Atomic m-register operations*

Michael Merritt† Gadi Taubenfeld†

December 15, 1993

Abstract

We investigate systems where it is possible to access several shared registers in one atomic step. We characterize those systems in which the consensus problem can be solved in the presence of faults and give bounds on the space required. We also describe a fast solution to the mutual exclusion problem using atomic m-register operations.

Key words: Fault-tolerance, shared memory, impossibility, lower bounds, agreement, consensus, mutual exclusion.

1 Introduction

Shared memory systems which support only atomic read and write operations have been extensively investigated, e.g. [1, 4, 12, 13, 14, 15, 18, 19]. This paper focuses attention on somewhat stronger systems in which it is possible to read or write several shared registers in one atomic step.

We say that a system supports atomic m-register operations if it is possible for a process to read *or* write m registers in one atomic step. Previous work shows that the ability to *read* multiple registers in a single operation does not by itself add computational power [2, 3, 1], while in some contexts, the ability to *write* several registers is more powerful than single-register primitives [11]. We explore these questions in more detail, examining complexity bounds as well as computability.

Powerful synchronization mechanisms such as semaphores, register-to-register swap, read-modify-write, and test-and-set have long been a part of operating system design. Herlihy's seminal paper demonstrated these primitives may be classified hierarchically, according to their computational power in the fail-stop fault model [11]. This has awakened a great deal of interest in the relative strengths of various synchronization primitives. The m-register primitive is of particular theoretical interest: as Herlihy showed, each successive value of m provides a quantum increase in the synchronization power of the primitive. Atomic m-registers can solve wait-free consensus for up to $2m-2$ processors, but no more. Hence, these primitives populate an infinite hierarchy between test-and-sets (which can be used to solve wait-free consensus for exactly 2 processors), and register-to-register swap or

*A preliminary version of this work appeared in the *Proceedings of the Fifth International Workshop on Distributed Algorithms*, Delphi, Greece, October 1991, pages 289-294.

†AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974.

read-modify-write (either of which can be used to solve wait-free consensus for any number of processors).

A still stronger model is one in which processors perform read-modify-write operations on multiple registers in a single atomic step. For example, this is the computational model underlying Unity [6]. An algorithm for computing graph connectivity in this model was presented in [9]. Because this model includes read-modify-write on a single register, it is much more powerful than reading *or* writing a bounded number of registers.

Wait-freedom corresponds to a model in which any number of processors may crash. Our first result explores the general case in which there is some bound, t, on the number of processors that may crash. (For n processors, the wait-free model corresponds to $t = n-1$.) The result was inspired by proofs in Fischer, Lynch and Paterson [10], Herlihy [11] and Loui and Abu-Amara [14], and shows that the consensus problem can be solved using n processes in systems which support atomic m-register operations if and only if $t \leq 2m-3$. The theorem holds also when a process can atomically read only a single register (but can atomically write m registers).

Two known results are special cases of this. Loui and Abu-Amara [14] proved the result for the case $m = 1$ (for any n and t), and Herlihy [11] proved it for the case $t = n-1$ (for any m).[1] The case where $m = 1$ and $t = n-1$, which itself is a special case of the Loui and Abu-Amara result, is proved also in [8, 11].

Next, we prove a lower bound on the number of shared registers necessary to solve the consensus problem. We show that for any n, m, and $t < n$, in a system that supports atomic m-register operations, every t-resilient consensus protocol for n processes must use at least $(t+3)\min(t+1,m) - \min(t+1,m)^2 - 1$ shared registers. That is, if $t+1 \leq m$, at least $2t+1$ registers are needed, and when $t+1 \geq m$, $(t+3)m - m^2 - 1$ are required. We also design a consensus protocol that uses $(t^2+3t+2)/2$ shared registers. The results depend on generalizations of Herlihy's arguments for the case $n = t-1$. This result indicates that the power of a synchronization primitive (in terms of the number of processes which can use it to reach consensus) can depend on *how many* instances of the primitive are available.

Finally, we use atomic m-register operations to design a solution to the mutual exclusion problem. Assuming a fault-free environment, and using the tree-based mutual exclusion algorithm of [16], we show that there is a starvation-free solution to the mutual exclusion problem with time complexity $3\lceil \log_m n \rceil$. The time complexity is defined as the number of accesses to the shared memory in order to enter the critical section in the absence of contention [13].

