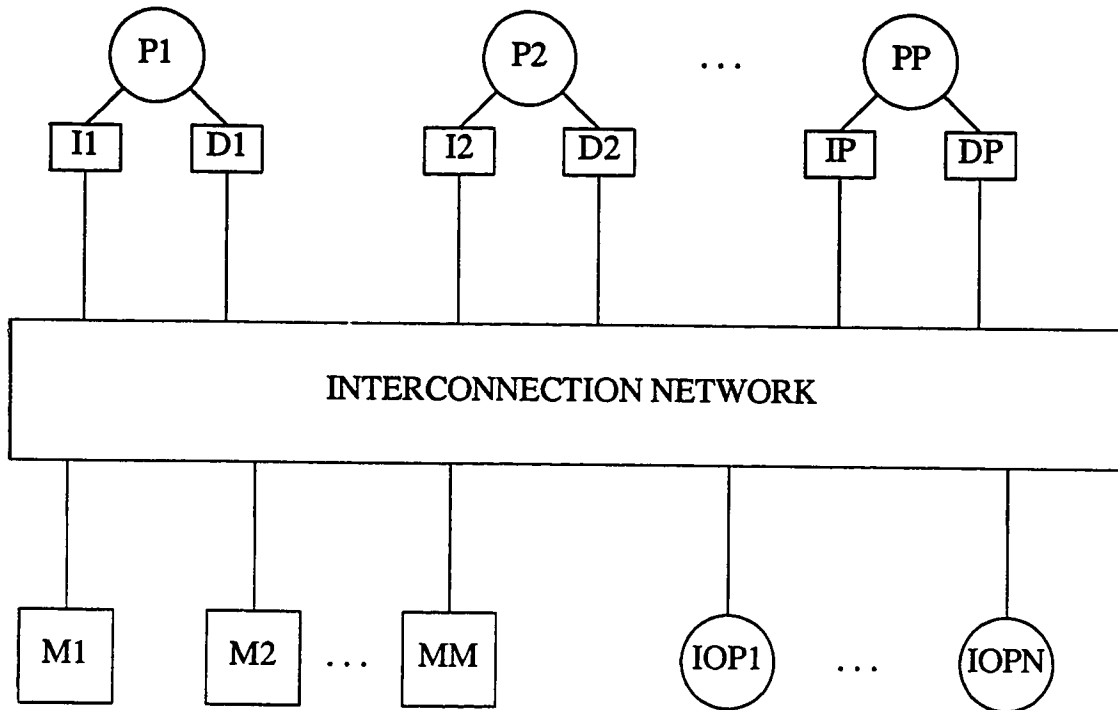


Figure 1.1. Tightly-Coupled Multiprocessor System.

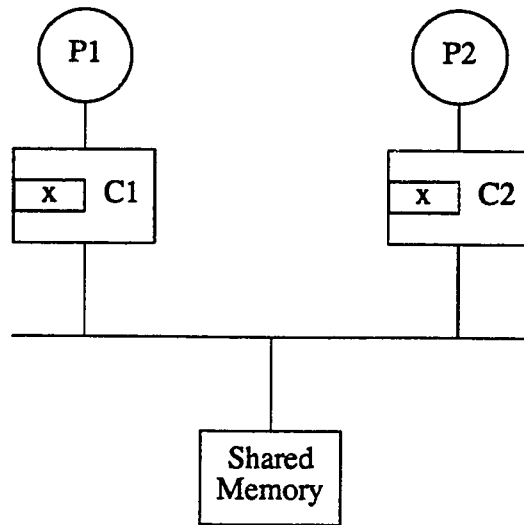
cessors share the same global memory. Each processor has its own private data and instruction caches.

Without the caches, the contention in the interconnection network and memory modules resulting from the processors' read and write requests would prohibitively degrade system performance by exhibiting unacceptable memory access times. The caches are introduced here for the same reasons they were introduced in uniprocessor systems; as high-speed buffers operating on the principle of locality effectively

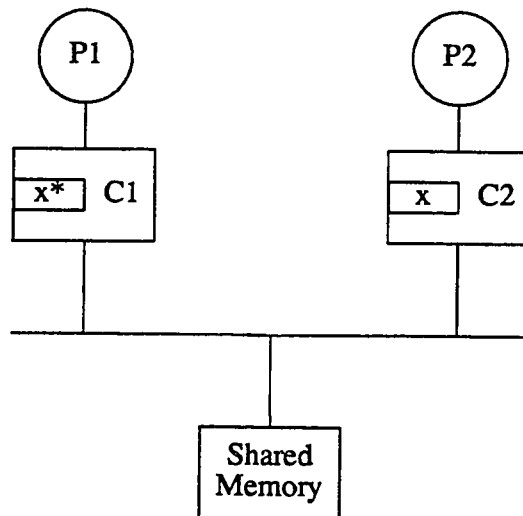
reducing the number of requests to main memory. While the inclusion of caches in uniprocessor systems significantly reduces the effective memory access time, resulting in significant performance improvements [7], the introduction of multiple caches with possible multiple copies of memory blocks may result in data inconsistencies.

There are three factors that contribute to the inconsistent data problem: (1) allowing shared modifiable data to be cached, (2) allowing processes to migrate to different processors and (3) I/O activity [8]. If shared modifiable data is allowed to be cached, consider the scenario illustrated in Figure 1.2. If two processes, A and B, running on separate processors (P1 and P2, respectively) both perform a read access to the same memory block, this block will be placed in the data cache of both processors. Process A modifies the block resulting in inconsistent data between the caches (i.e., if process B accesses this block, it will obtain stale data unless C2 is notified of the block modification by process A).

A similar situation occurs when a process is allowed to migrate as shown in Figure 1.3a. Here process A modifies a memory block while executing on P1. If the process migrates to P2 and reads this block it will obtain stale data. Figure 1.3b illustrates the inconsistencies that may occur as a result of I/O activity. Process A modifies a memory block while executing on P1 as shown in part a of this figure. If an I/O processor subsequently accesses this block it will obtain old data. If a multiprocessor system is to perform correctly, the possibility of having several different copies of the same data must be avoided.

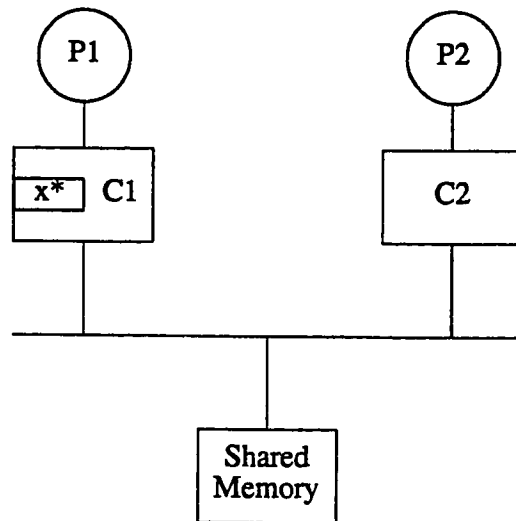


a. processes A and B read data x on P1 and P2 (respectively).

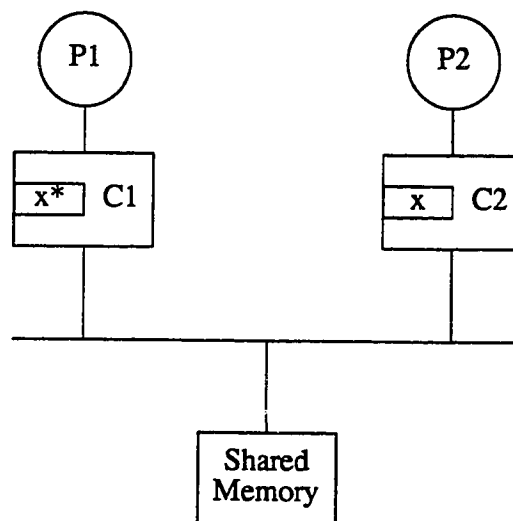


b. process A modifies data x resulting in inconsistent data between C1 and C2.

Figure 1.2. Data Inconsistencies when Caching Shared Modifiable Data.

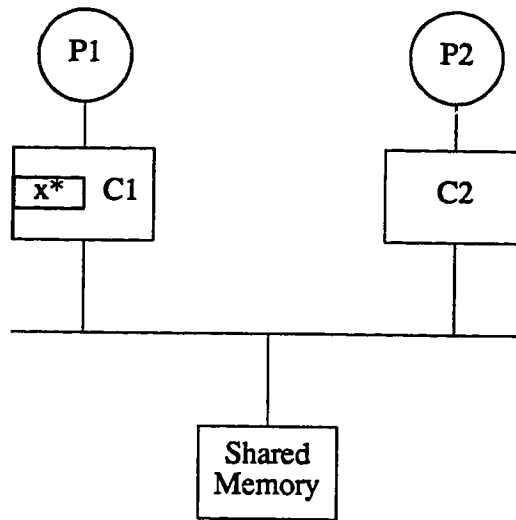


a. process A modifies data x while running on P1.

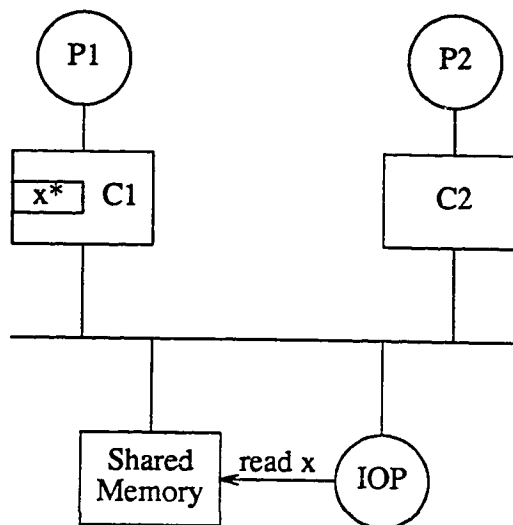


b. process A migrates to P2 and then references an old value of x .

Figure 1.3a. Data Inconsistencies when Processes Migrate.



a. process A modifies data x while executing on P1



b. the IOP references an old value of x

Figure 1.3b. Data Inconsistencies as a Result of I/O Activity

There are various methods presently available for solving this inconsistent data problem. All methods require overhead that results in some type of performance penalty. Although there has been some work done on evaluating the effects of system architecture on parallel PDE algorithm performance, there has been no previous work done on evaluating how maintaining cache coherence affects the performance of these algorithms. Since implementing a coherence protocol in multiprocessor systems may result in prohibitive performance degradation, an in-depth study is needed to discover how this architectural design feature effects algorithm performance. In this study, several parameters are defined to evaluate the particular coherence protocol used (discussed in Chapter 2). These parameters are then used in evaluating the performance of three popular synchronous iterative parallel PDE algorithms, the point Jacobi algorithm, red-black successive over-relaxation (SOR) and the preconditioned conjugate gradient algorithm (PCG).

The study was conducted to determine the extent to which implementing cache coherence degrades algorithm performance. Various cache design and other architectural features are examined by varying cache blocksize, cache size, the address mapping function, prefetching strategies and the total number of processors in the system. Two interconnection networks are also studied; single bus systems and the full crossbar network. The study was also administered to determine how PDE grid decomposition strategies affected algorithm performance while maintaining multi-cache coherence. Detailed presentations of the effective memory access times for the shared memory system and the algorithm execution time model is included to provide the performance metrics needed in evaluating system performance.

1.2. Related Work

Dubois and Briggs [9] evaluated the performance of a specific coherence protocol on a *general* workload model. Archibald and Baer [10] evaluated the effects of six coherence protocols, all based on bus architectures, using synthetic trace driven simulations of general workloads. Lee, *et. al.* [11] evaluated the effects of a software coherence algorithm on a high performance system with a pipelined multi-stage interconnection network. A general set of numerical subroutines spanning a wide range of scientific applications was used to model the workload. All of this work has evaluated the performance of coherence protocols on a *general workload*.

Fox and Otto [12] emphasized the importance of the computation to communication ratio and load balancing the execution of parallel algorithms. These two factors are the major causes of system performance degradation. Performance metrics used in this study were speedup and processor efficiency (speedup per node). They studied the solution to LaPlace's equation running on the hypercube system. The rectangular region was decomposed into square subregions with each subregion assigned to a hypercube node. One of the major differences between the hypercube and the targeted system of this study is the method of communication between processes. In the hypercube architecture, all processors have their own local memories. Processes running on processors (nodes) communicate via passing messages (either directly or indirectly). In the shared memory system used in this study, processes communicate by simply accessing the same memory block. It is the overhead incurred by implementing this communication while maintaining multi-cache coherence that is subject of this thesis.

Vrsalovic, *et. al.* [13] presented a analytic model for predicting multiprocessor performance. An iterative solution to Poisson's equation using a 5-point discretization of a square grid was used. The work concentrated on defining how the various PDE grid decomposition strategies affected the multiprocessor system speedup. Square, triangular and hexagonal grid partitionings were used and the hexagonal decomposition strategy had the best performance.

The generalized multiprocessor system used in the study consisted on N processors with associated local memory, global memory and an interconnection network for communication between the processors and global memory. All private data was placed in the local memories. Two approaches to the problem were used depending on whether or not multiple accesses to shared data by a process required maintaining local copies of it. Although not specifically stated in the paper, supporting local copies of shared data requires that they be read-only (see Figure 1.2) or modifiable when using a coherence protocol. In either case the overhead incurred may be represented by the parameters T_2 (global access time) and T_3 (processing time for copying a global data element) used in the execution time model. Their paper evaluated the performance degradation by presenting the memory access time only as a function of T_2 and T_3 . This thesis presents a more detailed performance evaluation by incorporating the effects of cache coherence into the effective memory access time.

Saltz, *et. al.* [14] presented the empirical results of a study on the solution of the heat equation by red-black SOR on the Intel iPSC Hypercube [15]. Since the communication overhead was substantially large for this system, the work concentrated

on reducing the amount of communication needed between nodes. Both rectangular and strip decomposition strategies were used and the PDE grid-size range was from 64x64 to 512x512 points. The performance metric used in this study was the communication cost per iteration. It was observed that for grid-sizes of 256x256 and smaller, rectangular decomposition provided the best performance whereas the strip decomposition performed better for the 512x512 grid-size.

Reed, *et. al.* [16] presented the analytic models to study the effects of stencil/partition/architecture trios on the performance of the solution to LaPlace's equation on a rectangular grid. Five different stencils were used (including the 5-point) with rectangular, square, triangular and hexagonal decomposition strategies and message passing and shared memory architectures. They discovered that the *trios* must be considered when evaluating multiprocessor performance. Observing only one or two components may result in suboptimal performance prediction. The shared memory architecture and execution time models presented were similar to the ones discussed by Vrsalovic, *et. al.*

Dubois [17] presented an analytic model of cache-based multiprocessors. His work included evaluating how cache coherence affected the performance of a parallel red-black SOR algorithm. However, only the fully-associative mapping strategy, one decomposition strategy and only a one-grid-element cache blocksize was considered. The work presented in this thesis varied the cache blocksize and used both the direct and 2-way set-associative mapping strategies. Furthermore, the results are based on simulations of the algorithm executions.

In [18], Dubois used an analytic model to measure the performance degradation manifested by cache coherence. In particular, an upper bound on the hit (miss) ratio degradation resulting from cache invalidations was evaluated. An infinite cache was assumed for steady state task executions. The model assumed there is no correlation between the reference streams of the processors. This is an unacceptable assumption for the synchronous algorithms considered in this study. Furthermore, realistic finite cache systems were not considered; however, it was noted that modeling such systems are very complex.

All of this previous work has concentrated on multiprocessor performance prediction when executing an iterative solution to LaPlace's or Poisson's equation; emphasizing the effects of the computation to communication ratio and the decomposition strategies on performance. The execution time models presented included a general representation for memory access times. This thesis presents a detailed presentation of the effective memory access times in shared memory systems and considers the effects of two PDE grid decomposition strategies (rectangular and square) on system performance.

1.3. Thesis Overview

Chapter 2 provides extensive background material on the cache coherence protocol used when simulating the execution of the algorithms. It also discusses the trace driven simulation philosophy used and presents a detailed explanation of the performance parameters obtained from the simulations, the penalties incurred by these parameters and the execution time model to be used in evaluating system perfor-

mance. Chapter 3 begins with a description of the parallel implementation of Jacobi's algorithm used and its simulation. The remainder of the chapter presents the performance results of the simulation and the execution time model. The same format is used in Chapters 4 and 5 for parallel implementations of red-black SOR and the PCG algorithm, respectively. Finally, the Conclusion (Chapter 6) summarizes the work presented and suggests directions for future research in this area.

CHAPTER 2

BACKGROUND

Several protocols for cache coherency have been discussed in the literature. This chapter commences with brief descriptions of these protocols. Then the performance parameters used in predicting system performance as a result of maintaining coherence are formulated. The penalties incurred for each parameter are discussed and an execution time model is derived incorporating these penalties. From this model, the algorithm speedup and the iteration time degradation are obtained. Finally, a discussion of the simulation philosophy used and the features simulated is included.

2.1. Cache Coherence

There are basically two types of cache coherence implementations; static and dynamic. The static protocol is a software controlled solution. Certain memory blocks are tagged as non-cacheable by the compiler or the user. These blocks contain shared, modifyable data such as semaphores, locks, barriers and other synchronization primitives as well as certain data structures such as job queues. Non-cacheable blocks are accessed directly from memory. Some shared blocks may be cacheable but only through critical sections, accessed (and protected) by locks. In this case, the processors are responsible for updating main memory before releasing the lock. Software protocols are usually used in systems with multi-stage interconnection networks. These systems are usually composed of hundreds or thousands of processors resulting

in prohibitive hardware complexity and memory access times for dynamic coherence solutions. The Honeywell Series 66, the Elxsi 6400 systems, the IBM RP3 [19], and the experimental VMP multiprocessor [20,21] use this solution to multi-cache inconsistencies.

The dynamic protocol may be sub-divided into four types: (1) the shared cache solution, (2) the classical solution, (3) broadcast protocols and (4) directory protocols. In the shared cache solution, all processors share a single cache. This cache is either adjacent to the processors or to primary memory. Utilizing a shared cache eliminates the coherence problem since only one cache is present in the system. This solution is generally infeasible because the bandwidth of the cache is insufficient to support the processing demands of the processors. Also, additional access delays may occur because the cache is usually physically distant from the processors. In general, all of the problems that originally led to the design of private caches such as interconnection network conflicts (only if the cache is placed adjacent to primary memory), memory access contention and access delays are present in this solution. More information on the shared cache solution may be found in [22].

2.1.1. Classical Solution

In the classical solution, all remote caches are informed of a block modification by receiving an invalidation signal that is broadcast by the cache of the requesting processor. Special invalidation busses that connect each cache to all other caches in the system are used for this broadcast. This scheme is usually used in conjunction with the write-through write policy. The invalidation traffic increases dramatically as the

number of processors increase. Consequently, this method becomes prohibitive for systems with more than two processors. This solution has been implemented on the IBM 370/168 and 3033 machines (dual processors).

2.1.2. Broadcast Protocols

The broadcast protocols are a compromise between the broadcasting on invalidations to all processors on every write access (the classical solution) and the inhibition of all ineffective invalidations. All protocols in this category are used on multiprocessors with bus architectures. Each cache can monitor the write requests of other caches by watching the bus. The remote blocks may be updated or invalidated. If the remote blocks are updated, the implementation has to be designed to prevent them from crossing. Updating the blocks require more data transfers than invalidations; however the blocks remain in the cache so the cache performance is not degraded as with invalidations. Broadcast protocols require a dual directory system to service processor and bus requests. They offer the advantages of modularity and extensibility of system design.

Broadcast implementations include the *write-once* scheme [23], and an extension (the CMU protocol) [24], the Illinois protocol [25] an economical solution [26], ownership protocols [27,28], the Firefly and Dragon protocols [10], a solution utilizing lock states for synchronization [29] and cache coherence support by the IEEE Futurebus [30]. A performance evaluation of several bus protocols is presented in [10]. The following sub-sections briefly describe a few broadcast protocols. A more detailed examination is found in the literature.

2.1.2.1. The Write-Once Protocol

In the write-once scheme, an initial write to a block updates this block in main memory. All subsequent writes are only written to the cached block. The block is updated in main memory when it is replaced or when a remote cache requests a copy. This scheme is an integration of the write-through and write-back write policies. All processor writes cause invalidations of the block if present in remote caches. Main memory supplies the copy of a block to a requesting cache unless it has been modified two or more times in a remote cache. In this case, the remote cache supplies the block and it is also concurrently updated in main memory.

This scheme does not take full advantage of the bus architecture. For example, if a cache requests a copy of a block that is present and unmodified in a remote cache, main memory supplies the data. A faster implementation would allow the remote cache to supply the data. This scheme is used in the Illinois protocol discussed below. Also, this protocol does not distinguish between a read-only block that is shared and one that is exclusive. This results in possible unnecessary invalidation signals broadcast on the bus. Furthermore, if a block is modified more than once, overhead is incurred as a result of the extra main memory update.

An extension of the write-once protocol is the CMU protocol outlined in [24]. In this scheme, a distinction is made between a block that has been invalidated in a cache and one that was replaced or never present. If a block has been invalidated its target address is still present in the cache directory. If a cached block is invalidated or shared, the corresponding processor's write request will generate a bus write, updat-

ing memory. All cache directories having the target address of this block also read the data from the bus. All bus reads also update invalidated cached data. This scheme dynamically defines a block as private if two or more write requests to the block occur without any remote requests to access the block^{*}. This protocol is optimized for efficient operation when one process modifies a data block to be read by several remote processes.

2.1.2.2. The Illinois Protocol

The Illinois protocol [25] is a low-overhead cache coherence solution using the write-back write policy. The state of each cached block is incorporated into each cache directory. No status information is associated with main memory. A cache miss results in a read broadcast to all caches and main memory. If the request was a write, an invalidation signal is also broadcast. If the block is located in a remote cache, that cache will supply the block. If several remote caches have copies of the block, a priority scheme chooses one cache to supply the data. In either case, memory is inhibited from supplying the data. If the remote block has modified the block resulting in a cache-main memory inconsistency, then main memory is concurrently updated. If a processor issues a write request all remote caches are invalidated.

2.1.2.3. The Ownership Protocol

In the Synapse's ownership protocol [27], each cached block has an associated owner. This owner is main memory if the block is shared or a cache if private. The

^{*} modifications may be made to define a block as private if x ($x \geq 2$) or more non-interleaved write requests are made to the block

owner always has the latest copy of the block. If a processor issues a read request to a remote block that is private, the remote cache sends a busy acknowledgement to the requestor, updates and passes ownership to main memory and invalidates its copy. The requestor then re-issues its request. This protocol does not fully utilize the bus broadcast capabilities. The Berkeley protocol [28] is also an ownership protocol. Like the Synapse protocol, the owner always has the latest copy of the block and only the owning block allows data modification. Unlike the Synapse protocol, when a processor requests to read a block owned by a remote cache, the remote cache supplies the data to the requestor. The data is not written back to main memory until the owned block is replaced. This insures a reliable system operation because although data is inconsistent with main memory, several copies of the modified block exist in the private caches.

2.1.3. Directory Protocols

Directory protocols maintain block states in a central or distributed directory as well as in local cache controllers. These protocols have been proposed for use primarily in multiprocessor systems with full-crossbar interconnection networks. Protocols utilizing global directories are discussed in [31, 32, 33, 34]. Although single bus protocols provide a more natural and faster method of maintaining coherence for sequential main memory accesses, distributed directory protocols provide concurrent main memory accesses resulting in reduced contention and possibly better overall system performance. The coherence protocol simulated is based on the directory method for the full crossbar network and a directory/broadcast protocol for the single

bus architecture.

2.2. Coherence Protocol Simulated

The cache coherence protocol simulated is based on the presence flag technique [32]. A cached block may be in one of four states as outlined below (a write-back write policy is assumed):

- (1) **Invalid (INV):** a block is not in the cache or it has been invalidated.
- (2) **Exclusive Read-only (EX):** a block is located in only one cache and it has not been modified.
- (3) **Read-only Shared (RO):** a block is located in 2 or more private caches and all copies are consistent with main memory.
- (4) **Exclusive Read-Write (RW):** a block is located in only one cache and it has been modified resulting in a cache-main memory inconsistency.

A distributed global directory consisting of $P+1$ bits for each main memory block is also used. The P presence bits corresponds to the P private caches in the system. If a block is located in a cache its corresponding presence bit is set. An additional modify bit is included to denote a cache-main memory inconsistency. An example of a distributed directory implementation is given in [9].

Since the global directory is distributed, is it assumed to be part of the memory controller of each main memory module. Commands from each cache controller (CC) to the memory module controller (MC) are used to implement the coherence protocol. These commands, outlined in Tables 2.1 and 2.2, are extensions to the commands given in [31].

When a processor attempts to modify data, it has to verify exclusive access to it. The operation used in this verification process is referred to as a *cross-interrogate*

COMMAND	DESCRIPTION
EXCLUSIVE READ	issued as a result of a write miss.
READ	issued as a result of a read miss.
REQUEST EXCLUSIVE	issued when the processor requests to modify a block presently in state RO.
REPLACE EX, REPLACE RO	issued as a result of a block replacement. This signals a global table modification.
REPLACE RW	issued as a result of block replacement. This signals global table modification and a main memory update.
MODIFY EXCLUSIVE	issued as a result of the local modification of a an EX block. This results in the MC setting the modified bit in the global directory.

Table 2.1 Commands from the Cache Controller to the Memory Controller.

COMMAND	DESCRIPTION
INVALIDATE RW	issued as a result of the REQUEST EXCLUSIVE command or a write request from an I/O controller.
INVALIDATE EX INVALIDATE RO	occurs as a result of the EXCLUSIVE READ or REQUEST EXCLUSIVE (INVALIDATE RO only) commands or a write request from an I/O controller. When the memory controller receives the REQUEST EXCLUSIVE command it searches the central directory to determine the caches owning a RO copy of the block. This signal is simultaneously sent to the CCs of these caches. A similar process occurs for the EXCLUSIVE READ and I/O controller commands. The MC then waits until an acknowledgement is received from these CCs before asserting the RO->RW signal.
RW->RO CHANGE	occurs as a result of the READ command from a CC and a set modified bit.
RW->EX CHANGE	occurs as a result of a read request from an I/O controller and a set modified bit.
RO->EX CHANGE	issued after the replacement of all but one RO copy of a block.
RO->RW CHANGE	used as an acknowledgement signal for the REQUEST EXCLUSIVE command. The signal is asserted after all INVALIDATE RO signals have been acknowledged from the CCs.
EX->RO CHANGE	occurs as a result of the READ signal, a single set presence bit and a reset modified bit.

Table 2.2 Commands from the Main Memory Controller to the Cache Controller.

(XI). A description of all XIs used in this protocol as well as the CC and MC signals used in their implementation is given in Table 2.3. The three basic XIs are XI change state (XICS), XI invalidate (XI-INV) and XI cast out (XICO). The XICS has the lowest performance penalty. This operation modifies the global and remote cache directories. XICS operations include RO->EX and EX->RO. The XI-INV (INV-RO and INV-EX) results in miss ratio degradation in addition to the directory modifications. The XICO (RW->RO, RW->EX and INV-RW) is a special invalidate and state-change operation that causes a main memory cycle penalty as a result of implementing the write-back write policy. This penalty is in addition to the global and remote cache directory modifications and a possible miss ratio degradation (INV-RW only). For bus implementations, a block is updated in memory concurrently with its placement in the cache initiating the RW->RO.

Two additional operations that result in performance overhead (but no remote cache directory modifications) are EX->RW and RO->RW. These operations set the modify bit in the global directory when a local cache possessing a valid copy of a block attempts to modify that block. The CC/MC signals defining these operations are MODIFY EXCLUSIVE/EX->RW CHANGE and REQUEST EXCLUSIVE/RO->RW CHANGE respectively. All global and remote directory modifications occur in read-modify-write cycles.

The state diagram illustrated in Figure 2.1 shows the coherence protocol implementation for block i in cache c_k ($1 \leq k \leq P$). The state of a block is changed as a result of one of four types of events; a local read, a remote read, a local write and a remote write access. A local read or write to block i occurs when the processor associated

XI	COMMANDS*	DESCRIPTION	PENALTY
XICS RO->EX EX->RO	replace RO RO->EX CHANGE read EX->RO CHANGE	changes the state of a remotely cached block	global and remote cache directory modification
XI-INV INV-RO INV-EX	exclusive read or request exclusive INVALIDATE RO exclusive read INVALIDATE RO	invalidates remote copies of a block	global and remote cache directory modification, miss ratio degradation of remote cache
XICO RW->RO RW->EX INV-RW	read RW->RO CHANGE read (I/O only) RW->EX CHANGE exclusive read INVALIDATE RW	special cases of XI-INV and XICS causing the greatest performance penalties	global and remote cache directory modification, a main memory update (CO) and the miss ratio degradation of a remote cache (INV-RO only)

*MC commands are capitalized

Table 2.3 Table of Cross-interrogates.

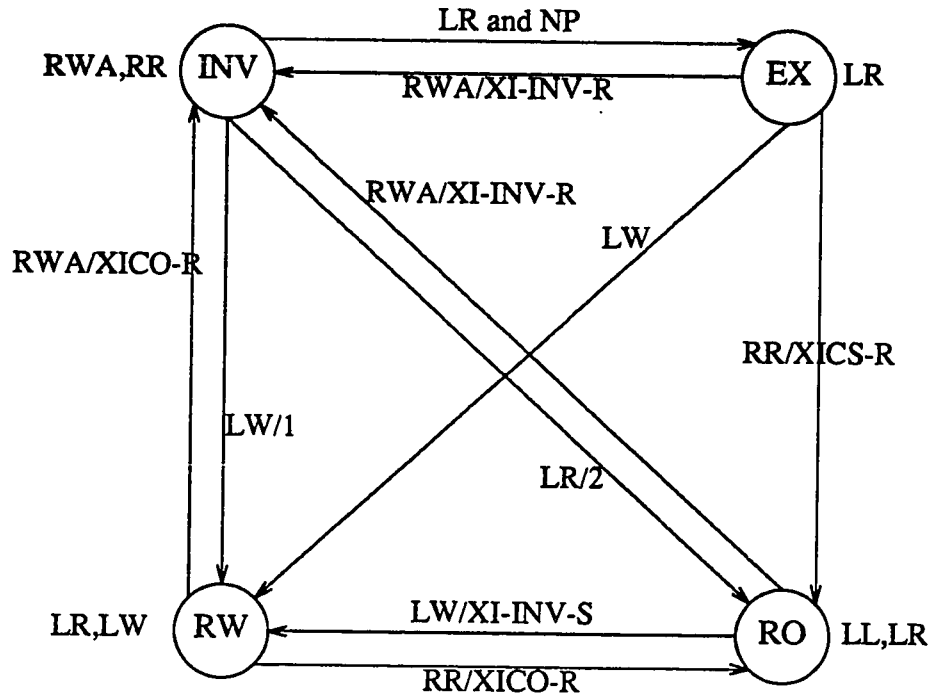
with the cache containing the block requests to read or write the data, respectively. A

remote read or remote write access occurs when any other processor requests to read or write the data in the block i . All remote events cause state changes as a result of *receiving* some type of XI (this excludes blocks in the invalid state). A local event may or may not cause the CC to *send* a XI. Some state change operations produce event/XI pairs as shown in Figure 2.1.

Three cache events signal the operation of the cache coherence protocol. They are a read miss, a write miss and a write hit. The procedures used to maintain coherence for each event are described below. All processor reads and writes are assumed to be globally performed as outlined in [35].

Read Miss. When this event occurs the global directory is consulted to check for possible copies of the block in a remote cache (full crossbar only). This directory is also updated to reflect the presence of the missed block. The local valid bit for the requested block is also set for the associated block frame in the cache directory. If the block is not present in any cache (all presence bits for the block are cleared in the global directory) then the local state of the block is set to EX and the block is transferred from main memory to the cache. For bus architectures, if a remote cache contains the block, it supplies the data to the requesting cache. If several remote caches have a copy of the block a priority scheme chooses one cache to supply the block.

If the global state of the block is EX then a XICS (EX- \rightarrow RO) occurs. If the global state is RO no remote cache directory action is needed. In both instances, the local state for the block is set to RO and the block is transferred from main memory (or a remote cache for single bus architectures) to the cache. If the global state of the block is RW the cache directory executes a XICO (RW- \rightarrow RO) operation. The block is then transferred from main memory to the cache and its local state is set to RO.



LR - local read
 RR - remote read
 LW - local write
 RWA - remote write access
 S (R) - signal sent (received)
 P - block is present in at least one remote cache
 NP - not P

1 - if P then XI-INV-S
 2 - if P then XICO-S (RW) or XICS-S (EX)

Figure 2.1 State Diagram of Coherence Protocol.

Write Miss. A write miss causes a global table lookup and a possible update. If the global state of the block is INV it is set to RW. If the global state is RO, one or more XI-INV-S (INV-RO) are executed. A XICO (INV-RW) is executed if the global state is RW. In all instances, the block is transferred from a remote cache (bus

architectures only) or main memory to the cache, the local state is set to RW, and the processor modifies the data in the cache.

Write Hit. If the global block state is EX it is set to RW. If the global state is RO at least one XI-INV (INV-RO) is executed. If the global state is RW no global action is necessary. If the local state is EX or RO it is changed to RW and the processor modifies the cached data.

Although not shown in Figure 1.1, the multiprocessor system used in this study includes a control signal bus (CSB) for the full-crossbar interconnection configuration. This CSB is used by each MC to simultaneously send invalidation signals to all necessary remote cache directories. All invalidation requests are buffered by the CCs to release the CSB as soon as possible. The MC also requests CCs to change its local state with the CSB. Furthermore, the CSB is used by the CCs to send acknowledgement signals to the MC.

Bitar and Despain [29] discussed four methods of implementing atomic read-modify-write instructions. Three of these methods require the cache to know at the beginning of the instruction that it is atomic. The fourth method uses two additional cache block states, the lock state and the lock-waiter state, to implement this atomic operation. The protocol simulated in this study used the second method outlined in Bitar and Despain. That is, the atom is located in one block and the cache requests write access to the block at the beginning of the instruction. Special atomic instructions are assumed to facilitate this implementation.

2.3. Performance Parameters and Penalties

There are four main sources of performance degradation when using the cache coherence protocol discussed previously. These sources are as follows [34]:

1. Degradation of the average miss ratio due to block invalidations
2. Increased traffic between caches to enforce consistency
3. Concurrent access to the global tables resulting in conflicts
4. Write-backs due to invalidation or state change of RW data

These sources were used to derive the performance parameters used to evaluate the coherence mechanism for the PDE algorithms and the system under study. The parameters used were the miss ratio degradation (MRD), the probability of a XICO (prXICO), the probability of a XICS (prXICS), and the invalidation ratio (IR). These parameters were determined for each processor in the multiprocessor system.

The miss ratio is obtained for both the enabled (EMR) and disabled (DMR) coherence protocol. The MRD is then calculated as shown below:

$$MRD = EMR / DMR \quad (2.1)$$

The prXICO is the fraction of processor references (shared and private) that cause a XICO. It measures the amount of overhead that causes the largest coherence maintenance penalty. The prXICO is calculated as shown in Equation (2.2).

$$prXICO = \frac{\text{number of INV-RWs} + \text{number of RW} \rightarrow \text{ROs}}{\text{total number of references}} \quad (2.2)$$

The prXICS is the fraction of processor references resulting in RO-→EX or EX-→RO operations. This measures the minimal overhead due to coherence maintenance. The prXICS is calculated as shown below:

$$prXICS = \frac{\text{number of } RO \rightarrow EXs + \text{number of } EX \rightarrow ROs}{\text{total number of references}} \quad (2.3)$$

The IR is the fraction of processor references that cause invalidations of RO and EX blocks. It measures the overhead that causes the miss ratio degradation of the private caches (INV-RW also causes miss ratio degradation but it is defined as a XICO due to the larger write-back penalty). The IR is calculated as shown below:

$$IR = \frac{\text{number of } INV-ROs + \text{number of } INV-EXs}{\text{total number of references}} \quad (2.4)$$

Let t_C be the cache cycle time, t_B the block transfer time, t_{GD} the global directory access time, t_{RD} the remote directory update time and w the probability of a write-back, all in terms of the number of processor cycles. The penalty due to a cache miss, t_{MS} , is therefore

$$t_{MS} = t_B + (w - prXICO)t_B + t_{GD} \quad (2.5)$$

The penalty due to a XICO, t_{XICO} , is

$$t_{XICO} = t_{GD} + t_{RD} + t_B \quad (2.6)$$

An upper bound on the penalties for a cache miss and for XICO operations are assumed in this study. In practice, the penalties for these parameters are implementation dependent. For example, in some systems, block transfers and global directory accesses occur concurrently. The global directory access times may therefore be eliminated from Equations 2.5 and 2.6 for these systems. This is because the time required for block transfers is longer.

The penalty due to a state-change operation, t_{XICS} , is

$$t_{XICS} = t_{GD} + t_{RD} \quad (2.7)$$

and the penalty due to a XI-INV, t_{INV} , is*

$$t_{INV} = t_{GD} + t_{RD} \quad (2.8)$$

The effective memory access time with enabled cache coherence for a P processor system ($P \geq 2$), t_{ME_P} , is therefore

$$t_{ME_P} = t_C + MRt_{MS} + prXICOt_{XICO} + prXICS t_{XICS} + IRt_{INV} \quad (2.9)$$

2.4. Execution Time Model

The execution time model used is a modification of the models used in [13] and [16]. The modification includes changes to the effective memory access time to include cache coherence maintenance. The time needed to complete the evaluation of one iteration of the algorithm in a P processor system ($P \geq 2$) is

$$t_{IT}^P = t_{CALC} + N_A [t_{ME_P} + t_{WT}] \quad (2.10)$$

where t_{CALC} is the processor calculation time for updating sub-grid points, N_A is the total number of memory accesses required per iteration per processor, t_{ME_P} is the effective memory access time (see Equation 2.9) and t_{WT} is the waiting time for the interconnection network. The processor calculation time is

$$t_{CALC} = N_{fp} \frac{M^2}{P} T_{fp} \quad (2.11)$$

where N_{fp} is the number of floating point operations required in evaluating each grid point, M^2 is the total number of grid points, P is the number of processors and T_{fp} is the time to execute a single floating point operation. Vrsalovic, *et. al.* [13] formalized

*invalidations also cause MRDs. This penalty is exemplified by the MR obtained from the simulations and by the MRD graphs presented later.

the memory access waiting time for synchronous systems to be

$$t_{WT} = \left[\frac{P}{S} - 1 \right] t_{ME_P} \quad (2.12)$$

where S is the number of processors that can simultaneously access shared memory. For a shared bus architecture, the number of simultaneous accesses is one. The full crossbar interconnection network is functionally an M -bus multiple bus system. To illustrate both upper and lower bounds on performance it is assumed that all modules service requests at all times (upper bound) or only one module services requests at all times (lower bound). In the former case the value of S is P . In the latter case the full crossbar is reduced to performing as a single bus architecture.

The speedup for a P -processor system is

$$S_P = \frac{t_{IT}^1}{t_{IT}^P} \quad (2.13)$$

where the uniprocessor iteration time is

$$t_{IT}^1 = M^2 \left[N_{fp} T_{fp} + t_{ME_1} \right] \quad (2.14)$$

The uniprocessor effective memory access time is

$$t_{ME_1} = t_C + UMR \left[t_B + w t_B \right] \quad (2.15)$$

Here UMR is the uniprocessor miss ratio.

If the prefetch-on-miss or tagged prefetching [36] fetch algorithms are used, the effective memory access time for both the uniprocessor and multiprocessor is extended to include the prefetching penalties incurred. Let the *prefetch ratio* (PR) be the ratio of the total number of prefetched blocks to the total number of processor

references. There are both external and internal cache references when a fetch algorithm other than demand fetching is used. External references are those initiated by the CPU and I/O processors and internal references are those initiated by the cache to determine if a block to be prefetched is present in the cache. These internal cache references are known as *prefetch lookups*. The ratio of the number of prefetch lookup cache accesses to the total number of references is known as the *lookup ratio* (LR).

Let t_{PF} and t_{LR} be the penalties that occur as a result of prefetching a block and of prefetch lookups, respectively. The effective memory access time is then extended as follows:

$$t_{ME_c^f} = t_{ME_c} + t_{PF}PR + t_{LR}LR \quad (2.16)$$

where c ($c \geq 1$) is the total number of processors in the system. It is assumed that the penalty for a prefetch is the same as that for a miss (see Equations 2.5 and 2.15). Additionally, the total number of prefetched blocks is included when calculating N_a for each algorithm. Table 2.4 presents a summary of the parameters used in the execution time model and their value ranges.

Maximum values are shown for the probability of a write-back in this table. The larger grid sizes exhibit these larger values, a result of block replacements and XICO operations. However, only XICO operations result in write-backs performed for smaller grid sizes. This is because all blocks composing these smaller grid sizes map into unique cache block frames, eliminating all block replacements. Therefore, the write-back probability ranges from the prXICO for smaller grid sizes to the maximum values shown in Table 2.4 for larger grid sizes.

PARAMETER	DESCRIPTION	VALUE/RANGE (in processor cycles)
t_C	cache cycle time	1
t_B	block transfer time	$1 + BS/8^*$
w	probability of a write-back	0.20 JAC 0.17 SOR 0.30 PCG
t_{GD}	global directory access time	2
t_{RD}	remote cache directory modification time	2
T_{fp}	time to execute a single floating point operation	4
N_{fp}	number of floating point operations required in evaluating each grid point	4 JAC 6 SOR varies PCG
N_A	total number of memory accesses required per iteration per processor	algorithm dependent
t_{LR}	prefetch lookup penalty	1

* BS - blocksize

Table 2.4 Summary of Parameters and Ranges

To isolate the effect of cache coherence on the performance of the algorithms studied, the iteration time degradation (ITD) is used. This parameter is the ratio of the time needed to execute one iteration of the parallel algorithm with cache coherence enabled to the time needed to execute one iteration of the parallel algorithm with coherence disabled as shown below:

$$ITD = \frac{t_{CALC} + N_a \left[t_{ME_p}^{EC} + t_{WT}^{EC} \right]}{t_{CALC} + N_a \left[t_{ME_p}^{DC} + t_{WT}^{DC} \right]} \quad (2.17)$$

where $t_{ME_p}^{EC}$ and t_{WT}^{EC} are given in Equations 2.9 and 2.12, respectively. The disabled coherence effective memory access time is given below

$$t_{ME_p}^{DC} = t_C + DMR \left[t_B + w t_B \right] \quad (2.18)$$

where DMR is the disabled coherence miss ratio. Finally, the interconnection network waiting time for disabled coherence is

$$t_{WT}^{DC} = \left[\frac{P}{S} - 1 \right] t_{ME_p}^{DC} \quad (2.19)$$

2.5. Simulation Methodology and Model

Trace driven simulations have been used extensively in the design of memory hierarchies for uniprocessors. This is because the performance of memories are highly dependent on the dynamic reference string of the processors. Consequently, this procedure should also be applied to the design of memory hierarchies in multiprocessors; however traces of multitasked MIMD systems are difficult to obtain. The communication and synchronization that occur between processes are not present in SISD systems. Furthermore, reference strings may vary from one run to the next.

In spite of these difficulties some major simulators have been implemented [37, 38, 39]; however, these simulators are complex with some tracing every instruction.

A different approach was used in this study and is similar to the simulation methodology described in [40]. Every instruction of the algorithm is symbolically executed and all memory accesses are traced. This is in direct contrast to the methodology described in [40] where only global events, i.e., references to shared data, synchronization primitives, etc., are traced. While global events do cause the XIs which lead to degradation in system performance, many global and private events result in the contention of their corresponding shared and private blocks (respectively) for the same private cache block frame. This directly affects the number of XIs executed as well as the overall number of cache misses. Since the computations for the multitasked PDE algorithms studied are homogeneous, the P individual traces from each process were interleaved by a modification of *rank interleaving* [40]. All processes for the PDE algorithms execute the same code on different data. This data is partitioned statically before execution of the processes.

It is assumed that the processor instructions are non-self-modifying. This facilitated the use of separate data and instruction caches. The multiprocessor trace is obtained from *data* references. A PDE sub-grid is statically allocated to each process. To isolate the effect of cache coherency on the *algorithms* simulated, it is assumed that processes are not allowed to migrate, requests from I/O processors are not included in the multiprocessor trace and interprocessor interrupts are disabled.

Since the processes are assumed to be executed on P homogeneous processors, for each event k in processor j , there $P-1$ concurrent events in the remaining $P-1$ processors. In the interleaved trace event k in any processor occupies position $kP + r$ where r is a random number uniformly distributed between 0 and $P-1$. The interleaved trace is then used to drive a cache coherence simulation. This simulation calculates the various performance parameters outlined in the previous section.

To evaluate the effect of various cache features on algorithm performance when enforcing coherency, a steady-state model of algorithm execution (warm start) is used. The fetching algorithms simulated are demand fetching, prefetch-on-miss and tagged prefetching [36]. Another prefetching algorithm, always prefetch, requires main memory to fetch on every reference for all processors. This prefetching algorithm is prohibitive for the underlying architecture assumed in this study and is not considered. The cache blocksize was varied from 2 to 32 32-bit words (8 to 128 bytes) and the write-back (or copy-back) write policy is used.

The penalty for cache misses in multiprocessor systems may be substantial. To effectively reduce this miss ratio, many high performance multiprocessor systems have large caches. The cache size range used in this empirical study is 32-64Kbytes. The direct mapping and two-way set associative placement policies are simulated. Memory constraints imposed upon the system used to execute the cache simulator prohibited the simulation of a fully associated or an n -way ($n > 2$) set associative mapping strategy. A least-recently-used (LRU) replacement policy is used for the set associative mapping strategy for all fetching strategies.

To decrease the effective memory access time virtual address caches were used. A primary concern in designing virtual address multi-cache systems is how to handle multiple virtual addresses that map to the same physical address. This is known as the *synonym* problem. This problem is eliminated by allowing all processes executing the algorithm to share the same virtual address space. For example, processes executing on the SPUR [5] system may share *segments* of virtual address space.

CHAPTER 3

JACOBI'S ITERATIVE ALGORITHM

This chapter presents a discussion of the numerical solution of Jacobi's synchronized iterative algorithm. This algorithm is generally considered a prototype parallel algorithm [2]. Also included is a description of the implementation and decomposition strategies simulated. The performance parameters obtained from the simulation are presented followed by the performance evaluation of this algorithm and finally, Jacobi's conclusions.

3.1. The Classical Algorithm

Discretizing LaPlace's equation by central differences on a rectangular region with Dirichlet boundary conditions results in Jacobi's synchronous iterative algorithm as shown in the equation below for the (k+1)st iterate. A detailed discussion and derivation of this equation is presented in [41].

$$u_{i,j}^{k+1} = \frac{1}{4} \left[u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k \right] \quad (3.1)$$

This computation consists of repetitively computing the average of the north, south, east and west neighbors of each grid point. The simulated parallel implementation of this algorithm is shown in Figure 3.1 for two iterations. Two copies of the grid are used in this computation, U and V. For each iteration, a copy of a grid is updated using the grid values from the other copy. The grid copies are decomposed into P contiguous sub-grids, each allocated to a single processor. The processors update

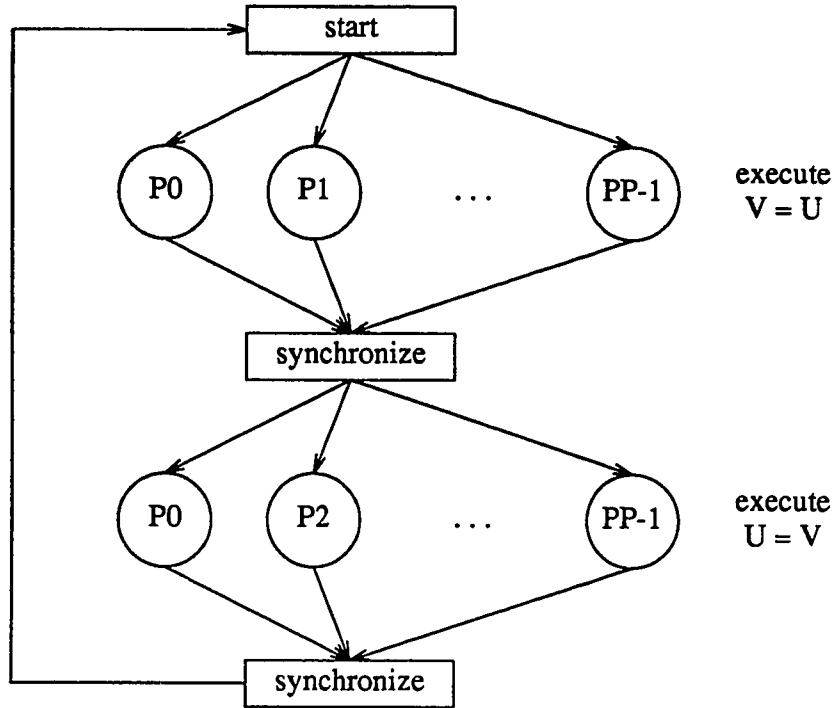


Figure 3.1. Parallel Implementation of Jacobi's Synchronous PDE Algorithm.

each sub-grid and then synchronize using any number of synchronization primitives such as barriers [42]. After synchronization, the processor continues the computation with the grid copies reversed. This completes two iterations.

The PDE grid is decomposed into squares and rectangular strips. For example, consider a rectangular grid consisting of m^2 points. This grid is decomposed into $\frac{P}{l}$

rectangular strips with l sub-grids per rectangular strip. This is shown in Figure 3.2 for $P=4$ and l ranging from 1 to 2 (a and b respectively). Table 3.1 lists the partitionings used for the various number of processors simulated. Each sub-grid consisted of $\frac{ml}{p} \times \frac{m}{l}$ grid points.

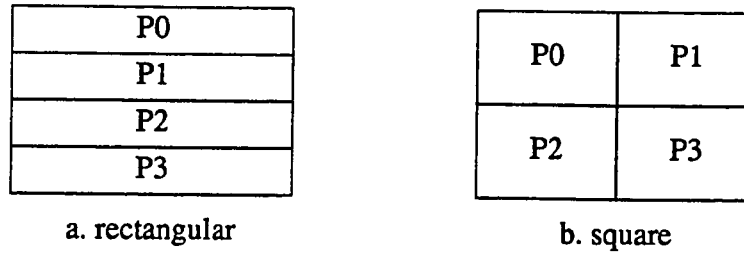


Figure 3.2 Decomposition Strategies for Jacobi's Algorithm Using 4 Processors.

P	l , Square	l , Rectangular
2	1	1
4	2	1
8	2	1
16	4	1

Table 3.1 Decomposition Parameter Values

3.2. Simulation Results

All data presented here represent the worst case parameters for all processors used in the multiprocessor system. Unless otherwise noted, all graphs present the simulation results using the demand fetching policy. Furthermore, some graphs presented in this thesis use the following acronyms: D or DM for direct mapping, S or SA for set-associative mapping, pf for prefetching, pom for prefetch-on-miss, tpf for tagged prefetching, dmf or df for demand fetching and 32K or 64K for 32Kbyte or 64Kbyte cache sizes, respectively.

The grid size is varied from 64x64 to 1024x1024 points for all simulations. The 64x64 grid is so small that blocks in both the U and V grids map into unique cache block frames. For larger grid sizes several memory blocks map into the same cache block frame. This eventuates as a result of block frame contention among blocks in the two grids and/or among blocks within the same grid.

A graphic representation of this contention is shown in Figure 3.3. This illustration assumes a 32Kbyte cache size, a 8-byte blocksize, a 256x256 point grid decomposed into sub-squares and a four processor system. When using the direct mapping strategy, the blocks composing each sub-square may be divided into four parts as shown for processor P0. The geometrically corresponding blocks of all four parts map into the same cache block frame (see blocks a, b, c, and d in figure). This is the source of *intragrid contention*. Similarly, contention may occur as a result of using two grids. For example blocks u and v of grids U and V, respectively (see Figure 3.3), map to the same block frame resulting in possible *intergrid contention*.

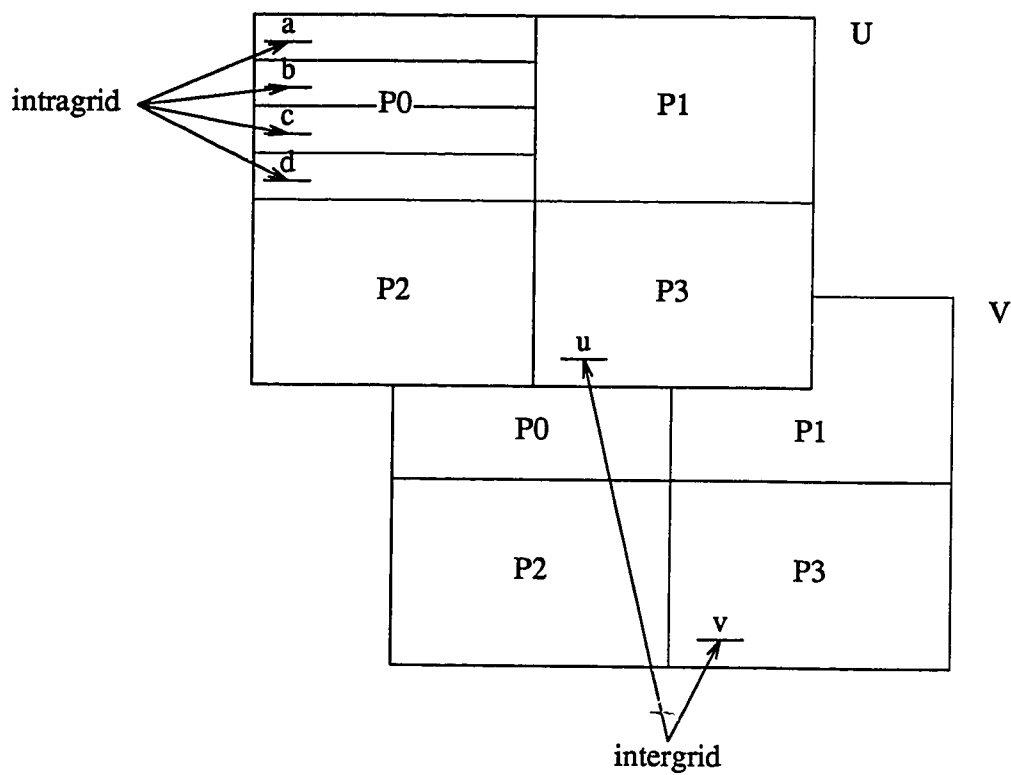


Figure 3.3 Intragrid and Intergrid Contention.

3.2.1 Miss Ratio/Miss Ratio Degradation

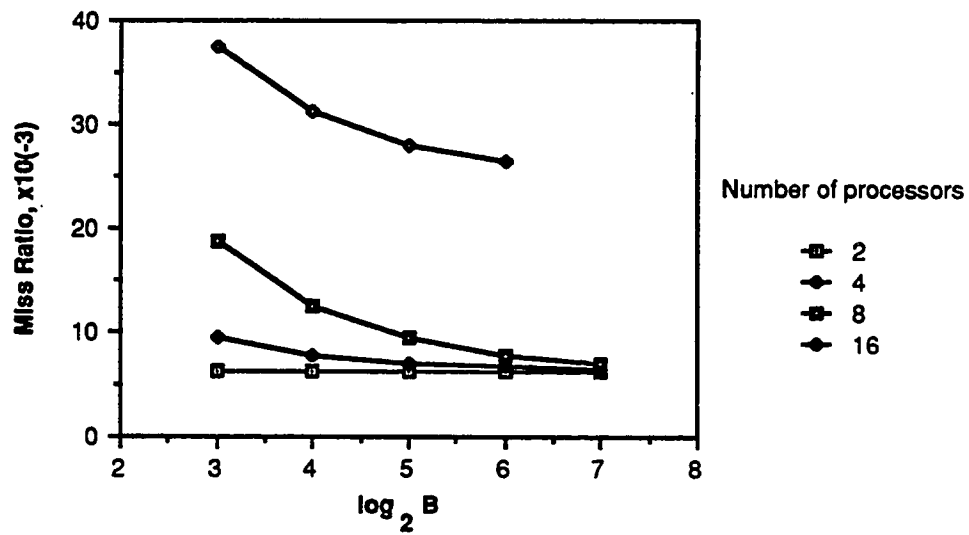
The MRD is infinite for the 64x64 point grid, independent of cache design alternatives, the number of processors considered and the decomposition strategy. Since all memory blocks referenced by each processor for this grid size map into unique cache block frames, all misses are a direct result of block invalidations. An infinite

MRD is therefore the result of the *principle of unique block frames* as formalized below.

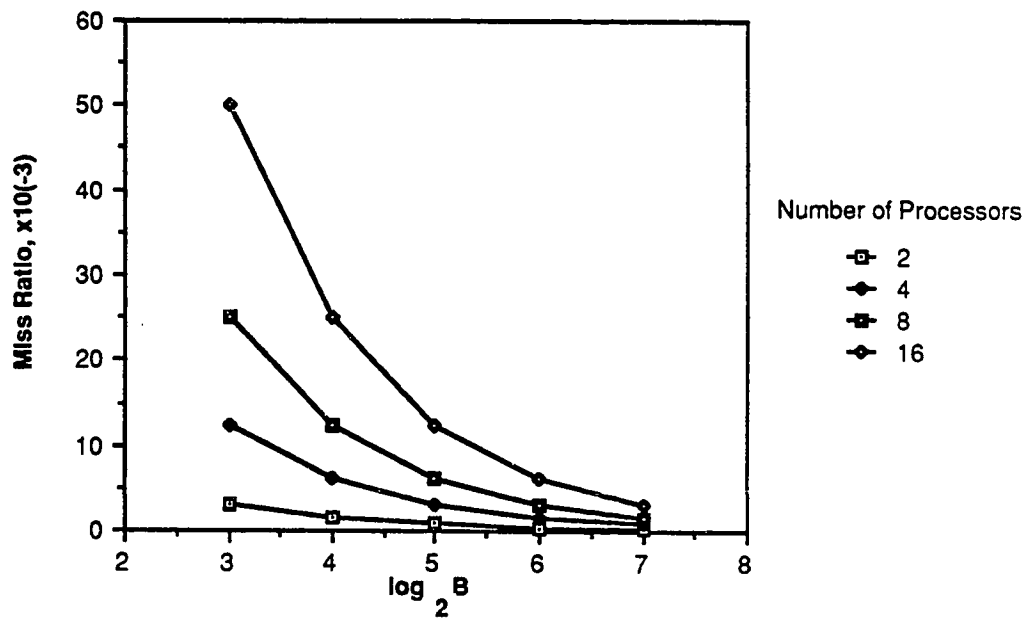
Principle of Unique Block Frames. If all memory blocks referenced by a processor executing a parallel algorithm with enabled cache coherence map into unique cache block frames, then the MRD for the algorithm is infinity. This value is independent of the cache design alternatives, number of processors, and domain decomposition strategies. It results from misses occurring only as a result of block invalidations received by a processor.

The MR versus the cache blocksize for the 64x64 point grid is shown in Figure 3.4 for both square and rectangular decompositions. In all instances this MR is 5% or less and its behavior is intuitive. For example, as the blocksize increases the MR decreases. Also, for a given blocksize, as the number of processors increase, the MR increases. All blocks composing this grid size map into unique cache block frames. Therefore, all misses are a direct result of block invalidations which in turn depend upon the number of shared modifiable blocks present in the cache. As the number of processors increases, the sub-grid size per processor decreases; however, the number of shared modifiable blocks eventually placed in the cache decreases at a slower rate for this algorithm. Therefore, the number of misses per processor also decreases at a slower rate. This explains the increase in the MR as the number of processors increases. Also note that for a 8-byte block the rectangular decomposition MR is larger than the square decomposition MR for all processor configurations except the two processor system.

As the cache blocksize is doubled the rectangular decomposition MR is reduced by one-half. This is not the case for the square decomposition strategy. In this situa-



a. Square Decomposition



b. Rectangular Decomposition

Figure 3.4 Miss Ratio versus Blocksize, 64x64 Grid

tion, the remote shared blocks (referenced by each processor) that are adjacent to the *vertical* sub-grid border produce a constant number of misses each iteration, independent of the cache blocksize. (This phenomenon is defined as the *principle of vertical shared blocks*). For this same reason, the MR for the 2 processor square decomposition strategy is constant. The MR for the 16 processor square decomposition strategy and 128 byte block is 0.18, an order of magnitude larger than all other miss ratios for this grid size. This is caused by the modifying of one shared block by two processors.

For grid sizes larger than 64x64 points, the major parameters effecting the MRD are the cache mapping function, the cache size, and the decomposition strategy. For the direct mapping strategy, there are no MRDs for these larger grids. This is true for both square and rectangular decompositions and all processor configurations. The justification for this MRD absence is intergrid contention. Although block invalidations are performed, other blocks mapping to the same cache block frame as the invalidated blocks are accessed *before* the invalidated blocks are referenced again by the processor. This results in the same MR for both enabled and disabled coherence.

The detrimental effect of intergrid contention is illustrated in Figure 3.5 for the MR versus cache blocksize with direct mapping placement. The MR varies from slightly over 0.5 to .41 as the blocksize varies from 8 to 128 bytes. This MR is independent of cache size, decomposition strategy and the number of processors. It is also relatively independent of the grid size. Therefore, as a result of high MRs, the two-grid implementation of this algorithm should be avoided.

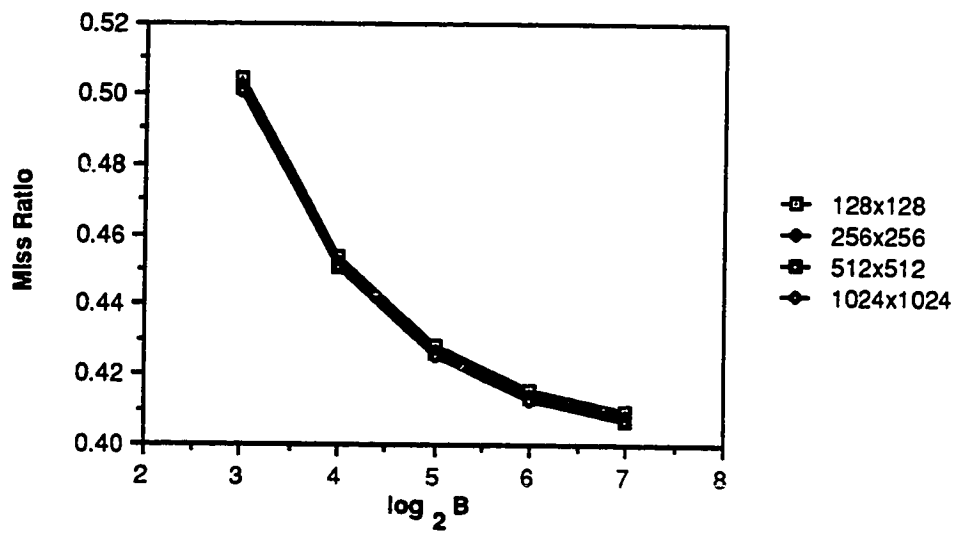


Figure 3.5 Miss Ratio versus Cache Blocksize, Direct Mapping

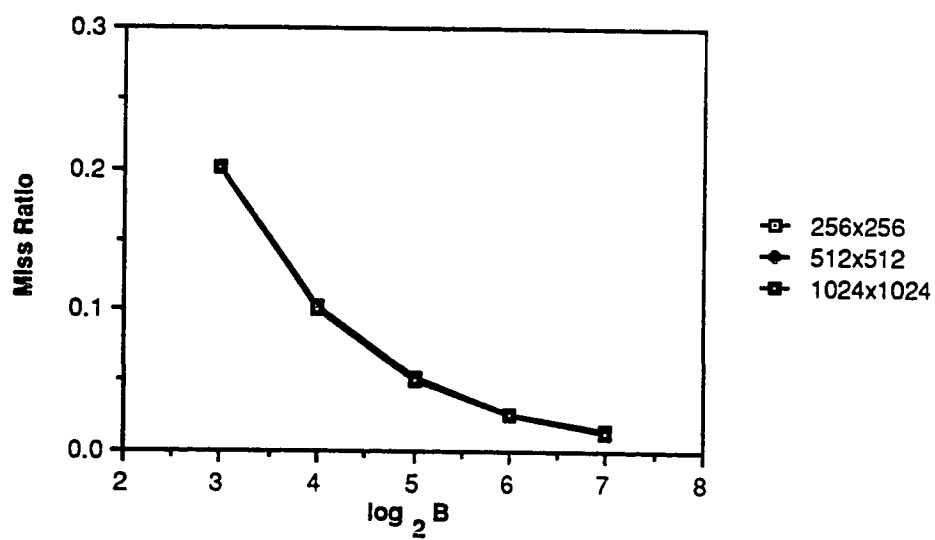
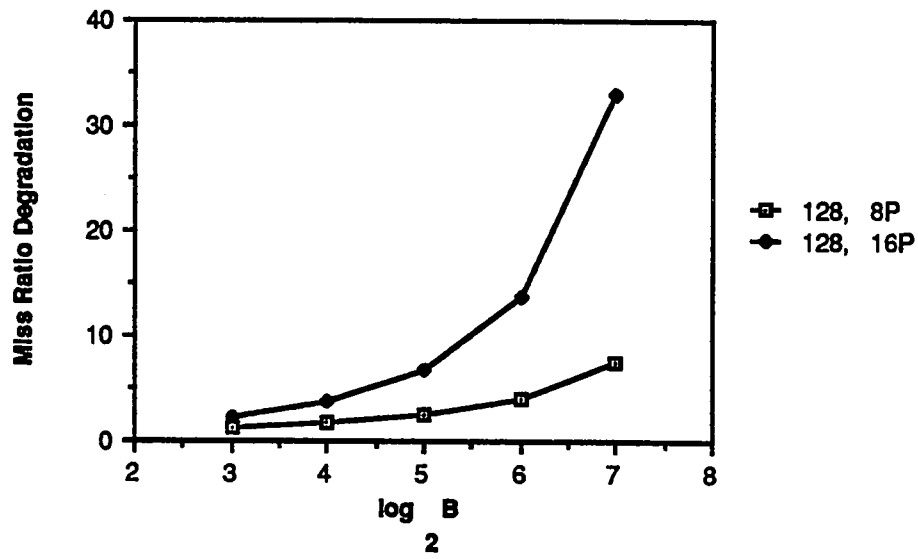


Figure 3.6 Miss Ratio versus Cache Blocksize, Set-Associative Mapping

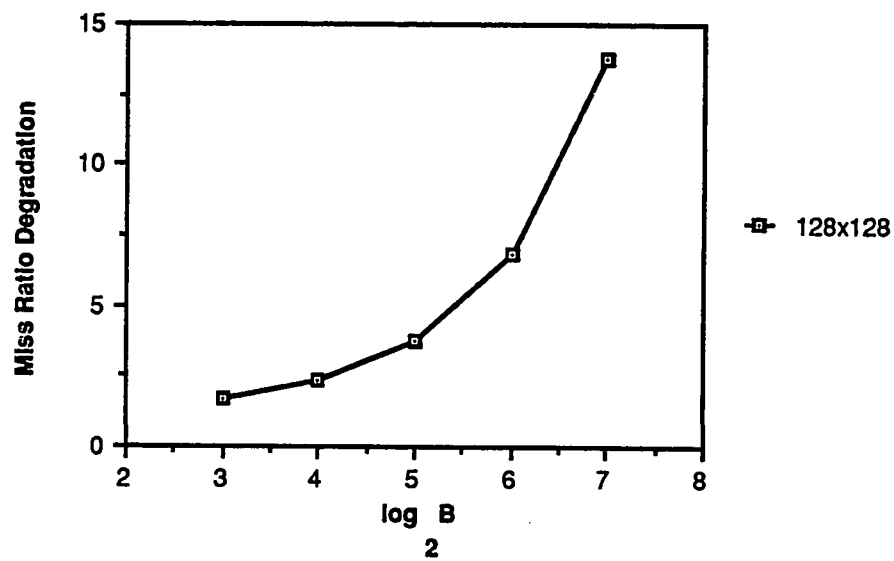
For the 2-way set-associative mapping strategy and grid sizes larger than 128x128 points the MRD is 1.0. This is true for all cache design parameters, both decomposition strategies and all processor configurations with one exception; the 256x256 point grid for the rectangular decomposition strategy and 64Kbyte cache size. Intragrid contention is the dominate factor resulting in the absence of the MRD for these large grid sizes. In the exceptional case, the MRD is infinity as a result of the principle of unique block frames. The MR versus cache blocksize for 2-way set-associative placement is shown in Figure 3.6. All grid sizes larger than 128x128 points are shown as well as some 128x128 point grid sizes. This MR is relatively high for block sizes smaller than 32 bytes; however, it indicates a better performance than direct mapping placement. This is largely the result of a reduction in intergrid contention.

While intergrid and intragrid contention is still present for the 2-way set-associative mapping strategy, the frequency of this contention is reduced. This is caused by the fact that *two* blocks map into the same set. Therefore it is possible for two geometrically corresponding blocks in the U and V grids or two blocks within the same grid to be present in the cache simultaneously. This is not feasible for the direct mapping strategy.

The reduction in the frequency of intergrid and intragrid contention results in the presence of a MRD for some processor configurations when using the 128x128 point grid. Figure 3.7 illustrates this MRD for the square decomposition strategy, 8 and 16 processors (32Kbyte cache), and 4 processors (64Kbyte cache). This figure shows that as the blocksize increases the MRD increases. While the number of



a. 32Kbyte Cache Size, 8 and 16 Processors



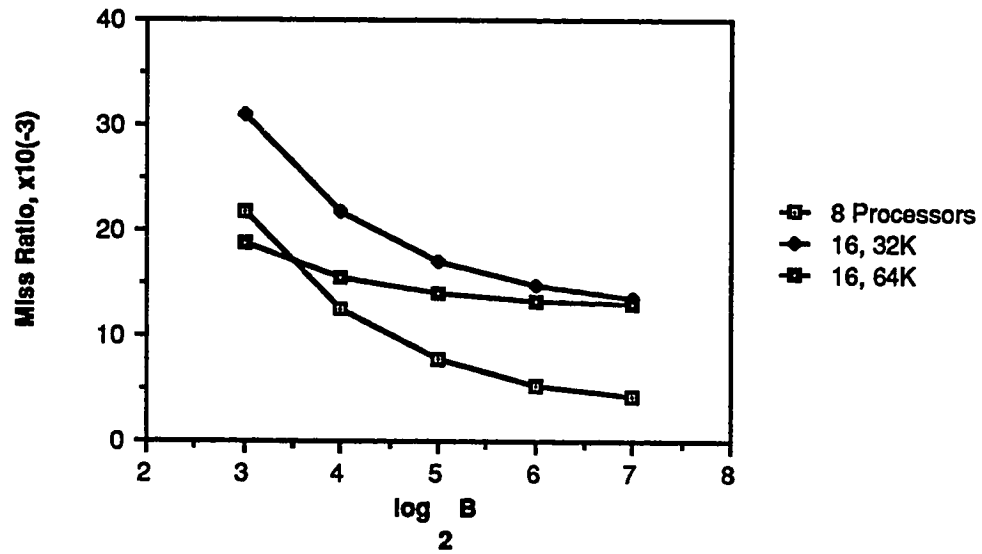
b. 64Kbyte Cache Size, 4 Processors

Figure 3.7 Miss Ratio Degradation versus Cache Blocksize, 128x128 Point Grid Size, Set-Associative Mapping, Square Decomposition.

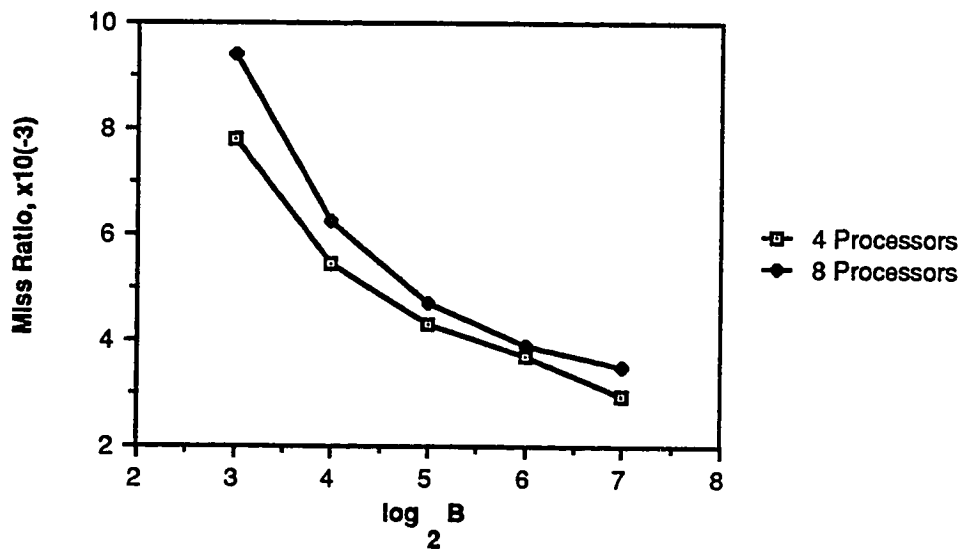
invalidated shared blocks along the horizontal sub-grid borders decrease as the blocksize increases, the number of invalidated shared blocks along the vertical sub-grid borders are independent of the blocksize. This is one cause of the increase of the enabled MR over the disabled MR. The number of misses occurring as a result of intergrid contention decreases as the blocksize increases resulting in a decrease in the MR for disabled coherence. This in turn results in an increase in the MRD as the blocksize increases. There is no MRD for a two processor system on a 128x128 grid (both decomposition strategies and cache sizes), a result of intergrid and intragrid contentions. This also holds for a four processor system when using both decomposition strategies (32Kbyte cache only). In all instances not previously discussed, the MRD for the 128x128 point grid is infinite as a result of the principle of unique cache blocks.

Figures 3.8 and 3.9 show the MR versus the cache blocksize for square and rectangular decomposition strategies, respectively. These MRs generally decrease as the blocksize increases. The rectangular decomposition strategy has a faster rate of decrease as a result of the absence of vertical shared blocks along its sub-grids. Furthermore, the rectangular decomposition strategy generally has a lower MR than the square decomposition strategy. Also, for both decomposition strategies, the MR increases as the number of processors increase for a given blocksize.

A closer look at the MRs presented show a optimum cache blocksize of 32 bytes. For example Figure 3.6 shows a MR of 0.05 for this blocksize (set-associative mapping). While smaller block sizes may reduce the block transfer time, a critical parameter in multiprocessor systems, they result in prohibitive MRs. Larger

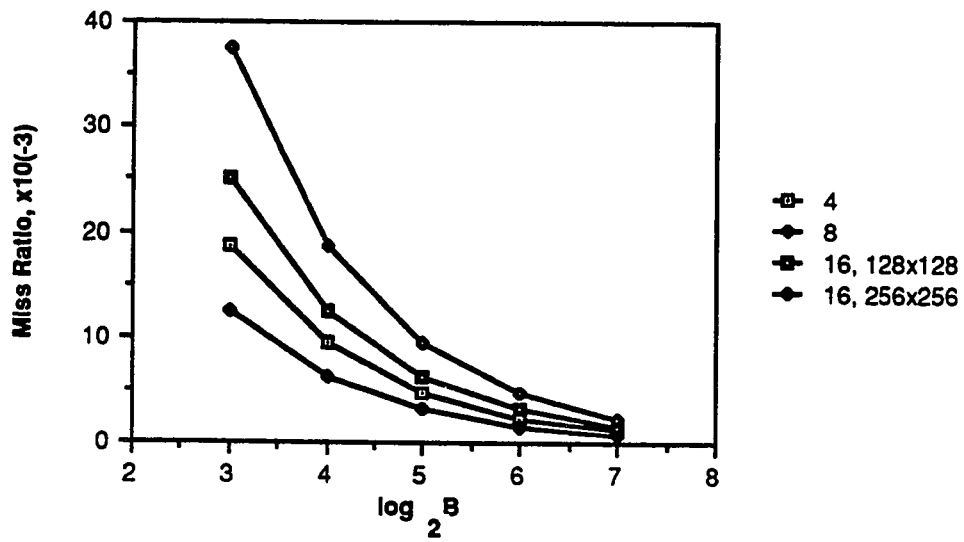


a. 32Kbyte Cache Size

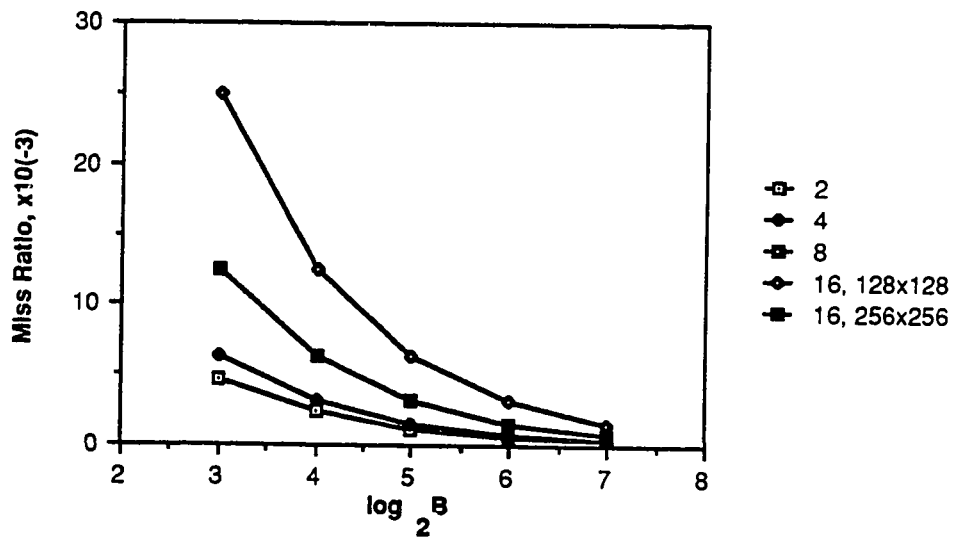


b. 64Kbyte Cache Size

Figure 3.8 Miss Ratio versus Cache Blocksize, Set-Associative Mapping, 128x128 Point Grid, Square Decomposition.



a. 32Kbyte Cache Size



b. 64Kbyte Cache Size

Figure 3.9 Miss Ratio versus Cache Blocksize, Set-Associative Mapping, 128x128 Point Grid, Rectangular Decomposition.

blocksizes produce smaller MRs but may result unacceptable block transfer times. Table 3.2 lists the MRD for 2-way set-associative mapping, a 32-byte blocksize and all other design alternatives and decomposition strategies. Unless otherwise noted, all graphs shown in this thesis assume a cache blocksize of 32 bytes.

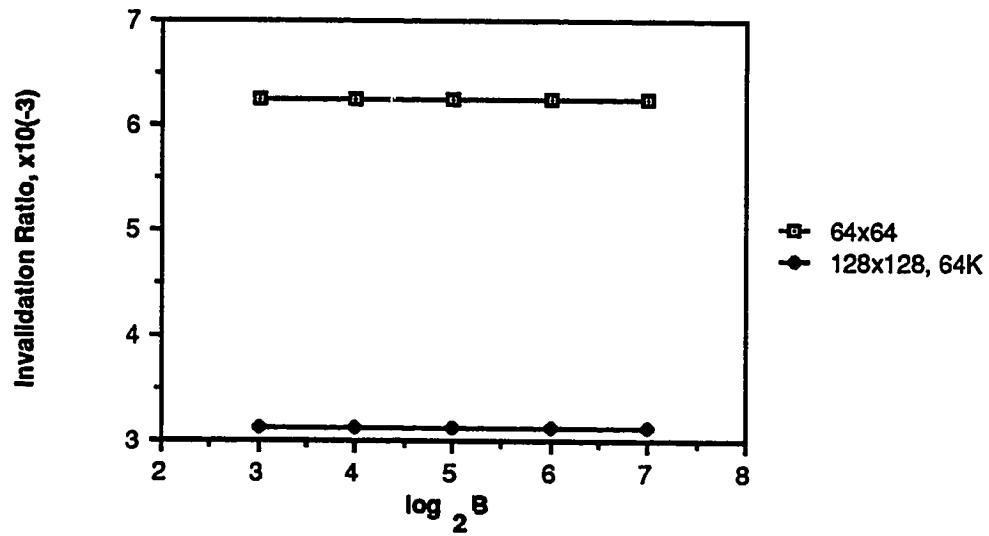
Cache Size	Partition	CPUs	PDE Gridsize				
			64	128	256	512	1024
32Kbytes	square	2	∞	1.00	1.00	1.00	1.00
		4	∞	1.00	1.00	1.00	1.00
		8	∞	1.68	1.00	1.00	1.00
		16	∞	3.78	1.00	1.00	1.00
	rectangular	2	∞	1.00	1.00	1.00	1.00
		4	∞	1.00	1.00	1.00	1.00
		8	∞	∞	1.00	1.00	1.00
		16	∞	∞	1.00	1.00	1.00
64Kbytes	square	2	∞	1.00	1.00	1.00	1.00
		4	∞	2.36	1.00	1.00	1.00
		8	∞	∞	1.00	1.00	1.00
		16	∞	∞	1.00	1.00	1.00
	rectangular	2	∞	1.00	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	1.00	1.00	1.00
		16	∞	∞	∞	1.00	1.00

Table 3.2 Miss Ratio Degradation, Set-Associative Cache with 32-byte Block.

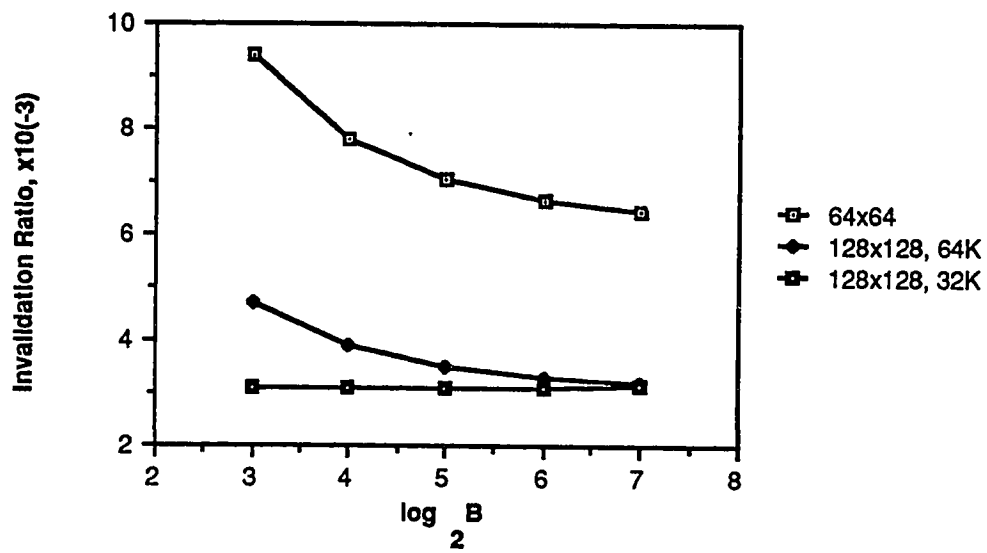
3.2.2. Invalidation Ratio

Figure 3.10 shows the IR versus the cache blocksize for 2 and 4 processors (a and b respectively), direct mapping, both cachesizes and the 64x64 and 128x128 point grid sizes. For grid sizes larger than 128x128 points, intragrid contention dominates direct mapping in such a way to eliminate all block invalidations for 2 and 4 processor systems. This also holds for the 128x128 point grid with the 32Kbyte cache size and the 2 processor system. The IRs for the 2 processor system are independent of the cache blocksize. This is a result of the principle of vertical shared blocks. With values of 6.25^{-3} and 3.125^{-3} for the 64x64 (32 and 64Kbytes) and 128x128 (64Kbytes only) point grids, respectively, the IR is relatively small for this system.

With the exception of the 128x128 point grid size for the 32-Kbyte cache the IR decreases as the cache blocksize increases for the 4 processor system (see Figure 3.10b). As the blocksize increases, fewer blocks can be placed in the cache simultaneously, therefore facilitating a smaller number of invalidations. Observe that the IR is not effected by the cache size for the 64x64 grid but it increases as the cache size is doubled for the 128x128 point grid (128-byte blocksize excluded). This results from the simple fact that the 64x64 point grid maps into unique cache block frames for both cache sizes. This does not suffice for the 128x128 point grid size. In fact more blocks map into the 64Kbyte cache permitting more shared blocks to be invalidated. The IR is constant for the 128x128 point grid and 32Kbyte cache as a result of the principle of vertical shared blocks. These shared blocks are the only ones invalidated for this cache size.



a. Two Processors



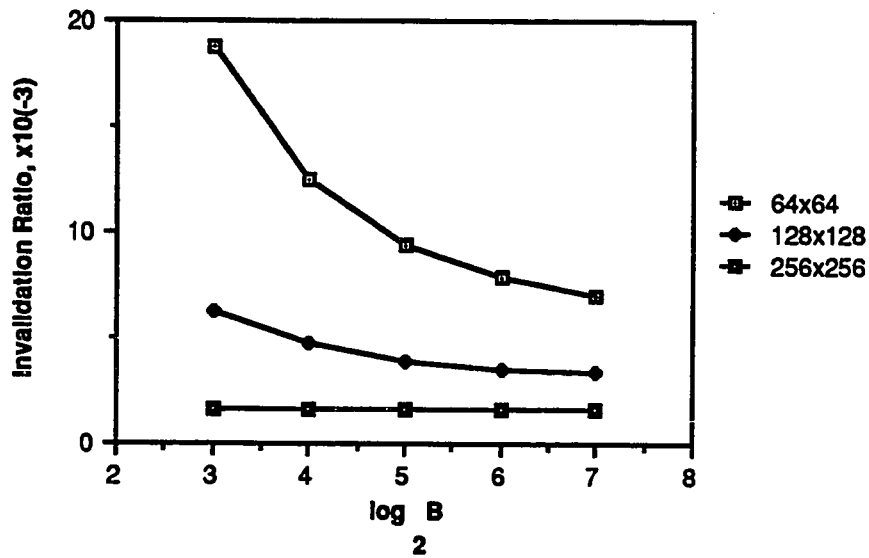
b. Four Processors

Figure 3.10 Invalidation Ratio versus Cache Blocksize, Square Decomposition, Direct Mapping Policy, Two and Four Processors.

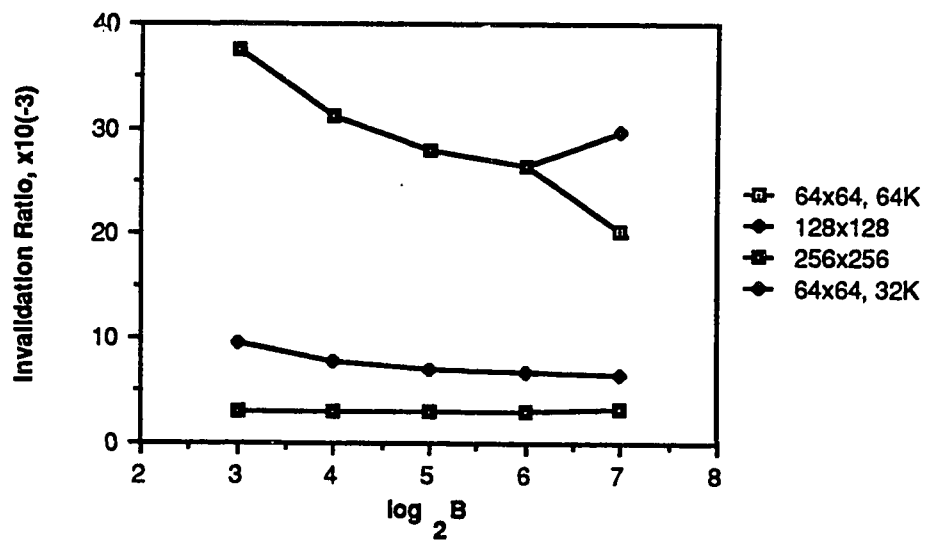
Figure 3.11 presents the IR versus the cache blocksize for 8 and 16 processors. All other design configurations are identical to those of the previous figure. For these processor configurations, the IR decreases as the blocksize increases with the exception of the 256x256 point grid size (both processors) and the 64x64 point grid with 16 processors and a 32Kbyte cache size. As a result of the principle of vertical shared blocks the IR is constant for the 256x256 point grid size. The IR decreases as the cache size is doubled for the 64x64 point grid size when using a 128-byte blocksize and 16 processors. This is because each block is shared by two processors. A significant number of invalidations occur as a result of this. While the INV-RWs are deterministic, all other invalidations are random in nature for this grid size/ blocksize attribute.

Figure 3.12 illustrates the IR versus the number of processors for direct mapping and a square grid decomposition. It shows an increase in the IR as the number of processors increase. While the parameter remains constant as the cache size increases for the 64x64 point grid, it increases from 0.0 to 3.5^{-3} for the 128x128 point grid (2 processors only) as the cache size is doubled. All IRs for direct mapping placement and square domain decomposition are less than 0.4. The IR is 0.0 for grid sizes not shown in this figure.

Figures 3.13 and 3.14 show the IR versus blocksize for 2-16 processors, direct mapping and rectangular grid decomposition. These figures also indicate a decrease in the IR as the blocksize increases. The 128x128 point grid for the 2 processor system and the 256x256 grid for the 8 processor system both show nonzero IRs for the 64Kbyte cache although the parameter is zero for the 32Kbyte cache. This is because



a. Eight Processors



b. Sixteen Processors

Figure 3.11 Invalidation Ratio versus Cache Blocksize, Square Decomposition, Direct Mapping Policy, Eight and Sixteen Processors.

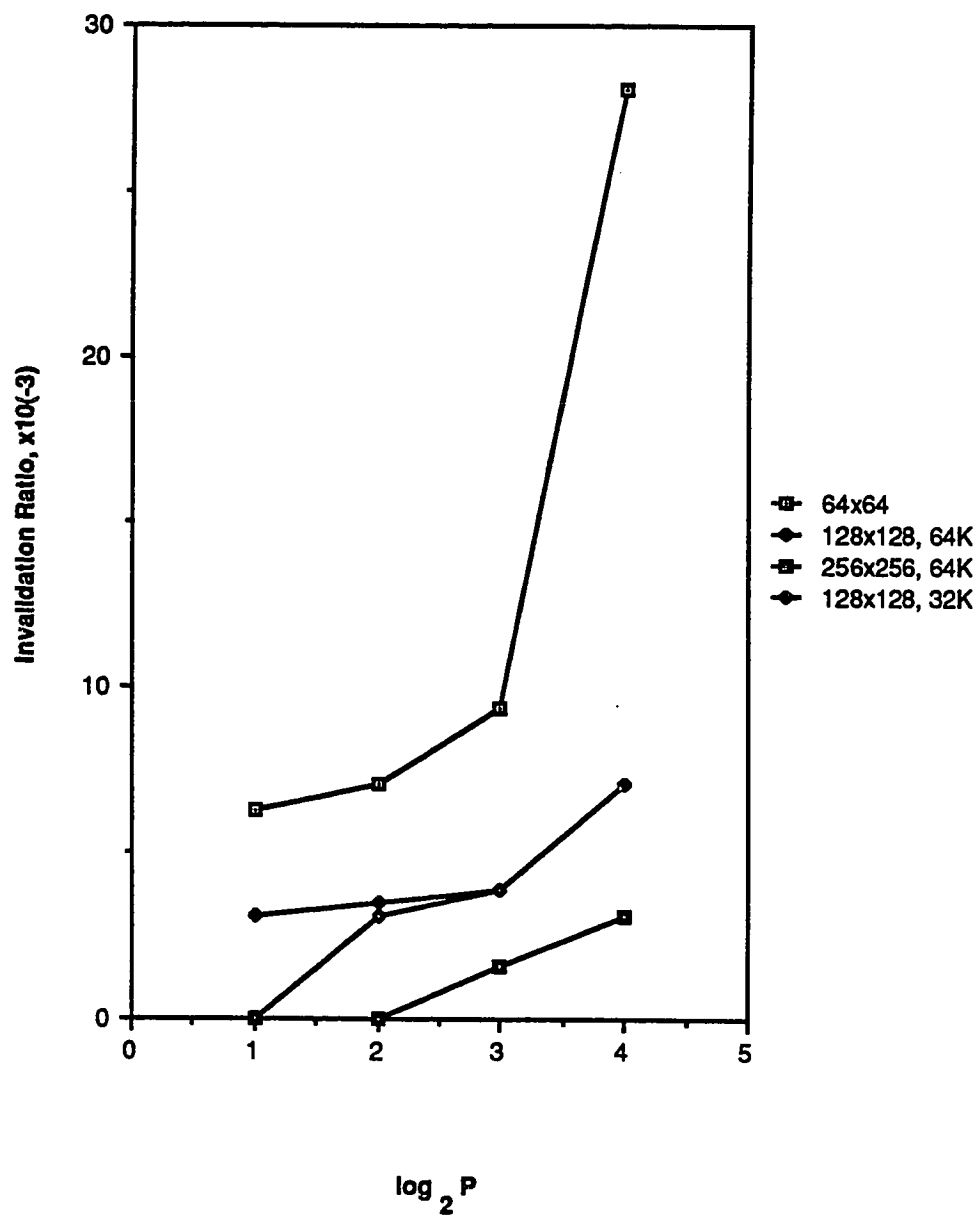
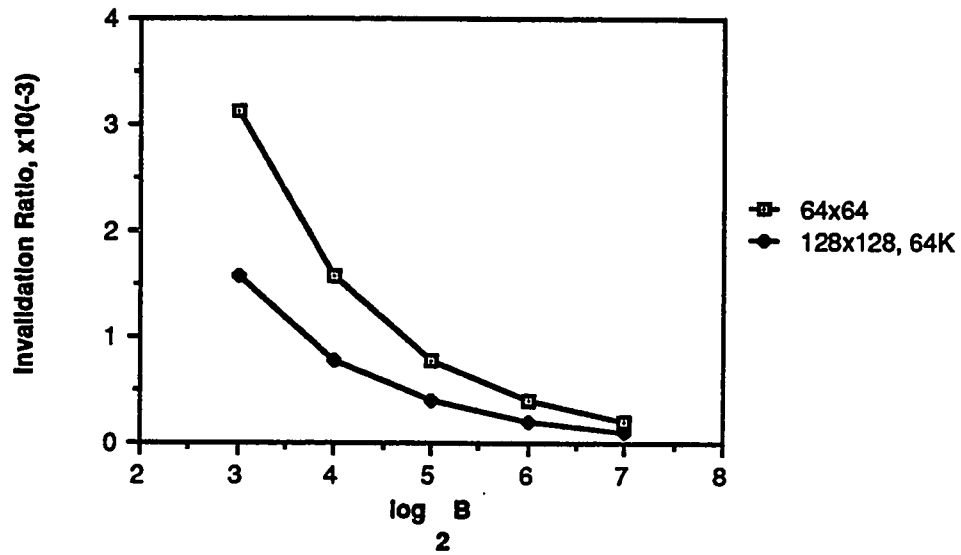
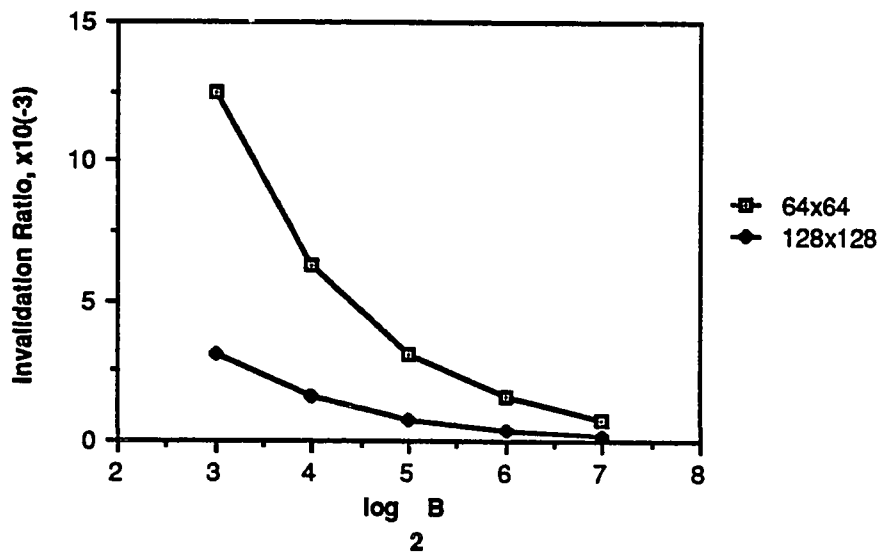


Figure 3.12 Invalidation Ratio versus the Number of Processors, Square Decomposition, Direct Mapping Policy, 32-byte Blocksize.

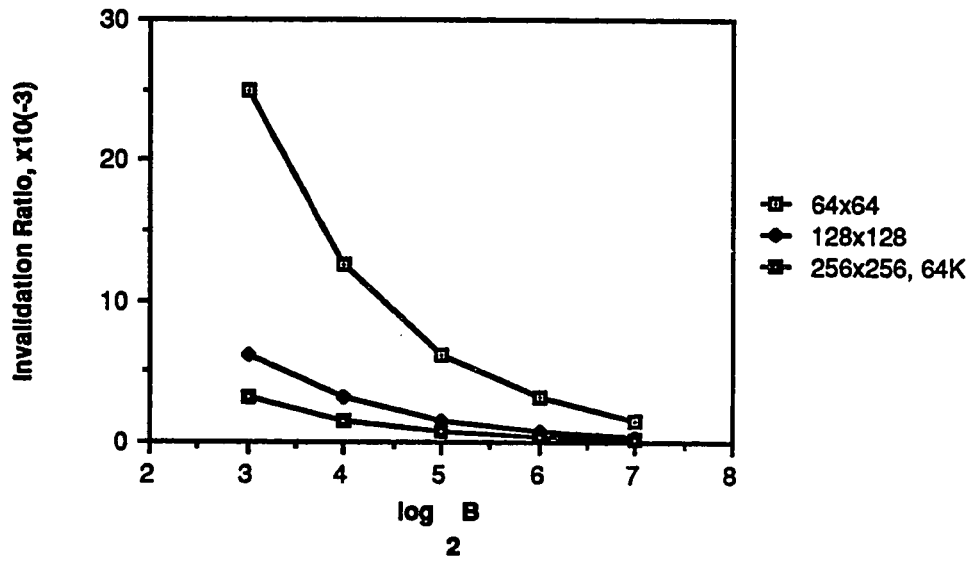


a. Two Processors

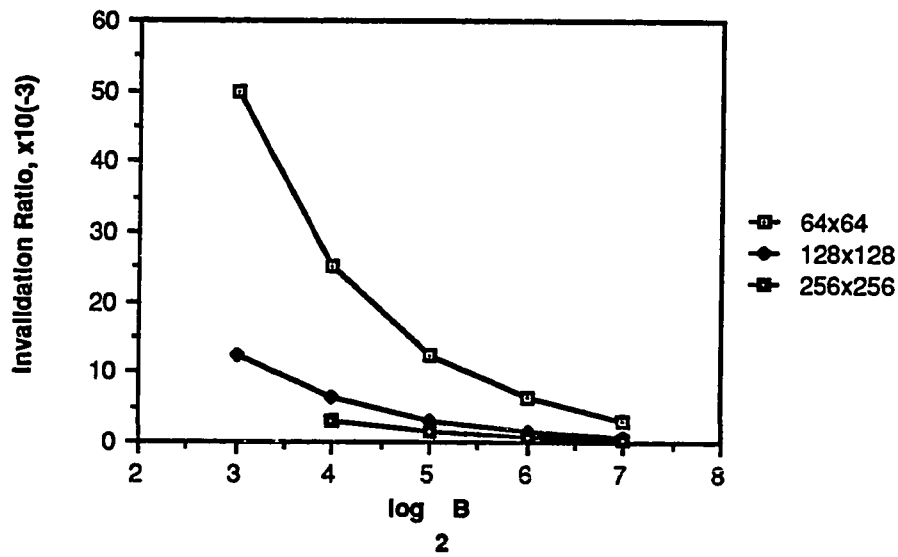


b. Four Processors

Figure 3.13 Invalidation Ratio versus Cache Blocksize, Rectangular Decomposition, Direct Mapping, Two and Four Processors.



a. Eight Processors



b. Sixteen Processors

Figure 3.14 Invalidation Ratio versus Cache Blocksize, Rectangular Decomposition, Direct Mapping, Eight and Sixteen Processors.

the increase in cache size allows more shared blocks to be cached simultaneously, allowing invalidations to occur. All other grid sizes show a IR independent of the cache size. Also note that for smaller grid sizes, the rectangular IR is larger than the square IR; however, the converse is true for larger grid sizes. The major cause of this behavior is the principle of vertical shared blocks for the square decomposition strategy. Grid sizes not shown have a no invalidation ratio. Figure 3.15 presents the IR versus the number of processors in the system for direct mapping and rectangular grid decomposition. Again, the IR increases as the number of processors increase. All IR values for direct mapping placement with rectangular domain decomposition are 0.05 or less.

