Time/Contention Trade-offs for Multiprocessor Synchronization*

James H. Anderson
Department of Computer Science
The University of North Carolina
Chapel Hill, North Carolina 27599-3175

Jae-Heon Yang
Department of Computer Science
The University of Maryland
College Park, Maryland 20742-3255

May 1994
Revised May 1995

Abstract

We establish trade-offs between time complexity and write- and access-contention for solutions to the mutual exclusion problem. The *write-contention* (*access-contention*) of a concurrent program is the number of processes that may be simultaneously enabled to write (access by reading and/or writing) the same shared variable. Our notion of time complexity distinguishes between local and remote accesses of shared memory.

We show that, for any N-process mutual exclusion algorithm, if write-contention is w, and if at most v remote variables can be accessed by a single atomic operation, then there exists an execution involving only one process in which that process executes $\Omega(\log_{vw} N)$ remote operations for entry into its critical section. We further show that, among these operations, $\Omega(\sqrt{\log_{vw} N})$ distinct remote variables are accessed. For algorithms with access-contention c, we show that the latter bound can be improved to $\Omega(\log_{vc} N)$. The last two of these bounds imply that a trade-off between contention and time complexity exists even if coherent caching techniques are employed.

In most shared-memory multiprocessors, an atomic operation may access only a constant number of remote variables. In fact, most commonly-available synchronization primitives (e.g., read, write, test-and-set, load-and-store, compare-and-swap, and fetch-and-add) access only one remote variable. In this case, the first and the last of our bounds are asymptotically tight.

Our results have a number of important implications regarding specific concurrent programming problems. For example, the time bounds that we establish apply not only to the mutual exclusion problem, but also to a class of decision problems that includes the leader-election problem. Also, because the execution that establishes these bounds involves only one process, it follows that "fast mutual exclusion" requires arbitrarily high write-contention. Although such conclusions are interesting in their own right, we believe that the most important contribution of our work is to identify a time complexity measure for asynchronous concurrent programs that strikes a balance between being conceptually simple and having a tangible connection to real performance.

Keywords: Concurrent programs, lower bounds, mutual exclusion, remote memory references, shared memory, time complexity.

*A preliminary version of this paper was presented at the 26th ACM Symposium on Theory of Computing [20]. Work supported, in part, by NSF Contracts CCR-9109497 and CCR-9216421, by the NASA Center for Excellence in Space Data and Information Sciences (CESDIS), and by an IBM Fund Award.

1 Introduction

The mutual exclusion problem is a fundamental paradigm for coordinating accesses to shared data on asynchronous shared-memory multiprocessing systems [6]. In this problem, accesses to shared data are abstracted as "critical sections" of code, and it is required that at most one process executes its critical section at any time. In this paper, we consider bounds on time for mutual exclusion, a subject that has received scant attention in the literature. Past work on the complexity of mutual exclusion has almost exclusively focused on space requirements [4]; the limited work on time bounds that has been done has focused on partially synchronous models [14].

The lack of prior work on time bounds for mutual exclusion within asynchronous models is probably due to difficulties associated with measuring the time spent within busy-waiting constructs. In fact, because of such difficulties, there has been scarcely little work of any kind on time bounds for asynchronous concurrent programming problems for which busy-waiting is inherent. One of the primary contributions of this paper is to show that it *is* possible to establish meaningful time bounds for such problems.

A natural approach to measuring the time complexity of a mutual exclusion algorithm would be to simply use the standard sequential programming measure of counting all operations. However, in any algorithm in which processes busy-wait, the number of operations needed for one process to get to its critical section is unbounded in the worst case. In other words, the standard sequential programming metric yields no useful information concerning the performance of such algorithms under contention.

In a recent paper, we proposed a time measure for concurrent programs that distinguishes between local and remote accesses of shared memory [19]. This measure is motivated by recent work on scalable synchronization constructs [3, 8, 15]. Informally, a shared variable access is *local* if does not require a traversal of the global interconnect between processors and shared memory, and is *remote* otherwise. Although the notion of a *locally* accessible *shared* variable may seem counterintuitive, there are two mainstream architectural paradigms that support it. In particular, on distributed shared-memory machines, a shared variable can be made locally accessible by storing it in a local portion of shared memory, and on cache-coherent machines, a shared variable can become locally accessible by migrating to a local cache line.

Under our proposed measure, the time complexity of a concurrent program is measured by counting only remote accesses of shared variables; local accesses are ignored. This measure satisfies two criteria that must be met by any reasonable complexity measure. First, it is conceptually simple. In fact, this measure is a natural descendent of the standard time complexity measure used in sequential programming. Second, this measure has a tangible connection with real performance, as demonstrated by a number of recently-published performance studies of synchronization algorithms [3, 8, 15, 19]. In each of these studies, those algorithms that minimize remote memory references exhibited the best performance under contention. All other proposed time complexity measures for asynchronous concurrent programs that we know of fail to satisfy at least one of these two criteria.[1]

We present several lower-bound results for mutual exclusion that are based on the time complexity measure mentioned above. These results establish trade-offs between time complexity and write- and access-contention for solutions to the mutual exclusion problem. The *write-contention* (*access-contention*) of a concurrent program is the number of processes that may be simultaneously enabled to write (access) the same shared variable. Limiting access-contention is an important consideration when designing algorithms for problems, such as mutual exclusion and shared counting, that must cope well with high competition

[1] Note that our time complexity measure cannot be used to make distinctions between programs that busy-wait on remote variables. However, many concurrent programming problems that require busy-waiting (including mutual exclusion) can be solved without busy-waiting on such variables. Using our time measure, all solutions to such problems in which processes busy-wait on remote variables are deemed as being equally "bad" — all have unbounded time complexity.

among processes [3, 10, 11, 17]. Performance problems associated with high access-contention can be partially alleviated by employing coherent caching techniques to reduce concurrent reads of the same memory location. However, even when such techniques are employed, limiting write-contention is still an important concern.

We show that, for any N-process mutual exclusion algorithm, if write-contention is w, and if each atomic operation accesses at most v remote variables, then there exists an execution involving only one process in which that process executes $\Omega(\log_{vw} N)$ remote operations for entry into its critical section. We further show that, among these operations, $\Omega(\sqrt{\log_{vw} N})$ distinct remote variables are accessed. For algorithms with access-contention c, we show that the latter bound can be improved to $\Omega(\log_{vc} N)$. We emphasize that all of our bounds are established in the absence of competition.

These results have a number of important implications. For example, because the first access of any variable causes a cache miss, the latter two bounds imply that a time/contention trade-off exists even if coherent caching techniques are employed. Also, because the execution that establishes these bounds involves only one process, it follows that so-called fast mutual exclusion algorithms — i.e., algorithms that require a process to execute only a constant number of remote memory references in the absence of competition [12] — require arbitrarily high write-contention in the worst case. These bounds apply not only to the mutual exclusion problem, but also to a class of decision problems that includes the leader-election problem.

In most shared-memory multiprocessors, an atomic operation may access only a constant number of remote variables. In fact, most commonly-available synchronization primitives access only one remote variable; examples include read, write, test-and-set, load-and-store, compare-and-swap, and fetch-and-add. If v is taken to be a constant, then our results imply that, for any N-process mutual exclusion algorithm with write-contention w, some process executes $\Omega(\log_w N)$ remote operations in the absence of competition for entry into its critical section. Further, among these remote operations, $\Omega(\sqrt{\log_w N})$ distinct remote variables are accessed. For algorithms with access-contention c, the latter bound is improved to $\Omega(\log_c N)$. It can be shown that the first and last of these bounds are asymptotically tight.

Related work includes previous research by Dwork et al. given in [7], where it is shown that solving mutual exclusion with access-contention c requires $\Omega((\log_2 N)/c)$ memory references. Our work extends that of Dwork et al. in several directions. First, the implications concerning fast mutual exclusion and cache coherence noted above do not follow from their work, because Dwork et al. do not consider competition-free executions, and because they do not count the number of distinct variables accessed by a process for entry into its critical section. Second, we consider programs in which atomic operations may access multiple shared variables, whereas they only consider reads, writes, and read-modify-writes. Third, in our main result, we restrict only write-contention, and if v is a constant, then we obtain a tight bound of $\Omega(\log_w N)$, which exceeds the bound established by them. Finally, and most importantly, Dwork et al. make no distinction between local and remote shared memory accesses. Because busy-waiting is required for mutual exclusion in general, an unbounded number of memory accesses (local or remote) are required in the worst case. It is our belief that time complexity results that do not distinguish between local and remote accesses of shared memory are of questionable value as a measure of performance of mutual exclusion algorithms under contention.

The rest of the paper is organized as follows. In Section 2, we present our model of shared memory systems. In Section 3, we define a simplified version of the mutual exclusion problem called the "minimal" mutual exclusion problem. The above-mentioned time bounds are then established in Sections 4 and 5. Concluding remarks appear in Section 6.

2 Shared-Memory Systems

Our model of a shared-memory system is similar to that given by Merritt and Taubenfeld in [16]; much of our notation is borrowed from Chandy and Misra [5]. A *system* $S = (C, P, V)$ consists of a set of computations C, a set of processes $P = \{1, 2, \ldots, N\}$, and a set of variables V. A *computation* is a finite sequence of events.

An *event* is denoted $[R, W, i]$, where $R = \{(x_j, u_j) | 1 \leq j \leq m\}$ for some m, $W = \{(y_k, v_k) | 1 \leq k \leq n\}$ for some n, and $i \in P$; this notation represents reading value u_j from variable x_j, for $1 \leq j \leq m$, and writing value v_k to variable y_k, for $1 \leq k \leq n$. Each variable in R (W) is assumed to be distinct. We say that this event *accesses* each such x_j and y_k. We use $R.var$ to denote the set of variables x_j such that $(x_j, u_j) \in R$ for some u_j, and $W.var$ to denote the set of variables y_k such that $(y_k, v_k) \in W$ for some v_k.

Each variable is *local* to at most one process and is *remote* to all other processes. (Note that we allow variables that are remote to *all* processes.) An *initial value* is associated with each variable. An event is *local* if it does not access any remote variable, and is *remote* otherwise.

We use $\langle e, \ldots \rangle$ to denote a computation that begins with the event e, and $\langle\rangle$ to denote the empty computation. We define the *length* of computation H, denoted $|H|$, as the number of events in H. $H \circ G$ denotes the computation obtained by concatenating computations H and G. If G is a subsequence of H, then $H - G$ is the computation obtained by removing all events in G from H. The value of variable x at the end of computation H, denoted $value(x, H)$, is the last value that is written to x in H (or the initial value of x if x is not written in H). The last event to write variable x in H is denoted $writer(x, H)$. If x is not written by any event in H, then we let $writer(x, H) = \perp$.

An *extension* of computation H is a computation of which H is a prefix. For a computation H and a set of processes Y, H_Y denotes the subsequence of H that contains all events in H of processes in Y.

Computations H and G are *equivalent* with respect to a set of processes Y, denoted $H[Y]G$, iff $H_Y = G_Y$. Note that $[Y]$ is an equivalence relation. We now present our model of shared-memory systems.

Definition: A *shared-memory system* $S = (C, P, V)$ is a system that satisfies the following properties.