Sections 2 through 4 assume that a process can either read m registers or write to m registers in one atomic step. Section 5 discusses the case of *mixed m-register operations*, where a process can "mix" reads and writes, accessing up to m registers atomically, choosing some to write and others to read. It turns out that mixing operations buys no additional computational power, except in the odd case where $t = 1$. (Theorem 5 shows that mixed 2-registers, with operations containing only single writes, suffice to solve consensus for $t = 1$ and any number of processors.)

[1] Although Herlihy assumes that a process may atomically read only a single register, the proof applies unchanged to the case considered here, in which multiple registers may be read atomically.

2 Asynchronous Shared Memory Systems

This section characterizes asynchronous shared memory systems which support atomic access to m registers, by stating axioms that any protocol operating in such systems satisfies. The axioms do not give a complete characterization of these systems; only those axioms are stated that are needed to prove the lower bounds. (This and the next two sections do not discuss *mixed m-register operations*, which are discussed in Section 5.)

We start with a formal description of the notion of a protocol. We assume a set of *register names* $\{r_0, r_1, \ldots\}$, a set of *process names* $\{p_0, p_1, \ldots\}$, and a set of *values*. An *event* corresponds to an atomic step performed by a process. We consider here only the following types of events.

1. $read_p(r_1, ..., r_m, v_1, ..., v_m)$ – process p reads the values $v_1, ..., v_m$ from registers $r_1, ..., r_m$ respectively;

2. $write_p(r_1, ..., r_m, v_1, ..., v_m)$ – process p writes the values $v_1, ..., v_m$ into registers $r_1, ..., r_m$ respectively.

We use the notation e_p to denote an instance of an arbitrary event at a process p, which may be an instance of either of the above types of events.

A *run* is a pair (f, S) where f is a function which assigns initial values to the registers and S is a finite or infinite sequence of events. When S is finite, we also say that the run is finite.

A protocol $Pr = (C, N, R)$ consists of a nonempty set C of runs, a set N of processes, and a set of registers R. For any event e_p at a process p in any run in C, the registers read from or written to in e_p must all be in R.

The *value* of a register at a finite run is the last value that was written into that register, or its initial value (determined by f) if no process wrote into the register. We use $value(r, x)$ to denote the value of register r at a finite run x.

Let $x = (f, S)$ and $x' = (f', S')$ be runs. Run x' is a *prefix* of x (and x is an *extension* of x'), denoted $x' \leq x$, if S' is a prefix of S and $f = f'$. When $x \geq x'$, $(x - x')$ denotes the suffix of S obtained by removing S' from S.

Let $\langle S; S' \rangle$ be the sequence obtained by concatenating the finite sequence S and the sequence S'. Then $\langle x; S' \rangle$ is an abbreviation for $(f, \langle S; S' \rangle)$.

For any sequence S, let S_p be the subsequence of S containing all events in S which involve p. Run (f, S) *includes* (f', S') if $f = f'$ and S'_p is a prefix of S_p for all $p \in N$. Runs (f, S) and (f', S') are *indistinguishable* for a set of processes P, denoted by $(f, S)[P](f', S')$, iff for all $p \in P$, $S_p = S'_p$ and $f(r) = f'(r)$ for every local register r of p. When $P = \{p\}$ we write $[p]$ instead of $[P]$. We assume throughout this paper that x is a run of a protocol if and only if all finite prefixes of x are runs. Notice that, by this assumption, if (f, S) is a run then also $(f, null)$ is a run, where *null* is the empty sequence.

Without loss of generality, we also assume that the processes are deterministic. That is, if $\langle x; e_p \rangle$ and $\langle x; e'_p \rangle$ are runs then $e_p = e'_p$.

DEFINITION **1** *An asynchronous protocol that supports atomic m-register operations is a protocol whose runs satisfy axioms A1 – A3.*

AXIOMS FOR ATOMIC m-REGISTER OPERATIONS

A1 Let $\langle x; write_p(r_1, ..., r_m, v_1, ..., v_m)\rangle$ and y be finite runs where $x[p]y$.
Then $\langle y; write_p(r_1, ..., r_m, v_1, ..., v_m)\rangle$ is a run.

A2 Let $\langle x; read_p(r_1, ..., r_m, v_1, ..., v_m)\rangle$ and y be finite runs where $x[p]y$.
Then $\langle y; read_p(r_1, ..., r_m, u_1, ..., u_m)\rangle$ is a run for some values $u_1, ..., u_m$.

A3 Let $\langle x; read_p(r_1, ..., r_m, v_1, ..., v_m)\rangle$ be a finite run.
Then $v_i = value(r_i, x)$ for all $1 \leq i \leq m$.