Figures 3.16 through 3.18 illustrate the IR versus the blocksize for 4-16 processors, 2-way set-associative placement and square domain decomposition. All three processor configurations show at least an order of magnitude decrease in the IR for grid sizes larger than 128x128 points, a result of intragrid contention. The IR decreases as the blocksize increases in all instances. In the 16 processor system, the IR increases as the blocksize increases from 64 to 128 bytes. This is because all blocks are shared by two processors causing more invalidations for this blocksize/grid size feature. The IR versus blocksize for the 2 processor system is identical to the IR for direct mapping placement as shown in Figure 3.13a. Figure 3.19 presents the IR versus the number of processors for 2-way set-associative placement and the square decomposition strategy.

Figures 3.20 through 3.23 show the IR versus blocksize for 2-way set-associative placement and the rectangular decomposition strategy for 2-16

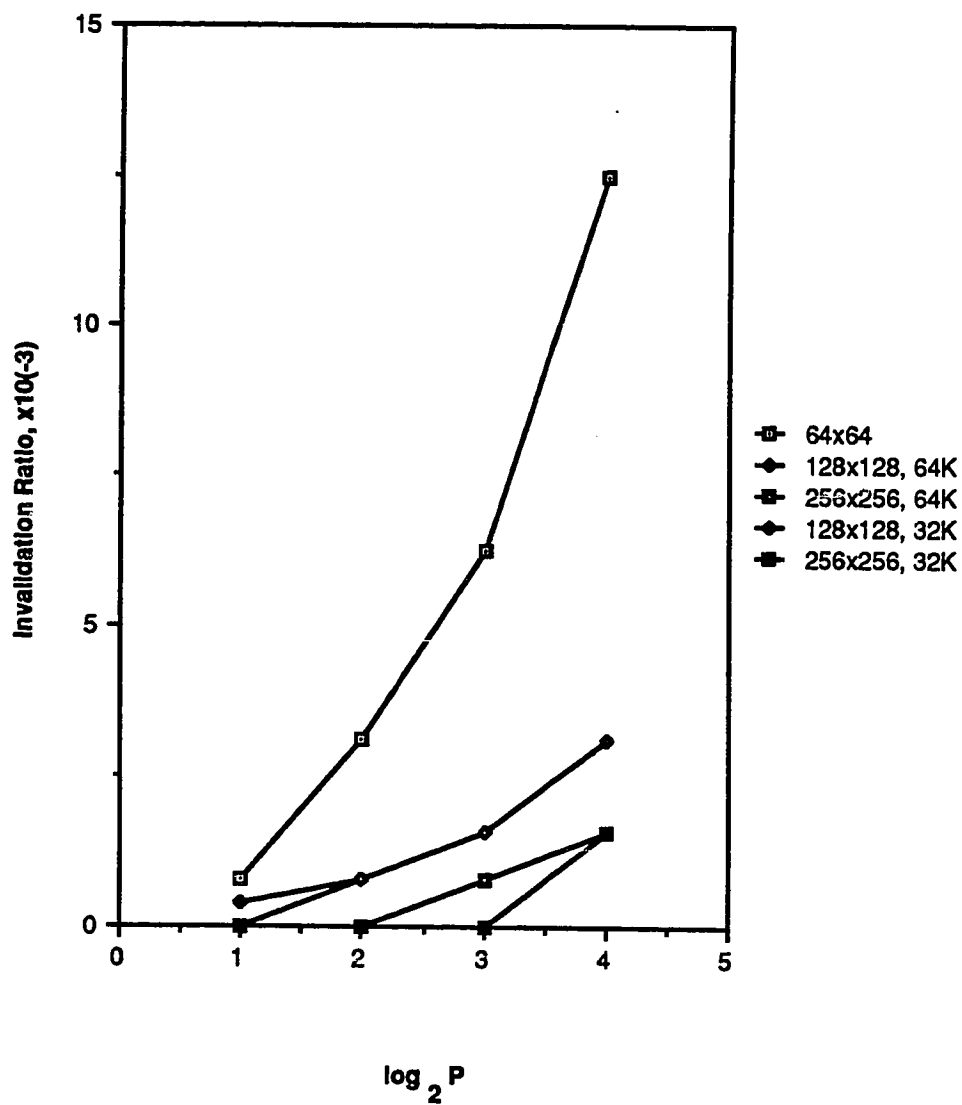
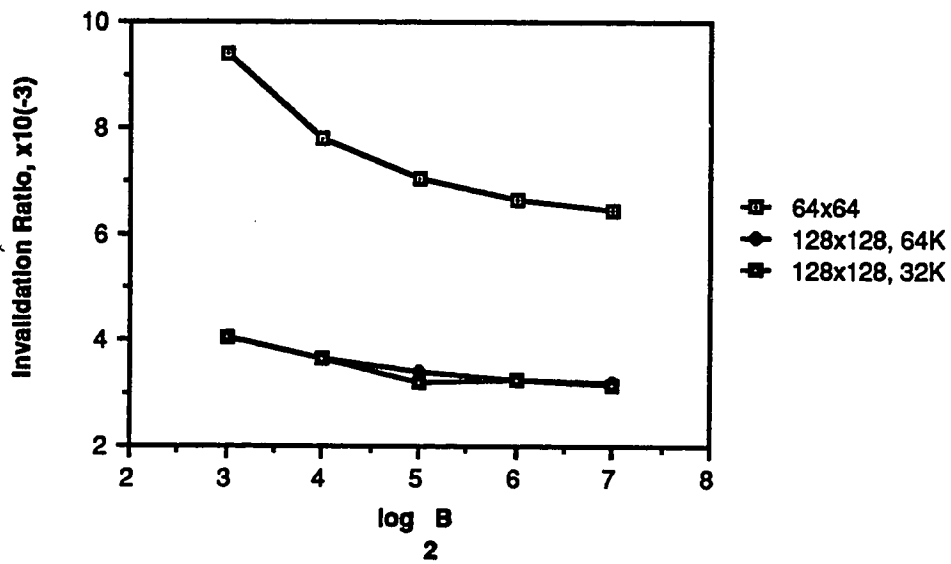
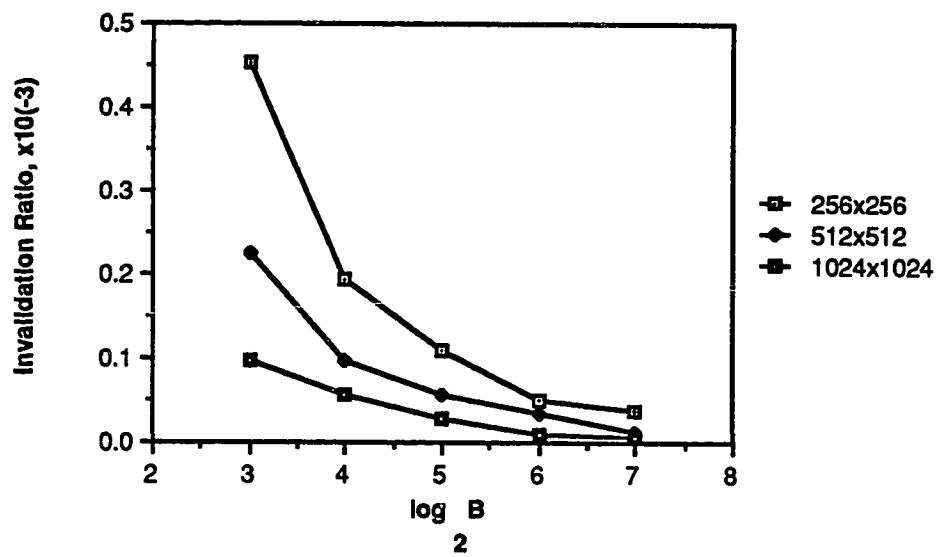


Figure 3.15 Invalidation Ratio versus the Number of Processors, Direct Mapping, Rectangular Decomposition, 32-byte Blocksize.

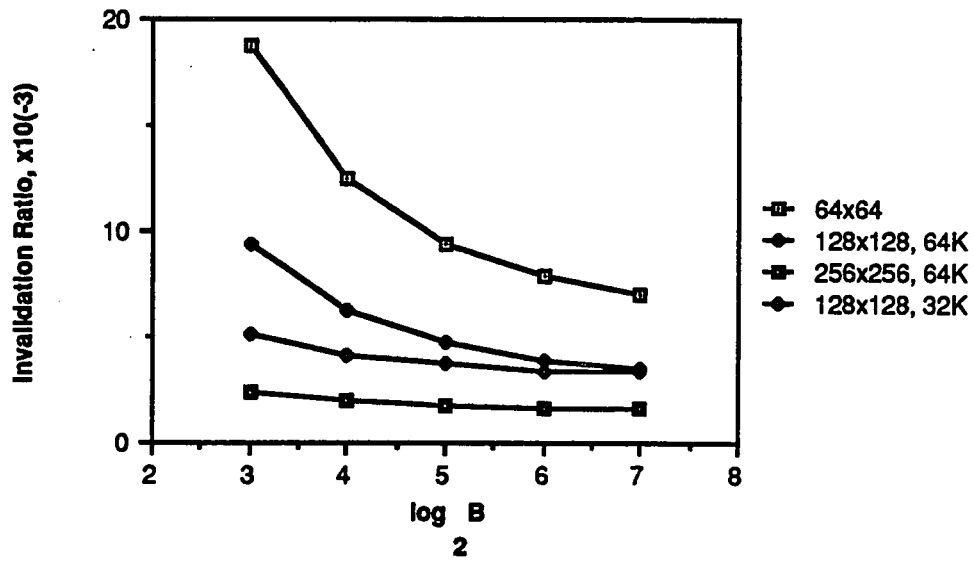


a. Smaller Grid Sizes

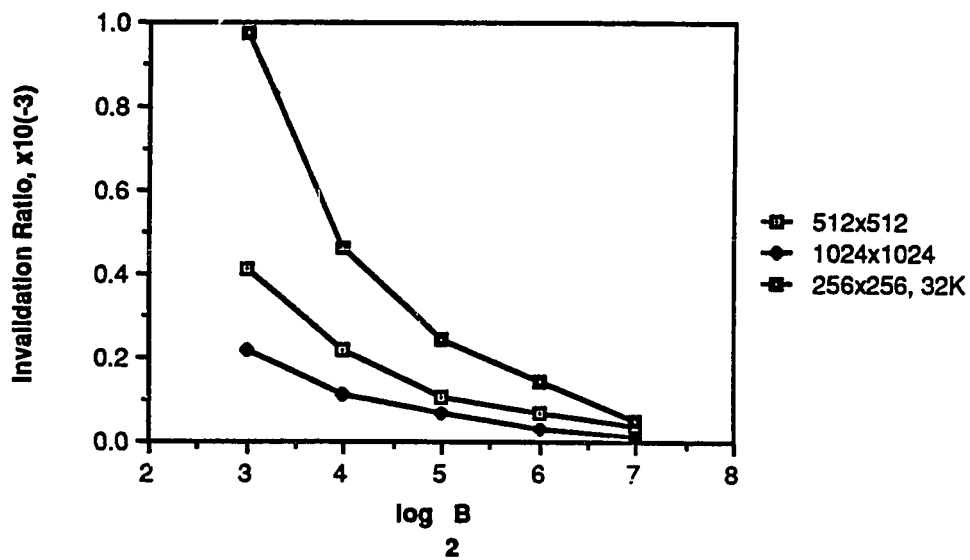


b. Larger Grid Sizes

Figure 3.16 Invalidation Ratio versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Four Processors.

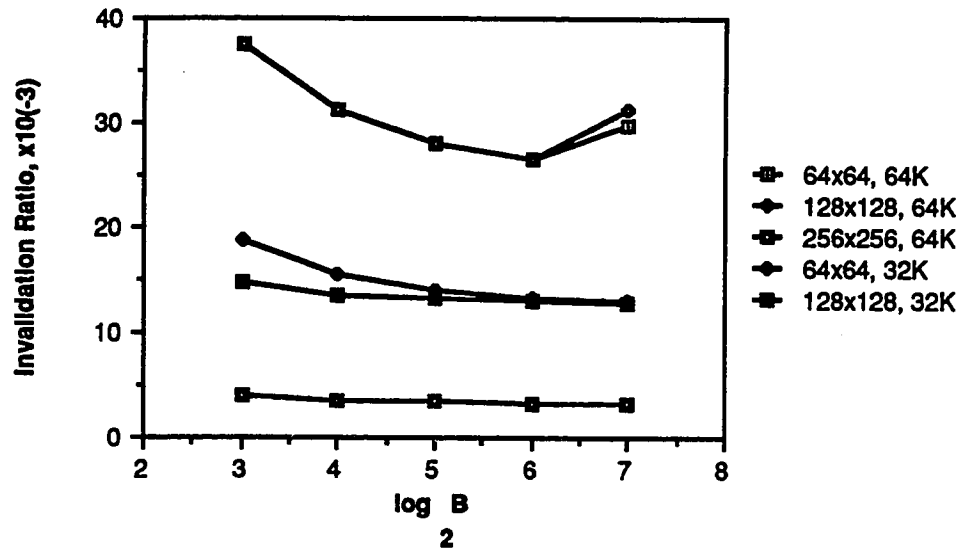


a. Smaller Grid Sizes

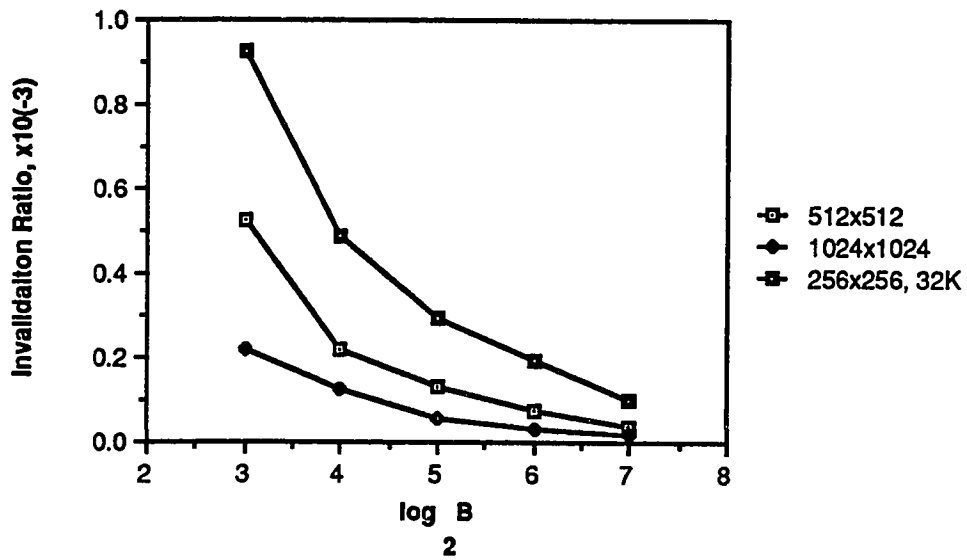


b. Larger Grid Sizes

Figure 3.17 Invalidation Ratio versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Eight Processors.

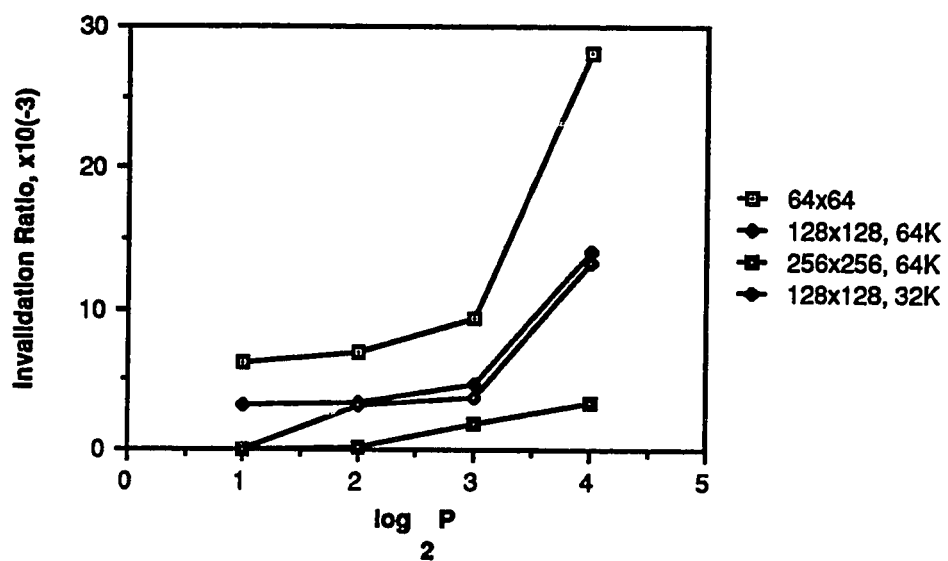


a. Smaller Grid Sizes

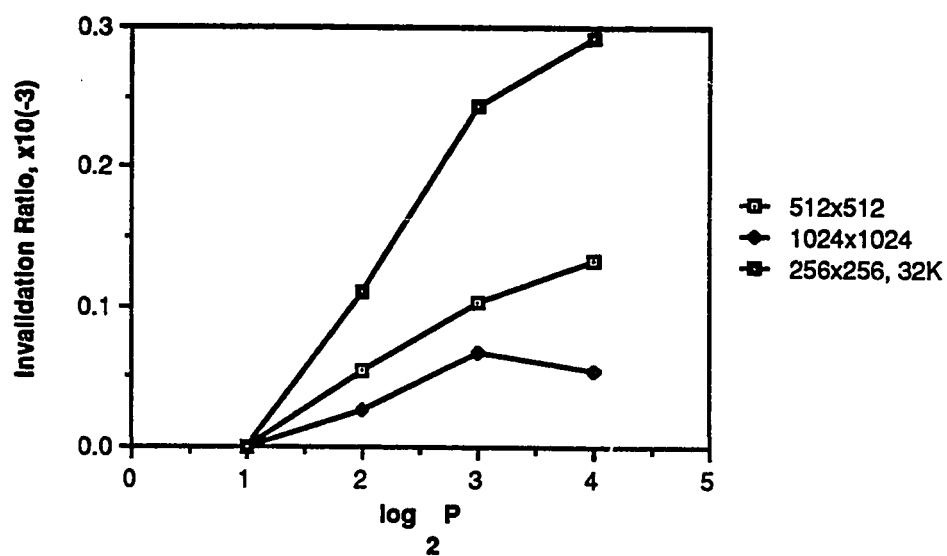


b. Larger Grid Sizes

Figure 3.18 Invalid ition Ratio versus Cache Block.size, Set-Associative Mapping, Square Decomposition, Sixteen Processors.

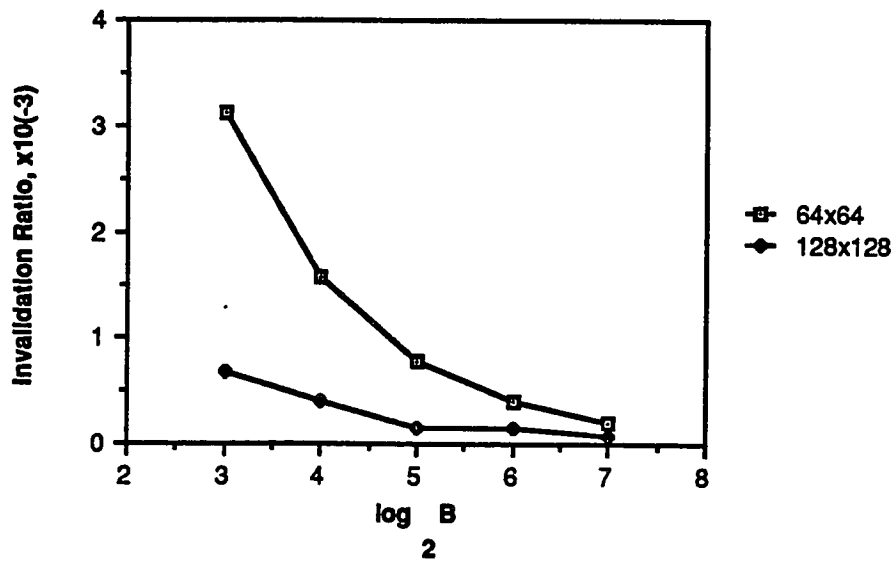


a. Smaller Grid Sizes

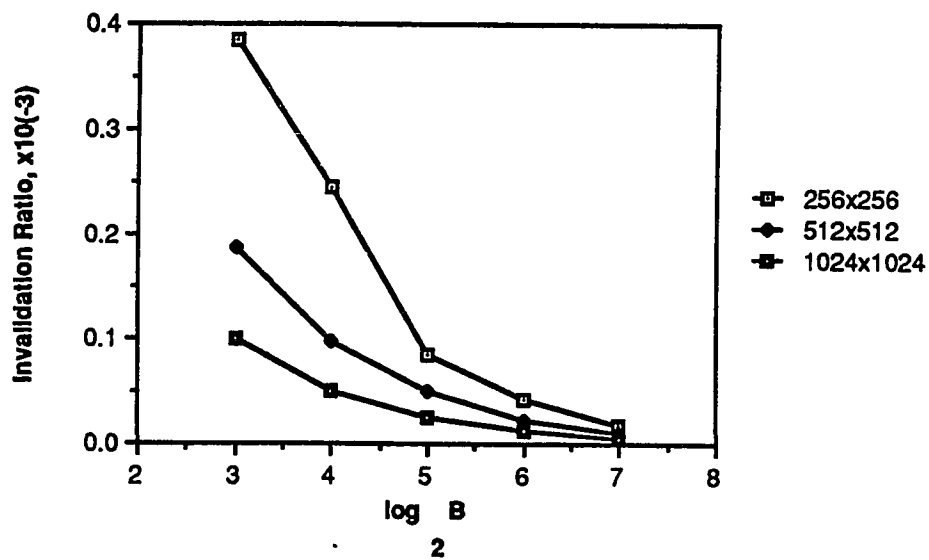


b. Larger Grid Sizes

Figure 3.19 Invalidation Ratio versus the Number of Processors, Set-Associative Mapping, Square Decomposition, 32-byte Blocksize.

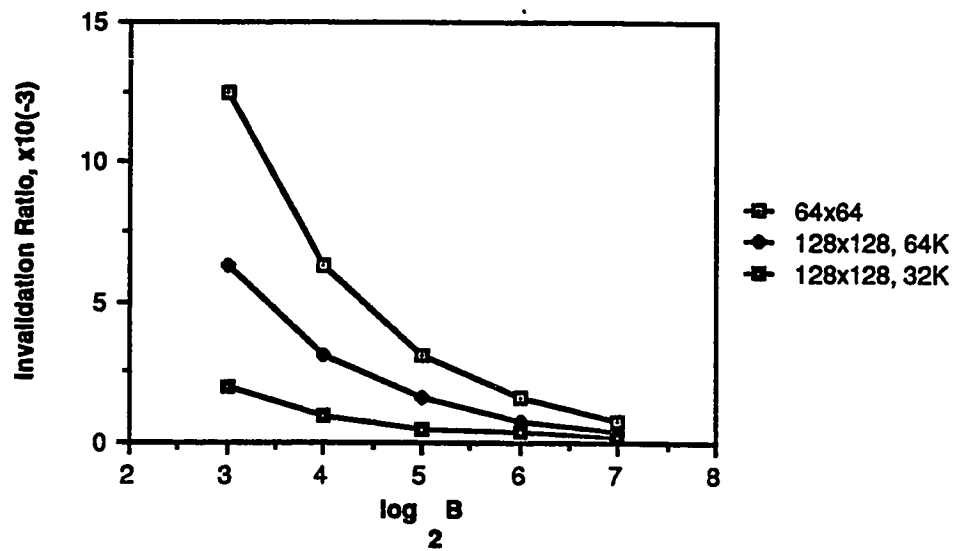


a. Smaller Grid Sizes

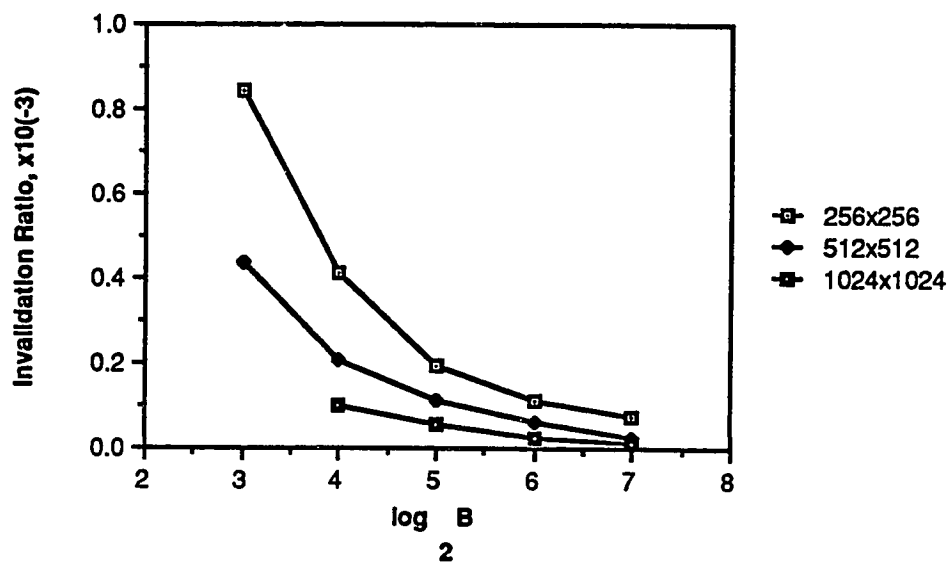


b. Larger Grid Sizes

Figure 3.20 Invalidation Ratio versus Cache Block size, Set-Associative Mapping, Rectangular Decomposition, Two Processors.

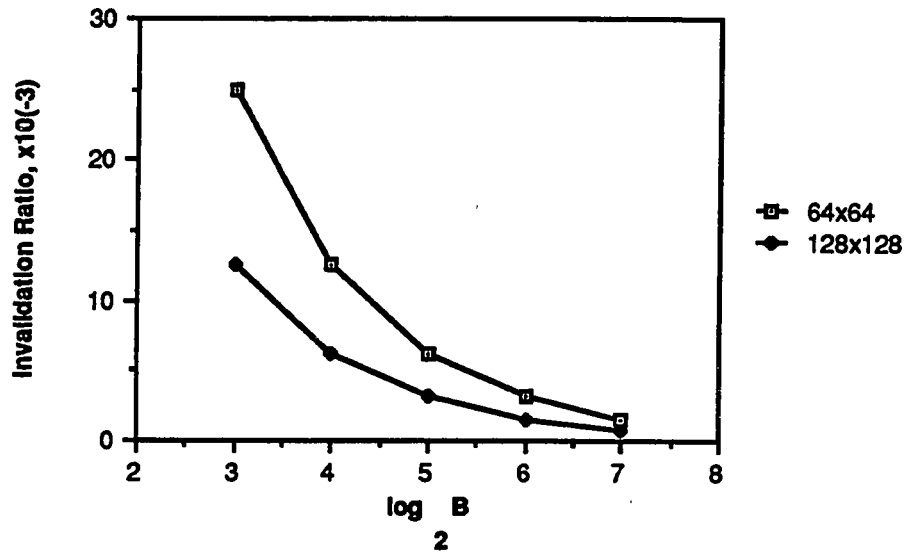


a. Smaller Grid Sizes

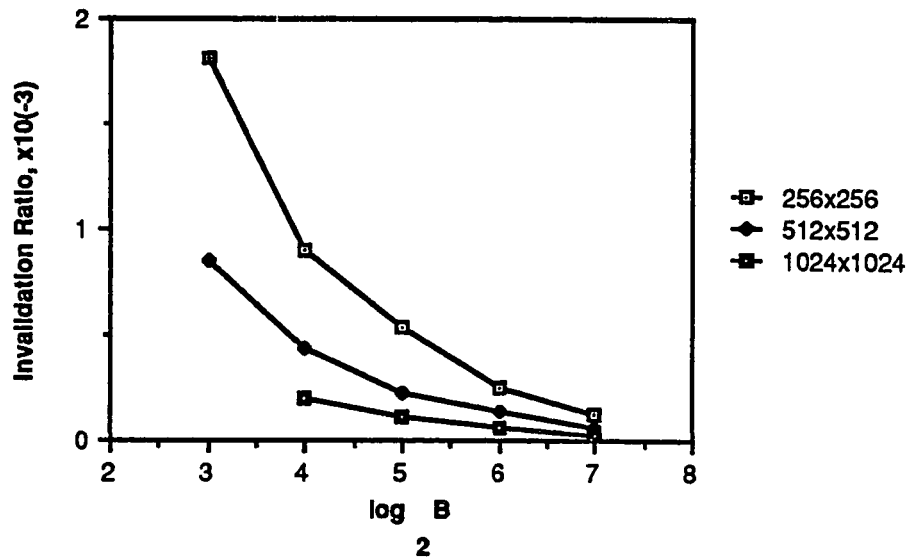


b. Larger Grid Sizes

Figure 3.21 Invalidation Ratio versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Four Processors.

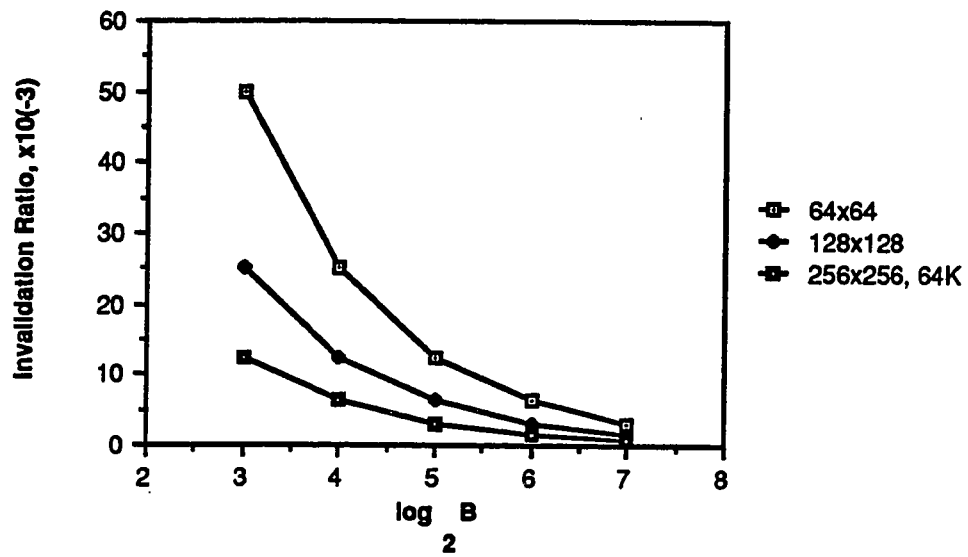


a. Smaller Grid Sizes

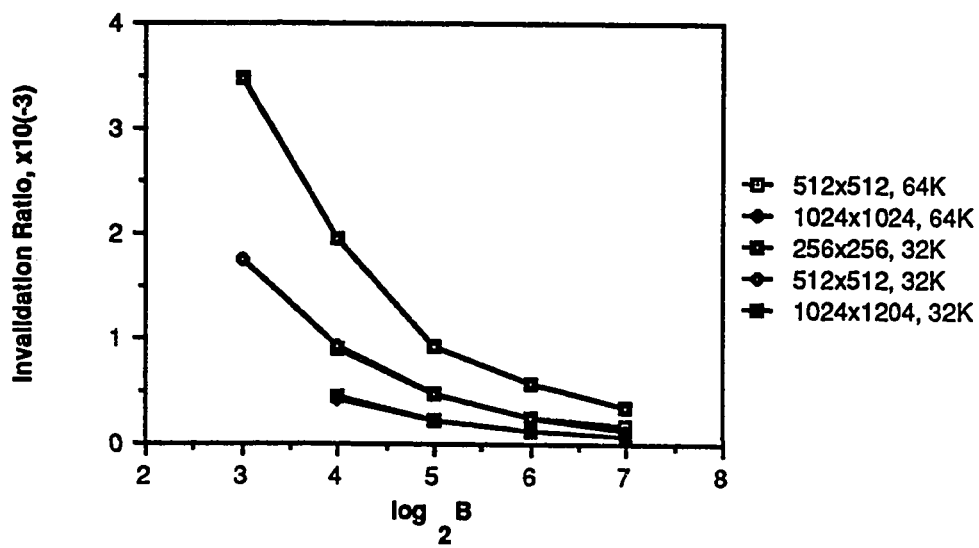


b. Larger Grid Sizes

Figure 3.22 Invalidation Ratio versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Eight Processors.



a. Smaller Grid Sizes



b. Larger Grid Sizes

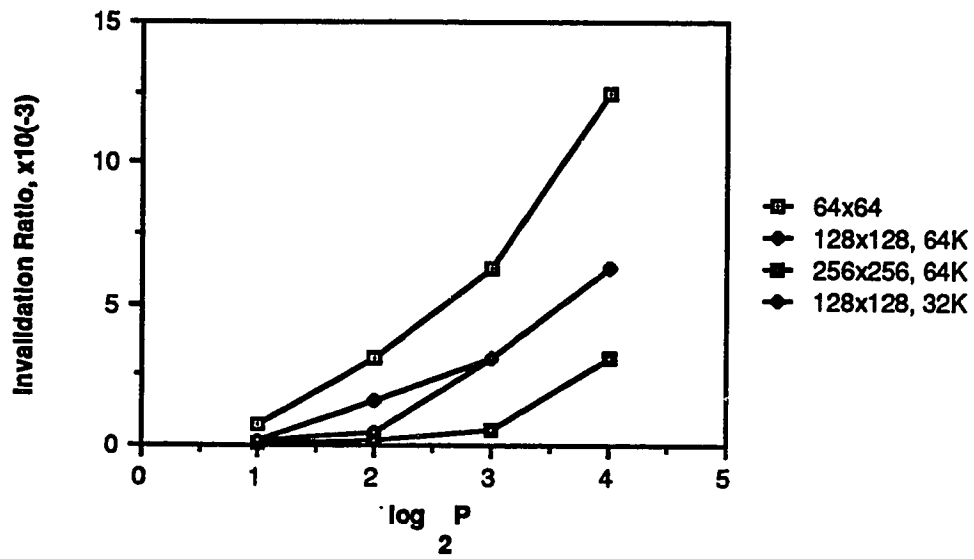
Figure 3.23 Invalidation Ratio versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Sixteen Processors.

processors. Again, this is an order of magnitude lower for grid sizes larger than 128x128 points. This IR also decreases as the blocksize increases. The rate of decrease, however, is faster than that of the square domain decomposition. In fact, for smaller blocksize the rectangular IR is larger than the square IR. As the cache blocksize increases the converse becomes true. This is the result of the principle of vertical shared blocks for the square domain decomposition. Figure 3.24 shows the IR versus the number of processors for these design features.

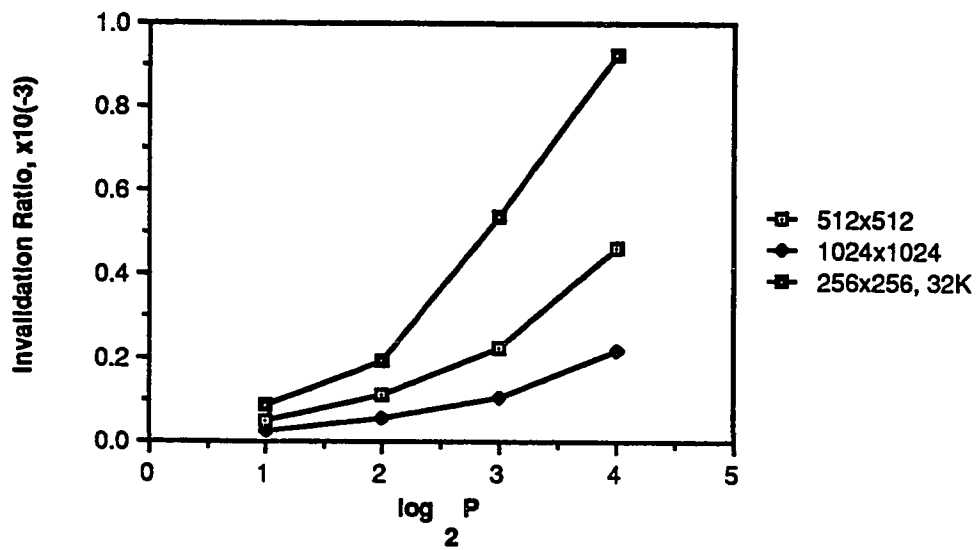
3.2.3. Cross-interrogate Cast-out

For Jacobi's iterative algorithm, the PDE grid is partitioned in such a way that no two processors will modify the same block. The exceptions are the modification of the shared variable, by all processors, used to synchronize the algorithm and executing the algorithm on a 64x64 point grid using a 16 processor system with a 128-byte blocksize. With these exceptions noted, there are no INV-RWs performed for this algorithm. Consequently, the prXICO for this algorithm consists only of implicit cast-outs (RW->ROs).

Figure 3.25 shows the prXICO versus the blocksize for a two processor system with direct mapped caches and the square decomposition strategy. The value for the 64x64 point is constant at 6.25^{-3} , the result of a constant number of shared blocks along the vertical border dividing the grid. Since all blocks for both grids map into unique block frames for this gridsize, the parameter is independent of the cache sizes considered. For the 128x128 point grid and the 64Kbyte cachesize the prXICO is a low 2.4^{-6} . In fact only one XICO is performed by one of the two processors for each



a. Smaller Grid Sizes



b. Larger Grid Sizes

Figure 3.24 Invalidation Ratio versus the Number of Processors, Set-Associative Mapping, Rectangular Decomposition, 32-byte Blocks size.

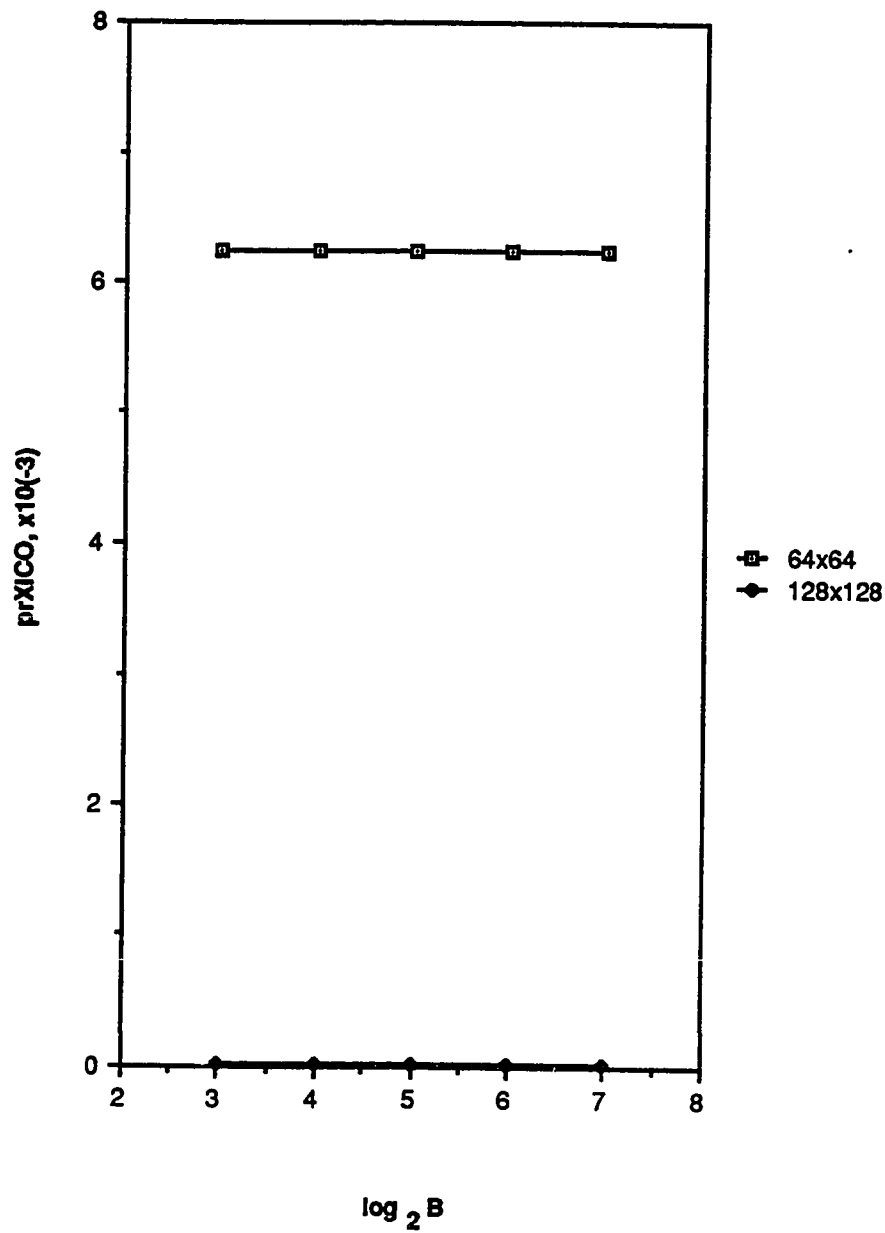


Figure 3.25 prXICO versus Cache Blocksize, Direct Mapping, Square Decomposition, Two Processors.

iteration as illustrated in Figure 3.26.

Part a of the figure shows how each shared block is tagged at the end of a $v^{k+1} = u^k$ iteration. All shared blocks mapped into the cache are RO copies of the U grid except the lower right shared block of P0. Since the algorithm modifies each point from left to right, top to bottom, this block was the last one modified by P0. Therefore, a modified V block is mapped into the cache of P0 as RW. Part b of the figure illustrates the process taken by P1 to modify a grid point in block a of the U grid during the next iteration. Observe that the western neighbor needed to modify this grid point is located in the shared block of the V grid present RW in the cache of P0. Processor P1 therefore initiates a XICO, the only one performed for this iteration.

Figures 3.27 through 3.29 show the prXICO versus blocksize (4-16 processors, respectively) for both cache sizes, square grid decomposition and all grid sizes simulated for the direct mapping strategy. Figures 3.30 through 3.33 present the same graphs for the rectangular grid decomposition. All figures indicate a decrease in the prXICO as the block size increases for the 64x64 grid size. Larger grid sizes show the prXICO between one and two orders of magnitude lower than the prXICO for the 64x64 point grid. This is attributed to the fact that for these larger grid sizes, only one XICO is performed by at least one processor per iteration. In addition to the exceptions discussed earlier, the exception is the square decomposition of the 128x128 point grid. The decomposition is such that for the 4 processor system, when the cache sizes doubles the number of XICOs double (from 1 to 2). At least one processor performs 2 XICOs for the 8 and 16 processor systems for this decomposition strategy. This is because more blocks map into the larger cache and one of them results in a

	$\underline{u_{RO}^0}$	$\underline{u_{RO}^1}$	
P0	$\underline{u_{RO}^0}$	$\underline{u_{RO}^1}$	P1
	$\underline{v_{RW}^0}$	$\underline{u_{RO}^1}$	

a. end of $v^{k+1} = u^k$ iteration

	$\underline{v_{RO}^0}$	$\underline{v_{RO}^1}$	
P0	$\underline{v_{RO}^0}$	$\underline{v_{RO}^1}$	P1
	$\underline{v_{RW}^0} \leftarrow \underline{a}$		

b. during the execution of the $u^{k+2} = v^{k+1}$ iteration,
P1 updates block a of u grid.

Figure 3.26 Example of an XICO.

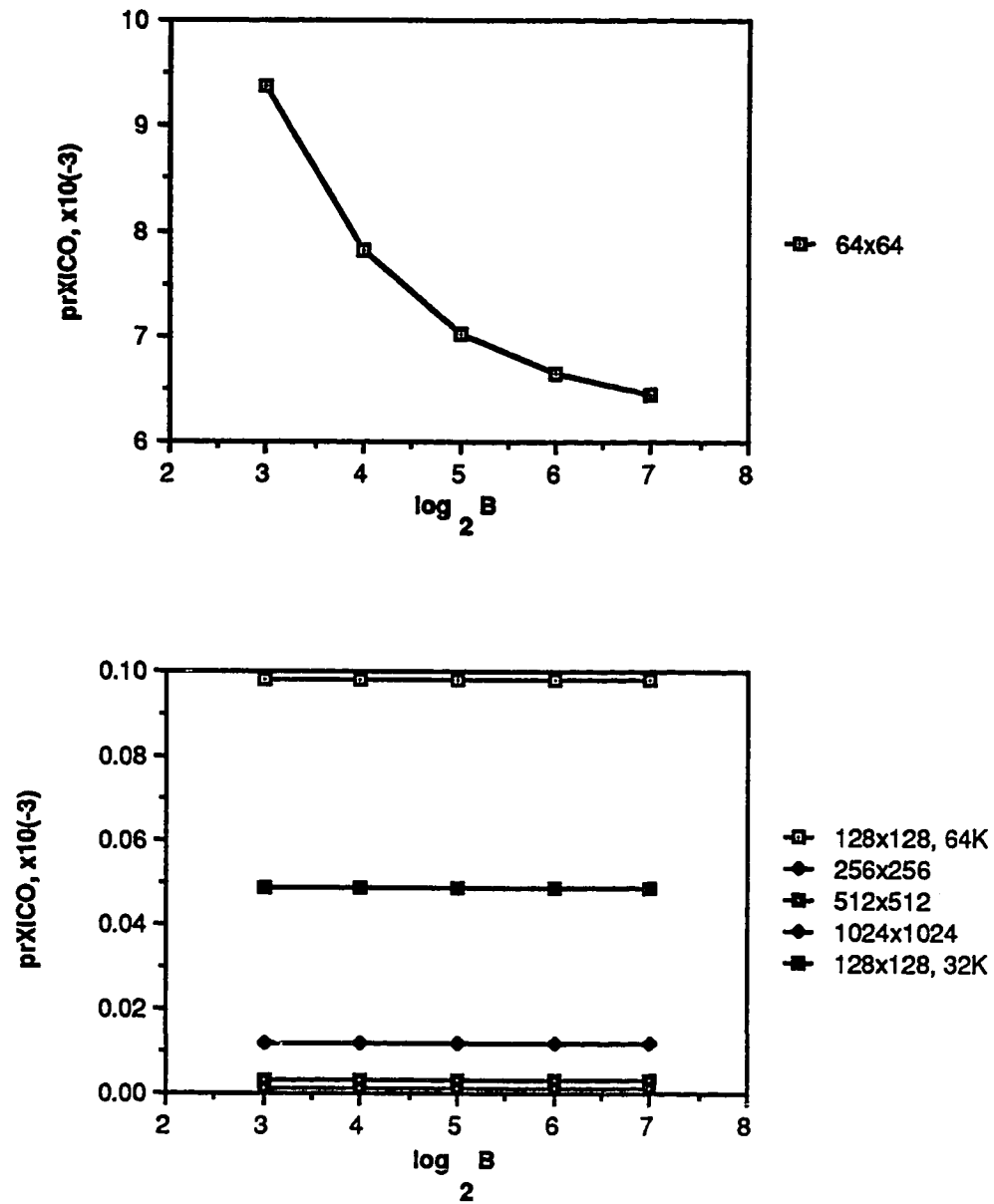


Figure 3.27 prXICO versus Cache Blocksize, Direct Mapping, Square Decomposition, Four Processors.

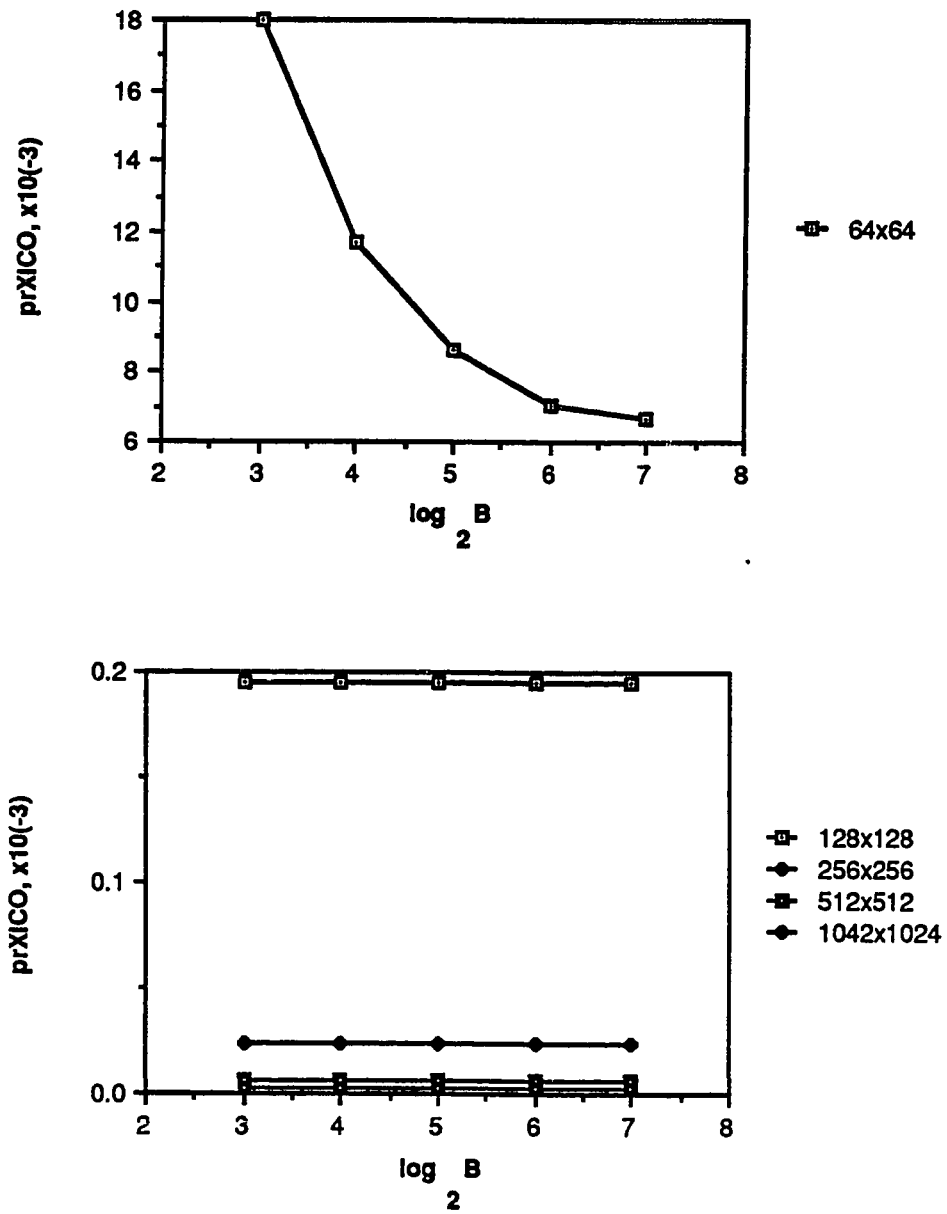


Figure 3.28 prXICO versus Cache Blocksize, Direct Mapping, Square Decomposition, Eight Processors.

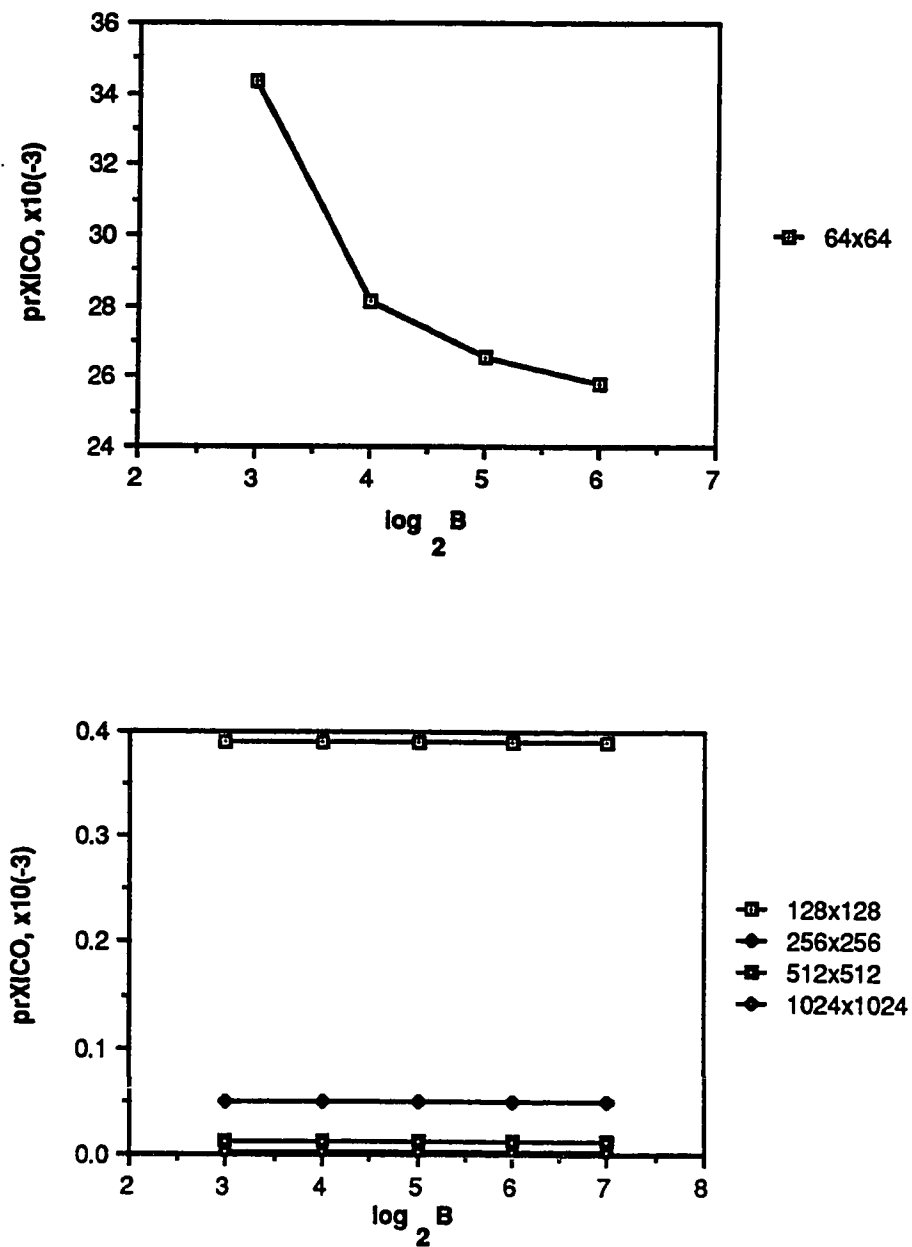


Figure 3.29 prXICO versus Cache Blocksize, Direct Mapping, Square Decomposition, Sixteen Processors.

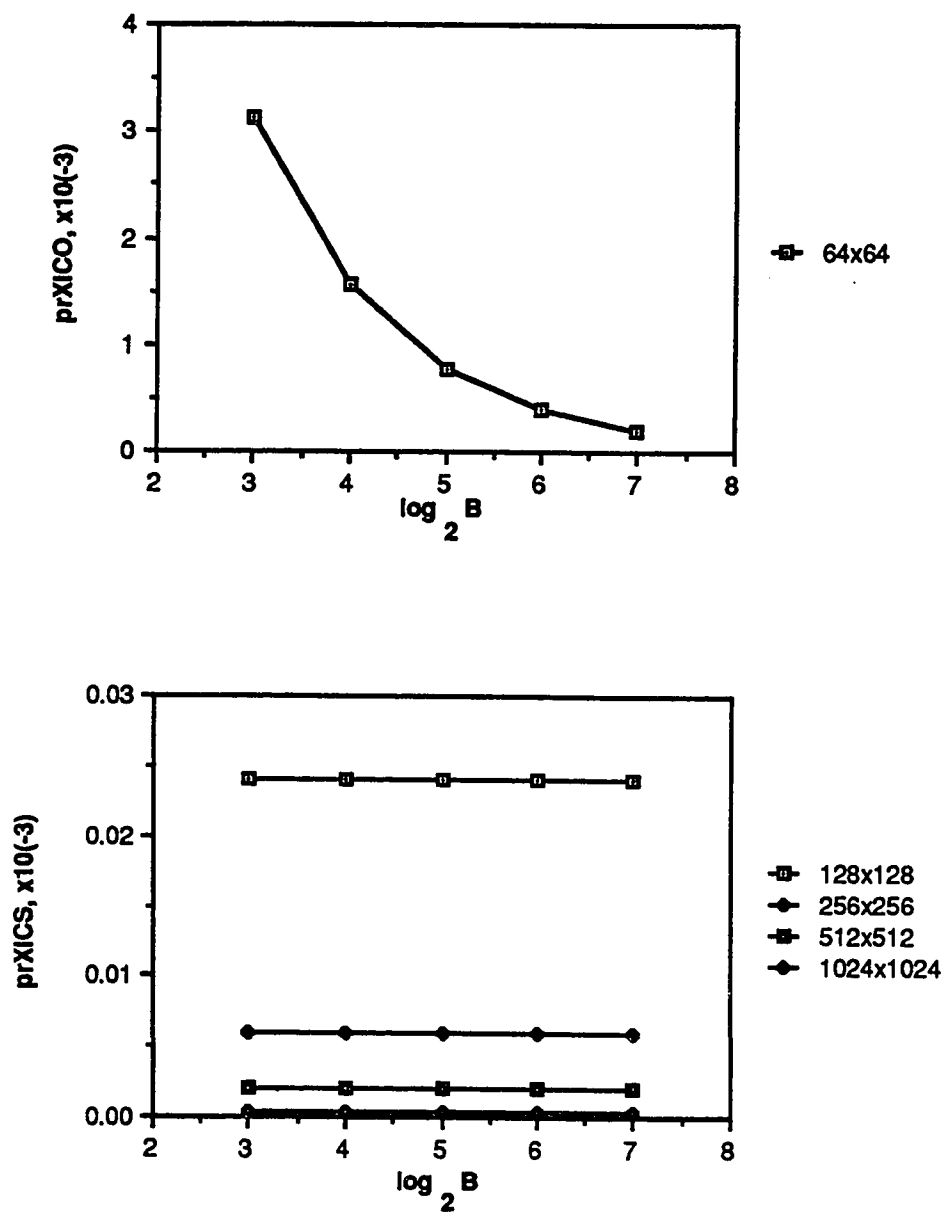


Figure 3.30 prXICO versus Cache Blocksize, Direct Mapping, Rectangular Decomposition, Two Processors.

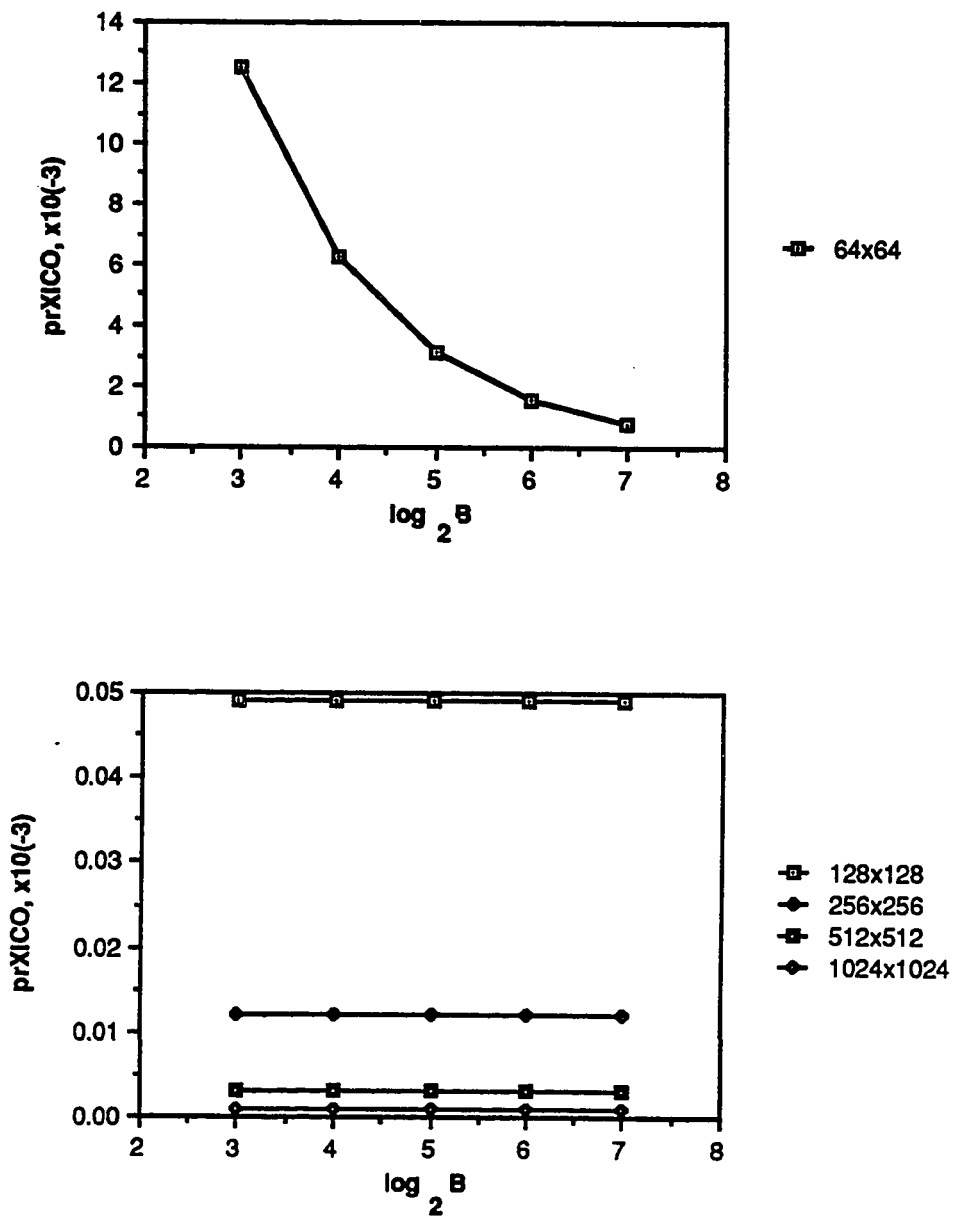


Figure 3.31 prXICO versus Cache Blocksize, Direct Mapping, Rectangular Decomposition, Four Processors.

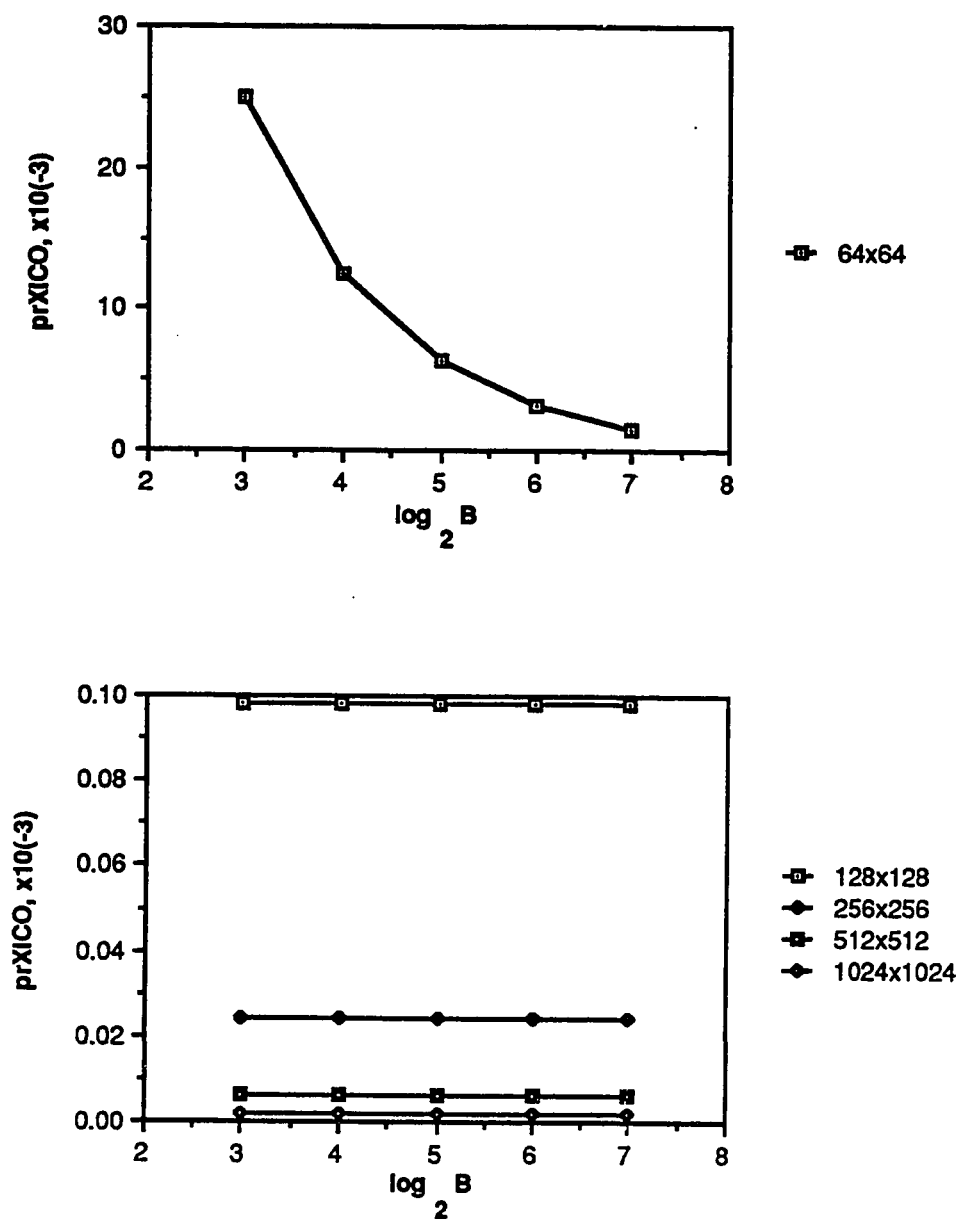


Figure 3.32 prXICO versus Cache Blocksize, Direct Mapping, Rectangular Decomposition, Eight Processors.

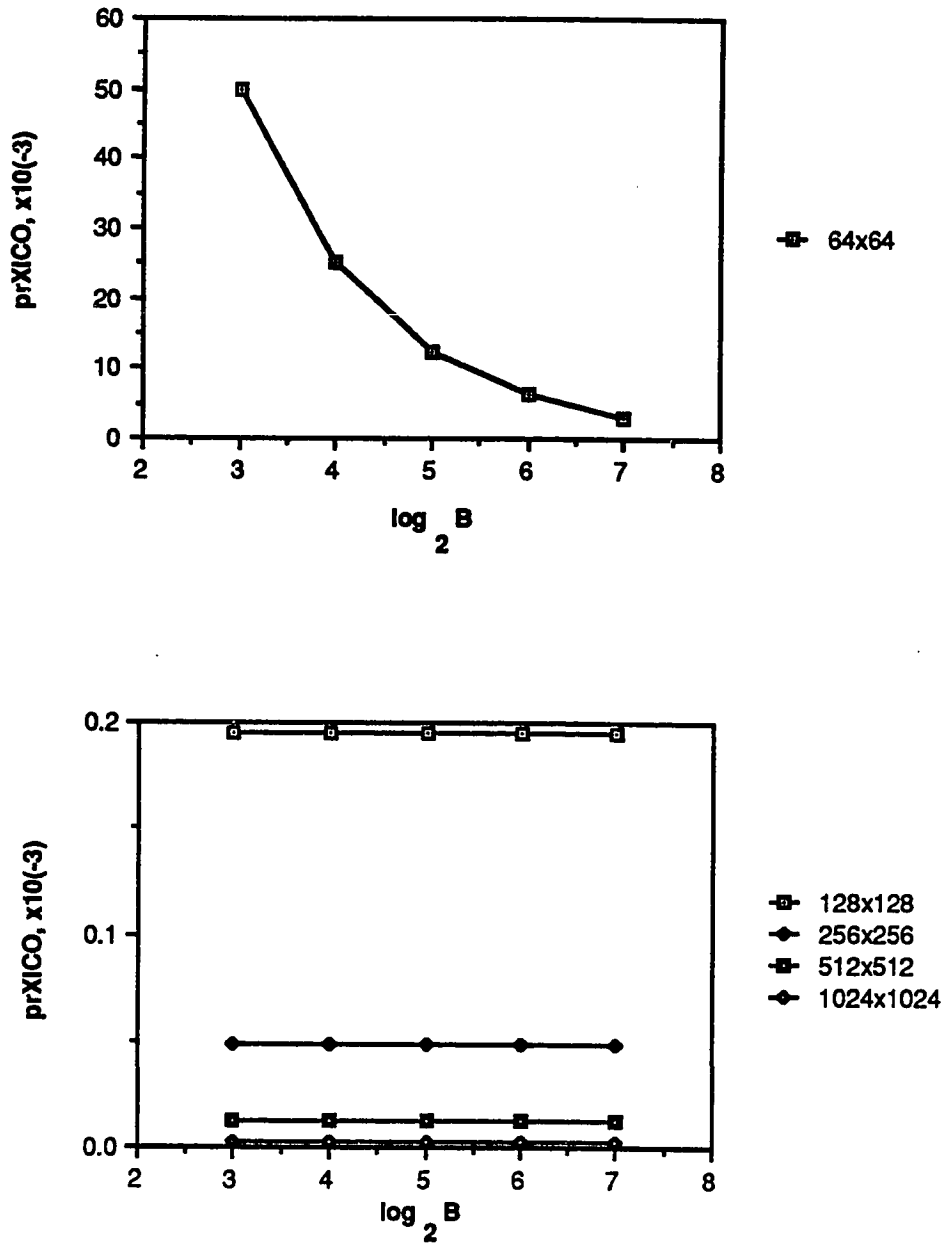


Figure 3.33 prXICO versus Cache Blocksize, Direct Mapping, Rectangular Decomposition, Sixteen Processors.

XICO for a processor.

Figure 3.34 presents the prXICO versus the number of processors for direct mapping, all grid sizes considered and both decomposition strategies. Part a shows this parameter for the two partitions of the 64x64 point grid size. As the number of processors increase, the prXICO increase. For this 32-byte blocksize, the square decomposition has a higher prXICO than the rectangular decomposition. A closer examination of the previous figures show the rectangular decomposition having a higher prXICO for lower blocksizes. In fact, for the rectangular decomposition, doubling the blocksize reduces the prXICO for the 64x64 point grid by one-half. This is not the case for the square decomposition as a result of the constant number of prXICOs that occur in conjunction with the shared blocks along the vertical boundaries. Therefore, as the blocksize doubles, the prXICO is reduced by less than one-half for the square decomposition.

Figure 3.34 b and c show an increase in the prXICO as the number of processors increases for the larger grid sizes. Also note the decrease in the prXICO as the grid size increases. This is a direct result of a higher frequency of intergrid and intragrid contentions for the larger grid sizes. Unlike the square decomposition strategy, at most only 1 XICO is performed on the 128x128 point grid per processor for the rectangular decomposition strategy. the prXICO for this grid size/partition feature is therefore half the value for the square partition. For grid sizes larger than 128x128 points the prXICO is independent of domain decomposition strategies.

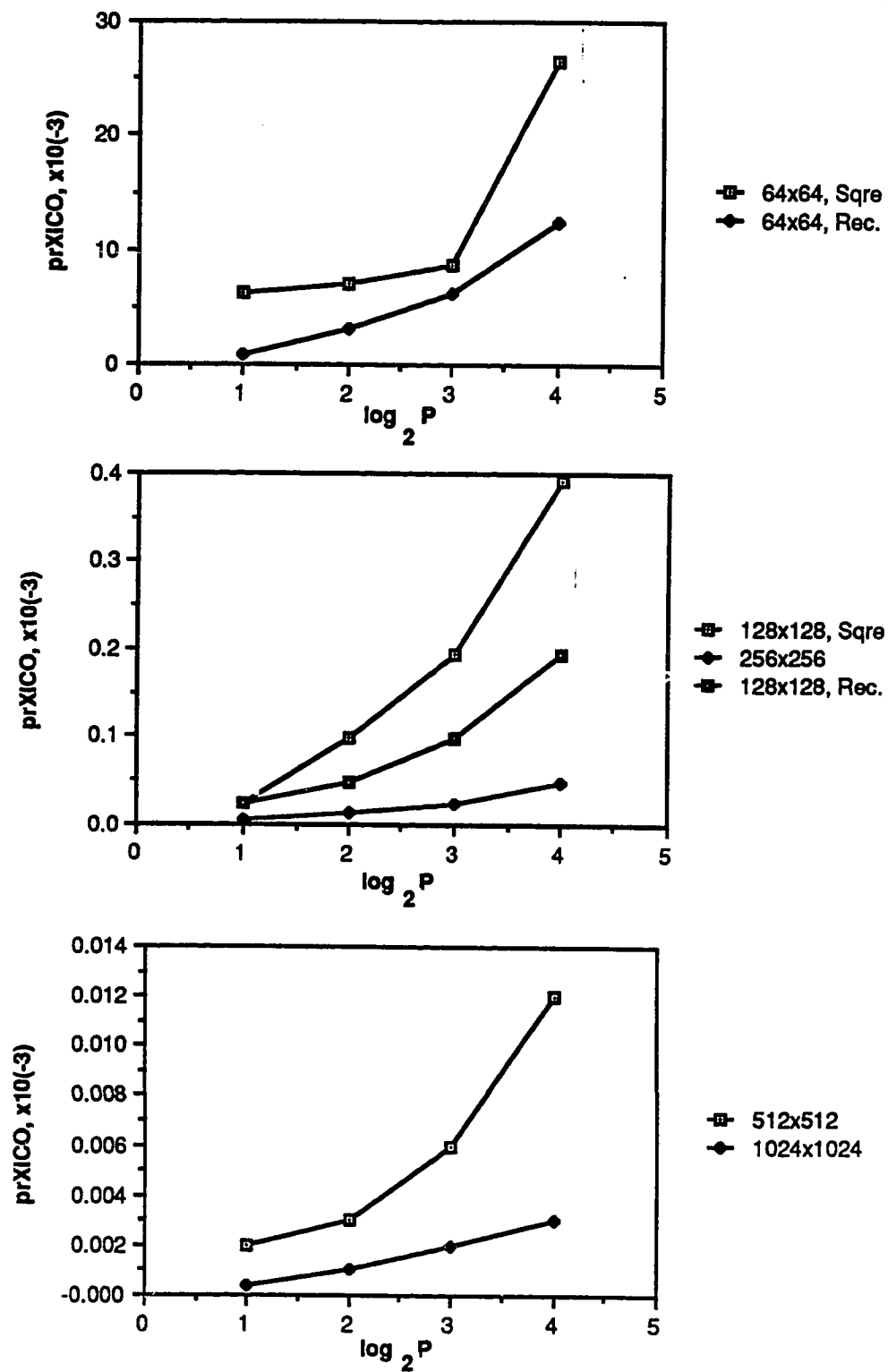
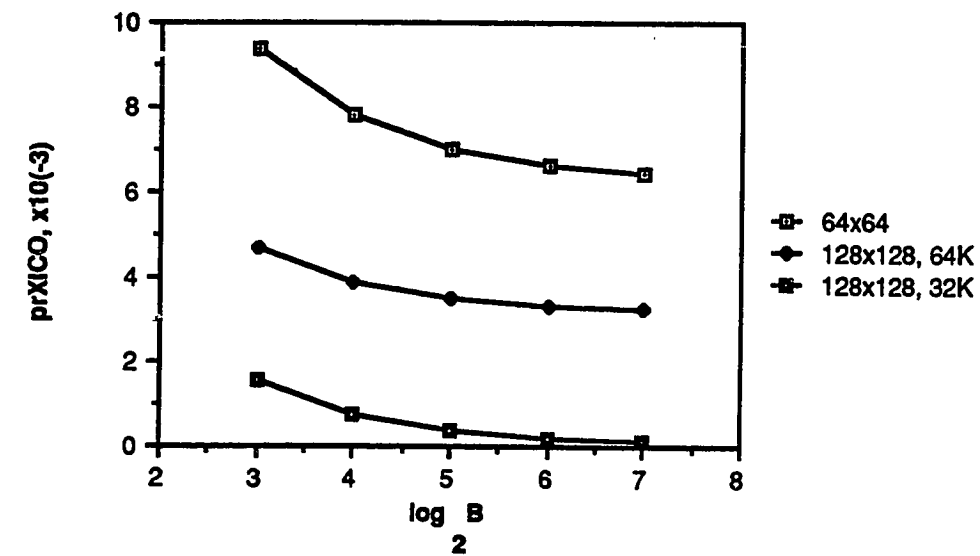


Figure 3.34 prXICO versus the Number of Processors, Direct Mapping, 32-byte Blocks.

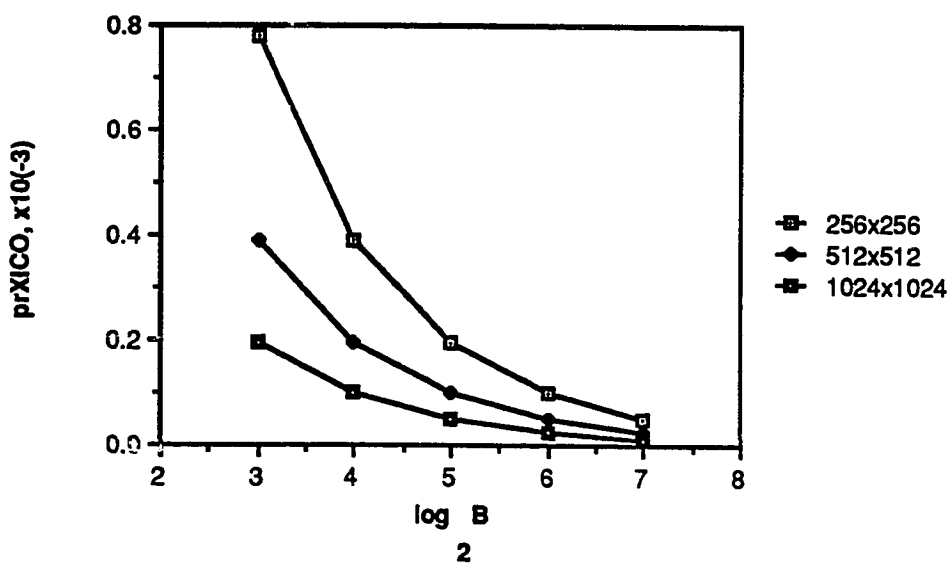
Figures 3.35 through 3.37 illustrate the prXICO versus blocksize for the square decomposition of all grid sizes with 2-way set-associative mapping for 4-16 processors. All graphs show a decrease in the prXICO as the blocksize increases. Also, the prXICO for grid sizes larger than 128x128 points is substantially lower than the smaller grid sizes; however, the differences are not as large as the direct mapping differences. This is because two blocks may map into the same cache set address. The prXICO for the 64x64 point grid for the 128-byte block and 16 processor system is 0.16 for both cache sizes. This value is an order of magnitude larger than the value for smaller block sizes for this grid. This results from the fact that two processors modify each block for this grid, causing numerous INV-RWs. The prXICO for the 2 processor system is 6.25^{-3} for the 64x64 point grid with set-associative mapping and square decomposition (all block sizes). Grid sizes larger than 64x64 points have no prXICO for this 2 processor system.

The prXICO versus the number of processors for a 32-byte block set-associative cache with a square partition of the grid is shown in Figure 3.38. For grid sizes smaller than 256x256 points, the value increases as the number of processors increase. The 64Kbyte cache value for the 128x128 point grid is larger than the smaller cache size for 4 or more processors because more blocks having the potential to be the object of a prXICO are present in the cache.

For grid sizes greater than or equal to 256x256 points the prXICO increases as the number of processors increase for 2 to 8 processors. The parameter value then remains constant as the number of processors increase from 8 to 16. This is because both processor configurations allocate a total of four processors to one column of the

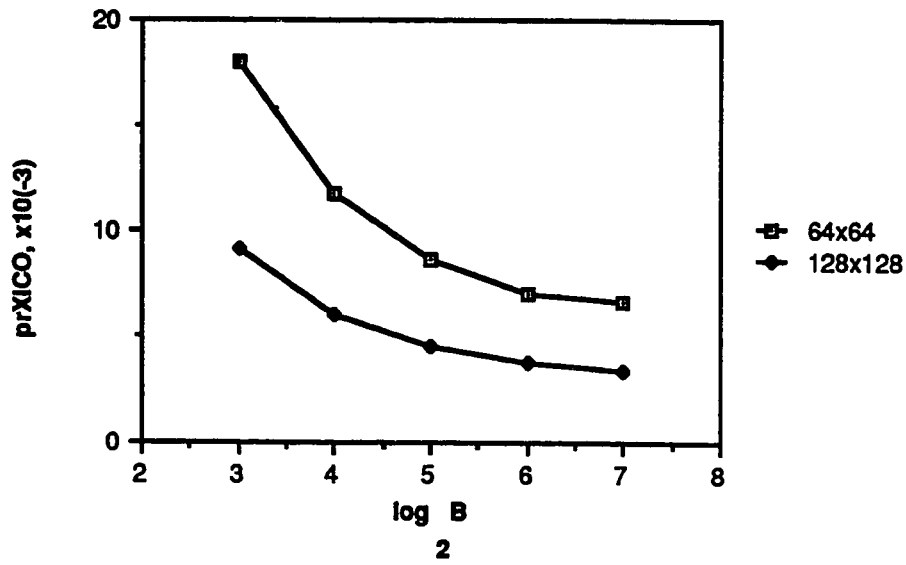


a. Smaller Grid Sizes

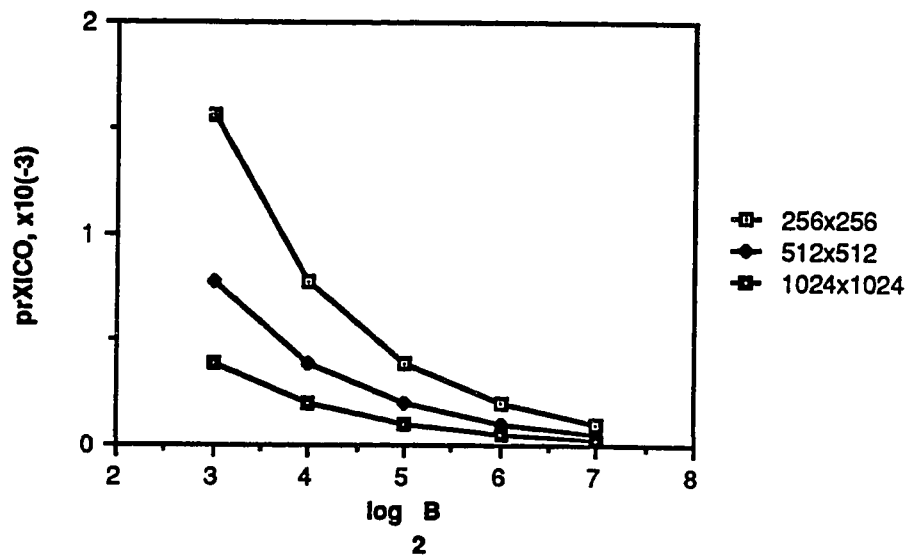


b. Larger Grid Sizes

Figure 3.35 prXICO versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Four Processors.

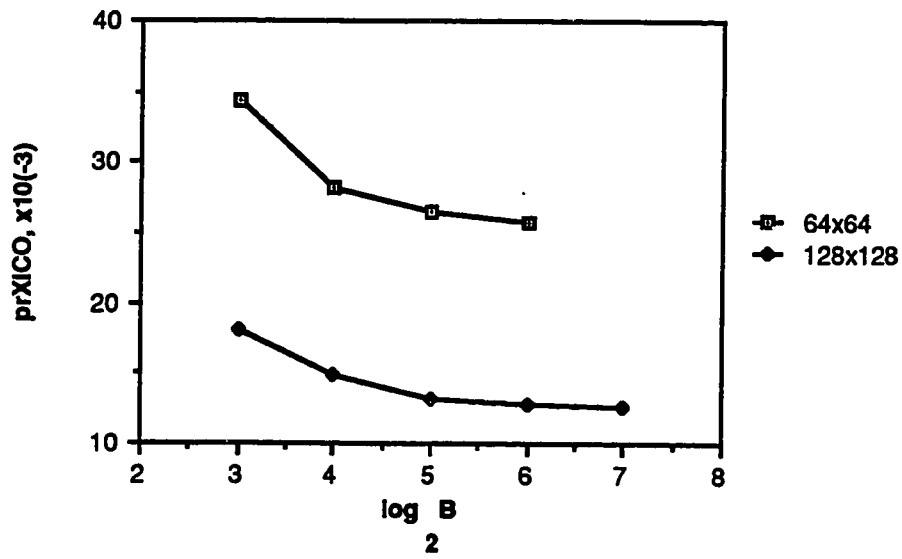


a. Smaller Grid Sizes

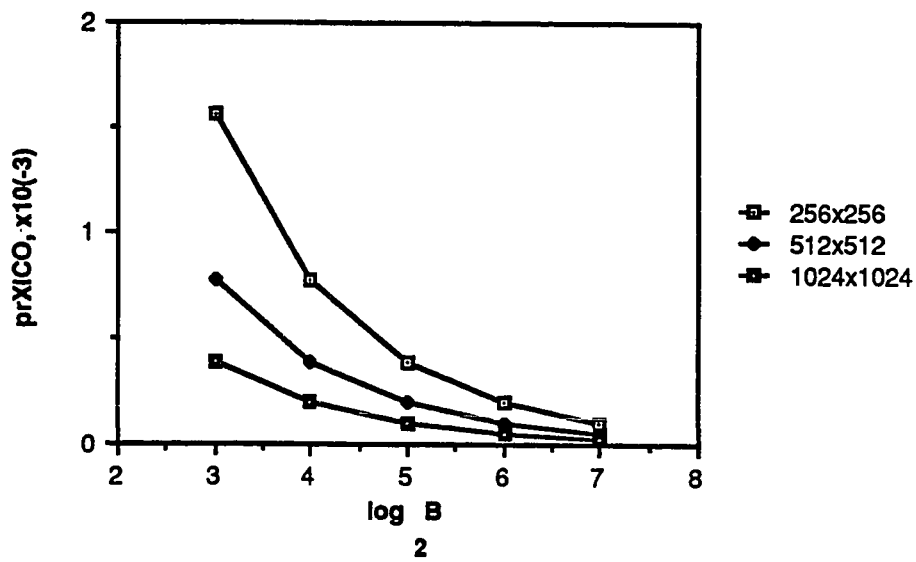


b. Larger Grid Sizes

Figure 3.36 prXICO versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Eight Processors.

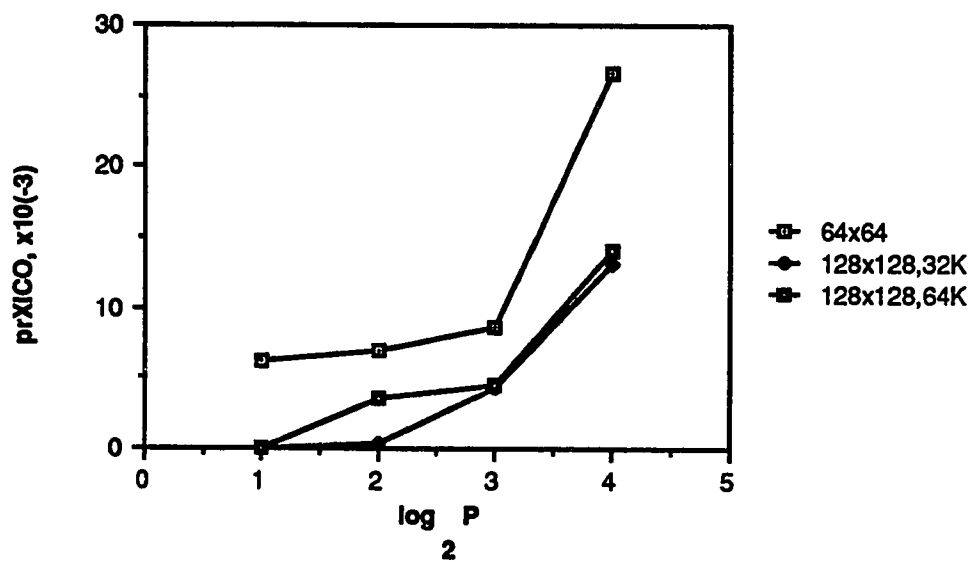


a. Smaller Grid Sizes

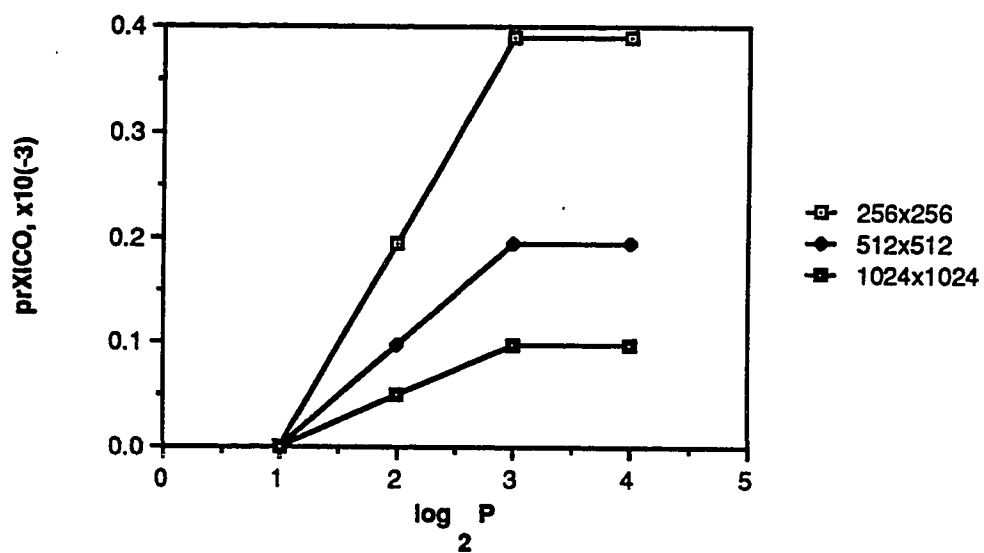


b. Larger Grid Sizes

Figure 3.37 prXICO versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Sixteen Processors.



a. Smaller Grid Sizes



b. Larger Grid Sizes

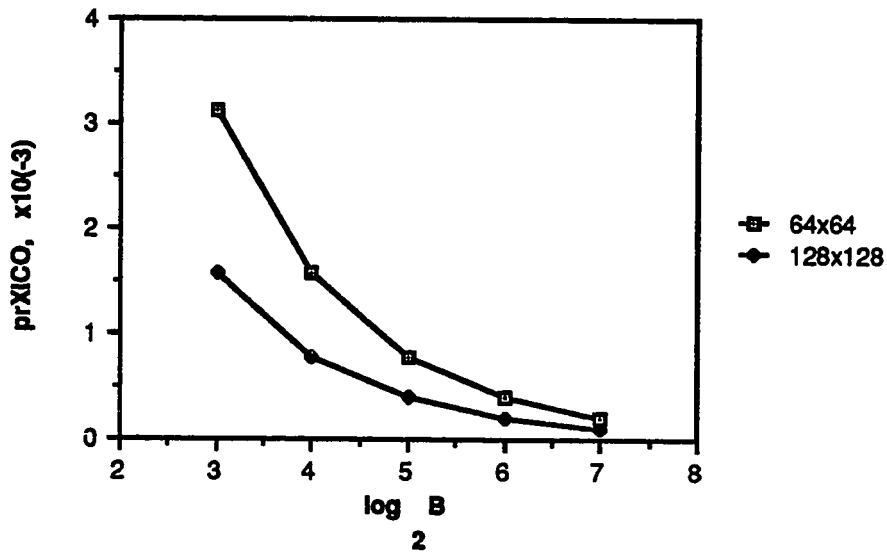
Figure 3.38 prXICO versus the Number of Processors, Set-Associative Mapping, Square Decomposition, 32-byte Blocksize.

PDE grid. The set-associative mapping structure essentially divides the sub-grid of each processor into four sections, enabling the same type of intragrid contention for both processor systems. The mapping structure and the decomposition strategy allows twice as many XICOs for the 8 processor system; however, the total number of references are double the number for the 16 processor system.

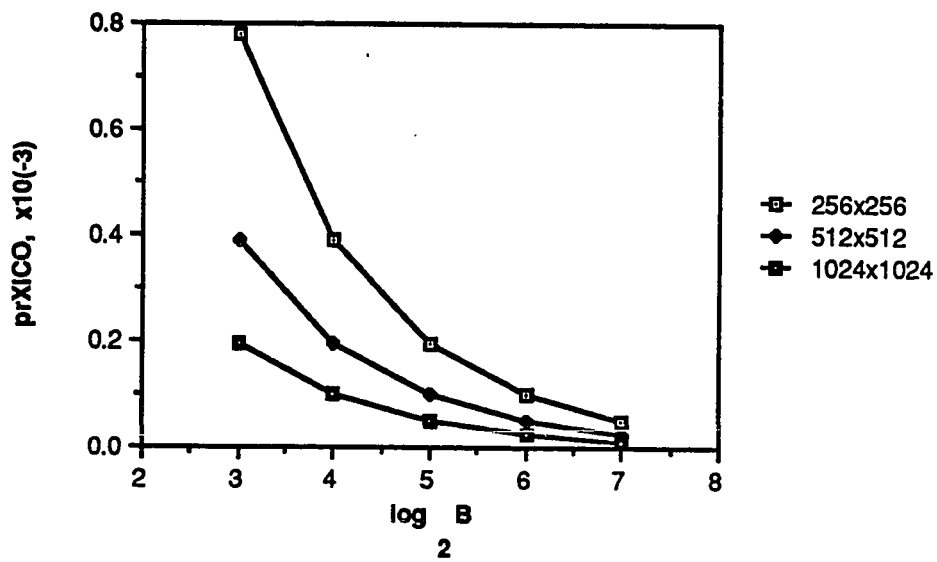
Figures 3.39 through 3.42 show the prXICO versus blocksize and PDE grid size for set-associative mapping and rectangular decomposition of the grid for 2-16 processors. For all processors, the value decreases as the blocksize increases. Also note the rate of decrease is faster for this partition than for the square decomposition. The prXICO increases as the number of processors increase for a given blocksize as shown in Figure 3.43. The 64kbyte cache value for the 128x128 point grid (4 processors) is slightly larger than the 32Kbyte cache value. This is also true for the 256x256 point grid using 8 and 16 processor systems. While the number of blocks that can be placed in the cache increases with an increase in the cache size, the mapping strategy is such that the prXICO is effected only by these 128x128 and 256x256 point grid sizes.

3.2.4. Cross-interrogate Change State

As outlined earlier, both EX->ROs and RO->EXs are known as XICs. RO->EX only occur as a result of directly replacing a RO block. EX->ROs occur only as a result of indirect block replacement. This latter fact is counterintuitive. Consider a grid small enough so that all blocks of both U and V copies can be placed in the cache simultaneously, i.e. the 64x64 point grid size. Under steady state execution

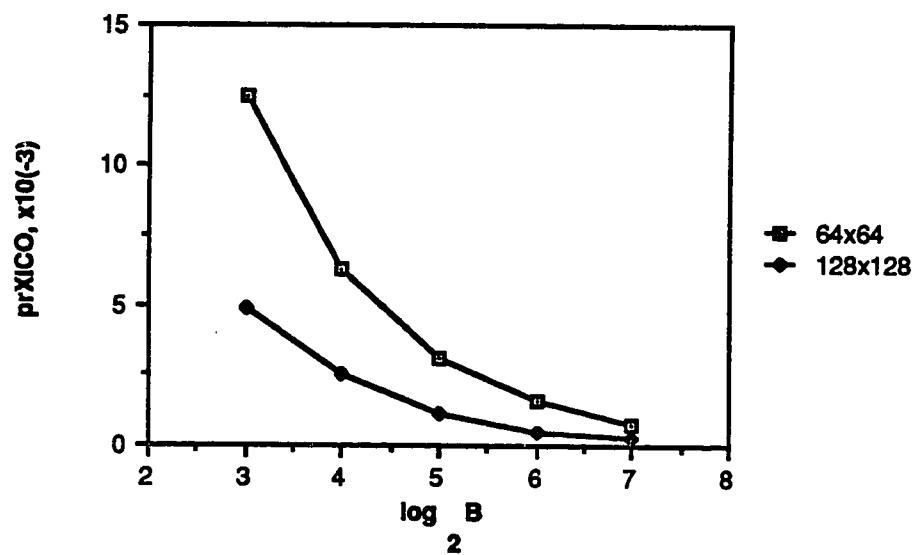


a. Smaller Grid Sizes

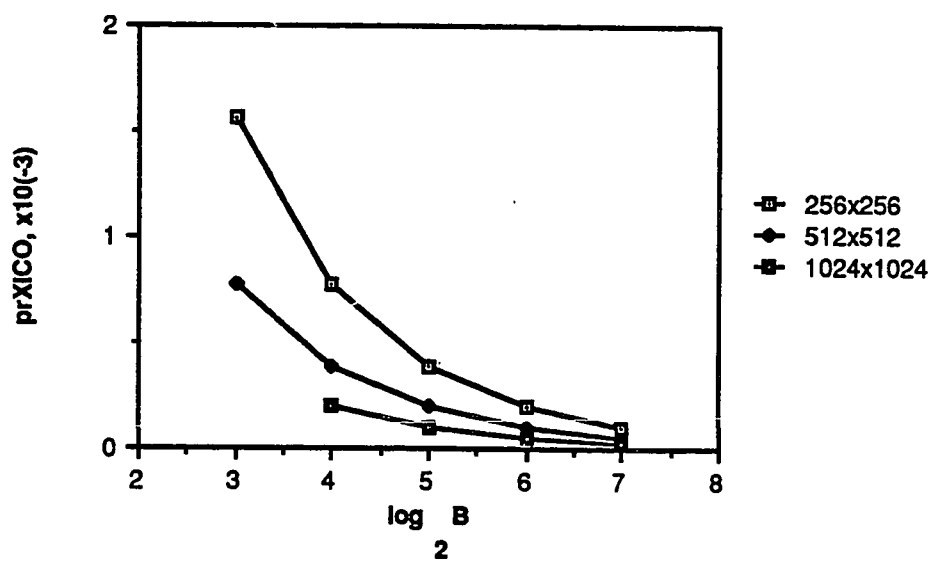


a. Larger Grid Sizes

Figure 3.39 prXICO versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Two Processors.