- (P1) If $H \in C$ and G is a prefix of H, then $G \in C$. Informally, every prefix of a computation is also a computation.
- (P2) If $H \circ \langle [R, W, i] \rangle \in C$, $G \in C$, $G[Y]H$, and $i \in Y$, and if for all $x \in R.var$, $value(x, G) = value(x, H)$ holds, then $G \circ \langle [R, W, i] \rangle \in C$. Informally, if two computations H and G are not distinguishable to process i, if i can execute $[R, W, i]$ after H, and if all variables in R have the same values after H and G, then i can execute $[R, W, i]$ after G.
- (P3) If $H \circ \langle [R, W, i] \rangle \in C$, $G \in C$, $G[Y]H$, and $i \in Y$, then $G \circ \langle [R', W', i] \rangle \in C$ for some R' and W' such that $R'.var = R.var$ and $W'.var = W.var$. Informally, if two computations H and G are not distinguishable to process i, and if i can execute $[R, W, i]$ after H, then i can read and write the same variables in the next operation after G. (Note that the values read or written might be different.)
- (P4) For any $H \in C$, $H \circ \langle [R, W, i] \rangle \in C$ only if for all $(x, v) \in R$, $v = value(x, H)$ holds. Informally, only the last value written to a variable can be read. □

For simplicity, we call a remote event a *remote read* if it reads a remote variable, and a *remote write* if it writes remote variables. Note that a remote event can be both a remote read and a remote write.

Consider a shared-memory system $S = (C, P, V)$. A computation H is a Y-computation iff either $H = \langle\rangle$ and $Y \subseteq P$, or Y is the minimal subset of P such that $H = H_Y$ holds. For simplicity, we abbreviate the

preceding definitions when applied to a singleton set of processes. For example, if $Y = \{i\}$, then we use H_i to mean $H_{\{i\}}$, i-computation to mean $\{i\}$-computation, and $[i]$ to mean $[\{i\}]$.

In the following sections, we establish time bounds involving three notions of contention, which are defined below. These definitions apply to a shared-memory system $S = (C, P, V)$. The first and strictest notion of contention we use is static in nature. It bounds the number of processes that may read or write a given shared variable in *throughout* any computation. The other two notions of contention that we employ are dynamic in nature. They bound the number of processes that may *simultaneously* write (access) the same memory location.

Definition: Consider a variable x in V. A process i in P is a *reader* (*writer*) of x iff there is an event of i that reads (writes) x in some computation in C. We say that x is a *k-reader* (*k-writer*) *variable* iff there are k readers (writers) of x. □

Definition: For $H \in C$ and $x \in V$, let $overwriters(x, H) \equiv \{i \mid H \circ \langle [R, W, i] \rangle \in C \text{ where } x \in W.var\}$. Then, the *write-contention* of S is $\mathbf{max}_{x \in V, H \in C}(|overwriters(x, H)|)$. □

Definition: Let $contenders(x, H) \equiv \{i \mid H \circ \langle [R, W, i] \rangle \in C \text{ where } x \in (R.var \cup W.var)\}$. Then, the *access-contention* of S is $\mathbf{max}_{x \in V, H \in C}(|contenders(x, H)|)$. □

3 Minimal Mutual Exclusion

Our main results concerning the mutual exclusion problem are based on a simplified version of the problem, which we call the "minimal mutual exclusion problem".

Minimal Mutual Exclusion Problem: We define the minimal mutual exclusion problem for a shared-memory system $S = (C, P, V)$ as follows. Each process $i \in P$ has a local variable $i.dine$ that ranges over $\{think, hungry, eat\}$. Variable $i.dine$ is initially $think$ and is accessed only by the following events:

$$\begin{aligned} Think_i &\equiv [\{\}, \{(i.dine, think)\}, i] \\ Hungry_i &\equiv [\{\}, \{(i.dine, hungry)\}, i] \\ Eat_i &\equiv [\{\}, \{(i.dine, eat)\}, i] \end{aligned}$$

The allowable transitions of $i.dine$ are as follows: for any $H \in C$, $H \circ \langle Think_i \rangle \in C$ iff $value(i.dine, H) = eat$; $H \circ \langle Hungry_i \rangle \in C$ iff $value(i.dine, H) = think$; and if $H \circ \langle Eat_i \rangle \in C$, then $value(i.dine, H) = hungry$. System S solves the minimal mutual exclusion problem iff the following requirements are satisfied.

- *Exclusion*: For any $H \in C$ and processes $i \neq j$, $value(i.dine, H) = eat \Rightarrow value(j.dine, H) \neq eat$.
- *Progress*: For any $H \in C$ and process $i \in P$, if H is an i-computation, then either H contains Eat_i, or there exists an i-computation G such that $H \circ G \circ \langle Eat_i \rangle \in C$. □

Note that the Progress requirement above is much weaker than that usually specified for the mutual exclusion problem. (This, of course, strengthens our impossibility results.) Note also that any solution to the leader election problem easily solves the minimal mutual exclusion problem. Thus, our time bounds apply not only to the mutual exclusion problem, but also to the leader election problem, and any other decision problem that can be used to directly solve leader election.[2]

[2]For example, the *ranking* problem. In this problem, each process is assigned a "rank" between 1 and N. The process that obtains a rank of 1 can be defined to be the "leader".

Before presenting our main results, we give bounds for the case of statically-defined contention. In this theorem and those that follow, we assume that S is a shared-memory system and that $i \in P$.

Theorem 1: For any $S = (C, P, V)$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, and if either all variables in V are k-reader variables, or all variables in V are k-writer variables, then there exists an i-computation in C that contains $\Omega(N/vk)$ remote events but no Eat_i event.

Proof: Suppose that all variables in V are k-reader variables. (A similar argument applies if all variables are k-writer variables.) By the Progress requirement of the minimal mutual exclusion problem, there exists an i-computation $H(i) \cdot Eat_i$ in C for each $i \in P$ such that $H(i)$ does not contain Eat_i. Let $C' = \{H(i) \mid i \in P\}$.

It can be shown that for each i and j such that $i \neq j$, $H(i)$ contains a write of a variable that is read in $H(j)$. (Otherwise, we could show that $H(i) \circ H(j) \circ Eat_i \circ Eat_j$ is a computation in C, violating the Exclusion requirement.) Select one such variable for each pair (i, j) where $i \neq j$. Let V' be the set of the variables selected.

Because each variable is a k-reader variable, $H(i)$ contains writes of at least $\lceil (N-1)/k \rceil$ variables in V'. If there exists $i \in P$ such that $\lceil (N-1)/2k \rceil$ such variables are remote to i, then the theorem easily follows. So, assume that each process $i \in P$ has at least $\lceil (N-1)/2k \rceil$ such variables, denoted as L_i, as local variables.

Observe that $L_i \subseteq V'$ and, because the variables in L_i are local to i, $L_i \cap L_j = \{\}$ holds for any $i \neq j$. By the construction of V', for each $x \in L_i$, there exists $H(j)$ in C' that contains a remote event reading x, where $j \neq i$. Thus, there exists a set of remote events in C' that collectively read at least $\lceil (N-1)/2k \rceil$ variables in L_i (remotely). Thus, there exists a set of remote events in C' that collectively read at least $\lceil N(N-1)/2k \rceil$ variables in V' (remotely). If each event accesses at most v remote variables, then by the pigeon-hole principle, there exists an i-computation in C' that contains at least $\lceil (N-1)/2vk \rceil$ remote events. □

For any N-process system S that satisfies the conditions of Theorem 1, some process i executes $\Omega(N/vk)$ remote events in the absence of competition. If we remove process i from system S, we obtain a system that satisfies the conditions of the theorem with N replaced by $N-1$. Thus, there is a process $j \neq i$ in system S that executes $\Omega((N-1)/vk)$ remote events in the absence of competition. Continuing in this manner, at least half the processes in S execute at least $\Omega(N/2vk)$ remote events in the absence of competition. Thus, we have the following corollary.

Corollary 1: For any system S satisfying the conditions of Theorem 1, there exist $\Omega(N)$ processes i in P for which the conclusion of the theorem holds. □

Similar corollaries apply to the theorems in the following sections.

In [2], a mutual exclusion algorithm requiring $O(N)$ remote memory references per critical section acquisition is given that employs only single-reader, single-writer variables. Thus, if v and k are taken to be positive constants, then the bound of Theorem 1 is asymptotically tight. In the remainder of the paper, we consider more interesting bounds based on dynamic notions of contention.

4 Main Result: Bounding Remote Events

In this section, we show that for any system with write-contention w, if an event may access at most v remote variables, then $\Omega(\log_{vw} N)$ remote events are required in the absence of competition to solve the minimal mutual exclusion problem. Formally, this result is stated as follows.

Theorem 3: For any $S = (C, P, V)$ with write-contention $w > 1$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, then there exists an i-computation in C that contains $\Omega(\log_{vw} N)$ remote events but no Eat_i event. □

This bound has important consequences for distributed shared-memory multiprocessing systems. On such systems, remote events require a traversal of the global interconnection network and hence are more expensive than local events. Thus, for such machines, the lower bound of Theorem 3 not only gives the inherent time complexity of implementing critical sections, it also bounds the communication complexity measured in terms of global traffic.

4.1 Proof Strategy

Theorem 3 is proved by considering a class of computations, as defined by a set of conditions. Each of these conditions refers to an arbitrary computation H in this class. The first condition is as follows.

- (C1) For events $[R, U, i]$ and $[T, W, j]$ in H, if $(R.var \cap W.var) \neq \{\}$ holds and $[T, W, j]$ precedes[3] $[R, U, i]$ in H, then $i = j$. Informally, no process reads a variable that is accessed by a preceding write of another process in H.

We will use this condition and those that follow to inductively construct longer and longer computations. Condition (C1) eliminates "information flow" between processes in the computations so constructed.

The first of the remaining conditions refers to "active" processes. If $H = \langle\rangle$ or $H_i \neq \langle\rangle$, then process i is *active* in H; otherwise i is *inactive* in H. The notion of an active process will arise in subsequent inductive proofs. Initially, all processes are active; in a non-null computation, only those processes that have taken steps are active.

- (C2) For any event $[R, W, i]$ in H, if $x \in (R.var \cup W.var)$, and if x is local to a process j that is active in H, then $i = j$. Informally, no local variable of an active process is accessed by other processes in H.
- (C3) For any events $[R, W, i]$ and $[T, U, j]$ in H, if $(W.var \cap U.var) \neq \{\}$, then $i = j$. Informally, each variable is written by at most one process in H.
- (C4) For any prefix G of H, $value(i.dine, G) \neq eat$. Informally, no process eats in H.

By (C2), "information flow" between processes can only occur through remote events in the computations we inductively construct. Condition (C3) makes it easier for us to make an active process inactive, i.e., remove its events from a given computation. In particular, because each variable is written by at most one process, if a process is made inactive, then the variables it writes simply take on their initial values. Condition (C4) arises because we intend to compute the time complexity required for a process to eat for the first time.

[3] Although our definition of an event allows multiple instances of the same event, we assume that such instances are distinguishable from each other. (For simplicity, we do not extend our notion of an event to include an additional identifier for distinguishability.)