Axiom *A1* means that if a write event which involves p can happen at a run, then the same event can happen at any run that is indistinguishable to p from it. Axiom *A2* means that if a process is "ready to read" some values from some registers then an event on some other process cannot prevent it from reading from those registers although it may read a different set of values than it would otherwise. Axiom *A3* means that it is possible to read only the last value that is written into a register.

We say that process p is *enabled* at run x if there exists an event e_p such that $\langle x; e_p\rangle$ is a run. For simplicity, we write xp to denote either $\langle x; e_p\rangle$ (when p is enabled in x), or x (when p is not enabled in x). For a set of processes $P = \{p_1, ..., p_k\}$ we write xP instead of $\langle xp_1...p_k\rangle$.[2] Hence, $xP^2 = xPP = \langle xp_1...p_kp_1...p_k\rangle$, and $xP^\infty = xPP... = \langle xp_1...p_kp_1...p_k...\rangle$.

For a write event e_p and run $\langle x; e_p\rangle$, we denote by $write\_set(x, p)$ the set of all the registers process p is writing to in the event e_p. If p is not enabled after x, or e_p is a read event, $write\_set(x, p)$ is empty. For a set of processes P, $write\_set(x, P) = \bigcup_{p\in P} write\_set(x, p)$.

We also write $P - p$ instead of $P - \{p\}$, $P + p$ instead of $P \cup \{p\}$, and by $\overline{p}$ we denote the set $N - p$ (i.e., the set of all processes excluding p). For $v \in \{0, 1\}$, let $\overline{v} = v + 1 \pmod 2$.

In order to discuss important properties of asynchronous protocols, we need the concept of a fair run. Process p is *correct* in a run y if for each run $x \leq y$, if p is enabled at x then some event in $(y - x)$ involves p. (As a consequence, p can be correct in a *finite* run y only if p is not enabled in y.) A run is ℓ-fair if at least ℓ processes are correct in it, and a run is P-fair, for P a set of processes, if all processes in P are correct in it.

The following lemmas, easy consequences of the axioms and definitions, are used in the proofs that follow.

Lemma 1 *Let $w = \langle f, S_1\rangle$, $x = \langle f, S_1; S_2\rangle$, and $y = \langle f', S'\rangle$ be runs of a protocol and P a set of processes such that*

- *w and y are finite,*
- *$w[P]y$,*
- *the values of all shared registers are the same in w and y,*
- *and S_2 contains only events of processes in P.*

[2]The definition of xP implicitly assumes an order on the process names in P that is respected in the order of events in $\langle xp_1...p_k\rangle$.

Then $z = \langle f', S'; S_2 \rangle$ is a run of the protocol and $x[P]z$.

PROOF: By induction on the length of S_2. □

Note that it is immediate from the axioms for atomic m-register operations that enabled processes cannot become disabled as a result of an event which involves some other process. That is, if p is enabled in x and $x[p]y$, then p is enabled in y.

Lemma 2 *Let $x = \langle f, S \rangle$ be a P-fair run of a protocol and $y = \langle f, S' \rangle$ be a run such that $x[P]y$. Then y is P-fair.*

PROOF: Suppose $p \in P$ and let y' be a finite run such that $y' \leq y$. If y' contains no events at p, let x' be *null*. Otherwise, let e'_p be the last event at p in y'. Then there is a corresponding event e_p in x. Let x' be the prefix of x ending in e_p. In either case, since $x[P]y$, it follows that $x'[p]y'$. If p is enabled in y' then (by the observation above) p is enabled in x'. It follows that $(x - x')$ and hence $(y - y')$ contain an event involving p. The lemma follows.□

Lemma 3 *For every finite run x of a protocol and every set of processes P there is a P-fair extension y of x such that $x[\overline{P}]y$. (That is, $y = \langle x; S \rangle$, where S contains only events of processes in P.)*

PROOF: From Axioms *A1* – *A3*, xP^∞ is a run and is a P-fair extension of x. □

3 Consensus

In this section we characterize those systems in which the consensus problem can be solved in the presence of faults and give bounds on the space required.

A *t-resilient consensus protocol* $Pr = (C, N, R)$ is a protocol for n processes, where each process has a local, binary, read-only input register and a local, binary, write-once output register.[3] The protocol Pr must satisfy the following conditions:

- The set of runs C includes 2^n empty runs, each with a different boolean combination as initial values in the input registers, and with identical initial values in the remaining registers.
- For every finite run x in C, and every process p, xp is in C.
- The set C is limit-closed: if every finite prefix of a run x is in C, then x is in C.
- Every $(n - t)$-fair run has a finite prefix in which all the correct processes decide on either 0 or 1 (i.e., each correct process writes 0 or 1 into its local output register), the decisions of all processes are the same, and this decision value is equal to the input value of some process.