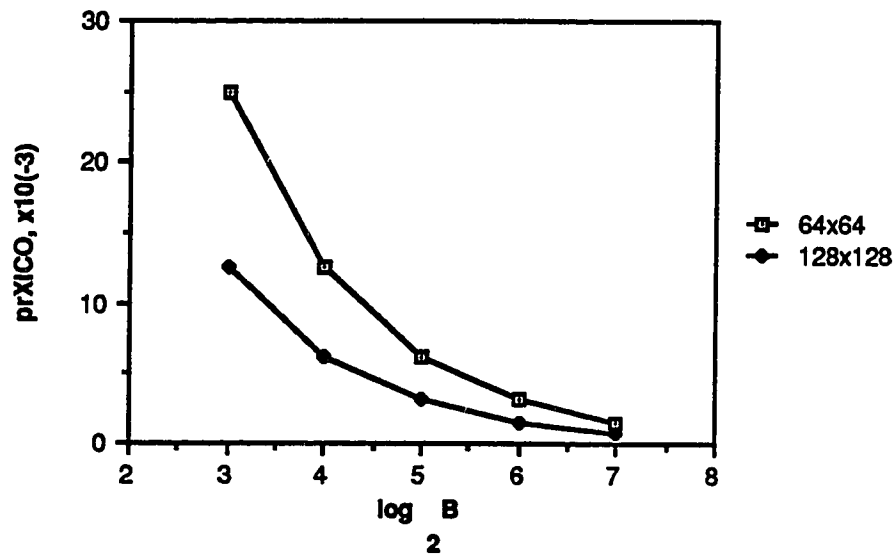


a. Smaller Grid Sizes

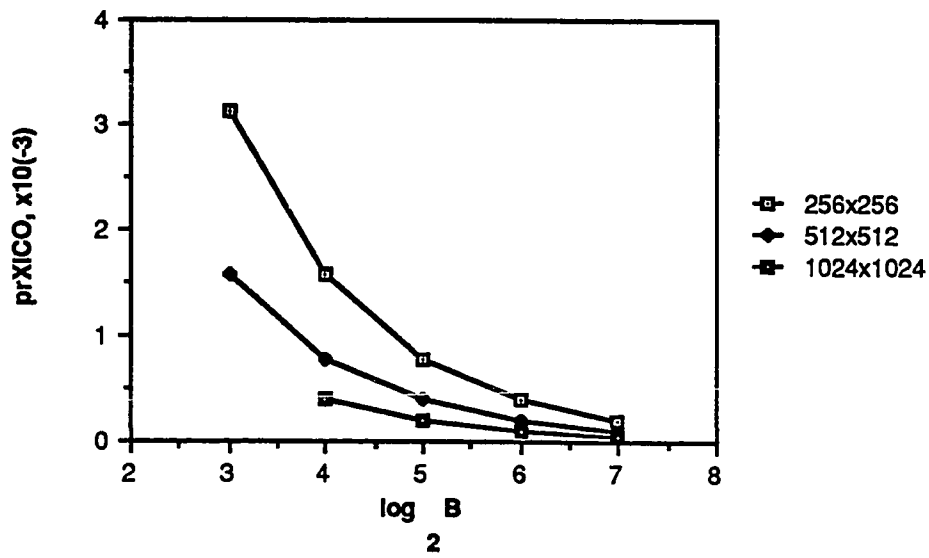


b. Larger Grid Sizes

Figure 3.40 prXICO versus Cache Blocks size, Set-Associative Mapping, Rectangular Decomposition, Four Processors.

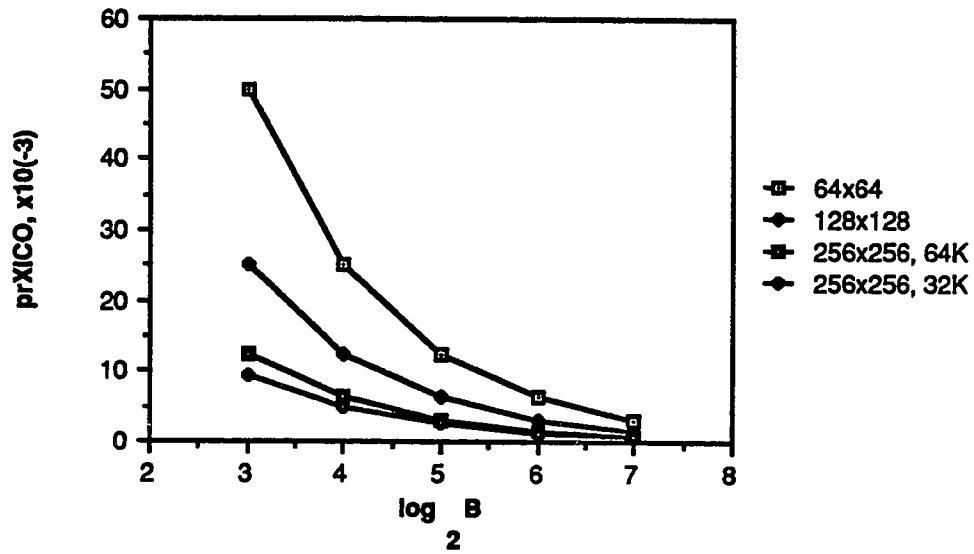


a. Smaller Grid Sizes

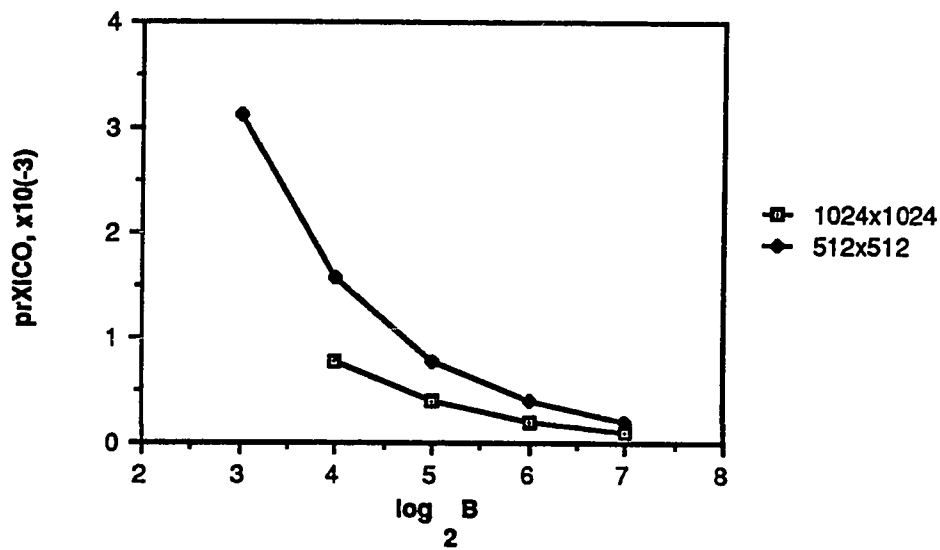


b. Larger Grid Sizes

Figure 3.41 prXICO versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Eight Processors.

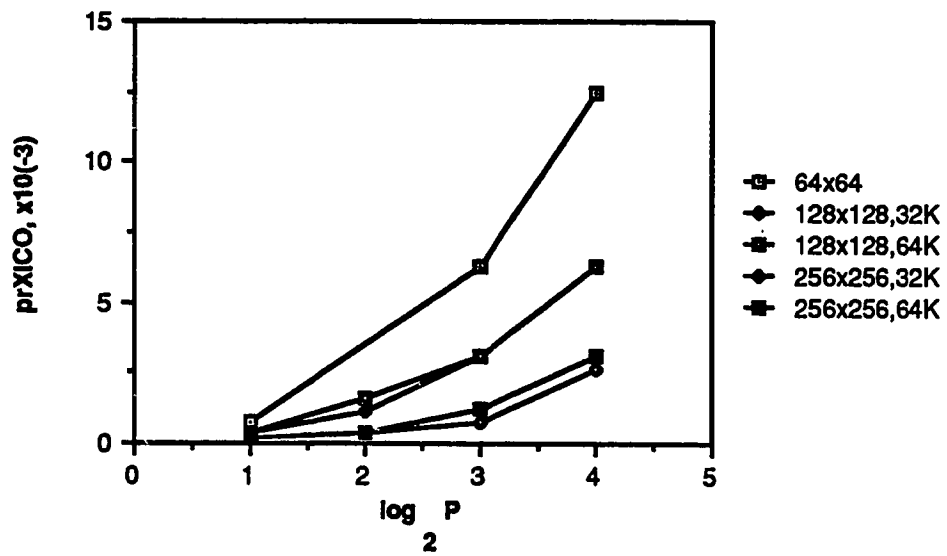


a. Smaller Grid Sizes

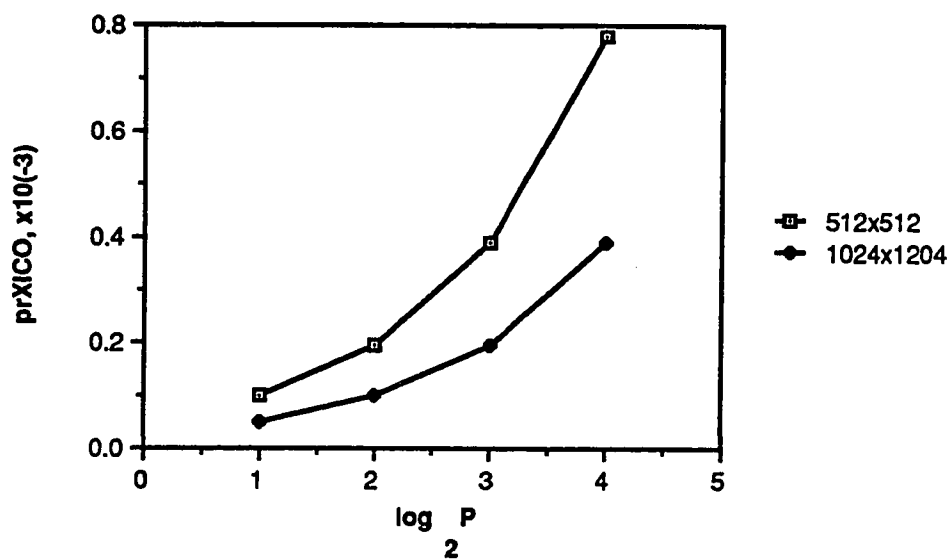


b. Larger Grid Sizes

Figure 3.42 prXICO versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Sixteen Processors.



a. Smaller Grid Sizes



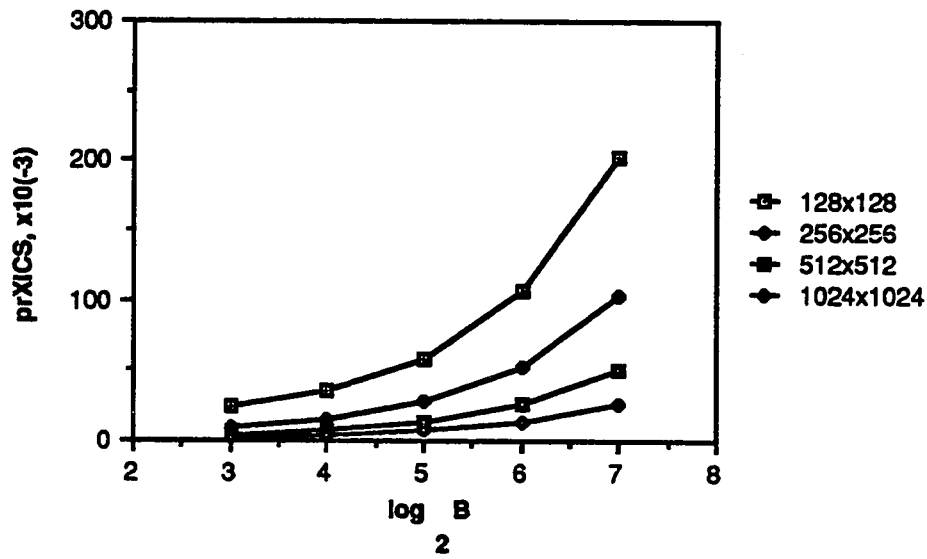
b. Larger Grid Sizes

Figure 3.43 prXICO versus the Number of Processors, Set-Associative Mapping, Rectangular Decomposition, 32-byte Blocksize.

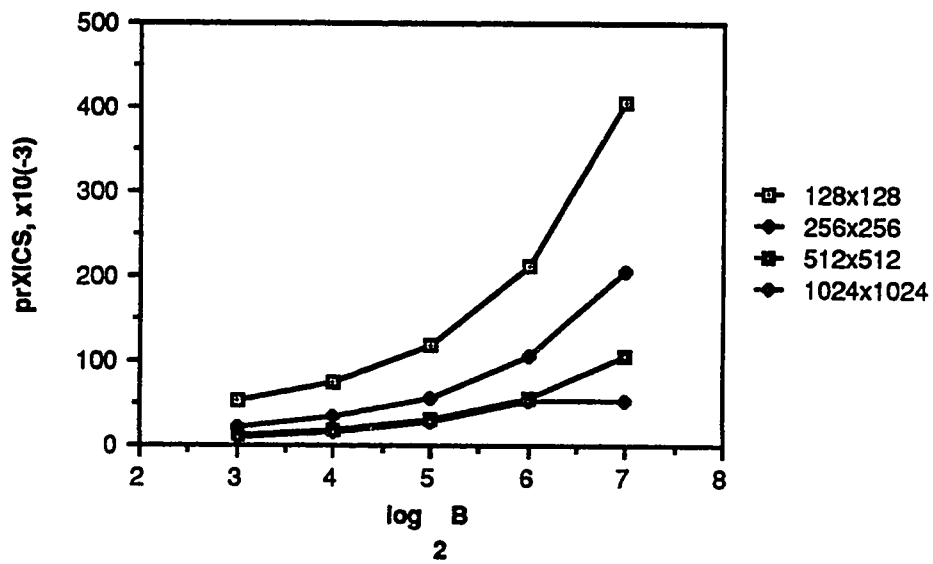
conditions (or a warm start cache) all blocks for both grids are tagged either RO or RW. In fact, for jacobi's algorithm, a block is tagged as EX only during the transient cold start program execution, when a remote processor executes a RO-EX on the block or when a processor attempts to read a block that has been replaced and is not present in any remote cache. Since the simulations assumes a warm start cache, only block replacements cause XICSs. Therefore, if all blocks map into unique cache block frames, the prXICS is zero. For the 64x64 point grid size this is in fact true.

The prXICS versus the cache blocksize is shown in Figure 3.44 for 2-16 processors, direct mapping and square decomposition of the PDE grid. This figure shows an increase in the prXICS as the blocksize increases. An explanation for this behavior is illustrated in Figure 3.45. This figure assumes a 4x8 point grid size and a 2-point blocksize. During the execution of the $U=V$ iteration, the modification of two of the points composing the right-most block allocated to P0 is shown. The figure also illustrates how the right-most blocks of P0 and the left-most blocks of P1 are shared. The accesses resulting in XICSs when modifying the right-most block of the second row of the sub-grid assigned to P0 are itemized chronologically. In the following discussion, use of the terms access or modify the V or U blocks is interpreted to mean to access or modify the PDE grid point located in the V or U block, respectively. Also, C0 and C1 means the private cache of P0 and P1, respectively.

The first XICS occurs when the V block directly north of the point to be updated is accessed. The block presently mapped into the cache frame this V block maps to is the geometrically equivalent U block. This U block was mapped into the cache as a result of a previous modification of grid point a . Since only one block can map to this

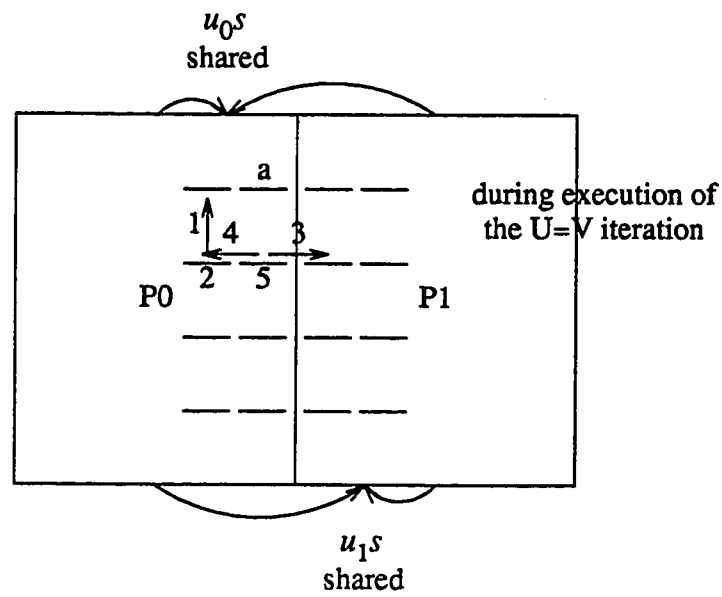


a. Two, Four and Eight Processors



b. Sixteen Processors

Figure 3.44 prXICS versus Cache Blocksize, Direct Mapping, Square Decomposition.



processor P0 performs the following XICSs in the cache of P1.

1. EX->RO
2. Modification of block results in RO->EX
3. EX->RO
4. EX->RO
5. Modification of block results in RO->EX

Figure 3.45 XICSs on a Direct Mapped Cache using a Square Grid Decomposition.

frame at a time (direct mapping) the V block replaces the U block. Since this same V block is already present as EX in C1, P0 executes a EX->RO in C1. The second XICS occurs when the first point in U of the block considered is updated in C0. The U

block replaces the V block already present as RO in C0. Since this V block is shared with C1, P0 executes a RO->EX in C1 as V is replaced in C0. The U block is then placed in C0 and modified. The next XICS occurs as the V grid point directly east of the second U grid point to be modified is read by P0. This grid point is located in the second row of the left-most V block of C1. This V block is already present as EX in C1. P0 therefore executes a EX->RO in C1 as the block is placed in C0.

Another XICS occurs as the V grid point west of the U grid point to be modified is read. This V grid will replace the U grid present C0 as a result of the last modified U point. Since this V is already present as EX in C1, P0 executes a EX->RO in C1 as the V grid is placed in C0. The final XICS resulting from modifying the block studied here, occurs when the last U grid point is modified. The U block replaces the geometrically equivalent V block already present RO in C0. Replacing this block results in P0 executing a RO->EX on the same V block in C1. The example presented here shows that the number of XICSs is directly related to the blocksize. In fact as the blocksize increases, the number of XICSs also increases. This explains the behavior for all the processors as illustrated in Figure 3.44. This figure shows a slight increase in the prXICO for the 16 processor system. This is because the square decomposition of the grid for this processor configuration results in the sharing of blocks in both the left-most and right-most column of some sub-grids.

Figure 3.46 shows the prXICS versus the number of processors for direct mapping and square grid decomposition. The 128x128 point grid shows slight increases as the number of processors increase from 2 to 8. A relatively dramatic increase is illustrated as the processor configuration increases to 16. All other grid sizes show a

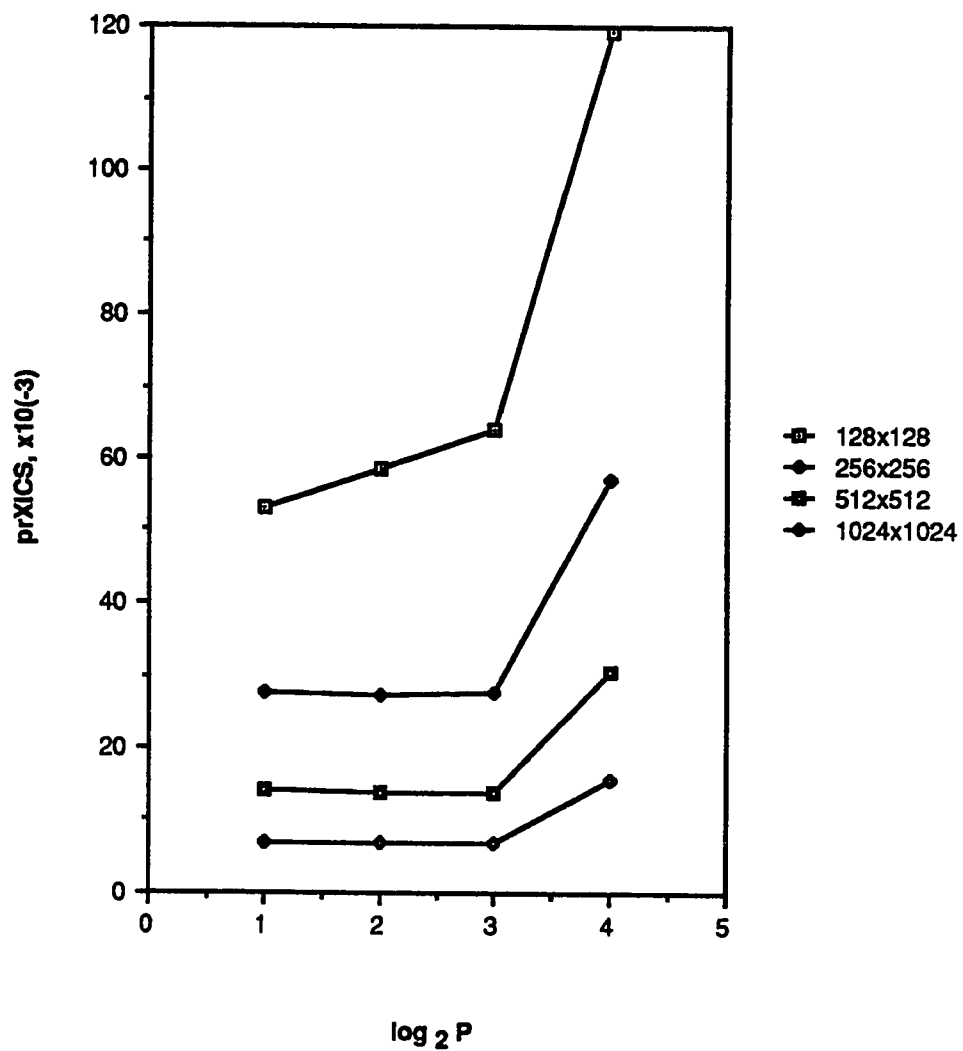
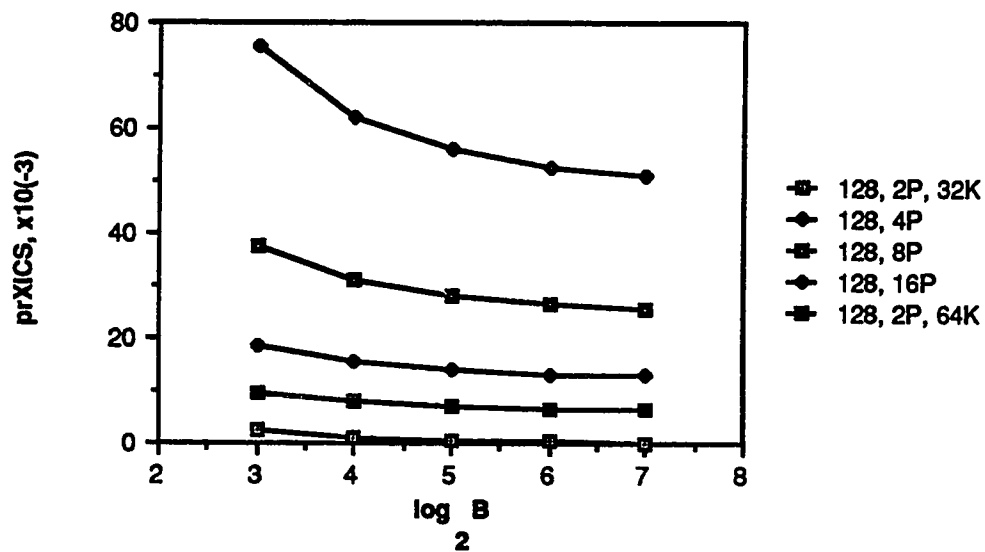


Figure 3.46 prXICS versus the Number of Processors, Direct Mapping, Square Decomposition, 32Kbyte and 64Kbyte Cache Sizes.

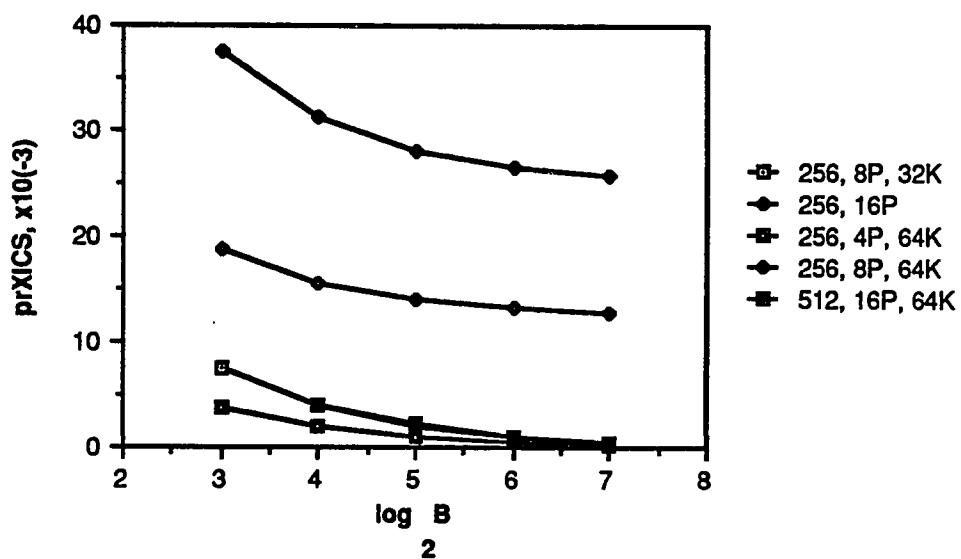
constant probability as the number of processors increase from 2 to 8 and a somewhat dramatic increase for the 16 processor system. The explanation for the 16 processor increase is given above. The 2 to 8 processor increase is the result of the fact that the 128x128 point grid is so small that intragrid contention does not effect the XICSs for the shared blocks along the horizontal sub-grid borders (this will be explained in detail below).

Figure 3.47 illustrates the prXICS versus the cache blocksize with direct mapping for the rectangular grid decomposition. This figure shows a decrease in this parameter as the blocksize increases, a direct contrast to the square partition. Also note the smaller values for this decomposition and generally the smaller grid sizes shown. The reasons for these drastic differences are illustrated in Figure 3.48. This figure assumes a 2 processor configuration, a 128x128 point grid size, a 2-point blocksize (a grid point is assumed to be 4 bytes) and a 32Kbyte cache. With these assumptions, the grid is divided so that the blocks composing the first row of the sub-grid allocated to P0 and the first row of the sub-grid allocated to P1 map to the same cache block frame as shown in the figure. This is also true for the blocks composing the last row of P0 and P1. At the end of a V=U iteration, the blocks composing the first row of P1 are the only ones shared by C0 and C1. Intragrid contention prevents the sharing of the blocks composing the last row of P0.

During the execution of the U=V iteration, the first XICS occurs when either P0 or P1 (the trace generator randomly chooses one of these references first) reads the V grid point at the head of arrow 1 shown in the figure. If P0 is chosen first, then the V block referenced replaces the shared U block in C0. P0 therefore executes a RO-EX

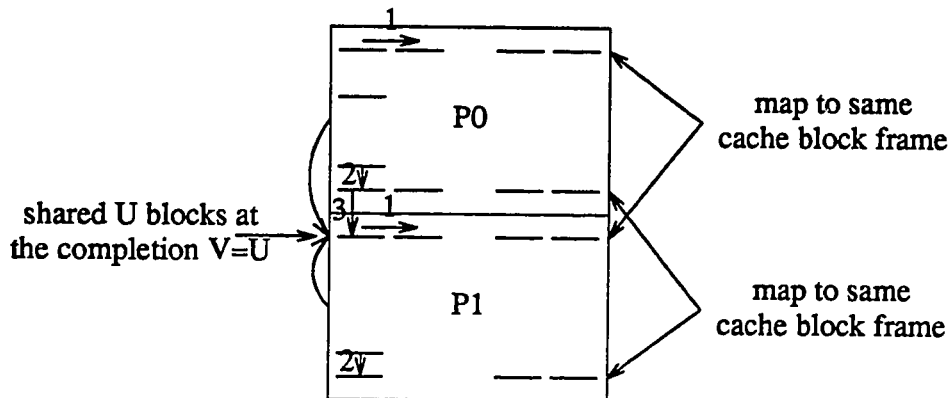


a. Smaller Grid Sizes



b. Larger Grid Sizes

Figure 3.47 prXICS versus Cache Blocksize, Direct Mapping, Rectangular Decomposition, All Processor Configurations.



some XICSs performed during the $U=V$ iteration

1. P0 or P1 executes a RO-EX on C1 or C0 respectively
2. P0 executes a EX- \rightarrow RO on C1 if P1 does not reference the block indicated first
3. P0 executes a EX- \rightarrow RO on C1

Figure 3.48 XICSs on a Direct Mapped Cache, Rectangular Decomposition.

in C1. Likewise, P1 performs a RO-EX in C0 if chosen first. This behavior occurs every time a V grid point in a new block along the first row of each sub-grid is read by P0 or P1. Another XICS may occur when P0 references a V grid point as shown in 2 of the figure. This V grid point is already present as EX in C1. Therefore, if the randomizer chooses the P0 reference before the corresponding P1 reference (for each new V block reference along the last row of each sub-grid), this processor will execute a EX- \rightarrow RO in C1. Finally, other XICSs occur when P0 reads the V grid points shown in 3 of the figure. Since these grid points are already present as EX in C1, P0 performs a EX- \rightarrow RO in C1 each time a new V block along this row is referenced.

All of these XICSs are directly related to the cache blocksize. Since these XICSs occur only during the first reference to a block, this explains the decrease in the parameter value as the blocksize increases. Also, for this partition, modifying blocks do not cause any XICSs. This partition, therefore, significantly reduces the values of the prXICS relative to the square partition. Figure 3.47 shows an increase in the prXICS as the cachesize increases from 32Kbytes to 64Kbytes for 2 processors and the 128x128 point grid size. This is because the increased cachesize allows the blocks composing the last row of P0 (Figure 3.48) to be shared causing an increase in the number of XICSs per iteration. As the grid size increases intragrid contention limits the number of shared blocks between processors. This in turn reduces the prXICS. For grid sizes not shown in the figure the prXICS is zero.

Figure 3.49 presents the prXICS versus the number of processors for direct mapping on a rectangular grid decomposition. The figure shows a general increase in this value as the number of processors increase. For 8 or fewer processors, this value is negligible for the 256x256 (32Kbyte cache) and 512x512 (64Kbyte cache) point grid sizes. This also holds for the 128x128 point grid with a 32Kbyte cache and a 2 processor system. The 16 processor system with a 64Kbyte cachesize is the only configuration where the 512x512 point grid has XICSs. This is because the decomposition strategy and the mapping function are such that shared blocks occur.

Figure 3.50 shows the prXICS versus blocksize for the set-associative mapping function, the square decomposition strategy, and 2 (a) and 4 (b) processors. Part a of this figure shows a constant value for each grid size. This is because the XICSs caused by replacing blocks as a result of intergrid contention (see Figure 3.45) is

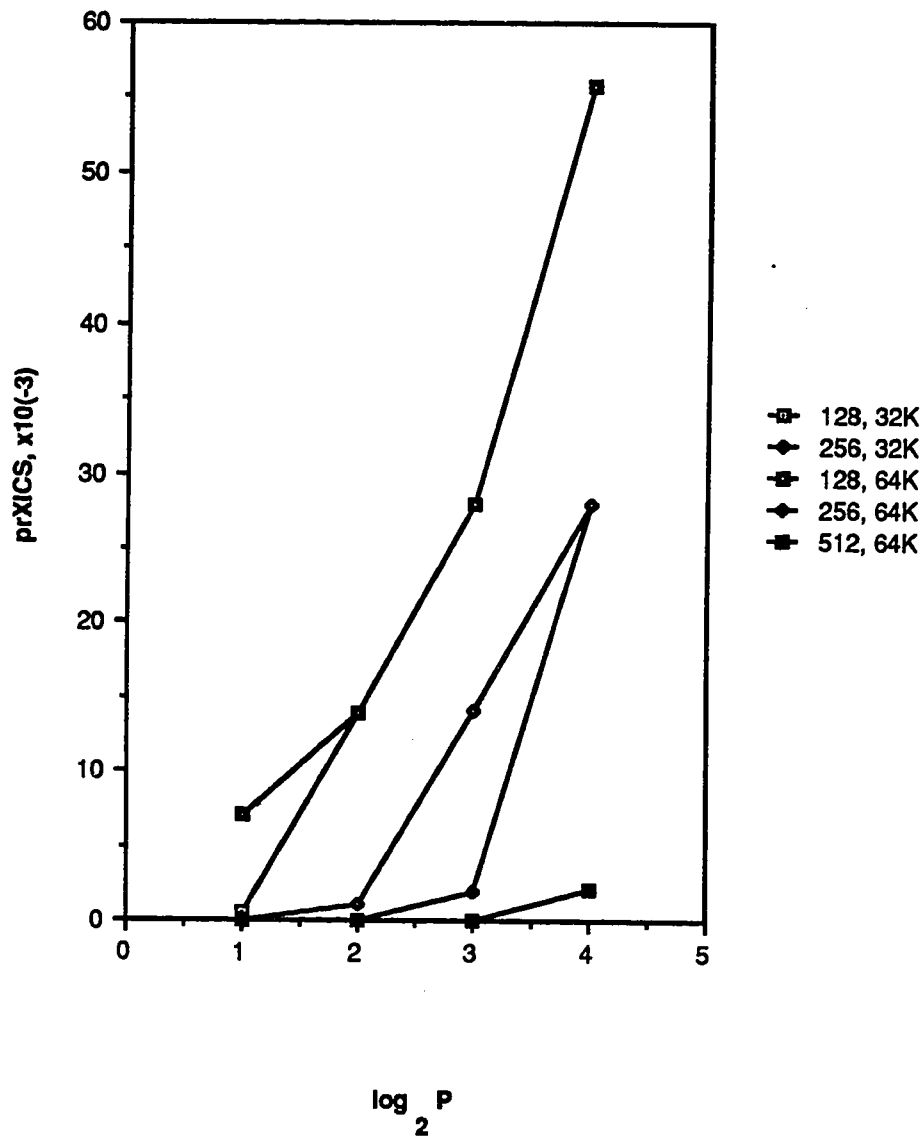
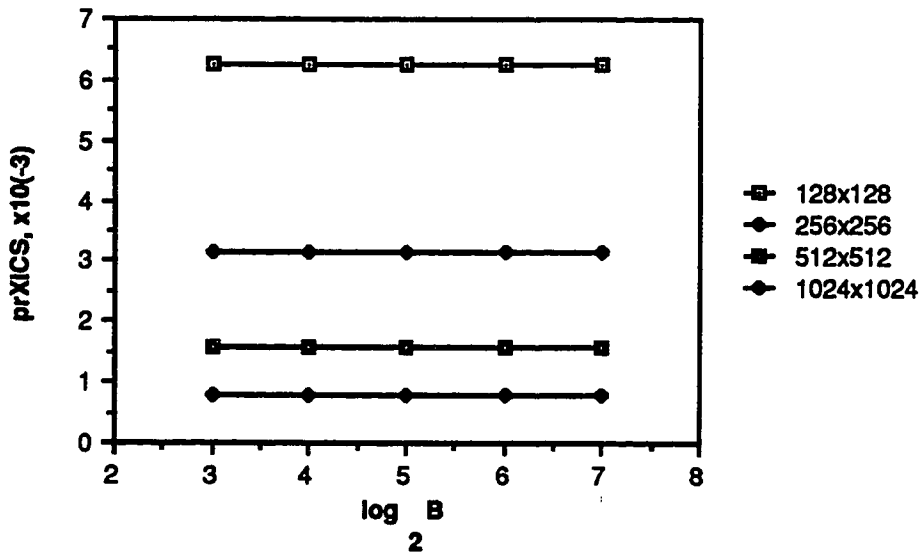
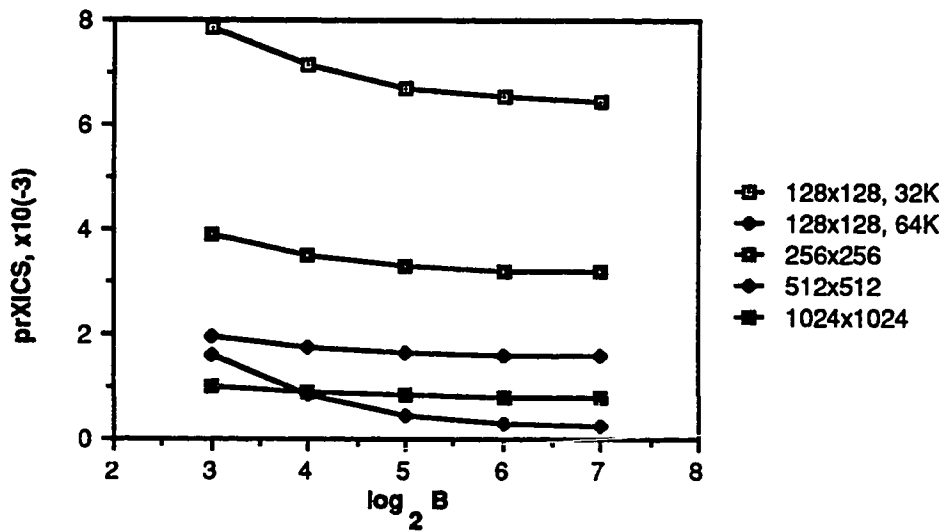


Figure 3.49 prXICS versus the Number of Processors, Direct Mapping, Rectangular Decomposition.



a. Two Processors



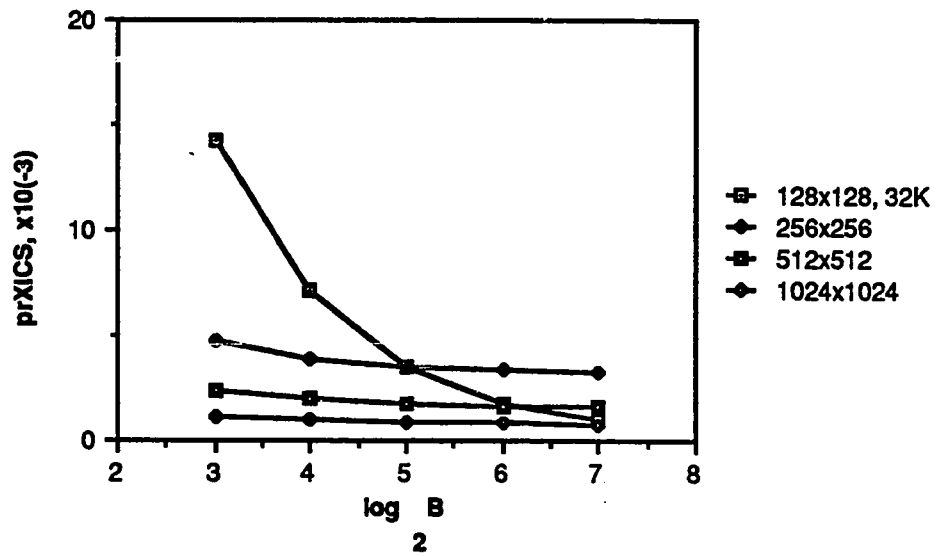
b. Four Processors

Figure 3.50 prXICS versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Two and Four Processors.

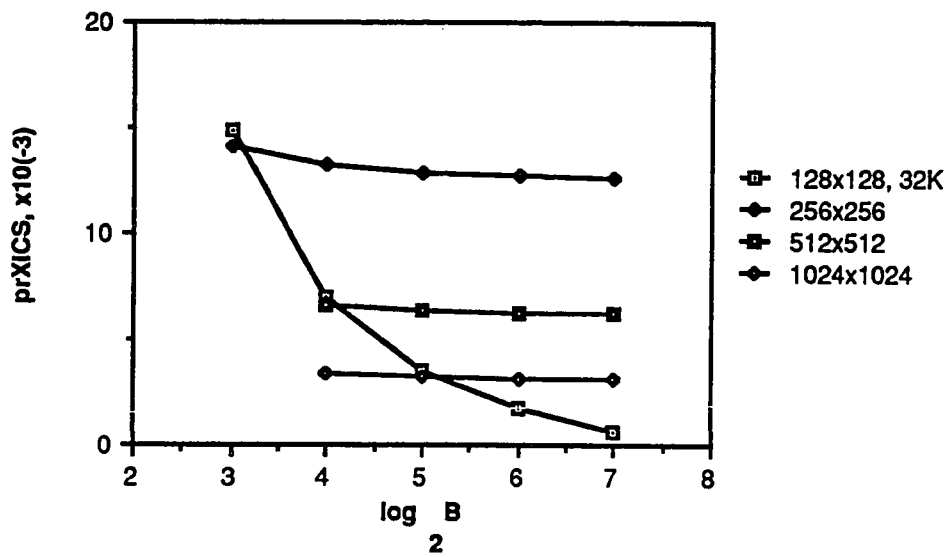
eliminated by the 2-way set-associative mapping function. The 2 processor system, therefore, results in a constant number of XICSs for each blocksize as a result of the constant number of blocks adjacent to the vertical sub-grid border. The 4 processor system shows a decrease in the prXICS as the blocksize increases. This is because the shared blocks along the horizontal sub-grid border result in a decrease in the number of XICSs as the blocksize increases as explained above.

Figure 3.51 shows this same prXICS for 8 (a) and 16 (b) processors. Again, the value decreases as the blocksize increases. The 128x128 point grid size (32Kbyte cache only) shows as comparatively significant decrease in the prXICS as the blocksize increases. This results from considerable sharing between shared blocks along the horizontal sub-grid. This sharing occurs as a result of the reduction in the intragrid contention for this grid size. The prXICS is zero for the 128x128 point grid and a 64Kbyte cache because all blocks map into unique cache block frames resulting in no block replacement.

Figure 3.52 shows the prXICS versus the number of processors for set-associative mapping and the square grid decomposition. For grid sizes larger than 128x128 points the prXICS remains constant for 2-8 processors and then increases considerably for 16 processors. This behavior is identical to the demand fetching curves for this parameter. The reason is also the same, i.e., the 16 processor system has two sets of shared blocks along the vertical sub-grid borders (for inner processors). This results in an increase in the number of XICSs per processor. The 128x128 point grid size prXICS decreases for 8 and 16 processors (32Kbyte cache size) and for 4-16 processor systems (64Kbyte cache). This is because more shared blocks



a. Eight Processors



b. Sixteen Processors

Figure 3.51 prXICS versus Cache Blocksize, Set-Associative Mapping, Square Decomposition, Eight and Sixteen Processors.

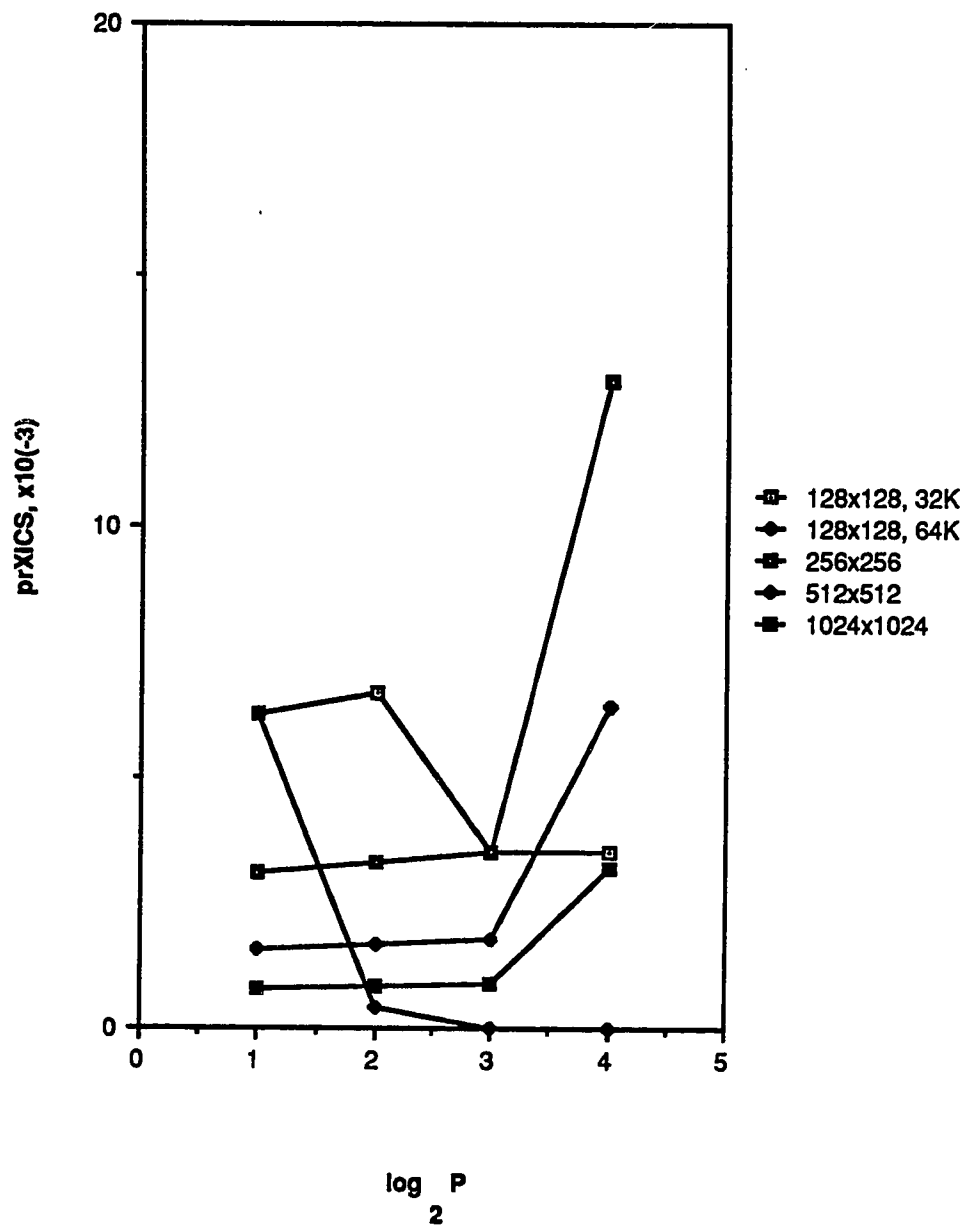


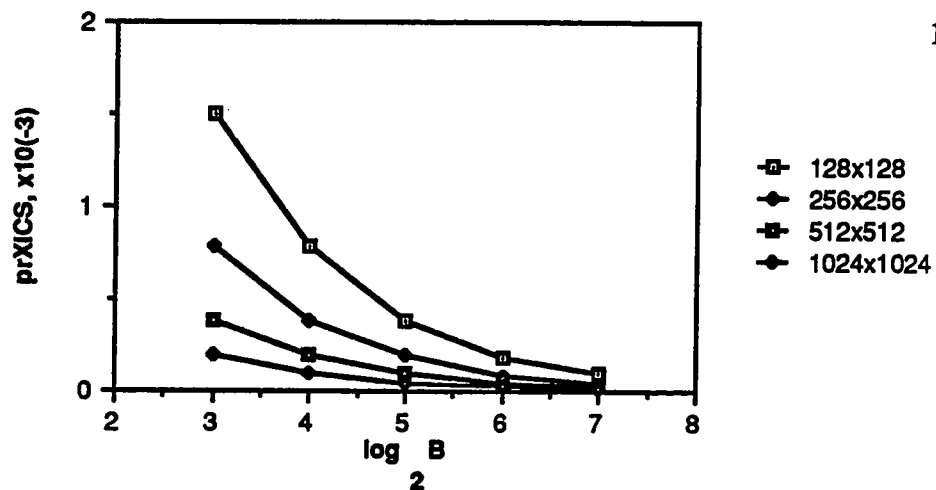
Figure 3.52 prXICS versus the Number of Processors, Set-Associative Mapping, Square Decomposition.

from the horizontal sub-grid borders can be mapped into the cache. This may cause an increase in the number of XICS, but eventually a point is reached where the number of replaced blocks decrease causing a decrease in the number of XICSs. Figure 3.52 shows this behavior for the 128x128 point grid (both cache sizes).

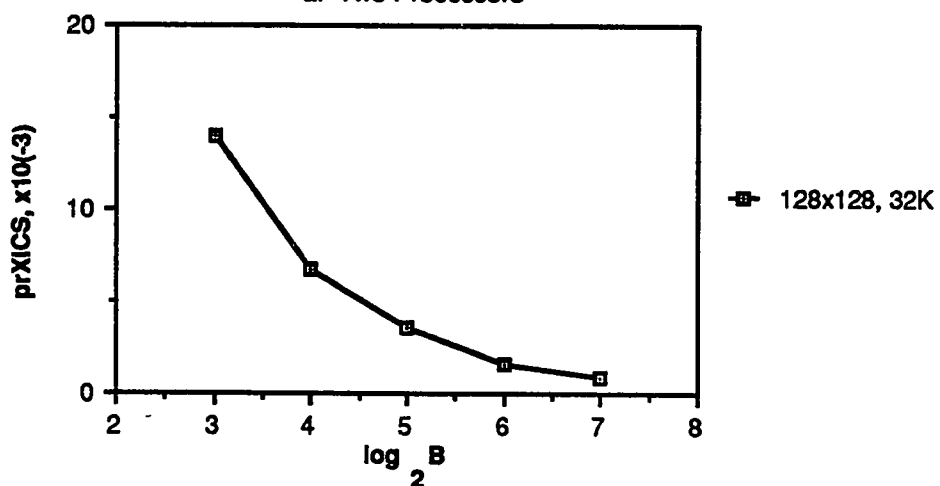
Figure 3.53 illustrates the prXICS versus the blocksize for set-associative mapping, the rectangular decomposition strategy and 2 and 4 processors. Once again, this rectangular partition is significantly lower than its square decomposition counterpart. Also, the value decreases as the blocksize increases. Figure 3.54 presents this same parameter for 8 (a) and 16 (b) processors. Figure 3.55 shows the prXICS versus the number of processors for the rectangular grid decomposition and set-associative mapping. All grid sizes larger than 128x128 show an increase in the prXICS as the number of processors increase. The size of the 128x128 grid for the various processor configurations is such that more shared blocks are present in the caches allowing more XICSs or no blocks are replaced (or all shared blocks are present) allowing no XICSs.

3.2.5. Prefetching Strategies

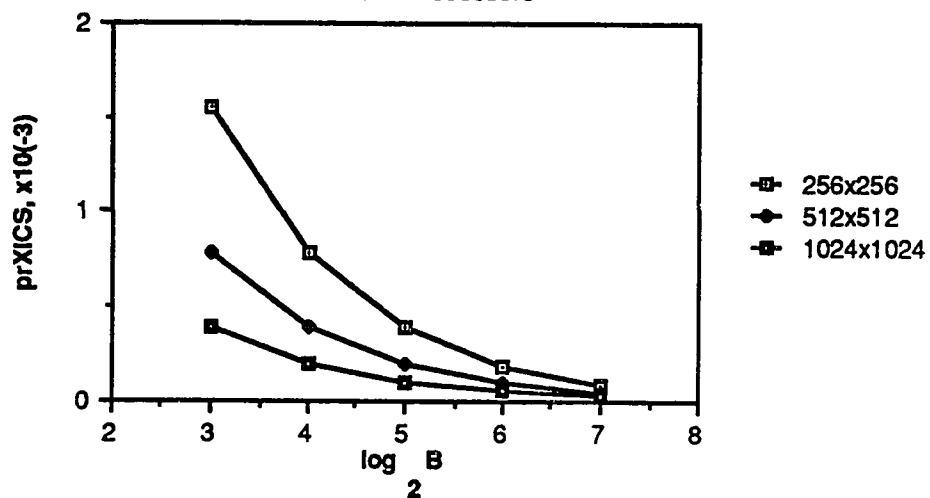
This section presents the MRs and XIs for the prefetch-on-miss and tagged prefetching strategies for Jacobi's algorithm. Additionally, the lookup ratios (LRs) and the prefetch ratios (PRs) are presented for all cache features, PDE grid sizes and decomposition strategies considered. For comparison, the demand fetch parameters are also presented.



a. Two Processors

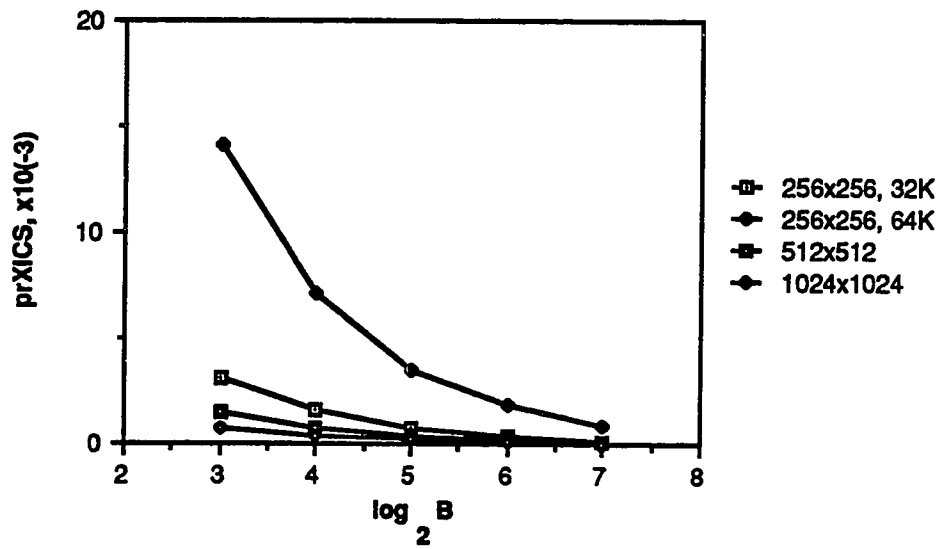


b. Four Processors

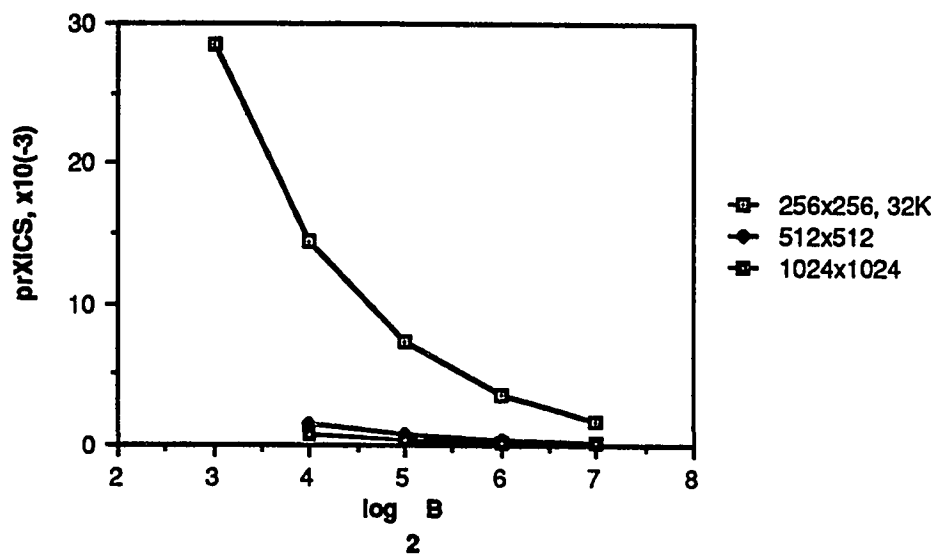


c. Four Processors

Figure 3.53 prXICS versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Two and Four Processors.



a. Eight Processors



b. Sixteen Processors

Figure 3.54 prXICS versus Cache Blocksize, Set-Associative Mapping, Rectangular Decomposition, Eight and Sixteen Processors.

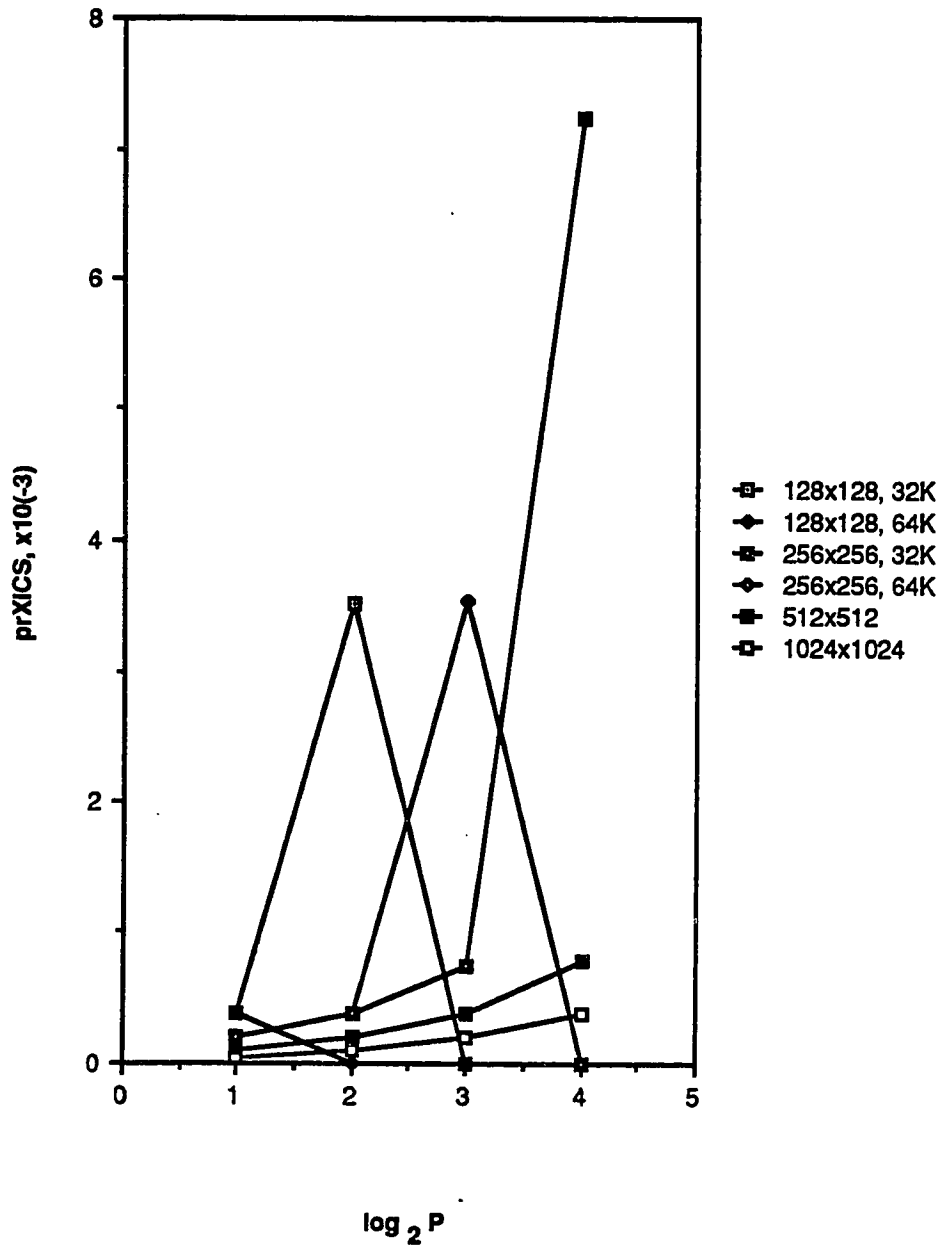
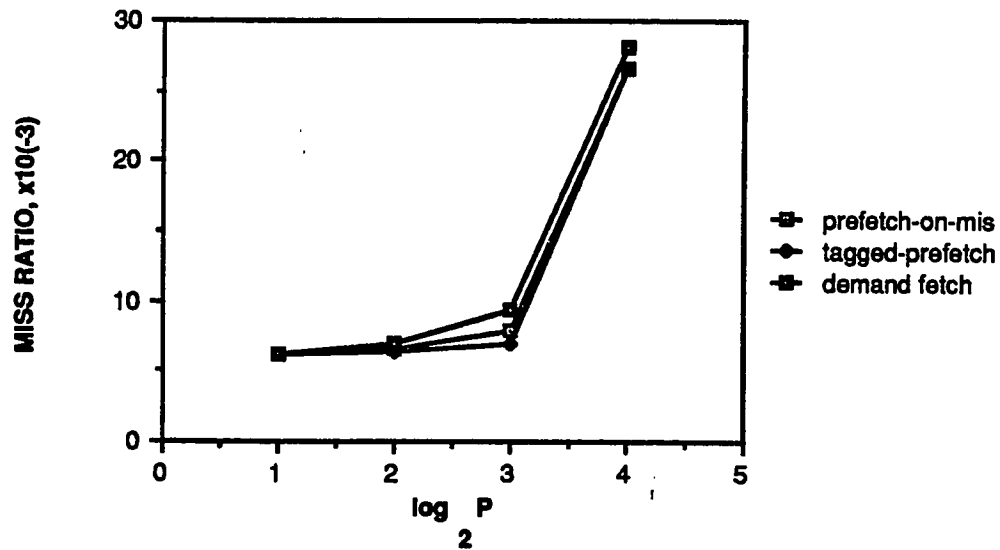


Figure 3.55 prXICS versus the Number of Processors, Set-Associative Mapping, Rectangular Decomposition, 32-byte Blocksize.

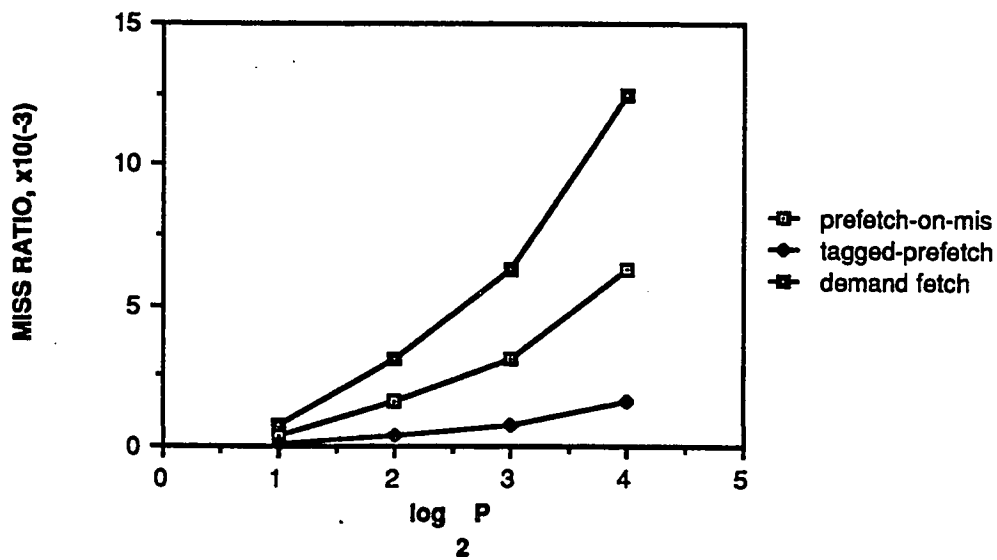
Figure 3.56 shows the MR versus the number of processors for demand fetching, prefetch-on-miss and tagged prefetching, the 64x64 point grid size and both partitions and placement algorithms. There is essentially no difference between the prefetch strategy MRs for 2 and 4 processors when using the square partition. For the 8 processor system, while the three MRs range from 0.006 to 0.009, demand fetching has the highest MR, followed by prefetch-on-miss and lastly, tagged prefetching. As the number of processors increase to 16, the tagged prefetching and prefetch-on-miss strategies are identical, with the demand fetching policy slightly higher.

While the tagged prefetching prefetch-on-miss policies exhibit slightly smaller MRs than demand fetching for larger processor configurations, the overhead penalty incurred as a result of using prefetching makes these strategies prohibitive for the square partition. Part b of this figure illustrates the prefetching MRs for the rectangular decomposition strategy. Here the prefetch-on-miss MR reduces the demand fetch MR by roughly one-half. Also, the tagged prefetching MR reduces the prefetch-on-miss MR by one-half and therefore the demand fetch MR by one-fourth. Note all MRs for this partition are significantly lower than the square partition MRs.

Figures 3.57 and 3.58 present the prefetching MRs versus the number of processors for grid sizes larger than 64x64 points and the square and rectangular decomposition strategies, respectively. The direct mapping placement strategy is used in both instances. These figures show that the intergrid contention is such that the prefetch-on-miss and tagged prefetching strategies increase the MR relative to demand fetching for all grid sizes shown. The prefetch-on-miss strategy has the worst performance, with a MR of approximately 0.46 for all grid sizes, followed by the tagged

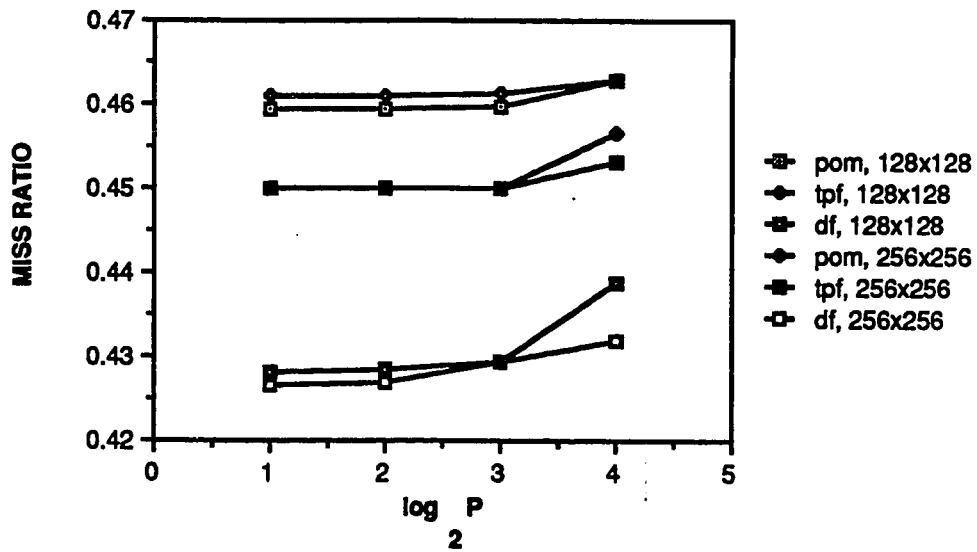


a. Square Decomposition

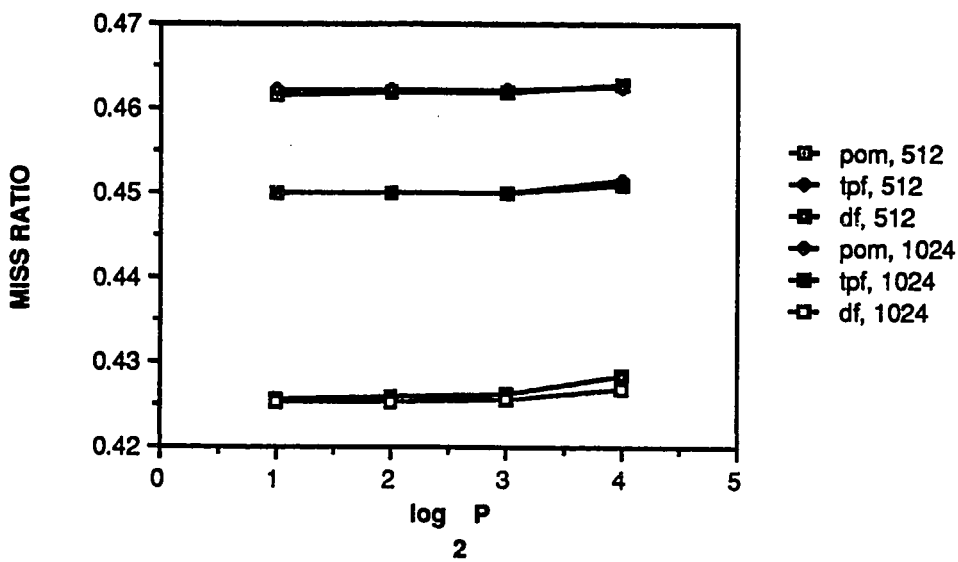


b. Rectangular Decomposition

Figure 3.56 Miss Ratio versus the Number of Processors, All Fetching Strategies, 64x64 Point Grid Size, 32-byte Blocksize.

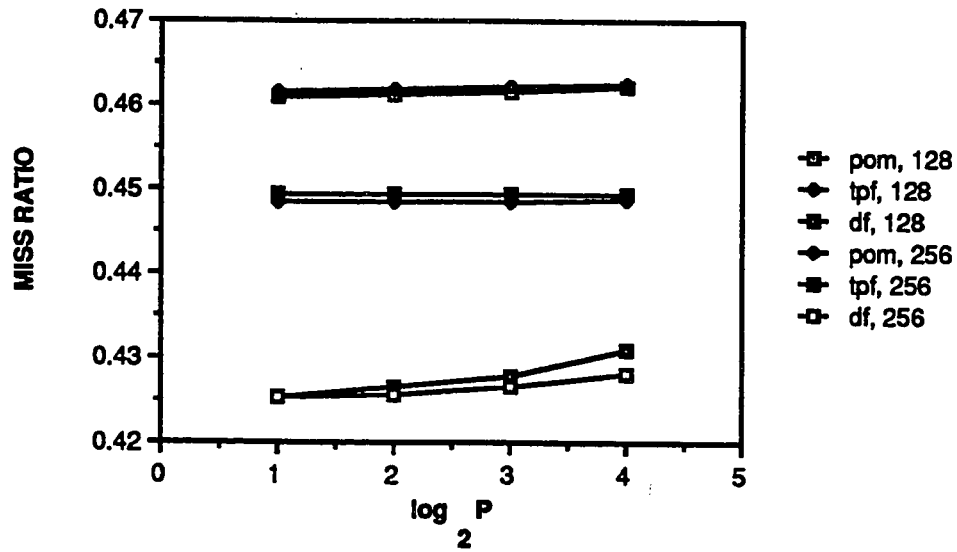


a. Smaller Grid Sizes

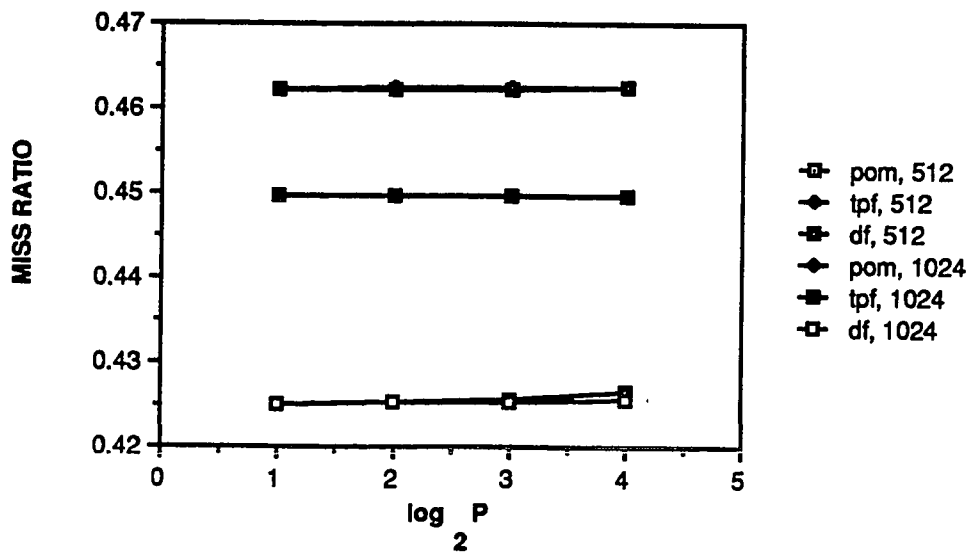


b. Larger Grid Sizes

Figure 3.57 Miss Ratio versus the Number of Processors, Direct Mapping, All Fetching Strategies, Square Decomposition, 32-byte Blocksize.



a. Smaller Grid Sizes



b. Larger Grid Sizes

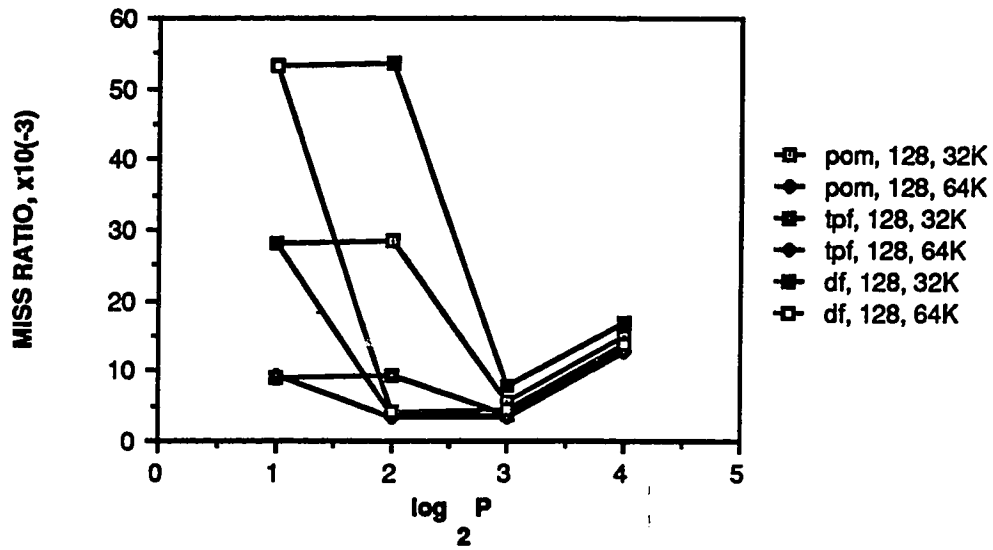
Figure 3.58 Miss Ratio versus the Number of Processors, Direct Mapping, All Fetching Strategies, Rectangular Decomposition, 32-byte Blocksize.

prefetching strategy with a MR of 0.45. Comparatively, the demand fetching policy has a MR of roughly 0.425 for all grid sizes considered.

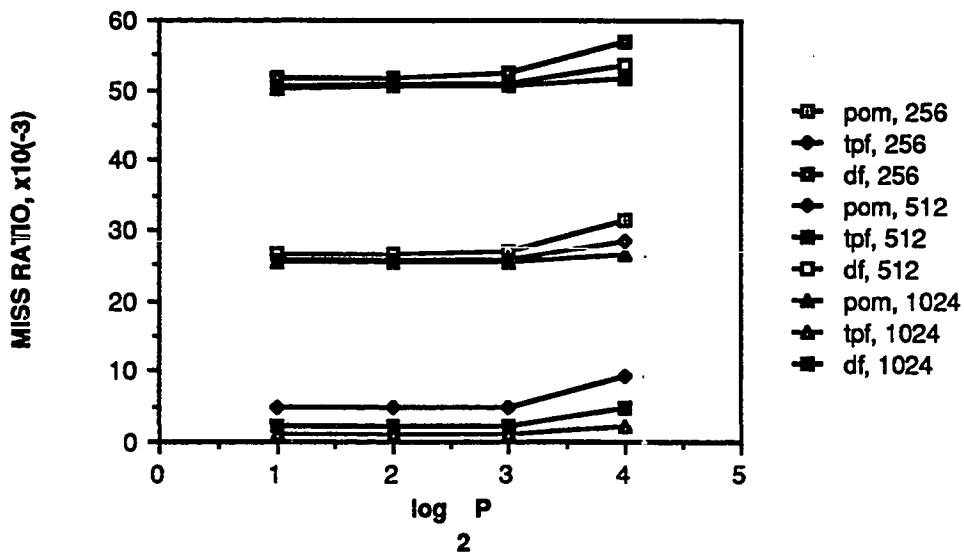
All of the blocks prefetched using the prefetch-on-miss strategy are eventually referenced; however, many blocks are referenced before these prefetched blocks. The prefetched blocks displace these referenced blocks causing additional contention and therefore increasing the MR. While the tagged prefetching strategy prefetches more blocks than prefetch-on-miss, these additional prefetched blocks serve to reduce the contention caused by prefetch-on-miss. This in turn reduces the overall MR of the algorithm.

Figures 3.59 and 3.60 show the same prefetching MRs for the 2-way set-associative mapping function. Here we see the demand fetching policy with the highest MR followed by the prefetch-on-miss strategy, and the tagged prefetching policy displaying the best performance. This is because the set-associative mapping function significantly reduces intergrid contention. This also explains the reduction in the MRs for this mapping function relative to direct mapping. Also note the reduction in the MR as the number of processors increase (16 processor system excluded) for the 128x128 point grid (both partitions) and the 256x256 point grid (rectangular partition only). This results from the presence of a MRD for these grids (see section 3.2.1).

Figure 3.61 shows the IR versus the number of processors for demand fetching and both prefetching strategies, the 64x64 point grid size and both decomposition strategies. The prefetch-on-miss and tagged prefetching IRs are relatively equal



a. Smaller Grid Sizes



b. Larger Grid Sizes

Figure 3.59 Miss Ratio versus the Number of Processors, All Fetching Strategies, Set-Associative Mapping, Square Decomposition, 32-byte Blocksize.

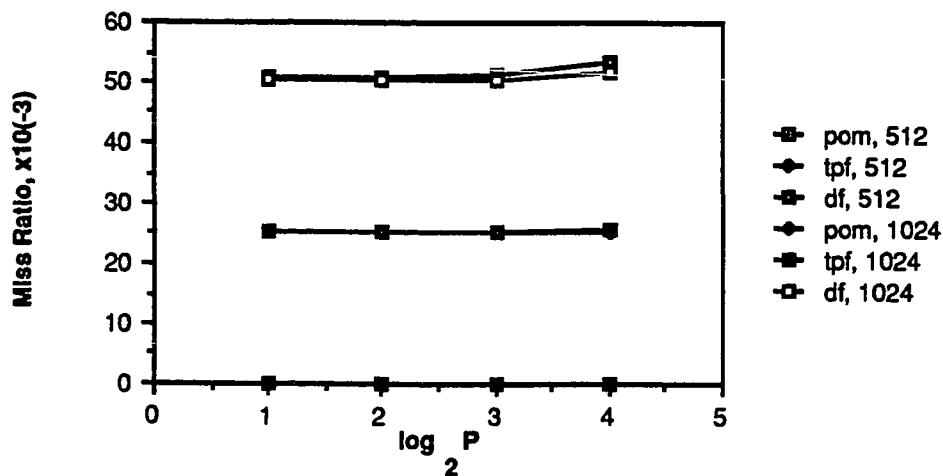
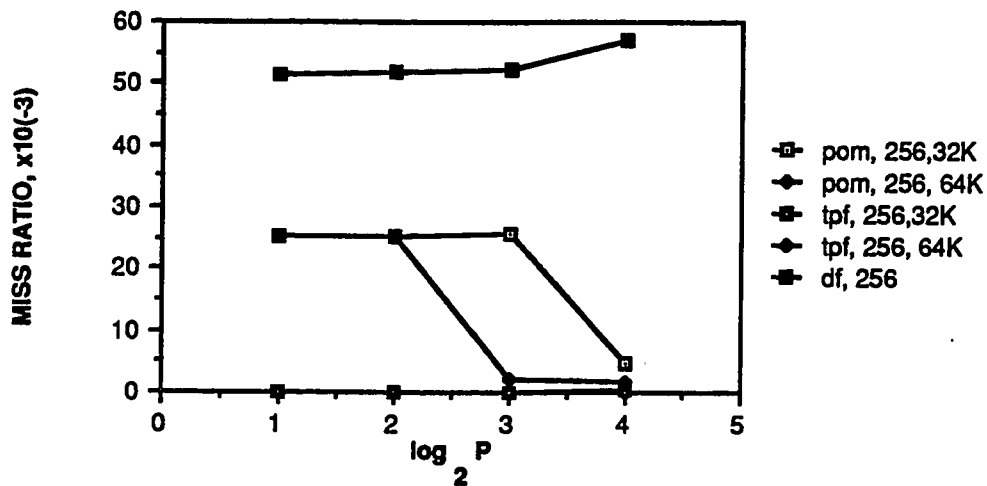
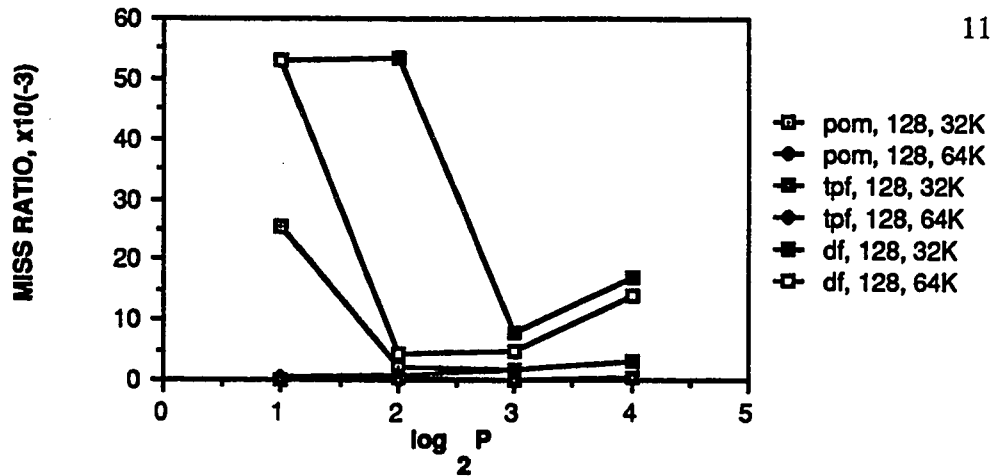


Figure 3.60 Miss Ratio versus the Number of Processors, All Fetching Strategies Set-Associative Mapping, Rectangular Decomposition, 32-byte Blocksize.

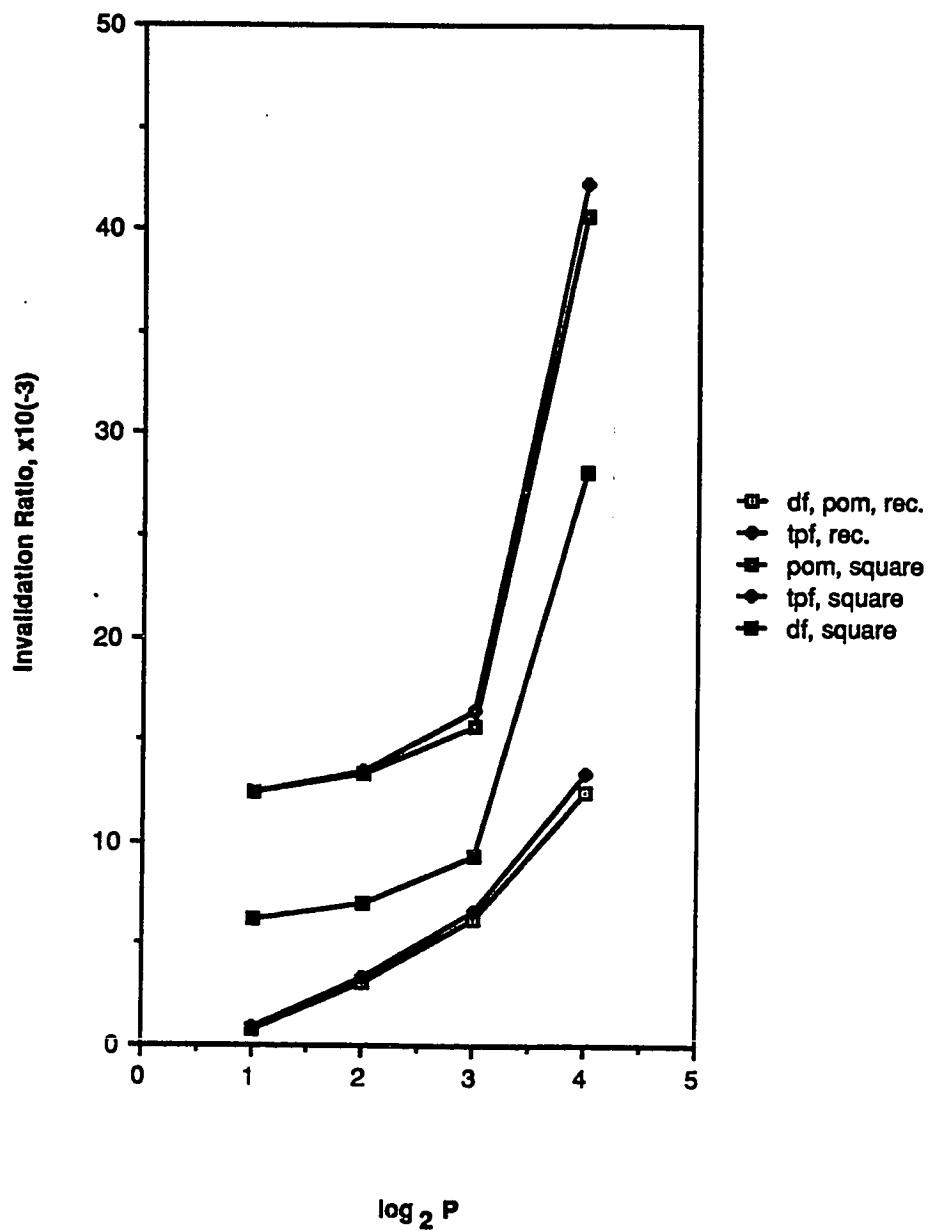
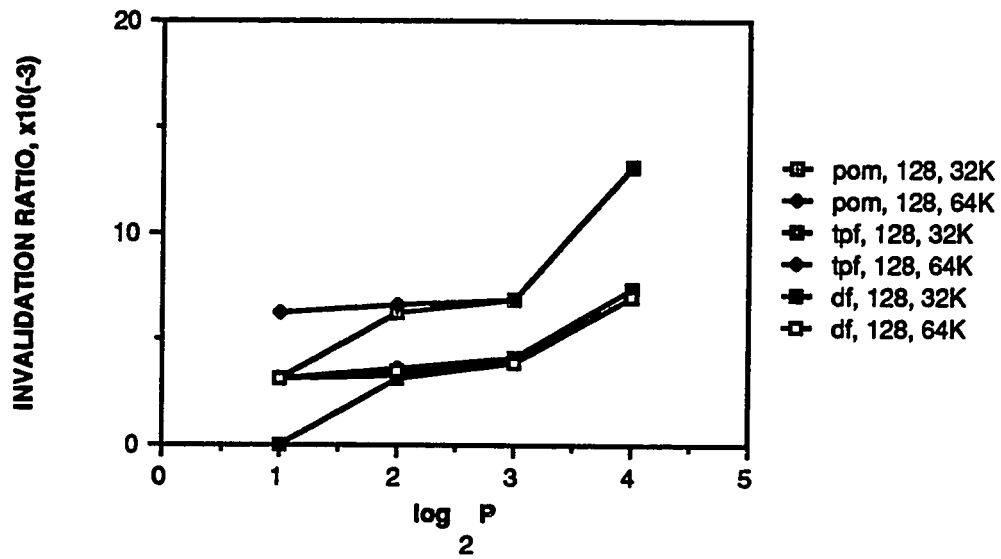


Figure 3.61 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, 64x64 Point Grid Size, 32 byte Blocksize.

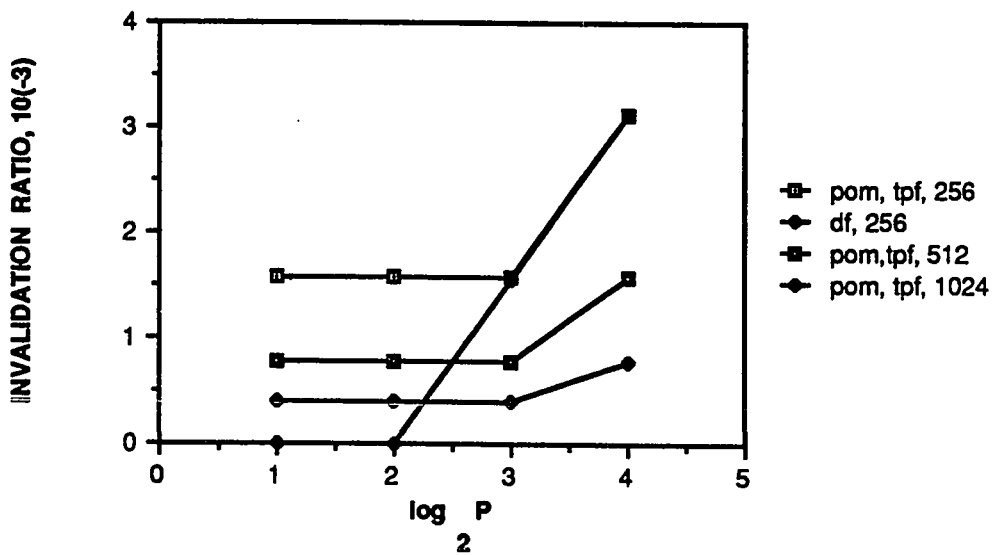
when compared with the smaller IR of the demand fetching policy (square decomposition only). Although there are slight differences between the prefetching MRs for the square decomposition of this grid size, the significant differences between the demand fetching IR and the IRs of the prefetching strategies results from the fact that the additional invalidations are caused by the prefetched blocks. For the rectangular partition, the prefetched blocks do not cause the additional invalidations as shown in the figure. In all instances, the IR increases as the number of processors increase.

Figure 3.62 illustrates the IR versus the number of processors for all prefetching strategies, all grid sizes greater than 64x64 points, direct mapping and the square decompositions. For the 128x128 point grid size, the prefetch-on-miss strategy possesses the highest IR. The tagged prefetching strategy is relatively equal to the demand fetch policy for this parameter. For the 2 processor system, demand fetching and prefetch-on-miss show an increase in the IR as the cache size increases. This results from the elimination of intragrid contention for this grid size. When using tagged prefetching, this intragrid contention elimination does effect the IR for one of the two processors. However, as the cache size is doubled, the processor effected increases its IR from 0.0 to a level equal to the 32Kbyte cache IR for the other processor. Therefore, the IR appears unchanged as the cache size increases for a 2 processor system. Figure 3.62 also illustrates an increase in the IR for the prefetching policies relative to demand fetching. The exception is the 8 and 16 processor systems of the 256x256 point grid size.

Figure 3.63 illustrates this same arrangement for the rectangular partition. Intragrid contention eliminates the IR for grid sizes larger than 256x256 points. For



a. 128x128 Point Grid Size



b. Larger Grid Sizes

Figure 3.62 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, Square Decomposition, Direct Mapping 32-byte Blocksize.

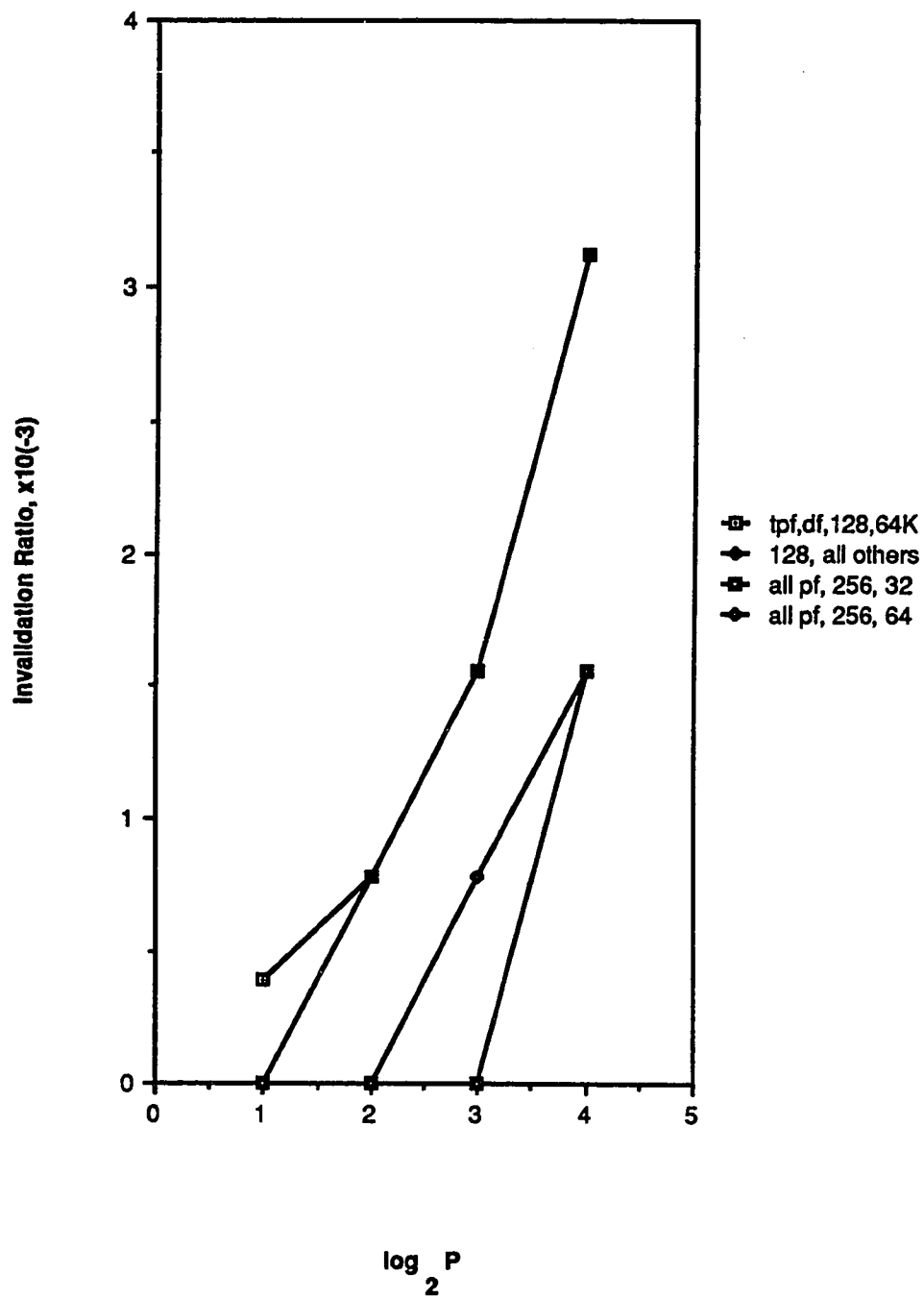


Figure 3.63 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, Rectangular Decomposition, Direct Mapping, 32-byte Blocksize.

128x128 and 256x256 point grid sizes, the prefetching strategies have no effect on the IR. Both strategies use the *one-block-lookahead* implementation and for this partition, most adjacent blocks are allocated to the same processor. This is not the case for the square decomposition, resulting in an increased IR for this partition.

Figure 3.64 presents the IR versus the number of processors for square decomposition, set-associative mapping and all prefetching strategies. For processor configurations larger than 2 and the 128x128 point grid size, there is an increase in the IR for the prefetching strategies. For the 2 processor system this also holds for the 64Kbyte cache size. The 32Kbyte cache size shows no IR for the prefetching strategies while demand fetching has a very small IR. The grid sizes larger than 128x128 points show significantly smaller IRs. The cache size appears to be the dominant factor in determining the IR for the 256x256 point grid while the tagged prefetching IR dominates the field for 512x512 and 1024x1024 point grid sizes.

Figure 3.65 illustrates this same IR for the rectangular partition. For the 128x128 point grid size, the prefetching strategies have no effect on the IR. This also holds for the 256x256 point grid size (64Kbyte cache only). For the 256x256 point grid and a 32Kbyte cache the prefetching IRs are lower than the demand fetching IR. Also note that the converse is true for grid sizes larger than 256x256 points. The major factors effecting these varying IRs are intergrid and intragrid contention.

The prXICO versus the number of processors for all prefetching strategies, cache sizes, mapping functions and partitions for the 64x64 point grid size is shown in Figure 3.66. Both prefetching strategies have a higher cast-out probability when

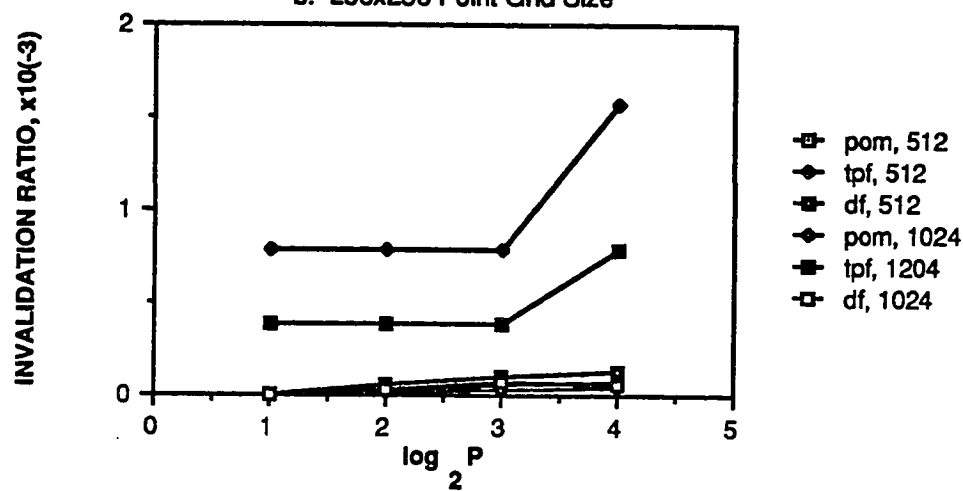
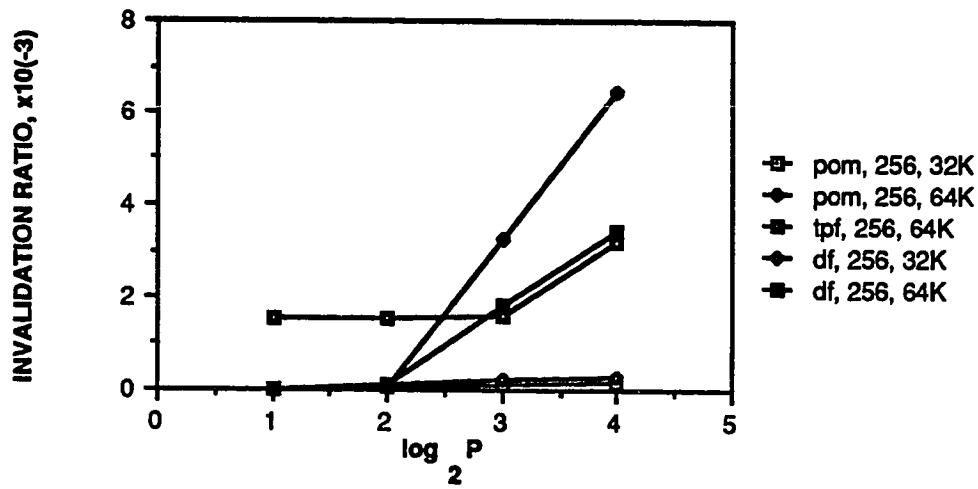
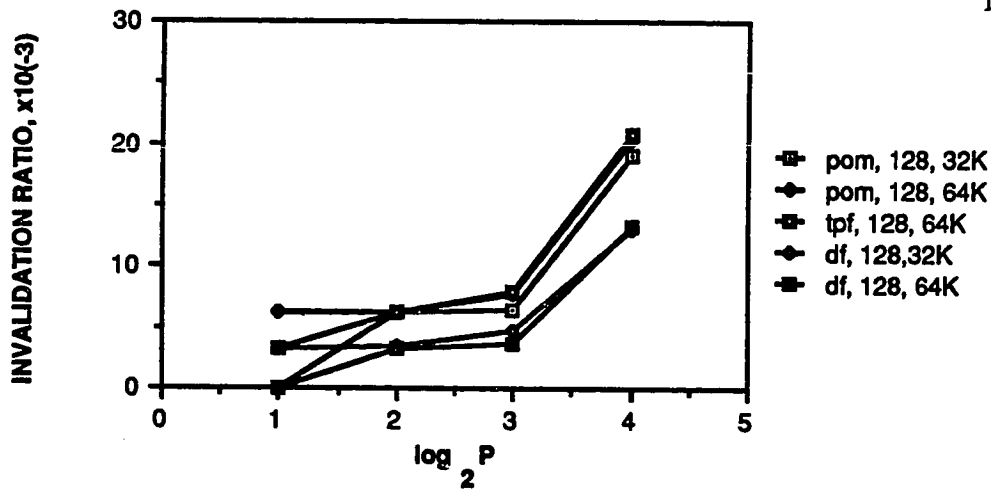


Figure 3.64 Invalidation Ratio versus the Number of Processors, Set-Associative Mapping, All Prefetching Strategies, Square Decomposition, 32-byte Blocksize.