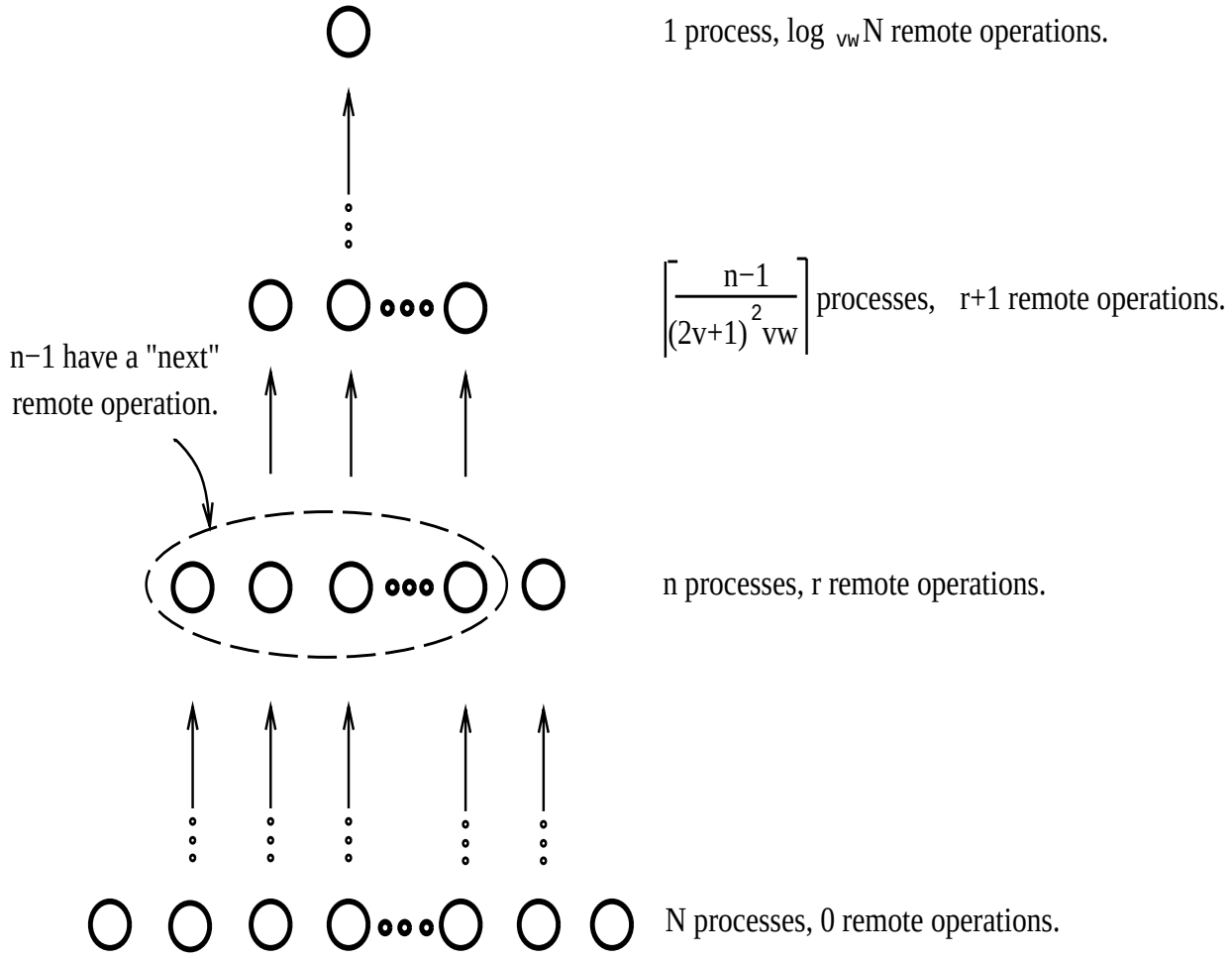

Figure 1: Proof Strategy.

The structure of our proof is depicted in Figure 1. In the induction step, we assume that there are n processes, each of which executes r remote operations, in a computation H that satisfies conditions (C1) through (C4). We show in Lemma 5 that, of these n processes, at least $n-1$ processes have a "next" remote operation to execute. Then, in Lemma 6, we identify a subset of $\lceil (n-1)/(2v+1)^2 vw \rceil$ processes that can execute a next remote operation without violating any of the conditions (C1) through (C4), and construct a computation G in which only these selected processes execute $r+1$ remote operations each. This induction step provides the $\Omega(\log_{vw} N)$ bound.

4.2 Proofs

We begin by presenting several lemmas that are needed to prove the main theorem. The first lemma directly follows from the definitions of $value(x, H)$ and $writer(x, H)$.

Lemma 1. $writer(x, H) = writer(x, G) \;\Rightarrow\; value(x, H) = value(x, G)$. □

We now present several technical lemmas. For most of these lemmas, we provide only an informal proof sketch in this section; detailed formal proofs are given in an appendix.

The next lemma gives us a means for projecting a computation onto a set of processes so that the resulting projection is itself a computation.

Lemma 2: For any $S = (C, P, V)$, if $G \circ H$ is a computation in C satisfying (C1), then for any $Y \subseteq P$, $G \circ H_Y \in C$.

Proof: For any process in Y, H_Y is not distinguishable from H. Thus, we can let processes in Y execute the same events after G as they execute in H. Note that (C1) implies that the prefix G does not affect the processes in Y. A full proof is presented in the appendix. □

The next two lemmas give us means for extending a computation. We will usually use these lemmas to extend a computation by appending local events.

Lemma 3: Consider $S = (C, P, V)$. Let F, G, and H be computations such that for some $i \in P$, F is an i-computation, no event in F accesses a variable that is written by processes other than i in either G or H, $H \in C$, and $G[i]H$. If $G \circ F \in C$, then $H \circ F \in C$.

Proof: Because no event in F accesses any variable that is written by another process in either G or H, process i cannot distinguish H from G, even if it executes all events in F. Thus, if $G \circ F \in C$, then $H \circ F \in C$. A more formal proof is presented in the appendix. □

Lemma 4: Consider $S = (C, P, V)$ and $Q \subseteq P$, where every process in Q is active in H. Without loss of generality, assume that the processes are numbered so that $Q = \{1, 2, \ldots, |Q|\}$. Let H and $L(j)$, $1 \leq j \leq |Q|$, be computations satisfying the following conditions: $L(j)$ is a j-computation; $H \circ L(j) \in C$; and no event in $L(j)$ accesses any variable that is accessed by other processes in $H \circ L(1) \circ L(2) \circ \cdots \circ L(|Q|)$. Then, $H \circ L(1) \circ L(2) \circ \cdots \circ L(|Q|) \in C$.

Proof: The lemma is established by inductively applying Lemma 3 to append each $L(j)$ in turn. A formal proof is given in the appendix. □

According to the next lemma, if n processes are competing for entry into their critical sections, and if each of these n processes has no knowledge of the others, then at least $n - 1$ of the processes have at least one more remote event to execute. To formally capture the latter, consider a system $S = (C, P, V)$ that solves the minimal mutual exclusion problem and let $i \in P$ and $H \in C$. We say that i *has a remote event after* H iff there exists an i-computation M such that M does not contain Eat_i, M has a remote event, and $H \circ M \in C$.

Lemma 5: Suppose that $S = (C, P, V)$ solves the minimal mutual exclusion problem. Let $Y \subseteq P$ be a set of n processes, and let H be a Y-computation in C satisfying (C1), (C2), and (C4). Then, at least $n - 1$ processes in Y have a remote event after H.

Proof: If there are two processes that do not have a remote event after H, then we can extend H by executing those processes and violate the Exclusion requirement. A formal proof is presented in the appendix. □

The next theorem by Turán [18] will be used in subsequent lemmas.

Theorem 2 (Turán): Let $G = \langle V, E \rangle$ be an undirected multigraph,[4] where V is a set of vertices and E is a set of edges. If the average degree is d, then there exists an independent set[5] with at least $\lceil |V|/(d+1) \rceil$ vertices. □

Our next lemma provides the induction step that leads to the lower bound in Theorem 3.

Lemma 6: Let $S = (C, P, V)$ be a shared-memory system with write-contention w that solves the minimal mutual exclusion problem. Let $Y \subseteq P$ be a set of n processes, and let H be a Y-computation in C satisfying (C1), (C2), (C3), and (C4) such that each process in Y executes r remote events in H. Suppose that each event accesses at most v remote variables. Then, there exist $Z \subseteq Y$, where $|Z| = \lceil (n-1)/(2v+1)^2 vw \rceil$, and a Z-computation G in C satisfying (C1), (C2), (C3), and (C4) such that each process in Z executes $r+1$ remote events in G.

Proof: The proof strategy is as follows. We show that there exists $Z \subseteq Y$ that can execute another remote event without violating any of the conditions (C1) through (C4). We "eliminate" processes not in Z, i.e., ones that may violate some condition. Finally, we construct a Z-computation G that satisfies (C1), (C2), (C3), and (C4).

Lemma 5 implies that there exists $Y1 \subseteq Y$, where $|Y1| \geq n-1$, such that the following holds: for any $i \in Y1$, there exists an i-computation $B(i)$ such that $H \circ B(i) \in C$, $B(i)$ does not contain Eat_i, and $B(i)$ has at least one remote event. For $i \in Y1$, let $B(i) = L(i) \circ \langle [R_i, W_i, i], \ldots \rangle$ where $[R_i, W_i, i]$ is the first remote event in $B(i)$. Note that, by (P1), the following holds.

$$H \circ L(i) \circ \langle [R_i, W_i, i] \rangle \in C \tag{1}$$

We construct $Y2$, a subset of $Y1$, as follows. First, select a process $i \in Y1$. Let $X_i = \{x \mid x \in W_i.var \text{ and } x \text{ is remote to } i\}$, i.e., X_i is the set of remote variables written by the event $[R_i, W_i, i]$. By assumption, $|X_i| \leq v$. Let $Q_{X_i} = \{j \mid j \in Y1 \ \wedge \ j \neq i \ \wedge \ (W_j.var \cap X_i) \neq \{\}\}$, i.e., Q_{X_i} includes those processes other than i that write variables in X_i. Because write-contention is w, it is straightforward to use Lemma 4 to show that $|Q_{X_i}| \leq v(w-1)$. Delete i and all processes in Q_{X_i} from $Y1$, and add i to $Y2$. Repeat the above procedure until $Y1$ becomes empty. It follows, by construction, that

$$|Y2| \geq \lceil (n-1)/vw \rceil \quad . \tag{2}$$

Now, we identify any possible "information flow" between the events $\{[R_i, W_i, i] \mid i \in Y2\}$ and the events of processes in $Y2$ in H. Recall that $\{[R_i, W_i, i] \mid i \in Y2\}$ contains events that can be applied after H. We construct a graph $\langle Y2, E \rangle$ as follows. (We do not distinguish a vertex representing p from the process p when this does not cause any confusion.) Informally, an edge joining two processes represents possible information flow between the two processes. Our proof strategy is to prohibit information flow between active processes. Suppose that $x \in R_p.var \cup W_p.var$ and x is remote to p. Without loss of generality, we assume x is local to q for some $q \neq p$. Note that q may or may not be a member of $Y2$. We construct E by the following rules.

- (R1): If $q \in Y2$, then introduce an edge (p, q).
- (R2): If there is process $w \in Y2$ that writes to x in H, where $w \neq p \ \wedge \ w \neq q$, then introduce an edge (p, w). Note that, because H satisfies (C2), $q \notin Y2$ holds.

[4] A *multigraph* is a graph in which multiple edges are allowed between any two vertices. For brevity, we will henceforth use "graph" to mean an undirected multigraph.

[5] An *independent set* of a graph $G = \langle V, E \rangle$ is a subset $V' \subseteq V$ of vertices such that no edge in E is incident to two vertices in V'.