[3]Formally, the runs in C include only read events, by the associated process, for each input register, and at most one write event, again by the associated process, for each output register. The input and output registers are not counted in our discussions of space.

3.1 Fault-tolerance

In Theorem 1, we characterize the cases in which the consensus problem can be solved in a system which support atomic m-register operations.

Theorem 1 *For any n, $1 \leq m$, and $1 \leq t < n$, in a system that supports atomic m-register operations, there is a t-resilient consensus protocol for n processes if and only if $t \leq 2m-3$.*

UPPER BOUND PROOF: The upper bound is an easy corollary of Herlihy's wait-free consensus protocol [11]:

Proposition 1 (Herlihy) *If $n \leq 2m-2$, in a system that supports atomic m-register operations, there is a wait-free consensus protocol using $n(n+1)/2$ m-registers. Moreover, this protocol has the property that for each processor p_i, there is a register r_i to which only p_i writes.*

If $t \leq 2m-3$, a t-resilient consensus protocol can be derived as follows: The $t+1$ processes $p_0, ..., p_t$ run Herlihy's protocol (letting $n = t+1$). Because that protocol is wait-free, in any $n-t$-fair run, at least one of these $t+1$ processes will decide, and all such decisions in a given run are the same. Before halting, each deciding processor p_i writes its decision value $v \in \{0,1\}$ to a an additional subfield of r_i that is initially $\perp$. (This requires p_i to keep in it's local state the last value written in other fields of r_i, so that v can be written without changing the fields used in the embedded consensus protocol.)

Each remaining process p_j reads each of the registers $r_0, \ldots, r_t$ over and over, until one of the new subfields is observed to be different from $\perp$. The value read is written to p_j's output register as its decision value.

It is easy to see that the resulting protocol is a t-resilient consensus protocol. Moreover, it uses no more registers than are needed by the embedded version of Herlihy's protocol run by $t+1$ processes, or $(t^2+3t+2)/2$ m-registers.

PROOF OF LOWER BOUND: The proof of the lower bound uses the following notions, abbreviations, and lemmas. The underlying ideas and much of the terminology are adapted from [10] (see also [17]).

A finite run x is *v-valent* if in all extensions of x where a decision is made, the decision value is v ($v \in \{0,1\}$). A run is *univalent* if it is either 0-valent or 1-valent, otherwise it is *bivalent*. Using Lemmas 1 and 2, we can prove the following simple observation.

Lemma 4 *In any t-resilient, m-register consensus protocol, if two univalent runs are indistinguishable for $n-t$ processes, and the values of the shared registers are the same at these runs, then the runs must have the same valency.*

PROOF: Let w and x be univalent runs and P a set of $n-t$ processes, such that $w[P]x$ and the values of the shared registers are the same at w and x. By Lemma 2, w has a P-fair extension, y, in which $y - w$ contains only events of processes in P. Let w be v-valent, for $v \in \{0,1\}$. Then by the definition of t-resilient consensus protocol, the processes in P decide v in y. By Lemma 1, $z = \langle x; (y-w) \rangle$ is a run of the protocol such that $z[P]y$, and by Lemma 2, z is P-fair. Since the members of P write v to their output registers in z, the definition of t-resilient consensus protocol implies that x is v-valent. $\square$

Lemma 5 *Let xp be a run of a 1-resilient, m-register consensus protocol and let Q be a set of processes where $p \notin Q$. Assume that for every $q \in Q$, run xq is v-valent and run xpq is $\overline{v}$-valent. Then $m \geq |Q| + 1$, and the number of registers used is at least $2|Q| + 1$.*

PROOF: **Case 1:** Suppose first that $write\_set(x, p) \subseteq write\_set(x, Q)$. Let Q_w be the subset of those processes in Q for which write events are enabled after x, and Q_r be the subset of Q for which read events are enabled after x. Then,

$$xQ_wQ_r[\overline{p}]xpQ_wQ_r.$$

Since xQ_wQ_r is v-valent, xpQ_wQ_r is $\overline{v}$-valent, and the values of all registers in these two runs are the same, by applying Lemma 4 we arrive at a contradiction. Thus, $write\_set(x, p)$, contains at least one register that is not written next by any process in Q. (This case also implies that the next step by p is a write.)