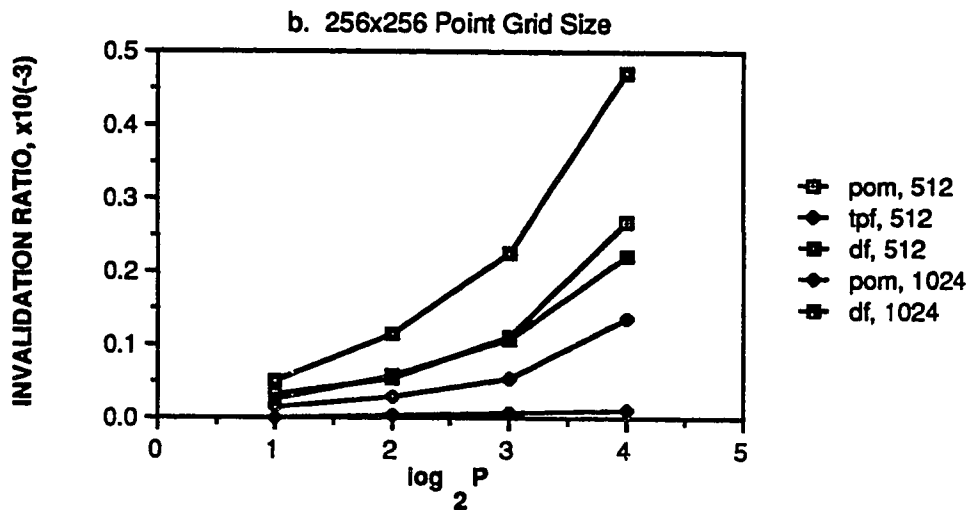
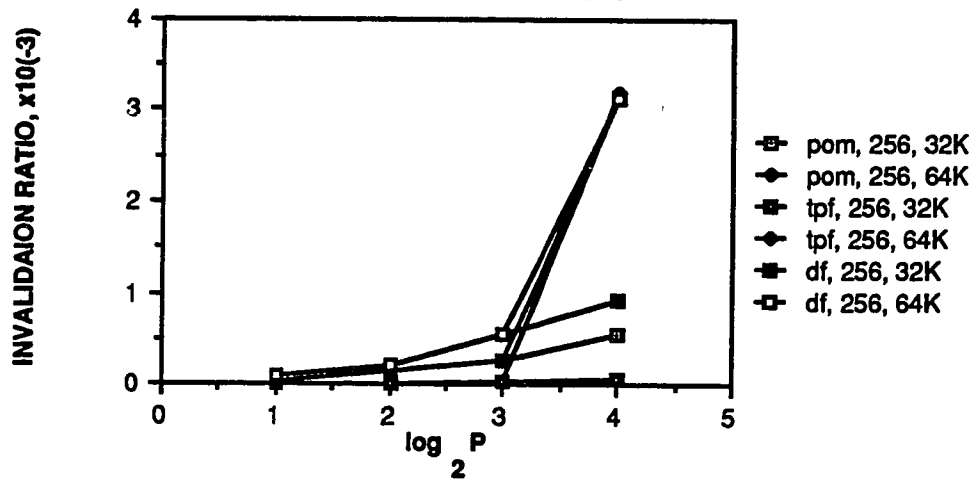
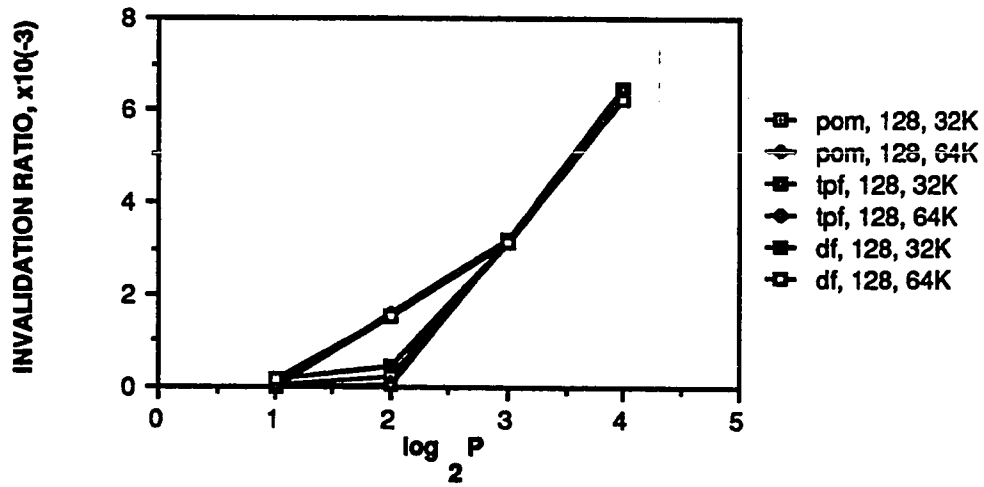


Figure 3.65 Invalidation Ratio versus the Number of Processors, Set-Associative Mapping, All Fetching Strategies, Rectangular Decomposition, 32-byte Blocksize.

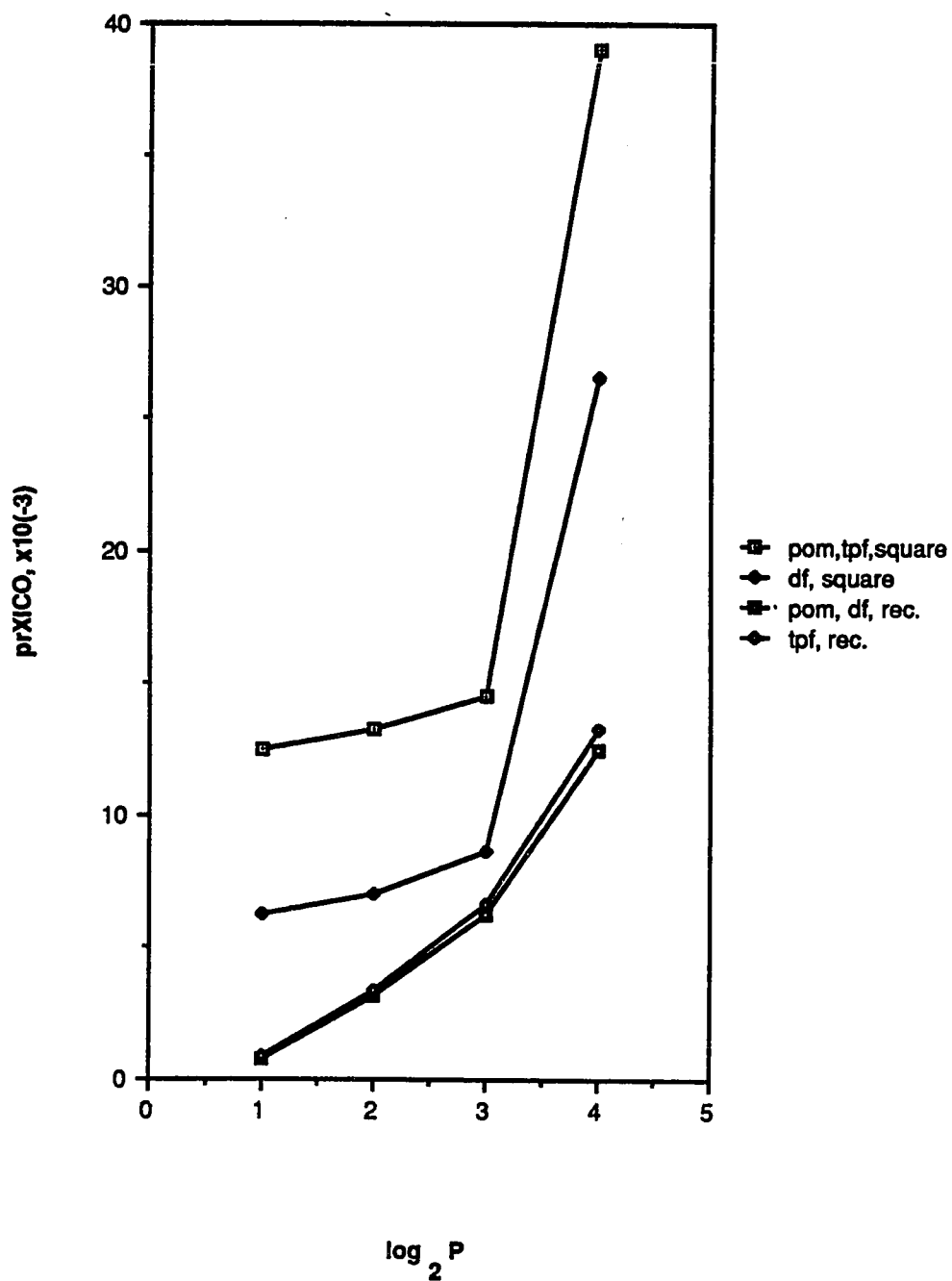


Figure 3.66 prXICO versus the Number of Processors, All Fetching Strategies, 64x64 Point Grid Size, 32-byte Blocksize.

compared with the demand fetching policy for the square decomposition. The prefetching prXICOs are virtually identical to the demand prXICO under rectangular decomposition for this grid size. The intergrid and intragrid contention caused by direct mapping is such that the prefetching prXICOs are identical to the prXICOs for the demand fetching policy and grid sizes larger than 64x64 points.

Figure 3.67 illustrates the prXICO versus the number of processors for all prefetching strategies, set-associative mapping and the square partition. For 8 and 16 processors and the 128x128 point grid size, the tagged prefetching and prefetch-on-miss fetch policies result in nearly identical prXICOs. These prXICOs are also greater than those of the demand fetching policy. For the 4 processor system, demand fetching and prefetch-on-miss have identical low prXICOs while the prXICO is approximately five times their value for the 32Kbyte cache size. When the cache size increases to 64Kbytes, the demand fetching prXICO increases, but not as high as the prefetch-on-miss prXICO. The tagged prefetching prXICO also increases as the cache size increases, although not at the rate of the prefetch-on-miss prXICO. This value equals that of the tagged prefetching prXICO for the 64Kbyte cache size. The 2 processor system has zero prXICOs for demand fetching and prefetch-on-miss; however, the tagged prefetching prXICO has small values for both the 32 and 64Kbyte cache sizes.

For grid sizes larger than 128x128 points the prefetch-on-miss and demand fetching prXICOs are identical with the probabilities, decreasing as the grid size increases. The tagged prefetching prXICOs are somewhat larger than the other fetching prXICOs and the value also decreases as the grid size increases. Note these larger

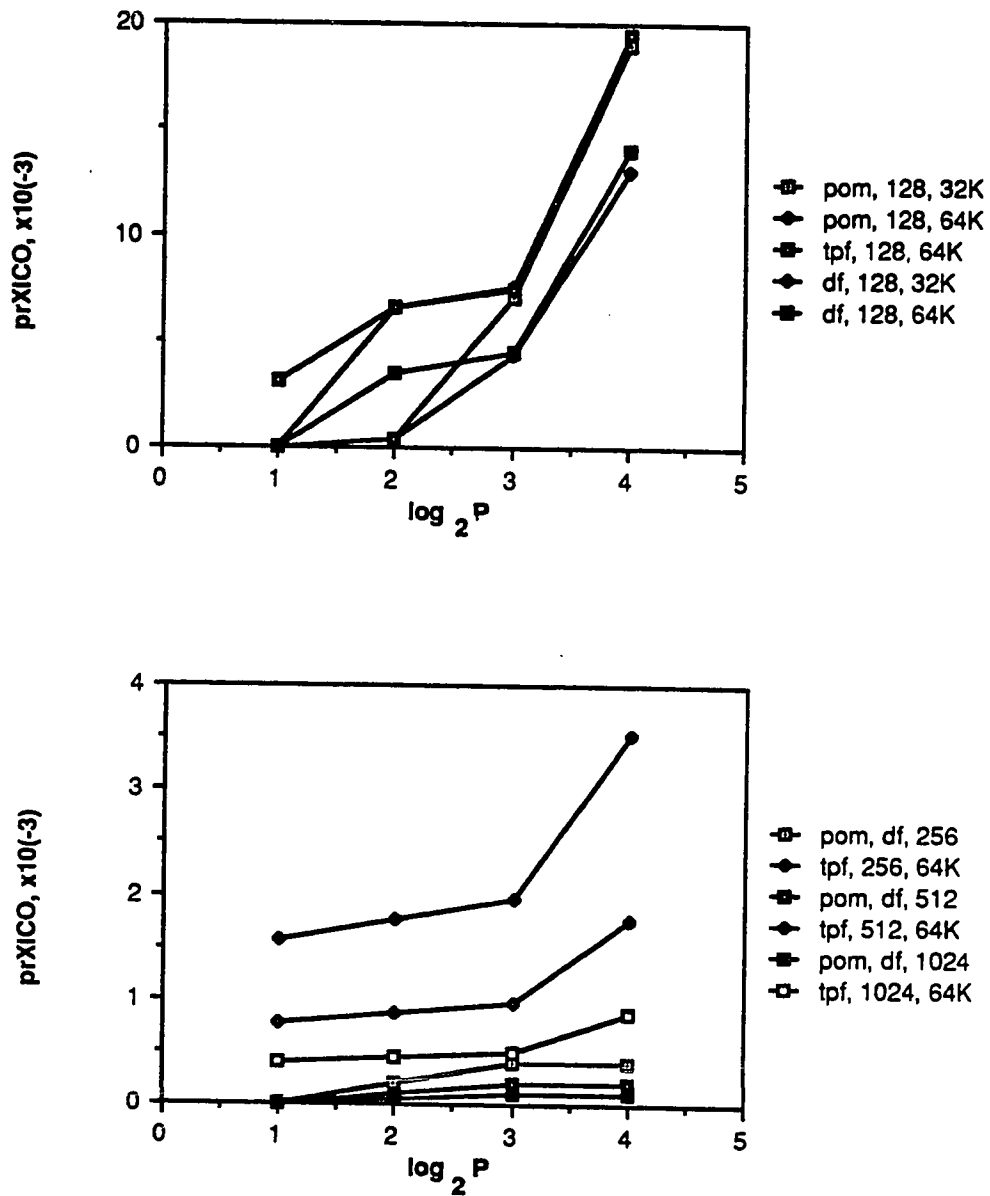
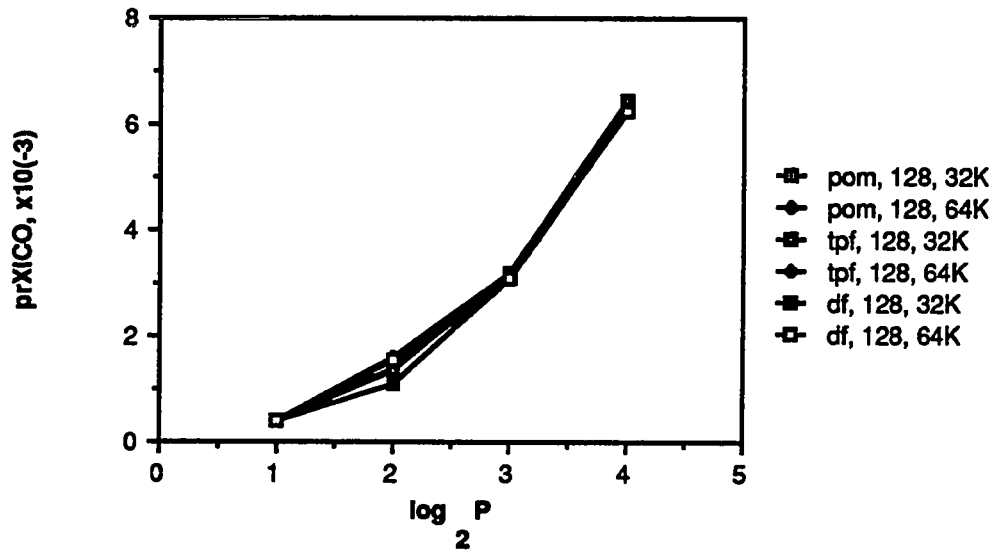


Figure 3.67 prXICO versus the Number of Processors, All Fetching Strategies, Set-Associative Mapping, Square Decomposition, 32-byte Blocksize.

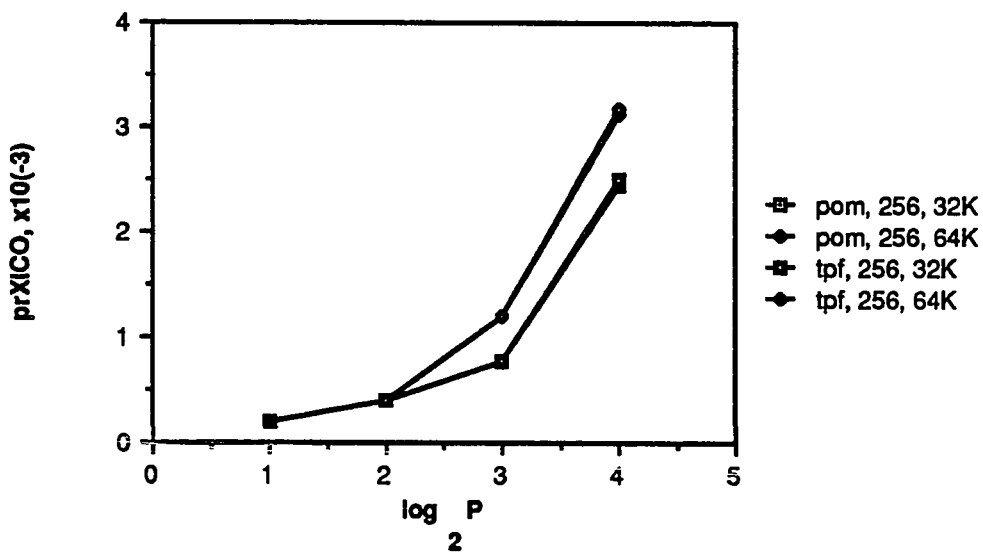
grid prXICOs are smaller than the 64x64 and 128x128 point values. Figure 3.68 presents this same prXICO versus the number of processors for the rectangular decomposition strategy. This figure indicates virtually no discrepancy between the three fetching policies simulated for the all grid sizes. The 256x256 point grid shows the same prXICO values for all fetching policies but the prXICO for the 64Kbyte cache size is somewhat larger than the 32Kbyte cache. The prXICO for grid sizes larger than 256x256 points are identical to the graphs shown in Figures 3.38 and 3.43.

Figure 3.69 presents the prXICS versus the number of processors for all fetching strategies, all grid sizes, direct mapping and square grid decomposition. The prefetching strategies produce larger prXICSs than demand fetching for all grid sizes. When using a 16 processor system, the tagged prefetching strategy produces the greatest probability followed by prefetch-on-miss and finally demand fetching. Tagged prefetching produces the most block replacements for this mapping strategy, followed by prefetch-on-miss and then demand fetching. Since XICSs are directly related to block replacements, this explains the increase in their probabilities of occurrence.

Figures 3.70 through 3.72 illustrate the same prXICSs versus the number of processors for direct mapping on a rectangular grid decomposition, set-associative mapping on a square grid decomposition and set-associative mapping on a rectangular grid decomposition, respectively. In all of these figures the behavior of the prXICS is identical to the graphs presented in Section 3.2.4. All of these figures show the tagged prefetching strategy having the largest probabilities followed by prefetch-on-miss and finally demand fetching. Once again, this is due to the increase in the frequency of replaced blocks.



a. 128x128 Point Grid Size



b. 256x256 Point Grid Size

Figure 3.68 prXICO versus the Number of Processors, All Fetching Strategies, Set-Associative Mapping, Rectangular Decomposition, 32-byte Blocksize.

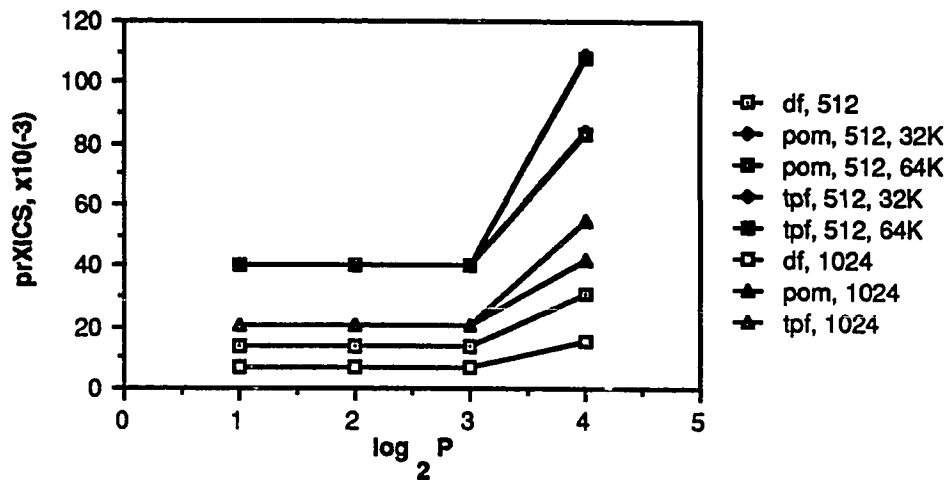
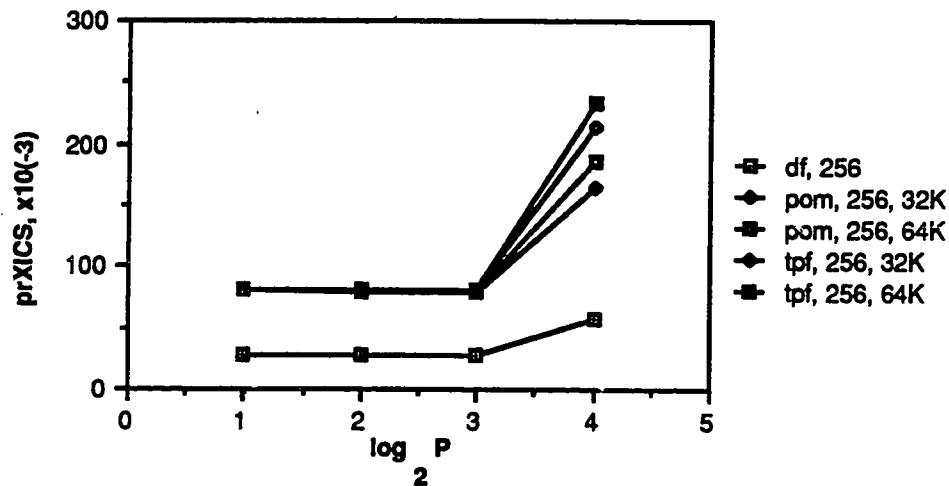
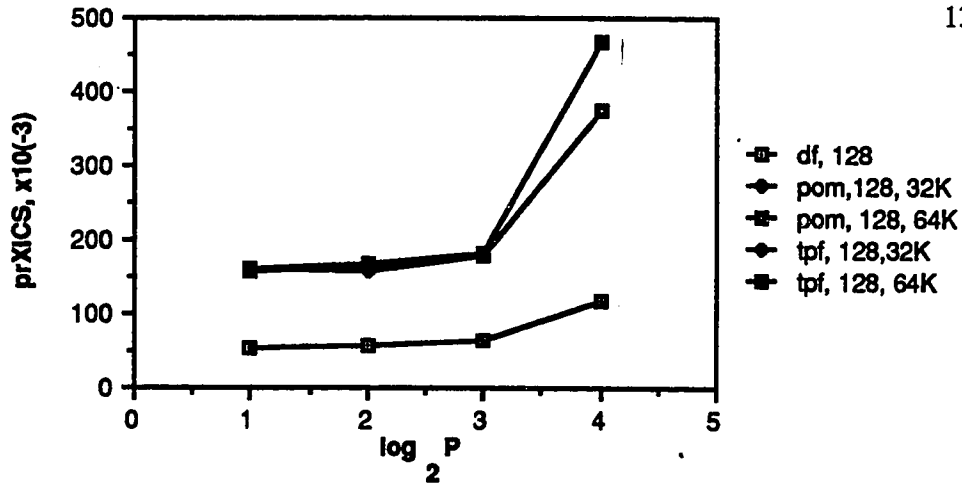


Figure 3.69 prXICS versus the Number of Processors, All Fetching Strategies, Direct Mapping, Square Decomposition, 32-byte Blocksize.

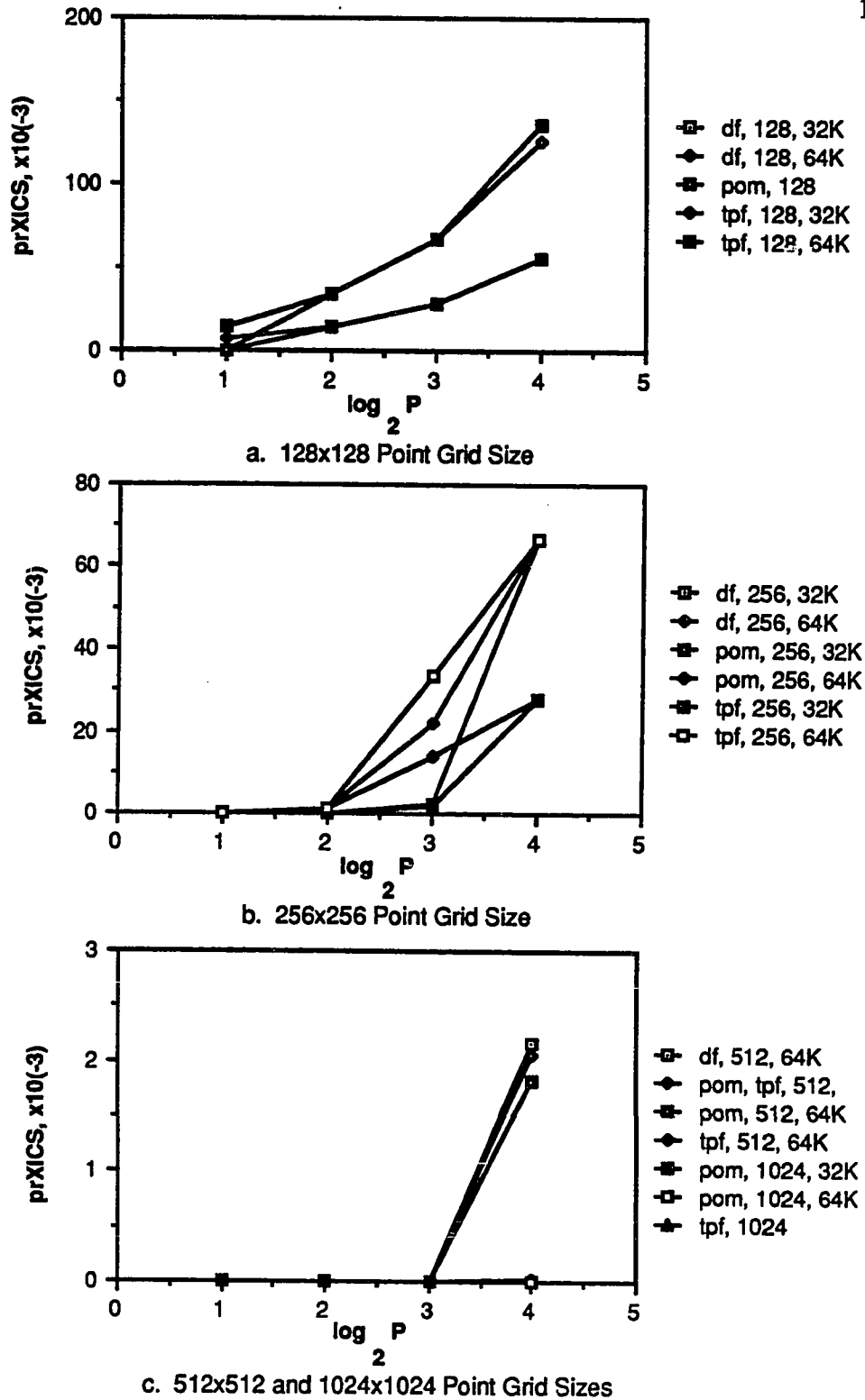
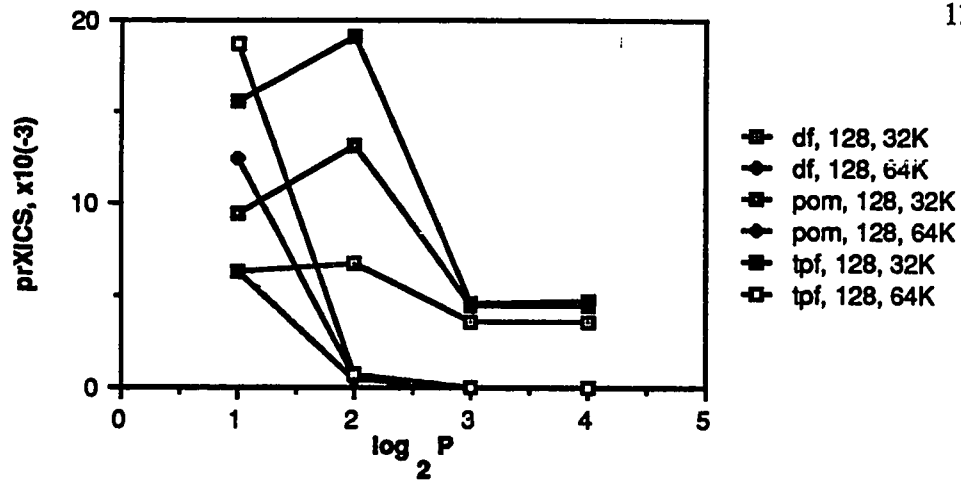
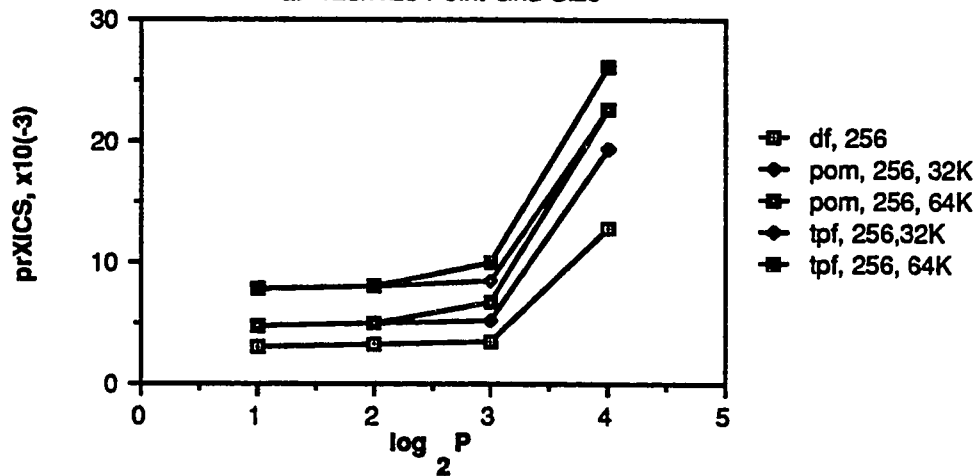


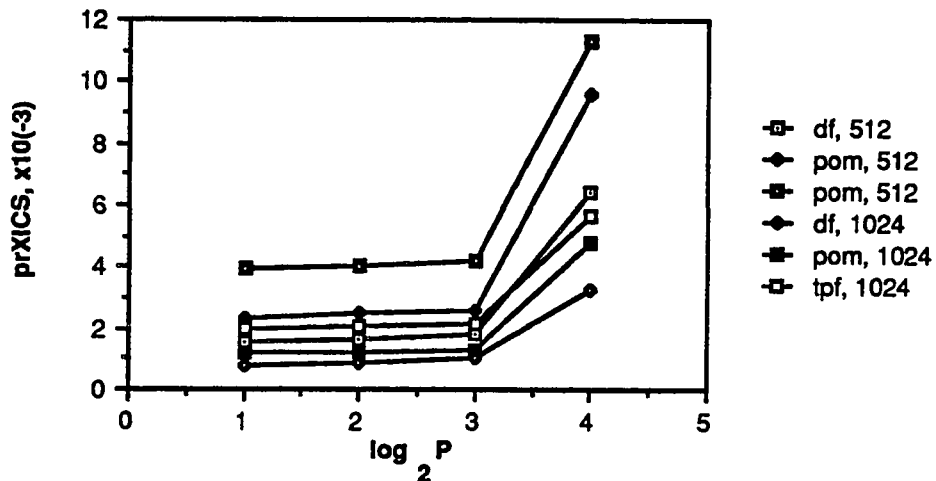
Figure 3.70 prXICS versus the Number of Processors, All Fetching Strategies, Direct Mapping, Rectangular Decomposition, 32-byte Blocksize.



a. 128x128 Point Grid Size



b. 256x256 Point Grid Size



c. 512x512 and 1024x1024 Point Grid Sizes

Figure 3.71 prXICS versus the Number of Processors, All Fetching Strategies, Set-Associative Mapping, Square Decomposition, 32-byte Blocksize.

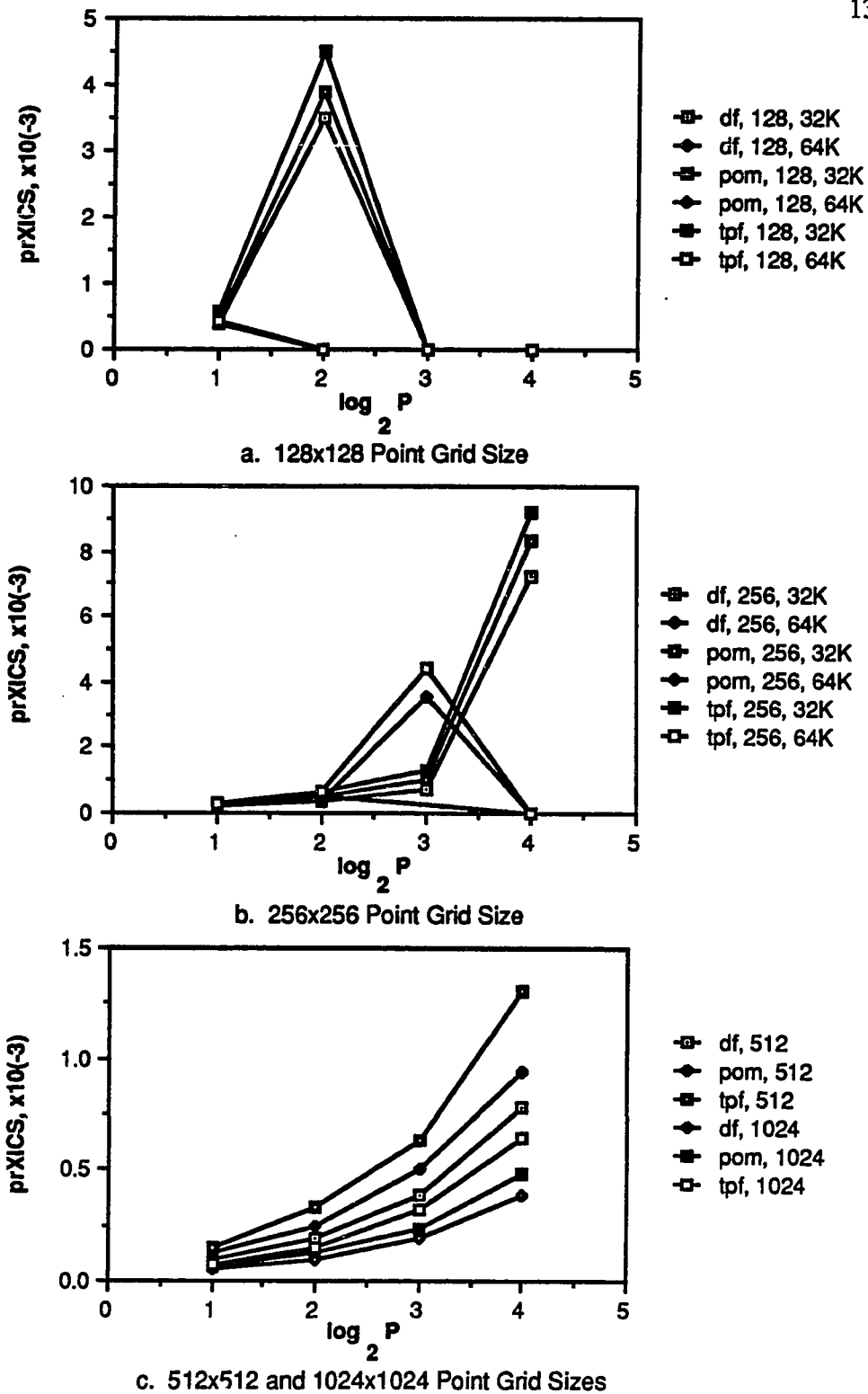


Figure 3.72 prXICS versus the Number of Processors, All Fetching Strategies, Set-Associative Mapping, Rectangular Decomposition, 32-byte Blocksize.

The prefetch ratio (PR) and the lookup (LR) ratio for all features simulated (including both prefetching policies) on the 64x64 point grid is presented in Figure 3.73. Part a of the figure illustrates these parameters for the square partition. For 2 and 4 processor systems, these ratios are nearly identical. As the number of processors increase to 8 and 16 processors, the LR becomes indicatively higher than the PR for both prefetching strategies. Also observe the slight differences between the prefetch-on-miss and tagged prefetching PRs and the between the prefetch-on-miss and tagged prefetching LRs for 8 and 16 processors. Part b of the figure indicates that the tagged prefetching strategy produces more overhead for the rectangular partition. Observe the LR and PR for this prefetching strategy is specifically higher than the value for prefetch-on-miss. Also, the LR is slightly higher than the PR for tagged prefetching using 4 or more processors. The graph also indicates no discrepancy between the LR and the PR for the prefetch-on-miss policy.

Figure 3.74 shows the LR and PR versus the number of processors for the demand fetching policy using the square decomposition of grids larger than 64x64 points. The figure shows these parameters remaining relatively constant as the number of processors increase. The slight increase in the prefetch ratios as the number of processors double from 8 to 16 is the only exception. Observe the tagged prefetching LRs and PRs are somewhat larger than the prefetch-on-miss LRs and PRs, respectively for all grid sizes. Also note the small differences between the LRs and PRs of the various grid sizes and the overall differences between the higher LRs and the PRs. Figure 3.75 illustrates these same parameters for the rectangular partition. This graph shows the behavior of the LRs and PRs to be independent of the

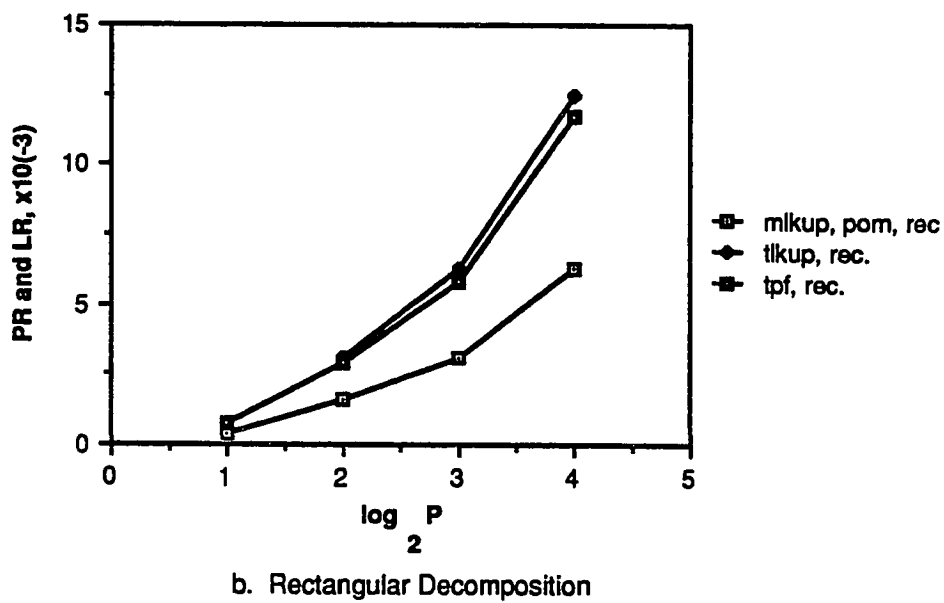
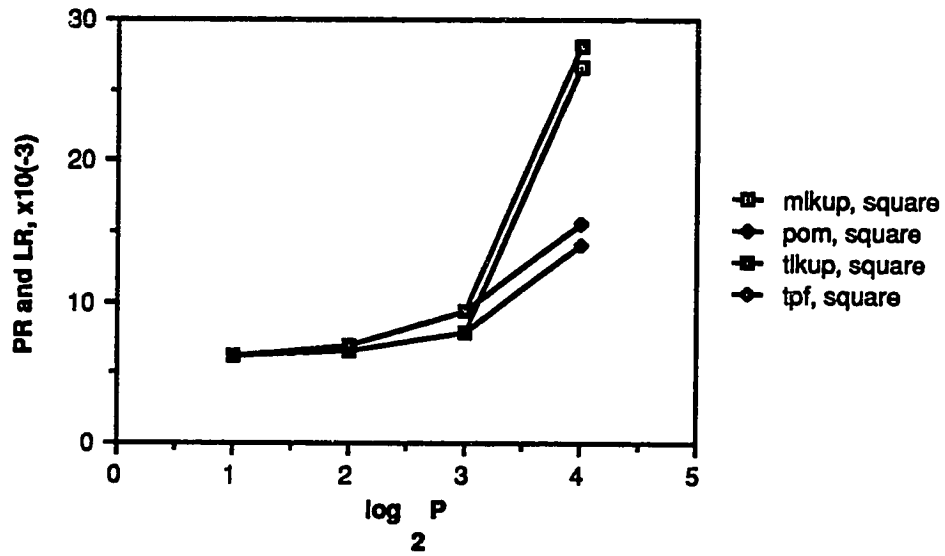
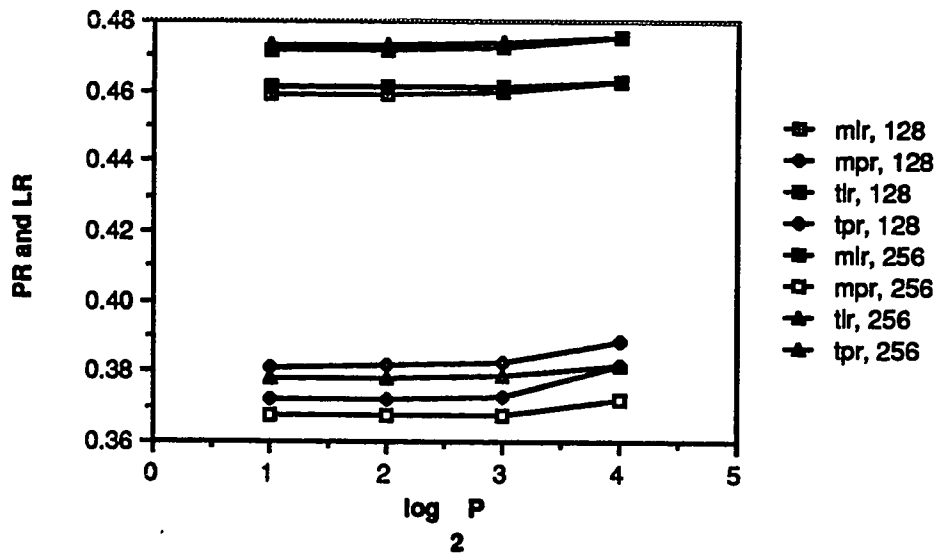
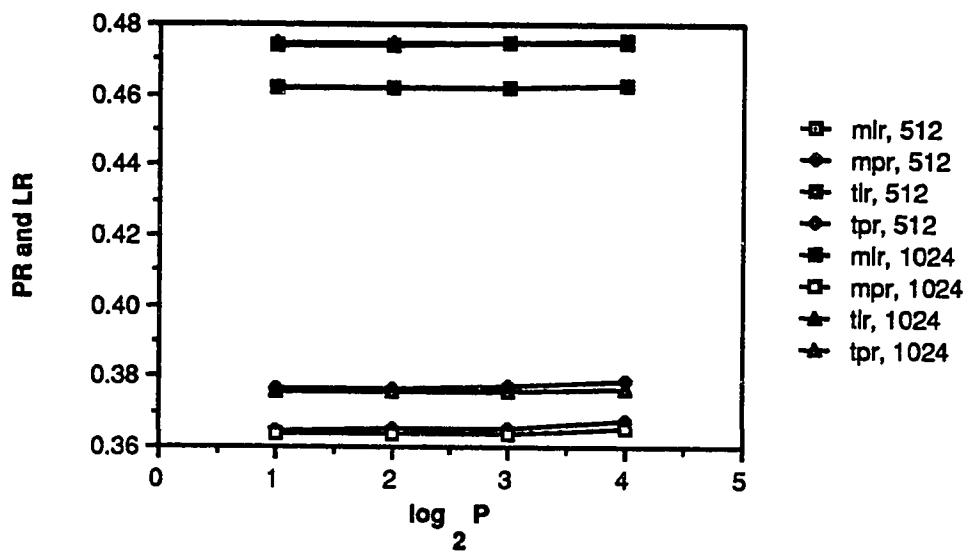


Figure 3.73 Prefetch Ratio and Lookup Ratio versus the Number of Processors, Both Mapping Policies, 64x64 Point Grid Size, 32-byte Blocksize.



a. Smaller Grid Sizes



b. Larger Grid Sizes

Figure 3.74 Prefetch Ratio and Lookup Ratio versus the Number of Processors, Square Decomposition, Direct Mapping, 32-byte Blocksize.

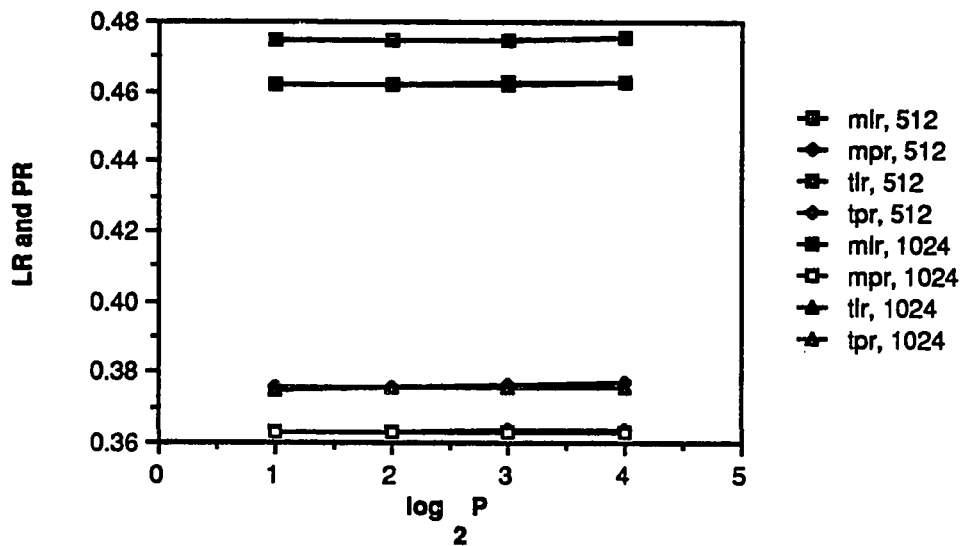
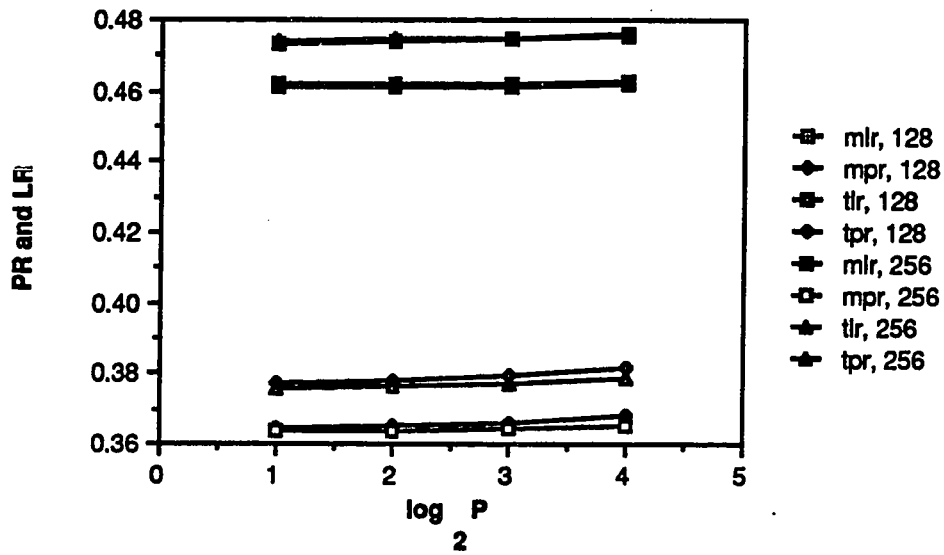


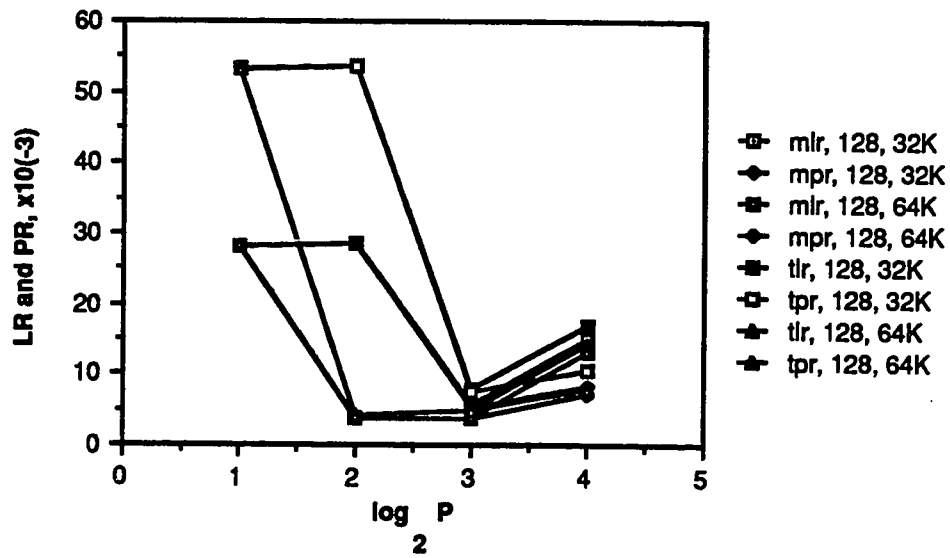
Figure 3.75 Prefetch Ratio and Lookup Ratio versus the Number of Processors, Rectangular Decomposition, Direct Mapping, 32-byte Blocksize.

decomposition strategy.

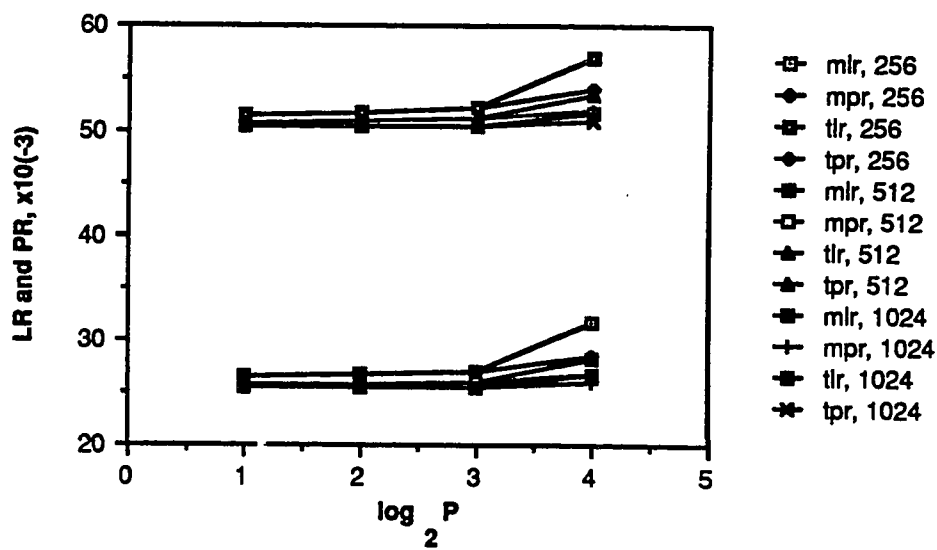
Figure 3.76 presents the LR and PR versus the number of processors for set-associative mapping and the square partition. Concentrating on the 128x128 point grid, for each cache size, the LR and PR is identical when the number of processors is less than 16. The 16 processor system has a LR on the average of 1.8 times the PR. The grid sizes larger than 128x128 points show nearly identical LRs and PRs for each prefetching strategy with the tagged prefetching policy parameters approximately twice the prefetch-on-miss values. Observe that all LRs and PRs for the set-associative mapping function are two orders of magnitude lower than the direct mapping values. Figure 3.77 presents these parameters for the rectangular decomposition strategy. The graphs generally show identical LRs and PRs for each simulation configuration with the tagged prefetching values approximately twice the prefetch-on-miss values.

3.3. Multiprocessor Speedup

Figure 3.78 presents the multiprocessor speedup versus the number of processors for 32 and 64Kbyte caches, direct mapping, square and rectangular decompositions and a single bus interconnection (SBI). The figure shows a square decomposition speedup ranging from 1.2 to 1.9 for the 64x64 point grid. All other grid sizes exhibit a speedup less than 0.5 and relatively constant for this partition. This performance is strictly prohibitive for Jacobi's algorithm. The major factor effecting this unacceptable degradation is the assumption that the maximum waiting time for a SBI is $P-1$ times the effective memory access time. The 64x64 point grid provides



a. 128x128 Point Grid Size



b. Larger Grid Sizes

Figure 3.76 Prefetch Ratio and Lookup Ratio versus the Number of Processors, Square Decomposition, Set-Associative Mapping, 32-byte Blocksize.

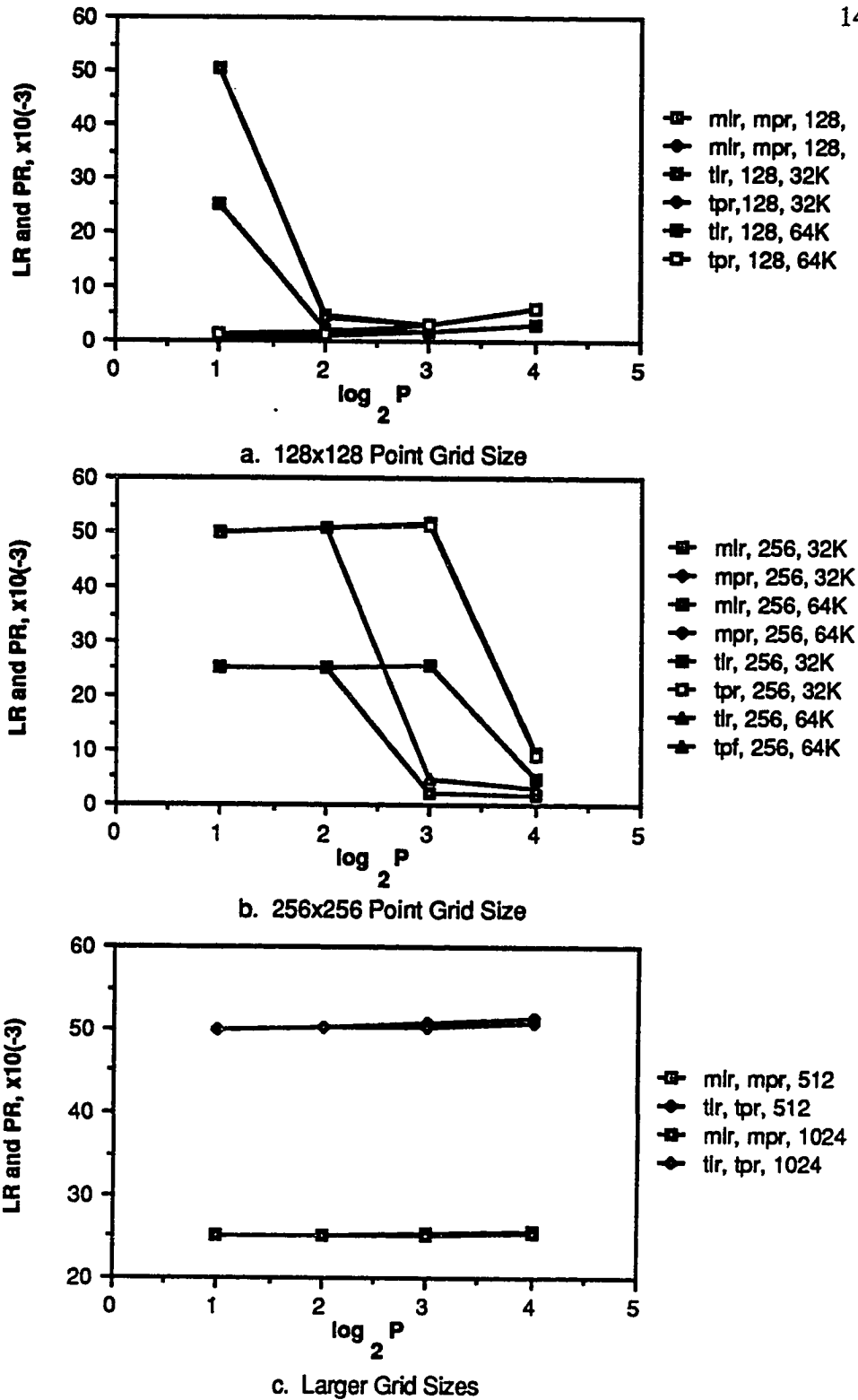
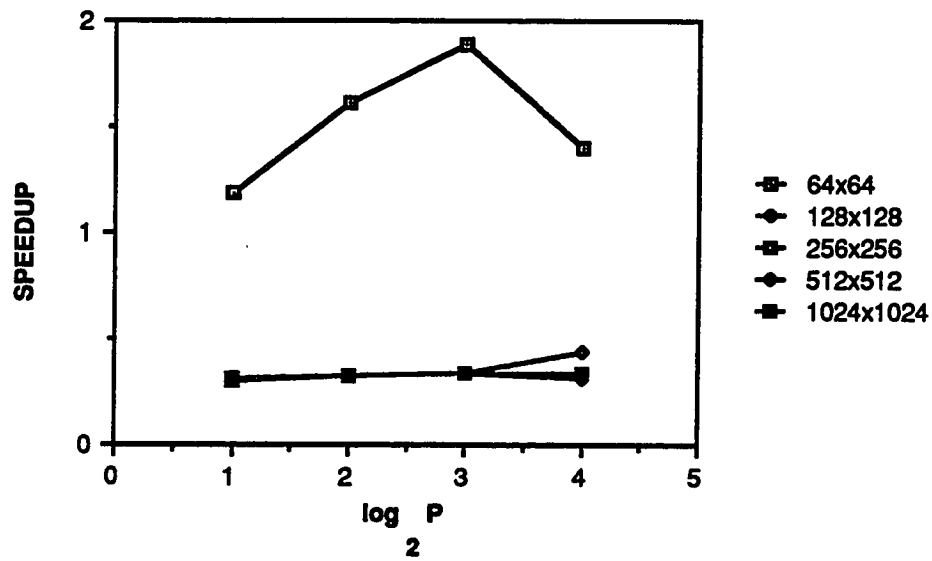
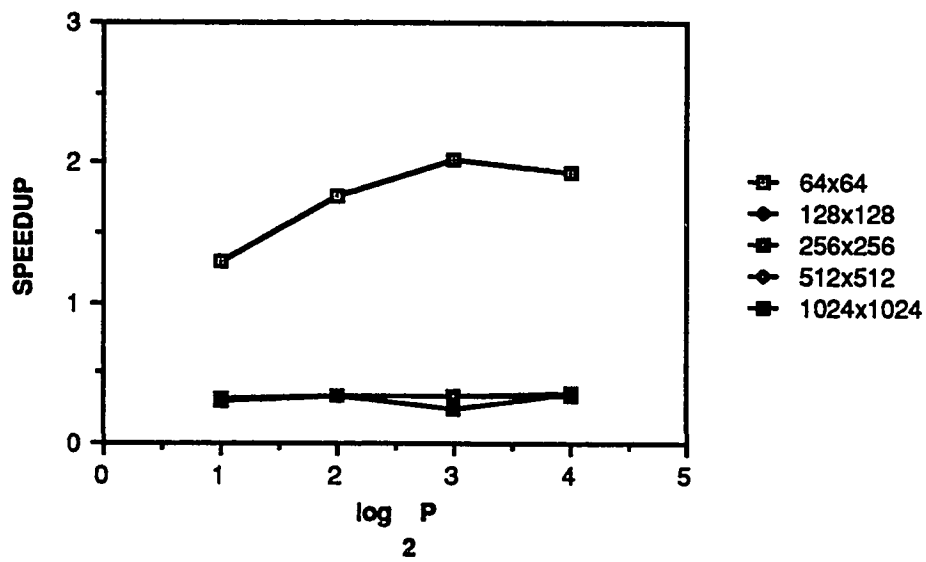


Figure 3.77 Prefetch Ratio and Lookup Ratio versus the Number of Processors, Rectangular Decomposition, Set-Associative Mapping, 32-byte Blocksize.



a. Square Decomposition



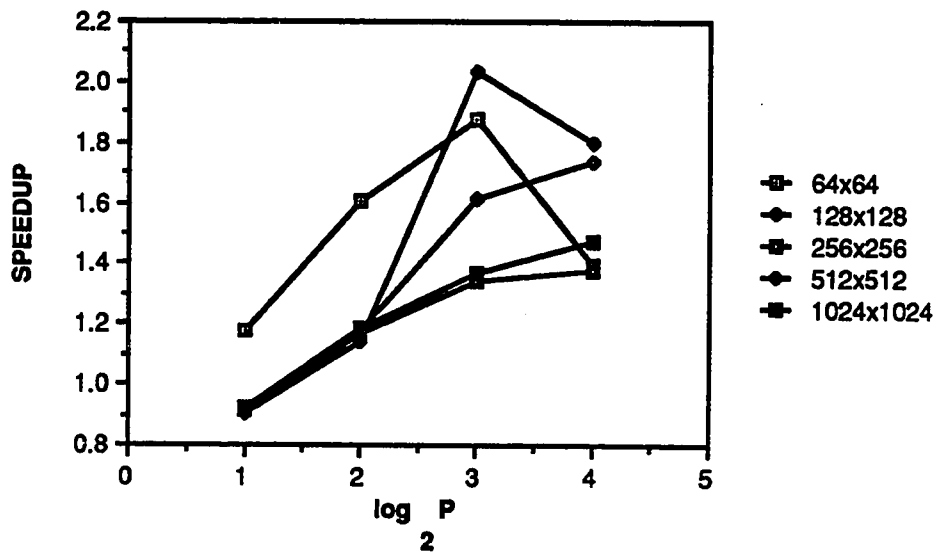
b. Rectangular Decomposition

Figure 3.78 Speedup versus the Number of Processors, Direct Mapping, Single Bus, 32-byte Blocksize.

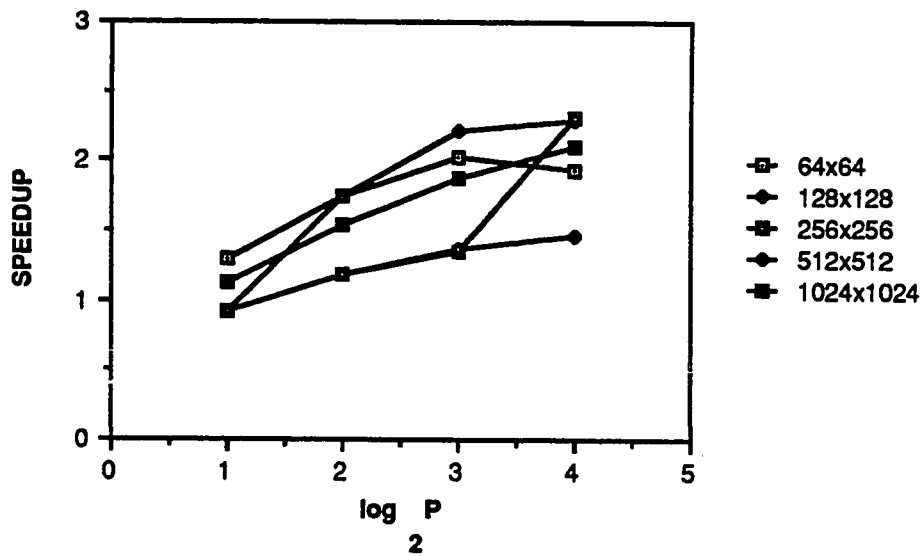
greater speedup because its MR is orders of magnitude lower than the other grid sizes. The rectangular decomposition curves are similar to those of the square decomposition. The 64x64 point grid speedup is slightly higher than the square partition because the MR is lower for this partition.

Figures 3.79 and 3.80 also illustrate the speedup versus the number of processors; however, the set-associative mapping strategy is presented for the 32Kbyte and 64Kbyte cache sizes, respectively. These curves show improved performance over the direct mapping strategy for grid sizes larger than 64x64 points. The performance for the 64x64 point grid is unchanged. For most of the grid sizes, the speedup increases only slightly as the number of processors increase. Again, the rectangular decomposition strategy is shown to have better performance than the square decomposition strategy.

Figure 3.81 shows the same direct mapping speedup curve for the full crossbar interconnection (FCI). It is assumed the waiting time for this interconnection structure is zero. As a result, the speedup curves improve significantly. Again, for both cache sizes and decomposition strategies, the 64x64 point grid sizes has better performance as a result of the relatively considerable improvement in its MR. The speedup for all the other grid sizes are identical. The rectangular decomposition strategy has slightly improved performance over the square decomposition strategy for the 64x64 point grid size. The speedup results for the set-associative mapping function and the full crossbar interconnection are shown in Figure 3.82. This graph also shows an improved speedup for grid sizes larger than 64x64 points.

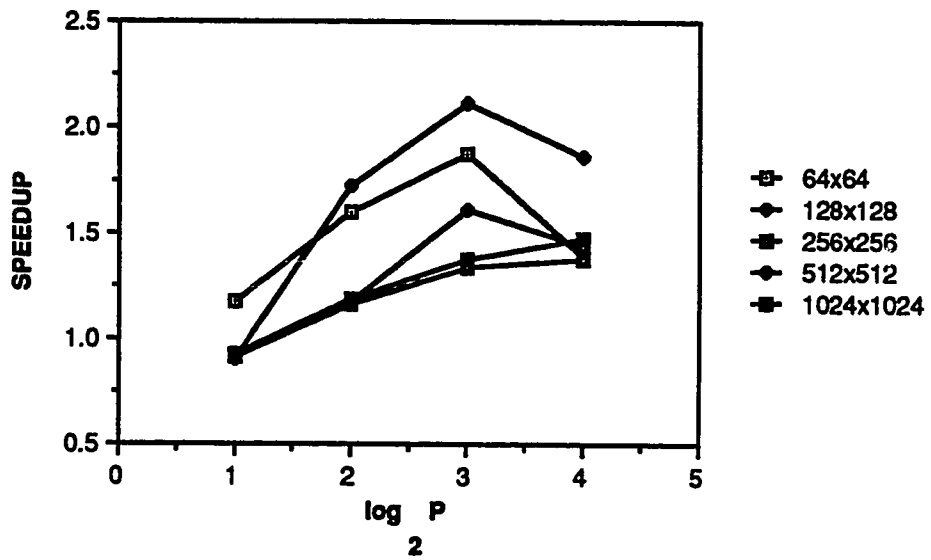


a. Square Decomposition

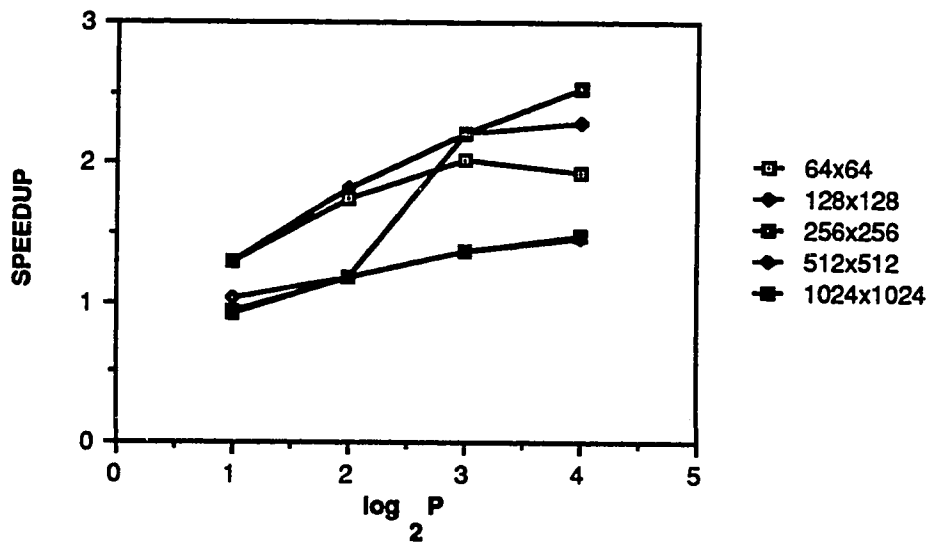


b. Rectangular Decomposition

Figure 3.79 Speedup versus the Number of Processors, Set-Associative Mapping, 32Kbyte Cache Size, 32-byte Blocksize.

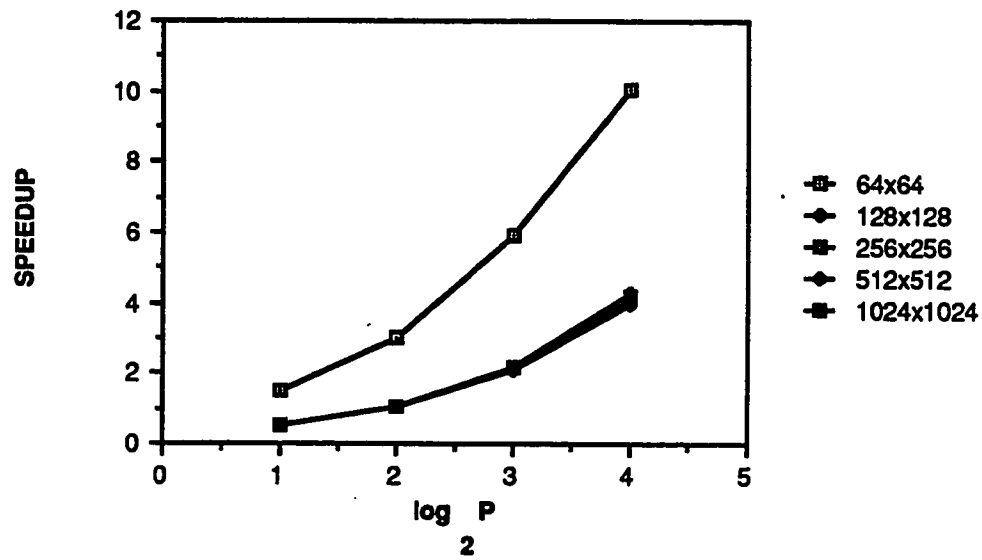


a. Square Decomposition

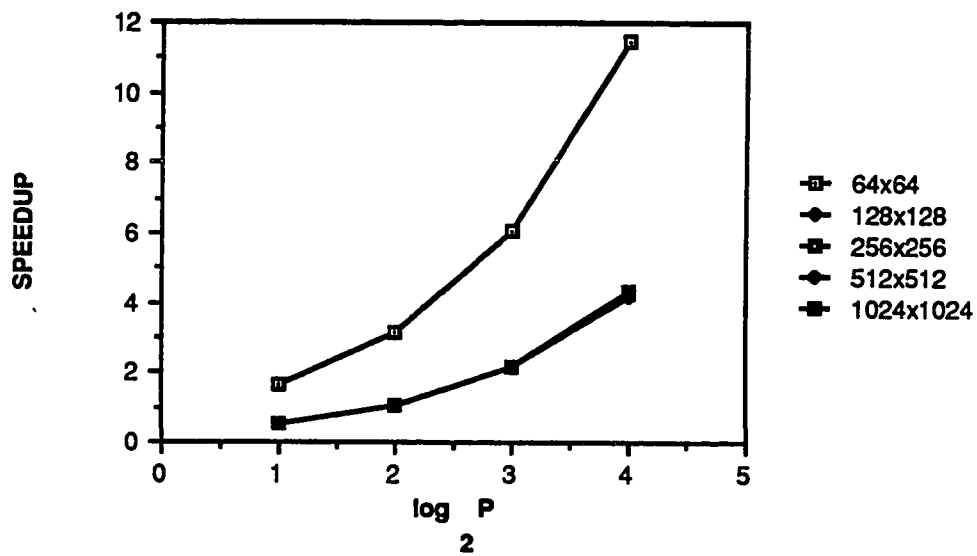


b. Rectangular Decomposition

Figure 3.80 Speedup versus the Number of Processors, Set-Associative Mapping, 64Kbyte Cache Size, 32-byte Blocksize.

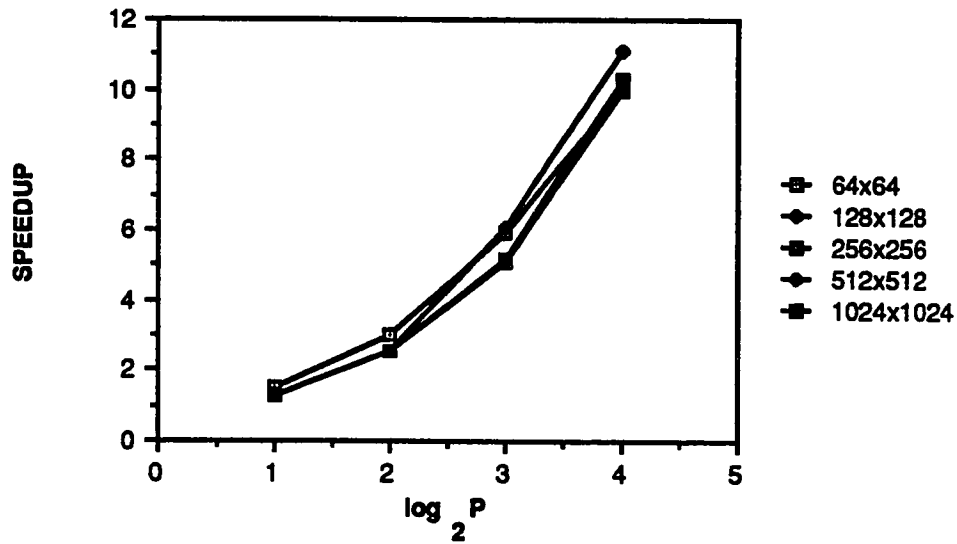


a. Square Decomposition

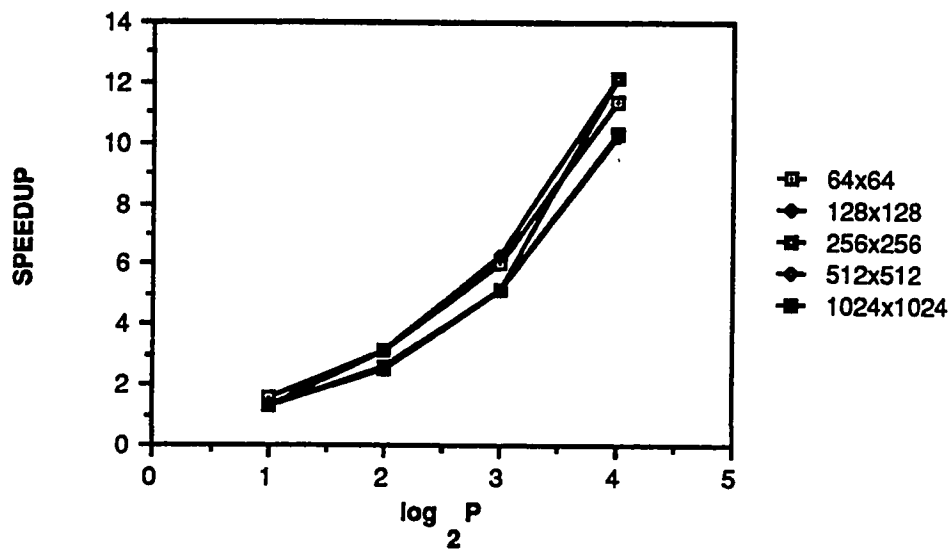


b. Rectangular Decomposition

Figure 3.81 Speedup versus the Number of Processors, Direct Mapping, Full Crossbar Interconnection, Both Cache Sizes, 32-byte Blocksize.



a. Square Decomposition



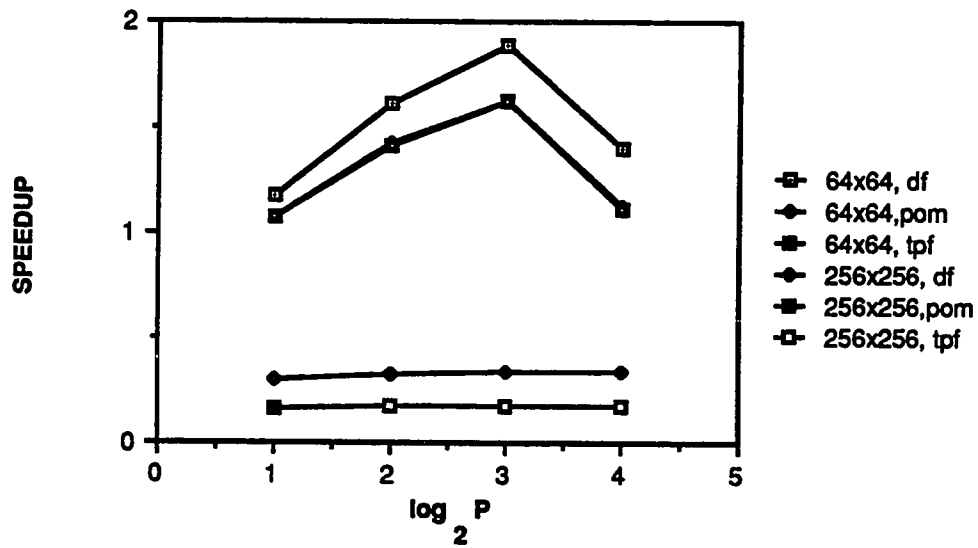
b. Rectangular Decomposition

Figure 3.82 Speedup versus the Number of Processors, Set-Associative Mapping, Full Crossbar Interconnection, Both Cache Sizes, 32-byte Blocksize.

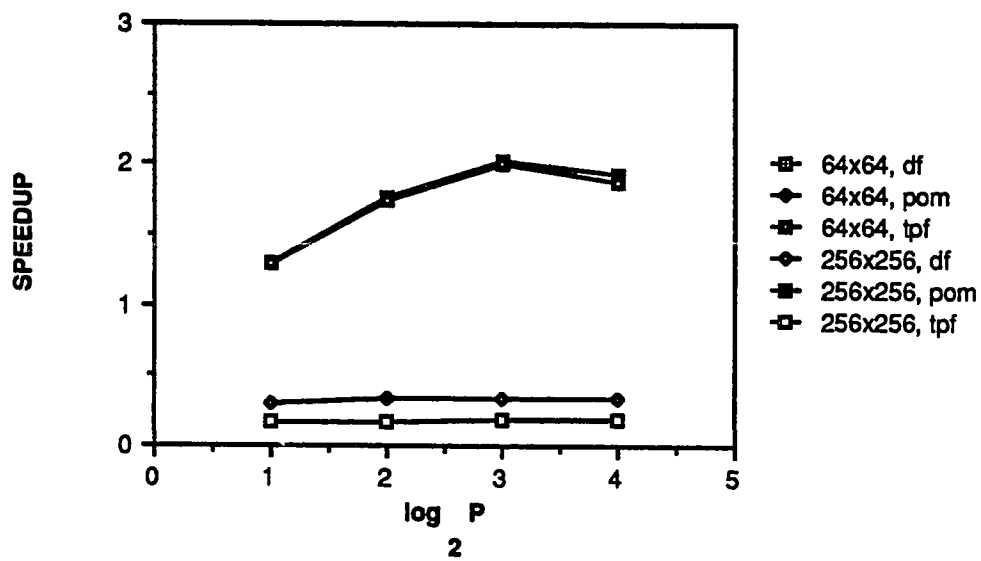
The speedup curves for all fetching strategies, direct mapping, SBI and 64x64 and 256x256 point grids are shown in Figure 3.83. (All other grid sizes produce speedups similar to the 256x256 point grid size). This figure shows demand fetching having the best speedup for the square decomposition strategy. Prefetch-on-miss and tagged prefetching have equal speedups for both grid sizes. All fetching policies have relatively equal speedups for the 64x64 point grid size using the rectangular partition. However, demand fetching has the best performance for the 256x256 point grid size and prefetch-on-miss and tagged prefetching have equal speedups for this partition also. The primary reason why demand fetching has the best speedup is that it has a smaller MR than the prefetching strategies for direct mapping.

Figure 3.84 illustrates the same speedup curves for the set-associative mapping function. For the square decomposition strategy, demand fetching has the best speedup for 2-4 processors. The prefetch-on-miss and tagged prefetching speedups are slightly higher than the demand fetching speedup for 16 processors (both grid sizes). For the rectangular decomposition strategy, both prefetching strategies perform better than demand fetching for the 2 processor system. As the number of processors increase to 4, the prefetching strategies outperform demand fetching only for the 256x256 point grid size. The 8 and 16 processor configurations see demand fetching outperforming the prefetching strategies for both grid sizes.

Figure 3.85 presents the speedup curves for the FBI, direct mapping and all fetching strategies. The square decomposition strategy shows demand fetching outperforming the prefetching strategies for the 16 processor system. In all other instances, the fetching strategies are equal for the 64x64 point grid size. Demand

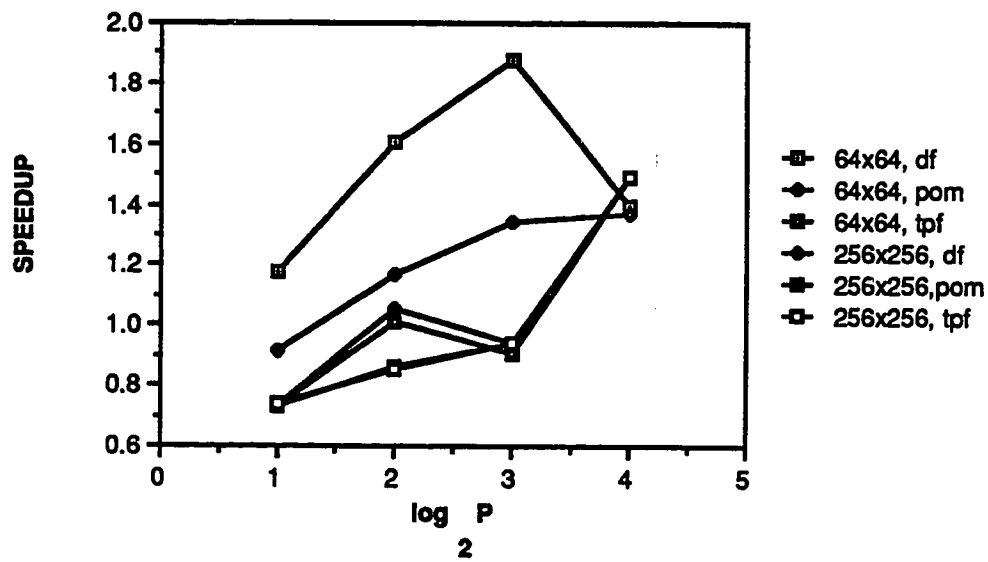


a. Square Decomposition

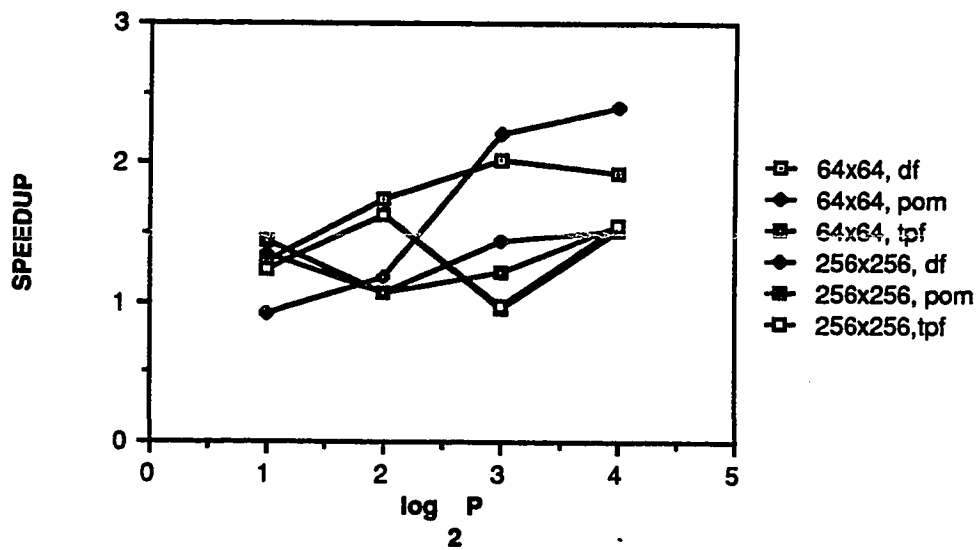


b. Rectangular Decomposition

Figure 3.83 Speedup versus the Number of Processors, Direct Mapping Single Bus, All Fetching Strategies, Both Cache Sizes, 32-byte Blocksize.

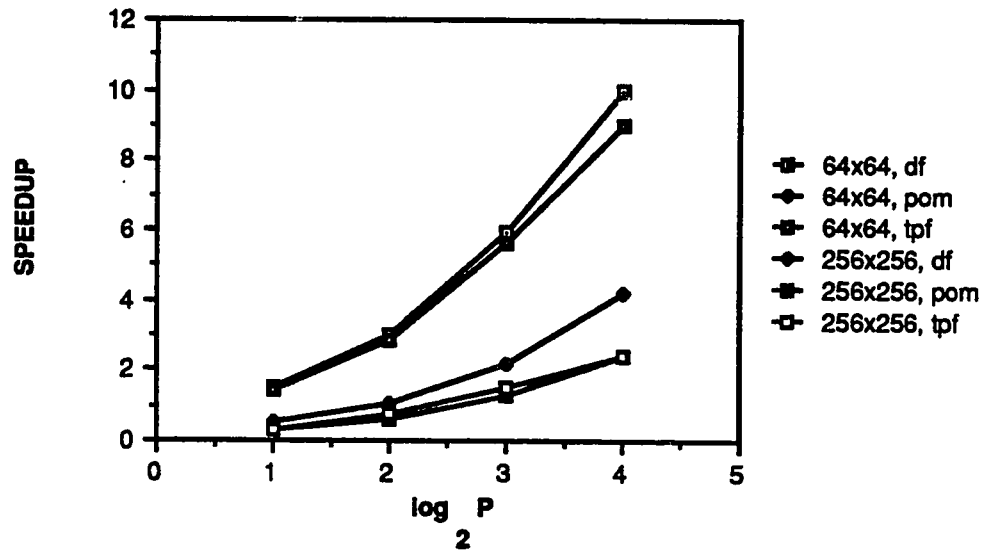


a. Square Decomposition

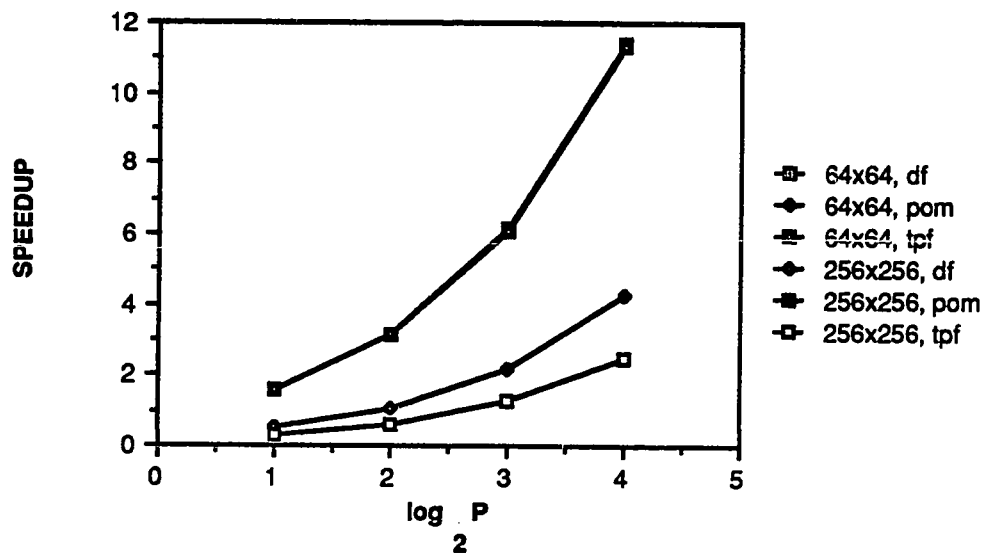


b. Rectangular Decomposition

Figure 3.84 Speedup versus the Number of Processors, Set-Associative Mapping, Single Bus, All Fetching Strategies, Both Cache Sizes, 32-byte Blocksize.



a. Square Decomposition



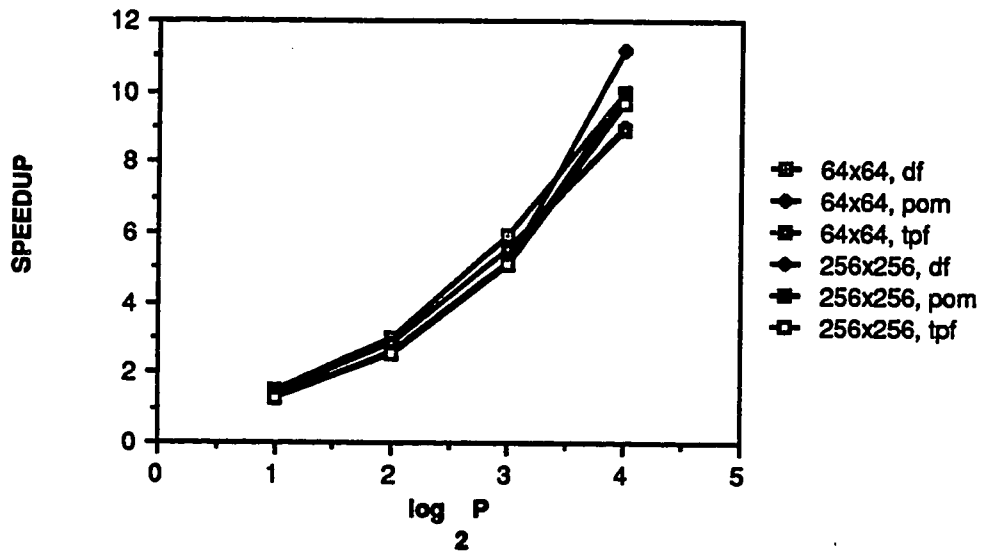
b. Rectangular Decomposition

Figure 3.85 Speedup versus the Number of Processors, Direct Mapping, Full Crossbar Interconnection, All Fetching Strategies, 32-byte Blocksize.

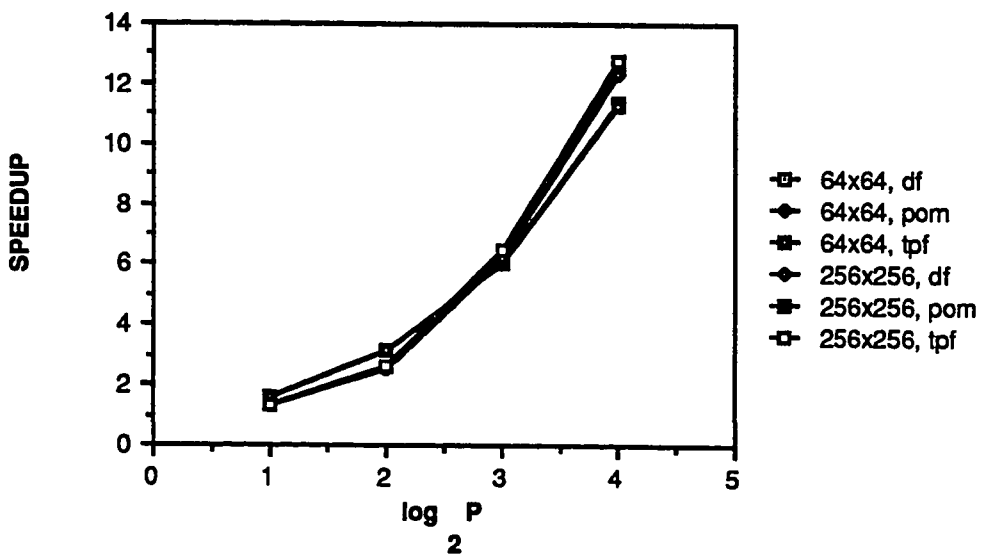
fetching outperforms both prefetching strategies for the 256x256 point grid size and both decomposition strategies. Figure 3.86 shows the speedup curves for set-associative mapping. Both decomposition strategies and grid sizes show very small differences between all of the speedup curves presented.

Figure 3.87 displays the iteration time degradation versus the number of processors for both cache sizes, mapping functions and interconnection networks. A square decomposition strategy and a 32-byte cache blocksize is assumed. The curves show the single bus interconnection network causing a greater degradation than the full crossbar system. The large degradation exhibited by the 64x64 point PDE grid size is a direct result of the infinite MRD for this grid size. Moreover, the direct mapping ITD is greater than its set-associative counterpart. This is attributed to intergrid and intragrid contention present for this mapping strategy. Jacobi's algorithm iteration time for enabled coherence is between 1.17 and 1.35 times the disabled coherence iteration time for the 256x256 point grid size, direct mapping and a single bus system. This multiplicative range is reduced to between 1.04 and 1.08 for the set-associative mapping strategy.

Figure 3.88 presents the ITD versus the number of processors for the rectangular decomposition strategy. The behavior of these curves are similar to the square decomposition ITD curves; however, the 64x64 point PDE grid size values are slightly smaller. For example, the square decomposition ITD ranges from 1.05 to 1.45 for direct mapping. The same ITD for the rectangular decomposition strategy ranges from 1.01 to 1.22. This is attributed to the fact that the rectangular decomposition strategy produces smaller MRs and XIs for the 32-byte blocksize.

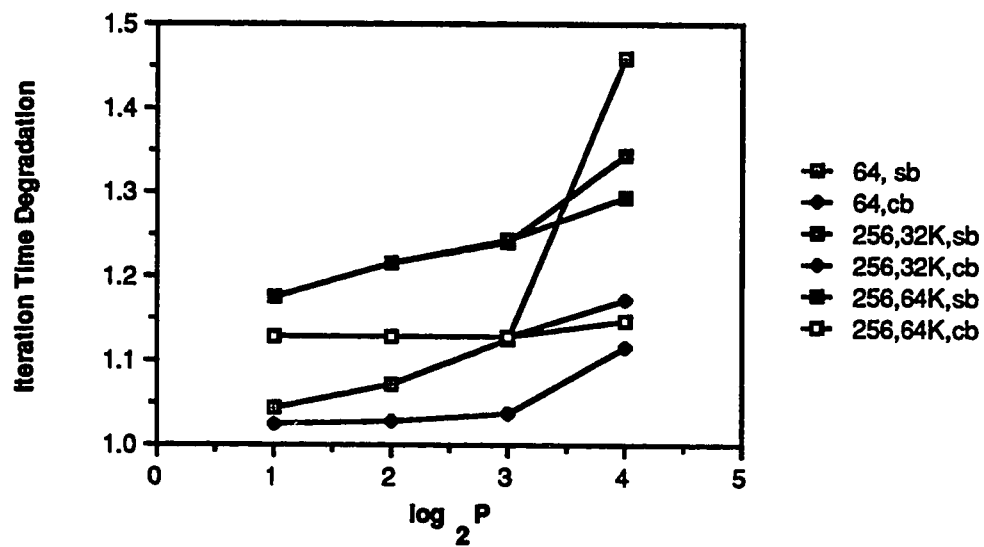


a. Square Decomposition

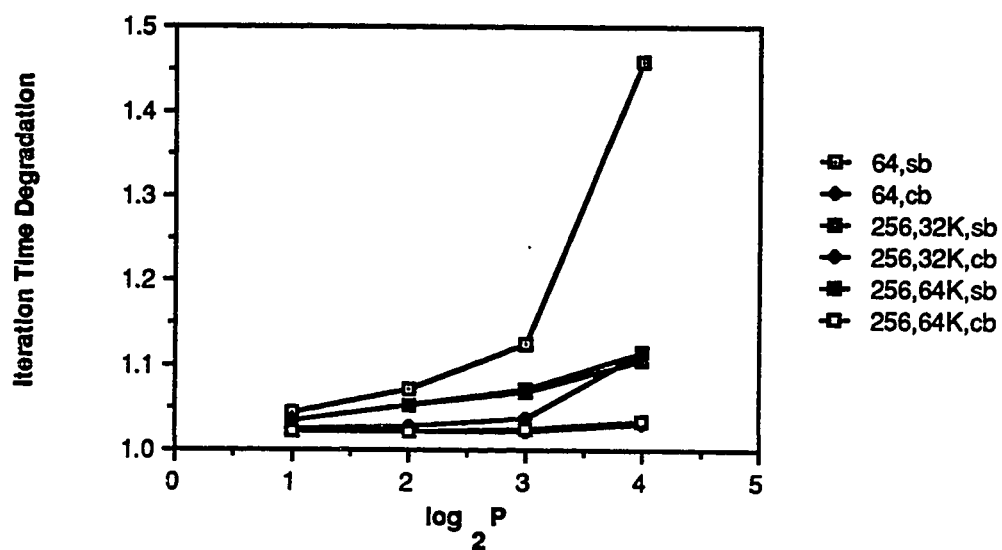


b. Rectangular Decomposition

Figure 3.86 Speedup versus the Number of Processors, Set-Associative Mapping, Full Crossbar Interconnection, All Fetching Strategies, 32-byte Blocksize.