Consider the event $[R_i, W_i, i]$, where $i \in Y2$. Because (R1) and (R2) are exclusive, at most one edge is introduced for each remote variable this event accesses. Therefore, because each event accesses at most v remote variables, at most v edges are introduced by this event in total. It follows that the average degree in $\langle Y2, E\rangle$ is at most $2v$. By Theorem 2 and (2), this implies that there exists a subgraph $\langle Y3, \{\}\rangle$ of $\langle Y2, E\rangle$, where

$$|Y3| \geq \lceil (n-1)/(2v+1)vw \rceil \quad . \tag{3}$$

Without loss of generality, assume the processes are numbered so that $Y3 = \{1, 2, \ldots, |Y3|\}$. Consider the following computation.

$$H' = H_{Y3} \circ L(1) \circ L(2) \circ \cdots \circ L(|Y3|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_{|Y3|}, W_{|Y3|}, |Y3|] \rangle$$

We will use H' to construct the computation G mentioned at the beginning of the proof. In order to motivate the construction of G, we first prove that H' satisfies conditions (C2) through (C4). We consider each of these conditions as a separate case. In these cases, we make use of the fact that, because H satisfies (C2) through (C4), H_{Y3} also satisfies (C2) through (C4).

Condition $(C2)$. No $L(i)$ accesses a remote variable, and hence, $H_{Y3} \circ L(1) \circ L(2) \circ \cdots \circ L(|Y3|)$ satisfies (C2). By (R1), no $[R_i, W_i, i]$ accesses a variable that is local to another process in $Y3$. Hence, H' satisfies (C2). □

Condition $(C3)$. H_{Y3} satisfies (C2) and (C3), and each $L(i)$ consists only of local events, so $H_{Y3} \circ L(1) \circ L(2) \circ \cdots \circ L(|Y3|)$ satisfies (C3). Hence, to complete the proof that H' satisfies (C3), it suffices to prove that for each distinct i and j in $Y3$, $[R_i, W_i, i]$ does not write a variable that is written by $[R_j, W_j, j]$ or by any event of process j in H_{Y3} or $L(j)$.

By (R1), $[R_i, W_i, i]$ does not access a variable that is local to process j. Hence, $[R_i, W_i, i]$ does not write a variable that is locally written by process j in H_{Y3} or any variable that is written by j in $L(j)$. By (R2), $[R_i, W_i, i]$ does not access a variable that is remotely written by j in H. Hence, $[R_i, W_i, i]$ does not write a variable that is remotely written by j in H_{Y3}. By the definition of $Y3$ (specifically, the construction of $Y2$), the remote variables written by $[R_i, W_i, i]$ and $[R_j, W_j, j]$ are distinct. Hence, $[R_i, W_i, i]$ does not write a variable that is written by $[R_j, W_j, j]$. Hence, we conclude that H' satisfies (C3). □

Condition $(C4)$. By construction, $L(i)$ does not contain Eat_i, and $[R_i, W_i, i] \neq Eat_i$. Hence, H' satisfies (C4). □

The above reasoning leaves only condition (C1). We now show that H' may violate this condition. By (R1) and (R2), for each $j \neq i$, $[R_i, W_i, i]$ does not read a variable that is written by any event of process j in H_{Y3} or $L(j)$. Note, however, that $[R_i, W_i, i]$ may read a variable that is written by $[R_j, W_j, j]$. Such conflicts are the only way that H' may violate (C1). We now apply another graph argument in order to eliminate such conflicts among the events $\{[R_i, W_i, i] \mid i \in Y3\}$. Suppose that $x \in R_p.var$ and x is remote to p. Then, we construct a graph $\langle Y3, E'\rangle$, where the edges in E' are defined according to the following rule.

- (R3): If there is a process $w \neq p$ such that $x \in W_w.var$ and $w \in Y3$, then introduce an edge (p, w).

Because H' satisfies (C3), p introduces at most one edge for each remote variable it reads. Because each event reads at most v remote variables, p introduces at most v edges in total. Thus, by Theorem 2 and (3), there exists a subgraph $\langle Z, \{\}\rangle$ of $\langle Y3, E'\rangle$, where

$$|Z| \geq \lceil (n-1)/(2v+1)^2 vw \rceil \quad . \tag{4}$$

The set Z represents the subset of the original n processes in Y that can execute another remote event without violating any of the conditions (C1) through (C4). We show this below.

Without loss of generality, assume the processes are numbered so that $Z = \{1, 2, \ldots, |Z|\}$. The computation G we seek is defined as follows.

$$G = H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_{|Z|}, W_{|Z|}, |Z|] \rangle$$

Observe that, because H' satisfies (C2) through (C4), G also satisfies (C2) through (C4). We now show that G satisfies (C1).

Condition (C1). Because H satisfies (C1) and (C2), H_Z satisfies (C1) and (C2). Hence, because each $L(i)$ consists only of local events, $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|)$ satisfies (C1). Let p be any process in Z. To complete the proof that G satisfies (C1), it suffices to prove that no variable x in $R_p.var$ is written in G by a process other than p.

We first show that x is not written by processes other than p in $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|)$. By (R1) and (R2), $[R_p, W_p, p]$ does not access a variable that is written in H by other processes in Z. This implies that x is not written by processes other than p in H_Z. (R1) implies that $[R_p, W_p, p]$ does not access a variable that is local to another process in Z. Because each $L(i)$ consists of only local events, this implies that x is not written by processes other than p in $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|)$. Furthermore, by (R3), x is not written by $[R_j, W_j, j]$, where $j \neq p$. Hence, we conclude that G satisfies (C1). □

To complete the proof of the lemma, we need to show that G is actually a computation in C. This is established in the following claim.

Claim 1: $G \in C$.

Proof: The proof is by induction on the subsequence $\langle [R_1, W_1, 1], \ldots, [R_{|Z|}, W_{|Z|}, |Z|] \rangle$.

Induction Base. We use Lemmas 2, 3, and 4 to establish the base case. Because H satisfies (C1), by Lemma 2, $H_Z \in C$. Consider $j \in Z$. By (1) and (P1), $H \cdot L(j) \in C$. Because $j \in Z$, $H[j]H_Z$. Because H and H_Z satisfy (C2), and because $L(j)$ consists only of local events, no event in $L(j)$ accesses any variable accessed by processes other than j in H or H_Z. By Lemma 3, this implies that $H_Z \cdot L(j) \in C$.

As above, because G satisfies condition (C2), no event in $L(j)$ accesses any variable accessed by another process in $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|)$. By Lemma 4, it follows that $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \in C$.

Induction Hypothesis. Assume that

$$H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_{j-1}, W_{j-1}, j-1] \rangle \in C \quad . \tag{5}$$

Induction Step. We use (P2) to prove that $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_j, W_j, j] \rangle \in C$. Because $j \in Z$, the following holds.

$$H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_{j-1}, W_{j-1}, j-1] \rangle \ [j] \ H \circ L(j) \tag{6}$$

Consider x in $R_j.var$. Because G satisfies (C1), x is not written by a process other than j in $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_{j-1}, W_{j-1}, j-1] \rangle$. Hence, we have the following.

$$\begin{aligned} (\forall x : x \in R_j.var \quad :: \quad & value(x,\ H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], \\ & [R_2, W_2, 2], \ldots, [R_{j-1}, W_{j-1}, j-1] \rangle \ = \ value(x,\ H \circ L(j))) \end{aligned} \tag{7}$$

By (1), (5), (6), (7), and (P2), we conclude that $H_Z \circ L(1) \circ L(2) \circ \cdots \circ L(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_j, W_j, j] \rangle \in C$. □

By construction, each process in Z executes $r+1$ remote events in G. As shown above, G satisfies conditions (C1) through (C4). Hence, by (4) and Claim 1, the lemma follows. □

We now prove our first main result, Theorem 3, which for convenience is restated below. According to this result, there exists a fundamental trade-off between write-contention and time-complexity in solutions to the mutual exclusion problem. This result also shows a trade-off between the degree of atomicity and time-complexity.

Theorem 3: For any $S = (C, P, V)$ with write-contention $w > 1$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, then there exists an i-computation in C that contains $\Omega(\log_{vw} N)$ remote events but no Eat_i event.

Proof: $\langle\rangle$ is a P-computation and satisfies (C1), (C2), (C3), and (C4). By repeatedly applying Lemma 6, this implies that there exists a computation F in C that satisfies (C1) and (C4) and that contains $\Omega(\log_{((2v+1)^2vw)} N) = \Omega(\log_{vw} N)$ remote events of some process i in P. By Lemma 2, $F_i \in C$ holds, from which the theorem follows. □

Corollary 2: For any system S satisfying the conditions of Theorem 3, there exist $\Omega(N)$ processes i in P for which the conclusion of the theorem holds. □

For the minimal mutual exclusion problem, the lower bound of Theorem 3 is asymptotically tight for any values of v and w. An algorithm solving the minimal mutual exclusion problem that matches the bound of the theorem can be obtained by using an "extended" test-and-set operation that simultaneously test-and-sets v variables. We show this by first explaining how to use the extended test-and-set operation to solve $((v+1)w/2)$-process minimal mutual exclusion in $O(1)$ time. This is done by partitioning the processes into $v+1$ groups of size $w/2$. A boolean variable, whose initial value is false, is associated with each pair of groups. Thus, each group is associated with v such variables. A process applies the extended test-and-set operation to all associated variables (in a single step), and eats only if it reads false from every variable accessed. It should be clear that this algorithm solves the minimal mutual exclusion problem in $O(1)$ time. Because each variable may be accessed only by processes in two groups of size $w/2$, the access-contention (and hence write-contention) is at most w. By applying this solution within a balanced $((v+1)w/2)$-ary tree with N leaves, it is possible to solve the minimal mutual exclusion problem for N processes in $O(\log_{vw} N)$ time with access-contention w. Thus, the bound of Theorem 3 is tight.

If v is taken to be a positive constant, then we can further show that the lower bound of Theorem 3 is asymptotically tight for solutions to the mutual exclusion problem for any value of w. In particular, an algorithm by Mellor-Crummey and Scott given in [15] solves the mutual exclusion problem for w processes, in $O(1)$ time, with access-contention (and hence write-contention) w. By applying this solution within a balanced w-ary tree with N leaves, it is possible to obtain an N-process $\Theta(\log_w N)$ mutual exclusion algorithm with access-contention w.

Note that Mellor-Crummey and Scott's algorithm uses load-and-store and compare-and-swap. Even with weaker atomic operations, logarithmic behavior can be achieved. In particular, an N-process $\Theta(\log_2 N)$ mutual exclusion algorithm based on read/write atomicity has been given previously by us in [19]. This algorithm has access-contention (and hence write-contention) two.

5 Bounds for Cache-Coherent Multiprocessors

On cache-coherent shared-memory multiprocessors, the number of remote memory references may be reduced: if a process repeatedly accesses the same remote variable, then the first access may create a copy of the variable in a local cache line, with further accesses being handled locally. In this section, we count the number of distinct remote variables a process must access to solve the minimal mutual exclusion problem. A lower bound on such a count not only implies a lower bound on the number of cache misses a process causes, but also implies that these cache misses will incur global traffic.

We prove two lower bounds, which are given in the following theorems.

Theorem 4: For any $S = (C, P, V)$ with access-contention $c > 1$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, then there exists an i-computation in C containing no Eat_i event in which $\Omega(\log_{vc} N)$ distinct remote variables are accessed. □

Theorem 5: For any $S = (C, P, V)$ with write-contention $w > 1$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, then there exists an i-computation in C containing no Eat_i event in which $\Omega(\sqrt{\log_{vw} N}\,)$ distinct remote variables are accessed. □

According to Theorem 4, if the conditions of Theorem 3 are strengthened so that at most c processes can concurrently access (read *or* write) any variable, then some process accesses $\Omega(\log_{vc} N)$ distinct remote variables before eating. According to Theorem 5, if the conditions of Theorem 3 are unchanged, i.e., write-contention is w, then some process accesses $\Omega(\sqrt{\log_{vw} N}\,)$ distinct remote variables before eating.

5.1 Proof Strategy

Our proof strategy rests on the distinction between "expanding" and "nonexpanding" events. Informally, an event of a process is an expanding event if it accesses some remote variable for the first time. We can similarly categorize an event as being an expanding read or write. This is illustrated in Figure 2, which is explained below. These terms are formally defined as follows.

Definition: Consider a remote event e of a process p in a computation H. Let X be the remote variables accessed by e. If e is the first event by p in H that accesses some variable in X, then we say that e is an *expanding event* in H. If e is a read (write) event, and if e is the first event by p in H that reads (writes) some variable in X, then we say that e is an *expanding read* (*write*) *event* in H. If e is neither an expanding read nor an expanding write, then we say that e is a *nonexpanding event* in H. □

An expanding event can be an expanding read, or an expanding write, or both. Note, however, that an expanding read (write) is not necessarily an expanding event. In Figure 2, where all variables are assumed to be remote to processes P and Q, E_0, E_1, E_3, and E_4 are expanding events. Also, E_0, E_2, E_3, and E_4 are expanding reads, and E_1, E_3, and E_4 are expanding writes. Although E_2 is an expanding read, it is not an expanding event.

We count the number of expanding events in order to determine the number of distinct remote variables accessed. Observe that if a process executes r expanding events, then it accesses at least r distinct remote variables.

Because the first result of this section is based on a restriction on all concurrent accesses (rather than only concurrent writes) of the same variable, it is necessary to replace condition (C3) by the following.

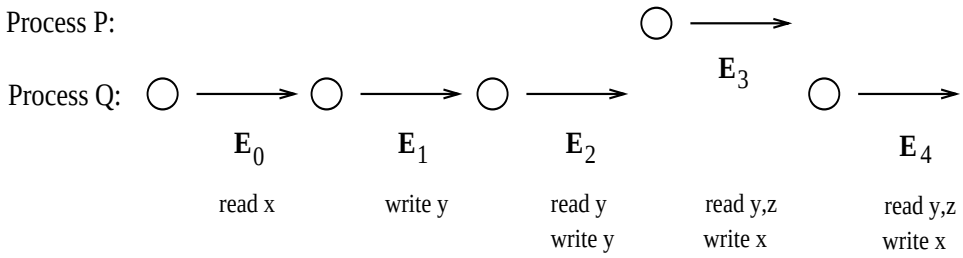

Figure 2: Process P executes event E_3, and process Q executes the other events depicted. Variables x, y, and z are remote to both processes. E_0, E_1, E_3, and E_4 are expanding events, E_0, E_2, E_3, and E_4 are expanding reads, and E_1, E_3, and E_4 are expanding writes.

- (C5) For any events $[R, W, i]$ and $[T, U, j]$ in H, if $((R.var \cup W.var) \cap (T.var \cup U.var)) \neq \{\}$, then $i = j$. Informally, each variable is accessed by at most one process in H.

The proof of Theorem 4 is based upon an inductive approach that is almost identical to that of Theorem 3. In the induction step, we assume that there are n processes, each of which executes r expanding remote operations, in a computation H that satisfies the conditions (C2), (C4), and (C5). Note that (C5) subsumes (C1) and (C3). In Lemma 7, we identify a subset of $\lceil (n-1)/(2v+1)vc \rceil$ processes that can execute a next remote operation without violating any of the conditions (C2), (C4), and (C5), and construct a computation G in which only these selected processes execute $r+1$ expanding remote operations each. This induction step provides the $\Omega(\log_{vc} N)$ bound.

In Theorem 5, we present a lower bound on the number of distinct remote variable accesses required for solving the minimal mutual exclusion problem with write-contention w. To facilitate the proof, we first define the notion of a "predecessor" event. This notion is illustrated in Figure 3, which is explained below.

Definition: Consider a computation H that contains a nonexpanding event e by process i. Let X denote the remote variables accessed by e. Let $S \equiv \{f \mid \text{for some } x \in X, f \text{ is the last event by } i \text{ in } H \text{ that accesses } x \text{ before } e\}$. Observe that $|S| \leq |X|$. The first event of S in H is called the *predecessor* of e in H. Note that any suffix of H that contains the predecessor of e contains events by i (before e) that collectively access all variables in X. □

In Figure 3, where x and y are assumed to be remote to processes P and Q, the predecessor of E_4 is E_0. To verify this, observe that E_0 is the last event of process Q that accesses x before E_4, and that E_2 is the last event of process Q that accesses y before E_4. Clearly, E_0 occurs before E_2. Note that any suffix of the given computation that contains E_0 contains events by Q (before E_4) that collectively access both x and y.

Now, we define the notion of a "critical" remote event. Such events are used in the proof of Theorem 5 to count the number of distinct remote variables a process must access in its entry section.

Definition: Consider a remote event e of a process i in a computation H. Event e is a *critical* event in H iff one of the following holds: e is an expanding write in H; e is an expanding read in H; e is a nonexpanding event and there is an expanding write by i between e and its predecessor in H. □

In Figure 3, E_0, E_1, and E_3 are critical events because they are expanding writes. E_2 is also a critical event because it is an expanding read. The nonexpanding event E_4 is a critical event because there is an expanding write E_1 between E_4 and its predecessor E_0.

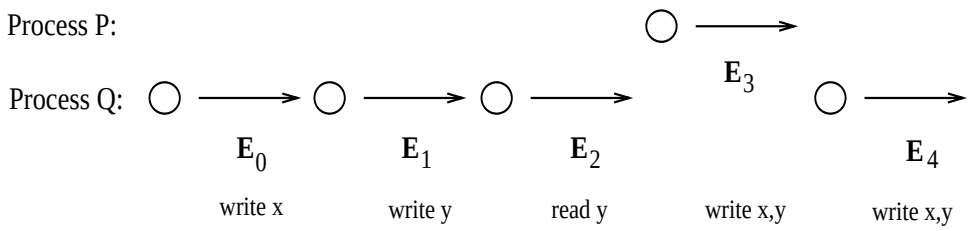

Figure 3: Process P executes E_3, and process Q executes the other events depicted. Variables x and y are remote to both processes. The predecessor of E_4 is E_0. E_0, E_1, and E_3 are critical events because they are expanding writes. E_2 is also a critical event because it is an expanding read. The nonexpanding event E_4 is a critical event because there is an expanding write E_1 between E_4 and its predecessor E_0.

In order to prove Theorem 5, we inductively construct a competition-free execution H in which some process executes $\Omega(\log_{vw} N)$ critical remote events before entering its critical section. Let D denote the number of distinct remote variables accessed in H, let W denote the number of expanding writes in H, let R denote the number of expanding reads in H, and let E denote the number of nonexpanding critical remote events in H. We show that $D \geq \mathbf{max}(W, R, E/W)$ holds, which implies that Theorem 5 holds.

5.2 Proofs

Our next lemma provides the induction step that leads to the lower bound in Theorem 4.

Lemma 7: Let $S = (C, P, V)$ be a shared-memory system with access-contention c that solves the minimal mutual exclusion problem. Let $Y \subseteq P$ be a set of n processes, and let H be a Y-computation in C satisfying (C2), (C4), and (C5) such that each process in Y executes r expanding remote events in H. Suppose that each event accesses at most v remote variables. Then, there exist $Z \subseteq Y$, where $|Z| = \lceil (n-1)/(2v+1)vc \rceil$, and a Z-computation G in C satisfying (C2), (C4), and (C5) such that each process in Z executes $r+1$ expanding remote events in G.

Proof: The proof strategy is as follows. We show that there exists $Z \subseteq Y$ that can execute another remote event without violating any of the conditions (C2), (C4), or (C5). We eliminate processes not in Z, i.e., ones that may violate some condition. Finally, we construct a Z-computation G that satisfies (C2), (C4), and (C5).

Because H satisfies (C5), it is possible to prove a result similar to Lemma 5 showing that there exists $Y1 \subseteq Y$, where $|Y1| \geq n-1$, such that the following holds: for any $i \in Y1$, there exists an i-computation $B(i)$ such that $H \circ B(i) \in C$, $B(i)$ does not contain Eat_i, and $B(i)$ has at least one expanding remote event. (If there are two processes that do not have an expanding remote event after H, then the Exclusion requirement can be violated; note that (C5) implies that these processes do not access any common variable in their entry sections.) For $i \in Y1$, let $B(i) = F(i) \circ \langle [R_i, W_i, i], \ldots \rangle$ where $[R_i, W_i, i]$ is the first expanding remote event in $B(i)$.

We construct $Y2$, a subset of $Y1$, as follows. First, select a process $i \in Y1$. Let $X_i = \{x \mid x \in R_i.var \cup W_i.var$ and x is remote to $i\}$, i.e., X_i is the set of remote variables accessed by the event $[R_i, W_i, i]$. By assumption, $|X_i| \leq v$. Let $Q_{X_i} = \{j \mid j \in Y1 \wedge j \neq i \wedge (R_j.var \cup W_j.var) \cap X_i \neq \{\}\}$, i.e., Q_{X_i} includes those processes other than i that access variables in X_i. Because access-contention is c, it is straightforward to use Lemma 4 to show that $|Q_{X_i}| \leq v(c-1)$. Delete i and all processes in Q_{X_i} from $Y1$, and add i to $Y2$.

Repeat the above procedure until $Y1$ is empty. By construction,

$$|Y2| \geq \lceil (n-1)/vc \rceil \quad . \tag{8}$$

Observe that if $i \in Y2$, $j \in Y2$, and $i \neq j$ hold, then $[R_i, W_i, i]$ and $[R_j, W_j, j]$ do not access a common variable. Thus, there is no information flow among $\{[R_i, W_i, i] \mid i \in Y2\}$. Now, we identify any possible information flow between $\{[R_i, W_i, i] \mid i \in Y2\}$ and the events in H of processes in $Y2$. Recall that $\{[R_i, W_i, i] \mid i \in Y2\}$ contains events that can be applied after H.

Suppose that $x \in R_p.var \cup W_p.var$ and x is remote to p. Without loss of generality, we assume x is local to q for some $q \neq p$. Note that q may or may not be a member of $Y2$. We construct E by the following rules.

- (R1): If $q \in Y2$, then introduce an edge (p, q).
- (R2): If there is process $w \in Y2$ that accesses x in H, where $w \neq p \ \wedge \ w \neq q$, then introduce an edge (p, w). Note that, because H satisfies (C2), $q \notin Y2$ holds.

Because (R1) and (R2) are exclusive, at most one edge is introduced for each remote variable an event accesses. Because each event accesses at most v remote variables, at most v edges are introduced for each remote event. We eliminate all edges by applying Theorem 2. The number of vertices is reduced by a factor of $1/(2v+1)$. These remaining vertices represent the subset of processes selected from the original n processes in Y. We use Z to denote this subset of Y. Note that, for any $i \in Z$, by Rule (R1), $[R_i, W_i, i]$ does not access any variable that is local to another process in Z, and by Rule (R2), it does not access a variable that is accessed in H by other processes in Z.