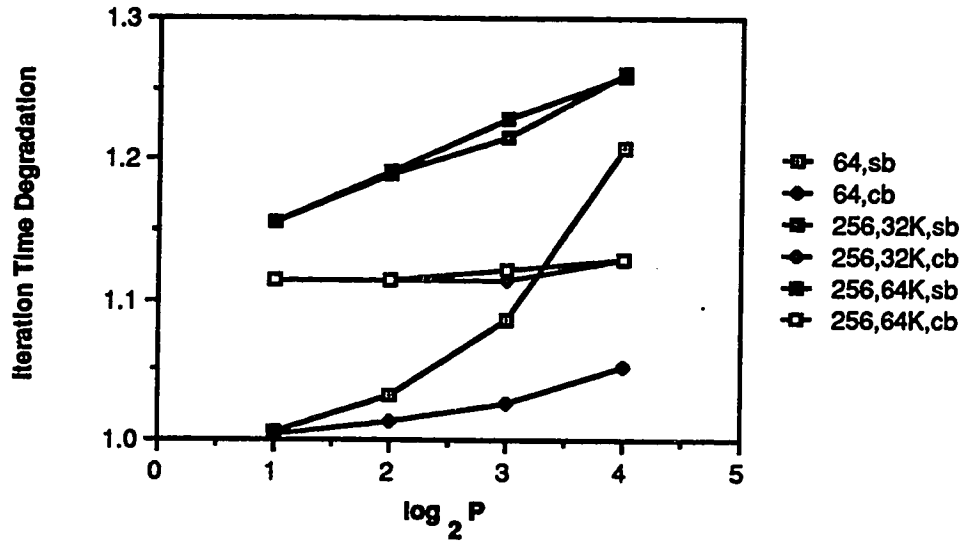


a. Direct Mapping

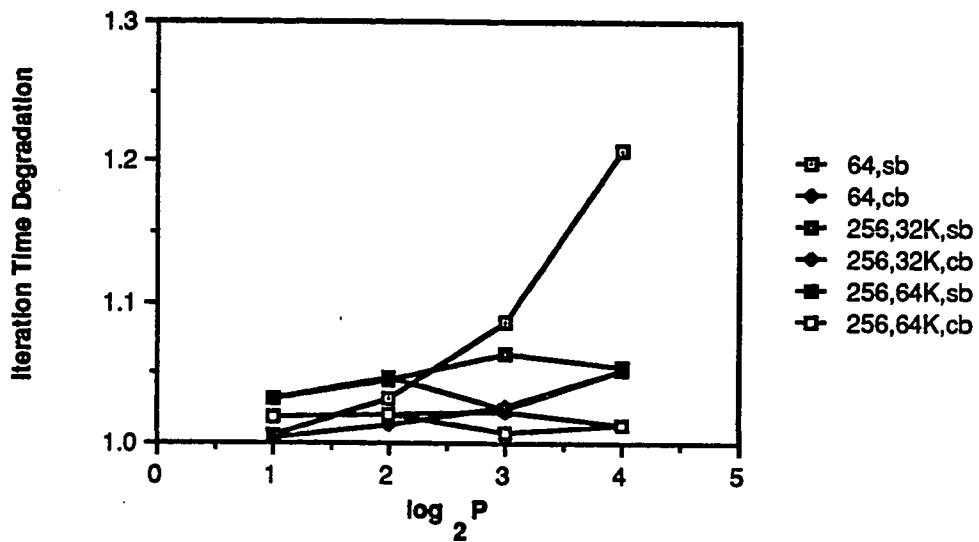


b. Set-Associative Mapping

Figure 3.87 Iteration Time Degradation versus the Number of Processors, Both Cache Sizes, Mapping Functions and Interconnection Networks, Square Decomposition.



a. Direct Mapping



b. Set-Associative Mapping

Figure 3.88 Iteration Time Degradation versus the Number of Processors, Both Cache Sizes, Mapping Functions and Interconnection Networks, Rectangular Decomposition.

3.4. Jacobi's Conclusions

The MR for the 64x64 point grid size decreases as the cache blocksize increases. This value also increases as the number of processors increase. The rectangular MR is generally larger for smaller blocksize; however, the converse becomes true as the blocksize increases. For grid sizes larger than 64x64 points the MR becomes prohibitive for the direct mapping strategy and marginally prohibitive for set-associative mapping. The MRD is present only for special cache features when considering the 128x128 point grid size. While the MRD is infinity for the 64x64 point grid size, intergrid and intragrid contention result in no MRD for the vast majority of the grid sizes and other features considered.

The IR generally decreases as the blocksize increases and it also increases as the number of processors increase (for a given blocksize). Like the MR, the IR for the rectangular domain decomposition strategy is larger than its square partition counterpart. This situation reverses as the blocksize increases. While intergrid and intragrid contention virtually eliminates the IR for grid sizes larger than 128x128 points when direct mapping is used, IR values do exist for the larger grid sizes under the set-associative mapping strategy. These values, however, are an order of magnitude lower than those of the smaller grid sizes. The IR generally increases as the number of processors increase for a given blocksize.

The prXICO is extremely small and remains constant for all grid sizes larger than 64x64 points when direct mapping is used. With the exception of the 2 processor system and the square domain decomposition, the prXICO decreases as the blocksize

decreases for the 64x64 point grid. A decrease in the frequency of intergrid and intragrid contention for the set-associative mapping strategy results in a higher prXICO for the larger grid sizes, however, the values are an order of magnitude lower than the smaller grid values. This probability also decreases as the blocksize decreases. The prXICO generally increases as the number of processors increase.

The prXICS increases as the blocksize increases for the square partition and direct mapping. In contrast, the value decreases as the blocksize increases for the rectangular partition. Both the square and rectangular decomposition strategies show a decrease in the prXICS as the blocksize increases for the set-associative mapping strategy. The prXICO also increases as the number of processors increase. The prefetching strategies considered have a detrimental effect on the MR for direct mapping. These same prefetching strategies serve to reduce the MR for the set-associative mapping strategy. The IR, prXICO and prXICS generally increase when prefetching is used.

Intergrid and intragrid contention results in higher iteration time degradation for the direct mapping policy. The single bus system also possesses a higher ITD than the full crossbar system, a direct result of a 0 waiting time lower bound for the full crossbar system. As the number of processors increase the ITD increases for the direct mapping strategy. This parameter also increases as the number of processors increase for set-associative mapping but at a slower rate.

In summary, using two PDE grids to execute this algorithm causes considerable coherence overhead for direct mapping strategy. The set-associative mapping strategy

results in relatively reduced overhead. The prefetching strategies simulated do not reduce this overhead. In many instances, prefetching results in greater performance degradation. The cache size generally has little effect on the performance parameters studied. Overall, the major causes of the performance degradation as a result of executing this algorithm, include the cache mapping strategy, the grid decomposition strategy, the cache blocksize and the number of processors used in the multiprocessor system.

CHAPTER 4

SUCCESSIVE OVER-RELAXATION

Although Jacobi's iterative algorithm is a natural for parallelization, its slow convergence rate results in the infrequent use of this algorithm. This convergence rate can be improved by the using the successive over-relaxation algorithm (SOR). This chapter commences by presenting a brief background discussion on this algorithm. The performance parameters obtained from the simulation results are then presented followed the multiprocessor speedup curves. Finally, the iteration time degradation is presented, followed by SOR conclusions.

4.1. Red-Black SOR

The convergence rate of Jacobi's algorithm may be accelerated by making a larger change to the $u_{i,j}$ grid point for each iteration. This change is made by the using the most recently updated values of the grid points (the Gauss-Seidel method) and by using a relaxation factor, ω , as shown in the equation below [3, 41]:

$$u_{i,j}^{k+1} = \left(1 - \omega\right)u_{i,j}^k + \frac{1}{4} \left[u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k \right] \quad (4.1)$$

While ω usually ranges from 1 to 2, there are numerous static and dynamic methods used to obtain the optimum value of this parameter. The simulations for this work assumes ω is obtained prior to the actual execution of SOR and remains constant throughout its duration.

The sequential nature of Equation 4.1 is such that attempts at parallelization does not produce consistent ordering (see [3] for an in depth discussion on this matter). If the PDE grid points are numbered by using the red-black method as shown in Figure 4.1 for a 4x4 grid size, then two passes over the grid, one for the red points and one for the black point result in acceptable ordering per iteration. Applying Equation 4.1 to this ordering results in the following equation for the first pass:

$$u_{i,j}^{k+1} = \left[1 - \omega\right] u_{i,j}^k + \frac{1}{4} \left[u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k \right] \text{ for all } i+j \text{ odd} \quad (4.2)$$

and for the second pass,

$$u_{i,j}^{k+1} = \left[1 - \omega\right] u_{i,j}^k + \frac{1}{4} \left[u_{i+1,j}^{k+1} + u_{i-1,j}^{k+1} + u_{i,j+1}^{k+1} + u_{i,j-1}^{k+1} \right] \text{ for all } i+j \text{ even} \quad (4.3)$$

.	.	.	.
B2	R4	B6	R8
.	.	.	.
R2	B4	R6	B8
.	.	.	.
B1	R3	B5	R7
.	.	.	.
R1	B3	R5	B7

Figure 4.1. Red-Black Ordering of 4x4 PDE Grid.

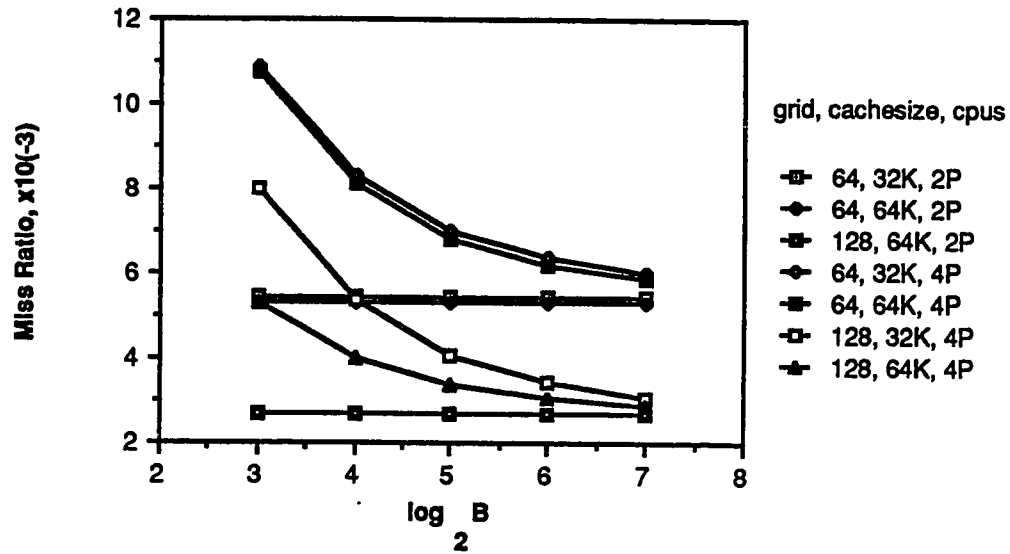
These equations show that updating each grid point consists of independent evaluations using grid points from previous passes. Therefore, these equations are easily parallelized and the same PDE grid decompositions used in the previous chapter are used for this algorithm. Also note that only one PDE grid is needed to update the points; however, two barrier synchronizations are needed for each iteration.

4.2. Simulation Results

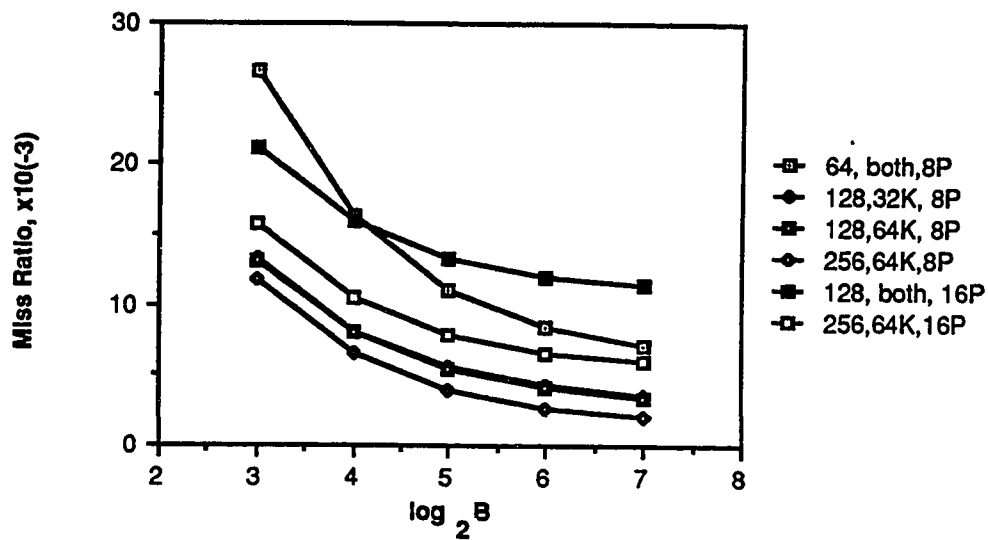
4.2.1. Miss Ratio/Miss Ratio Degradation

Figure 4.2 illustrates the MR versus the cache blocksize for small grid sizes, the square decomposition strategy, both cache sizes and all processors. The graph values are independent of the mapping function with a few exceptions discussed later. Part a of the figure shows this parameter for two and four processors. As a result of the principle of vertical shared blocks, the MR remains constant as the cache blocksize is varied for the 2 processor system. The MR discrepancies between the two cache sizes for the 64x64 point grid size and the 2 processor system are negligible. This is also true for the same grid size and the 4 processor system. The MR decreases as the cache blocksize increases when using 4 processors. Also note the decrease in the MR as the cache size doubles for the 128x128 point grid size and 4 processors. Part b of Figure 4.2 also shows a decrease in the MR as the cache blocksize decreases for eight and sixteen processors; however the MRs are somewhat larger for these processor configurations.

The MR for the 64x64 point grid size and the 16 processor system is not shown in Figure 4.2 because it changes dramatically as the cache size increases from 64 to



a. Two and Four Processors

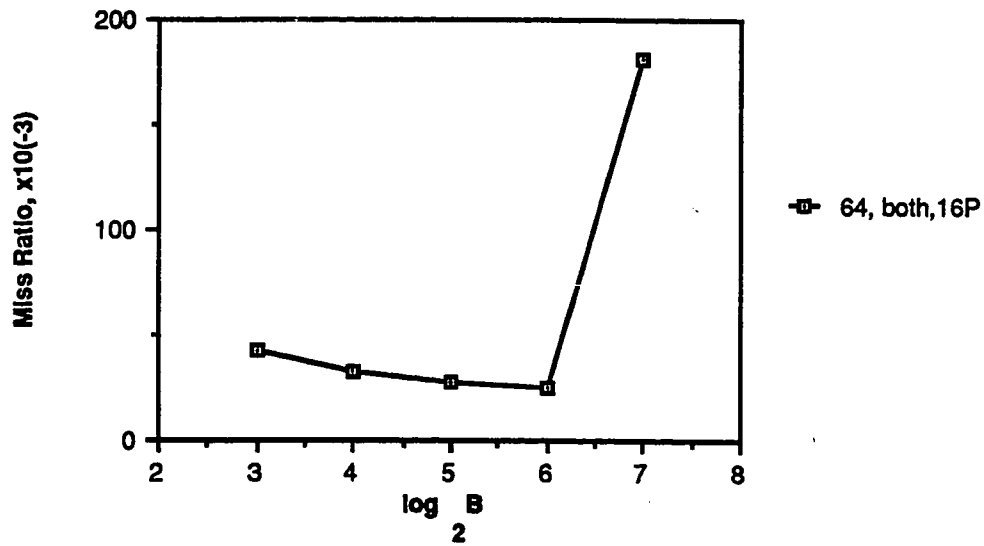


b. Eight and Sixteen Processors

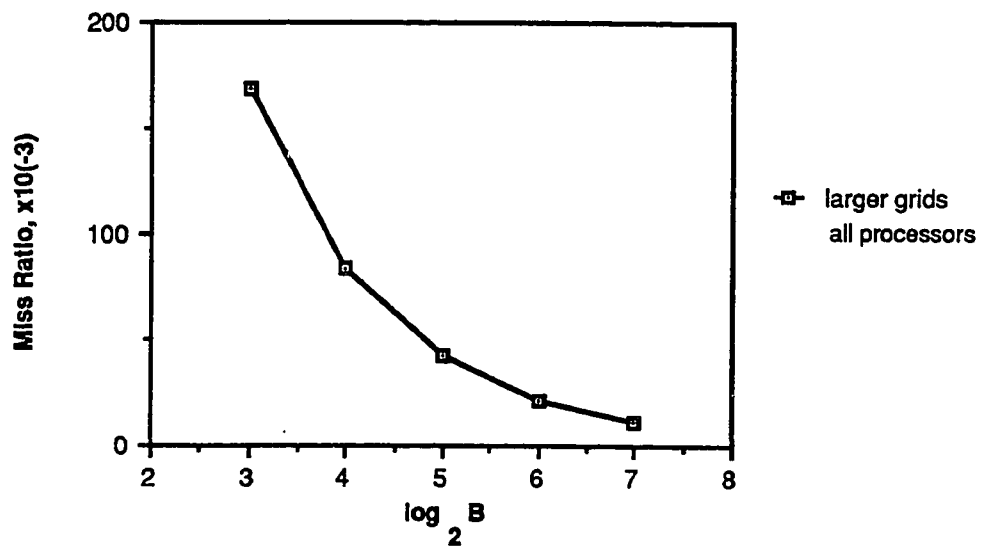
Figure 4.2 Miss Ratio versus Cache Blocksize, Square Decomposition, Small Grid Sizes, All Processors.

128 bytes. The MR for this grid/processor configuration is shown in part a of Figure 4.3. The parameter decreases as the cache blocksize increases from 8 to 64 bytes. The drastic increase in value for the 128 byte cache size is attributed to the modification of each cache block by two processors. The additional misses are a direct result of invalidations resulting from this two processor block modification. Part b of Figure 4.3 presents the MR versus the cache blocksize for the square decomposition strategy, all processors, both cache sizes and mapping functions (exceptions to discussed later), and all grid sizes not shown in Figure 4.2. While this MR also decreases as the cache blocksize increases, the value of this parameter is significantly higher than the MR for the smaller grid sizes. This occurs as a result of intragrid contention.

Figures 4.4 and 4.5 show the MR versus the cache blocksize for the same simulated features of the previous two figures with one exception; the rectangular decomposition strategy is used. Figure 4.4 presents the MRs for smaller grid sizes and 2-8 processors. This figure shows that the rectangular MR is generally larger than the square MR for smaller cache sizes. The reverse is true for the larger cache sizes. Also the rectangular MR curves have steeper slopes; a direct result of the effect of the principle of vertical shared blocks on the square MR. Figure 4.5a shows the rectangular MR for the grid sizes not shown in the previous figure and 2-8 processors. Since this figure is identical to Figure 4.5b the decomposition strategies have no effect on the MR of this algorithm for larger grid sizes. Part b of this figure shows the rectangular MR for the 16 processor system. While this MR is higher than all other MRs for smaller grid sizes, the rectangular MRs for the larger grid sizes are identical to the values presented for the other processor configurations.

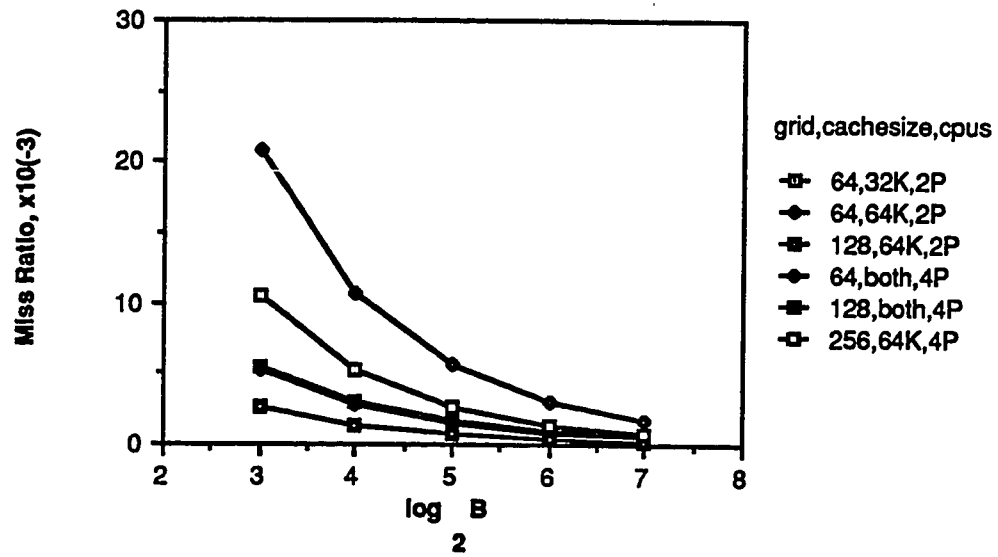


a. 64x64 Point Grid Size, 16 Processors, Both Cache Sizes

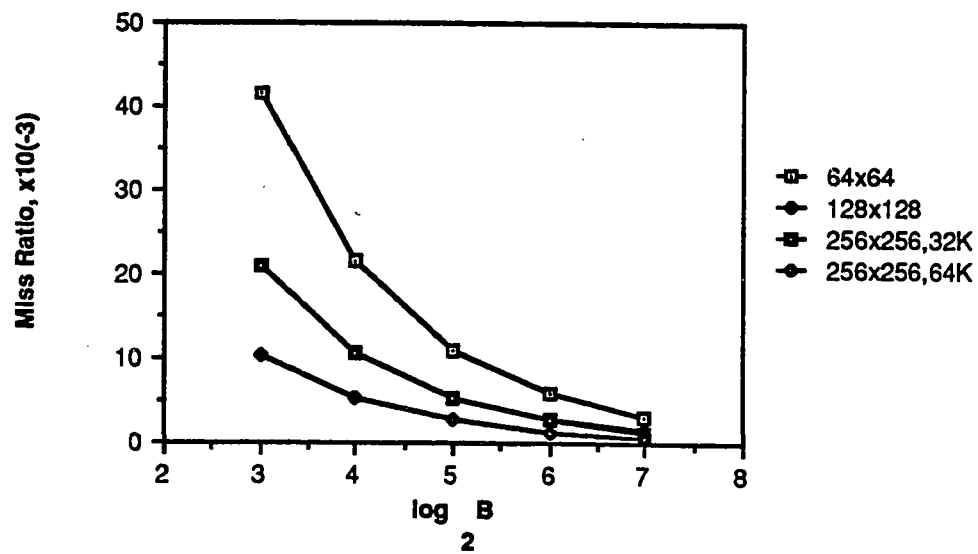


b. Larger Grid Sizes, All Processors, Both Cache Sizes

Figure 4.3 Miss Ratio versus Cache Blocksize, Square Decomposition, 64x64 Point Grid Size (a) and Larger Grid Sizes (b), All Processors.

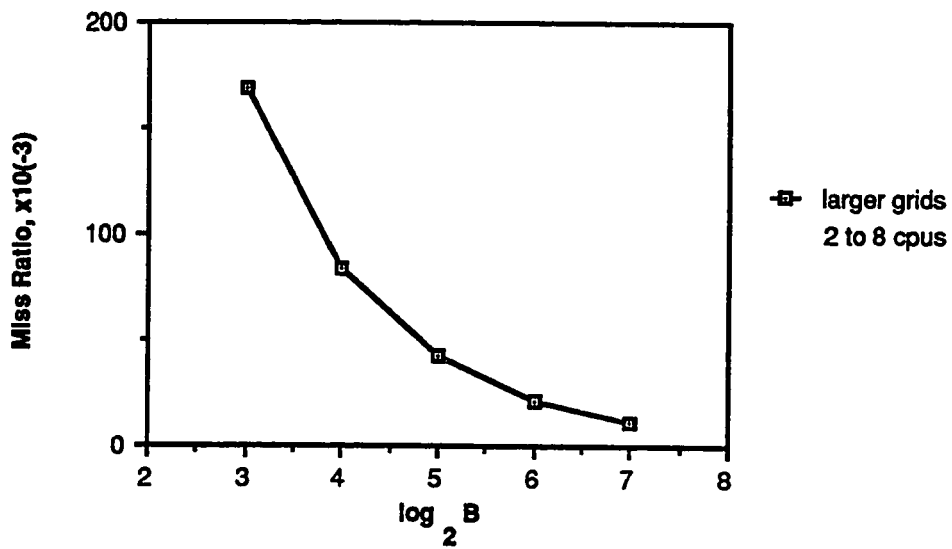


a. Two and Four Processors

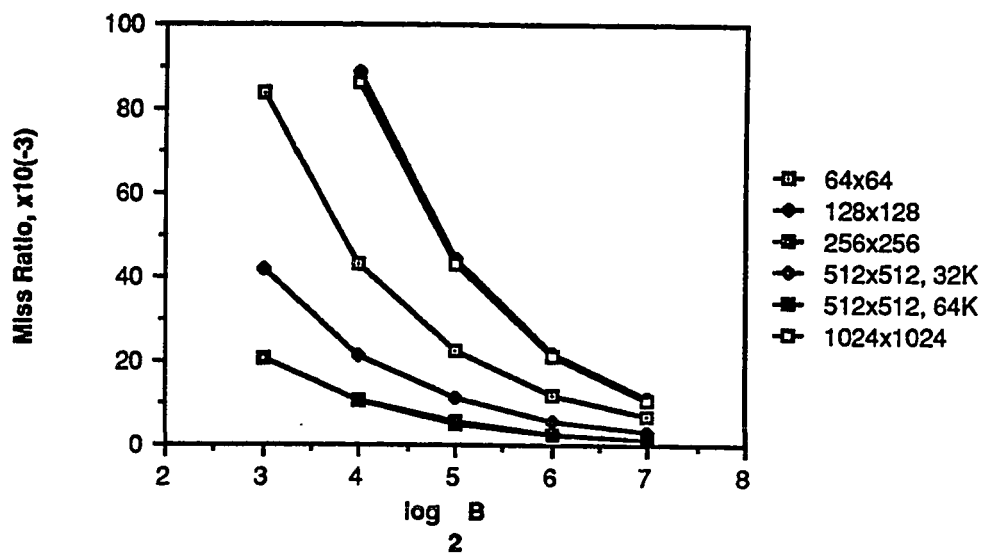


b. Eight Processors

Figure 4.4 Miss Ratio versus Cache Blocksize, Rectangular Decomposition, Small Grid Sizes, Both Cache Sizes, Two to Eight Processors.



a. Larger Grid Sizes, Two, Four and Eight Processors



b. Sixteen Processors

Figure 4.5 Miss Ratio versus Cache Blocksize, Rectangular Decomposition, All Grid Sizes and Processor Configurations, Both Cache Sizes.

Figure 4.6 illustrates the MR versus the number of processors for both the square and rectangular decomposition strategies, all grid sizes, both cache sizes considered and a 32-byte blocksize. Parts a and b of this figure show significant decreases in the MR as the cache blocksize is doubled. This results from a transition from some intragrid contention to little or no intragrid contention as the cache size increases. For the cache blocksize observed, there are very few differences between the MR of the two decomposition strategies.

Practically all of the MRs for set-associative mapping are identical to the corresponding direct mapping MRs shown in Figures 4.2 through 4.6. In fact, Figure 4.7 illustrates the few exceptions. Part a of the figure shows the MR curves for the square decomposition strategy. For all of the features shown, the direct mapping strategy results in a lower MR when compared with the set-associative mapping strategy. Also note that the discrepancies between the two MRs decrease as the cache blocksize increases. The reason for the larger set-associative MRs is that for the features indicated, the mapping structure coupled with the LRU replacement algorithm results in additional misses when compared with the direct mapping function. The rectangular decomposition strategy also exhibits a smaller direct mapping MR for the features presented in part b of the figure. There are no significant differences between the MRs versus the number of processors for the mapping functions studied.

Table 4.1 illustrates the MRD for both cache sizes and decomposition strategies as well as all processor configurations simulated for both direct (a) and set-associative (b) mapping. The table indicates that the 64x64 and 128x128 point grid sizes are small enough to produce a infinite MRD for this algorithm. In contrast, the

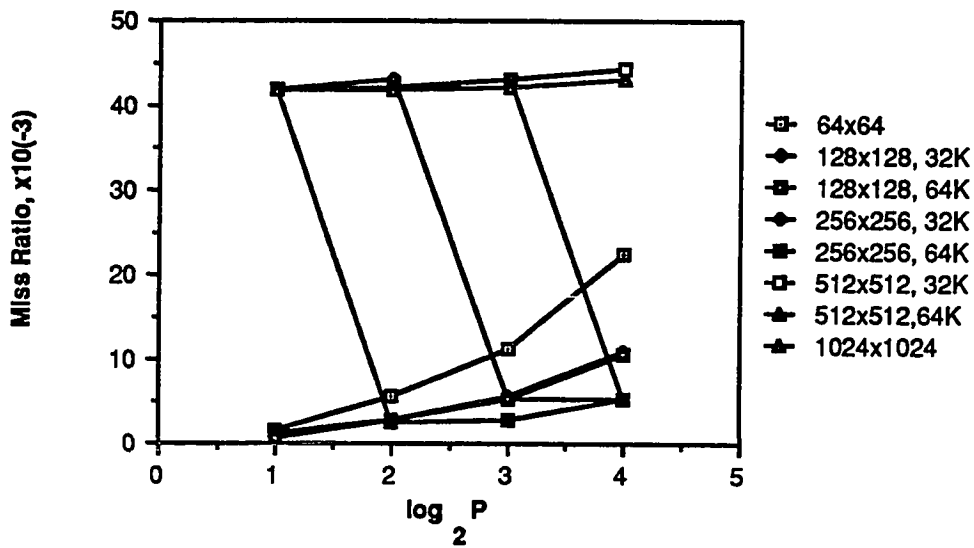
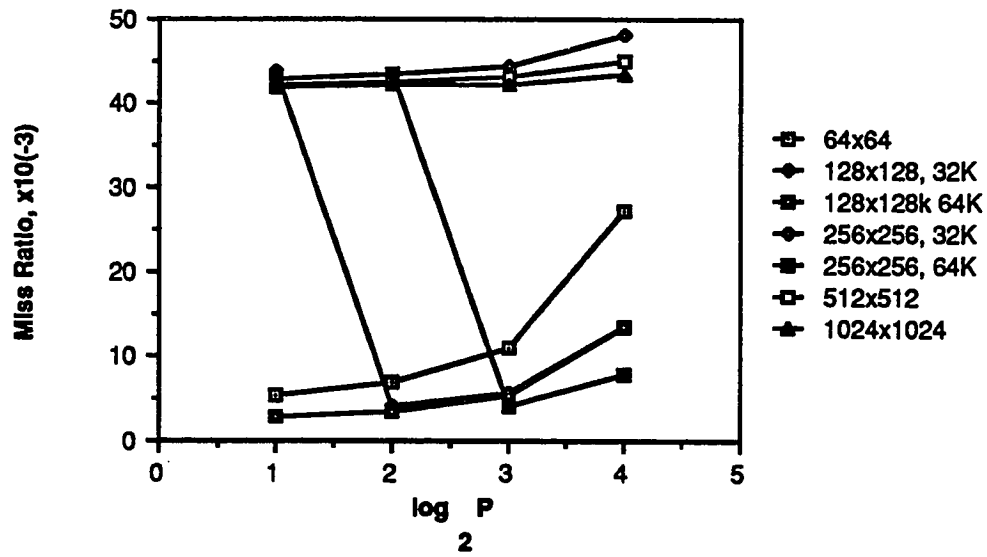
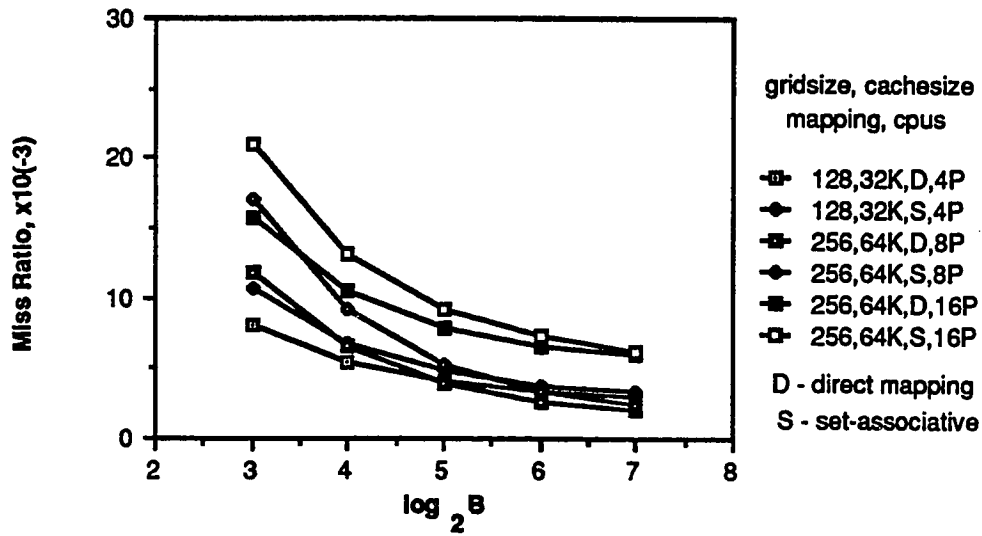
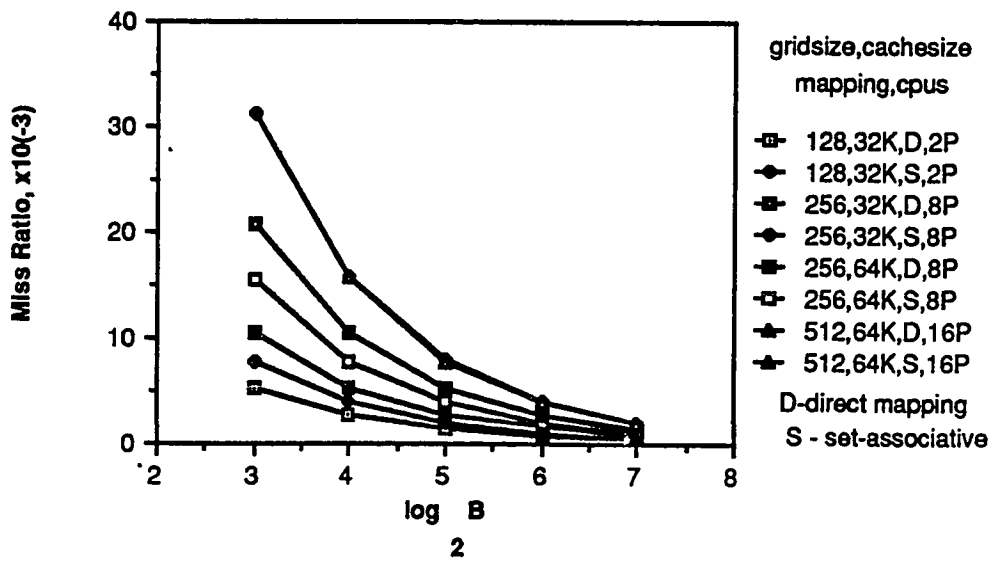


Figure 4.6 Miss Ratio versus the Number of Processors, Square and Rectangular Decompositions, 32-byte Blocksize.



a. Square Decomposition



b. Rectangular Decomposition

Figure 4.7 Comparisons of the Miss Ratio versus the Cache Blocksize, Direct and Set-Associative Mapping, Square and Rectangular Decomposition.

Cache Size	Partition	CPUs	PDE Gridsize				
			64	128	256	512	1024
32Kbytes	square	2	∞	1.00	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	1.00	1.00	1.00
		16	∞	∞	1.00	1.00	1.00
	rectangular	2	∞	9.04	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	17.07	1.00	1.00
		16	∞	∞	∞	1.00	1.00
64Kbytes	square	2	∞	∞	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	∞	1.00	1.00
		16	∞	∞	∞	1.00	1.00
	rectangular	2	∞	∞	1.00	1.00	1.00
		4	∞	∞	16.88	1.00	1.00
		8	∞	∞	∞	1.00	1.00
		16	∞	∞	∞	32.76	1.00

a. Direct Mapping

Cache Size	Partition	CPUs	PDE Gridsize				
			64	128	256	512	1024
32Kbytes	square	2	∞	1.00	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	1.00	1.00	1.00
		16	∞	∞	1.00	1.00	1.00
	rectangular	2	∞	6.34	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	11.67	1.00	1.00
		16	∞	∞	∞	1.00	1.00
64Kbytes	square	2	∞	∞	1.00	1.00	1.00
		4	∞	∞	1.00	1.00	1.00
		8	∞	∞	∞	1.00	1.00
		16	∞	∞	∞	1.00	1.00
	rectangular	2	∞	∞	1.00	1.00	1.00
		4	∞	∞	11.67	1.00	1.00
		8	∞	∞	∞	1.00	1.00
		16	∞	∞	∞	11.67	1.00

b. Set-Associative Mapping

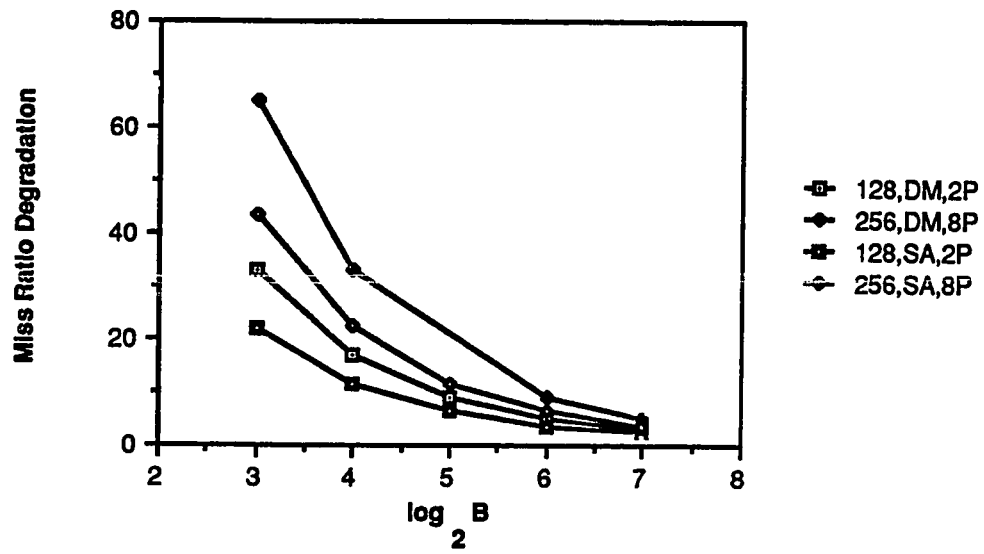
Table 4.1 Miss Ratio Degradation, 32-byte Blocksize.

larger grid sizes are such that no degradation in the MR is produced by parallelizing SOR. There are several instances where a non-infinite MRD exist. Figure 4.8 graphically presents these MRDs. Observe that only the rectangular decomposition strategy produces these MRDs.

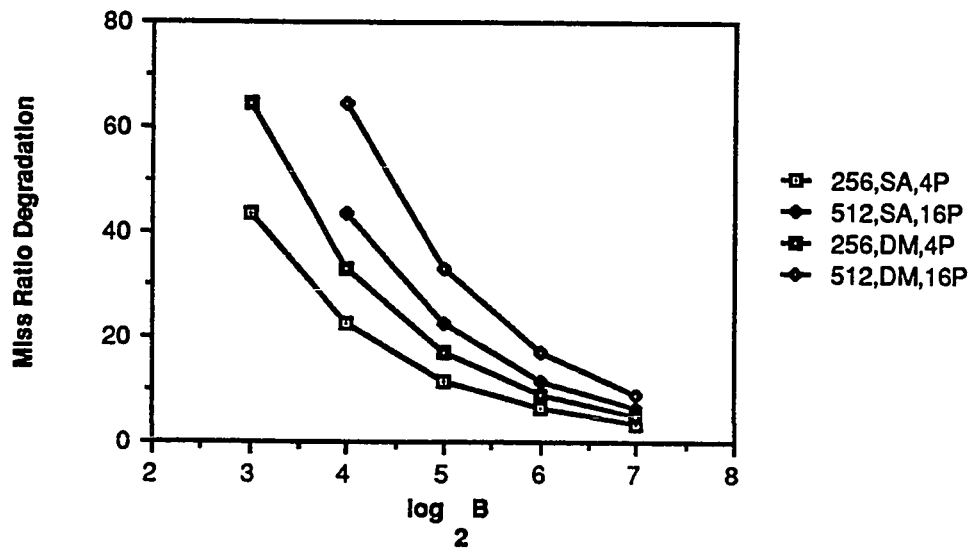
4.2.2. Invalidation Ratio

Figure 4.9 illustrates the IR versus the cache blocksize for both mapping strategies and cache sizes, all grid sizes, the square partition and two (a) and four (b) processors. The two processor IR has a constant value for each grid size considered, a result of the constant number of vertically adjacent shared blocks for this partition. The four processor system illustrates a decrease in the parameter as the cache blocksize increases. This decrease is more dramatic as the blocksize increases from 8 to 64 bytes. For blocksizes greater than 64 bytes the IR differences are negligible, especially for the larger grid sizes. The phenomenon is a direct result of the vertically adjacent shared blocks for the square decomposition strategy. Also observe that the four processor system has IR approximately twice the two processor values for the smaller blocksizes. As the blocksize increases the four processor values asymptotically approach the two processor IRs.

Figure 4.10 presents the IR with all the features of the previous figure for eight (a) and sixteen (b) processors. The graphs indicate an increase in the IR as the number of processors increase. The general decrease in value as the cache blocksize increases is present for both processor systems shown in the figure. Observe that the IR for the 64x64 grid point, the 128-byte blocksize and the sixteen processor system

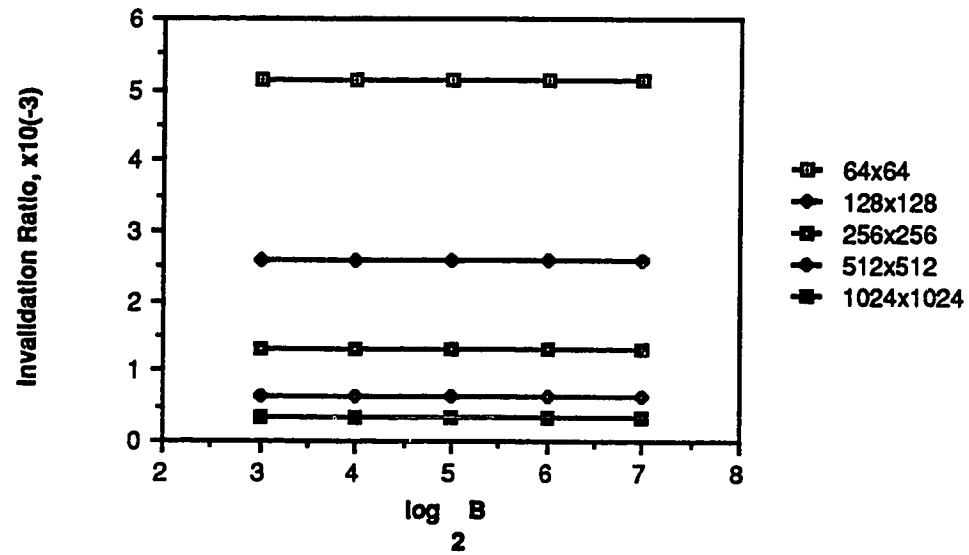


a. 32Kbyte Cache

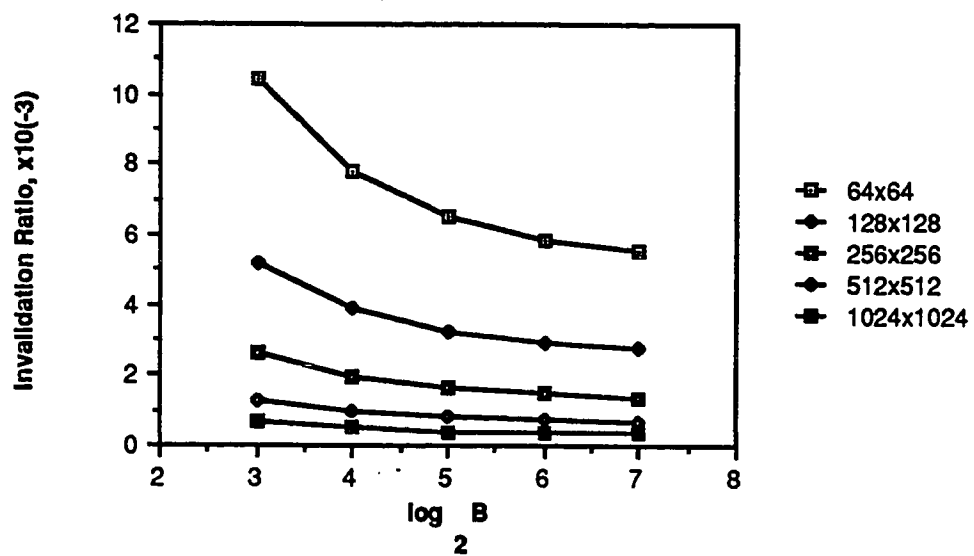


b. 64Kbyte Cache

Figure 4.8 Miss Ratio Degradation versus Cache Blocksize, Rectangular Decomposition Strategy, Both Cache Sizes.

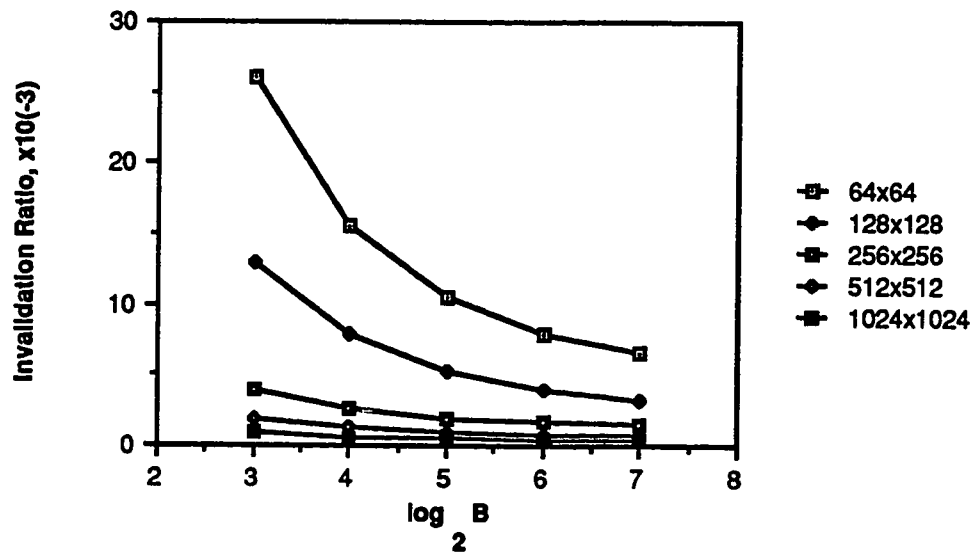


a. Two Processors

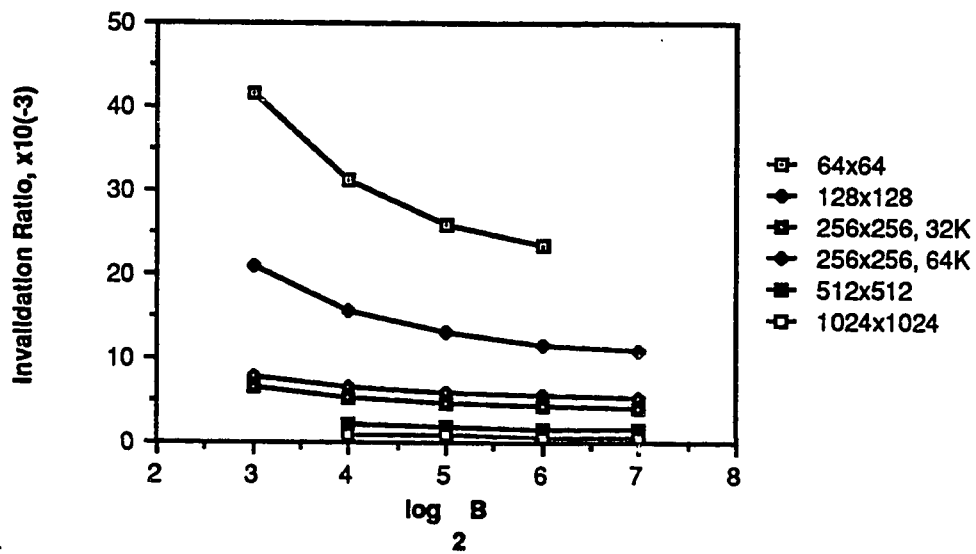


b. Four Processors

Figure 4.9 Invalidation Ratio versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Square Decomposition Strategy, Two and Four Processors.



a. Eight Processors



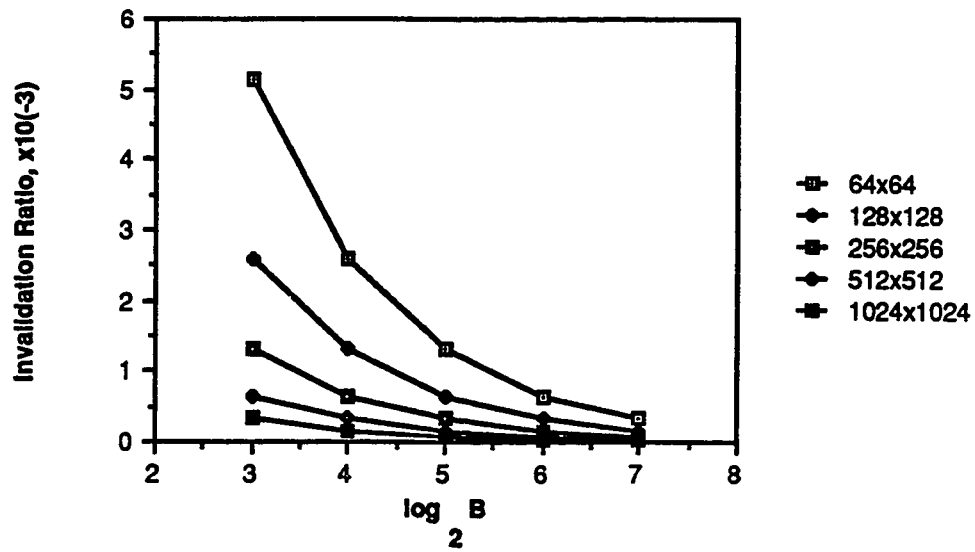
b. Sixteen Processors

Figure 4.10 Invalidation Ratio versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Square Decomposition Strategy, Eight and Sixteen Processors.

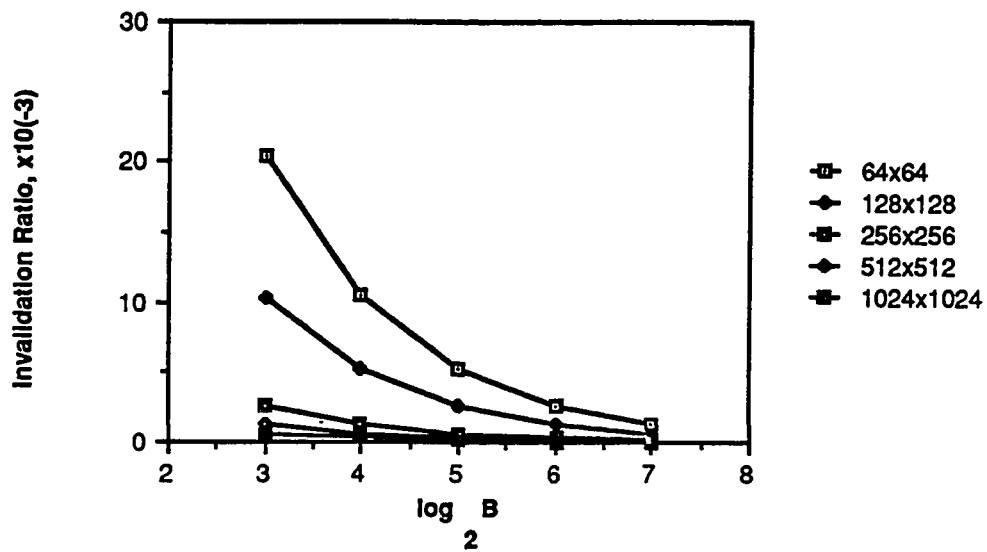
is not shown. This is because one block is "allocated" to two processors for this partition, significantly increasing the IR by two orders of magnitude. Viewing part b of the figure one notes the slight increase in the IR for the 16 processor 256x256 point grid size and 32Kbyte cache size when compared to the 64Kbyte value. The square decomposition strategy for this grid size is such that more shared blocks can be placed in the 64Kbyte cache over a longer reference range and future invalidations are therefore possible.

Figures 4.11 and 4.12 illustrate the IR versus the cache blocksize for the rectangular decomposition strategy and all other simulated cache features and grid sizes for two (4.11a), four (4.11b), eight (4.12a), and sixteen (4.12b) processors. These graphs show a decrease in the IR as the cache blocksize increases for all processor configurations. Also observe that the slopes of the curves are steeper when compared to the square decomposition strategy. The rectangular IRs are larger than the square IRs for smaller blocksizes; however, the converse is true for the smaller blocksizes. The 256x256 point grid size and the eight processor system (Figure 4.12) shows a slightly smaller IR for the 32Kbyte cache size when compared to the larger cache sizes for the same reasons discussed previously.

Figure 4.13 shows the IR versus the number of processors for all grid sizes and cache sizes considered, the 32byte cache blocksize and the square (a) and rectangular (b) decomposition strategies. Both figures show an intuitive increase in the IR as the number of processors increase. The square IRs are higher for two and four processors while the rectangular IRs are higher for the larger processor systems. Also note that for grid sizes larger than 256x256 points the IR is virtually constant for two to eight

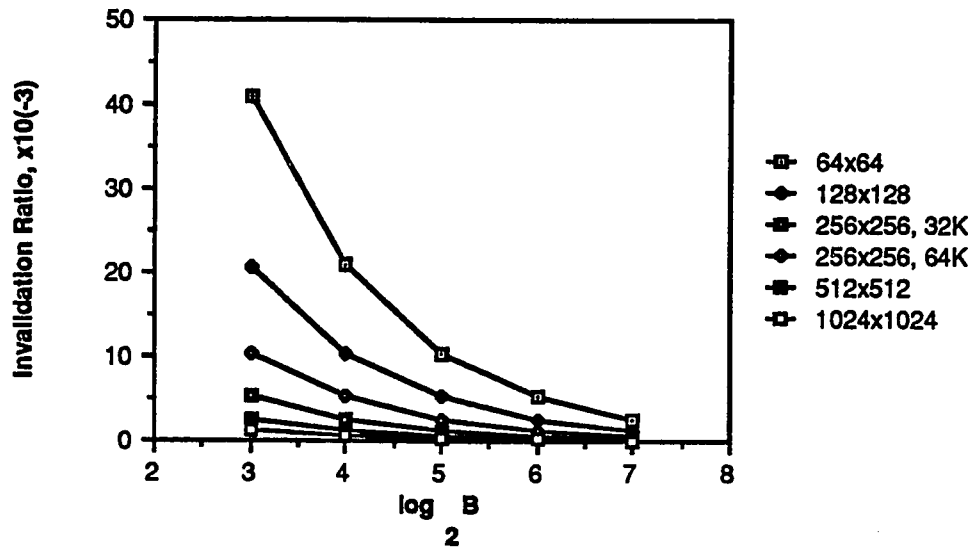


a. Two Processors

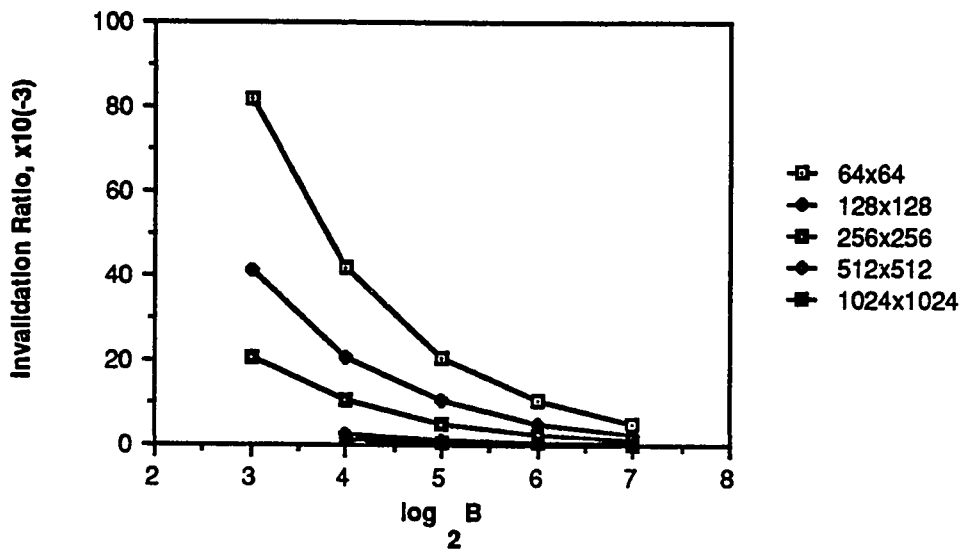


b. Four Processors

Figure 4.11 Invalidation Ratio versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Rectangular Partition, Two and Four Processors.

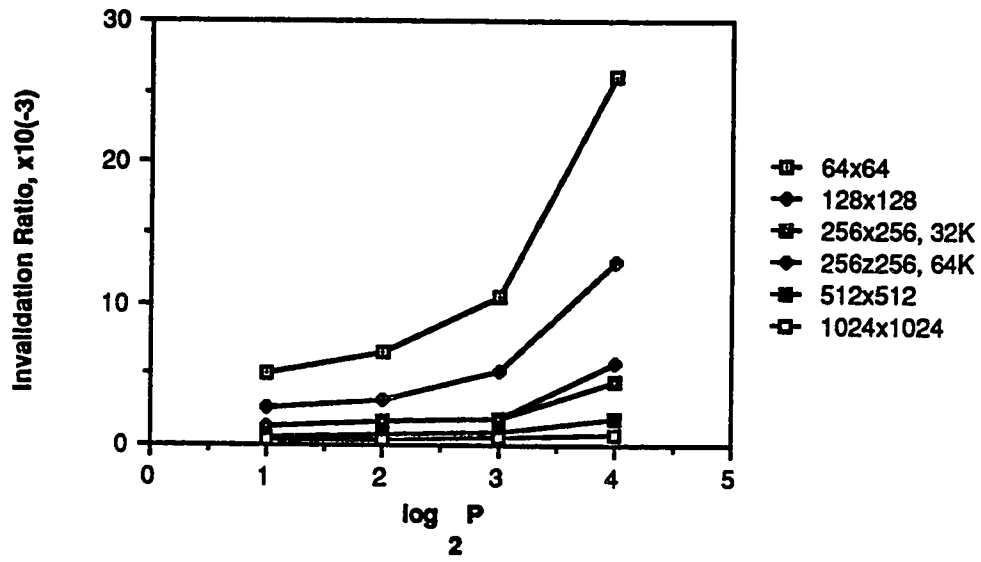


a. Eight Processors

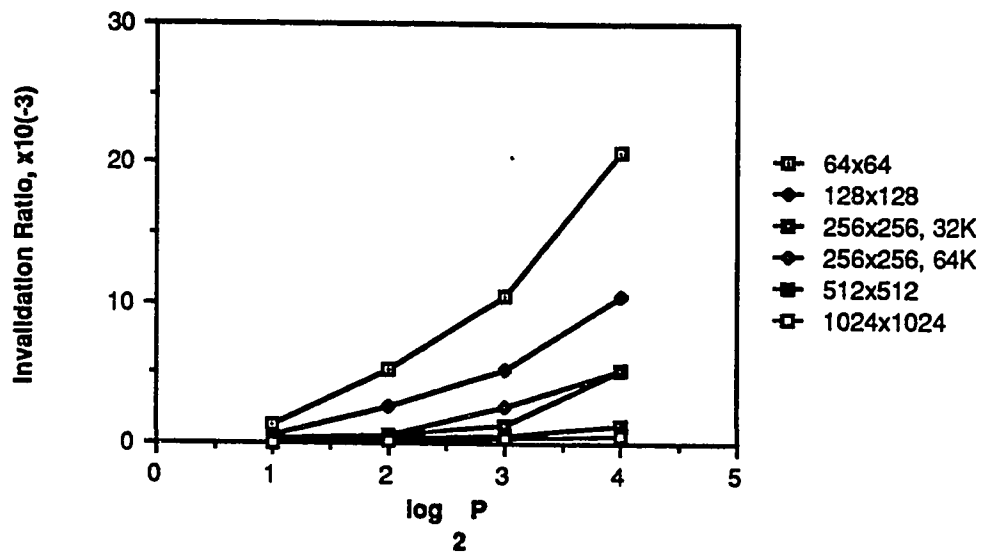


b. Sixteen Processors

Figure 4.12 Invalidation Ratio versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Rectangular Partition, Eight and Sixteen Processors.



a. Square Decomposition Strategy



b. Rectangular Decomposition Strategy

Figure 4.13 Invalidation Ratio versus the Number of Processors, Both Mapping Strategies and Cache Sizes, 32-byte Cache Blocksize.

processors with a slight increase for the 16 processor system.

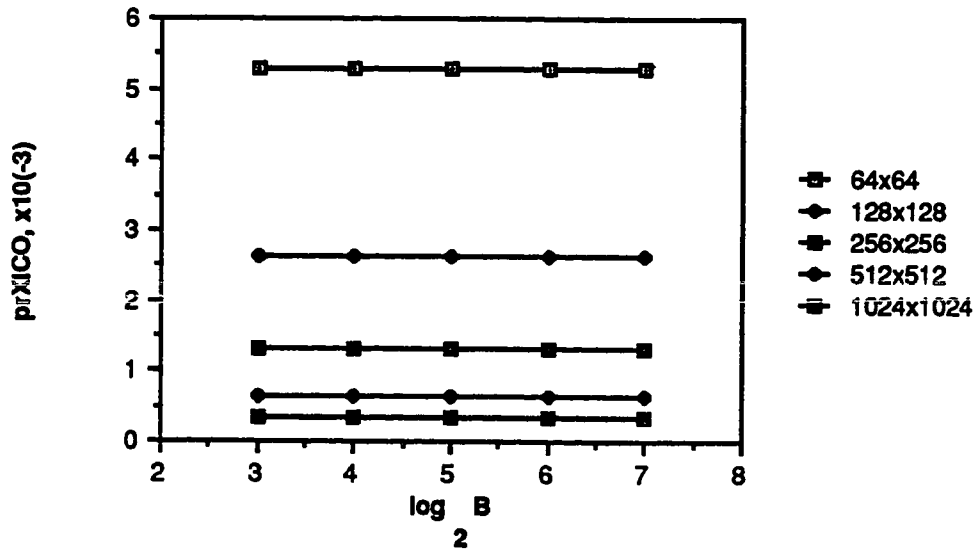
4.2.3. Probability of a XICO

Figure 4.14 illustrates the prXICO versus the cache blocksize for both mapping strategies and cache sizes, the square partition and two (a) and four (b) processors. Part a of the figure shows that the prXICO is independent of the cache blocksize for a two processor system. However, this parameter decreases as the cache blocksize increases for the four processor system. The decrease also occurs for eight and sixteen processor systems as shown in Figure 4.15.

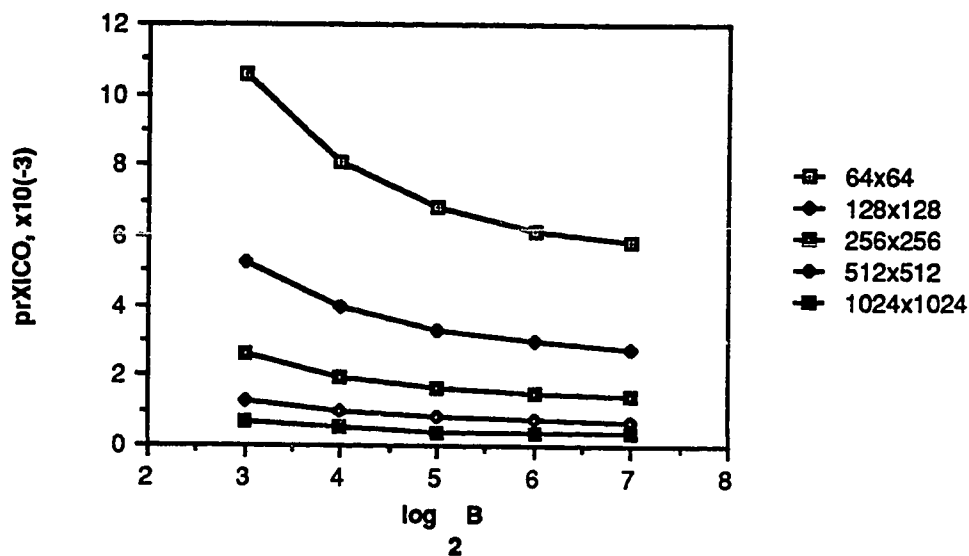
Figures 4.16 and 4.17 present the prXICO versus the cache blocksize for the rectangular partition, both mapping strategies and cache sizes considered and all processor configurations. All graphs show a decrease in this parameter as the cache blocksize increases. Finally, Figure 4.18 illustrates the prXICO versus the number of processors for both mapping strategies and cache sizes, a 32-byte cache blocksize and the square (a) and rectangular (b) partitions. While the eight and sixteen processor prXICOs are independent of the decomposition strategy, the rectangular partition produces slightly smaller XICOs than the square partition for two and four processor systems. The 256x256 (both partitions) and 512x512 (rectangular partition only) point grid sizes have slightly higher prXICOs for the 64Kbyte cache size.

4.2.4. Probability of a XICS

Figure 4.19 represents the prXICS versus the cache blocksize for the square decomposition strategy, both cache sizes and mapping strategies and two and four processor systems. This parameter is also independent of the cache blocksize for the

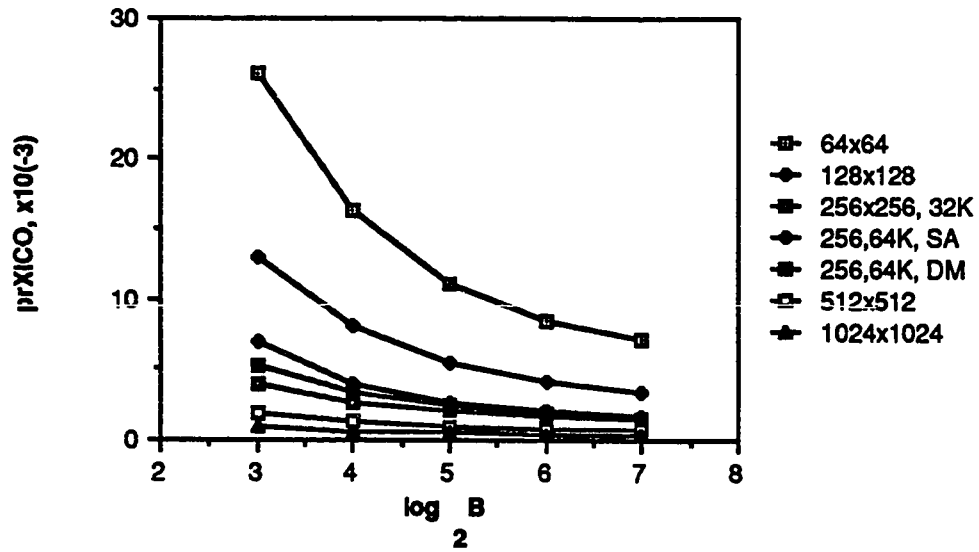


a. Two Processors

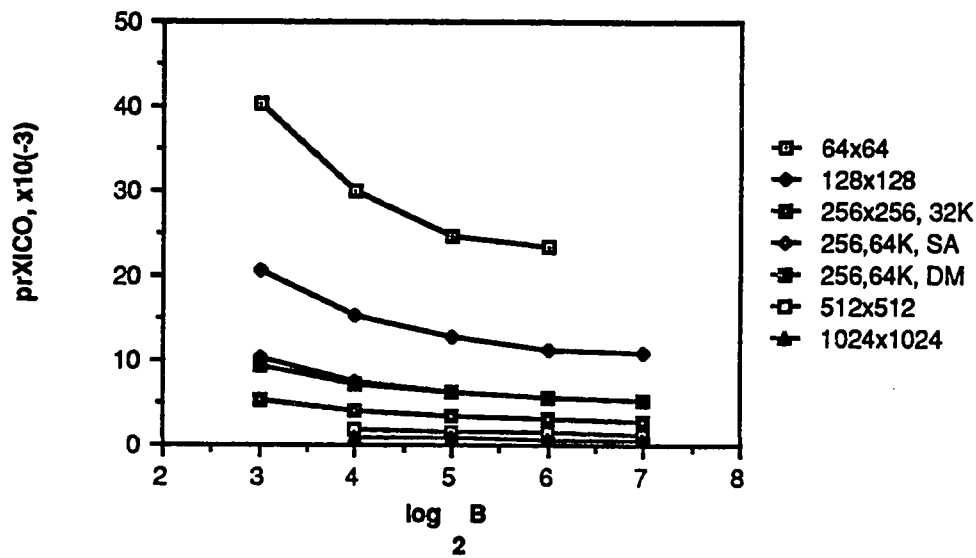


b. Four Processors

Figure 4.14 prXICO versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Square Decomposition, Two and Four Processors.

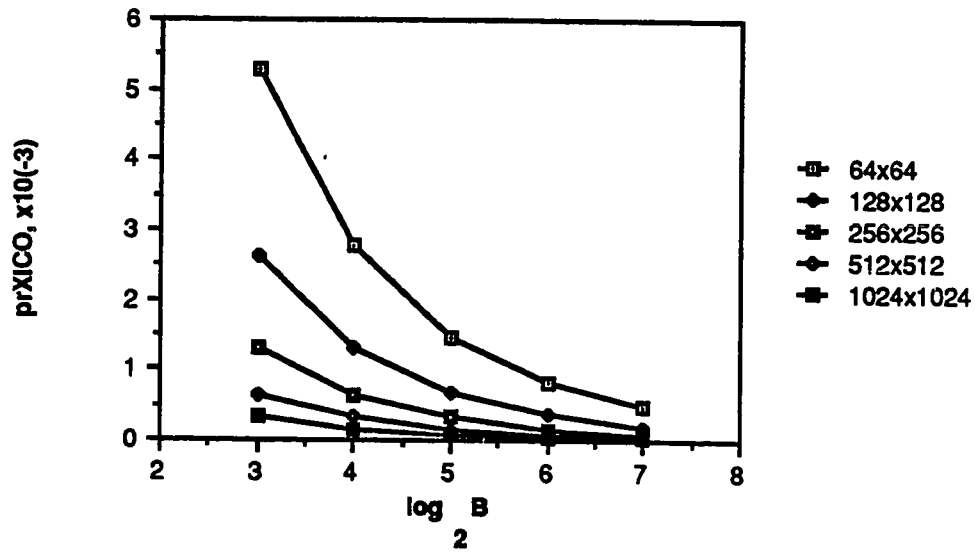


a. Eight Processors

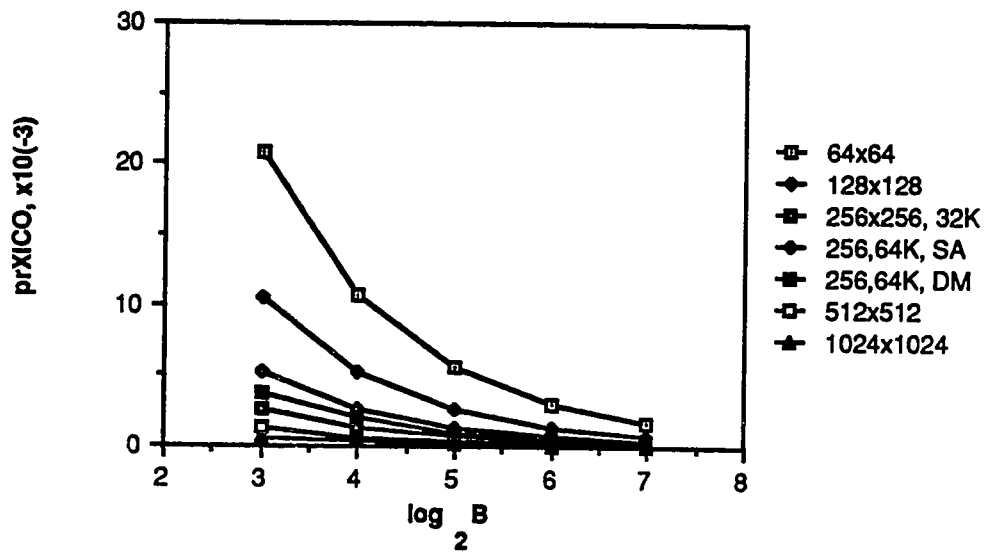


b. Sixteen Processors

Figure 4.15 prXICO versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Square Decomposition, Eight and Sixteen Processors.

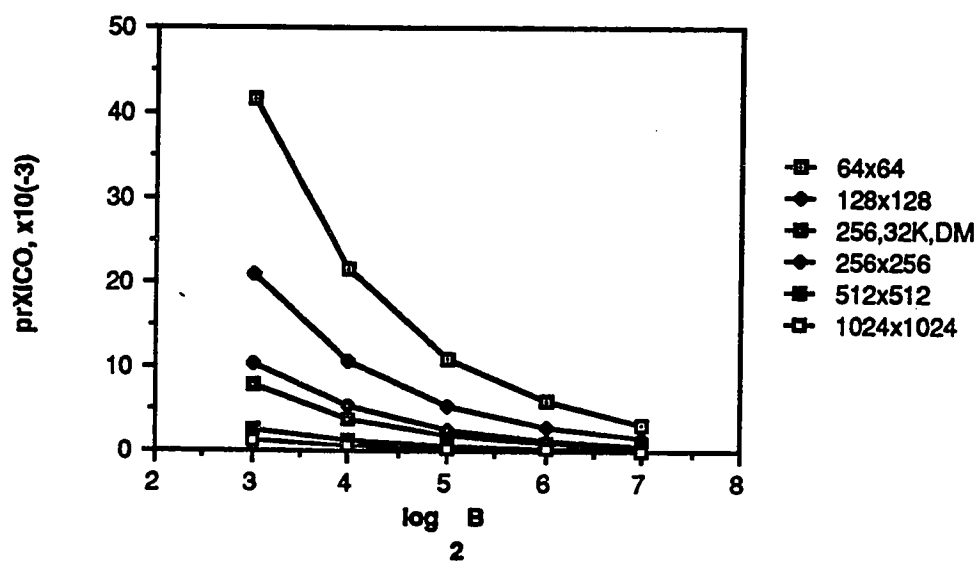


a. Two Processors

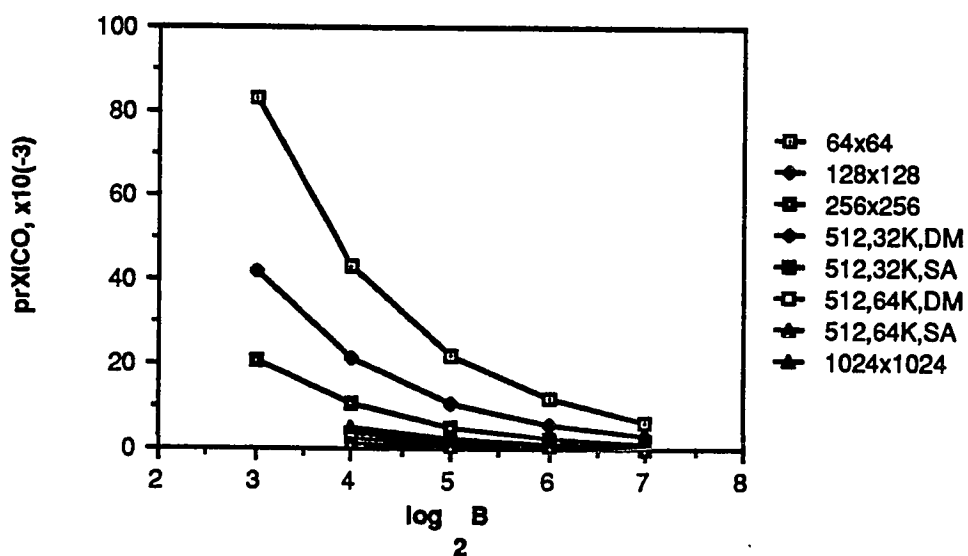


b. Four Processors

Figure 4.16 prXICO versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Rectangular Decomposition, Two and Four Processors.

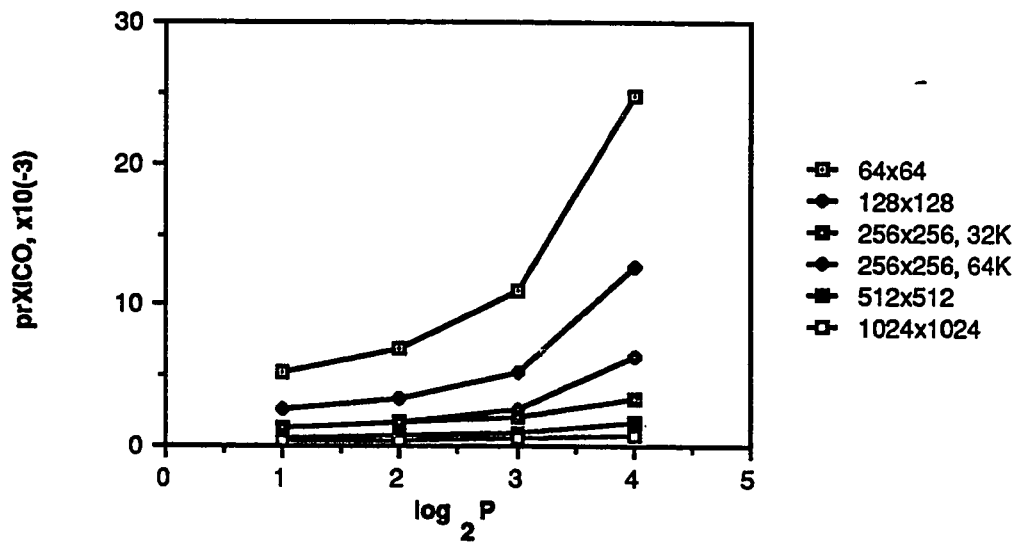


a. Eight Processors

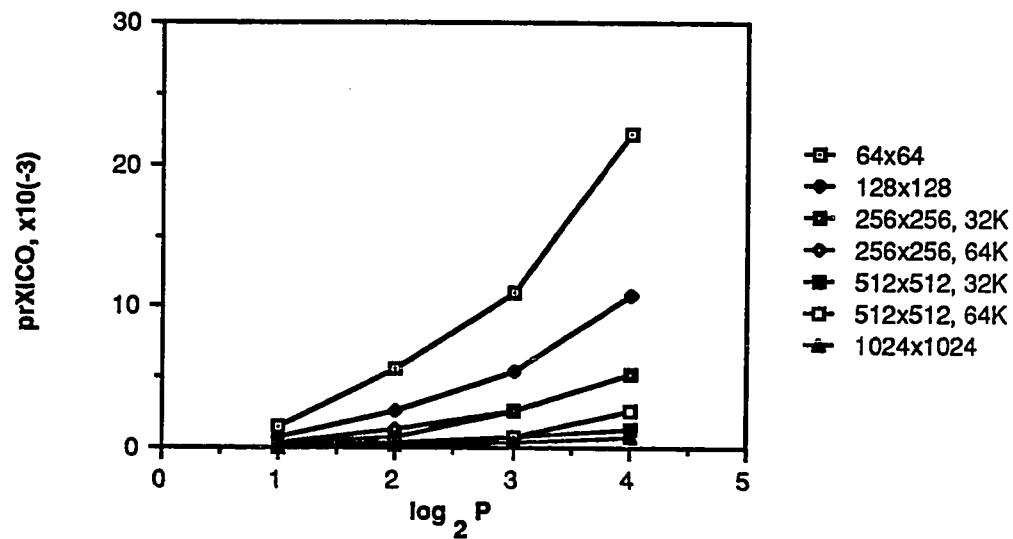


b. Sixteen Processors

Figure 4.17 prXICO versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Rectangular Decomposition, Eight and Sixteen Processors.



a. Square Decomposition



b. Rectangular Decomposition

Figure 4.18 prXICO versus the Number of Processors, Both Mapping Strategies and Cache Sizes, 32-byte Cache Blocksize.

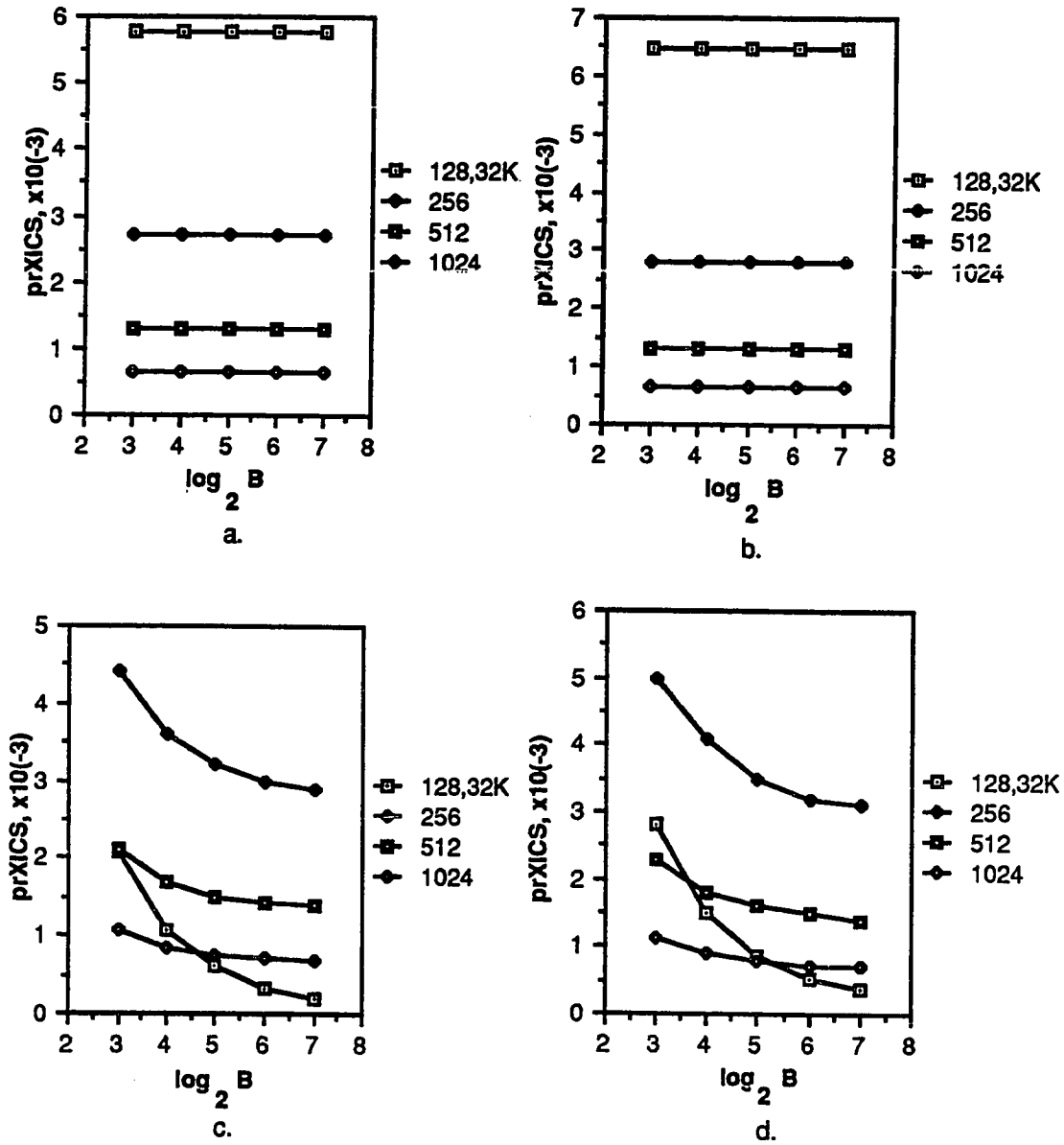


Figure 4.19 prXICS versus Cache Blocksize, Both Cache Sizes, Square Decomposition; a) Two Processors, Direct Mapping, b) Two Processors, Set-Associative Mapping, c) Four Processors, Direct Mapping, and d) Four Processors, Set-Associative Mapping.

two processor system. Also the set-associative value is larger than its direct mapping counterpart for the 128x128 point grid sizes (two processors). The same observation is made for the four processor system for 256x256 and 512x512 point grid sizes also; however, the parameter decreases as the cache blocksize increases.

Figure 4.20 offers the same parameters and variable features for eight and sixteen processor systems. This parameter decreases as the cache blocksize increases for all grid sizes except the 64x64 point size (direct mapping and 32Kbyte cache size only) where the value is independent of the cache blocksize. Also observe that for the 256x256 point grid size the XICS probability is higher for the 32Kbyte cache size. This is because more RO blocks are replaced for this cache size resulting in more RO->EXs occurring. Furthermore, the set-associative mapping strategy produces a higher prXICS than the direct mapping strategy for both processor configurations.

Figures 4.21 and 4.22 represent the prXICS versus the cache blocksize for both cache sizes and mapping strategies, the rectangular decomposition strategy and all processor configurations considered. All graphs show sharp decreases in value as the cache blocksize increases from 8 to 32 bytes and relatively smaller decreases in value for block sizes larger than 32 bytes. Again, the set-associative prXICSs are larger than the direct mapping values. Likewise, the rectangular prXICSs are larger than their square counterparts.

Figure 4.23 illustrates the prXICS versus the number of processors for both decomposition strategies and mapping strategies and a 32 byte blocksize. These curves increase in value and the number of processors increase for grid sizes larger

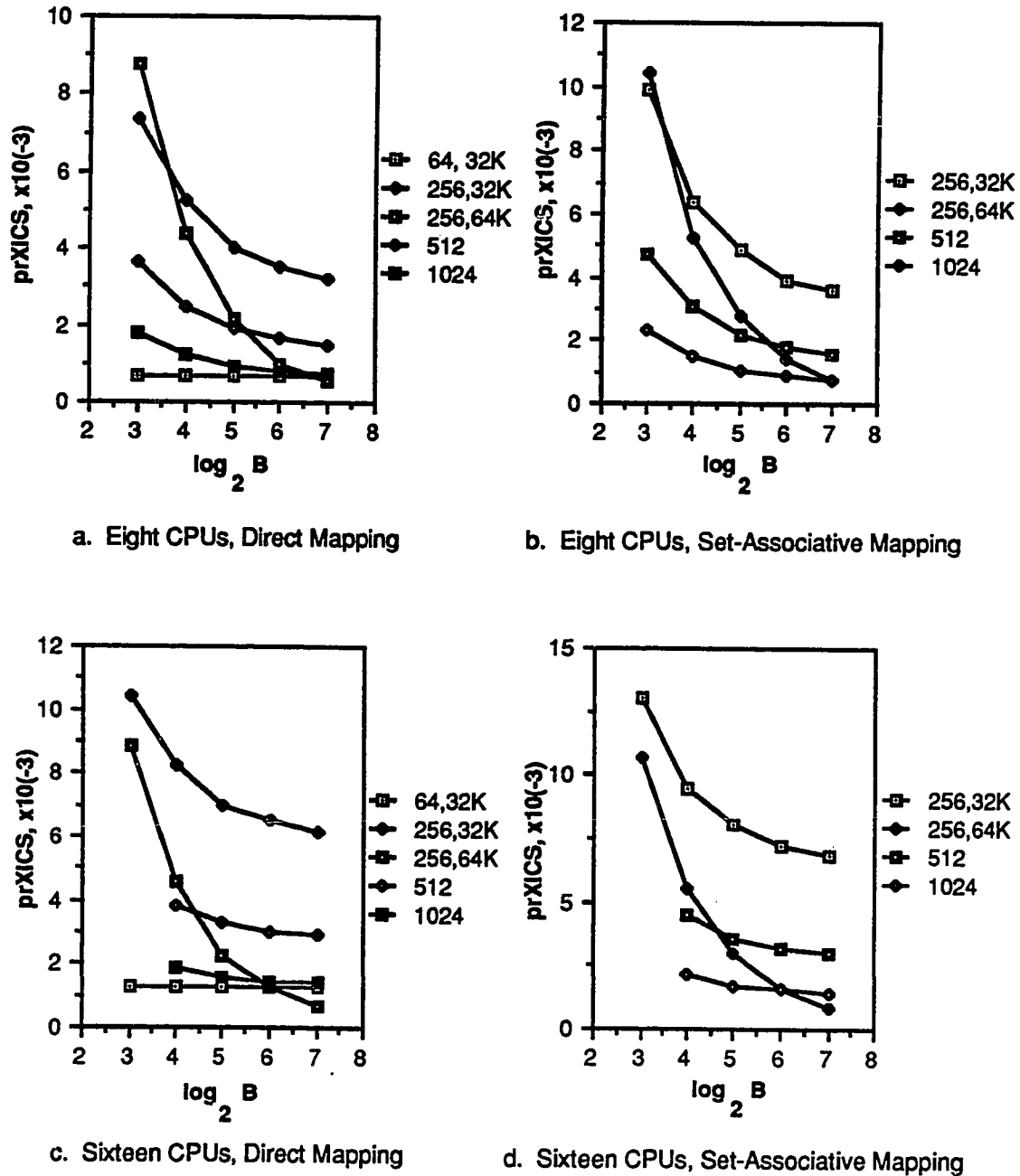
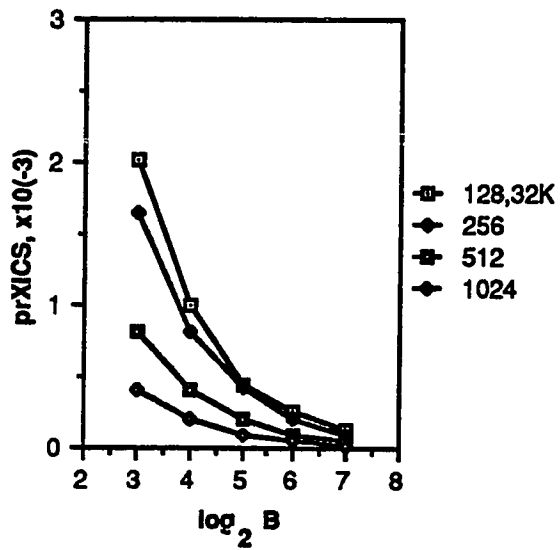
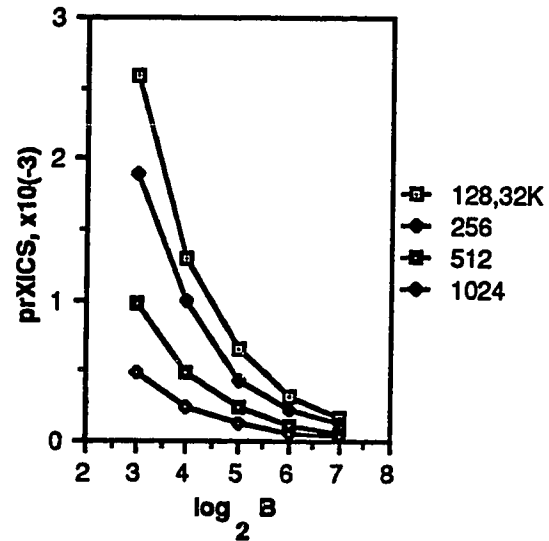


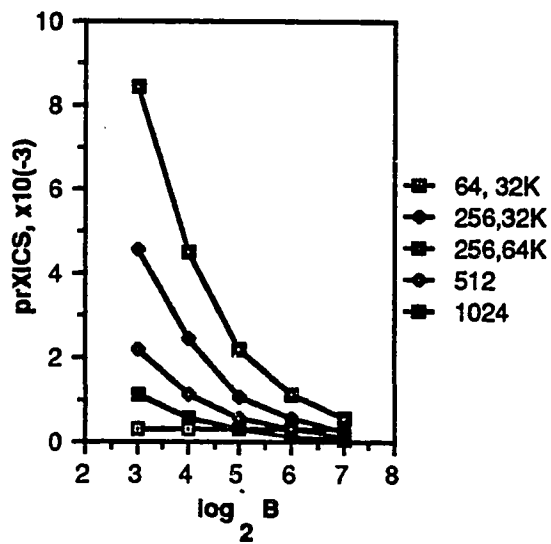
Figure 4.20 prXICS versus Cache Blocksize, Both Cache Sizes, Square Decomposition.



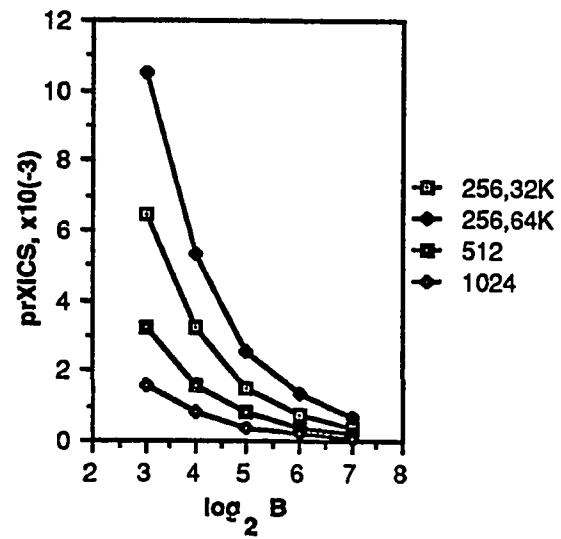
a. Two CPUs, Direct Mapping



b. Two CPUs, Set-Associative Mapping

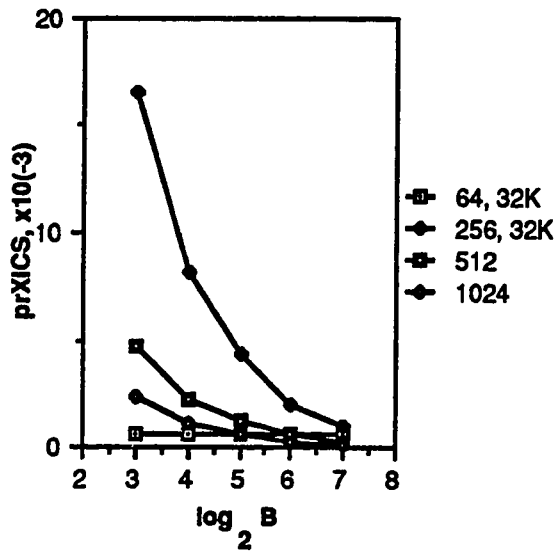


c. Four CPUs, Direct Mapping

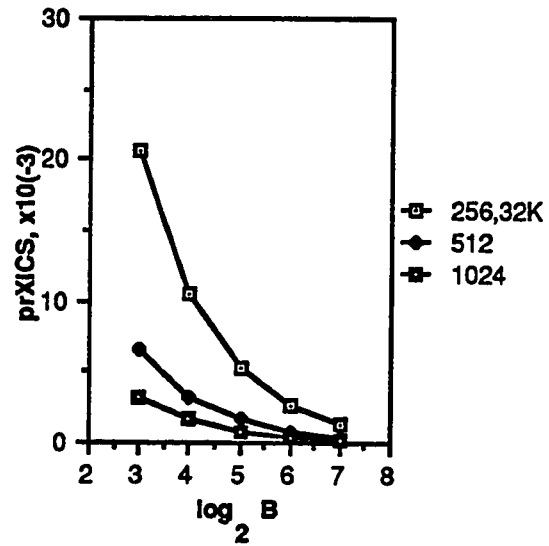


d. Four CPUs, Set-Associative Mapping

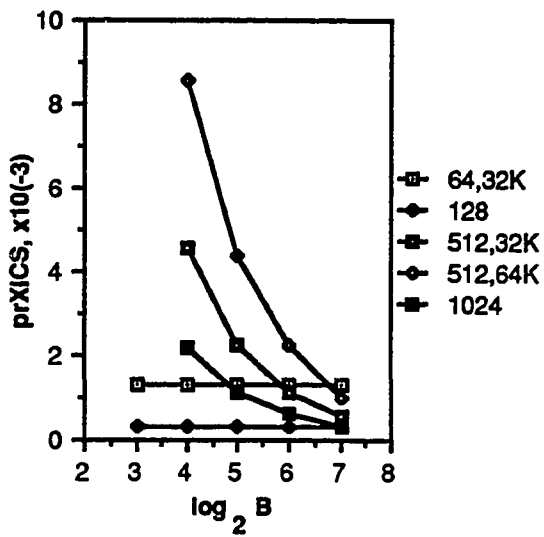
Figure 4.21 prXICS versus Blocksize, Both Cache Sizes, Rectangular Decomposition.



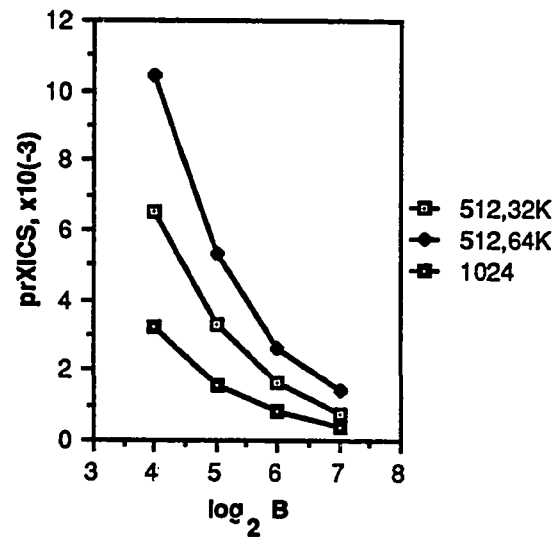
a. Eight CPUs, Direct Mapping



b. Eight CPUs, Set-Associative Mapping

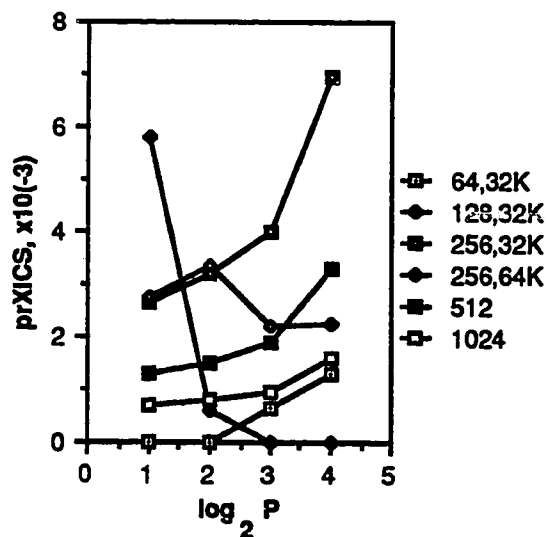


c. Sixteen CPUs, Direct Mapping

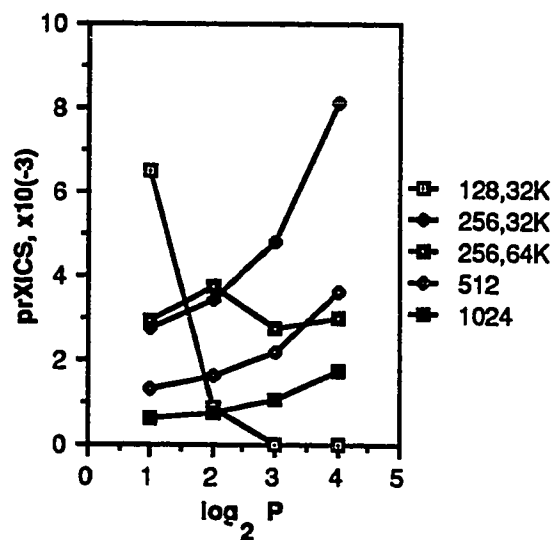


d. Sixteen CPUs, Set-Associative Mapping

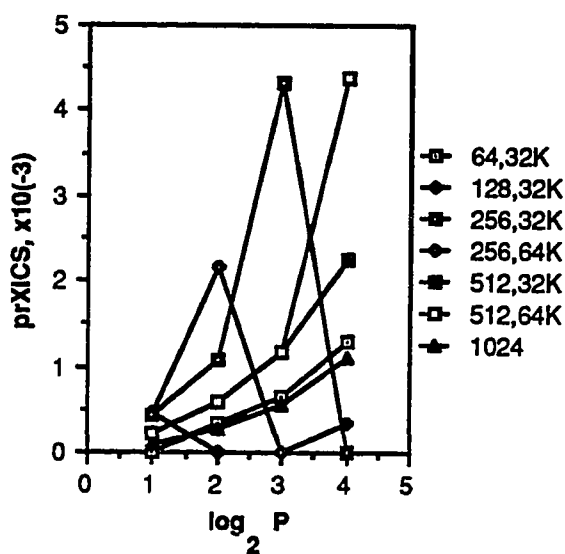
Figure 4.22 prXICS versus Blocksize, Both Cache Sizes, Rectangular Decomposition.



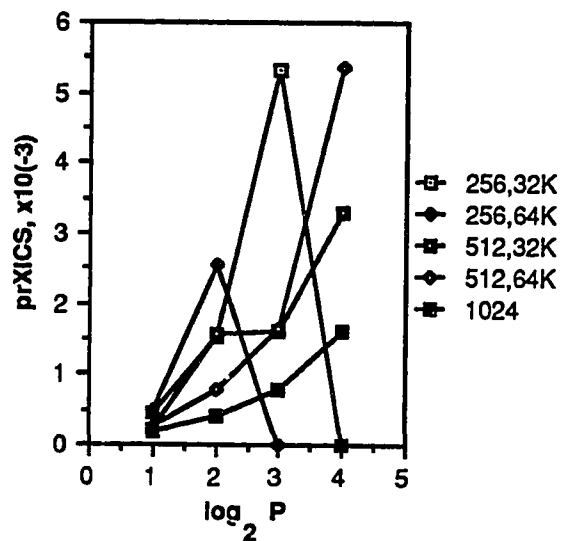
a. Square Partition, Direct Mapping



b. Square Partition, Set-Associative Mapping



c. Rectangular Partition, Direct Mapping



d. Rectangular Partition, Set-Associative Mapping

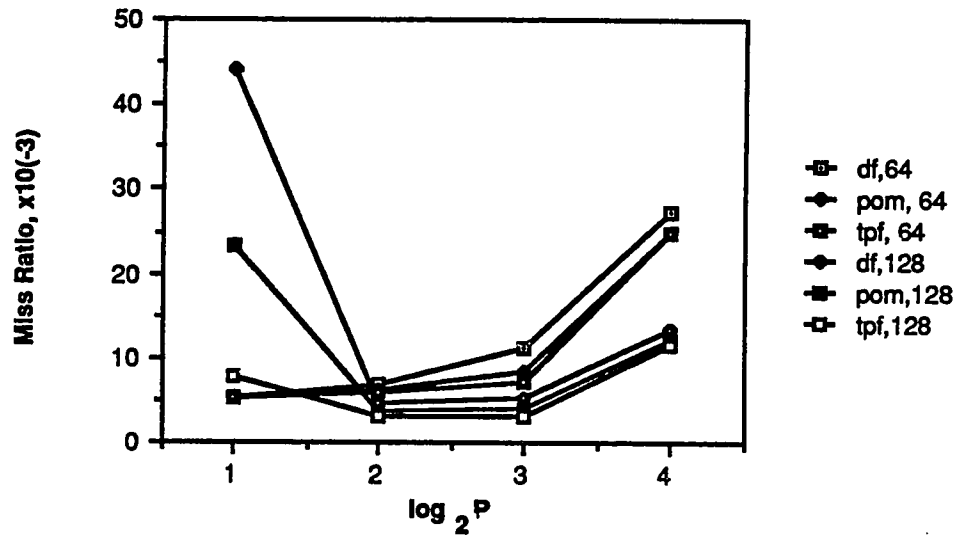
Figure 4.23 prXICS versus the Number of Processors, 32-byte Blocksize.

than 256x256 points. However, the locality of references for the 128x128 and 256x256 (rectangular partition only) point grid sizes are such that nonuniform behavior is exhibited for the prXICS at these grid sizes. The parameter decreases as the cache blocksize increases for the 128x128 point grid size and it increases, reaches its peak, and then decreases in value for the 256x256 point grid size. This is caused by the fact that the locality of references for the grid size/cache size tuple varies in such a way as to produce the behavior shown.

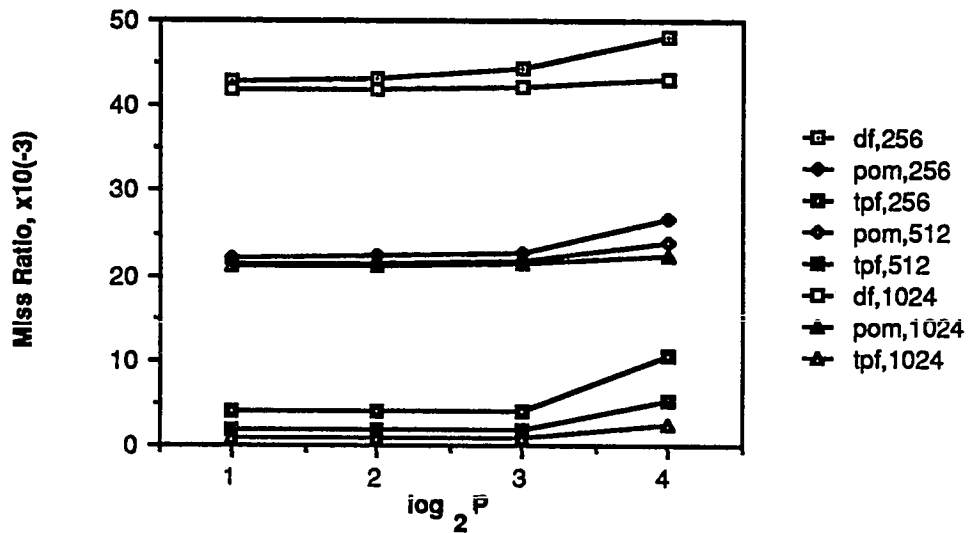
4.2.5. Prefetching Strategies

Figure 4.24 offers the MR versus the number of processors for all fetching strategies considered, the 32Kbyte cache size, all grid sizes, the square decomposition strategy, both mapping functions and a 32-byte blocksize. For grid sizes smaller than 256x256 points, the MR increases as the number of processors increase with demand fetching exhibiting the worst performance and tagged prefetching the best performance. The MRs are relatively larger for the two processor system because the subgrids are so large that all of the reference locality is not captured (intragrid contention). For larger grid sizes, tagged prefetching has the best performance, followed by prefetch-on-miss and finally, demand fetching. Notice the relative independence of the MR on the cache blocksize.

Figure 4.25 presents the same MR versus the number of processors and other variable features for the rectangular partition. Here the MR increases as the number of processors increase for the smaller grid sizes and remains relatively constant for the larger grid sizes. Again, the tagged prefetching strategy offers the best perfor-



a. Smaller Grid Sizes



b. Larger Grid Sizes

Figure 4.24 Miss Ratio versus the Number of Processors, All Fetching Strategies, 32Kbyte Cache Size, 32-byte Blocksize, All Grid Sizes, Square Decomposition.

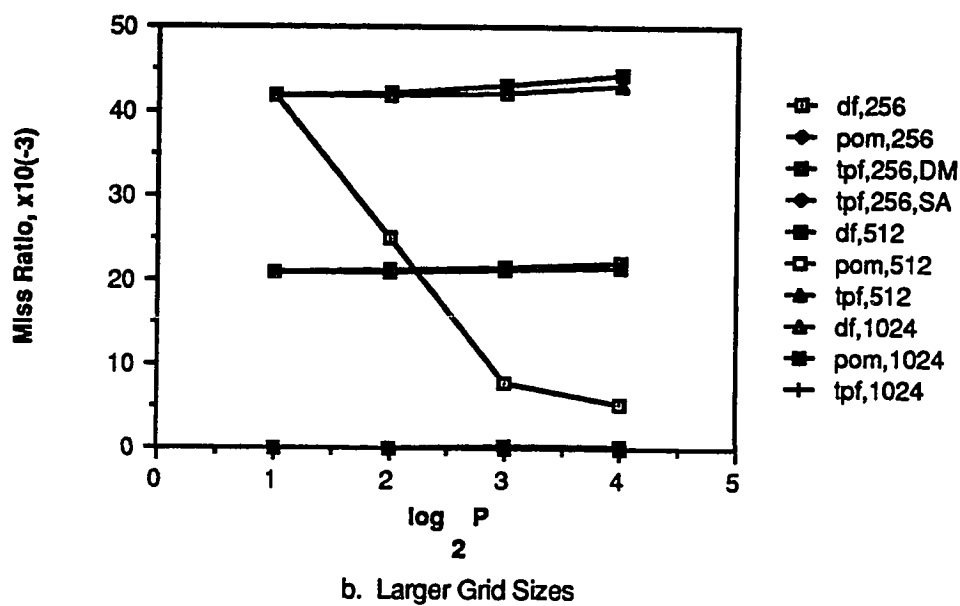
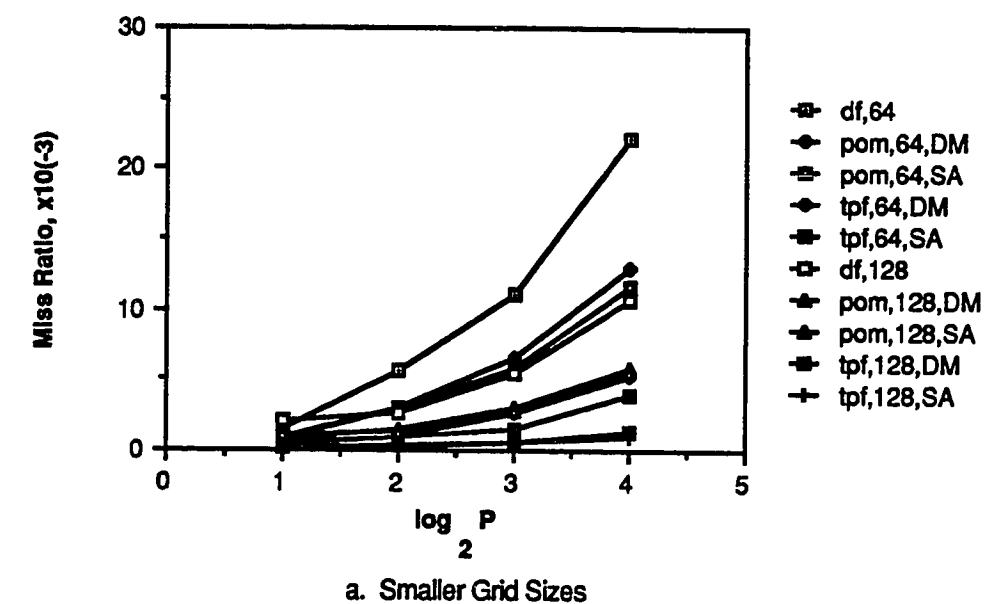


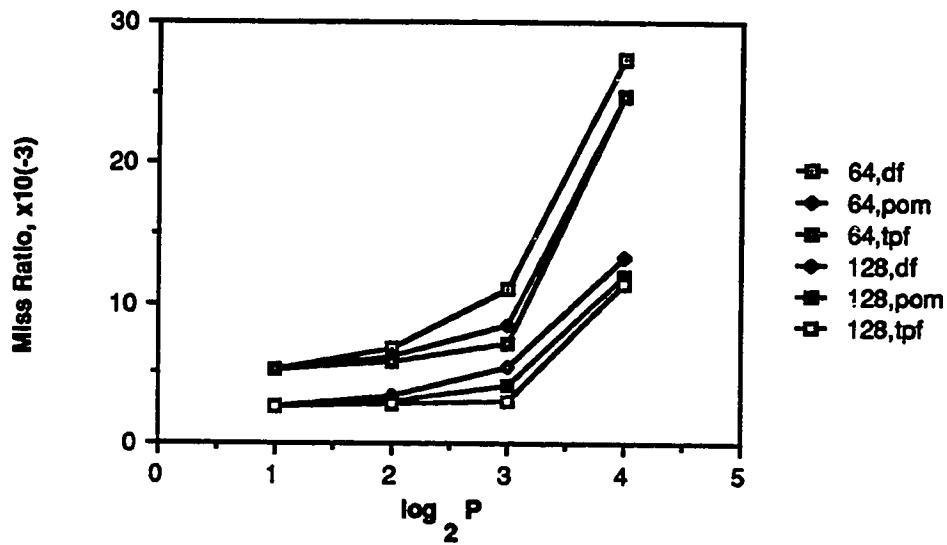
Figure 4.25 Miss Ratio versus the Number of Processors, All Fetching Strategies, 32Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Both Mapping Functions.

mance, followed by prefetch-on-miss and demand fetching, respectively. The 256x256 point demand fetching MR decreases as the number of processors increase. The cause of this behavior is discussed in the section 4.2.1.

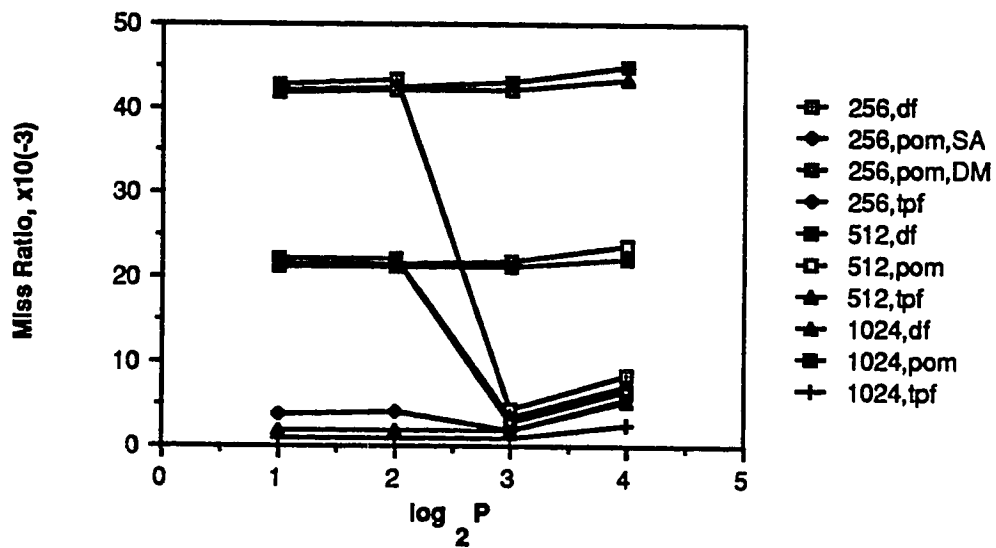
Figures 4.26 and 4.27 illustrate the same MR versus the number of processors as the previous two figures present; however, a 64Kbyte cache is assumed. All graphs generally show an increase in MR value as the number of processors increase for smaller grid sizes and a relative constant value for larger grid sizes. All exceptions are demand fetching values discussed in section 4.2.1. Again, tagged prefetching represents the best performance, followed by prefetch-on-miss and demand fetching.

Figures 4.28 and 4.29 illustrate the IR versus the number of processors for all fetching strategies considered, all grid sizes, both mapping functions, the square partition, a 32-byte blocksize and 32Kbyte (Figure 4.28) and 64Kbyte (Figure 4.29) cache sizes. All figures show that tagged prefetching results in the largest invalidation ratio, followed by prefetch-on-miss and finally demand fetching. The IR decreases as the grid size increases for all fetching strategies. There is no discrepancy between both prefetching IRs and the demand fetching IR for the rectangular decomposition strategy.

Figures 4.30 and 4.31 present the prXICO versus the number of processors for all fetching strategies, 32Kbyte (Figure 4.30) and 64Kbyte (Figure 4.31) cache sizes, all grid sizes, both mapping functions and a 32-byte blocksize. All graphs show an increase in this parameter as the number of processors increase. The demand fetching strategy exhibits the best performance while both prefetching strategies yield equal

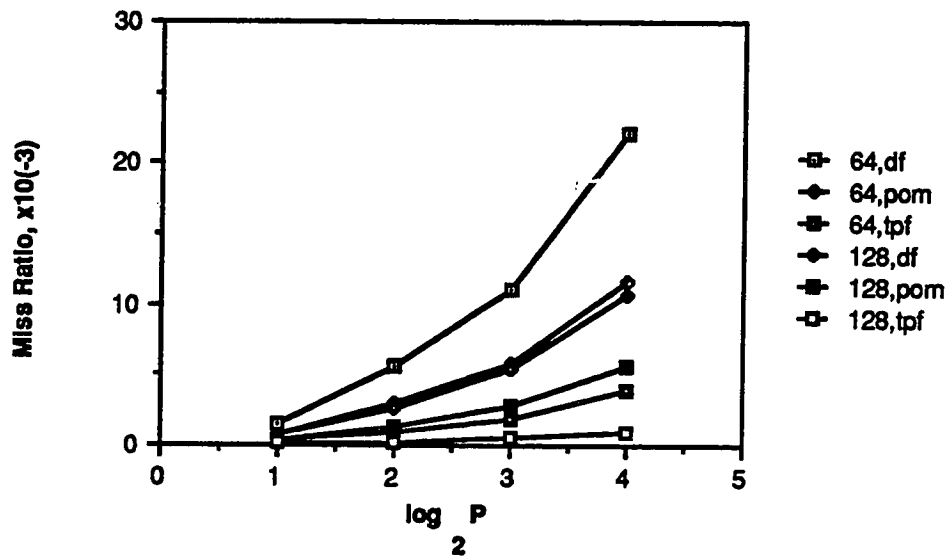


a. Smaller Grid Sizes

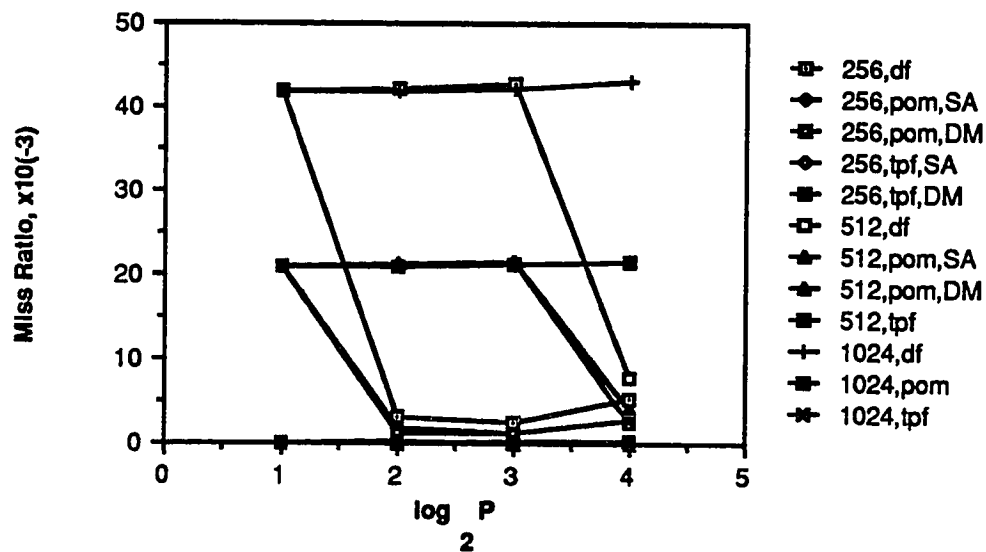


b. Larger Grid Sizes

Figure 4.26 Miss Ratio versus the Number of Processors, All Fetching Strategies, 64Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Both Mapping Functions.

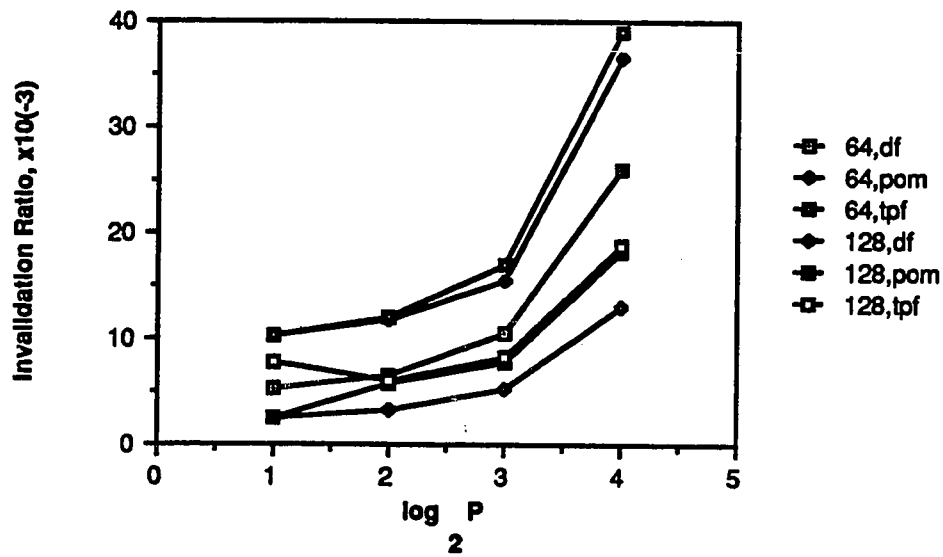


a. Smaller Grid Sizes

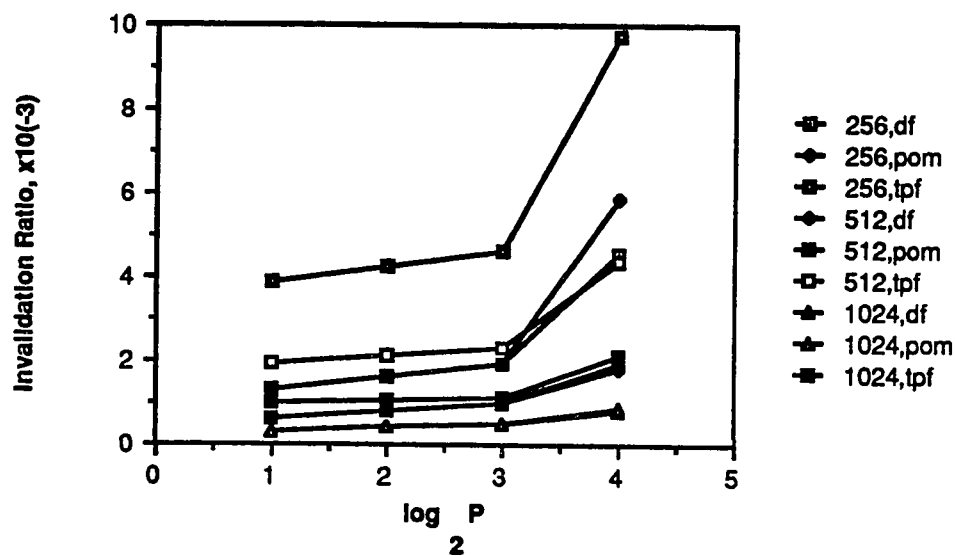


b. Larger Grid Sizes

Figure 4.27 Miss Ratio versus the Number of Processors, All Fetching Strategies, 64Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Both Mapping Functions.

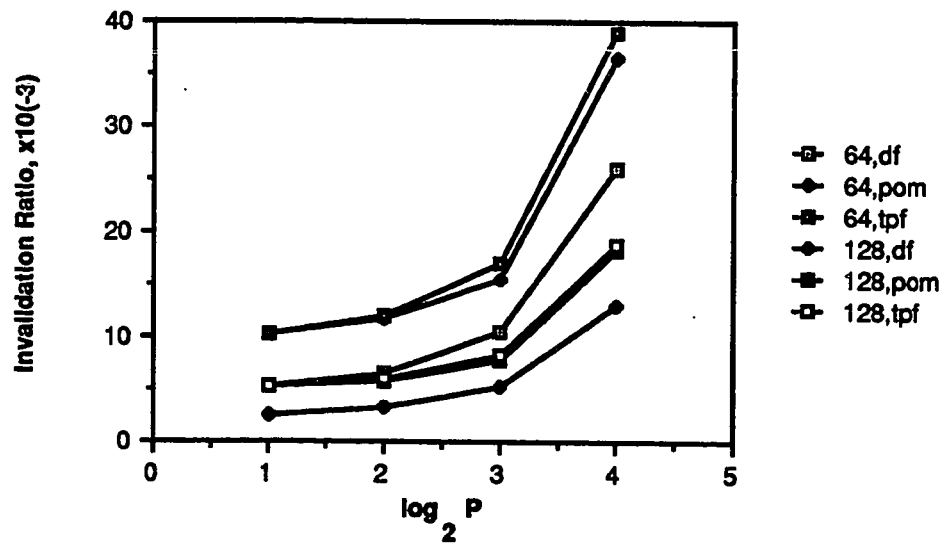


a. Smaller Grid Sizes

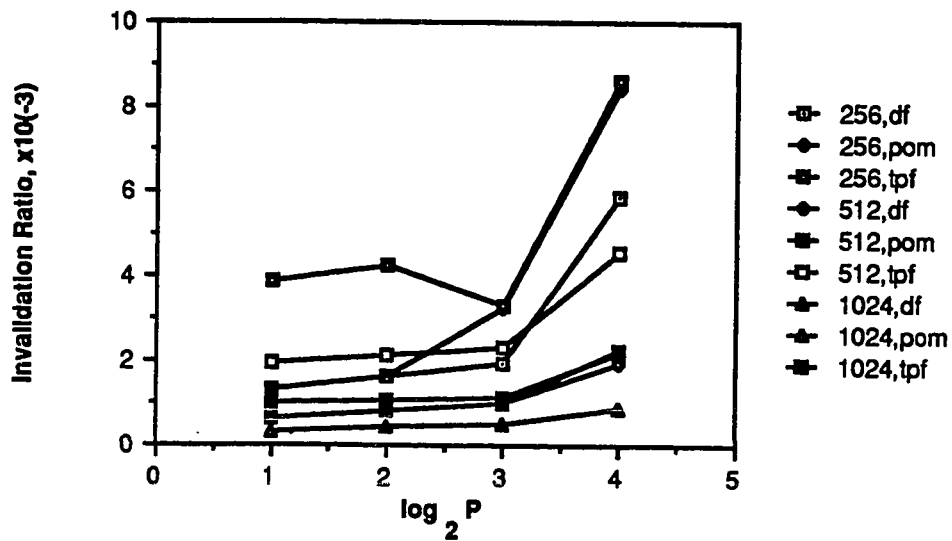


b. Larger Grid Sizes

Figure 4.28 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, 32Kbyte Cache Size, Square Decomposition, Both Mapping Functions.

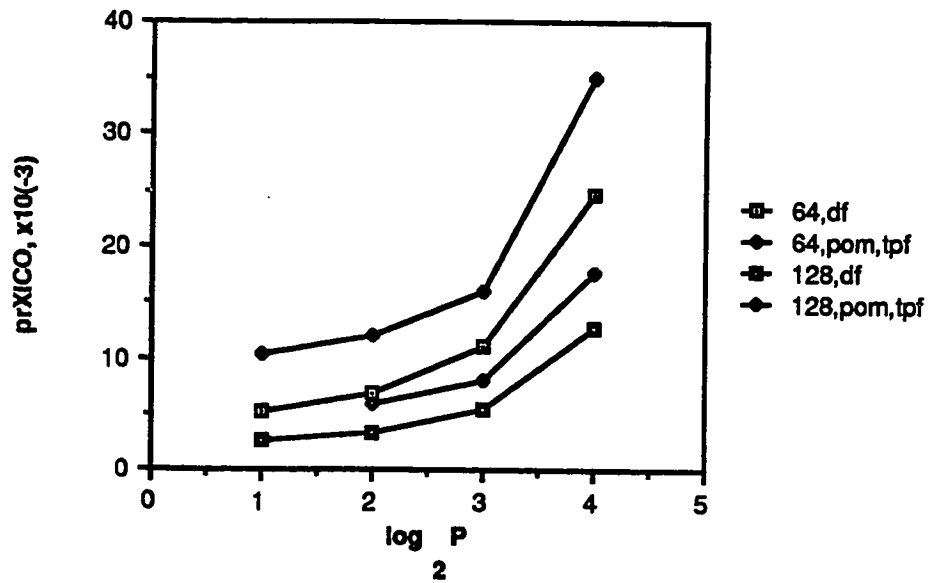


a. Smaller Grid Sizes

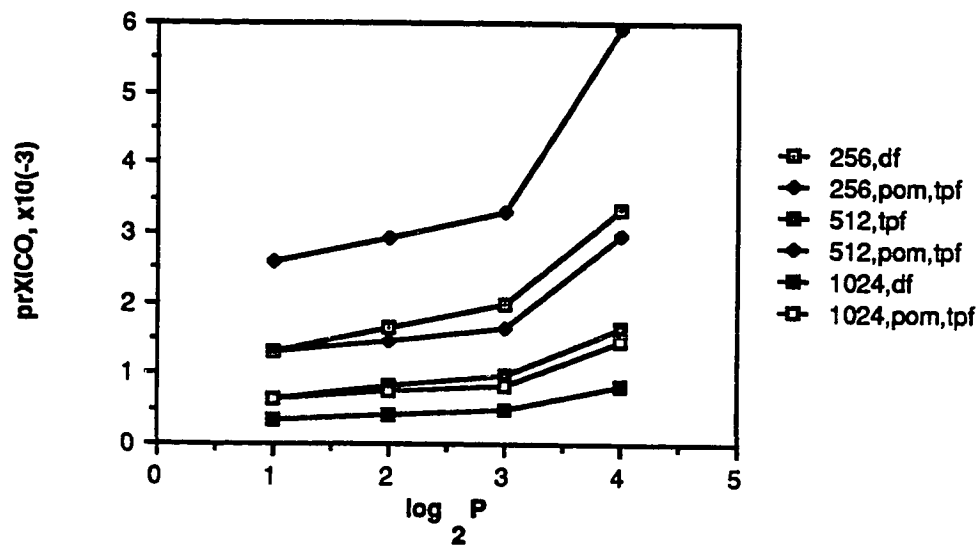


b. Larger Grid Sizes

Figure 4.29 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, 64Kbyte Cache Size, Square Decomposition, Both Mapping Functions.

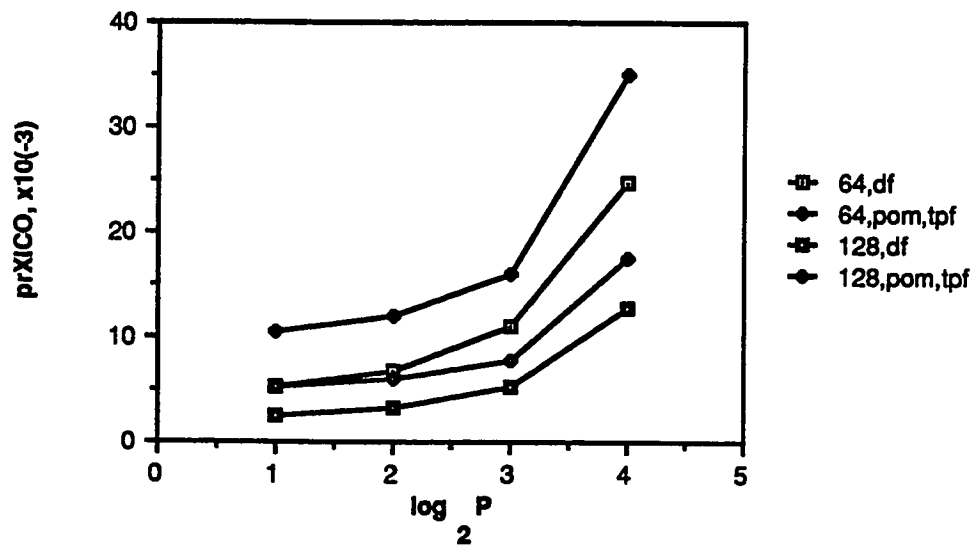


a. Smaller Grid Sizes

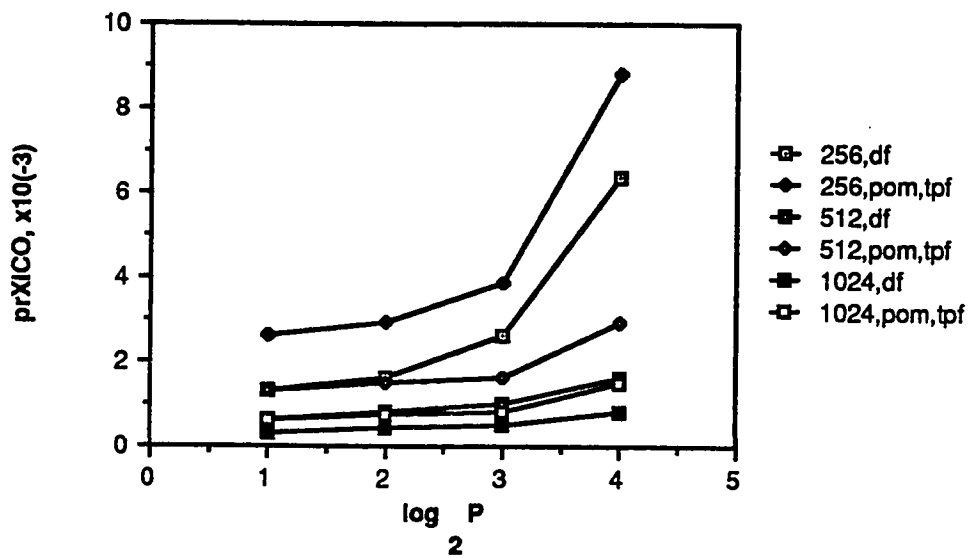


b. Larger Grid Sizes

Figure 4.30 prXICO versus the Number of Processors, All Fetching Strategies, 32Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Both Mapping Functions.



a. Smaller Grid Sizes



b. Larger Grid Sizes

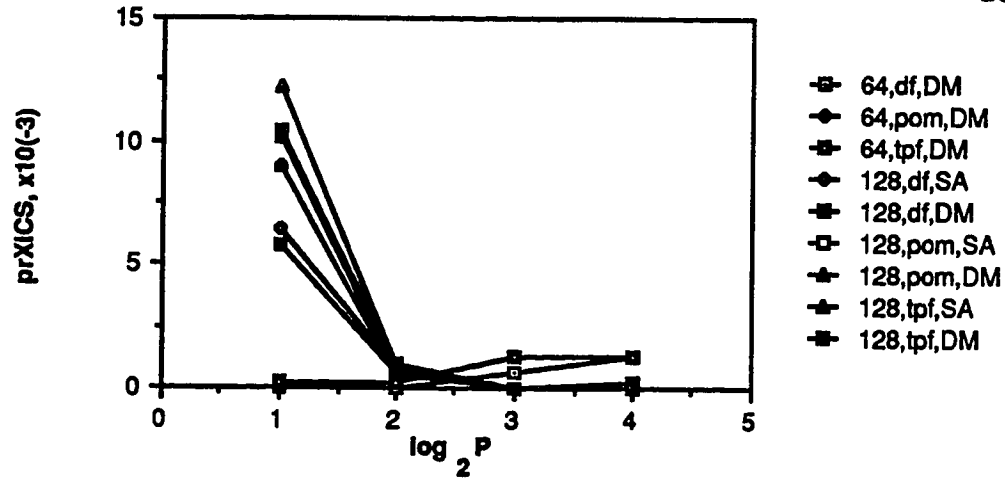
Figure 4.31 prXICO versus the Number of Processors, All Fetching Strategies, 64Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Both Mapping Functions.

but higher probabilities. For larger grid sizes, when the cache size is doubled, the prXICO increases slightly for all fetching strategies. Like the IR, there is no difference between the prXICO for all fetching strategies considered.

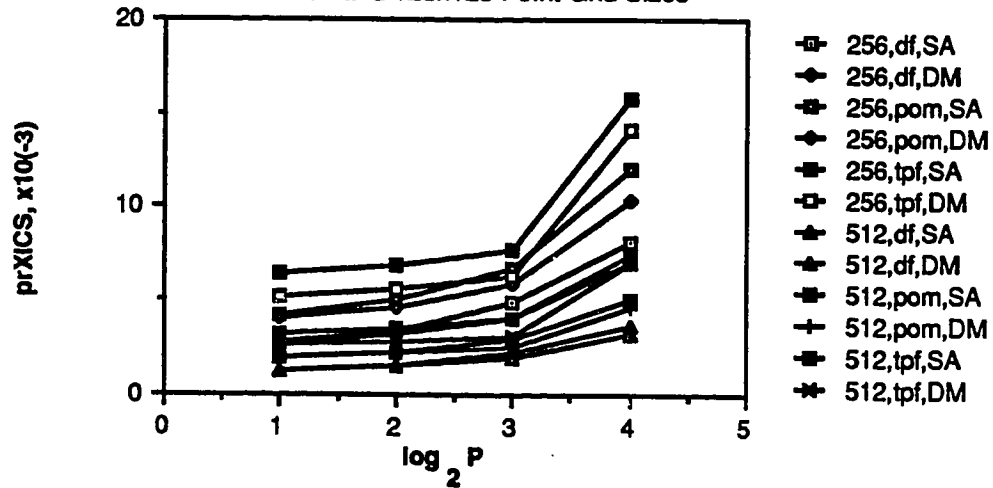
Figure 4.32 offers the prXICS versus the number of processors for all fetching strategies, the 32Kbyte cache size, a 32-byte blocksize, all grid sizes, the square partition and both mapping functions. The graphs shown indicate an increase in the prXICS as the number of processors increase for grid sizes larger than 128x128 points. Furthermore, the set-associative mapping function produces a slightly higher prXICS than direct mapping. Again, demand fetching manifests a better performance, followed by prefetch-on-miss and finally, tagged prefetching.

The smaller grid sizes show a decrease in the prXICS as the number of processors increase. This is because the more processors used in the system, the smaller the sub-grids updated by each processor and therefore the better the reference locality. The 64x64 and 128x128 point grid sizes are so small that all references are captured by the larger processor systems. Since most XICSs are RO->EXs performed as a result of replacing blocks, the amount performed by these small grid/larger processor tuples are negligible.

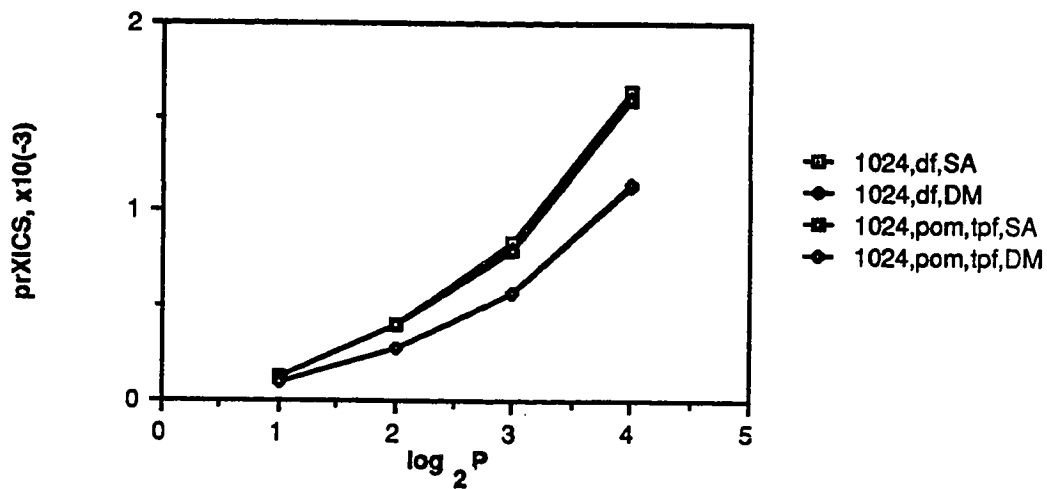
Figure 4.33 represents the prXICS versus the number of processors and all other features of the previous figure for the rectangular partition. All of the values shown here are relatively small in comparison with the square partition values. In such cases demand fetching offers better performance, followed by both prefetching strategies. The 64x64, 512x512 and 1024x1024 point grid sizes offer an increase in the prXICS



a. 64x64 and 128x128 Point Grid Sizes

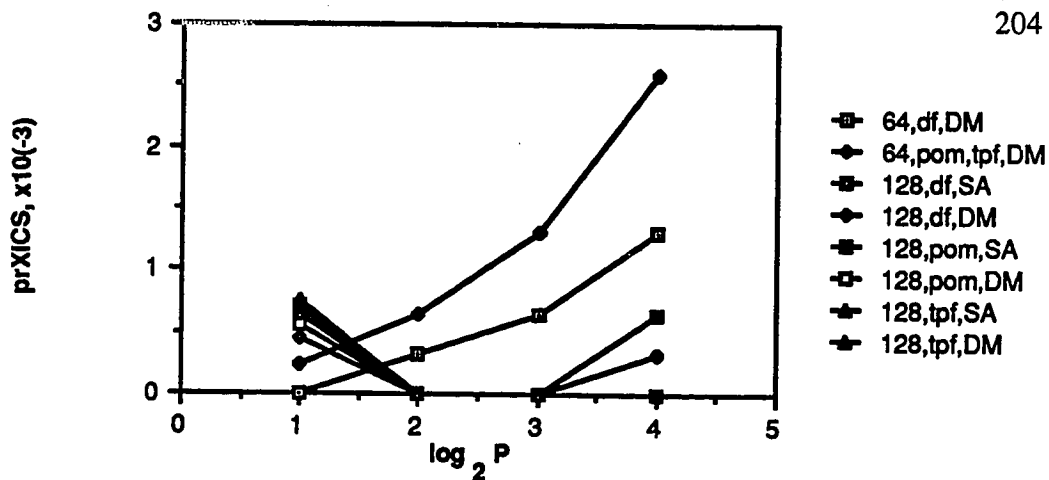


b. 256x256 and 512x512 Point Grid Sizes

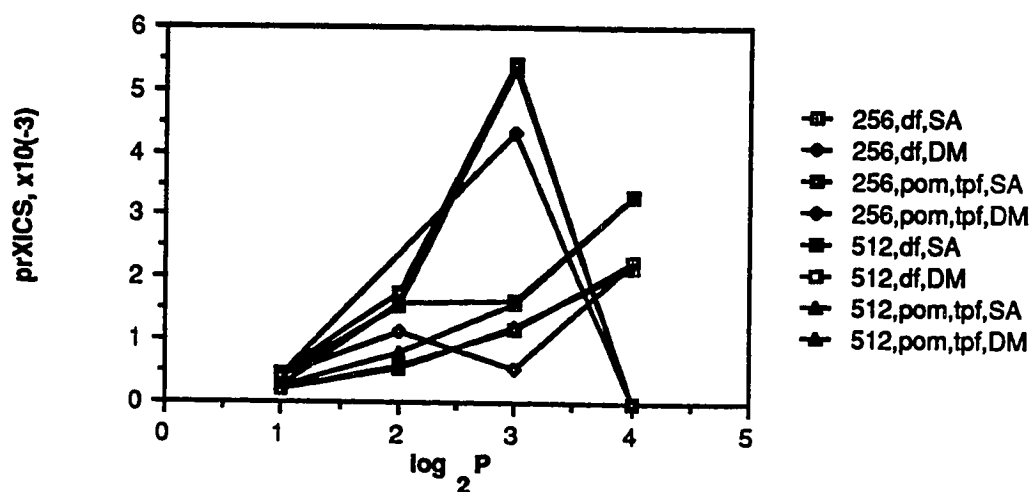


c. 1024x1024 Point Grid Size

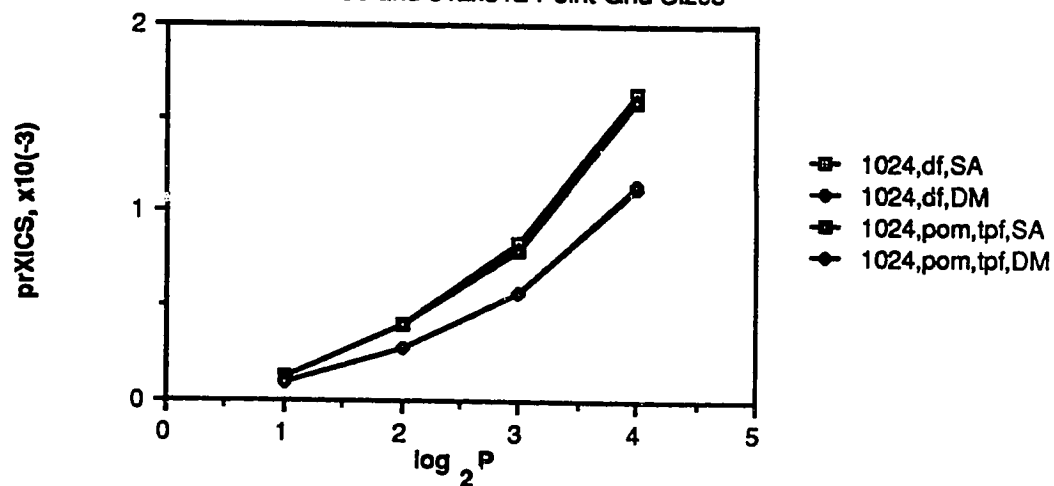
Figure 4.32 prXICS versus the Number of Processors, All Fetching Strategies, 32Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Both Mapping Strategies.



a. 64x64 and 128x128 Point Grid Sizes



b. 256x256 and 512x512 Point Grid Sizes



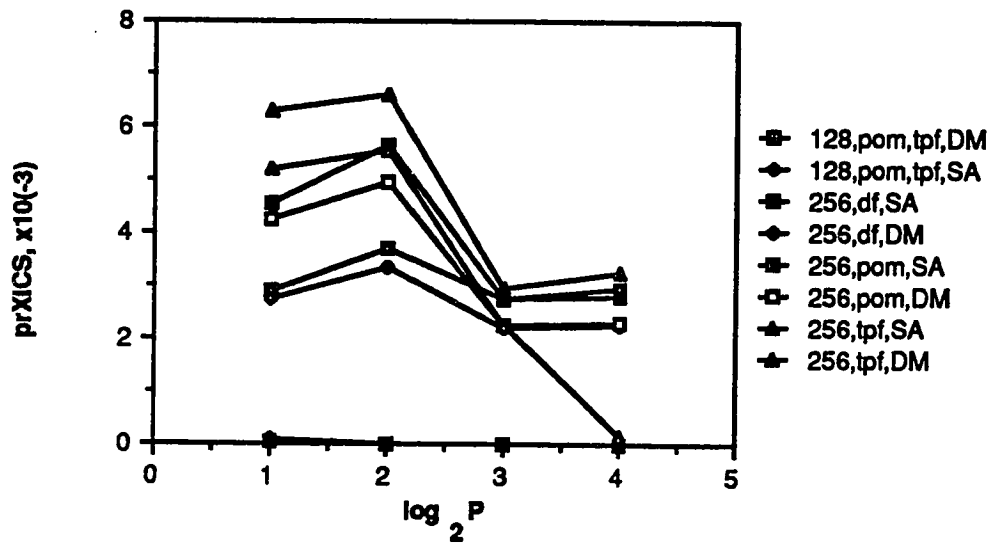
c. 1024x1024 Point Grid Size

Figure 4.33 prXICS versus the Number of Processors, All Fetching Strategies, 32Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Both Mapping Strategies.

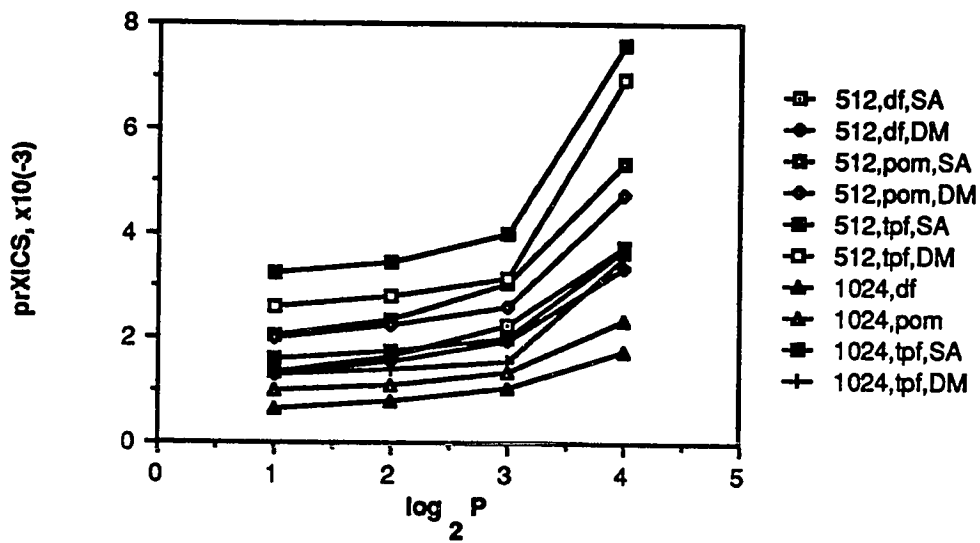
as the number of processors increase. The 128x128 point grid size offers a small prXICS for two processors, decreases to zero for four and eight processors, and then increases slightly in value for 16 processors. The reason for this behavior is directly related to the reference behavior for this algorithm. For this grid size, only zero to one XICS occurs per iteration. Likewise, the purpose of the nonlinear behavior represented by the 256x256 point grid size depends upon the randomness of the zero to three XICSs performed per iteration.

Figure 4.34 illustrates the prXICS versus the number of processors as shown in Figure 4.32; however, the 64Kbyte cache size is represented. The 128x128 point values are negligible while the 256x256 point grid size demonstrates an increase in value as the number of processors increase from two to four and from eight to sixteen. There is an increase in value for the eight processor system because the locality of references is almost optimal for this grid size/8-processor tuple. Both the 512x512 and 1024x1024 point grid sizes show an increase in value as the number of processor increase. In all instances, direct mapping offers the better performance. Also, demand fetching offers the best performance followed by prefetch-on-miss and finally tagged prefetching.

Figure 4.35 presents the prXICS versus the number of processors for all fetching strategies, the 64Kbyte cache size, all grid sizes, both mapping functions and the rectangular decomposition strategy. The behavior of the graphs is similar to the behavior of the graphs shown in the previous figure, with the exception that the values of the prXICS are somewhat smaller. In fact, for SOR, the prXICS is never greater than 0.02 and for the most part is usually less than 0.01.

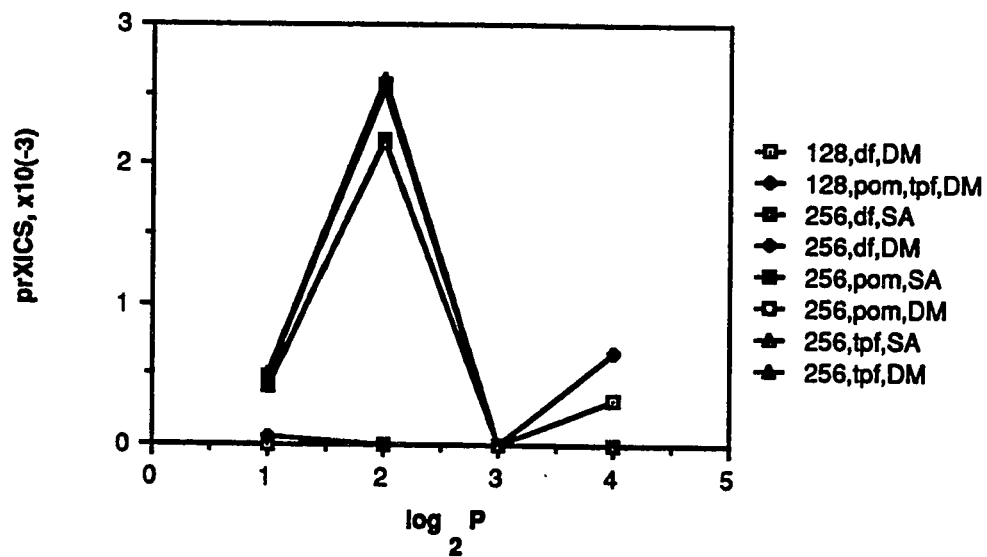


a. 256x256 and 512x512 Point Grid Sizes

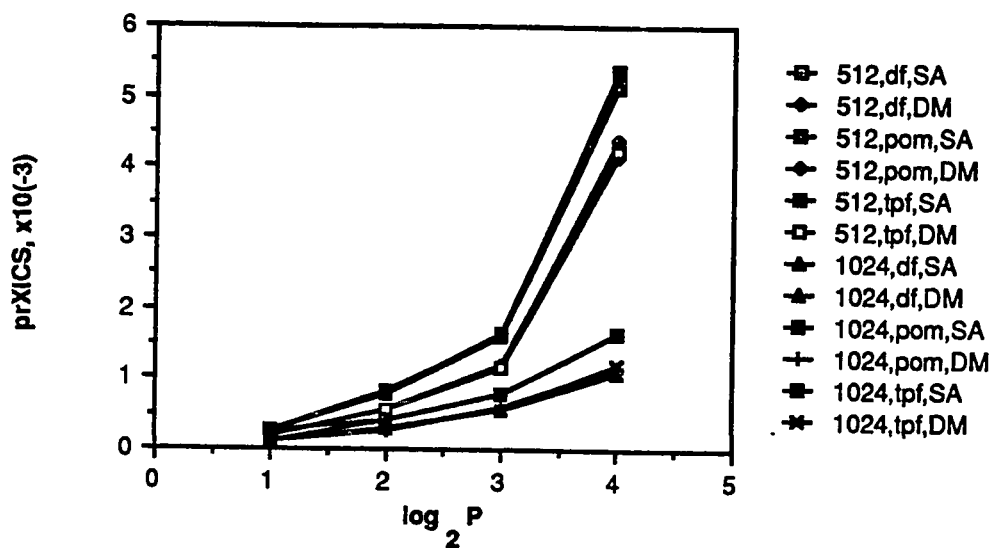


b. 512x512 and 1024x1024 Point Grid Sizes

Figure 4.34 prXICS versus the Number of Processors, All Fetching Strategies, 64Kbyte Cache Size, 32-byte Blocksize, Square Decomposition Both Mapping Functions.



a. 128x128 and 256x256 Point Grid Sizes



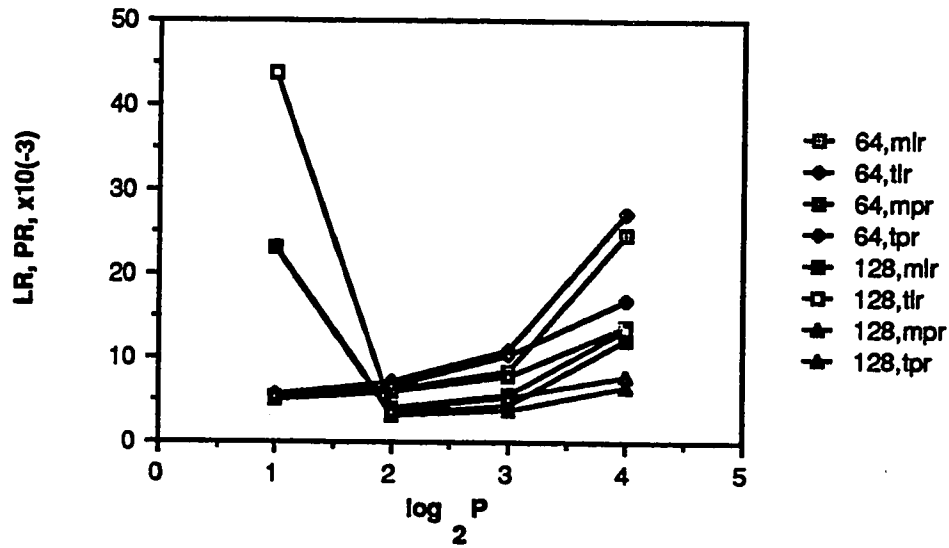
b. 512x512 and 1024x1024 Point Grid Sizes

Figure 4.35 prXICS versus the Number of Processors, All Fetching Strategies, 64Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Both Mapping Functions.

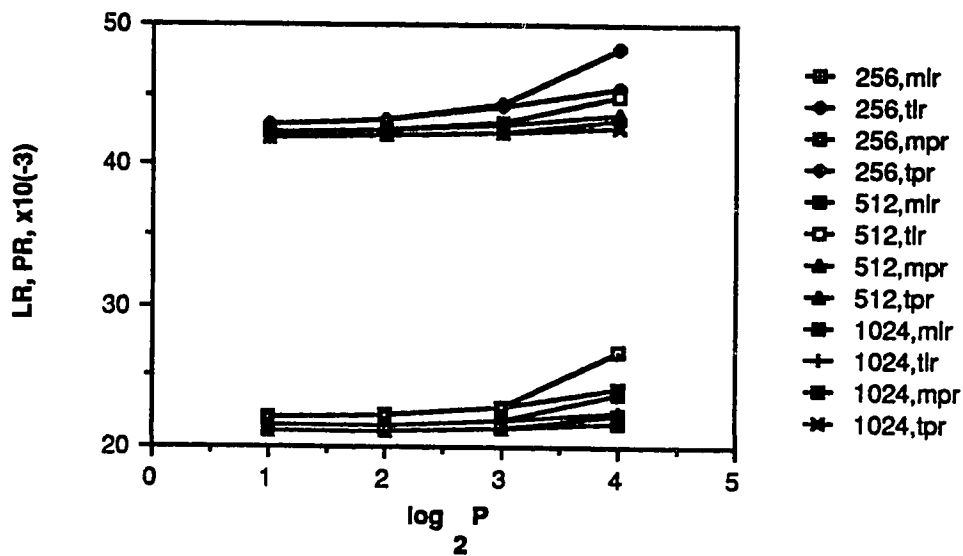
Figures 4.36 and 4.37 illustrate the lookup and prefetch ratios versus the number of processors for the square partition, direct mapping and the 32Kbyte (4.36) and 64Kbyte (4.37) cache sizes. All graphs indicate relative constant LR and PRs and the number of processors increase for grid sizes larger than 128x128 points. The only exception is the 256x256 point grid size (64Kbyte cache) where the values decrease as the number of processors increase from four to eight processors; a result of a significant change in locality for this grid size. Observe that the prefetch-on-miss ratios demonstrate better performance than their tagged prefetching counterparts. In all instances, the intuitive behavior of a higher lookup ratio is exhibited. A larger cache size is better able to capture the reference locality for the smaller grid sizes and two processor system. Otherwise, the ratios increase as the number of processors increase.

Figures 4.38 and 4.39 also illustrate the same LR and PR versus the number of processors for the rectangular decomposition strategy (all other features are identical to the previous two figures). These graphs show the prefetch-on-miss LR and PRs offering the better performance. For smaller grid sizes, the graphs show an increase in the LR and PR as the number of processors increase for both cache sizes. The larger grid sizes show constant values for these parameters except where the grid size/processor tuple is such that the cache captures a better locality of reference. In such instances sharp decreases in the ratios are observed.

Figures 4.40 through 4.43 illustrate the lookup and prefetch ratios versus the number of processors and other features identical to the previous four figures with one exception; the set-associative mapping function is used. The behavior of these

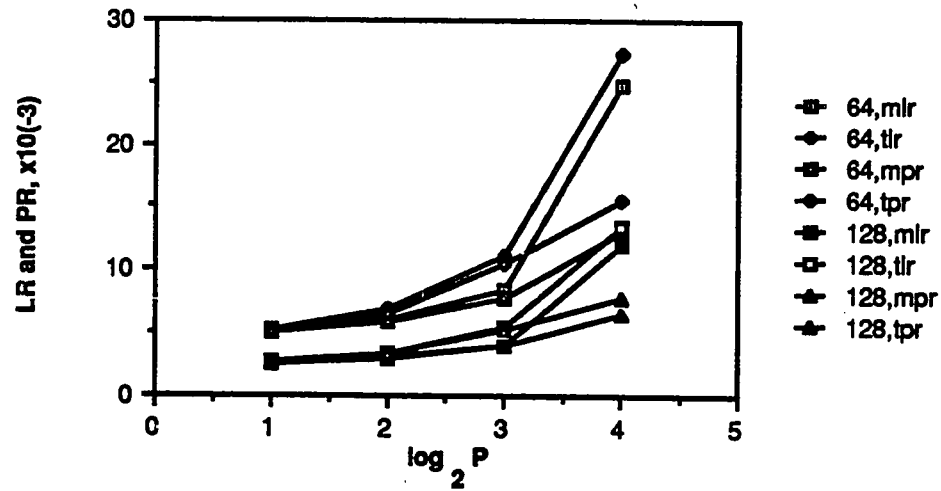


a. Smaller Grid Sizes

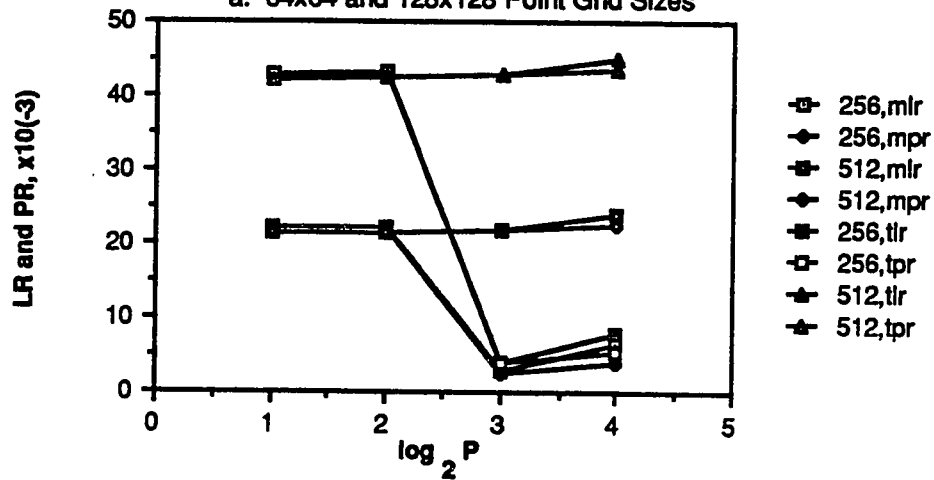


b. Larger Grid Sizes

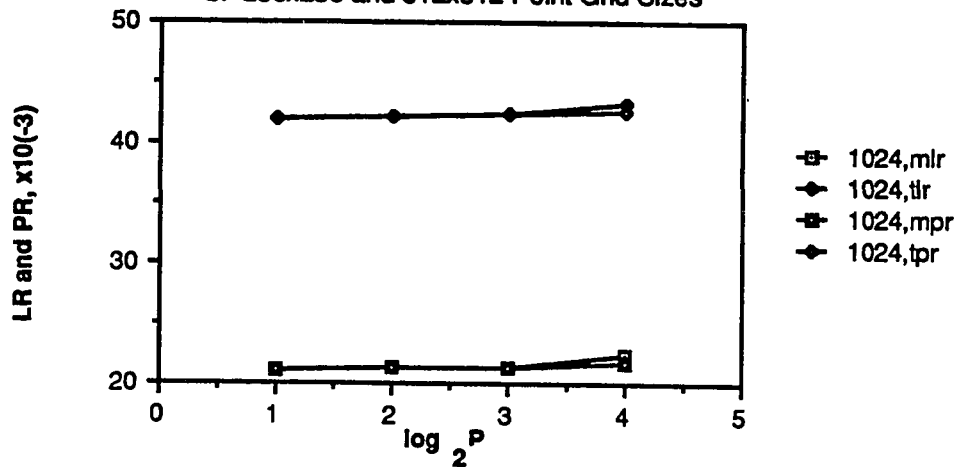
Figure 4.36 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 32Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Direct Mapping.



a. 64x64 and 128x128 Point Grid Sizes

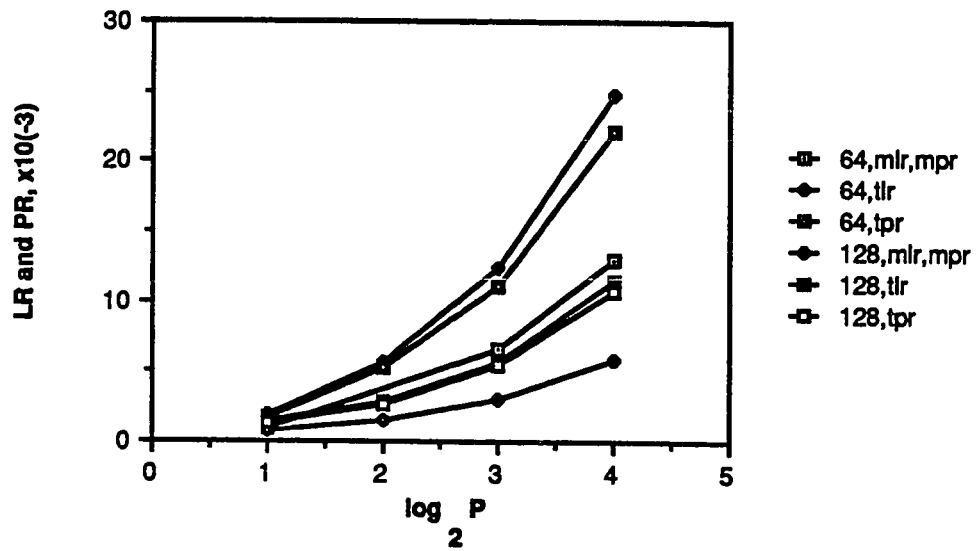


b. 256x256 and 512x512 Point Grid Sizes

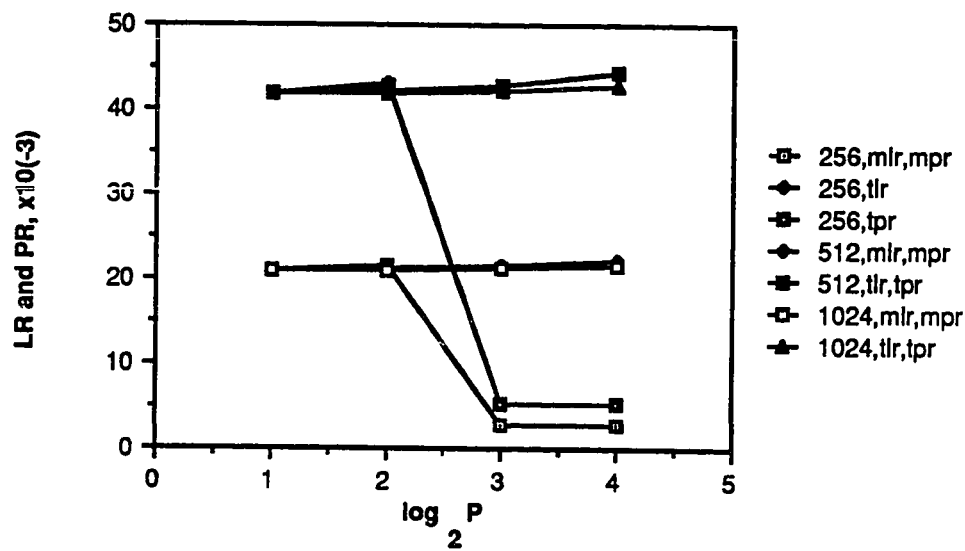


c. 1024x1024 Point Grid Size

Figure 4.37 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 64Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Direct Mapping.

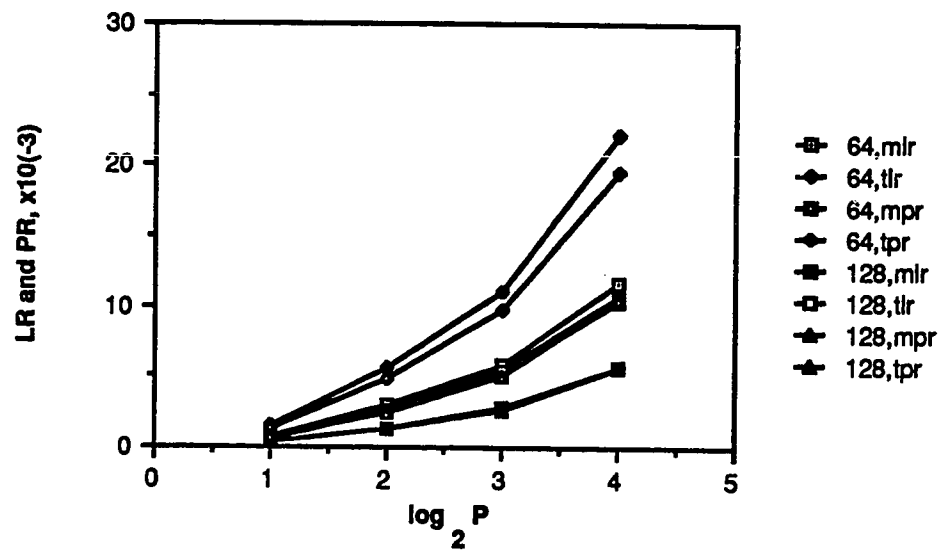


a. Smaller Grid Sizes

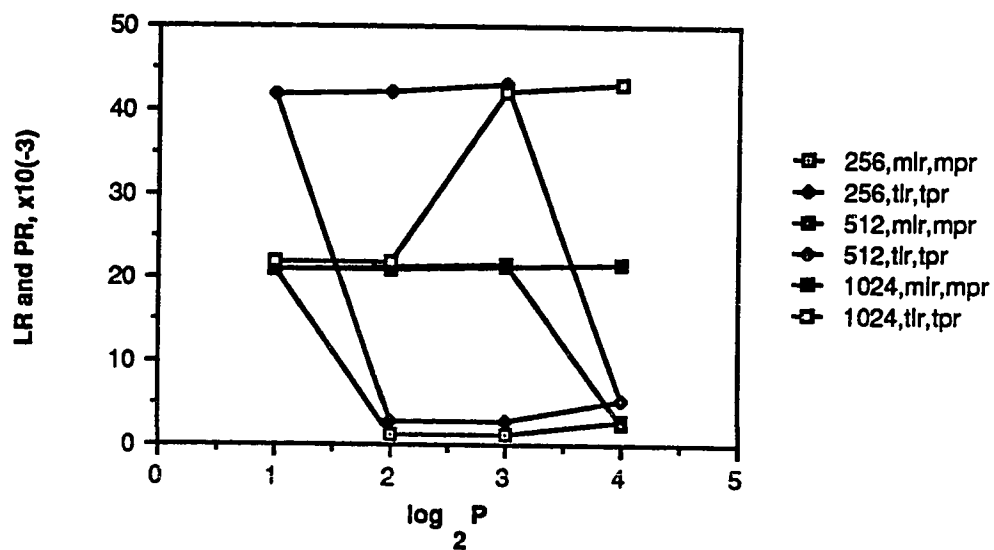


b. Larger Grid Sizes

Figure 4.38 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 32Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Direct Mapping.

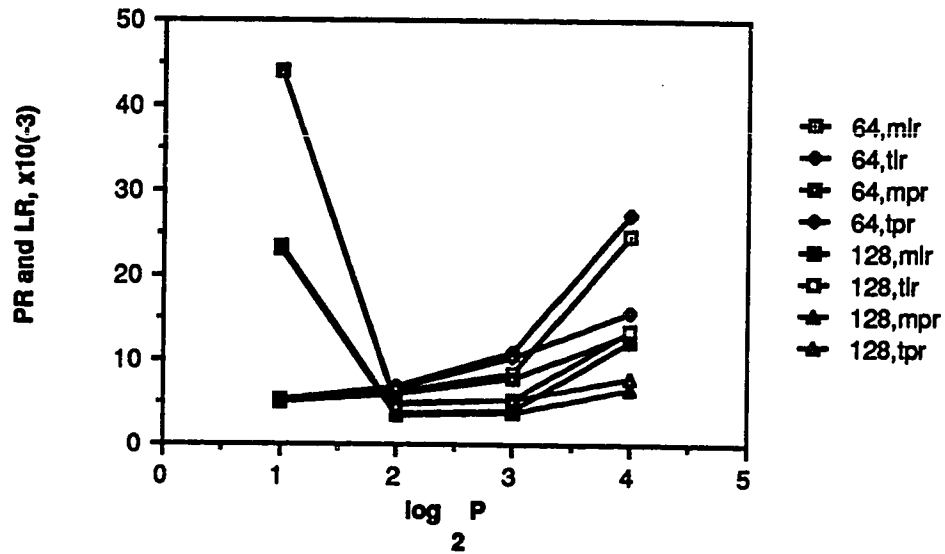


a. Smaller Grid Sizes

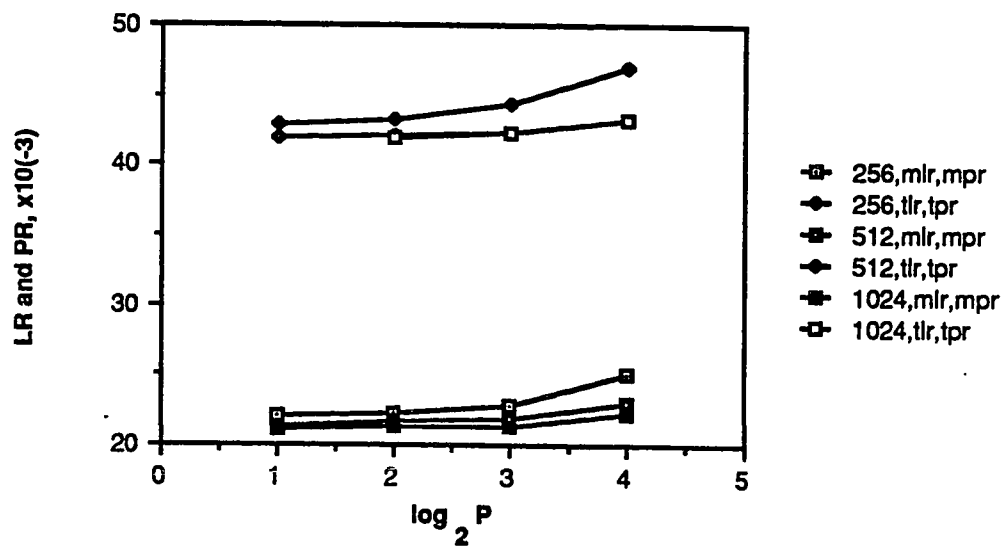


b. Larger Grid Sizes

Figure 4.39 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 64Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Direct Mapping.

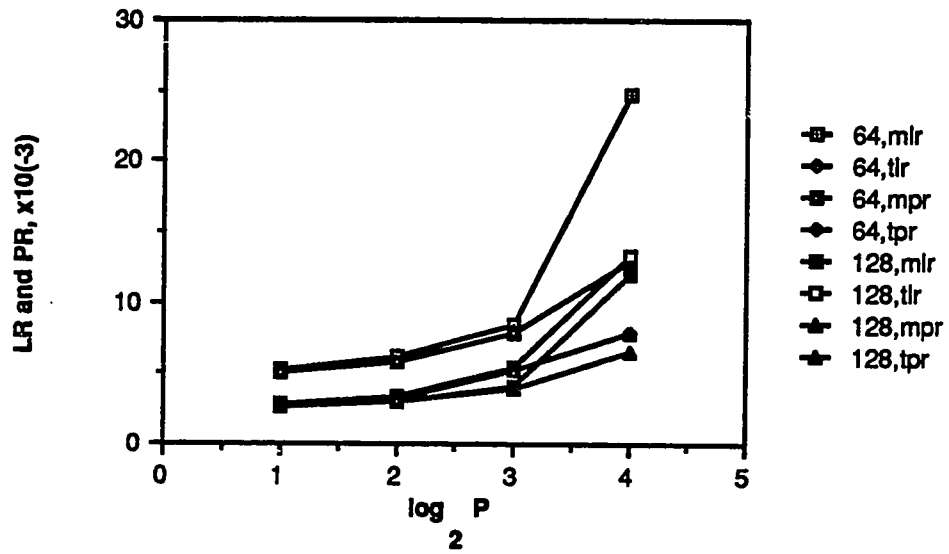


a. Smaller Grid Sizes

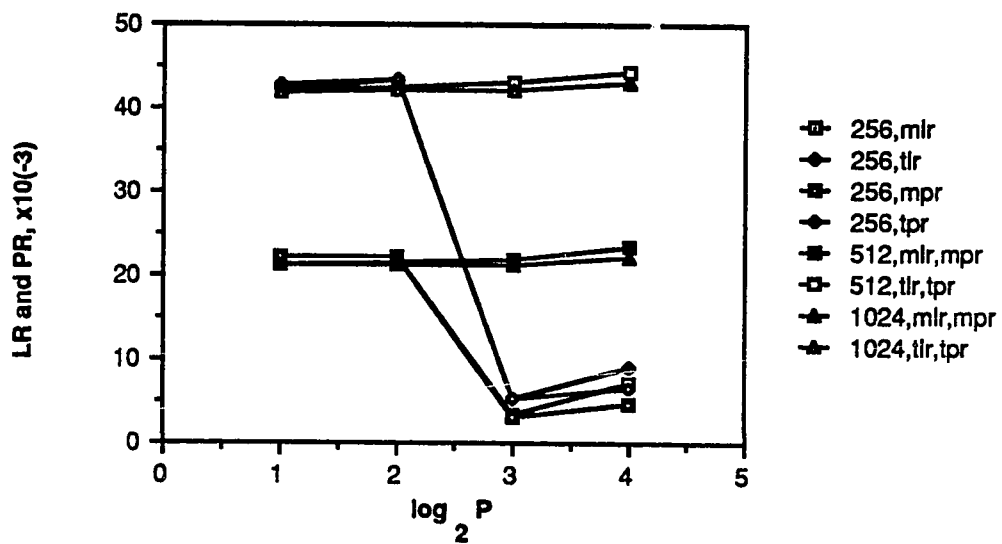


b. Larger Grid Sizes

Figure 4.40 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 32Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Set-Associative Mapping.

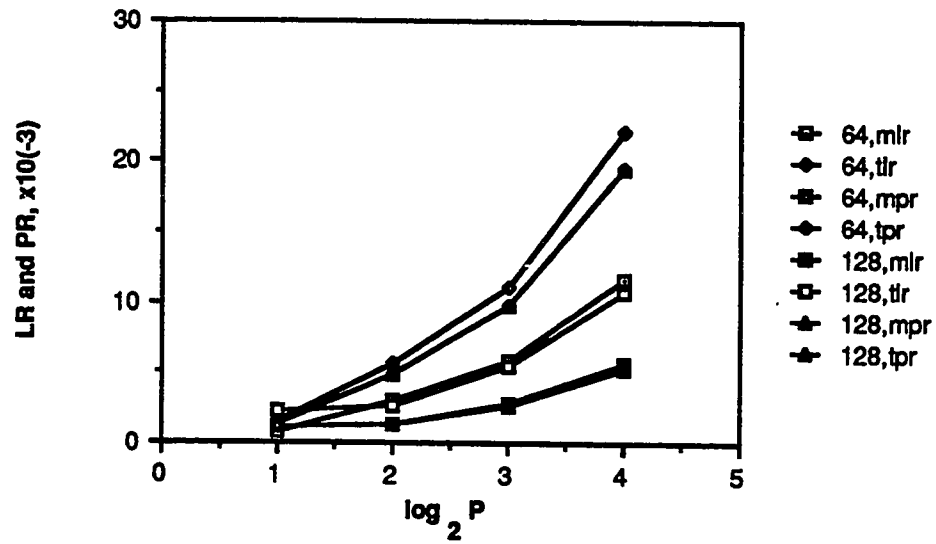


a. Smaller Grid Sizes

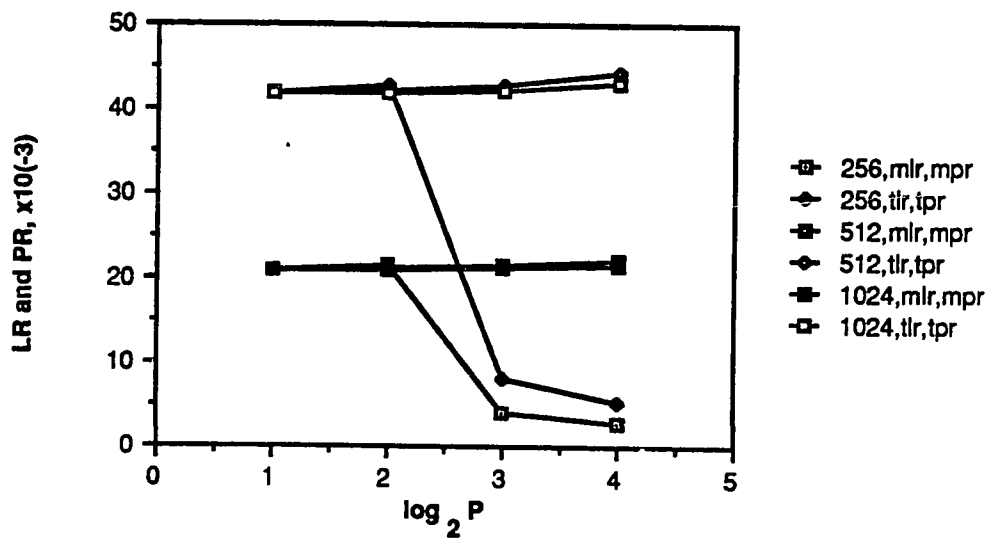


b. Larger Grid Sizes

Figure 4.41 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 64Kbyte Cache Size, 32-byte Blocksize, Square Decomposition, Set-Associative Mapping.

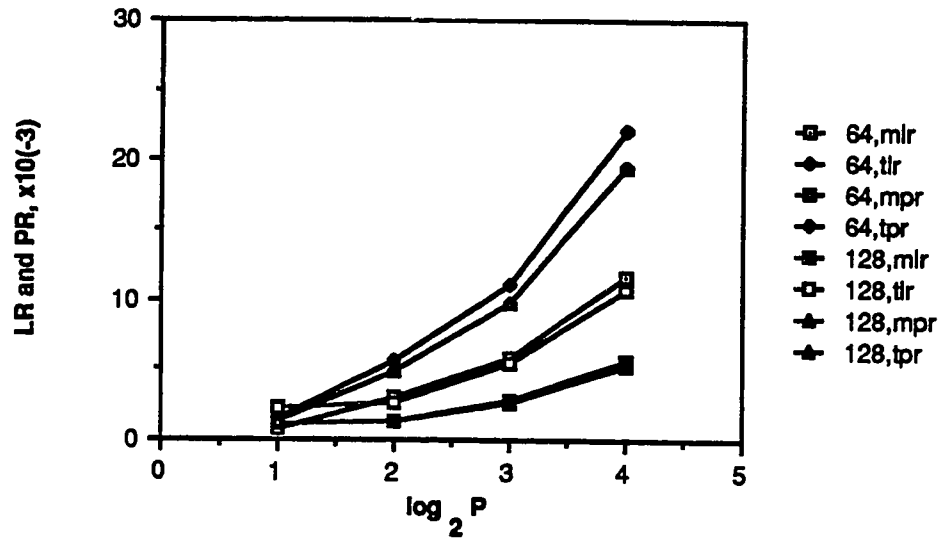


a. Smaller Grid Sizes

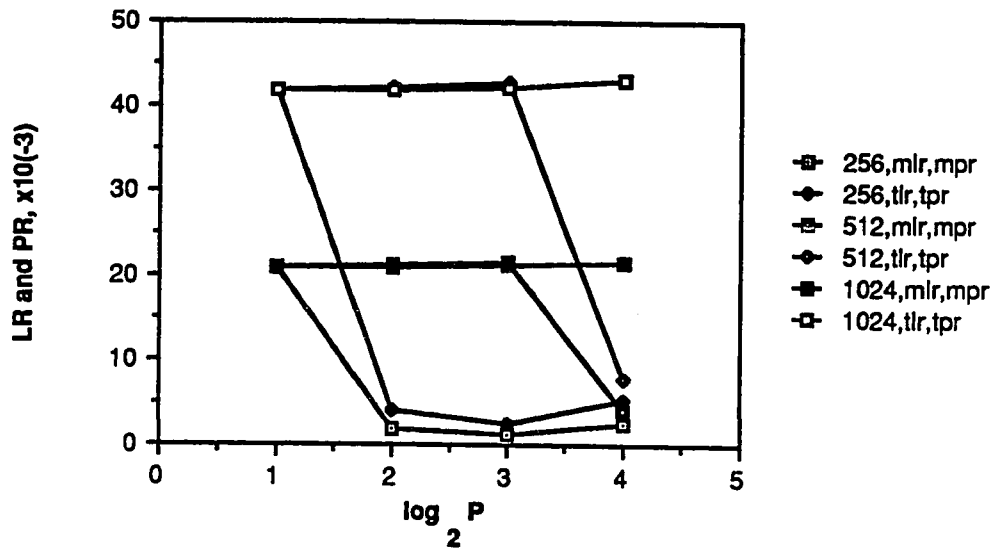


b. Larger Grid Sizes

Figure 4.42 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 32Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Set-Associative Mapping.



a. Smaller Grid Sizes



b. Larger Grid Sizes

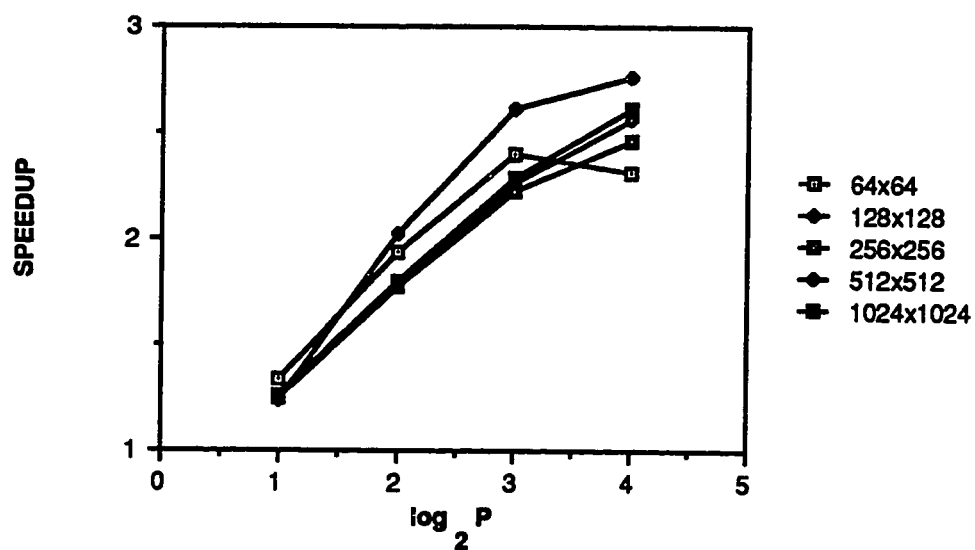
Figure 4.43 Lookup Ratio and Prefetch Ratio versus the Number of Processors, 64Kbyte Cache Size, 32-byte Blocksize, Rectangular Decomposition, Set-Associative Mapping.

graphs parallels those offered for the direct mapping function; however, for some grid sizes the direct mapping LR and PR are slightly higher. In general, the LR is intuitively higher than the PR and the tagged prefetching ratios are higher than the prefetch-on-miss ratios.

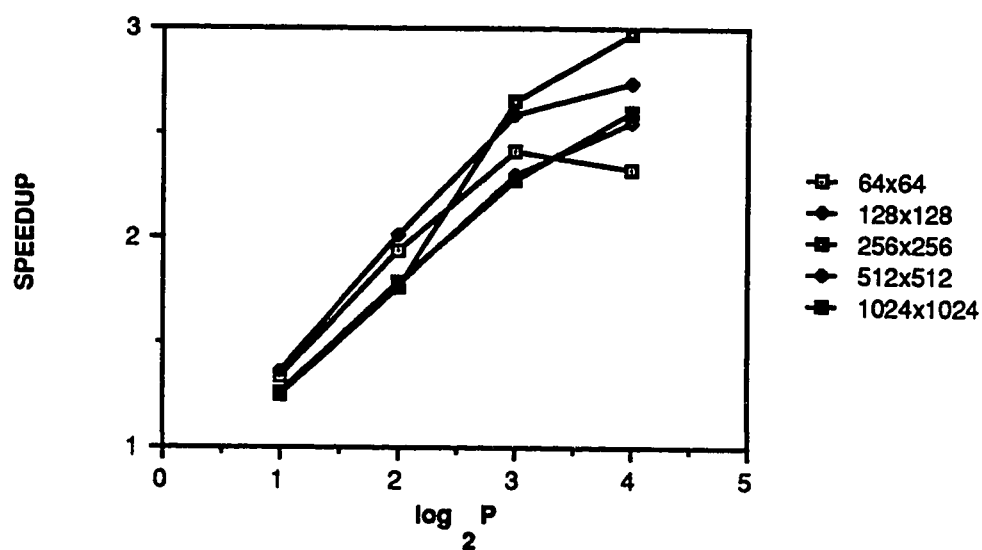
4.3. Multiprocessor Speedup

Figure 4.44 represents the speedup versus the number of processors for demand fetching, both cache sizes, all grid sizes, the square partition and a single bus interconnection. The graphs show a speedup range of 1.25 to 3 for 2 to 16 processors, respectively. As the cache size doubles, the speedups remain relatively constant for all grid sizes except the 256x256 point grid size. In this case the speedup increases slightly. Figure 4.45 also illustrates this same speedup for the rectangular partition. These graphs show a slightly higher speedup than the square partition.

Figures 4.46 and 4.47 also illustrate the speedup versus the number of processors and other features identical to the previous speedup curves with the exception that the prefetch-on-miss fetching strategy is used. All of these graphs show a speedup range from 1 to 3 as the number of processors vary from 2 to 16. However the 128x128 and 256x256 point grid sizes show a decrease in speedup for the 16 processor system. Also there is no significant discrepancy between the two cache sizes and between the two decomposition strategies. Figures 4.48 and 4.49 also present the speedup curves for the tagged prefetching strategy with results parallel to the prefetch-on-miss strategy.

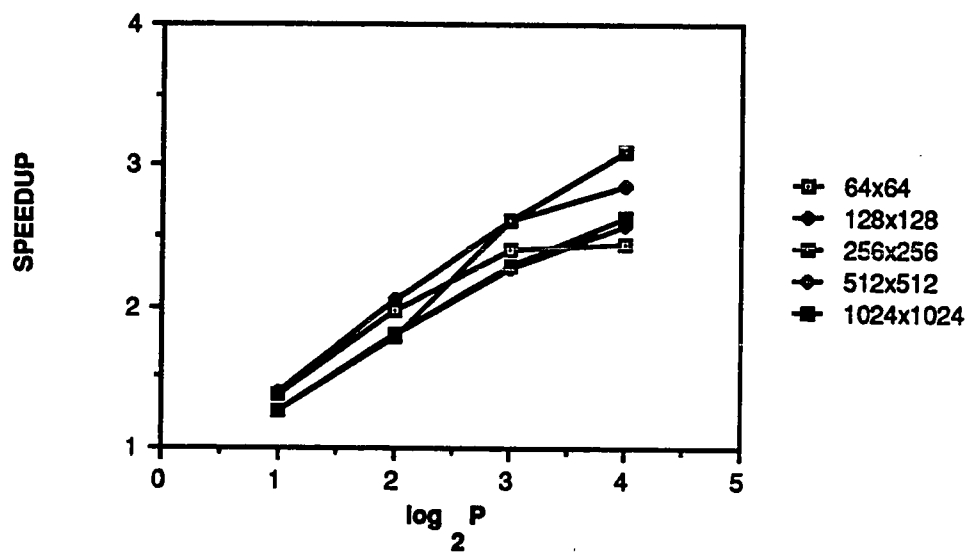


a. 32Kbyte Cache Size

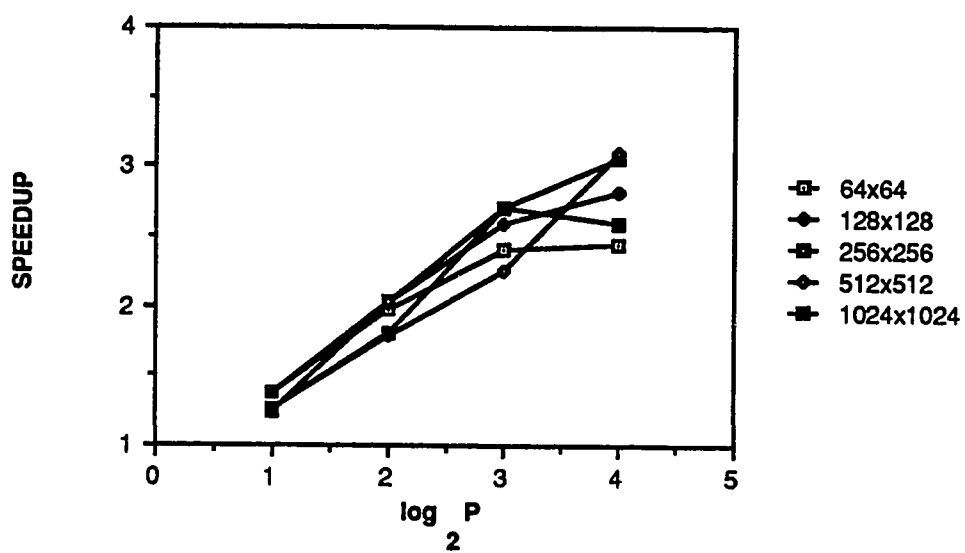


b. 64Kbyte Cache Size

Figure 4.44 Speedup versus the Number of Processors, Demand Fetching, Square Decomposition, Both Cache Size, Single Bus, 32-byte Blocksize.

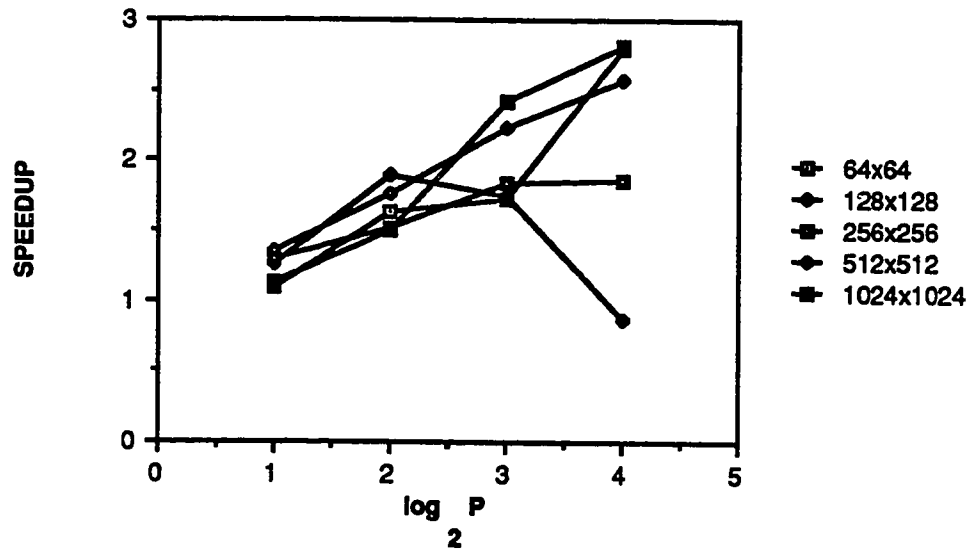


a. 32Kbyte Cache Size

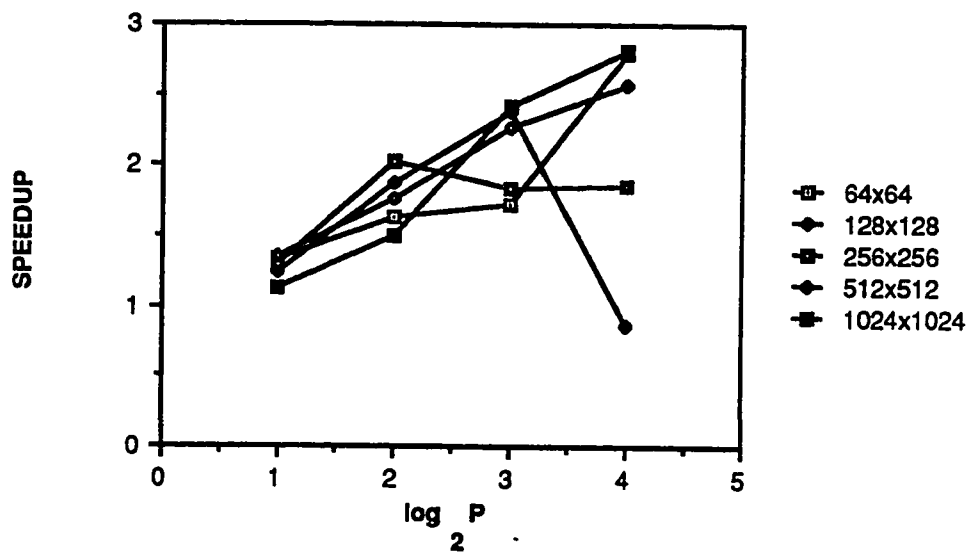


b. 64Kbyte Cache Size

Figure 4.45 Speedup versus the Number of Processors, Demand Fetching, Rectangular Decomposition, Both Mapping Functions, Single Bus, 32-byte Blocksize.

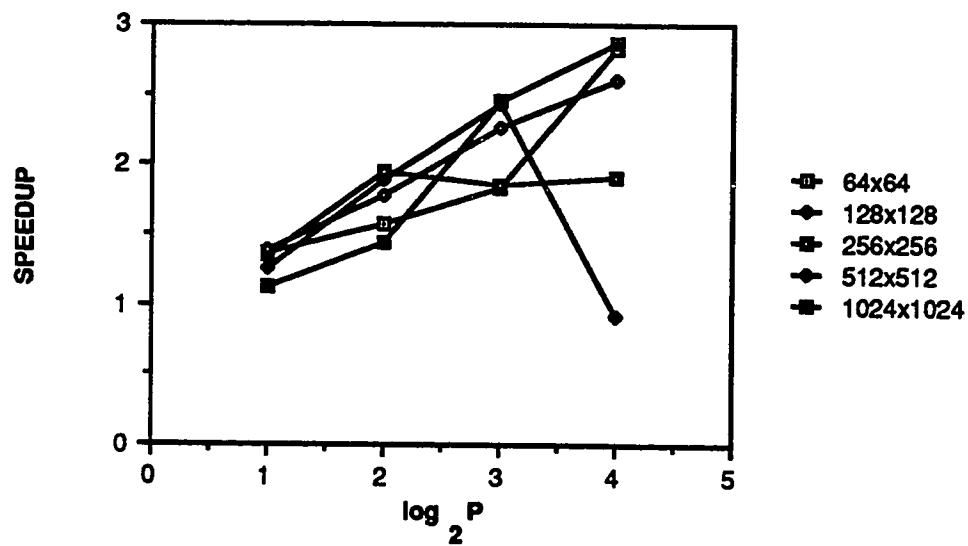


a. 32Kbyte Cache Size

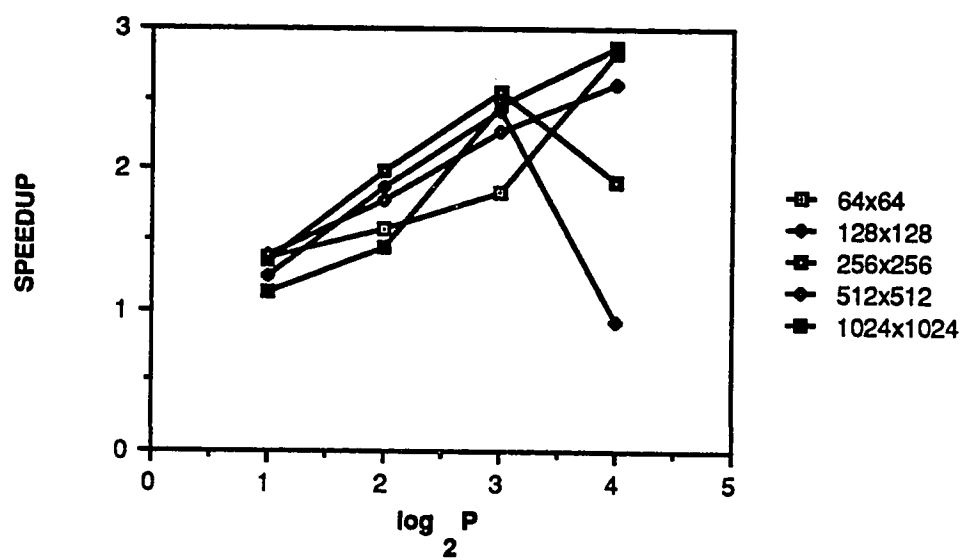


b. 64Kbyte Cache Size

Figure 4.46 Speedup versus the Number of Processors, Prefetch-on-miss, Square Decomposition, Both Mapping Functions, All Grid Sizes, 32-byte Blocksize.