Without loss of generality, assume the processes are numbered so that $Z = \{1, 2, \ldots, |Z|\}$. By (8), we have $|Z| \geq \lceil (n-1)/(2v+1)vc \rceil$. The computation G we seek is defined as follows.

$$G = H_Z \circ F(1) \circ F(2) \circ \cdots \circ F(|Z|) \circ \langle [R_1, W_1, 1], [R_2, W_2, 2], \ldots, [R_{|Z|}, W_{|Z|}, |Z|] \rangle$$

Because H satisfies (C5), H also satisfies (C1). Thus, by Lemma 2, $H_Z \in C$. It is straightforward to use this fact to prove that $G \in C$.

By construction, each process in Z executes $r+1$ expanding remote events in G. To complete the proof of Lemma 7, it suffices to prove that G satisfies (C2), (C4), and (C5). We consider each of these conditions as a separate case. In these cases, we make use of the fact that, because H satisfies (C2), (C4), and (C5), H_Z also satisfies (C2), (C4), and (C5).

Condition $(C2)$. Because H_Z satisfies (C2), and because no $F(i)$ contains an expanding remote event, $H_Z \circ F(1) \circ F(2) \circ \cdots \circ F(|Z|)$ satisfies (C2). By (R1), no $[R_i, W_i, i]$ accesses a variable that is local to another process in Z. Hence, G satisfies (C2).

Condition $(C4)$. By construction, $F(i)$ does not contain Eat_i, and $[R_i, W_i, i] \neq Eat_i$. Hence, G satisfies (C4).

Condition $(C5)$. H_Z satisfies (C2) and (C5), and each $F(i)$ does not contain an expanding remote event, so $H_Z \circ F(1) \circ F(2) \circ \cdots \circ F(|Z|)$ satisfies (C5). Hence, to complete the proof that G satisfies (C5), it suffices to prove that for each distinct i and j in Z, $[R_i, W_i, i]$ does not access a variable that is accessed by $[R_j, W_j, j]$ or by any event of process j in H_Z or $F(j)$.

Because $F(j)$ contains no expanding remote event, any variable accessed by process j in $F(j)$ is either local to j or accessed remotely by j in H. By (R1), $[R_i, W_i, i]$ does not access a variable that is local to process j. By (R2), $[R_i, W_i, i]$ does not access a variable that is remotely accessed by j in H. Hence, $[R_i, W_i, i]$ does not access a variable that is remotely accessed by j in H_Z. By the definition of Z (specifically, the

construction of $Y2$), the remote variables accessed by $[R_i, W_i, i]$ and $[R_j, W_j, j]$ are distinct. Hence, $[R_i, W_i, i]$ does not access a variable that is accessed by $[R_j, W_j, j]$. Hence, we conclude that G satisfies (C5).

This concludes the proof of Lemma 7. □

We now prove Theorem 4, which is restated below.

Theorem 4: For any $S = (C, P, V)$ with access-contention $c > 1$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, then there exists an i-computation in C containing no Eat_i event in which $\Omega(\log_{vc} N)$ distinct remote variables are accessed.

Proof: $\langle\rangle$ is a P-computation and satisfies (C2), (C4), and (C5). By repeatedly applying Lemma 7, this implies that there exists a computation F in C that satisfies (C4) and (C5) (and hence C(1)) and that contains $\Omega(\log_{vc} N)$ expanding remote events of some process i in P. By Lemma 2, $F_i \in C$ holds, from which the theorem follows. □

Corollary 3: For any system S satisfying the conditions of Theorem 4, there exist $\Omega(N)$ processes i in P for which the conclusion of the theorem holds. □

Observe that the minimal mutual exclusion algorithm based on the extended test-and-set operation mentioned after Corollary 2 has time complexity $\Theta(\log_{vc} N)$. This implies that the lower bound of Theorem 4 is asymptotically tight for the minimal mutual exclusion problem for any values of v and c.

Also, note that the tree-based mutual exclusion algorithms mentioned after Corollary 2 have time complexity $\Theta(\log_c N)$. Thus, for the mutual exclusion problem, the lower bound of Theorem 4 is asymptotically tight for any value of c, if v is taken to be a positive constant.

In the remainder of this section, we prove a lower bound on the number of distinct remote variable accesses required for solving the minimal mutual exclusion problem with write-contention w.

The next lemma is a variation of Lemma 5 that deals with critical remote events. Suppose that $S = (C, P, V)$ solves the minimal mutual exclusion problem and let $i \in P$ and $H \in C$. Corresponding to the definition prior to Lemma 5, we say that *i has a critical remote event after H* iff the following holds: there exists a remote event e of process i, and an i-computation L consisting of local events, each differing from Eat_i, such that $H \circ L \circ e \in C$ holds, where e is critical in $H \circ L \circ e$.

Lemma 8: Suppose that $S = (C, P, V)$ solves the minimal mutual exclusion problem. Let $Z \subseteq P$ be a set of n processes, and let H be a Z-computation in C satisfying (C1), (C2), (C3), and (C4). Then, there exists a Z-computation H' in C satisfying (C1), (C2), (C3), and (C4) such that H' contains all events contained in H and at least $n-1$ processes in Z have a critical remote event after H'.

Proof: Lemma 5 implies that at least $n-1$ of the processes in Z have a remote event after H. If all $n-1$ of these remote events are critical after H, then the conclusion of the lemma holds. So, assume that one of these events is noncritical after H. Then, there exists a process p in Z and a computation

$$H \circ L \circ \langle e \rangle \in C \quad , \tag{9}$$

where L is a p-computation consisting of only local events, and e is a noncritical remote event of p in $H \circ L \circ \langle e \rangle$. Because e is noncritical, we have

$$H \circ L \circ \langle e \rangle \;=\; X \circ \langle f \rangle \circ Y \circ L \circ \langle e \rangle \quad , \tag{10}$$

where f is the predecessor of e in $H \circ L \circ \langle e \rangle$, and Y contains no expanding write by p.

Let $G \equiv X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ (Y - Y_p)$. Observe that G is a Z-computation. In the following paragraphs, we show that $G \in C$ holds, and then show that G satisfies (C1) through (C4). Observe that G contains all events contained in H and more remote events than H. By the Progress requirement, this implies that we can apply this argument only a finite number of times, i.e., if we repeatedly apply Lemma 5 and construct a new computation in the manner in which G is constructed, then we eventually obtain a computation H' such that applying Lemma 5 yields $n-1$ processes in Z, each of which has a *critical* remote event after H'. By our construction, H' is a computation in C, satisfies (C1) through (C4), and contains all events contained in H.

To begin the construction of G, note that, because $H \in C$, (10) implies $H = X \circ \langle f \rangle \circ Y \in C$. Furthermore, by assumption, H satisfies (C1). Hence, by Lemma 2, we have the following.

$$X \circ \langle f \rangle \circ Y_p \in C \tag{11}$$

We now apply Lemma 3 to prove that $X \circ \langle f \rangle \circ Y_p \circ L \in C$ holds. In applying Lemma 3, we use the following assertions.

$$X \circ \langle f \rangle \circ Y \ [p] \ X \circ \langle f \rangle \circ Y_p \tag{12}$$

$$X \circ \langle f \rangle \circ Y \circ L \in C \tag{13}$$

(12) holds by definition, and (13) follows from (9), (10), and (P1).

Because H satisfies (C2), by (10), both $X \circ \langle f \rangle \circ Y$ and $X \circ \langle f \rangle \circ Y_p$ also satisfy (C2). Also, recall that L is a p-computation consisting of local events and that p is active in H. Thus, no event in L accesses a variable that is written by processes other than p in either $X \circ \langle f \rangle \circ Y$ or $X \circ \langle f \rangle \circ Y_p$. Hence, by (11), (12), (13), and Lemma 3, the following holds.

$$X \circ \langle f \rangle \circ Y_p \circ L \in C \tag{14}$$

The next step in the proof is to use (P2) to establish that $X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle$ is in C, where e is as defined at the beginning of the proof. Let $e = [R_p, W_p, p]$. The following assertion follows from (10).

$$X \circ \langle f \rangle \circ Y_p \circ L \ [p] \ H \circ L \tag{15}$$

Because $H \circ L \circ \langle e \rangle$ satisfies (C1), for all $x \in R_p.var$, the following holds.

$$value(x, \ X \circ \langle f \rangle \circ Y_p \circ L) = value(x, \ H \circ L) \tag{16}$$

By (9), (14), (15), (16), and (P2), it follows that

$$X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \in C \quad . \tag{17}$$

We now show that G is in C by establishing the following claim.

Claim 2: $X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ (Y - Y_p) \in C$.

Proof: Let $(Y - Y_p) = \langle e_0, e_1, \ldots, e_m \rangle$. The proof is by induction on $|Y - Y_p|$.

Induction Base. By (17), $X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \in C$ holds.

Induction Hypothesis. Suppose that $X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle \in C$ holds.

Induction Step. We prove that $X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ \langle e_0, e_1, \ldots, e_m \rangle \in C$ holds. Without loss of generality, assume that $Y = Q \circ \langle e_m \rangle \circ T$. Then, by (P1), (13) implies that the following holds.

$$X \circ \langle f \rangle \circ Q \circ \langle e_m \rangle \in C \tag{18}$$

Let $e_m = [R, W, i]$ for some $i \neq p$. Because $i \neq p$, the following holds.

$$X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle \ [i] \ X \circ \langle f \rangle \circ Q \tag{19}$$

Let $x \in R.var$. We now show that x is not written by any event in Y_p, L, or $\langle e \rangle$. Suppose that x is written by e or by an event in Y_p. e is noncritical and hence is not an expanding write. Also, Y_p does not contain any expanding write by p. Thus, by (10), x is also written by p in $X \circ \langle f \rangle$. Because $i \neq p$, this implies that H does not satisfy (C1), which is a contradiction.

Now, suppose that x is written by an event in L. Recall that L consists only of local events of p. Thus, event $e_m = [R, W, i]$, which is in H, reads a local variable of process $p \neq i$. Because p is active in H, this implies that H does not satisfy (C2), which is a contradiction. Thus, we conclude that x is not written by any event in Y_p, L, or $\langle e \rangle$. This implies that, for each x in $R.var$, $writer(x,\ X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle) = writer(x,\ X \circ \langle f \rangle \circ Q)$ holds. By Lemma 1, this implies that the following holds.

$$value(x,\ X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle) = value(x,\ X \circ \langle f \rangle \circ Q) \tag{20}$$

By the induction hypothesis, (18), (19), (20), and (P2), $X \circ \langle f \rangle \circ Y_p \circ L \circ \langle e \rangle \circ \langle e_0, e_1, \ldots, e_m \rangle \in C$. □

Having shown that G is in C, we now show that G satisfies (C1) through (C4). Observe that the events in $L \circ \langle e \rangle$ are the only events in G that are not in H. L consists only of local events of process p, none of which are Eat_p. Also, e, being a noncritical remote event, does not access any remote variable that p does not access in H. Hence, because H satisfies (C2) through (C4), it follows that G also satisfies (C2) through (C4).

As for (C1), our proof obligation is to show that no event in G reads a variable previously written by another process. Because H satisfies (C1), by (10), no event in $X \circ \langle f \rangle \circ Y_p$ reads a variable previously written by another process.

Now, consider events in $L \circ \langle e \rangle \circ (Y - Y_p)$. Observe that L consists only of local events of p, p is active in H, and H satisfies (C2). Hence, no event in L reads a variable that is previously written by another process in G.

If e reads a variable that is previously written by another process in G, then that variable is written in X, because $\langle f \rangle \circ Y_p \circ L$ consists of events by p. If e reads a variable that is written by another process in X, then, by the definition of a predecessor, there exists an event in $\langle f \rangle \circ Y_p$ that accesses that same variable. However, this implies that H violates (C1) or (C3), which is a contradiction.

Finally, because H satisfies (C1), no event in $Y - Y_p$ reads a variable written by another process in $X \circ \langle f \rangle \circ Y_p$. By the reasoning at the end of the proof of Claim 2, no event in $Y - Y_p$ reads a variable that is written by p in $Y_p \circ L \circ \langle e \rangle$. We conclude that G satisfies (C1).

We have shown that if some process in Z has a next remote event after H that is noncritical, then there exists a Z-computation in C satisfying (C1), (C2), (C3), and (C4) that contains more remote events than H. As noted previously, if this argument could be applied repeatedly, then it would be possible to construct a computation in C that violates the Progress requirement. This proves the lemma. □

The next lemma is a stronger version of Lemma 6 in which only critical remote events are counted rather than all remote events.

Lemma 9: Let $S = (C, P, V)$ be a shared-memory system with write-contention w that solves the minimal mutual exclusion problem. Let $Y \subseteq P$ be a set of n processes, and let H be a Y-computation in C satisfying (C1), (C2), (C3), and (C4) such that each process in Y executes r critical remote events in H. Suppose that each event accesses at most v remote variables. Then, there exist $Z \subseteq Y$, where $|Z| = \lceil (n-1)/(2v+1)^2 vw \rceil$, and a Z-computation G in C satisfying (C1), (C2), (C3), and (C4) such that each process in Z executes $r+1$ critical remote events in G.

Proof: Lemma 8 implies that there exists $Y1 \subseteq Y$, where $|Y1| \geq n-1$, such that the following holds: for any $i \in Y1$, there exists an i-computation $L(i)$ consisting of local events, such that $H \circ L(i) \circ [R_i, W_i, i] \in C$, where $[R_i, W_i, i]$ is a critical remote event in $H \circ L(i) \circ [R_i, W_i, i]$. The rest of the proof is identical to that of Lemma 6. □

We now prove Theorem 5, which is restated below. According to this theorem, among the $\Omega(\log_{vw} N)$ remote events mentioned in Theorem 3, $\Omega(\sqrt{\log_{vw} N})$ distinct remote variables are accessed.

Theorem 5: For any $S = (C, P, V)$ with write-contention $w > 1$ that solves the minimal mutual exclusion problem, if each event accesses at most v remote variables, then there exists an i-computation in C containing no Eat_i event in which $\Omega(\sqrt{\log_{vw} N})$ distinct remote variables are accessed.

Proof: $\langle\rangle$ is a P-computation and satisfies (C1), (C2), (C3), and (C4). By repeatedly applying Lemma 9, this implies that there exists a computation F in C that satisfies (C1) and (C4) and that contains $\Omega(\log_{vw} N)$ critical remote events of some process i in P. By Lemma 2, $F_i \in C$. Let W denote the number of expanding writes in F_i, let R denote the number of expanding reads in F_i, and let E denote the number of nonexpanding critical remote events in F_i. Then, because F_i contains $\Omega(\log_{vw} N)$ critical remote events,

$$(W + R + E) \geq c \cdot \log_{vw} N \tag{21}$$

holds for some positive constant c. Let D denote the number of distinct remote variables accessed in F_i. Observe that D is at least as big as W and R. Also, D is at least as big as the number of distinct remote variables accessed by events in E. The following claim provides an upper bound on the number of events in E.

> **Claim 3:** There are at most D nonexpanding critical events between two successive expanding writes in F_i.
>
> **Proof:** Let x and y denote two successive expanding writes in F_i, and let $F_i = X \circ \langle x \rangle \circ Y \circ \langle y \rangle \circ Z$. By assumption, Y does not contain an expanding write. Let $e_0, e_1, \ldots, e_m$ denote the nonexpanding critical events in Y. By the definition of a critical event, their predecessors in F_i appear in X. We claim that each e_j, where $1 \leq j \leq m$, accesses a remote variable that is not accessed in $e_0, \ldots, e_{j-1}$. Otherwise, the predecessor of e_j in F_i is not an event in X, which is a contradiction. Because e_0 accesses at least one remote variable, $e_0, e_1, \ldots, e_m$ access at least $m+1$ distinct remote variables. Thus, $m < D$ holds, which proves the claim. □

By Claim 3, at most D nonexpanding critical events may occur between an expanding write and the next expanding write (if any). In addition, by the definition of a critical event, no nonexpanding critical remote

events may exist before the first expanding write. Thus, we have at most D nonexpanding critical remote events per expanding write, i.e., $E \leq DW$. Because $D \geq W$ and $D \geq R$ hold, this implies that

$$D \geq \mathbf{max}(W, R, E/W) \quad . \tag{22}$$

We now show that $D \geq m \cdot \sqrt{\log_{vw} N}$ for some positive constant m. Assume, to the contrary, that $D < m \cdot \sqrt{\log_{vw} N}$. Then, by (22), we have $W < m \cdot \sqrt{\log_{vw} N}$ and $R < m \cdot \sqrt{\log_{vw} N}$. By (21), this implies that

$$\frac{E}{W} > \frac{c \cdot \log_{vw} N - 2m \cdot \sqrt{\log_{vw} N}}{m \cdot \sqrt{\log_{vw} N}} \quad .$$

By (22), this inequality implies that $D \geq s \cdot \sqrt{\log_{vw} N}$ for some positive constant s. □

Corollary 4: For any system S satisfying the conditions of Theorem 5, there exist $\Omega(N)$ processes i in P for which the conclusion of the theorem holds. □

6 Concluding Remarks

In this paper, we have shown that, for any N-process minimal mutual exclusion algorithm, if write-contention is w, and if each atomic operation accesses at most v remote variables, then there exists an execution involving only one process in which that process executes $\Omega(\log_{vw} N)$ remote operations for entry into its critical section. We have also shown that, among these operations, $\Omega(\sqrt{\log_{vw} N})$ distinct remote variables are accessed. For algorithms with access-contention c, we have shown that the latter bound can be improved to $\Omega(\log_{vc} N)$.

These time bounds establish that trade-offs exist both between time complexity and write- and access-contention, and between time complexity and atomicity. Because any algorithm that solves the leader election or mutual exclusion problems also solves the minimal mutual exclusion problem, these trade-offs apply to these problems as well. It is interesting to note that our bounds also apply when using other means of measuring the time complexity of busy-waiting. For example, a spin-loop of a process might be counted as one time unit. Because our bounds are obtained in the absence of competition, they still hold for this model.

Although the time bounds we establish are oriented towards programs that busy-wait, they also have implications regarding mutual exclusion mechanisms that are based on blocking. In particular, while blocking can be used to synchronize multiple processes on a single processor, busy-waiting is still fundamental for synchronization across processors [13]. Our bounds imply that tradeoffs exist between contention and time complexity and between atomicity and time complexity in any multiprocessor setting, even if blocking is used for synchronization within a processor.

For wait-free algorithms, Herlihy has characterized synchronization primitives by consensus number [9]. Such a characterization is not applicable when waiting is introduced. One way of determining the power of synchronization primitives in this case is to compare the time complexity of mutual exclusion using such primitives. For instance, it is possible to solve the mutual exclusion problem with $O(1)$ time complexity using load-and-store or fetch-and-add, while the best-known upper bound for read/write algorithms is $O(\log_2 N)$ [19]. If a lower-bound result could be proved showing that this gap is fundamental, then this would establish that reads and writes are weaker than read-modify-writes from a performance standpoint. This would provide contrasting evidence to Herlihy's hierarchy, from which it follows that reads and writes are weaker than read-modify-writes from a *resiliency* standpoint. It is interesting to note that there exist read/write mutual exclusion algorithms with write-contention N that have $O(1)$ time complexity in the absence of

competition [1, 12, 19]. Thus, establishing the above-mentioned lower bound for read/write algorithms will require proof techniques that differ from those given in this paper.

We do not know whether the bound given in Theorem 5 is tight. We conjecture that this bound can be improved to $\Omega(\log_{vw} N)$, which has a matching algorithm when v is taken to be a constant [19].

One may be interested in determining the effect of contention on space requirements. It is quite easy to show that solving the minimal mutual exclusion problem with write-contention w requires at least N/w variables. In particular, it can be shown that every process writes a variable before eating. So, consider the computation in which every process is enabled to perform its first write. Because write-contention is w, the total number of variables enabled to be written is $\Omega(N/w)$. It can be shown that this bound is tight; it is possible to obtain a deadlock-free solution to mutual exclusion with write-contention w by arranging test-and-set variables in a balanced w-ary tree with $\lceil N/w \rceil$ leaves.

In conclusion, it is our belief that the most important contribution of this paper is to show that meaningful time bounds can be established for concurrent programming problems for which busy-waiting is inherent. We hope that our work will spark new work on time complexity results for such problems.

Acknowledgements: We would like to thank Gadi Taubenfeld for prompting us to consider the bounds for cache-coherence presented in Section 5. We would like to thank Faith Fich and Samir Khuller for informing us of Turán's theorem. By using this theorem, we were able to obtain slightly better bounds than we originally reported in [20]. We would also like to thank Sanglyul Min for his helpful comments on an earlier draft of this paper.

Appendix: Additional Proofs

In this section, we give full proofs of Lemmas 2, 3, 4, and 5.

Proof of Lemma 2: As in the statement of the lemma, assume that $G \circ H$ is a computation in C satisfying (C1), and $Y \subseteq P$. We prove that $G \circ H_Y \in C$ by induction on the length of H_Y.

Induction Base. Because $G \circ H \in C$ holds, by (P1), $G \in C$ holds.

Induction Hypothesis. Suppose that Lemma 2 holds for H_Y if $|H_Y| = m$.

Induction Step. We now consider H_Y of length $m+1$. Let $H_Y = \langle e_0, e_1, \ldots, e_{m-1}, e_m \rangle$. Let $H = H' \circ \langle e_m \rangle \circ H''$.

By (P1), $G \circ H' \in C$. Observe that $G \circ H'_Y = G \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle$. Hence, by the induction hypothesis, $G \circ H'_Y \in C$. Next, we prove $G \circ H_Y \in C$ by considering two cases. Let $e_m = [R, W, i]$ for some $i \in Y$.

Because $G \circ H' \circ \langle e_m \rangle$ is a prefix of $G \circ H$, by (P1), $G \circ H' \circ \langle e_m \rangle \in C$. Note that $G \circ H' \,[Y]\, G \circ H'_Y$. Thus, to prove that $G \circ H_Y \in C$, it suffices to prove that, for any $x \in R.var$, $value(x, G \circ H') = value(x, G \circ H'_Y)$. In particular, if the latter holds, then (P2) implies that $G \circ H_Y = G \circ H'_Y \circ \langle e_m \rangle \in C$ also holds. If $R = \{\}$, then this remaining proof obligation is vacuous, so in the remainder of the proof, assume that $R \neq \{\}$.