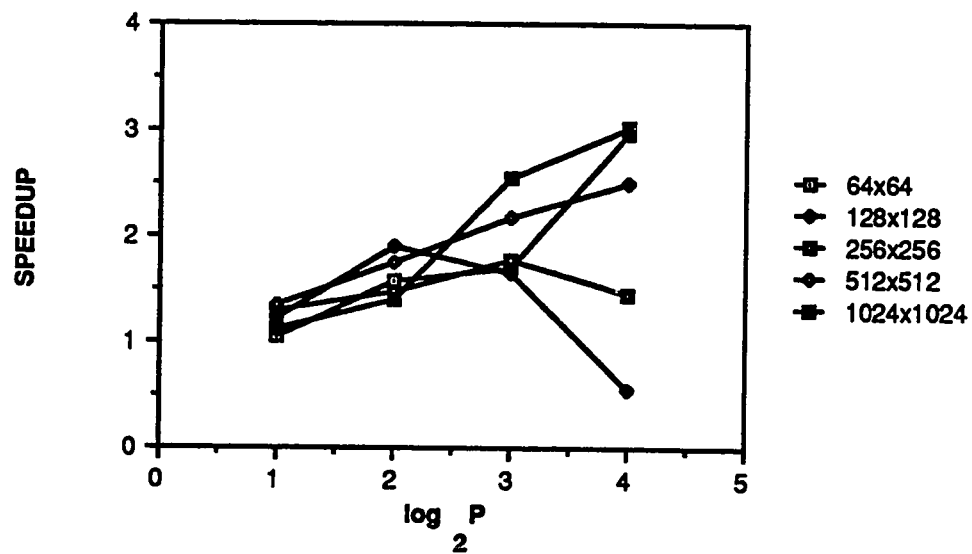


a. 32Kbyte Cache Size

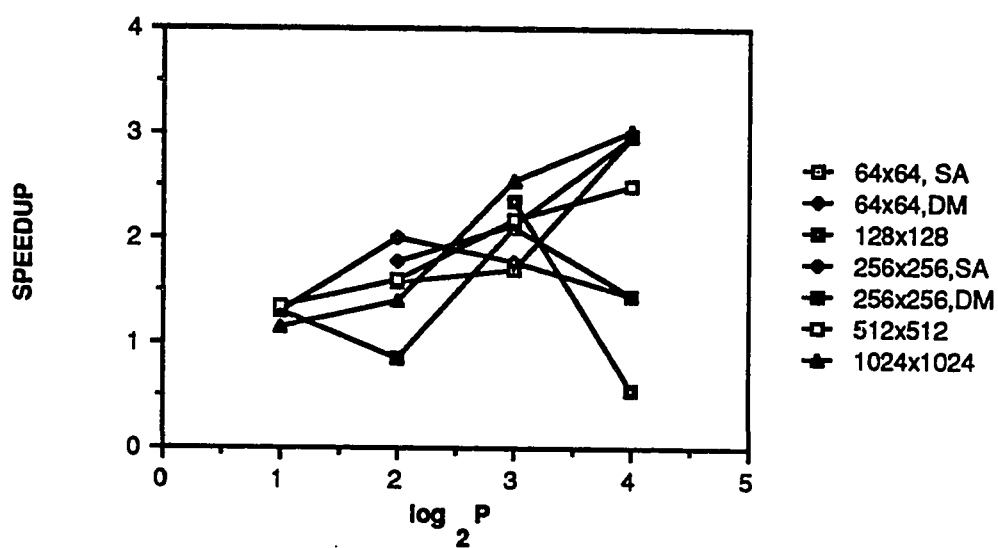


b. 64Kbyte Cache Size

Figure 4.47 Speedup versus the Number of Processors, Prefetch-on-miss, Rectangular Decomposition, Both Cache Sizes and Mapping Functions, All Grid Sizes, 32-byte Blocks.

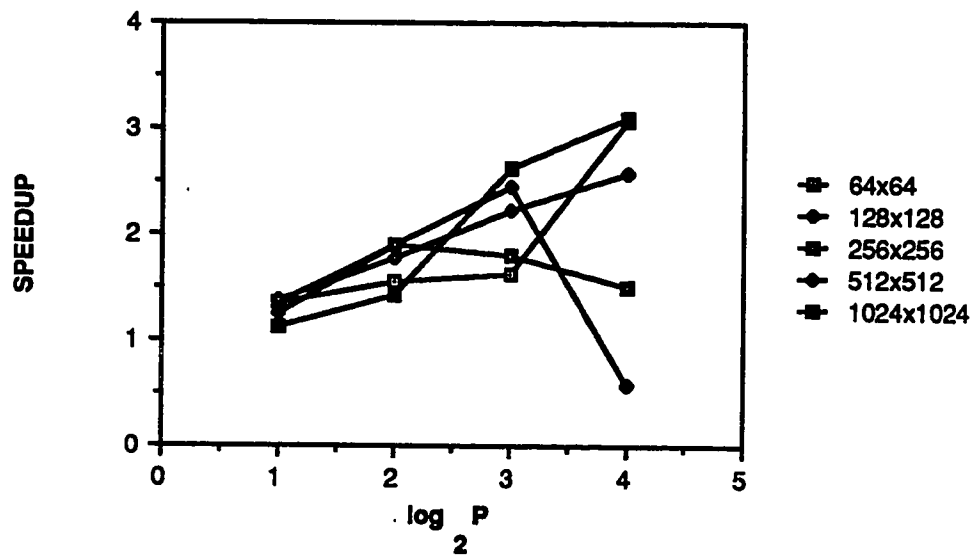


a. 32Kbyte Cache Size

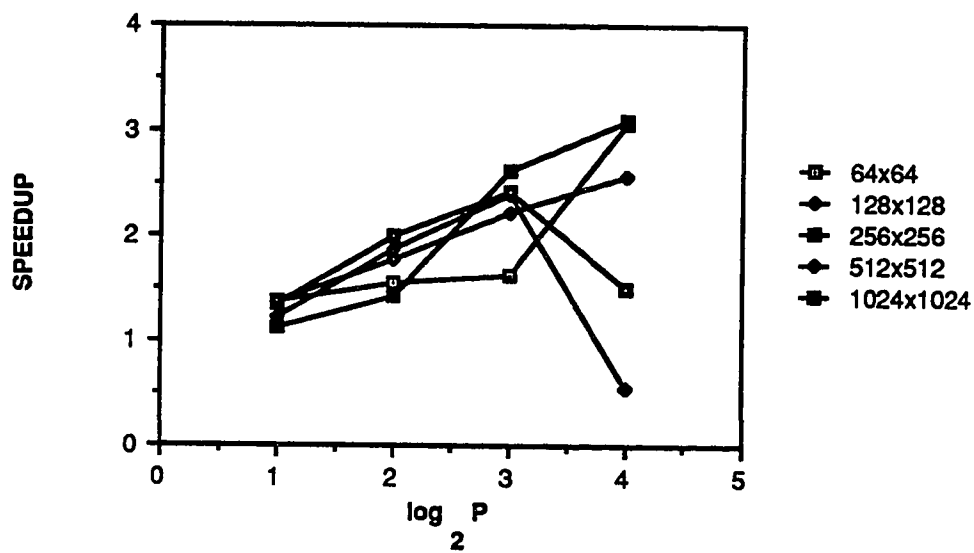


b. 64Kbyte Cache Size

Figure 4.48 Speedup versus the Number of Processors, Tagged Prefetching, Square Decomposition, Both Cache Sizes and Mapping Functions, All Grid Sizes, 32-byte Blocks.



a. 32Kbyte Cache Size



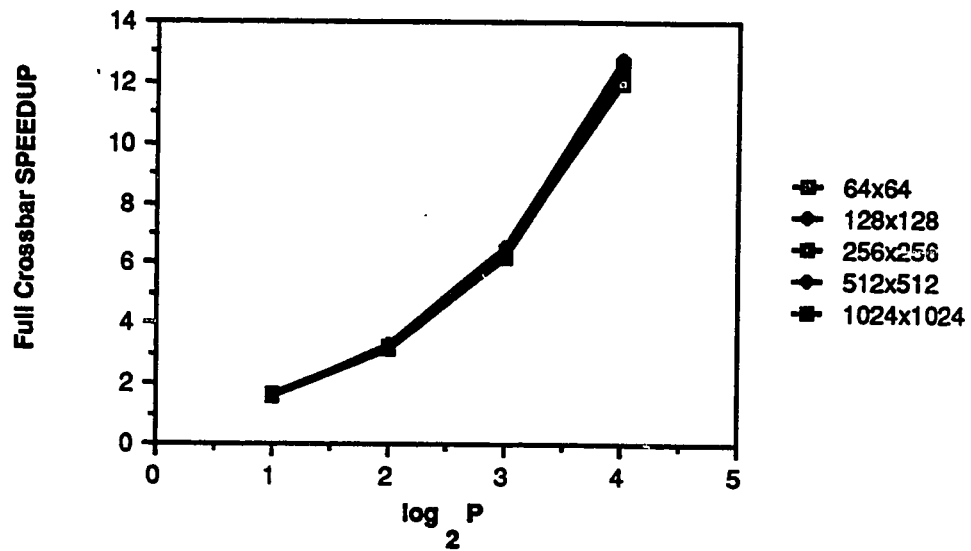
b. 64Kbyte Cache Size

Figure 4.49 Speedup versus the Number of Processors, Tagged Prefetching, Rectangular Decomposition, Both Cache Sizes and Mapping Functions, All Grid Sizes, 32-byte Blocks.

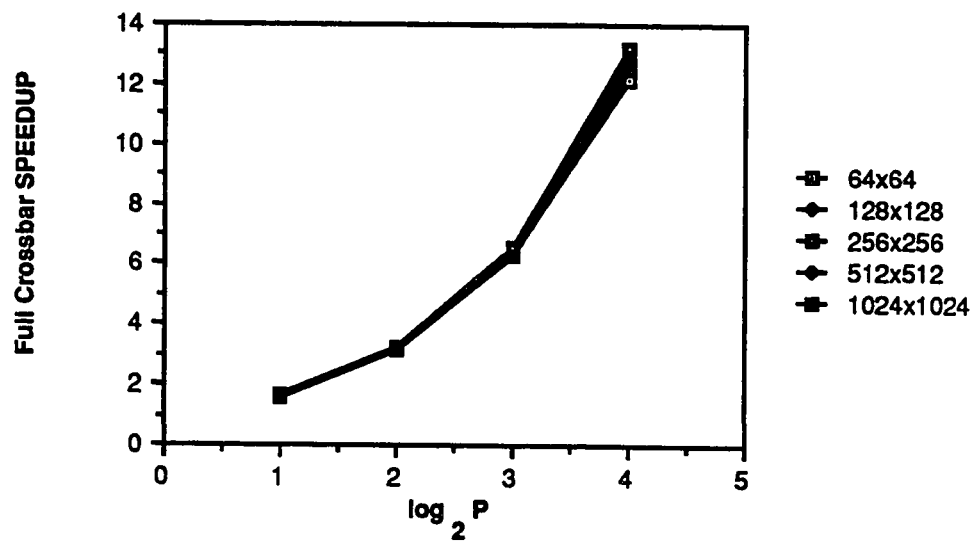
Figure 4.50 shows the speedup versus the number of processors for the demand fetching policy, both cache sizes and mapping functions, the full crossbar interconnection, and all grid sizes. The curves show that the speedup is independent of the PDE grid size with a range from slightly less than 2 to 13 as the number of processors vary from 2 to 16. Observe that the rectangular decomposition strategy is minimally higher than the square partition.

Figures 4.51 and 4.52 present the full crossbar speedups for the prefetching-on-miss and tagged prefetching fetching strategies, respectively. With the exception of the 128x128 and 256x256 point grid sizes, these speedup curves are similar to the demand fetching speedups. The 128x128 and 256x256 point grid sizes have smaller speedups for the 16 processor system for prefetch-on-miss strategy (in comparison with demand fetching speedups) and even smaller values for tagged prefetching.

Figure 4.53 displays the iteration time degradation versus the number of processors for both cache sizes, mapping functions and interconnection networks. The square decomposition strategy is used and a 32-byte blocksize is assumed. Both mapping functions show a range between 1.05 and 1.45 for the 64x64 point PDE grid size when using the single bus interconnection network. This is the largest degradation present, a direct result two factors. Firstly, the infinite MRD for the 64x64 point grid size results in a substantial increase in the enabled coherence effective memory access time over the disabled coherence access time. Secondly, the additional waiting time resulting from the single bus system contributes to the iteration time degradation. This waiting time is not a factor for the full crossbar system.

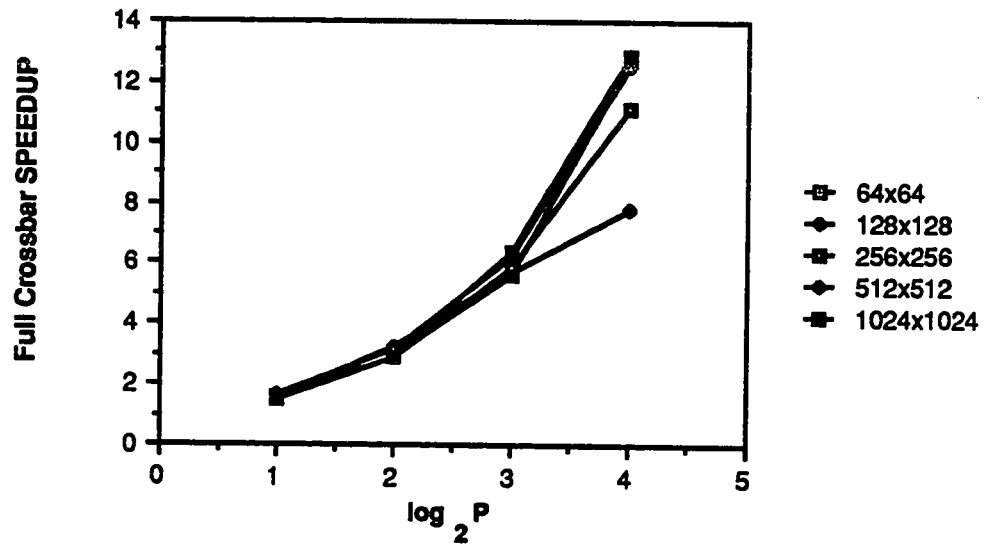


a. Square Decomposition

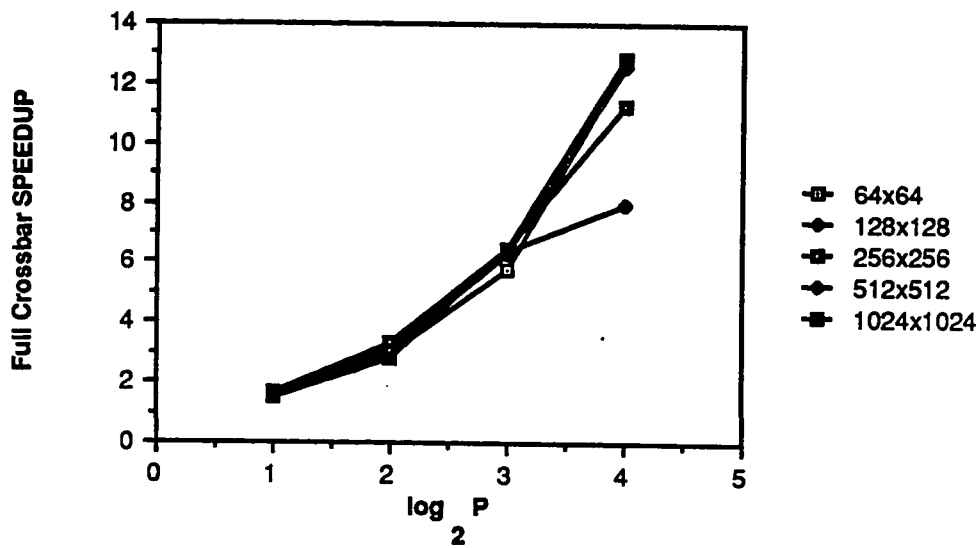


b. Rectangular Decomposition

Figure 4.50 Speedup versus the Number of Processors, Demand Fetching, Both Cache Sizes and Mapping Functions, 32-byte Blocksize, All Grid Sizes.

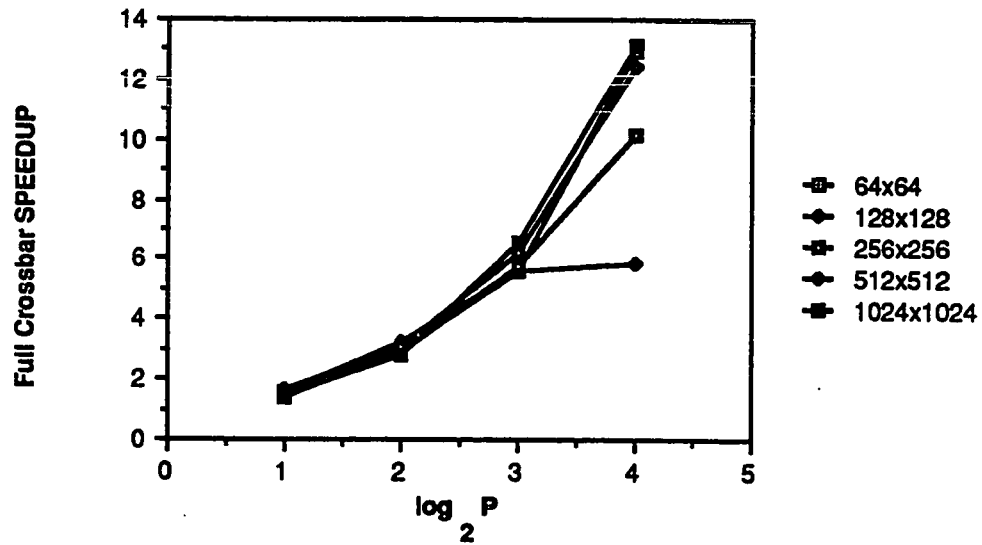


a. Square Decomposition

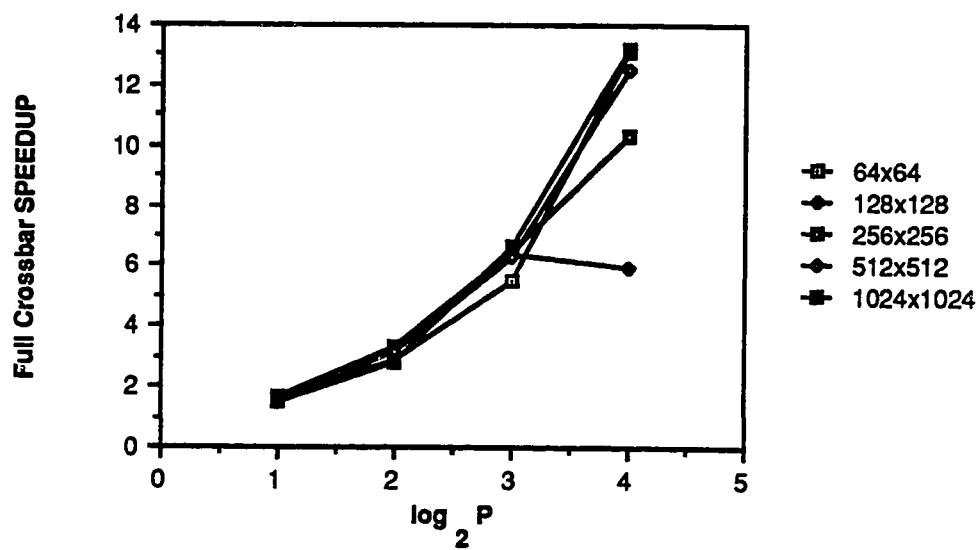


b. Rectangular Decomposition

Figure 4.51 Speedup versus the Number of Processors, Prefetch-on-miss, Both Cache Sizes and Mapping Functions, 32-byte Blocksize, All Grid Sizes.

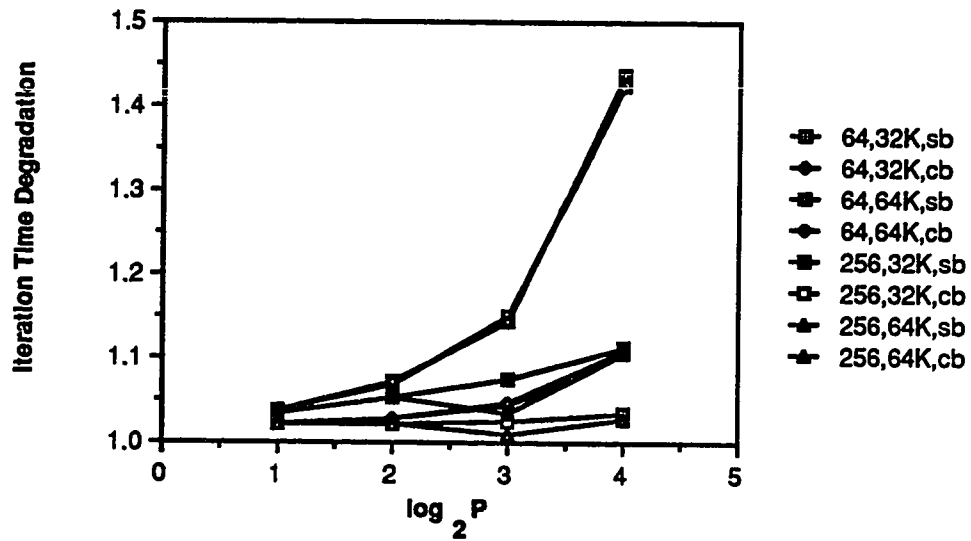


a. Square Decomposition

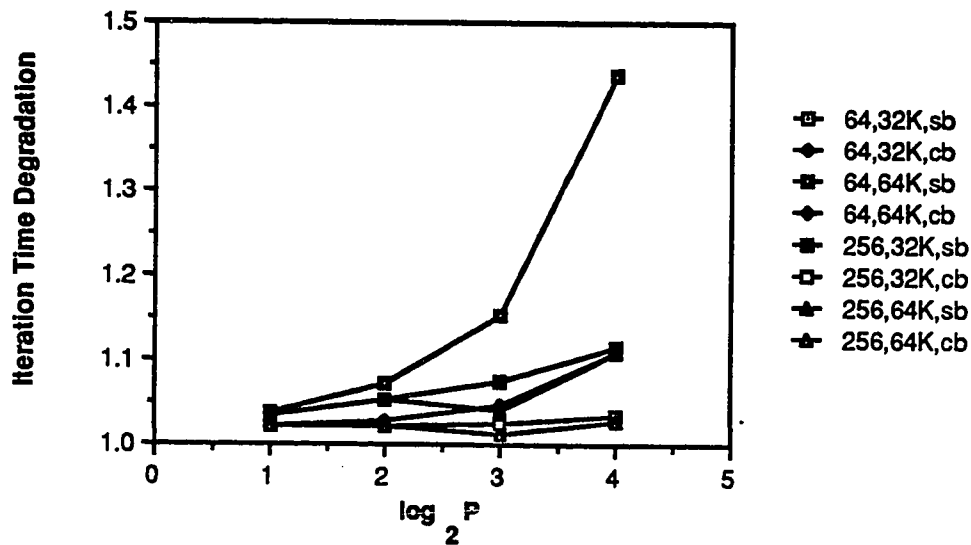


b. Rectangular Decomposition

Figure 4.52 Speedup versus the Number of Processors, Tagged Prefetching, Both Cache Sizes and Mapping Functions, 32-byte Blocksize, All Grid Sizes.



a. Direct Mapping



b. Set-Associative Mapping

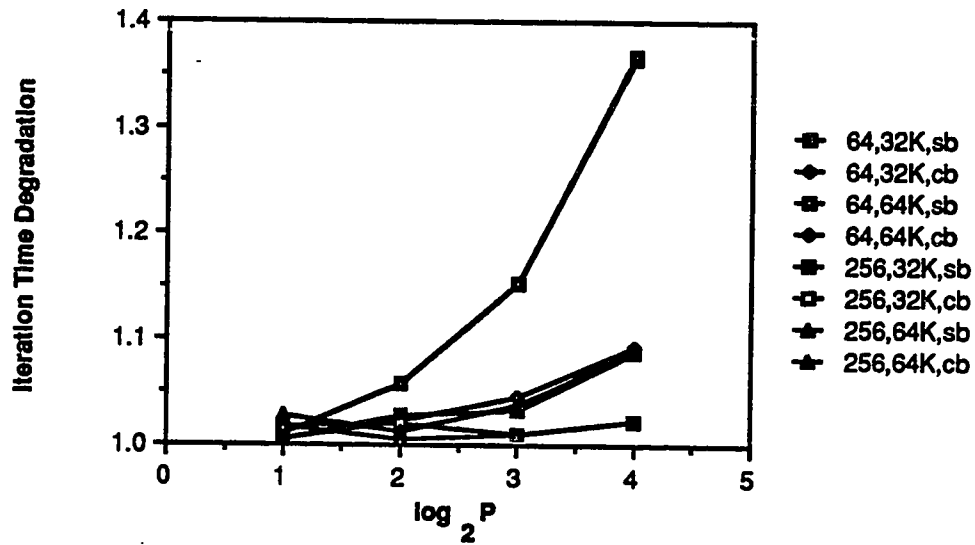
Figure 4.53 Iteration Time Degradation versus the Number of Processors, Both Cache Sizes, Mapping Functions and Interconnection Networks, Square Decomposition.

For PDE grid sizes larger than 64×64 points, the iteration time degradation range is between 1.01 to 1.1. Both mapping strategies show this degradation is greater for single bus systems as discussed above. Figure 4.54 presents the iteration time degradation for the rectangular decomposition strategy. The behavior of the curves shown in this graph are very similar to the square decomposition curves; however, with a range between 1.02 and 1.38, the degradation for the 64×64 point PDE grid size is smaller. This is because for the 32-byte blocksize, the rectangular MR and XIs are slightly higher than their square decomposition counterparts.

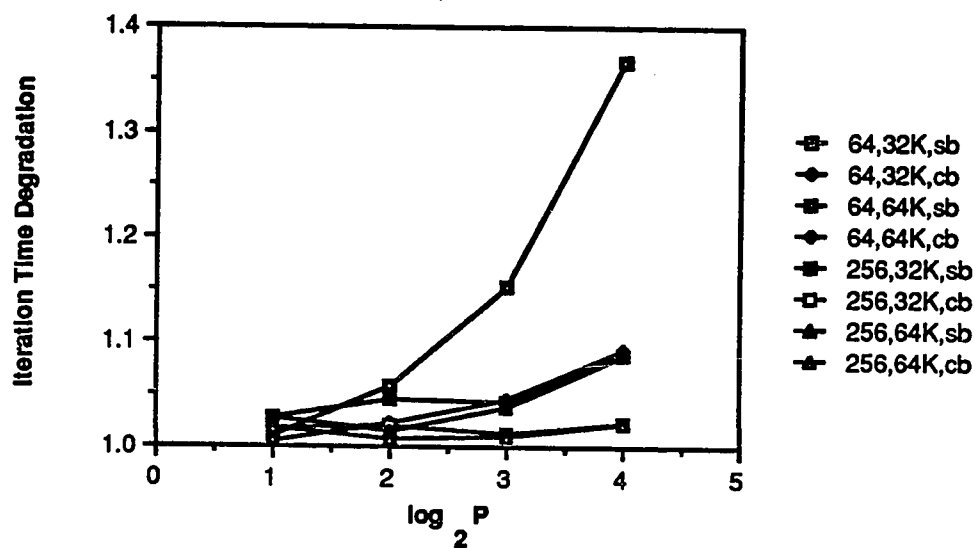
4.4. SOR Conclusions

When demand fetching is used the MR generally decreases as the cache block-size increases. The value usually increases as the number of processors increase for smaller grid sizes and is independent of the number of processors in the system for larger grid sizes. For smaller grid sizes its value is small (not greater than 0.03); however, larger grid sizes produce MRs up to 0.2. The direct mapping function has a better MR than set-associative mapping while the square decomposition strategy results in a better MR for lower blocksizes. On the other hand the rectangular partition produces a better MR for larger blocksizes. For the most part, the MRD is infinite for the 64×64 and 128×128 point grid sizes and 1.0 for the larger grid sizes.

The IR and prXICO also decrease as the cache blocksize increases but increase as the number of processors increase; however, their values are generally lower than the MR values. Furthermore, the IR and prXICO curves are very similar in nature. This is because they occur as a result of accessing the same shared blocks. Moreover,



a. Direct Mapping



b. Set-Associative Mapping

Figure 4.54 Iteration Time Degradation versus the Number of Processors, Both Cache Sizes, Mapping Functions and Interconnection Networks, Rectangular Decomposition.

the prXICS also has similar properties but its value is generally lower than all of the other parameters considered.

Introducing prefetching strategies produces similar behavior for the MR curves with better performance for tagged prefetching, followed by prefetch-on-miss. In contrast, while the prefetching IR, prXICO and prXICS parameters result in behavior similar to their demand fetching counterparts, the tagged prefetching values exhibit the worst performance, followed by prefetch-on-miss and finally demand fetching. The lookup ratio and prefetch ratio generally increase as the number of processors increase for smaller grid sizes and are independent of the processor configuration for larger grid sizes. Therefore, their behavior parallels the MR. In fact, the LR is identical to the MR for the prefetch-on-miss fetching strategy. The tagged prefetching LR is typically higher than the prefetch-on-miss LR. This also holds for the PR parameters. Finally, the PR is normally lower than the LR.

With a lower speedup bound ranging from 1.25 to 3 for 2 to 16 processors, respectively, the single bus interconnection network has prohibitive potential. Utilizing the prefetching strategies does not improve this speedup, in fact the parameter decreases for some grid sizes. In comparison, an upper bound on the speedup for the full crossbar interconnection ranges from slightly less than 2 to 13 for 2 to 16 processors. However, the prefetching strategies do not improve upon this performance.

The enabled coherence protocol results in a SOR iteration time that is between 1.01 and 1.10 times the disabled protocol time for PDE grid sizes larger than 64x64 points, and a 32-byte blocksize. The smaller grid enabled coherence protocol runs

between 1.05 and 1.45 times the disabled protocol time. The single bus degradation time is intuitively higher than the full crossbar degradation time. The mapping strategy and the decomposition strategy have no appreciable effects on the iteration time degradation.

In summary, the MR, IR and prXICO are the major causes of performance degradation for SOR. While the prefetching strategies lower the MR they increase the IR, prXICO and prXICS. This results in lower speedup for some PDE grid sizes. The rectangular decomposition strategy produces a minimally higher speedup, a direct result of a slight improvement in all parameters studied (over the square decomposition strategy). Finally, the direct mapping function counterintuitively performs better for several PDE grid sizes considered. All in all, cache coherence results in a iteration time degradation of less than 1.5.

CHAPTER 5

THE PRECONDITIONED CONJUGATE GRADIENT ALGORITHM

The point Jacobi and SOR approaches to the solution of the equation shown below:

$$Ax = b \quad (5.1)$$

where A is a large $M \times M$ symmetric positive definite matrix, is to transform the problem as shown below:

$$Cx = Dx + b \quad (5.2)$$

Here, $A = C - D$ and the problem is solved iteratively. Another transformation of Equation 5.1 is done by setting the situation up as a minimization problem. One such minimization is known as the conjugate gradient problem [43]. This chapter begins by briefly discussing the preconditioned conjugate gradient algorithm (PCG) as well as the parallel implementation used in this study. The performance parameters obtained from the simulation of this parallel algorithm are then presented, followed by the multiprocessor speedup curves, the iteration time degradation and finally, PCG's conclusions.

5.1. Preconditioned CG

The CG algorithm, is derived in [43]; however, its convergence rate is a function of the condition number [44]. For the matrix systems considered in this study the condition number is poor and therefore the convergence rate for CG is unaccept-

able. This rate is improved by using a preconditioning matrix M with properties identical to A such that $A = M - N$. This results in the PCG algorithm shown in Table 5.1. Unfortunately, this algorithm exhibits a very low degree of parallelism. Some attempts have been made to modify PCG to accommodate more parallelism [45]. The modified PCG (MPCG) used in this study is derived by Meurant in [46] for block tridiagonal linear systems. This MPCG algorithm is shown in Table 5.2. The preconditioning matrix M for MPCG was chosen as follows:

$$r^1 = b - Ax^1$$

$$p^0 \text{ arbitrary}$$

For $k = 1, 2, \dots$

$$Mz^k = r^k$$

$$b_k = (Mz^k, z^k) / (Mz^{k-1}, z^{k-1})$$

$$b_1 = 0$$

$$p^k = z^k + b_k p^{k-1}$$

$$a_k = (Mz^k, z^k) / (Ap^k, p^k)$$

$$x^{k+1} = x^k + a_k p^k$$

$$r^{k+1} = r^k - a_k Ap^k$$

Table 5.1 The Preconditioned Conjugate Gradient Algorithm.

$$r^1 = b - Ax^1$$

$$Mz^1 = r^1$$

$$p^1 = z^1$$

For $k = 1, 2, \dots$

$$Mv^k = Ap^k$$

$$a_k = (r^k, z^k) / (Ap^k, p^k)$$

$$s_{k+1} = a_k^2 (v^k, Ap^k) - (r^k, z^k)$$

$$b_{k+1} = s_{k+1} / (Mz^{k-1}, z^{k-1})$$

$$x^{k+1} = x^k + a_k p^k$$

$$r^{k+1} = r^k - a_k Ap^k$$

$$z^{k+1} = z^k - a_k v^k$$

$$p^{k+1} = (z^k - a_k v^k) + b_{k+1} p^k$$

Table 5.2 The Modified Preconditioned Conjugate Gradient Algorithm.

$$M = \Pi \Delta^{-1} \Pi$$

(5.3)

where Π is shown below for two processors.

$$\Pi = \begin{bmatrix} \Delta_1 & 0 & & & & & & & & & \\ A_2 & \Delta_2 & & & & & & & & & \\ & \cdot & \cdot & & & & & & & & \\ & & \cdot & 0 & & & & & & & \\ & & & A_{n/2} & \Delta_{n/2} & A_{n/2+1}^T & & & & & \\ & & & 0 & \Delta_{n/2+1} & A_{n/2+2}^T & & & & & \\ & & & & \cdot & \cdot & & & & & \\ & & & & & \Delta_{n-1} & A_n^T & & & & \\ & & & & & 0 & \Delta_n & & & & \end{bmatrix}$$

Δ is a block tridiagonal matrix with diagonal blocks Δ_i , $i=1, \dots, n$. The matrices Δ_i are computed as shown below:

$$\Delta_1 = D_1,$$

$$\Delta_i = D_i - A_i \Omega_{i-1}(1) A_i^T, \quad 2 \leq i \leq n/2,$$

$$\Delta_n = D_n,$$

$$\Delta_i = D_i - A_{i+1}^T \Omega_{i+1}(1) A_{i+1}, \quad i = n, \dots, n/2 - 1.$$

$\Omega(j)$ is a symmetric banded matrix with $2j+1$ diagonals whose elements are the same as those of Δ_i^{-1} . Therefore the parallel solution of $My = c$ [46] is:

$$w_1 = \Omega_1(3)c_1,$$

$$w_i = \Omega_j(3)(c_i - A_i w_{i-1}), \quad i = 2, \dots, n/2 - 1,$$

$$w_n = \Omega_n(3)c_n,$$

$$\begin{aligned}
w_i &= \Omega_i(3)(c_i - A_{i+1}^T w_{i+2}), \quad i = n-1, \dots, n/2 + 1, \\
w_{n/2} &= \Omega_{n/2}(3)(c_n - A_{n/2} w_{n/2} - A_{n/2+1}^T w_{n/2+1}), \\
y_{n/2} &= w_{n/2} \\
y_i &= w_i - \Omega_i(3)A_{i+1}^T y_{i+1}, \quad i = n/2 - 1, \dots, 1, \\
y_i &= w_i - \Omega_i(3)A_i y_{i-1}, \quad i = n/2 + 1, \dots, n.
\end{aligned}$$

The MPCG for a two processor system is implemented as shown in Tables 5.3a and 5.3b for processors P_1 and P_2 , respectively.

Both processors compute their respective sections of c_i , w_i , s^1 and s^2 and then synchronize. Then the $n/2$ component of v^k is evaluated by P^1 . After another barrier synchronization, the processors compute their portions of v^k and s^3 . After a third synchronization, each processor computes its own copy of the scalars s . Finally, the x , r , z and p vectors are computed by the processors and the processors are synchronized before beginning the next iteration. Similar decompositions of Π and the algorithm are used for four, eight and sixteen processors.

5.2. Simulation Results

5.2.1. Miss Ratio/Miss Ratio Degradation

Figure 5.1 shows the MR versus the cache blocksize for both mapping strategies and cache sizes, all processor configurations and the 64x64 and 128x128 (64Kbyte cache only) point grid sizes. These curves show a decrease in the MR as the cache blocksize increases from 8 to 32 bytes and a relatively constant value for larger blocksizes. The 32Kbyte cache size exhibits a larger MR than the 64Kbyte cache for

$$c_i = \begin{bmatrix} Ap^k \end{bmatrix}_i, \quad i = 1, \dots, n/2$$

$$w_i, \quad i = 1, \dots, n/2 - 1$$

$$s_1^1 = \sum_{i=1}^{n/2} \left(\begin{bmatrix} Ap^k \end{bmatrix}_i, \begin{bmatrix} p^k \end{bmatrix}_i \right)$$

$$s_1^2 = \sum_{i=1}^{n/2} \left(\begin{bmatrix} r^k \end{bmatrix}_i, \begin{bmatrix} z^k \end{bmatrix}_i \right)$$

$$\begin{array}{c} \text{SYNC} \\ \text{compute} \begin{bmatrix} v^k \end{bmatrix}_{n/2} \\ \text{SYNC} \end{array}$$

$$\begin{bmatrix} v^k \end{bmatrix}_i, \quad i = n/2 - 1, \dots, 1$$

$$s_1^3 = \sum_{i=1}^{n/2} \left(\begin{bmatrix} v^k \end{bmatrix}_i, \begin{bmatrix} Ap^k \end{bmatrix}_i \right)$$

$$\begin{array}{c} \text{SYNC} \\ s^1 = s_1^1 + s_2^1 \end{array}$$

$$s^2 = s_1^2 + s_2^2$$

$$s^3 = s_1^3 + s_1^3$$

$$\begin{bmatrix} x^{k+1} \end{bmatrix}, \begin{bmatrix} r^{k+1} \end{bmatrix}, \quad i = 1, \dots, n/2$$

$$\begin{bmatrix} z^{k+1} \end{bmatrix}, \begin{bmatrix} p^{k+1} \end{bmatrix}, \quad i = 1, \dots, n/2$$

$$\text{SYNC}$$

Table 5.3a MPCG Algorithm Executed by Processor P_1 .

$$c_i = \left[Ap^k \right]_i, \quad i = n/2 + 1, \dots, n$$

$$w_i, \quad i = n, \dots, n/2 + 1$$

$$s_2^1 = \sum_{i=n/2+1}^n \left(\left[Ap^k \right]_i, \left[p^k \right]_i \right)$$

$$s_2^2 = \sum_{i=n/2+1}^n \left(\left[r^k \right]_i, \left[z^k \right]_i \right)$$

SYNC

SYNC

$$\left[v^k \right]_i, \quad i = n/2 + 1, \dots, n$$

$$s_2^3 = \sum_{i=n/2+1}^n \left(\left[v^k \right]_i, \left[Ap^k \right]_i \right)$$

SYNC

$$s_1 = s_1^1 + s_2^1$$

$$s^2 = s_1^2 + s_2^2$$

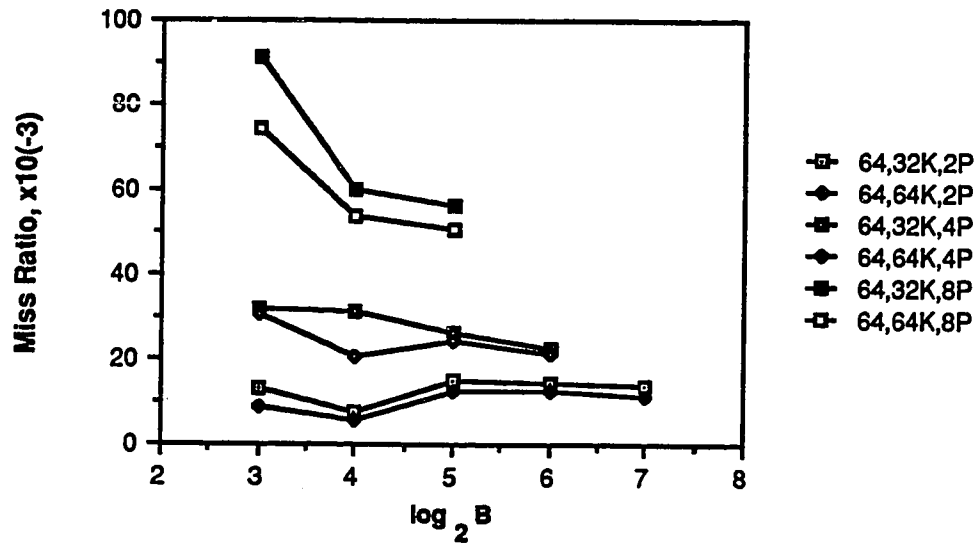
$$s^3 = s_1^3 + s_2^3$$

$$\left[x^{k+1} \right], \left[r^{k+1} \right], \quad i = n/2 + 1, \dots, n$$

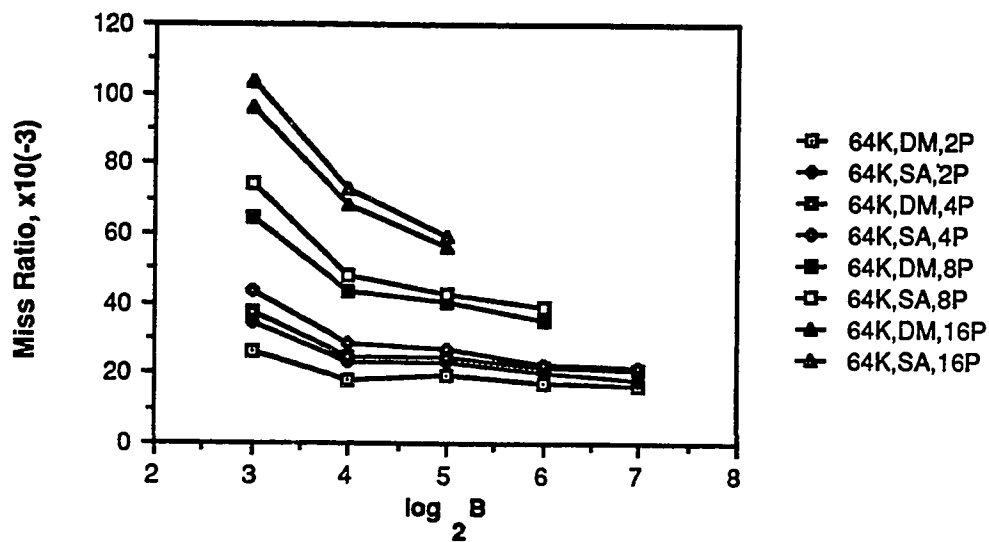
$$\left[z^{k+1} \right], \left[p^{k+1} \right], \quad i = n/2 + 1, \dots, n$$

SYNC

Table 5.3b MPCG Algorithm Executed by Processor P_2 .



a. 64x64 Point Grid Size



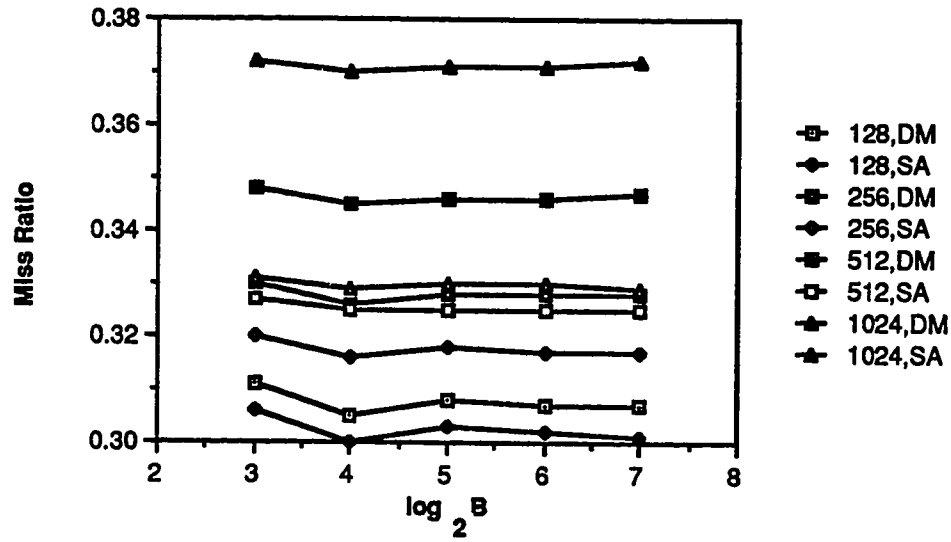
b. 128x128 Point Grid Size, 64Kbyte Cache Size

Figure 5.1 Miss Ratio versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Smaller Grid Sizes, All Processor Configurations.

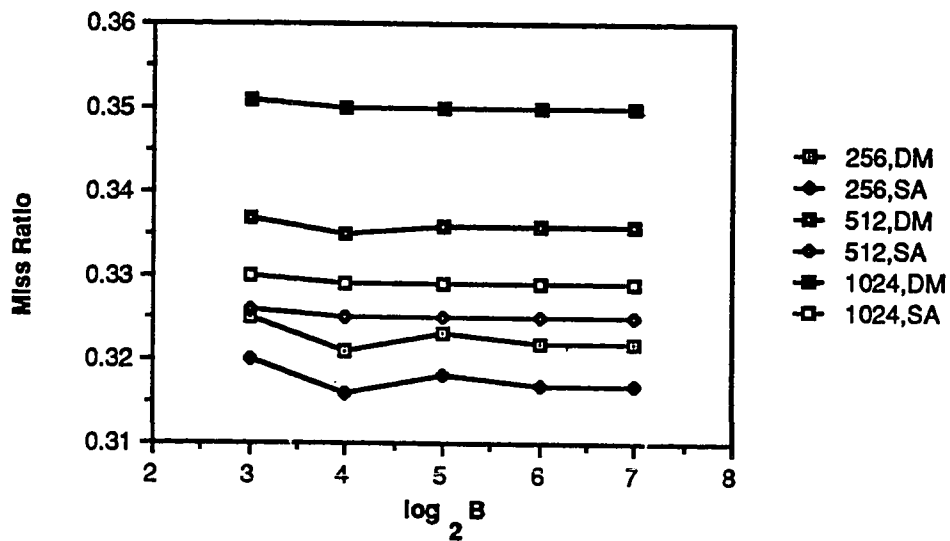
the 64x64 point grid size. Also observe an increase in the MR as the number of processors and the PDE grid size increase. For the 128x128 point grid size, the direct mapping strategy has slightly better performance than the set-associative mapping strategy. This is attributed to the contention resulting from using the LRU replacement policy for 2-way set-associativity and this algorithm. The MR for some of the larger cache sizes are not shown because their values increase substantially. This is largely due to the placement contention among the small number of larger blocks.

Figure 5.2 presents the MR versus the cache blocksize for the larger grid sizes and a two processor system. The parameter is generally constant as the blocksize varies and its values are larger than the smaller grid size values. In fact, as the PDE grid size increases, the MR increases. The set-associative mapping strategy and the 64Kbyte cache sizes exhibits better performance for these larger grid sizes. This is in direct contrast to the smaller grid sizes where direct mapping has the better performance. This is because the contention as a result of using the LRU replacement policy is offset by the increased contention due to the larger grid sizes. This increased frequency is larger when using direct mapping. This also explains the relatively constant MR over the range of block sizes considered. All other processor configurations exhibit this same behavior.

Figure 5.3 illustrates the MR versus the number of processors for a 32-byte blocksize. For smaller grid sizes, this parameter increases as the number of processors increase. The values are independent of the blocksize for larger grid sizes. There is also an order of magnitude increase in the MR for the larger grid sizes. This results from the fact that the smaller grid sizes allow the cache to capture all reference

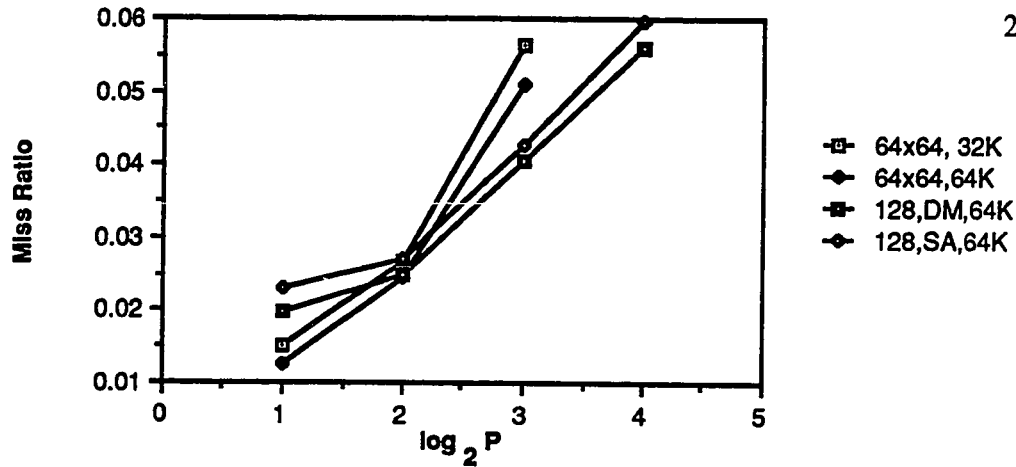


a. 32Kbyte Cache Size

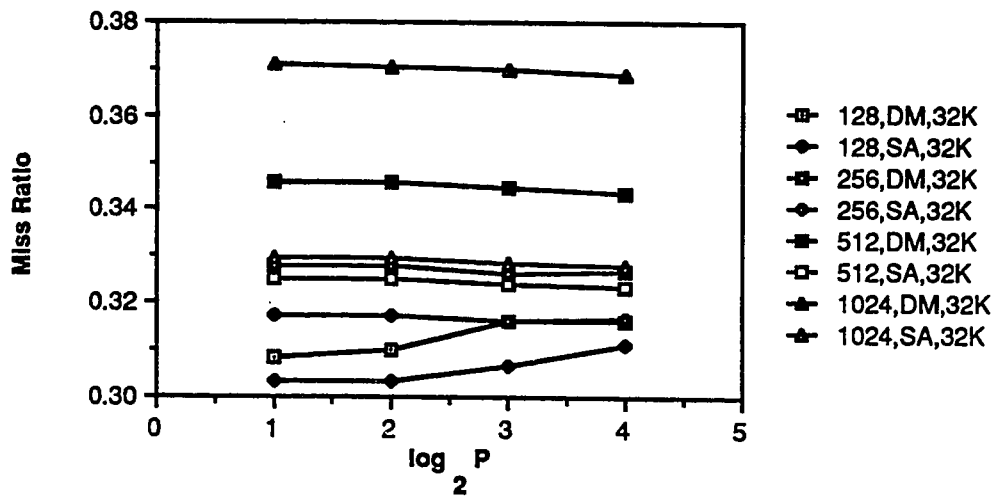


b. 64Kbyte Cache Size

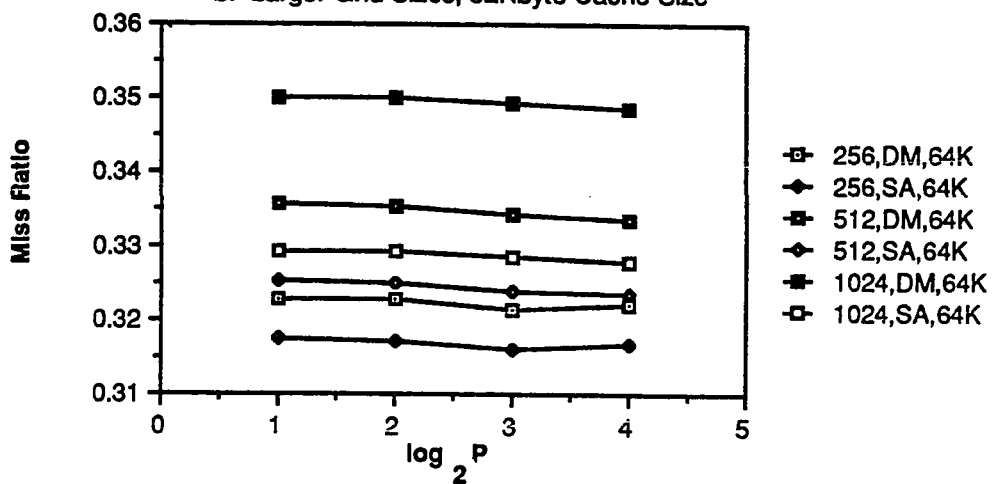
Figure 5.2 Miss Ratio versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, Larger Grid Sizes, Two Processors.



a. Smaller Grid Sizes



b. Larger Grid Sizes, 32Kbyte Cache Size



c. Larger Grid Sizes, 64Kbyte Cache Size

Figure 5.3 Miss Ratio versus the Number of Processors, Both Mapping Strategies and Cache Sizes, All Grid Sizes and Processor Configurations, 32-byte Blocksize.

localities. This is also demonstrated by the MRD shown in Table 5.4. Here all misses are a direct result of invalidations for the 64x64 point grid size. The 128x128 point grid size (64Kbyte cache) also has a relatively higher MRD.

5.2.2. Invalidation Ratio

Figure 5.4 displays the IR versus the cache blocksize for both mapping functions and cache sizes, all grid sizes and two (a) and four (b) processors. This parameter decreases as the cache blocksize and PDE grid sizes increase. The set-associative IRs are slightly higher than the direct mapping IRs. This is because the lower MR for set-associative mapping enables more invalidateable blocks to be present in the cache. The sharp increase in the IR as the blocksize increases from 64 to 128 bytes (four processor system) results from the fact that the 128-byte block is so large that a large percentage of the cacheable blocks are shared and modifiable.

Figure 5.5 shows this same IR versus cache blocksize for eight (a) and sixteen (b) processor systems. These curves also show a decrease in value as the cache blocksize increases. Figure 5.6 presents the IR versus the number of processors for both mapping functions and cache sizes, all grid sizes and a 32-byte cache blocksize. The figure shows an increase in the IR as the number of processors increase. Also observed is the relative independence of the IR on the cache mapping function.

5.2.3. prXICO

Figure 5.7 illustrates the prXICO versus the cache blocksize for both mapping strategies and cache sizes, all grid sizes and two (a) and four (b) processors. The graphs show a decrease in value as the blocksize varies from 8 to 16 bytes, an

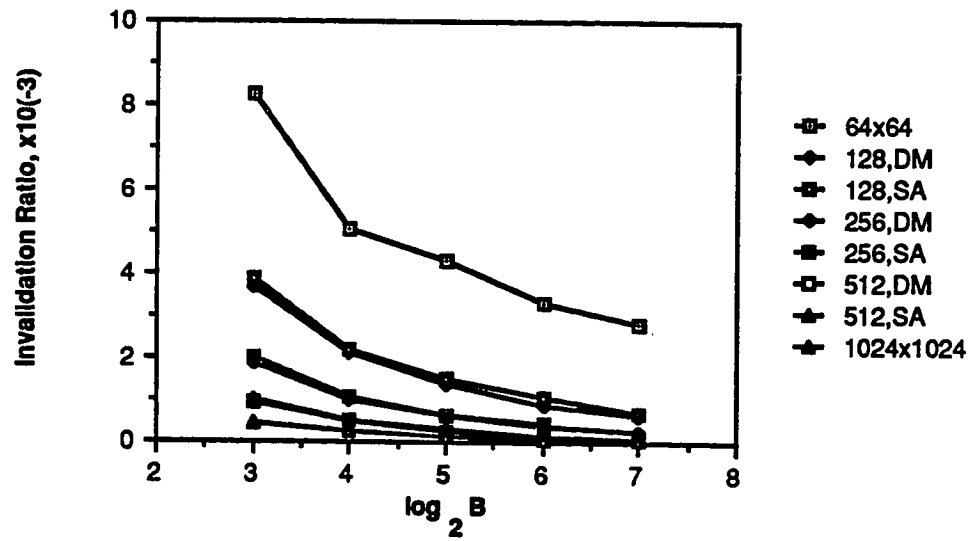
Cache Size	CPUs	PDE Gridsize				
		64	128	256	512	1024
32Kbytes	2	∞	1.02	1.01	1.00	1.00
	4	∞	1.03	1.01	1.01	1.00
	8	∞	1.07	1.03	1.01	1.00
	16		1.16	1.05	1.02	1.01
64Kbytes	2	∞	1.49	1.01	1.00	1.00
	4	∞	1.82	1.02	1.01	1.00
	8	∞	2.21	1.03	1.01	1.00
	16		5.10	1.06	1.02	1.01

a. Direct Mapping

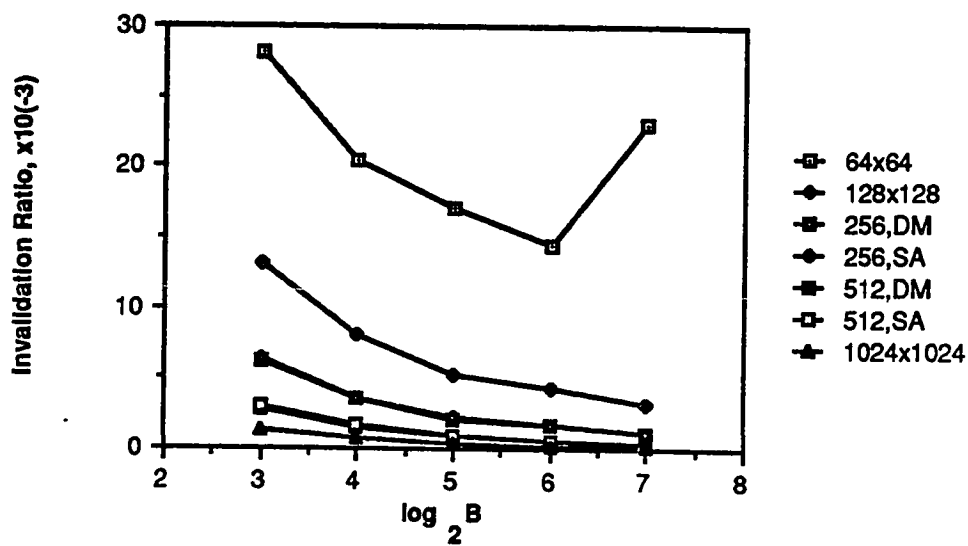
Cache Size	CPUs	PDE Gridsize				
		64	128	256	512	1024
32Kbytes	2	∞	1.02	1.01	1.01	1.00
	4	∞	1.04	1.02	1.01	1.00
	8	∞	1.07	1.03	1.01	1.00
	16		1.17	1.06	1.02	1.01
64Kbytes	2	∞	1.41	1.01	1.01	1.00
	4	∞	1.65	1.02	1.01	1.00
	8	∞	1.94	1.03	1.01	1.01
	16		4.60	1.06	1.02	1.01

b. Set-Associative Mapping

Table 5.4 MPCG Miss Ratio Degradation, 32-byte Blocksize.

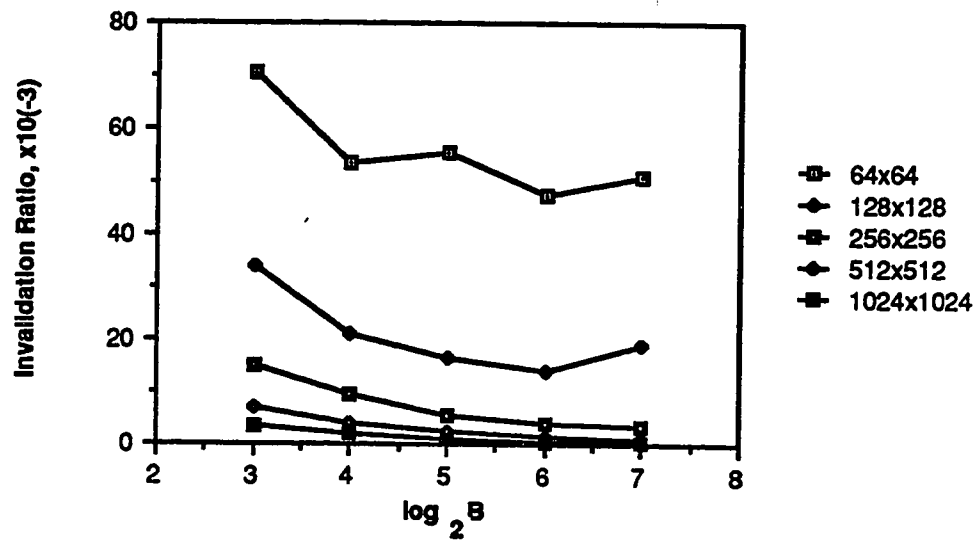


a. Two Processors

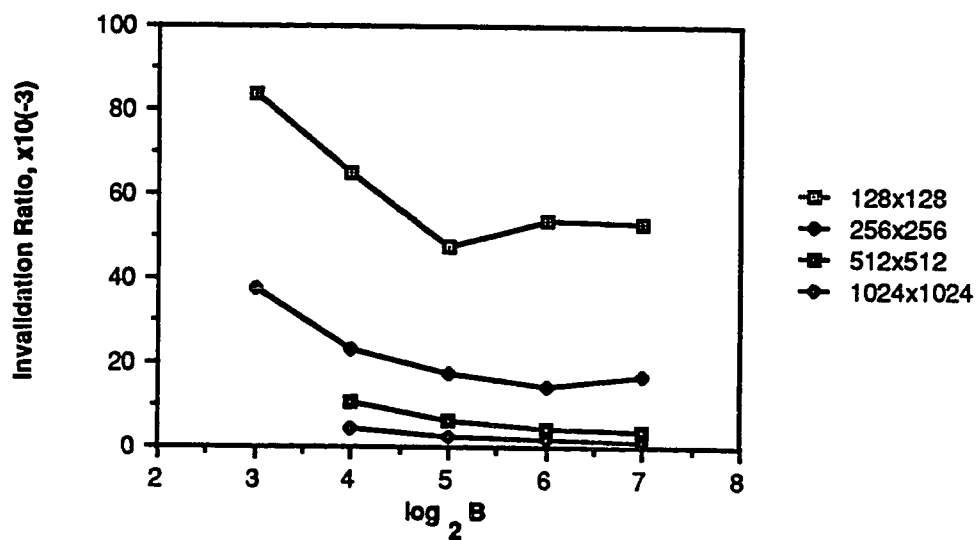


b. Four Processors

Figure 5.4 Invalidation Ratio versus Cache Blocksize, Both Mapping Functions and Cache Sizes, All Grid Sizes, Two and Four Processors.

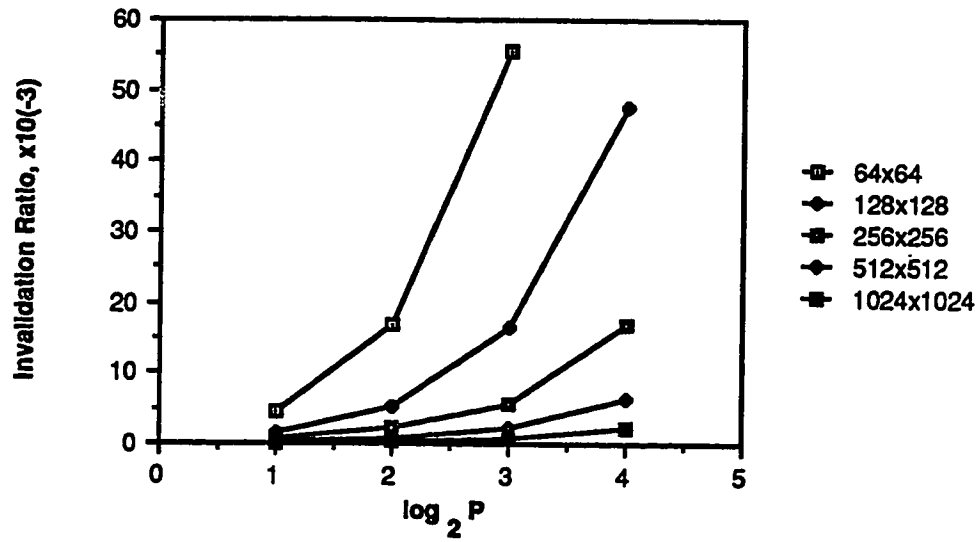


a. Eight Processors

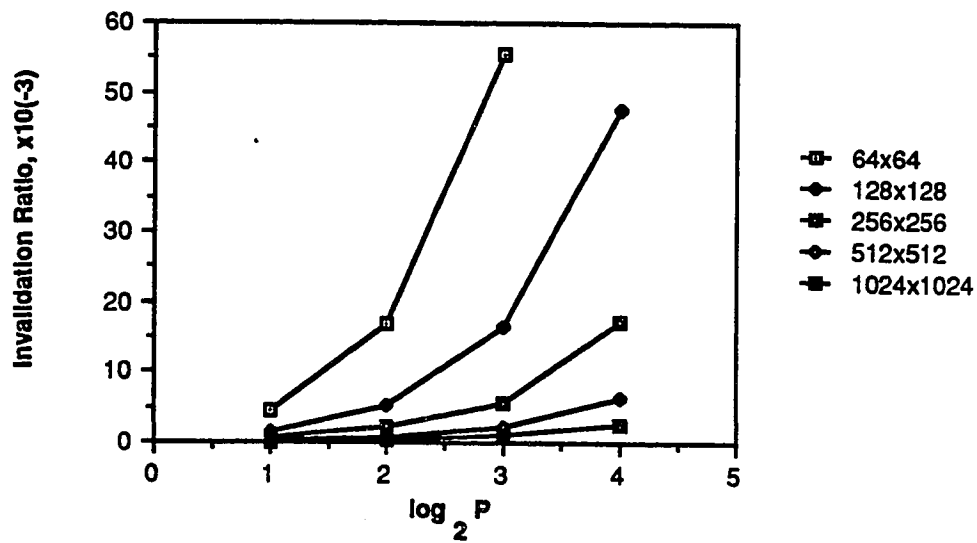


b. Sixteen Processors

Figure 5.5 Invalidation Ratio versus Cache Blocksize, Both Mapping Functions and Cache Sizes, All Grid Sizes, Eight and Sixteen Processors.

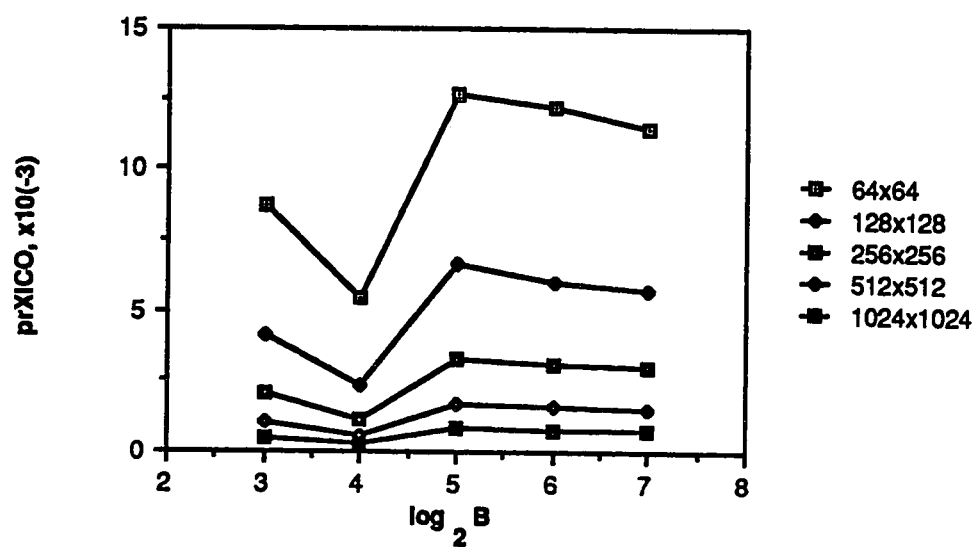


a. Direct Mapping

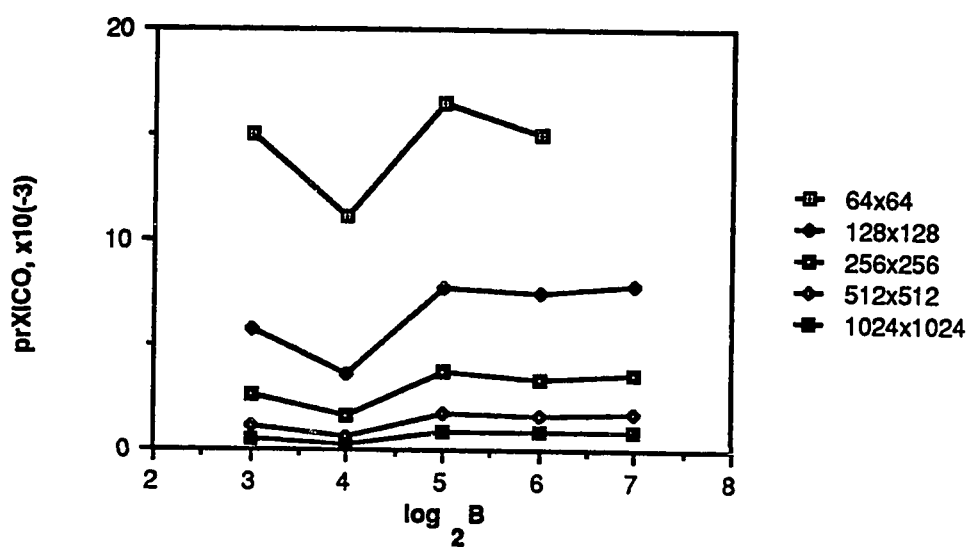


b. Set-Associative Mapping

Figure 5.6 Invalidation Ratio versus the Number of Processors, Both Mapping Functions and Cache Sizes, All Grid Sizes, 32-byte Blocksize.



a. Two Processors



b. Four Processors

Figure 5.7 prXICO versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, All Grid Sizes, Two and Four Processors.

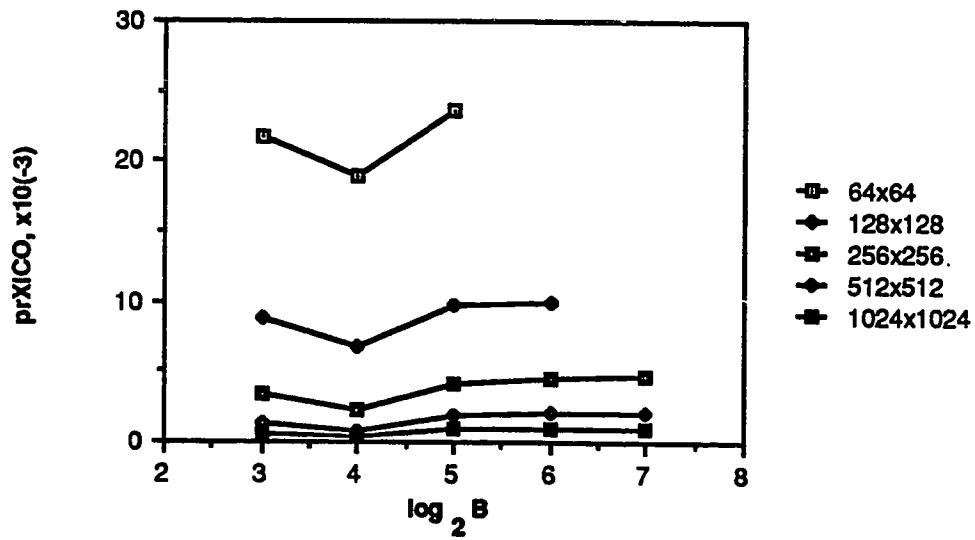
increase in value as the blocksize increases from 16 to 32 bytes and relatively small decreases in value as the blocksize varies from 32 to 128 bytes. The increase in this parameter as the blocksize increases from 16 to 32 bytes is attributed to the fact that the 32-byte cache size marks a transition that allows a shared block to be modified by more than one processor, increasing the number of XICOs performed per algorithm iteration.

Figure 5.8 presents the prXICO versus the cache blocksize for the eight (a) and sixteen (b) processor systems. The behavior of the graphs are similar to the two and four processor curves. The prXICO versus the number of processors for both mapping strategies and cache sizes, all grid sizes and a 32-byte blocksize is shown in Figure 5.9. Like the IR, this parameter increases as the number of processors increase.

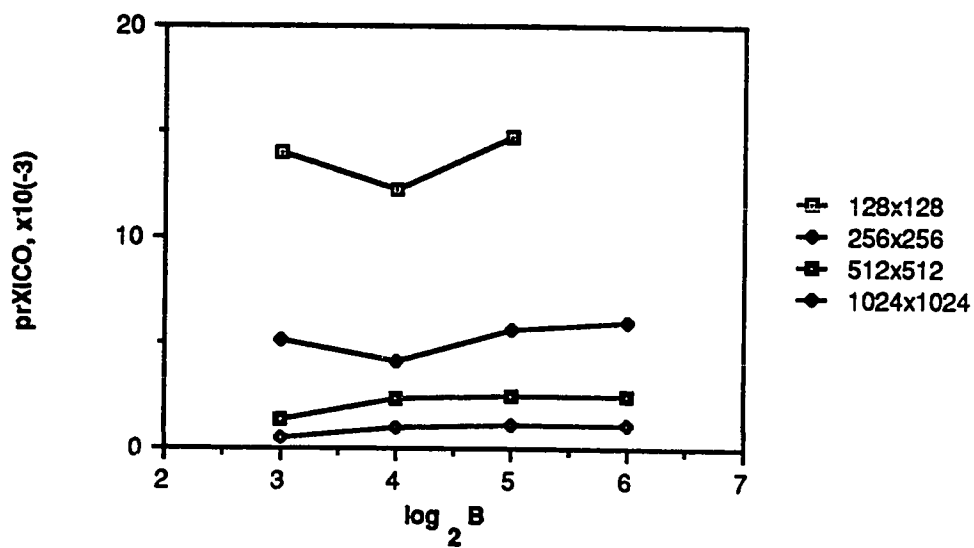
5.2.4. prXICS

Figure 5.10 presents the prXICS versus the cache blocksize for both mapping strategies and cache sizes, all grid sizes and two processors. The direct mapping curves generally decrease as the blocksize increases from 8 to 16 bytes followed by an increase in value for the larger grid sizes. This is because increased contention facilitated by direct mapping results in an increase in the number of RO->EXs performed per iteration. This is not present for the set-associative mapping strategy; therefore, the prXICS decreases as the blocksize increases. Also observe the decrease in this parameter as the cache size doubles.

Figure 5.11 displays the prXICS versus the cache blocksize and all other features considered for the four processor system. This parameter decreases in value



a. Eight Processors



b. Sixteen Processors

Figure 5.8 prXICO versus Cache Blocksize, Both Mapping Strategies and Cache Sizes All Grid Sizes, Eight and Sixteen Processors.

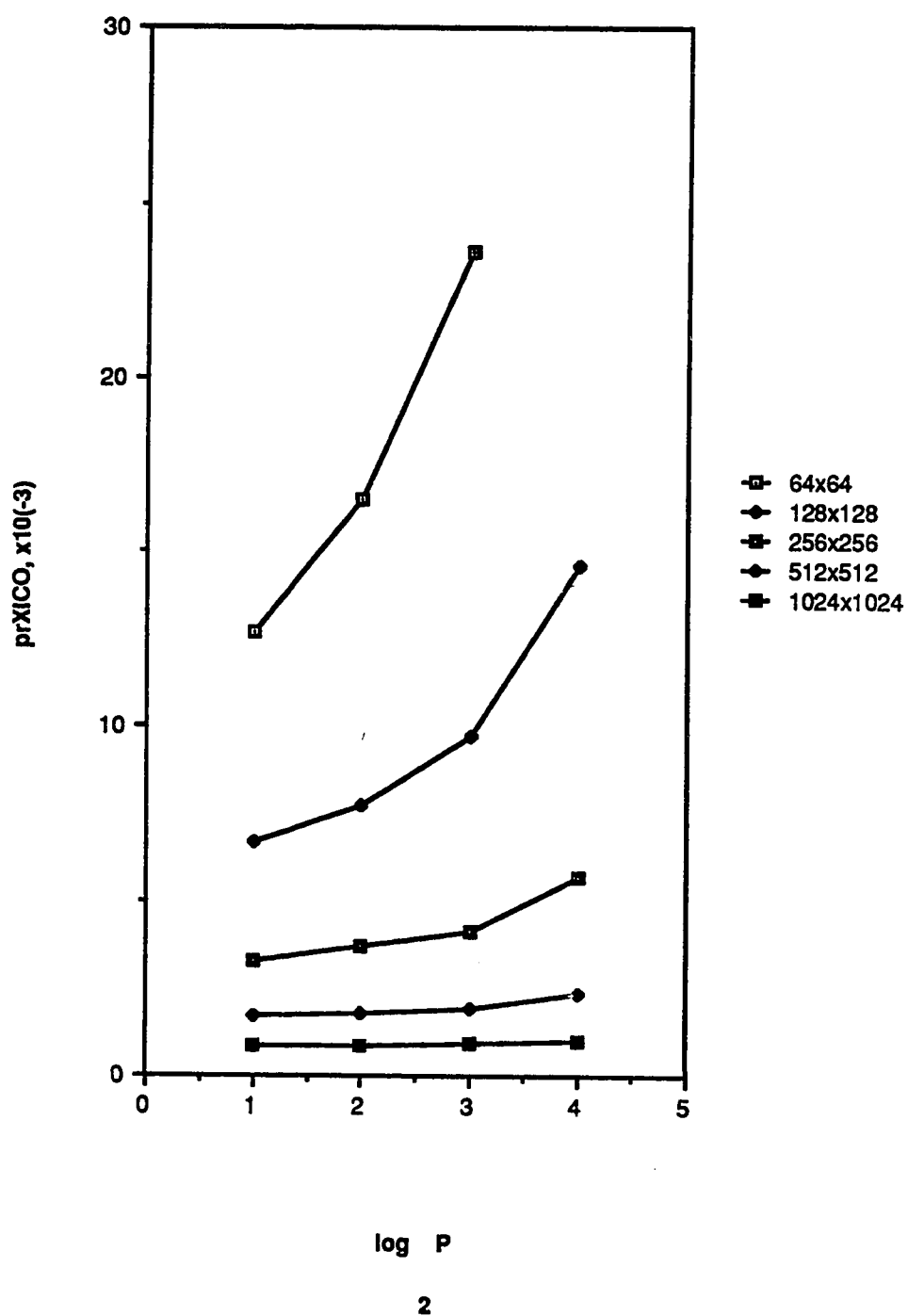
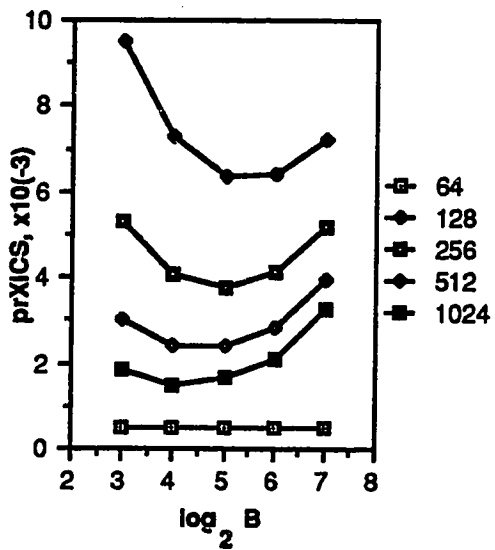
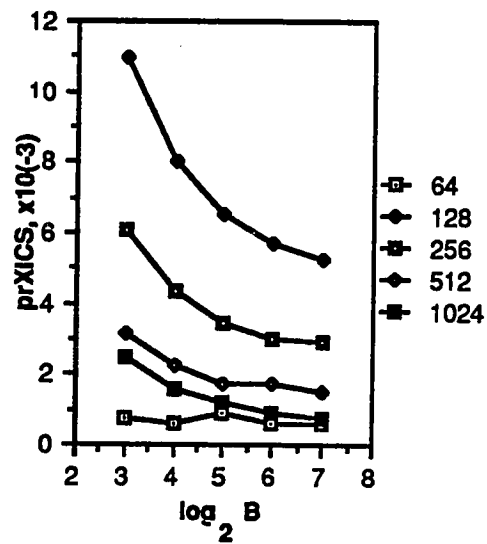


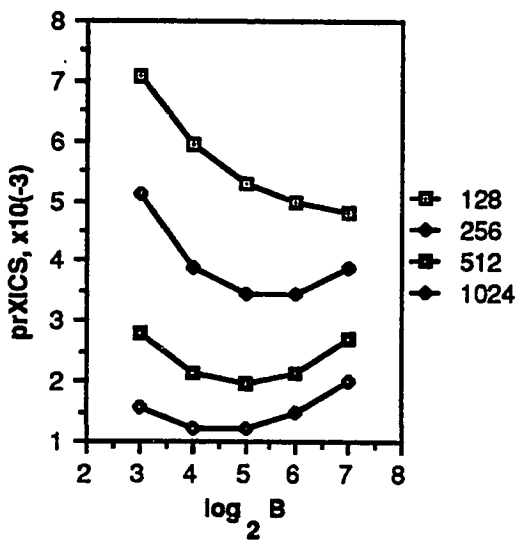
Figure 5.9 prXICO versus the Number of Processors, Both Mapping Strategies and Cache Sizes, All Grid Sizes, 32-byte Blocksize.



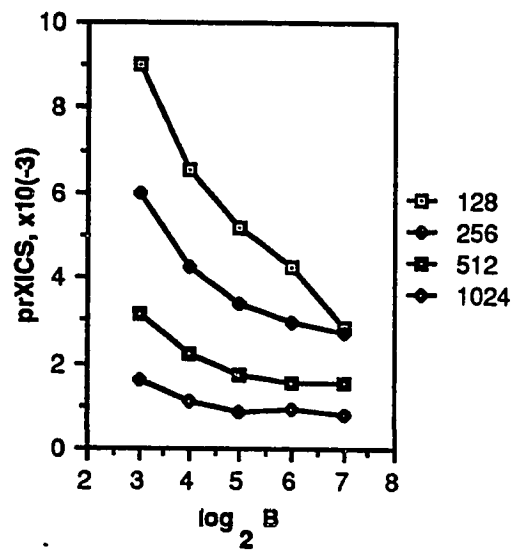
a. 32Kbyte Cache, Direct Mapping



b. 32Kbyte Cache, Set-Associative Mapping



c. 64Kbyte Cache, Direct Mapping



d. 64Kbyte Cache, Set-Associative Mapping

Figure 5.10 prXICS versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, All Grid Sizes, Two Processors.

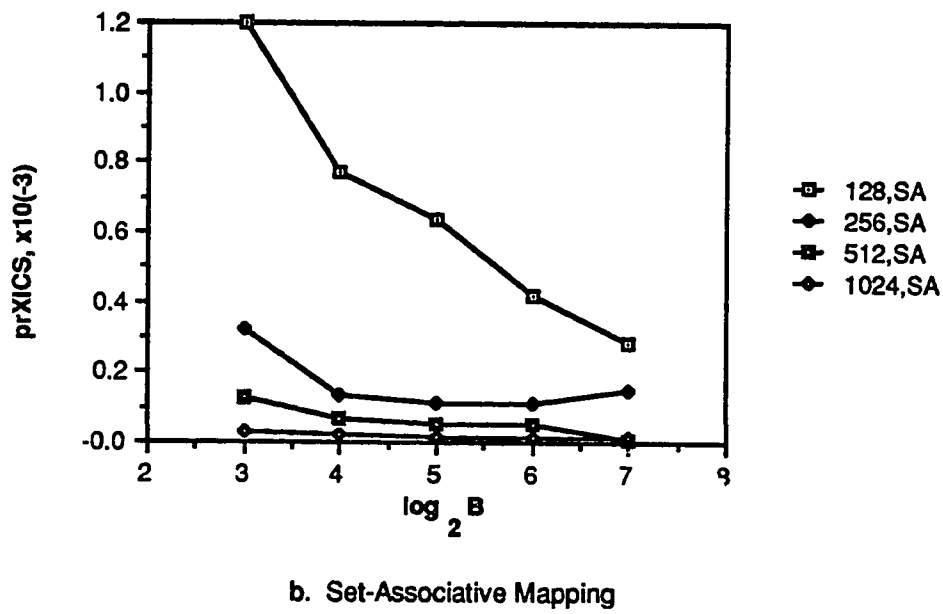
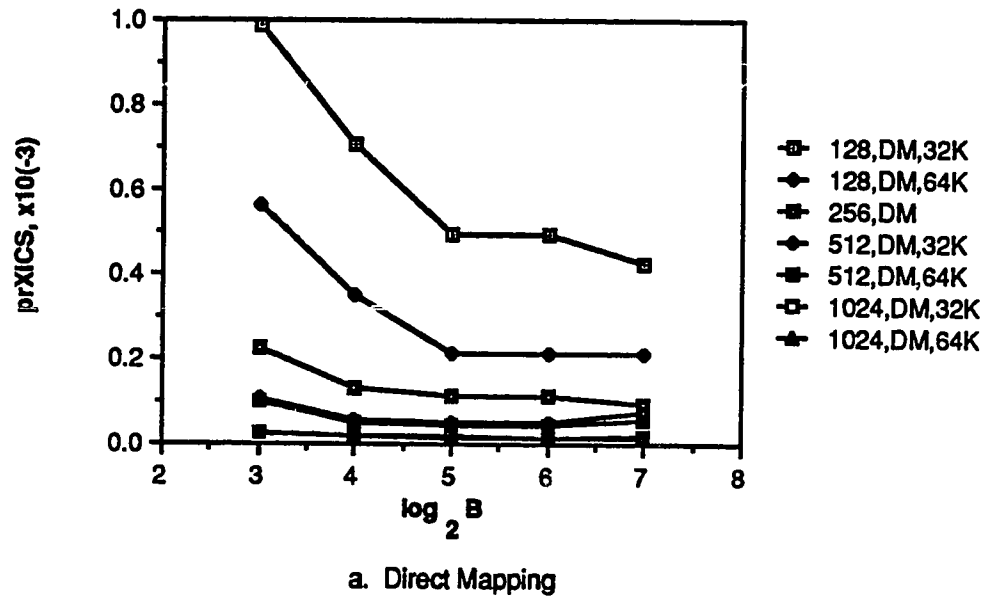
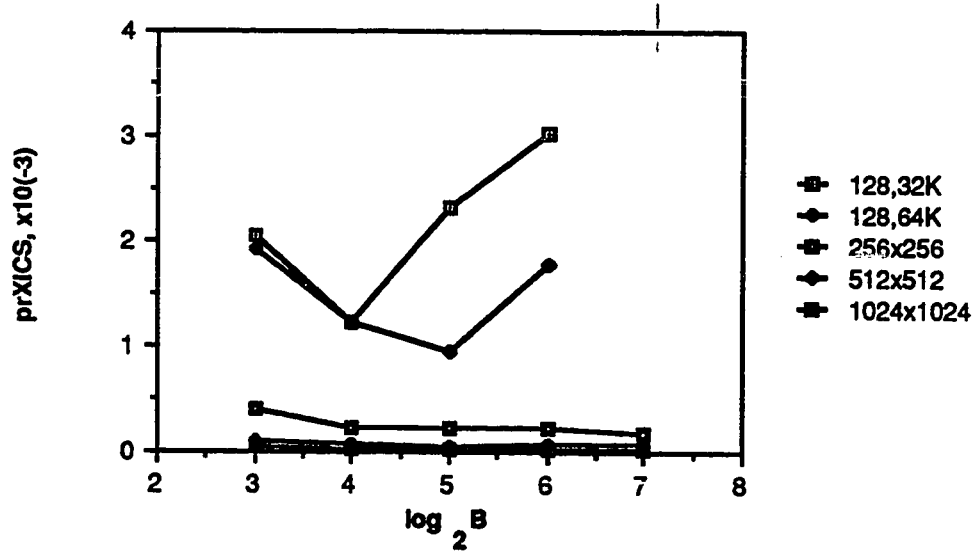


Figure 5.11 prXICS versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, All Grid Sizes, Four Processors.

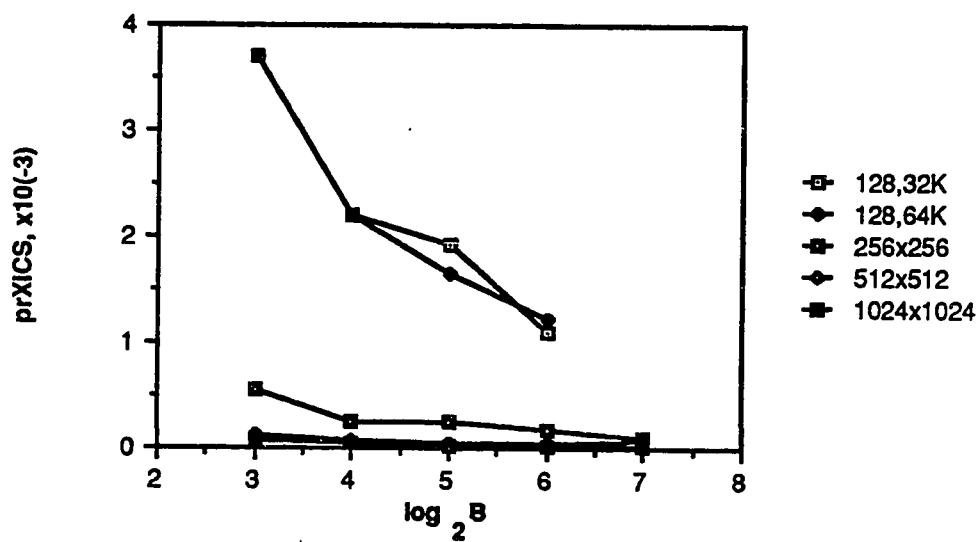
as the blocksize increases from 8 to 32 bytes and remains relatively constant for larger block sizes. The 128x128 point grid size and the set-associative mapping strategy is the only exception. This is because the grid size is so small that the locality is captured and the 2-way set-associative mapping function allows fewer block contentions as the blocksize increases. This in turn reduces the number of RO->EXs performed per iteration, the major cause of XICSs.

Figure 5.12 illustrates the prXICS versus the cache blocksize and all other features simulated for the eight processor system. For grid sizes larger than 128x128 points, this parameter decreases slightly as the cache blocksize increases. When using direct mapping, the 128x128 point grid prXICO decreases as the blocksize increases from 8 to 16 bytes and then increases for the larger grid sizes. This is because the increased contention for the larger block sizes causes more replacements and therefore more RO->EXs to be performed. The 2-way set-associative mapping function reduces this contention, resulting in a decrease in the prXICS as the blocksize increases from 8 to 128 bytes. Although not shown, the 128-byte blocksize (set-associative mapping) is large enough to increase block contention to the point where the prXICS increases substantially in value. Finally, for the 128x128 point grid size, observe the reduction in the prXICS as the cache size doubles.

Figure 5.13 shows the prXICS versus the cache blocksize and all other features considered for the sixteen processor system. The curves show a decrease in this parameter as the cache blocksize increases; however, the larger grid sizes do not demonstrate considerable discrepancies in value. The prXICS values for the smaller grid sizes and larger block sizes are not shown because their values are an order of

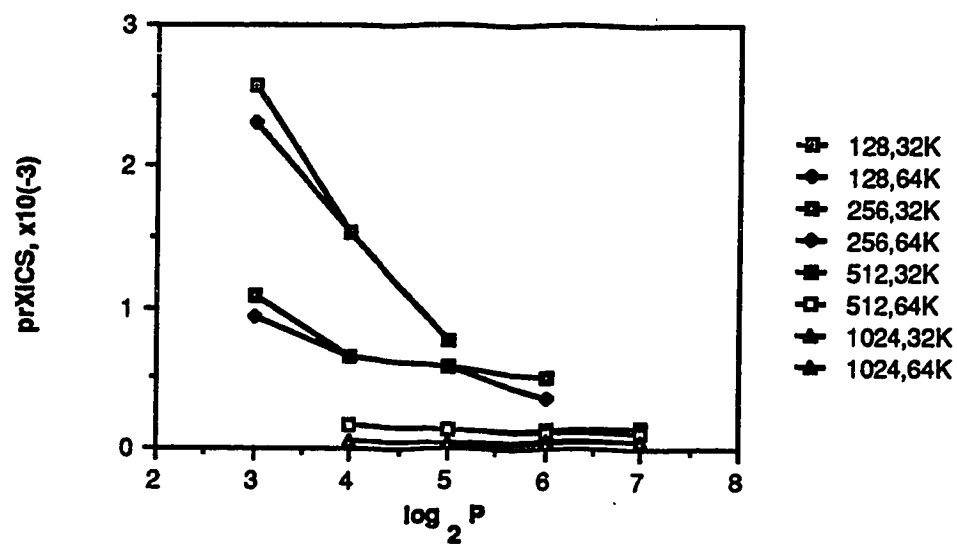


a. Direct Mapping

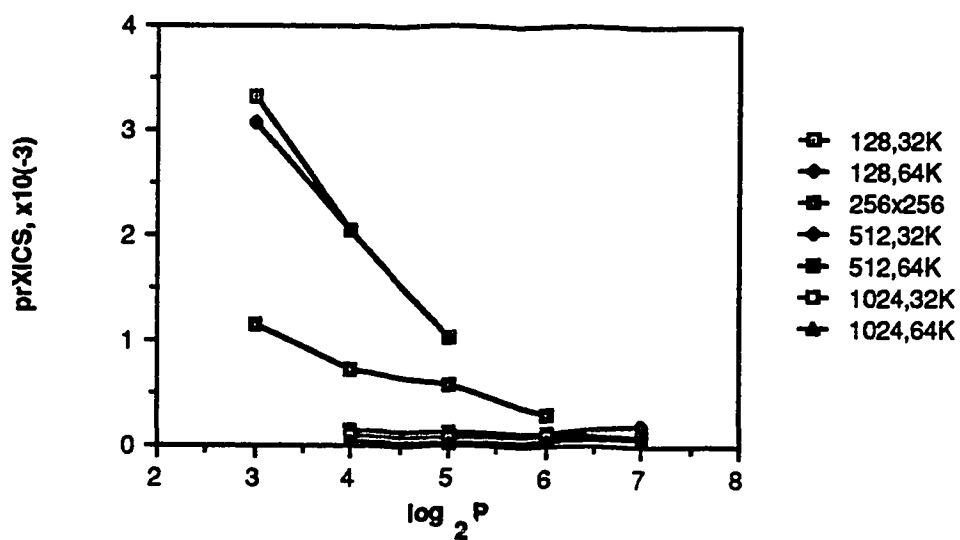


b. Set-Associative Mapping

Figure 5.12 prXICS versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, All Grid Sizes, Eight Processors.



a. Direct Mapping



b. Set-Associative Mapping

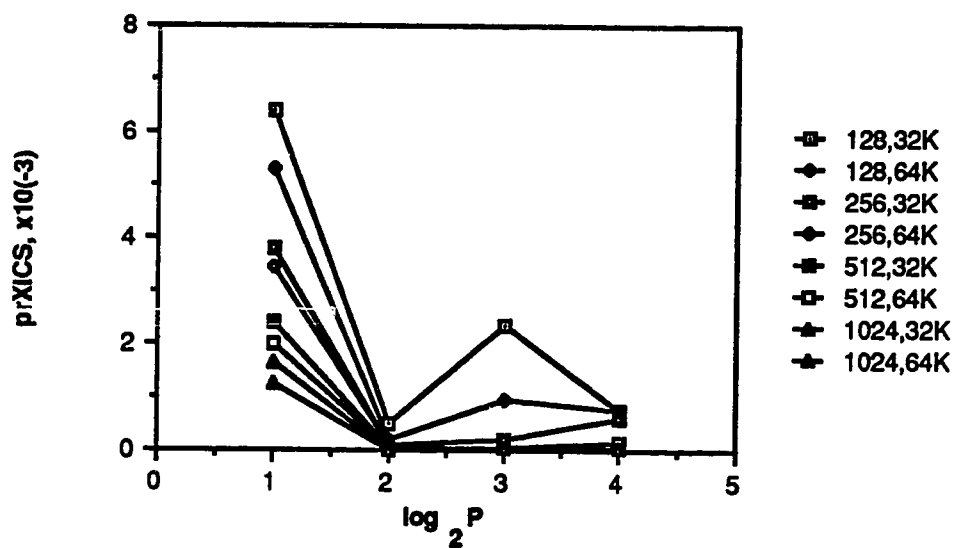
Figure 5.13 prXICS versus Cache Blocksize, Both Mapping Strategies and Cache Sizes, All Grid Sizes, Sixteen Processors.

magnitude larger than the values shown in the figure. This primarily results from increased contention and therefore an increase in the number of RO->EXs performed for these larger block sizes.

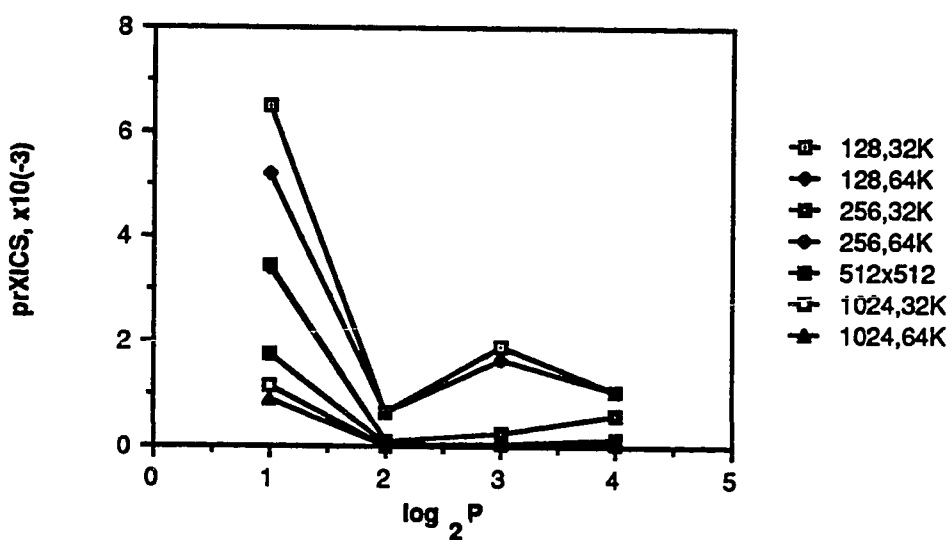
Figure 5.14 presents the prXICS versus the number of processors for both mapping strategies and cache sizes, all grid sizes and a 32-byte block size. All curves show a sharp decrease in value as the number of processors increase from two to four processors. The parameter then increases as the number of processors increase beyond four with the exception of the 128x128 point grid size. In this case, the prXICS decreases as the number of processors increase from eight to sixteen. This is because the grid size/processor tuple is such that fewer block contentions occur.

5.2.5. Prefetching Strategies

Figure 5.15 offers the MR versus the number of processors for all fetching strategies, small grid sizes, both mapping functions and cache sizes and a 32-byte block size. While all curves indicate an increase in this parameter as the number of processors increase, the tagged prefetching policy exhibits the best performance, followed by prefetch-on-miss and finally demand fetching. Also, as the number of processors increase, the differences between these fetching strategies increase. There are no differences between the values of the direct mapping and set-associative MRs for the 64x64 point grid size; however, direct mapping performs better than set-associative mapping for the 128x128 point grid size. This is because the set-associative mapping strategy does not effectively capture the reference locality for this grid size/algorithm combination.



a. Direct Mapping



b. Set-Associative Mapping

Figure 5.14 prXICS versus the Number of Processors, Both Mapping Strategies and Cache Sizes, All Grid Sizes, 32-byte Blocksize.

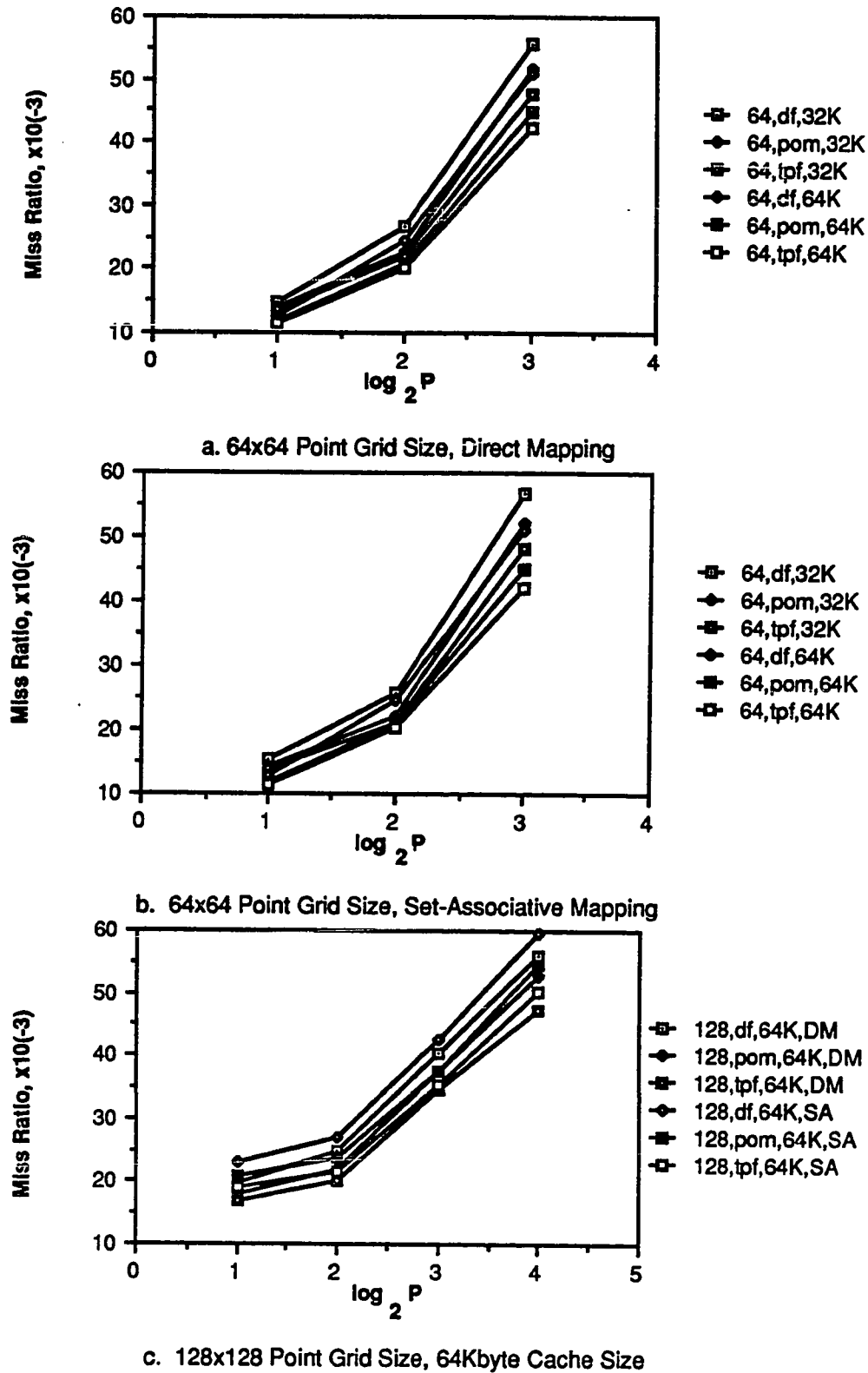
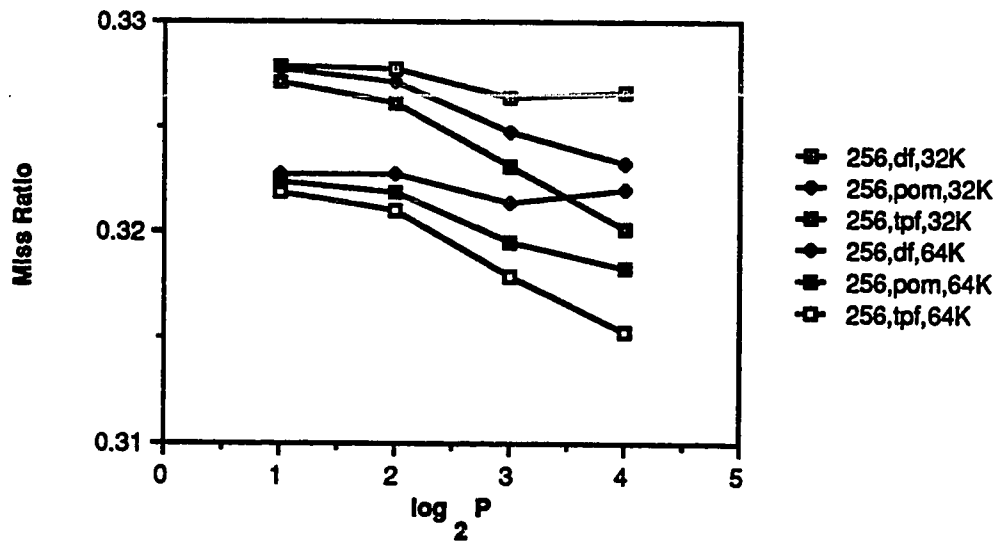


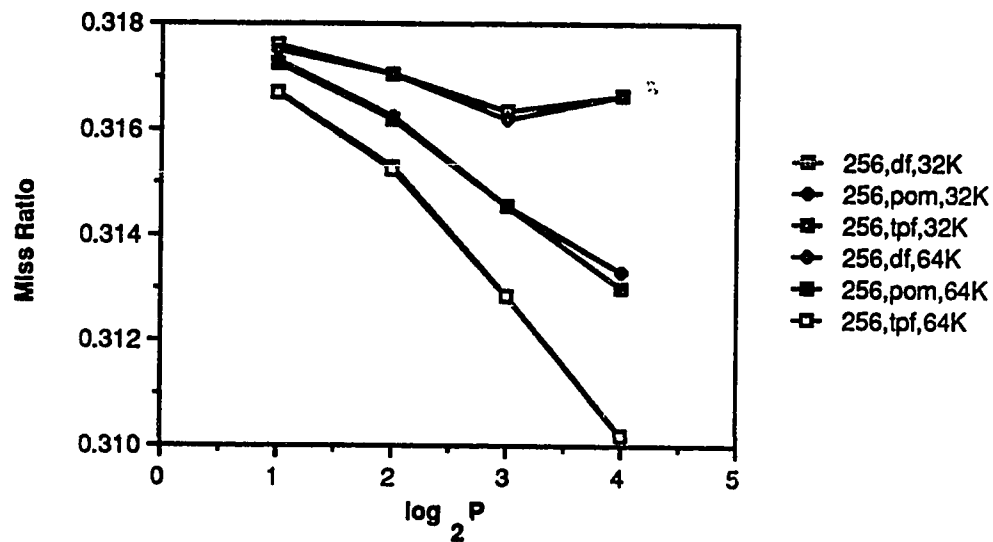
Figure 5.15 Miss Ratio versus the Number of Processors, All Fetching Strategies, Small Grid Sizes, Both Mapping Functions and Cache Sizes, 32-byte Blocksize.

Figure 5.16 presents the MR versus the number of processors for all fetching strategies, both mapping functions and cache sizes, the 256x256 point grid size and a 32-byte blocksize. All curves show tagged prefetching exhibiting the better performance, followed by prefetch-on-miss and demand fetching, respectively. The set-associative mapping function performs better than direct mapping and the 64Kbyte cache size exhibits the better performance for direct mapping. There are negligible differences between the MRs of the two cache sizes for the set-associative mapping strategy. All curves also show slight decreases in value as the number of processors increase. This is because the cache does not capture an acceptable portion of the reference locality for PCG. As a result, for each iteration, every first reference to a block causes a miss. For a given grid size, as the number of processors increase, the number of blocks referenced per algorithm iteration decreases, resulting in a decrease in the MR. All other larger grid sizes possess MR characteristics similar to this 256x256 point grid size.

Figure 5.17 illustrates the IR versus the number of processors for all fetching strategies, both cache sizes, direct mapping and a 32-byte blocksize. This parameter increases as the number of processors increase. For all grid sizes, both tagged prefetching and prefetch-on-miss possess a higher IR than demand fetching. Since the MRs for these prefetching strategies are smaller than the demand fetching MR, more invalidateable blocks remain in the cache, resulting in the higher IRs. Note that there are very small discrepancies between the two prefetching IRs. Also observe that as the PDE grid size increases the IR decreases. Figure 5.18 shows the IR versus the number of processors for the set-associative mapping strategy. The behavior of these

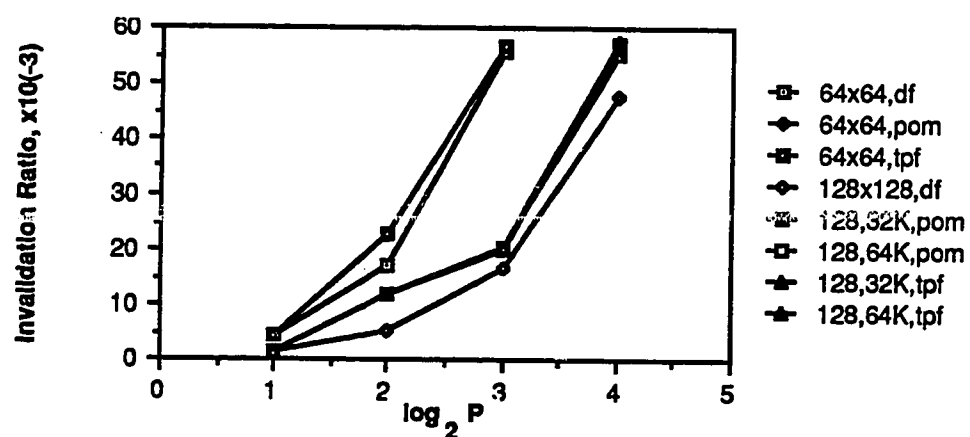


a. Direct Mapping

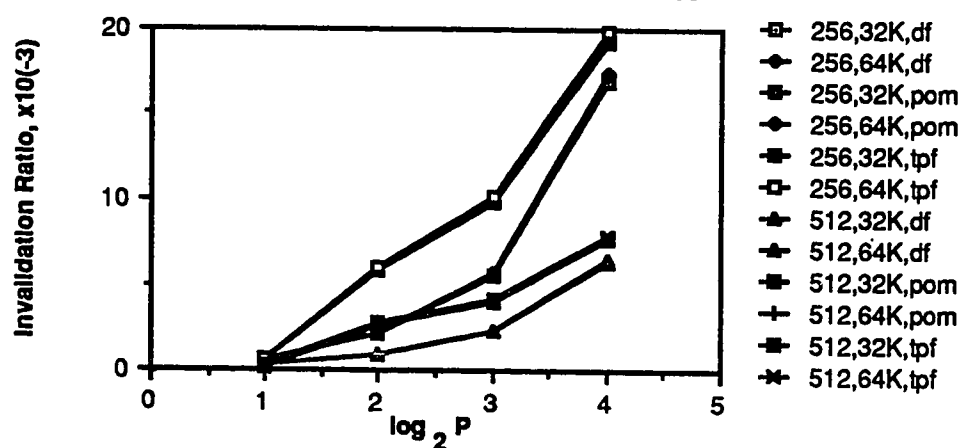


b. Set-Associative Mapping

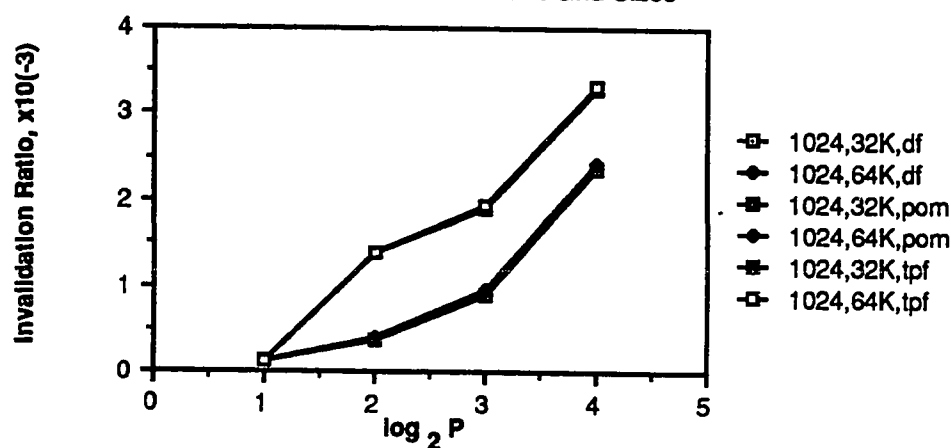
Figure 5.16 Miss Ratio versus the Number of Processors, All Fetching Strategies, Both Mapping Functions and Cache Sizes, 256x256 Point Grid Size, 32-byte Blocksize.



a. 64x64 and 128x128 Point Grid Sizes

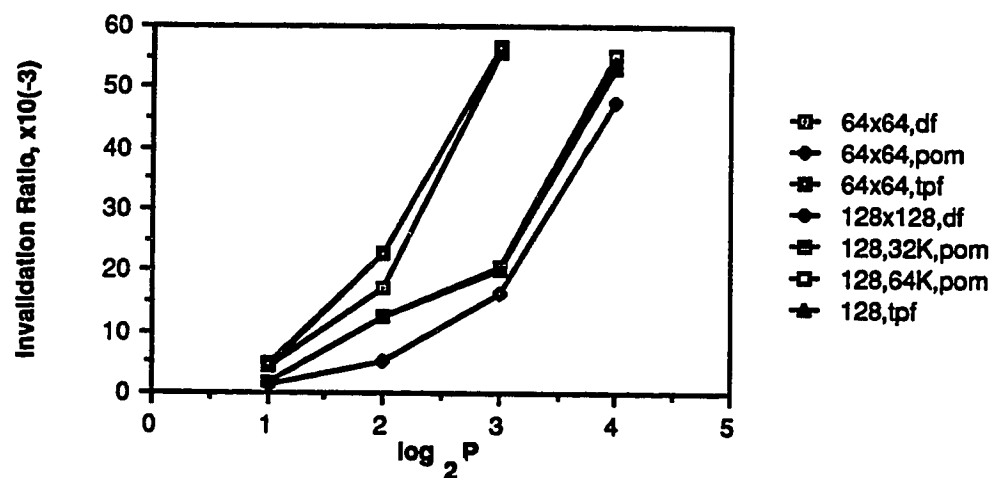


b. 256x256 and 512x512 Point Grid Sizes

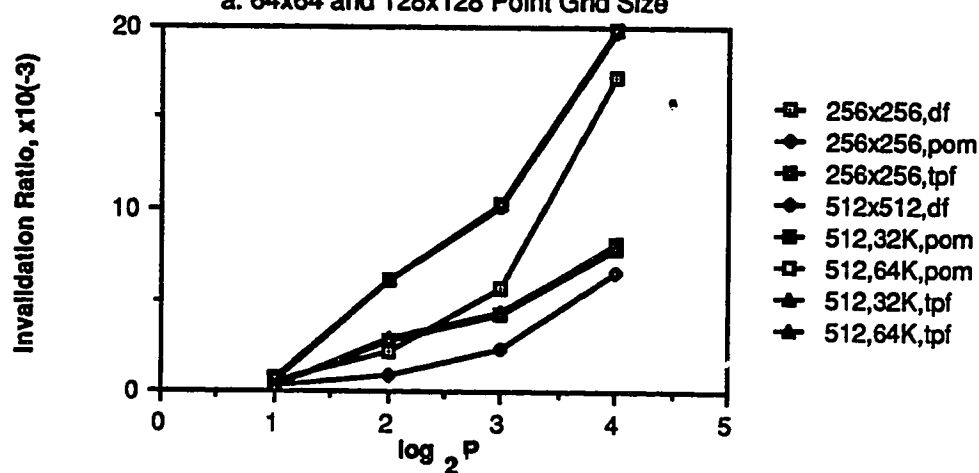


c. 1024x1024 Point Grid Size

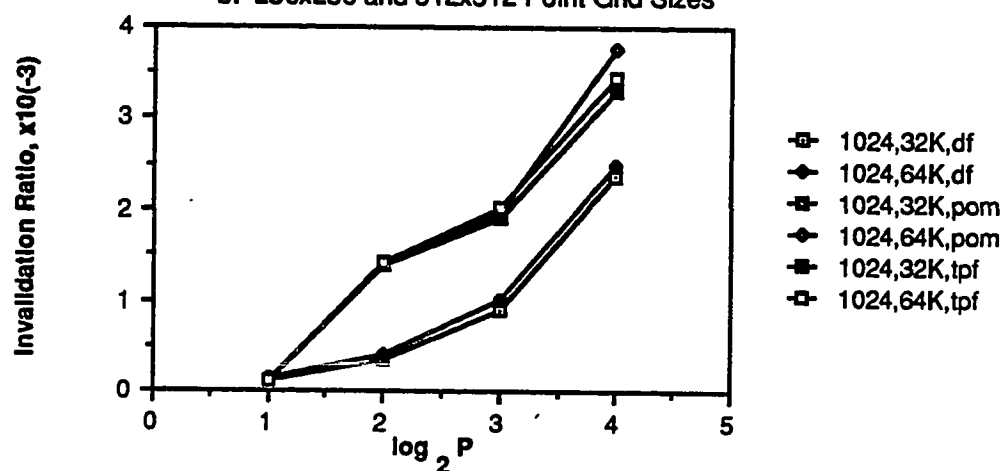
Figure 5.17 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, Both Mapping Functions and Cache Sizes, Direct Mapping, 32-byte Blocksize.



a. 64x64 and 128x128 Point Grid Size



b. 256x256 and 512x512 Point Grid Sizes



c. 1024x1024 Point Grid Size

Figure 5.18 Invalidation Ratio versus the Number of Processors, All Fetching Strategies, Both Mapping Functions and Cache Sizes, Set-Associative Mapping, 32-byte Blocksize.

curves are very similar to the direct mapping IRs.

Figure 5.19 displays the prXICO versus the number of processors for all fetching strategies, both cache sizes, direct mapping and a 32-byte blocksize. This parameter increases as the number of processors increase with tagged prefetching and prefetch-on-miss having approximately equal but higher probabilities when compared to demand fetching. The lower MRs for these prefetching strategies also increase the probability of casting out a cached block. Also note that as the grid size increases the prXICO decreases. Figure 5.20 presents the prXICO versus the number of processors for the 2-way set-associative mapping strategy. Like the IR, this parameter also possesses characteristics similar to its direct mapping associate.

Figure 5.21 presents the prXICS versus the number of processors for all fetching strategies, both mapping functions and cache sizes, smaller grid sizes and a 32-byte blocksize. For the 64x64 point grid size (part a) the 64Kbyte prXICSs are very small. On the other hand, the 32Kbyte prXICSs increase as the number of processors increase. This is because more blocks are replaced in the smaller cache size resulting in a greater number of RO->EX operations performed for a given grid size. The tagged prefetching values are slightly higher than the prefetch-on-miss values and the set-associative prXICS exhibits better performance than the direct mapping values. This also occurs as a result of the increase in the frequency of replaced blocks for tagged prefetching and direct mapping. When prefetch-on-miss is used with a 32Kbyte cache size the prXICS for the 128x128 point grid size increases by an order of magnitude. The grid size/cache size/algorithm combination are the three major factors contributing to this substantial increase. Also observe the set-associative

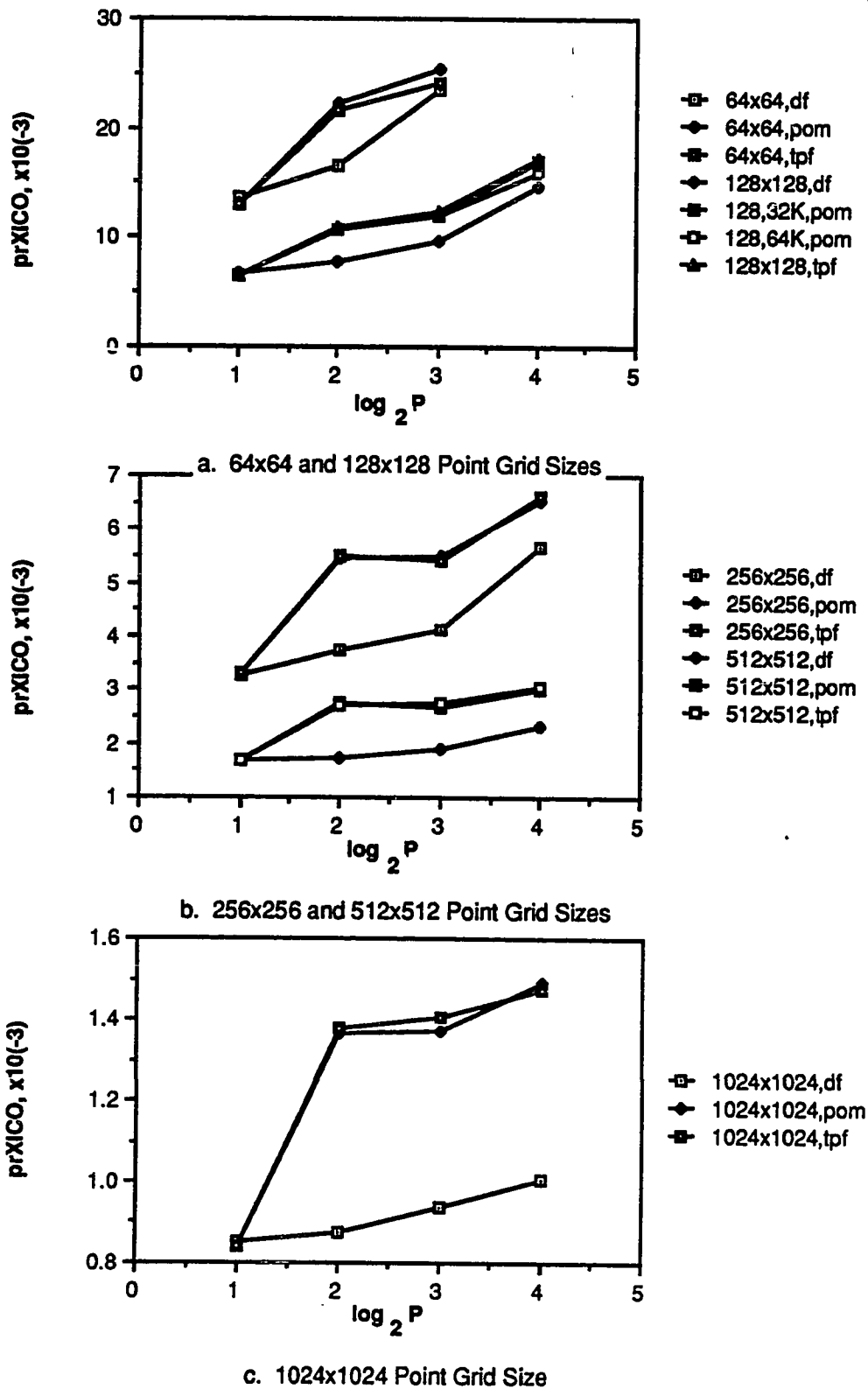
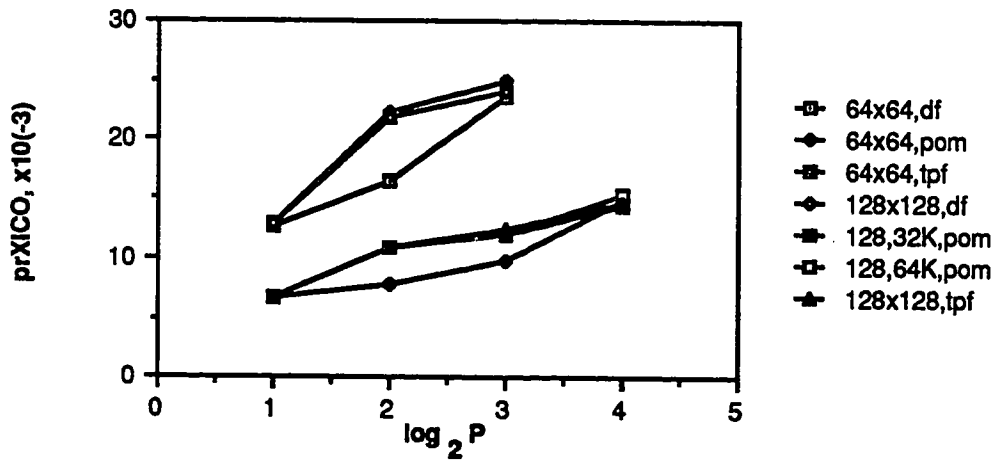
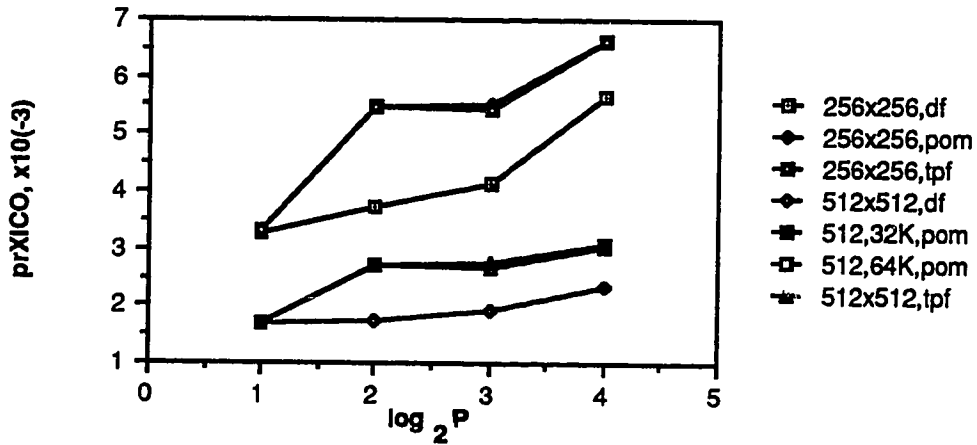


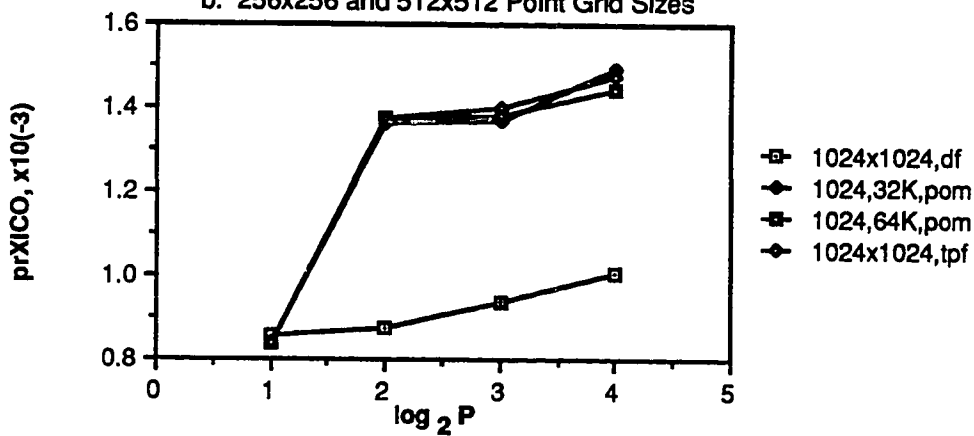
Figure 5.19 prXICO versus the Number of Processors, All Fetching Strategies, Both Mapping Functions and Cache Sizes, Direct Mapping, 32-byte Blocksize.



a. 64x64 and 128x128 Point Grid Sizes

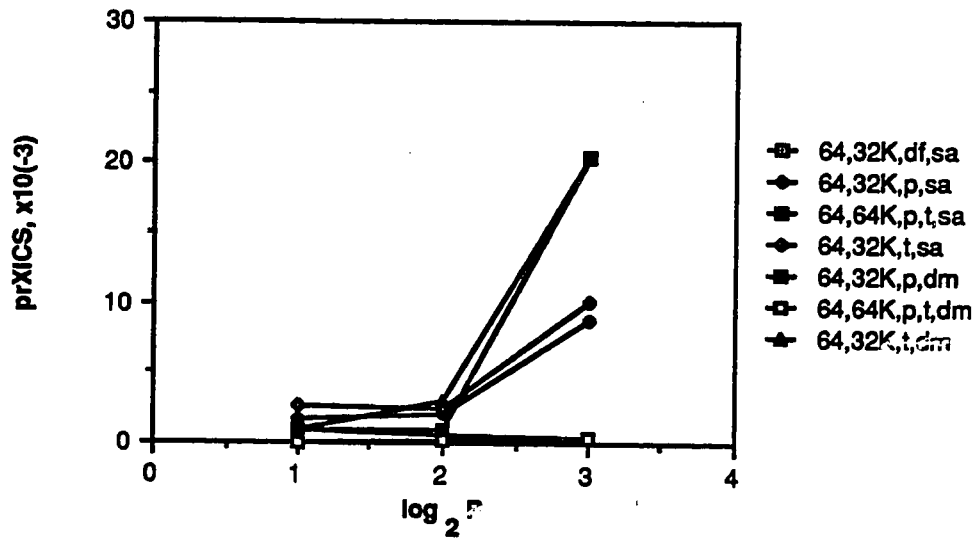


b. 256x256 and 512x512 Point Grid Sizes

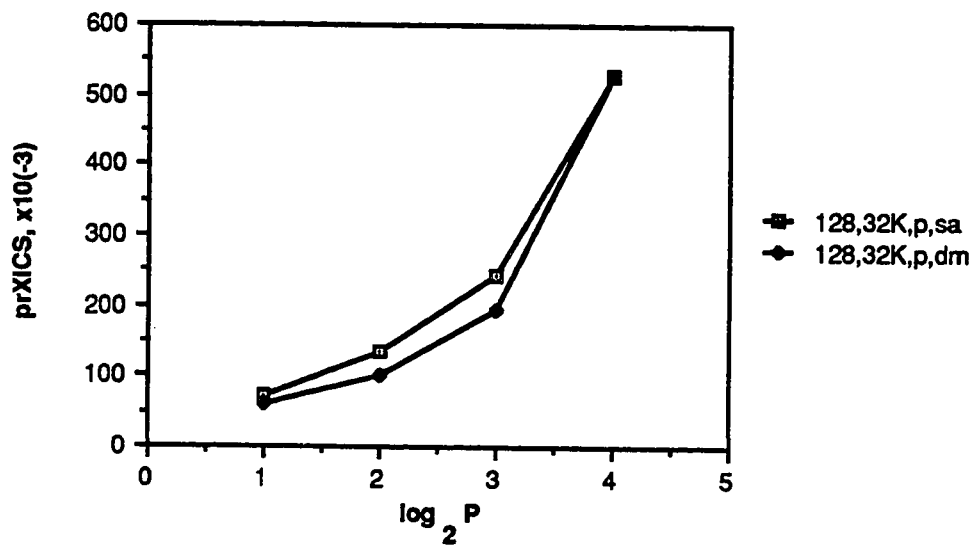


c. 1024x1024 Point Grid Size

Figure 5.20 prXICO versus the Number of Processors, All Fetching Strategies, Both Mapping Functions and Cache Sizes, Set-Associative Mapping, 32-byte Blocksize.



a. 64x64 Point Grid Size



b. 128x128 Point Grid Size, 32Kbyte Cache Size, Prefetch-on-miss

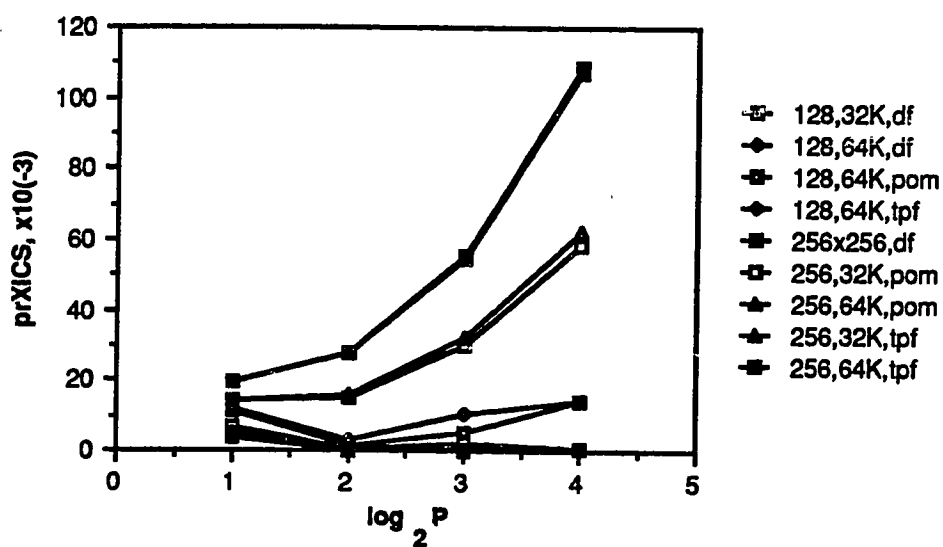
Figure 5.21 prXICS versus the Number of Processors, All Fetching Strategies, Both Mapping Functions and Cache Sizes, Smaller Grid Sizes, 32-byte Blocksize.

prXICS is larger than its demand fetching counterpart. This counterintuitive behavior is explained in Section 5.2.4.

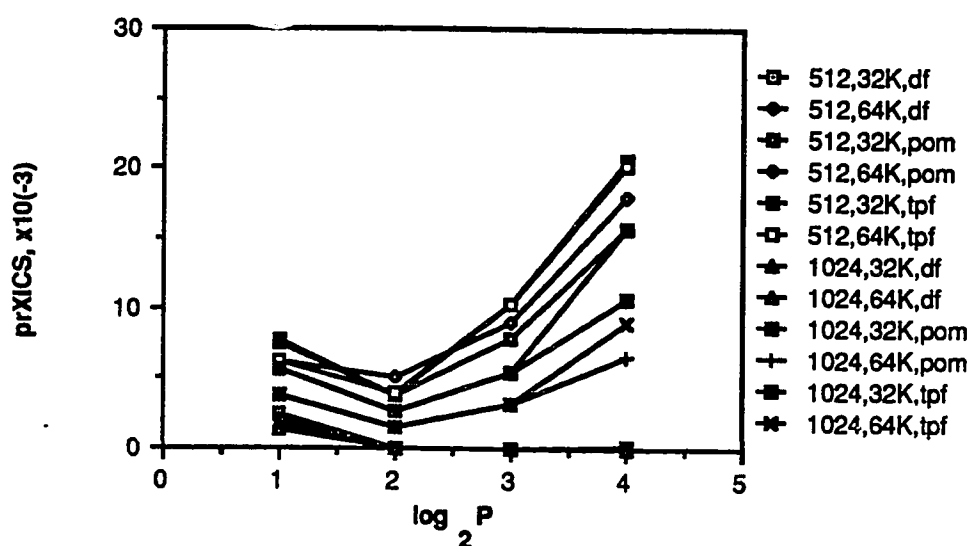
Figure 5.22 illustrates the prXICS versus the number of processors for all fetching strategies, both cache sizes, larger grid sizes, direct mapping and a 32-byte block-size. This parameter increases as the number of processors increase from four to sixteen. For grid sizes larger than 128x128 points this parameter decreases as the number of processors increase from two to four. This behavior is explained in Section 5.2.4. Again, demand fetching has the best performance, followed by prefetch-on-miss and finally tagged prefetching. Also, for reasons discussed above, the 64Kbyte cache has a higher prXICS than the 32Kbyte cache size. Furthermore, as the PDE grid size increases, the prXICS decreases.

Figure 5.23 displays the prXICS versus the number of processors and other features identical to the previous figure with the exception that the set-associative mapping function is used. Excluding the prefetching 256x256 point prXICS (64Kbyte cache only) the behavior of this parameter for this mapping function is similar to that of the direct mapping prXICS; however, the set-associative prXICS is generally less than its direct mapping associate. For the 64Kbyte cache size, the prefetching, set-associative 256x256 point prXICS is higher than the direct mapping value. This is because the grid size/cache size/cache placement combination is conducive for this behavior.

Figures 5.24 and 5.25 present the LR and PR versus the number of processors for both prefetching strategies and cache sizes, a 32-byte blocksize and direct and

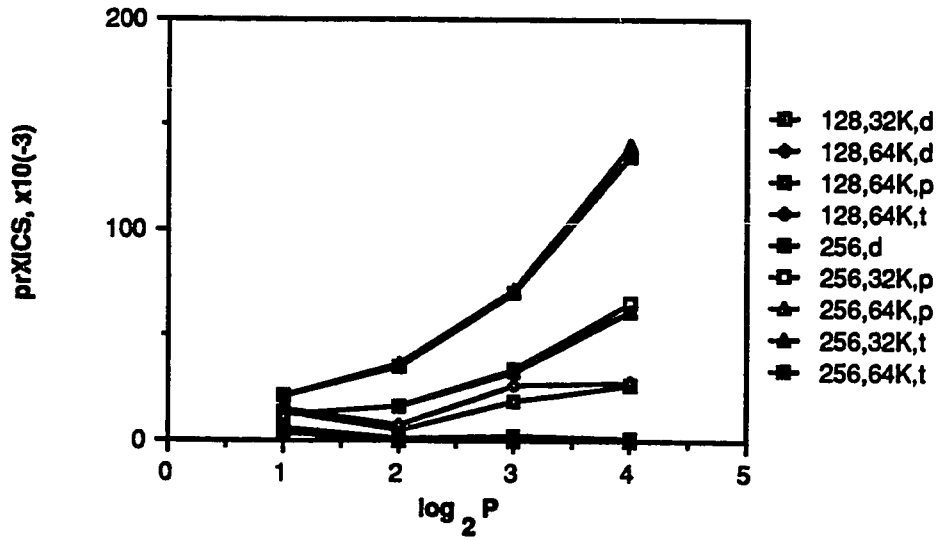


b. 128x128 and 256x256 Point Grid Sizes

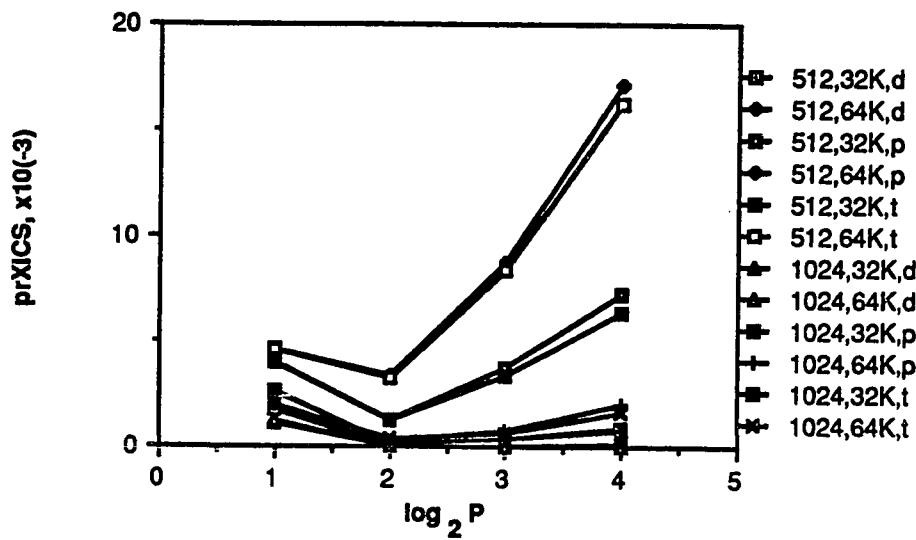


b. 512x512 and 1024x1024 Point Grid Sizes

Figure 5.22 prXICS versus the Number of Processors, All Fetching Strategies, Both Cache Sizes, Larger Grid Sizes, Direct Mapping, 32-byte Blocksize.

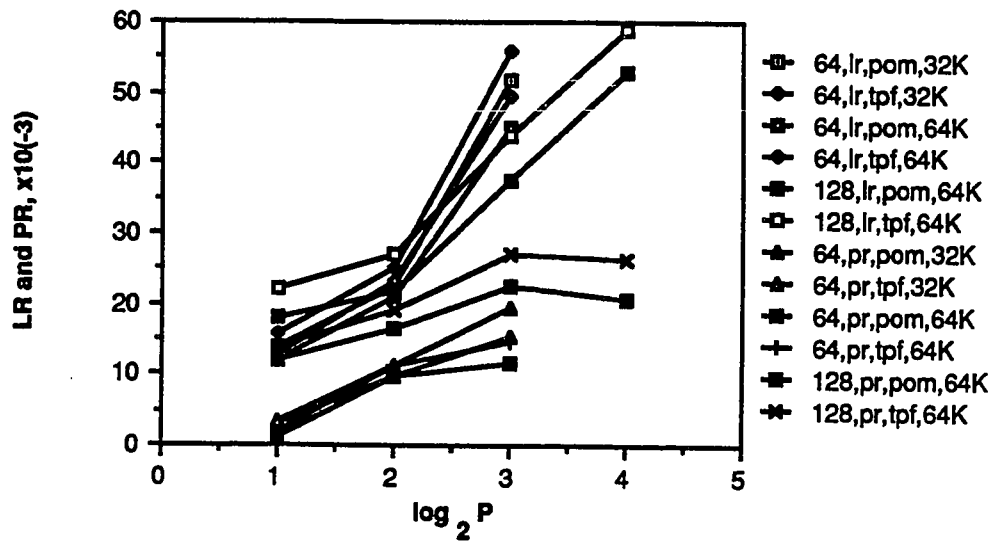


a. 128x128 and 256x256 Point Grid Sizes

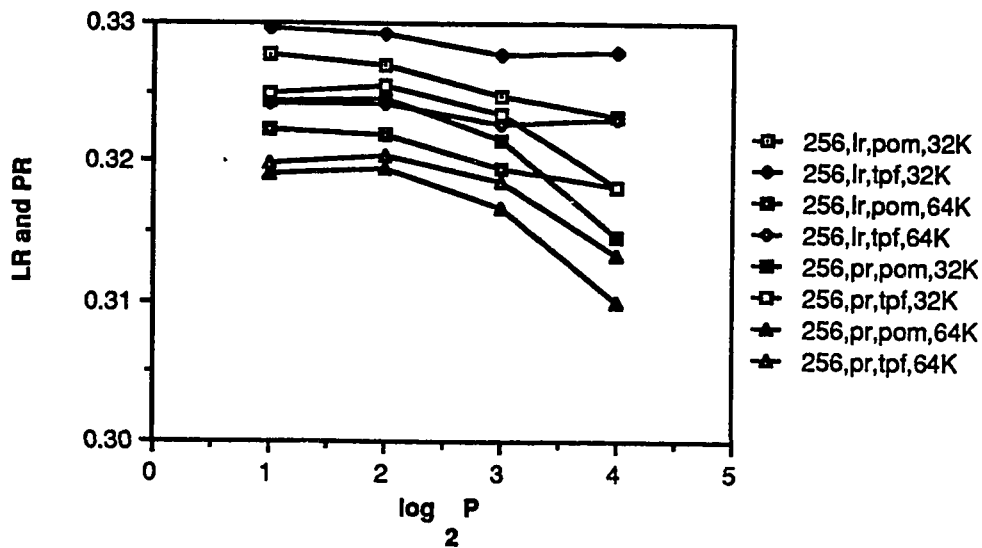


b. 512x512 and 1024x1024 Point Grid Sizes

Figure 5.23 prXICS versus the Number of Processors, All Fetching Strategies, Both Cache Sizes, Larger Grid Sizes, Set-Associative Mapping, 32-byte Blocksize.

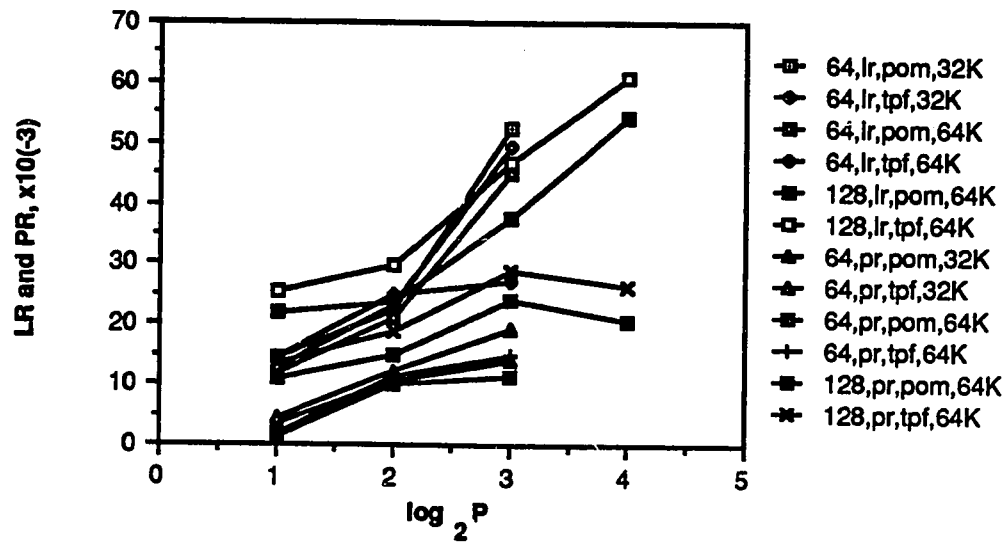


a. 64x64 and 128x128 Point Grid Sizes

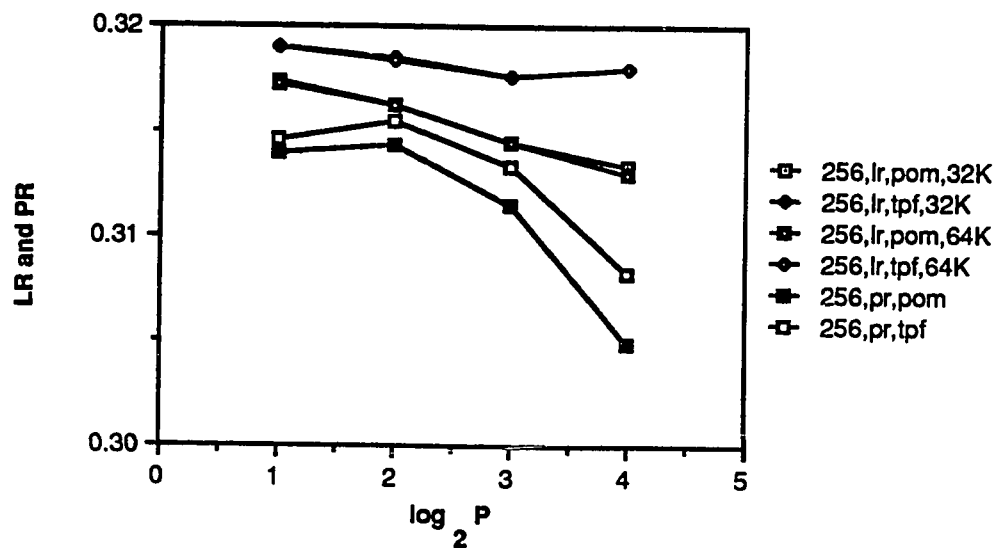


b. 256x256 Point Grid Size

Figure 5.24 Lookup Ratio and Prefetch Ratio versus the Number of Processors, Both Prefetching Strategies and Cache Sizes, Direct Mapping, 32-byte Blocksize.



a. 64x64 and 128x128 Point Grid Sizes



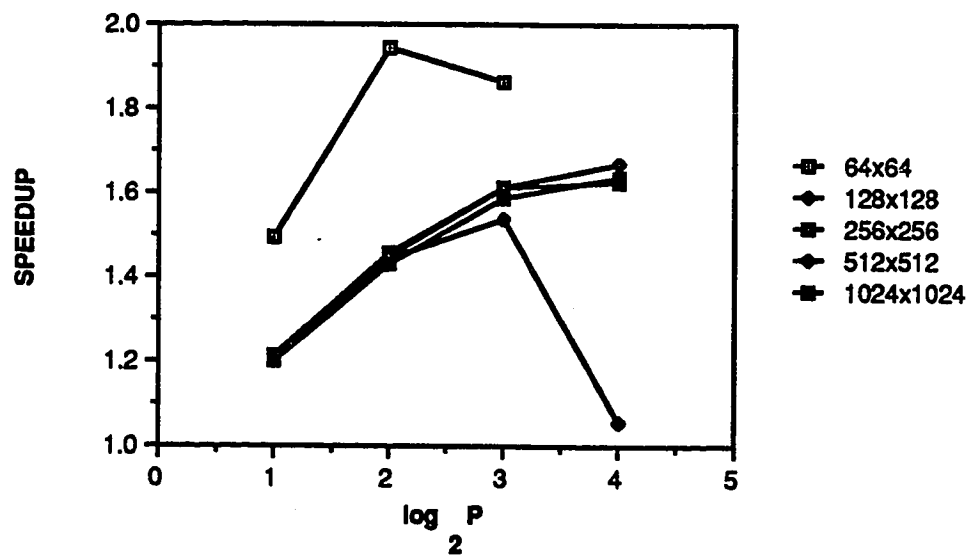
b. 256x256 Point Grid Size

Figure 5.25 Lookup Ratio and Prefetch Ratio versus the Number of Processors, Both Prefetching Strategies and Cache Sizes, Set-Associative Mapping, 32-byte Blocksize.

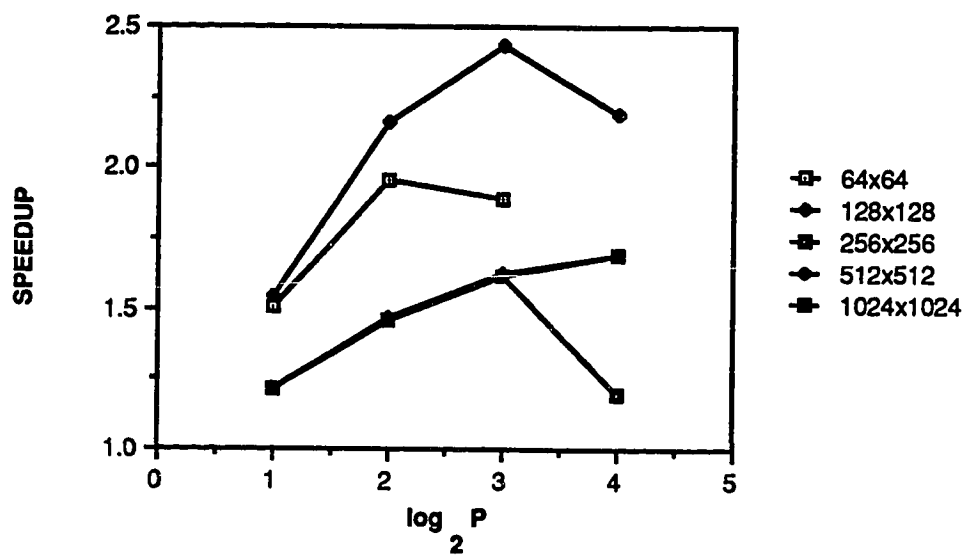
set-associative mapping, respectively. The characteristics of these graphs are identical to the demand fetching graphs presented and discussed in the previous section. The prefetching behavior parallels all of the prefetching behavior discussed thus far for this algorithm. That is, prefetch-on-miss exhibits better performance than tagged prefetching. Also observe the intuitive behavior of the higher LRs. Finally, for reasons discussed earlier, set-associative mapping has better performance than direct mapping for the larger grid sizes; however, the converse is true for the smaller grid sizes shown.

5.2.6. Multiprocessor Speedup

Figure 5.26 presents the speedup versus the number of processors for direct mapping, both cache sizes, all grid sizes, a 32-byte blocksize and the single bus interconnection network. While the speedup increases as the number of processors increase for large grid sizes, the actual values obtained are far from linear. Furthermore, the 128x128 and 256x256 point grid sizes reach a peak value when eight processors are used. The 64x64 point grid sizes reaches a peak value for four processors. The major cause of the low speedups is the single bus interconnection network. As the number of processors increase, the upper bound on the interconnection waiting time increases linearly. As the cache size doubles the speedup increases for the 128x128 point grid size. This is a result of the decrease in the MR for this grid size as the cache doubles. Figure 5.27 illustrates this same speedup versus the number of processors for the set-associative mapping strategy. These curves are similar to the direct mapping speedups.

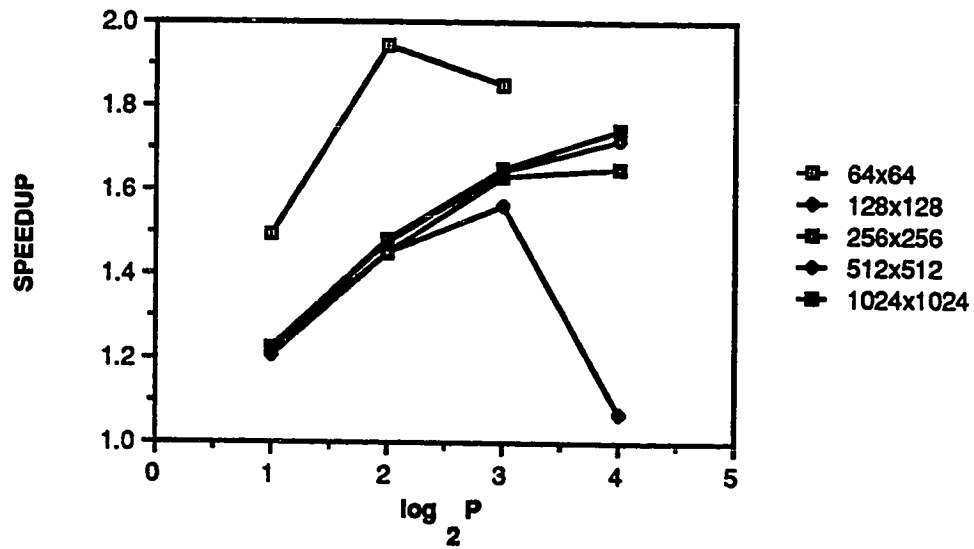


a. 32Kbyte Cache Size

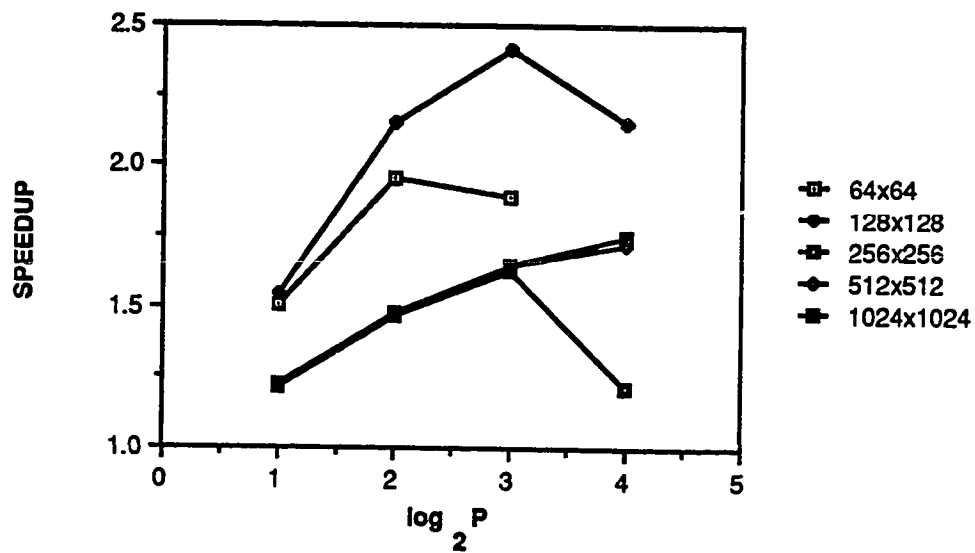


b. 64Kbyte Cache Size

Figure 5.26 Speedup versus the Number of Processors, Direct Mapping, Both Cache Sizes, All Grid Sizes, 32-byte Blocksize, Single Bus Interconnection.



a. 32Kbyte Cache Size

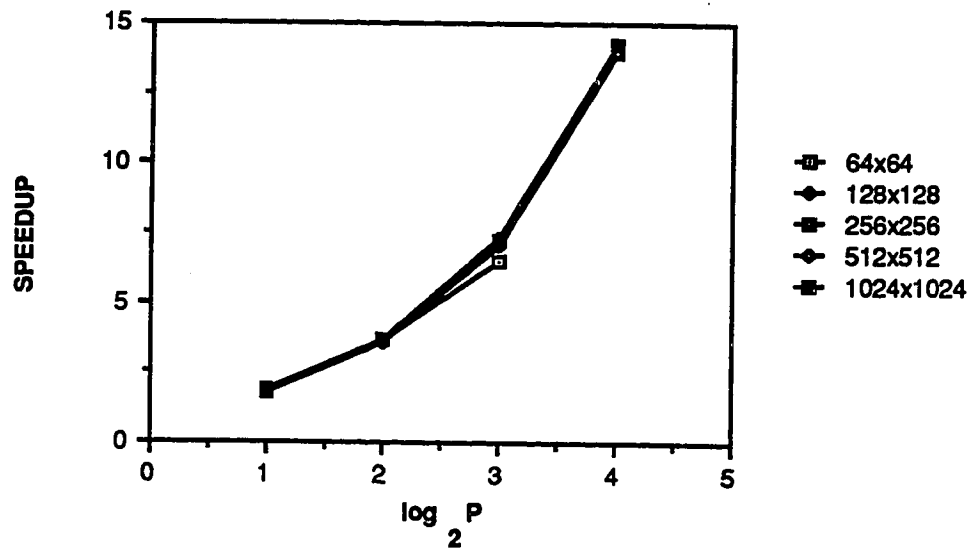


b. 64Kbyte Cache Size

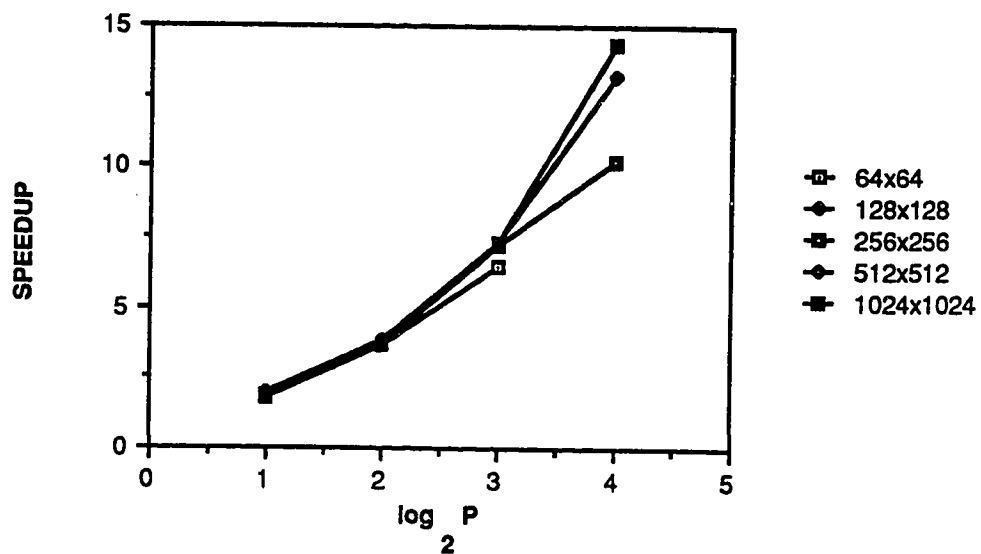
Figure 5.27 Speedup versus the Number of Processors, Set-Associative Mapping, Both Cache Sizes, All Grid Sizes, 32-byte Blocksize, Single Bus Interconnection.

Figure 5.28 displays the speedup versus the number of processors for direct mapping, both cache sizes, all grid sizes a 32-byte blocksize and the full crossbar interconnection network. The speedups shown in this figure increase as the number of processors increase for all grid sizes. Assuming a lower bound interconnection waiting time of 0, the full crossbar performance is considerably better than the single bus network. Moreover, the set-associative mapping speedups are somewhat higher than the direct mapping values as shown in Figure 5.29. Both mapping strategies show the lower speedups obtained by the 128x128 and 256x256 point grid sizes for the 64Kbyte cache size and the 16 processor system. The 128x128 point grid size also exhibits a lower speedup value for the 32Kbyte cache size and the 16 processor system. With these exceptions there are no appreciable differences between the speedup values as the cache size increases.

Figure 5.30 illustrates the speedup versus the number of processors for all fetching strategies, direct mapping, both cache sizes, a 32-byte blocksize and a single bus interconnection network. The 64x64 and 256x256 point grid sizes are observed. Both of these grid sizes show demand fetching exhibiting better performance than the identical speedups for prefetch-on-miss and tagged prefetching. The additional time needed to prefetch blocks, lookup blocks in cache directories and service the additional XIs required by prefetching do not offset the decrease in the MR for these strategies. This directly effects the effective memory access time and therefore the waiting time for the single bus systems. This results in the lower speedups obtained by these prefetching strategies. The increase in the cache size does cause slight increases in the speedups for all fetching strategies and grid sizes considered here.

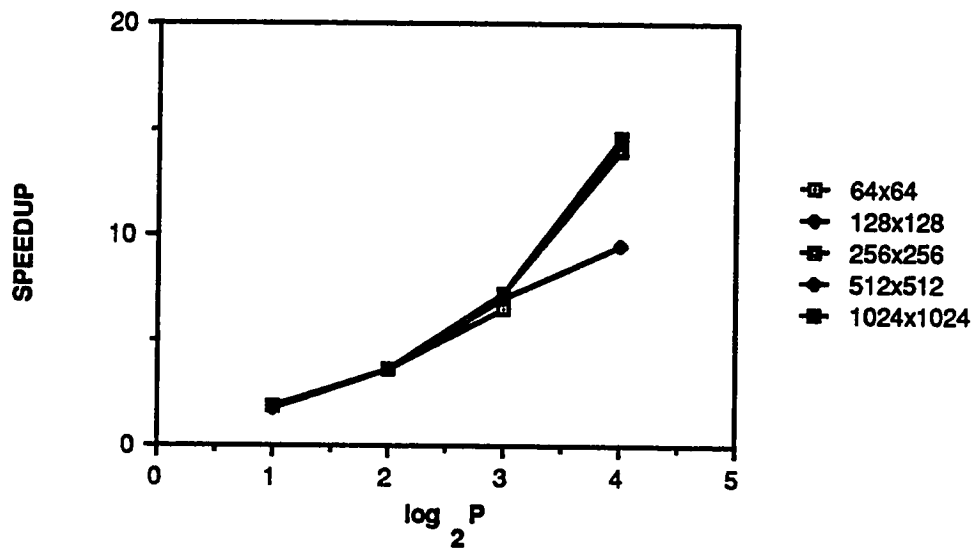


a. 32Kbyte Cache Size

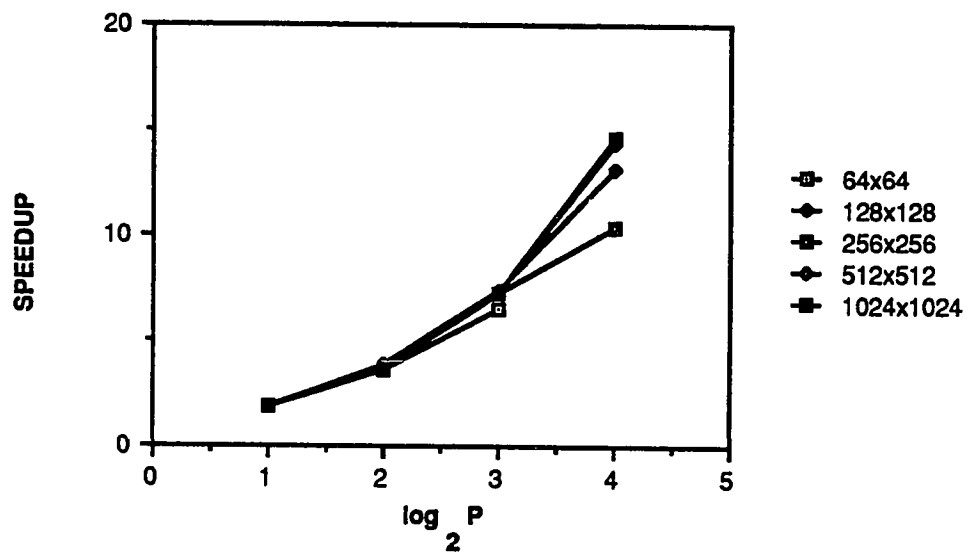


b. 64Kbyte Cache Size

Figure 5.28 Speedup versus the Number of Processors, Direct Mapping, Both Cache Sizes, All Grid Sizes, 32-byte Blocksize, Full Crossbar Interconnection.

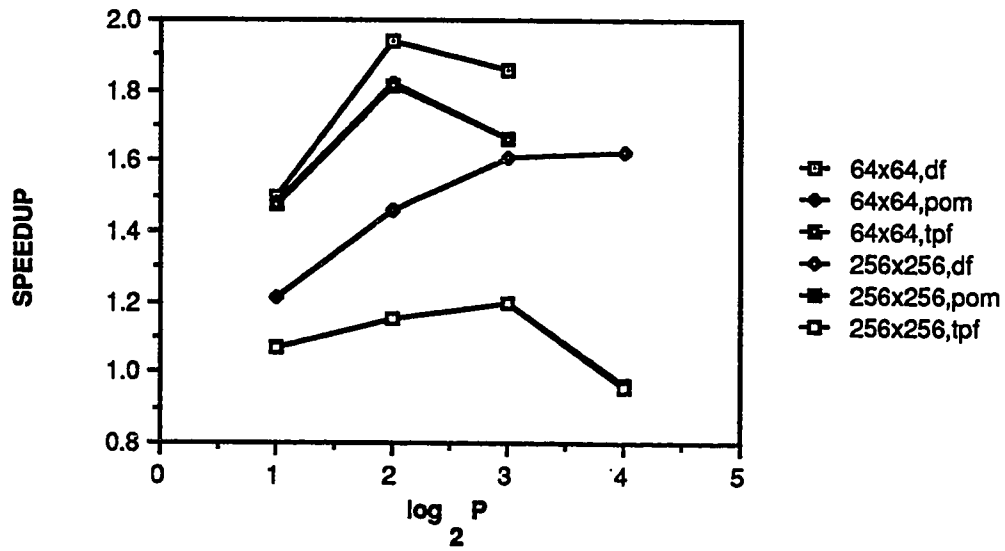


a. 32Kbyte Cache Size

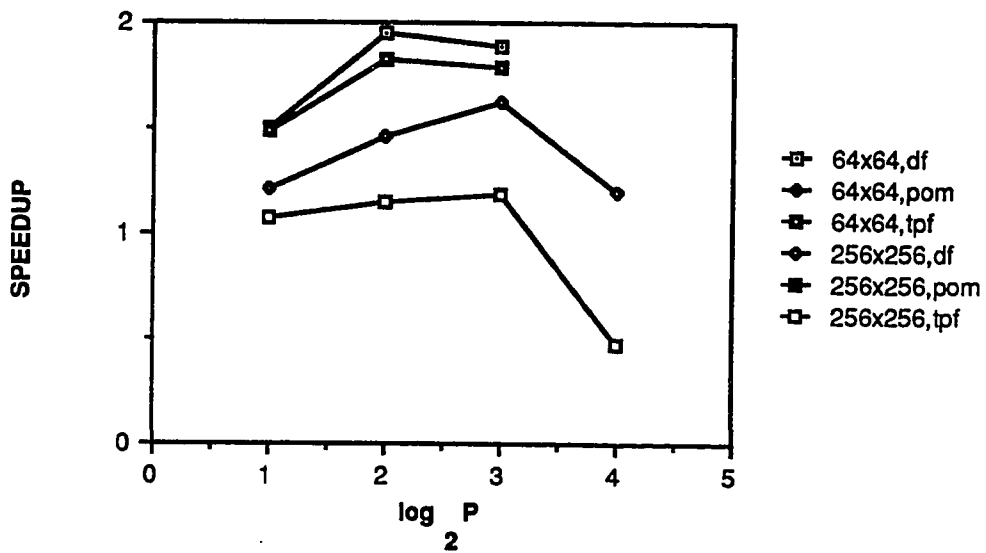


b. 64Kbyte Cache Size

Figure 5.29 Speedup versus the Number of Processors, Set-Associative Mapping, Both Cache Sizes, All Grid Sizes, 32-byte Blocksize, Full Crossbar Interconnection.



a. 32Kbyte Cache Size



b. 64Kbyte Cache Size

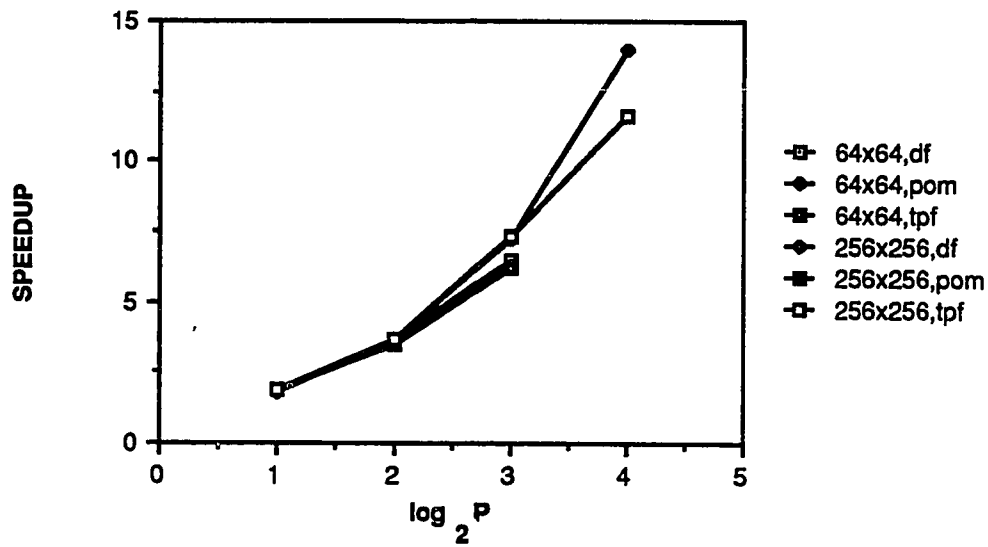
Figure 5.30 Speedup versus the Number of Processors, All Fetching Strategies, Direct Mapping, Both Cache Sizes, 32-byte Blocksize, Single Bus Interconnection.

Figure 5.31 presents the speedup versus the number of processors for all fetching strategies, the set-associative mapping function, both cache sizes, a 32-byte blocksize and the full crossbar system. The two, four and eight processor systems show identical speedup values for all fetching strategies. Demand fetching exhibits better performance than both prefetching strategies for sixteen processors and the 256x256 point grid size. While the cache size does not effect the speedup of the 64x64 point grid size, the 256x256 point grid size speedup decreases as the cache size increases for all fetching strategies. This behavior is also shown in Figure 5.29.

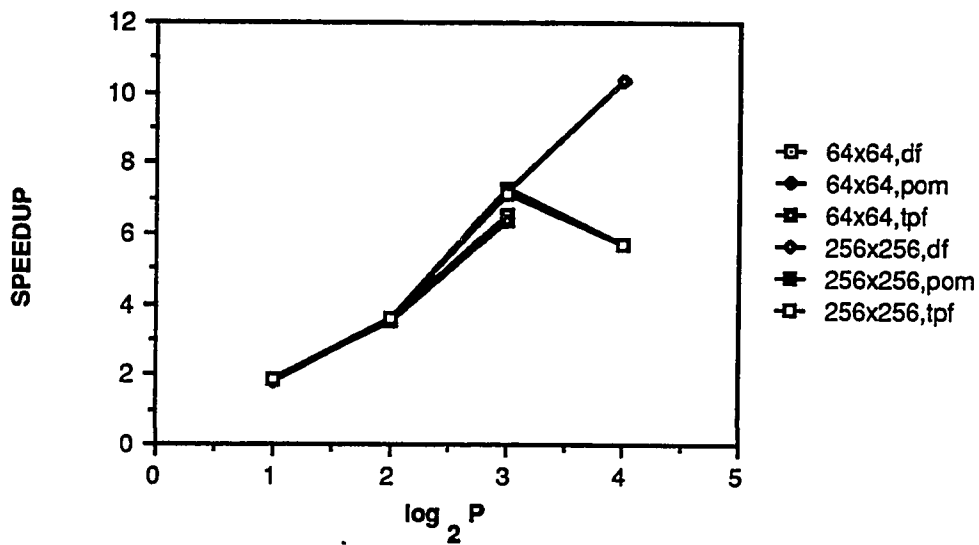
Figure 5.32 displays the iteration time degradation versus the number of processors for both cache sizes, mapping functions and interconnection networks considered, a 32-byte blocksize and the 64x64 and 256x256 point PDE grid sizes. This degradation increases as the number of processors increase. Also, the degradation is greater for the single bus interconnection network. This is because in addition to the effective memory access time degradation, the enabled coherence waiting time also increases the amount of time needed to execute each algorithm iteration. This waiting time degradation is not a factor when using the lower bound waiting time of 0 for the full crossbar interconnection network. In general, these curves show the time needed to execute one iteration of PCG with enabled coherence to be between 1.05 and 1.65 times the disabled coherence iteration time.

5.3. PCG Conclusions

For larger grid sizes, the MR is generally independent of the cache bocksize and the number of of processors. Its value ranges from 0.30 to 0.38 dependent upon the

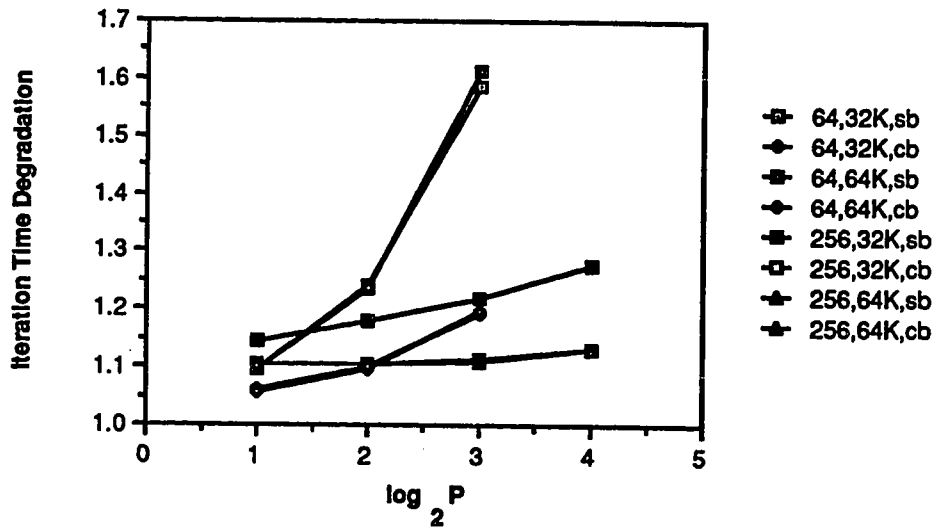


a. 32Kbyte Cache Size

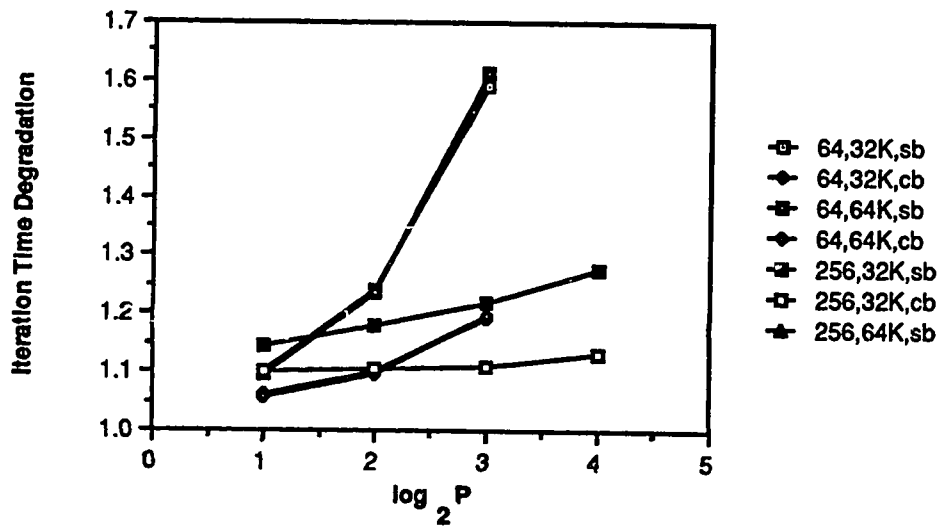


b. 64Kbyte Cache Size

Figure 5.31 Speedup versus the Number of Processors, All Fetching Strategies, Set-Associative Mapping, Both Cache Sizes, Full Crossbar Interconnection.



a. Direct Mapping



b. Set-Associative Mapping

Figure 5.32 Iteration Time Degradation versus the Number of Processors, Both Cache Sizes, Mapping Functions and Interconnection Networks, 32-byte Cache Blocksize.

PDE grid size. The set-associative mapping strategy exhibits the better performance for the larger grid sizes. For smaller grid sizes the MR decreases as the cache blocksize increases and it increases as the number of processors increase. For a 32-byte blocksize, the smaller grid size MRs range from 0.01 to 0.06 as the number of processors increase. Finally, the MR for the 128x128 point grid size and 64Kbyte cache size has better performance when direct mapping is used as discussed in section 5.2.1.

The IR generally decreases as the cache blocksize and the PDE grid size increases for all processor configurations. This value also increases as the number of processors increase. For a 32-byte blocksize, the IR ranges from 0.005 to 0.055 as the number of processors increase. The prXICO initially decreases as the cache blocksize increases from 8 to 16 bytes, then it increases as the blocksize increases from 16 to 32 bytes and finally, it gradually decreases as the cache blocksize increases beyond 32 bytes. This occurs for all processor configurations. Also, this parameter decreases as the PDE grid size increases. Furthermore, for a given blocksize, the prXICO increases as the number of processors increase. For a 32-byte blocksize the prXICO ranges from 0.005 to 0.015 as the number of processors increase for a 128x128 point PDE grid size.

The behavior of the prXICS for the PCG is not as consistent as the other parameters used in this study; however, this parameter generally decreases as the cache blocksize increases. As the number of processors increase from two to four, the prXICS decreases. It then increases as the number of processors increase for larger grid sizes or it increases as the number of processors increase from four to eight and then decreases as the number of processors decrease from eight to sixteen (smaller

grid sizes). The value of the prXICS is generally lower than the other parameters observed in this study. For example, this parameter ranges from 0.005 to 0.001 as the number of processors increase for the 128x128 point PDE grid size. In summarizing the demand fetching parameters, the MR has the larger value for the PCG algorithm. This is followed by the IR, an order of magnitude lower than the MR for large PDE grid sizes. The prXICO is slightly lower than the IR and finally, the prXICS is substantially lower than the prXICO.

Like Jacobi's algorithm and SOR, the PCG prefetching strategies generally reduce the MR but increases the IR, the prXICO and the prXICS. Tagged prefetching exhibits the best MR of all fetching strategies, followed by prefetch-on-miss and finally demand fetching. All other parameters studied show demand fetching exhibiting the best performance, followed by prefetching-on-miss and finally tagged prefetching. In fact, the prefetching prXICSs are considerably higher than their demand fetching counterpart for the 128x128 and 256x256 point grid sizes. The LR and PR also contribute to the performance degradation of the algorithm. Intuitively, the LR is larger than the PR and the tagged prefetching LR and PR are larger than the prefetch-on-miss values.

The single bus speedup curves are extremely low for this algorithm. This is largely the result of the long waiting time imposed by the bus. For a 32Kbyte cache size, the PCG algorithm speedup ranges from 1.2 to 1.7 as the number of processors increase from 2 to 16 processors for the large PDE grid sizes. A lower bound waiting time of zero for the full crossbar interconnection network results much larger speedup values for this system. The same 32Kbyte cache size has a speedup range from 1.8 to

14.8 as the number of processors increase from 2 to 16. Moreover, the prefetching speedup values are lower than the demand fetching values, particularly for the single bus interconnection network. The iteration time degradation increases as the number of processors increase for a given blocksize. For the 256x256 point grid size, the enabled coherence protocol executes the PCG algorithm between 1.15 and 1.25 times the disabled algorithm (per iteration).

CHAPTER 6

CONCLUSION

This chapter begins by comparing the values of the performance parameters obtained for each parallel PDE algorithm studied. This includes the algorithm speed-ups and ITDs. The motivations of this work are then discussed briefly, followed by explanations as to what knowledge was acquired. Finally, suggestions for future work are delineated.

6.1. Algorithm Comparisons

In the following discussions use of the phrase *smaller grid sizes* refers to those grid sizes where all data used in executing the algorithm maps into unique cache block frames with one exception. That being some grid sizes and the 32Kbyte cache size where some processor references to blocks holding synchronization primitives contend with references to other data blocks.

For smaller grid sizes the MR decreases as the cache blocksize increases and it increases as the number of processors increase. In instances where more than one PDE grid size falls into the small grid size category, the MR decreases as the PDE grid size increases. In cases where there are differences between the two cache sizes, the 64Kbyte cache size exhibited the better MR. Moreover, for smaller grid sizes, the MR is independent of the mapping strategy. For the 64x64 point grid size, the SOR MR is higher than Jacobi's MR and the PCG MR exhibited the worst performance for

this grid size.

For larger grid sizes the MR decreases as the cache blocksize increases for Jacobi and SOR but remains relatively constant as the blocksize is varied for PCG. Furthermore, for all three algorithms, this parameter is generally independent of the number of processors for these larger grid sizes. Additionally, the SOR and Jacobi MRs are relatively independent of the PDE grid size, and the decomposition strategy. Moreover, SOR exhibits the best MR, followed by Jacobi and then PCG if set-associative mapping is considered or by PCG and then Jacobi if direct mapping is considered.

For all three algorithms considered, the IR decreases as the cache blocksize and the PDE grid size increase and it increases as the number of processors increase. The decomposition strategies considered for SOR and Jacobi show that the rectangular IRs are larger for smaller blocksizes. The converse is true for larger blocksizes. There are no IRs for larger grid sizes when executing Jacobi's algorithm (direct mapping only). This is because cache blocks exhibiting invalidation potential are replaced before they (invalidations) occur. PCG has the largest IR followed by SOR. Although smaller, Jacobi's IRs are relatively close to the SOR values, especially for smaller grid sizes. Since a smaller MR provides an environment for more block invalidations, the slightly larger IRs for SOR are intuitive. However, while the PCG MRs are larger than Jacobi and SOR, so are the IRs. This is because the PCG write-back probability (0.3) is larger than the SOR (0.17) and Jacobi (0.2) values.

The prXICO generally decreases as the cache blocksize increases and it increases as the number of processors increase. The exceptions are the direct mapping strategy when executing SOR and the smaller blocksizes when executing PCG. Jacobi's algorithm has the lowest prXICO, followed by SOR and finally PCG, having the highest prXICO. Like the other parameters considered, the prXICS increases as the cache blocksize increases and it decreases as the number of processors increase. However, when executing Jacobi's algorithm using the square decomposition strategy and the direct mapping function, this parameter increases as the cache blocksize increases (see section 3.2.4 for an explanation). This decomposition strategy/mapping function combination also produces the largest prXICS for all algorithms considered. When considering the other decomposition strategy/mapping function combinations, SOR produces the largest prXICS and PCG exhibits the smallest values of these parameters.

This lower value for the PCG prXICS is inconsistent with the PCG behavior of the previously discussed parameters when compared to the Jacobi and SOR parameters. This is because prXICSs occur as a result of replacing RO blocks located in two caches. When executing PCG, this does not occur as often as it does for Jacobi's algorithm and SOR.

The relationship between the algorithm parameters when prefetching is used is identical to the relationship between these parameters when demand fetching is used. That is, the PCG prefetching MRs are larger than the Jacobi values which in turn are larger than the SOR values. Also, the PCG prefetching IRs and prXICOs are larger than the SOR prefetching parameters which in turn are larger than the Jacobi

parameters. Furthermore, the SOR prefetching prXICSs are larger than the Jacobi and PCG prXICSs, respectively. This is explained by the fact that the behavior of the prefetching parameters relative to the demand fetching values are the same for all three algorithms considered. Moreover, the PCG LR and PRs are larger than the SOR and Jacobi values, respectively.

Although the behavior of the prefetching parameters is similar to the demand fetching measures, the actual values are quite different. Both prefetching strategies intuitively reduce the MR over demand fetching; however, the probability of all XI operations actually increase (over demand fetching values) when prefetching. The explanation for this behavior arises from the fact that for the three algorithms considered, the successful prediction of the reference activity increases the number of shared modifiable blocks in the private caches. This in turn increases the probability of XI operations. Since prefetching increases the interconnection network waiting time, causing considerable performance degradation, this increase in XI operations adds to this degradation and results in prohibitive performance degradation for these algorithms.

For the single bus system, the SOR algorithm exhibits the largest speedup, followed by Jacobi's algorithm and finally, PCG. This is largely the result of the MRs of each algorithm, coupled with the single bus waiting time, since they (MRs) are typically an order of magnitude larger than the other parameters evaluated. The PCG algorithm exhibits the best speedup followed closely by SOR and Jacobi's algorithm, respectively. PCG also has the highest ITD followed by approximately equal SOR and Jacobi ITDs. The direct mapping values are the only exception. When using this

mapping function, Jacobi's algorithm has the highest ITD (64x64 point grid size excluded) followed by PCG and SOR, respectively.

6.2. Research Summaries

Unfortunately, the advances obtained in developing parallel processing systems have not occurred in the design of parallel algorithms. In fact, only recently has considerable work been done in this area. The motivations of this thesis research were to determine the extent of the performance degradation of specific parallel algorithms (PDEs) as a result of implementing cache coherence in a multiprocessing system. The effects of other architectural multiprocessor features directly related to cache design as well as the effects of parallel PDE algorithm design characteristics on the performance of these algorithms also served as inducements for this study. This work shows that the MR is the dominant parameter for all algorithms considered. Whereas the MR is used in evaluating the performance of caches in general, the ITD is used to study the unique performance degradation resulting from cache coherence.

Varying cache design features such as the mapping function, cache size and cache blocksize does not seem to have a significant impact on the performance degradation of the algorithm. For example, the ITDs for PCG are relatively independent of the mapping function as shown in Figure 5.32. The most significant impact on the ITD is the PDE grid size and the interconnection network. More specifically, when using a single bus system, if the PDE grid size is such that all data blocks map into unique cache block frames, then the degradation resulting from implementing cache coherence is optimum. This is verified by the 64x64 point ITDs of all three algo-

rithms. The values of these ITDs are relatively higher than all other grid size/interconnection network combinations.

The ITDs of each algorithm show that implementing cache coherence results in a parallel algorithm iteration time that is no greater than 1.6 (PCG) or 1.45 (SOR and Jacobi) times the disabled coherence execution time. Furthermore, on the average, implementing the coherence protocol evaluated in this study results in SOR and Jacobi parallel algorithm iteration times between 5 to 10 percent slower than their disabled coherence times. When using the direct mapping strategy, the enabled coherence parallel algorithm iteration time falls to between 10 to 30 percent slower than the disabled coherence times for Jacobi's algorithm. Moreover, the PCG enabled coherence parallel algorithm iteration time is generally 10 to 30 percent slower than the disabled coherence times.

The behavior of all other parameters considered in this study is intuitive. That is, the single-bus system, the square decomposition strategy (for a 32-byte blocksize), 32Kbyte cache size and the smaller grid sizes produce higher performance degradations than the full crossbar system, the rectangular decomposition strategy, the 64Kbyte cache size and the larger grid sizes, respectively. With the exception of the direct mapping/Jacobi's algorithm combination, the ITDs are relatively independent of the cache mapping function. As a result of the considerable performance degradation for this exception, it is suggested that the two grid implementation of Jacobi's algorithm studied here should not be used. Furthermore, since the two grid implementation prohibitively degrades Jacobi's algorithm and since PCG generally has a higher ITD, SOR is best suited for the coherence algorithm considered in this

work. PCG is second best, followed by Jacobi's algorithm.

6.3. Future Work

This work concentrated on how a specific implementation of one particular feature of a general purpose shared-memory multiprocessor architecture, cache coherence, effects the performance of three parallel PDE algorithms. All performance parameters obtained in this study are strongly dependent upon the actual algorithm implementations evaluated. Therefore, generalizations about the parallel PDE algorithms considered cannot be made from the results obtained. This work serves as a cutting edge. Numerous extensive studies are needed to obtain more detailed insights on the performance of parallel PDE algorithms.

Firstly, other implementations of the algorithms evaluated, particularly SOR and PCG should be studied. For Jacobi's algorithm and SOR, only 5-point discretizations on a square grid with rectangular and square decomposition strategies were considered. Future work may include 7-point, 9-point, 13-point or other point discretizations. Also, other decomposition strategies, like triangular and hexagonal [16] should be evaluated. Various preconditioning matrices have been proposed for parallel PCGs. The performance degradations resulting from these implementations need to be researched. Moreover, the effects of other coherence protocols (software or hardware implementable) on several implementations of individual parallel PDE algorithms and/or the effects individual coherence protocols of the performance of other algorithms should be considered.

Furthermore, only data independent algorithms are considered in this work. Suggestions for future work include evaluating the performance degradation resulting from implementing cache coherence on data dependent algorithms such as sorting and searching. Moreover, only upper and lower speedup bounds are considered in this work. A detailed model of the interconnection networks, evaluating the average waiting times, is needed. This model may then be used in conjunction with parameters used to measure performance degradation resulting from cache coherence. Finally, a close examination of other multiprocessor architectural features such as pipelining, input/output (I/O) and other interconnection networks is needed to provide a better understanding of the underlying factors that cause performance degradations in parallel algorithms.

REFERENCES

- [1] Voigt, Robert G., "Where are the Parallel Algorithms?," Proceedings of the 1985 National Computer Conference, vol. 54, pp.329-334 (1985).
- [2] Ortega, J. A. and Voigt, R. G., "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Review*, 27(2), pp.140-240 (June 1985).
- [3] Young, D., *Iterative Solution of Large Linear Systems*, New York: Academic Press (1971).
- [4] Manuel, Tom, "How Sequent's New Model Outruns Most Mainframes," *Electronics*, 60(11), pp.76-79 (May 28, 1987).
- [5] Hill, Mark, Eggers, Susan, Larus, Jim, Taylor, George, Adams, Glenn, Bose, B. K., Gibson, Garth, Hansen, Paul, Kelier, Jon, Kong, Shing, Lee, Corinna, Lee, Daebum, Pendleton, Joan, Ritchie, Scott, Wood, David, Zorn, Ben, Hilfinger, Paul, Hodges, Dave, Katz, Randy, Ousterhout, John, and Patterson, Dave, "Design Decisions in SPUR," *Computer*, 19(10), pp.8-22 (November, 1986).
- [6] Tucker, S. G., "The IBM 3090 System: An Overview," *IBM System Journal*, 25(1), pp.4-19 (1986).
- [7] Smith, Alan J., "Cache Memories," *Computing Surveys*, 14(3), pp.473-530 (September 1982).
- [8] Dubois, Michel and Briggs, Faye A., "Synchronization, Coherence and Ordering of Events in Multiprocessors," *Computer*, 21(2), pp.9-21 (February, 1988).
- [9] Dubois, M. and Briggs, F. A., "Effects of Cache Coherency in Multiprocessors," *IEEE Transactions on Computers*, C-31(11) (November 1982).
- [10] Archibald, James and Baer, Jean-Loup, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Transactions on Computer Systems*, 4(4), pp.273-298 (November, 1986).

- [11] Lee, Roland L., Yew, Pen-Chung, and Lawrie, Duncan H., "Multiprocessor Cache Design Considerations," *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pp.253-262 (June 1987).
- [12] Fox, Geoffrey C. and Otto, Steve W., "Algorithms for Concurrent Processors," *Physics Today*, 37(5), pp.50-59 (May, 1984).
- [13] Vrsalovic, Dalibor, Gehringer, Edward F., Segall, Zary Z., and Siewiorek, Daniel P., "The Influence of Parallel Decomposition Strategies on the Performance of Multiprocessors Systems," *Proceedings of the 12th Annual International Symposium on Computer Architecture*, pp.396-405 (June, 1985).
- [14] Saltz, Joel H., Naik, Vijay K., and Nicol, David M., "Reduction of the Effects of the Communication Delays in Scientific Algorithms on Message Passing MIMD Architectures," ICASE Report, no. 86-4, NASA Langley Research Center, Hampton, VA (January, 1986).
- [15] Rattner, J., "Concurrent Processing: A New Direction in Scientific Computing," *Proceedings of the 1985 National Computer Conference*, vol. 54, pp.157-166 (1985).
- [16] Reed, Daniel A., Adams, Loyce M., and Patric, Merrell L., "Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processors Systems," *IEEE Transactions of Computers*, C-36(7), pp.845-858 (July, 1987).
- [17] Dubois, Michel, "Throughput Analysis of Cache-Based Multiprocessors with Multiple Buses," *IEEE Transactions on Computers*, 37(1), pp.58-70 (January 1988).
- [18] Dubois, Michel, "Effect of Invalidations on the Hit Ratio of Cache-Based Multiprocessors," *Proc. of the 1987 International Conference on Parallel Processing*, pp.255-257 (August 1987).
- [19] Brantley, W. C., McAuliffe, K. P., and Weiss, J., "RP3 Processor-Memory Element," *Proceedings of the 13th Annual Symposium of Computer Architecture* (June 1986).
- [20] Cheriton, David R., Slavenburg, Gert A., and Boyle, Patrick D., "Software-Controlled Caches in the VMP Multiprocessor," *Proceedings of the 13th Annual Symposium on Computer Architecture*, pp.366-374 (June 1986).

- [21] Cheriton, David R., Gupta, Anoop, Boyle, Patric D., and Goosen, Hendrick A., "The VMP Multiprocessor: Initial Experience, Refinements, and Performance Evaluation," (manuscript), Department of Computer Science, Stanford University, CA (1987).
- [22] Yeh, P. C., "Shared Cache Organization for Multiple-Stream Computer Systems," Technical Report R-904, Corrdinated Science Lab., University of Illinois (January, 1981).
- [23] Goodman, James R., "Using Cache Memory to Reduce Processor-Memory Traffic," Proceedings of the 10th Annual Symposium on Computer Architecture, pp.124-131 (June 1983).
- [24] Rudolph, Larry and Segall, Zary, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," 11th Annual International Symposium on Computer Architecture, pp.340-347 (June 1984).
- [25] Papamarcos, Mark S. and Patel, Janak H., "A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories," 11th Annual International Symposium on Computer Architecture, pp.348-354 (June 1984).
- [26] Archibald, J. and Baer, J. L., "An Economical Solution to the Cache Coherence Problem," 11th Annual International Symposium on Computer Architecture, pp.355-362 (June 1984).
- [27] Frank, Steven J., "Tightly Coupled Multiprocessor Systems Speeds Memory-access Times," *Electronics*, 57(1), pp.164-169 (January 12, 1984).
- [28] Katz, R. H., Eggers, S. J., Wood, D. A., Perkins, C. L., and Sheldon, R. G., "Implementing a Cache Consistency Protocol," Proceedings of the 12th Annual International Symposium on Computer Architecture, pp.276-283 (June 1985).
- [29] Bitar, P. and Despain, A., "Multiprocessor Cache Synchronization: Issues, Innovations, Evolution," Proceedings of the 13th Annual International Symposium on Computer Architecture (June 1986).
- [30] Sweazey, Paul and Smith, Alan Jay, "A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus," Proceedings of the 13th Annual International Symposium on Computer Architecture, pp.414-423 (June, 1986).

- [31] Tang, C. K., "Cache System Design in the Tightly Coupled Multiprocessor System," Proc. 1976 AFIPS National Computer Conference, pp.740-753 (1976).
- [32] Censier, L. M. and Feautrier, P., "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, C-27(12), pp.1112-1118 (December 1978).
- [33] Yen, Wei C., Yen, David W. L., and Fu, King-Sun, "Data Coherence Problem in a Multicache System," *IEEE Transactions on Computers*, C-34(1), pp.56-65 (January, 1985).
- [34] Hwang, K. and Briggs, F. A., *Computer Architecture and Parallel Processing*, New York: McGraw-Hill (1984).
- [35] Scheurich, Christoph and Dubois, Michel, "Correct Memory Operation of Cache-Based Multiprocessors," 14th Annual International Symposium on Computer Architecture, pp.234-243 (June 1987).
- [36] Smith, Alan Jay, "Sequential Program Prefetching in Memory Hierarchies," *Computer*, 11(12), pp.7-21 (December, 1978).
- [37] Abraham, S., Gottlieb, A., and Kruskal, C., "Simulating Shared-Memory Parallel Computers," Ultracomputer Note Number 70, Courant Institute, NYU (April 1984).
- [38] Abu-Sufah, W. and Kwok, A. Y., "Performance Prediction Tools for Cedar: A Multiprocessor Supercomputer," Proceedings of the 12th Annual Symposium on Computer Architecture, pp.406-413 (June 1985).
- [39] Axerold, T. S., Dubois, P., and Elgroth, P., "A Simulator for MIMD Performance Prediction--Application to the S-1 MkIIa Multiprocessor," *Parallel Computing*, 1(3), North-Holland, pp.237-274 (1984).
- [40] Dubois, Michel, Briggs, Faye' A., Patil, Indira, and Balakrishnan, Meera, "Trace-Driven Simulations of Parallel and Distributed Algorithms in Multiprocessors," International Conference on Parallel Processing (August 1986).

- [41] Evans, D. J., "Parallel SOR Iterative Methods," *Parallel Computing*, 1(1), pp.3-18 (1984).
- [42] Jayasimha, Doddaballapur N., "Parallel Access to Synchronization Variables," Proc. of the 1987 International Conference on Parallel Processing, pp.97-99 (August 1987).
- [43] Lewis, John Gregg and Rehm, Ronald G., "The Numerical Solution of a Non-separable Elliptic Partial Differential Equation by Preconditioned Conjugate Gradients," *Journal of Research of the National Bureau of Standards*, 85(5), pp.387-390 (September-October, 1980).
- [44] Concus, P., Golub, G., and O'Leary, D., "Sparse Matrix Computations," A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations, ed. Rose, Academic Press, New York (1976).
- [45] Lichnewsky, A., "High Speed Computations," Some Vector and Parallel Implementations of Preconditioned Conjugate Gradient Algorithms, ed. J. S. Kowalik, pp.343-359, Springer, Berlin (1984).
- [46] Meurant, Gerard, "Multitasking the Conjugate Gradient Method on the CRAY X-MP/48," *Parallel Computing*, 5(3), pp.267-280 (1987).