We consider two cases according to whether x is written in $G \circ H'$. If $writer(x, G \circ H') = \perp$, then $writer(x, G \circ H'_Y) = \perp$, and by Lemma 1, $value(x, G \circ H') = value(x, G \circ H'_Y)$. If $writer(x, G \circ H') = [L, U, j]$, then because $G \circ H$ satisfies (C1), $j = i$. It follows that $writer(x, G \circ H'_Y) = [L, U, j]$, and by Lemma 1, $value(x, G \circ H') = value(x, G \circ H'_Y)$. This concludes the proof of Lemma 2. □

Proof of Lemma 3: As in the statement of the lemma, assume the following: (i) F is an i-computation; (ii) no event in F accesses a variable that is written by processes other than i in either G or H; (iii) $H \in C$;

(iv) $G[i]H$; and (v) $G \circ F \in C$. We prove that $H \circ F \in C$ by induction on the length of F.

Induction Base. If $|F| = 0$, then $H \circ F = H$. By assumption (iii), $H \in C$ holds.

Induction Hypothesis. Suppose that Lemma 3 holds when $|F| = m$.

Induction Step. We now consider F of length $m+1$. Let $F = \langle e_0, e_1, \ldots, e_m \rangle$. We use (P2) to prove that $H \circ \langle e_0, e_1, \ldots, e_m \rangle \in C$. By assumption (v),

$$G \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle \circ \langle e_m \rangle \in C \quad . \tag{23}$$

By (P1), (23) implies that $G \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle \in C$ holds. Thus, by the induction hypothesis, we have the following.

$$H \circ \langle e_0, e_1, \ldots, e_{m-1} \rangle \in C \tag{24}$$

By assumption (iv), $G[i]H$ holds, so the following holds.

$$G \circ \langle e_0, e_1, \ldots e_{m-1} \rangle \ [i] \ H \circ \langle e_0, e_1, \ldots e_{m-1} \rangle \tag{25}$$

Let $e_m = [R, W, i]$. By assumption (ii), $[R, W, i]$ does not access a variable that is written by processes other than i in either G or H. Thus, each x in $R.var$ is not written by other processes in either G or H. Thus, $G[i]H$ implies that $writer(x, G) = writer(x, H)$, which implies that $writer(x, G \circ \langle e_0, e_1, \ldots e_{m-1} \rangle) = writer(x, H \circ \langle e_0, e_1, \ldots e_{m-1} \rangle)$. By Lemma 1, this implies that the following holds.

$$value(x, G \circ \langle e_0, e_1, \ldots e_{m-1} \rangle) = value(x, H \circ \langle e_0, e_1, \ldots e_{m-1} \rangle) \tag{26}$$

Thus, by (23), (24), (25), (26), and (P2), we have $H \circ \langle e_0, e_1, \ldots, e_m \rangle \in C$. □

Proof of Lemma 4: Let Q, H, and $L(j)$ be defined as in the statement of the lemma. In particular, we have the following: (i) $L(j)$ is a j-computation; (ii) $H \circ L(j) \in C$; and (iii) no event in $L(j)$ accesses any variable that is accessed by other processes in $H \circ L(1) \circ L(2) \circ \cdots \circ L(|Q|)$. We prove that $H \circ L(1) \circ L(2) \circ \cdots \circ L(|Q|) \in C$ by induction on $|Q|$.

Induction Base. By (P1) and assumption (ii), $H \in C$.

Induction Hypothesis. Assume that $H \circ L(1) \circ L(2) \circ \cdots \circ L(j-1) \in C$, where $1 \leq j \leq |Q|$.

Induction Step. We use Lemma 3 to prove that $H \circ L(1) \circ L(2) \circ \cdots \circ L(j) \in C$. By assumption (i),

$$H \ [j] \ H \circ L(1) \circ L(2) \circ \cdots \circ L(j-1) \quad . \tag{27}$$

By (27), the induction hypothesis, and assumptions (i), (ii), and (ii), Lemma 3 implies that $H \circ L(1) \circ L(2) \circ \cdots \circ L(j) \in C$ holds. □

Proof of Lemma 5: Let H and Y be as defined in the statement of the lemma, i.e., $Y \subseteq P$, $|Y| = n$, and H is a Y-computation in C satisfying (C1), (C2), and (C4). We show that at least $n-1$ processes in Y have a remote event after H.

Assume to the contrary that $\{i, j\} \subseteq Y$ have no remote event after H. Because H satisfies (C1), by Lemma 2, $H_i \in C$. Also, because H satisfies (C4), H_i satisfies (C4). Hence, because S satisfies the Progress requirement, there exists an i-computation G such that $H_i \circ G \circ \langle Eat_i \rangle \in C$, and G does not contain Eat_i. Similarly, there exists a j-computation G' such that $H_j \circ G' \circ \langle Eat_j \rangle \in C$, and G' does not contain Eat_j. We consider three cases.

Case 1. G contains a remote event. Let $G = F \circ \langle [R, W, i], \ldots \rangle$, where $[R, W, i]$ is the first remote event in G. We prove that i has a remote event after H, which is a contradiction to our assumption. In particular, we use (P3) to prove that $H \circ F \circ \langle [R', W', i] \rangle \in C$, where $R'.var = R.var$ and $W'.var = W.var$. Because $H_i \circ G \circ \langle Eat_i \rangle \in C$ holds, by (P1), we have the following.

$$H_i \circ F \circ \langle [R, W, i] \rangle \in C \tag{28}$$

We now use Lemma 3 to prove that $H \circ F \in C$. The following assertions are used in applying Lemma 3.

$$H \in C \tag{29}$$

$$H_i \ [i] \ H \tag{30}$$

$$H_i \circ F \in C \tag{31}$$

(29) holds by the definition of H, (30) holds by the definition of $[i]$, and (31) follows from (28) and (P1). Observe that F is an i-computation consisting of local events. Thus, because i is active in H and H_i, and because both H and H_i satisfy (C2), no event in F accesses a variable that is written by processes other than i in either H or H_i. Hence, by (29), (30), (31), and Lemma 3, the following holds.

$$H \circ F \in C \tag{32}$$

Observe that (30) implies that the following holds.

$$H \circ F \ [i] \ H_i \circ F \tag{33}$$

By (28), (32), (33), and (P3), $H \circ F \circ \langle [R', W', i] \rangle \in C$, where $R'.var = R.var$ and $W'.var = W.var$. Because $H \circ F \circ \langle [R', W', i] \rangle \in C$, i has a remote event after H, which is a contradiction.

Case 2. G' contains a remote event. We can prove that j has a remote event after H. The proof is similar to that of Case 1, and hence is omitted.

Case 3. G and G' do not contain any remote event. We prove that S does not solve the minimal mutual exclusion problem.

We first use Lemma 3 to prove that $H \circ G \circ \langle Eat_i \rangle \in C$ holds. By assumption, we have the following.

$$H_i \circ G \circ \langle Eat_i \rangle \in C \tag{34}$$

Observe that $G \circ \langle Eat_i \rangle$ is an i-computation consisting of local events. Thus, because i is active in H and H_i, and because H and H_i both satisfy (C2), no event in $G \circ \langle Eat_i \rangle$ accesses a variable that is written by processes other than i in either H or H_i. Hence, by (29), (30), (34), and Lemma 3, $H \circ G \circ \langle Eat_i \rangle \in C$. Similarly, $H \circ G' \circ \langle Eat_j \rangle \in C$.

Let $F = H \circ G \circ \langle Eat_i \rangle \circ G' \circ \langle Eat_j \rangle$. It is straightforward to use Lemma 4 to prove that $F \in C$. (Let $L(1) = G \circ \langle Eat_i \rangle$ and let $L(2) = G' \circ \langle Eat_j \rangle$.) Note that $value(i.dine, F) = eat \ \wedge \ value(j.dine, F) = eat$ holds, which implies that S does not solve the minimal mutual exclusion problem. □

References

[1] R. Alur and G. Taubenfeld, "Results about Fast Mutual Exclusion", *Proceedings of the Thirteenth IEEE Real-Time Systems Symposium*, December, 1992, pp. 12-21.

[2] J. Anderson, "A Fine-Grained Solution to the Mutual Exclusion Problem", *Acta Informatica*, Vol. 30, No. 3, 1993, pp. 249-265.

[3] T. Anderson, "The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, January, 1990, pp. 6-16.

[4] J. Burns and N. Lynch, "Bounds on Shared Memory for Mutual Exclusion", *Information and Computation*, Vol. 107, 1993, pp. 171-184.

[5] K. Chandy and J. Misra, "How Processes Learn", *Distributed Computing*, Vol. 1, No. 1, 1986, pp. 40-52.

[6] E. Dijkstra, "Solution of a Problem in Concurrent Programming Control", *Communications of the ACM*, Vol. 8, No. 9, 1965, pp. 569.

[7] C. Dwork, M. Herlihy, and O. Waarts, "Contention in Shared Memory Algorithms", *Proceedings of the 25th ACM Symposium on Theory of Computing*, May, 1993, pp. 174-183.

[8] G. Graunke and S. Thakkar, "Synchronization Algorithms for Shared-Memory Multiprocessors", *IEEE Computer*, Vol. 23, No. 6, June 1990, pp. 60-69.

[9] M. Herlihy, "Wait-Free Synchronization", *ACM Transactions on Programming Languages and Systems*, Vol. 13, No. 1, 1991, pp. 124-149.

[10] M. Herlihy, B-H. Lim, and N. Shavit, "Low Contention Load Balancing on Large-Scale Multiprocessors", *Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures*, July, 1992, pp. 219-227.

[11] M. Herlihy, N. Shavit, and O. Waarts, "Low Contention Linearizable Counting", *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, October, 1991, pp. 526-535.

[12] L. Lamport, "A Fast Mutual Exclusion Algorithm", *ACM Transactions on Computer Systems*, Vol. 5, No. 1, February, 1987, pp. 1-11.

[13] B.-H. Lim and A. Agarwal, "Waiting Algorithms for Synchronization in Large-Scale Multiprocessors", *ACM Transactions on Computer Systems*, Vol. 11, No. 3, August, 1993, pp. 253-294.

[14] N. Lynch and N. Shavit, "Timing-Based Mutual Exclusion", *Proceedings of the Thirteenth IEEE Real-Time Systems Symposium*, December, 1992, pp. 2-11.

[15] J. Mellor-Crummey and M. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors", *ACM Transactions on Computer Systems*, Vol. 9, No. 1, February, 1991, pp. 21-65.

[16] M. Merritt and G. Taubenfeld, "Knowledge in Shared Memory Systems", *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, August, 1991, pp. 189-200.

[17] G. Pfister and A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks", *IEEE Transactions on Computers*, Vol. C-34, No. 11, November, 1985, pp. 943-948.

[18] P. Turán, "On an extremal problem in graph theory" (in Hungarian), *Mat. Fiz. Lapok*, Vol. 48, 1941, pp. 436-452.

[19] J.-H. Yang and J. Anderson, "Fast, Scalable Synchronization with Minimal Hardware Support", *Proceedings of the Twelfth ACM Symposium on Principles of Distributed Computing*, August, 1993, pp. 171-182. A revised version entitled "A Fast, Scalable Mutual Exclusion Algorithm" is scheduled to appear in *Distributed Computing*.

[20] J.-H. Yang and J. Anderson, "Time Bounds for Mutual Exclusion and Related Problems", *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, ACM, New York, pp. 224-233, May 1994.