Case 2: Suppose now that there exists $q \in Q$ such that $write\_set(x, p) \cap write\_set(x, q) \subseteq write\_set(Q - q)$. Then,

$$xqp(Q_w - q)(Q_r - q)[N]xpq(Q_w - q)(Q_r - q).$$

But $xqp(Q_w - q)(Q_r - q)$ is v-valent, $xpq(Q_w - q)(Q_r - q)$ is $\overline{v}$-valent, and the values of all registers in these two runs are the same, again Lemma 4 leads to a contradiction. Thus, for every process $q \in Q$, $write\_set(x, p)$ contains at least one register both p and q write next that is not written next by any other process in Q. Case 1 and Case 2 imply that $write\_set(x, p) \geq |Q| + 1$. (Case 2 also implies that the next step by every process in Q is a write, and hence $Q_w = Q$.)

Case 3: Finally, suppose that there exists $q \in Q$ such that $write\_set(x, q) \subseteq write\_set(x, Q - q)$ $\cup$ $write\_set(x, p)$. Then,

$$xqp(Q - q)[\overline{q}]xp(Q - q).$$

But $xqp(Q - q)$ is v-valent, $xp(Q - q)$ is $\overline{v}$-valent, and the values of all registers in these two runs are the same, once again applying Lemma 4 leads to a contradiction. Thus, $write\_set(x, q)$, contains at least one register that is not written next by any other process in $Q + p$.

The lower bound bound, $2|Q| + 1$, on the number of shared registers follows: By Case 1, at least one register is written next by p and by no process in Q, by Case 2, $|Q|$ registers are each written next by p and a distinct process in Q, and by Case 3, $|Q|$ additional registers are each written next by a distinct process in Q and no other process in $Q + p$. □

To finish the proof of the lower bound, assume Pr is a t-resilient consensus protocol. We construct a run x and set P of at least $(t + 1)/2$ processes matching the premises of Lemma 5.

To construct x, we need to begin with a bivalent run x_0. Following the argument in [10], we show that a bivalent empty run must exist. Suppose every empty run is univalent. The empty run with all 0 inputs is 0-valent, and similarly the empty run with all 1 inputs is 1-valent. By changing one input from 0 to 1, we arrive at two empty runs x_0 and x_1 that differ only at the value of a single input, for process p, such that x_0 is 0-valent and x_1 is 1-valent. Then all processes but p decide 0 in the $(N - p)$-fair extension x'_0 of x_0, in which p takes no steps. By induction and limit-closure, there is a corresponding extension x'_1 of x_1

such that $x'_0[\overline{p}]x'_1$. But it follows that the processes in x'_1 decide 0, and x_1 is not 1-valent, a contradiction. Hence, a empty bivalent run x_0 exists.

We begin with x_0 and pursue the following round-robin *bivalence-preserving scheduling* discipline:

$x := x_0$; $P := \phi$; $i := 0$;
`repeat`
    `if` x has a bivalent extension $x'p_i$
    `then` $x := x'p_i$
    `else` $P := P + p_i$
    $i := i + 1 (\text{mod } n)$
`until` $|P| = t + 1$.

If this procedure does not terminate, then there is an $(n - t)$-fair run with only bivalent finite prefixes. However, the existence of such a run contradicts the definition of t-resilient consensus protocols. Hence, the procedure will terminate with some bivalent finite run x, and a set P of $t + 1$ processes such that any extension $x'p$ of x, for any process p in P, is univalent. Suppose that for some $p \in P$, the run xp is v-valent. Since x is bivalent, there is a (shortest) extension x' of x which is $\overline{v}$-valent.

Consider the runs y between x and x', $x \leq y \leq x'$, and the valency of yp, for $p \in P$. One of two cases holds:

1. For some $x \leq y \leq x'$ the set P is partitioned into nonempty sets P_0 and P_1, such that for all $p \in P_0$, yp is 0-valent, and for all $p \in P_1$, yp is 1-valent. Assume without loss of generality that P_1 is the bigger of the two sets. Then applying Lemma 5 to y, P_1, and some $q \in P_0$, yields that $m \geq |P_1| + 1 \geq (t + 1)/2 + 1$.

2. For some $x \leq yq \leq x'$, it is the case that $q \notin P$, for all $p \in P$ the run yp is v-valent and the run yqp is $\overline{v}$-valent. Then, applying Lemma 5 to y, P, and q implies that $m \geq |P| + 1 = t + 2$.

Thus, we conclude that $t \leq 2m - 3$. □

Obviously, the theorem holds also when a process can atomically read only a single register (but can atomically write to m registers). The same bound holds for such problems as leader election. Section 5 discusses a variation of this proof that shows that the same result holds for mixed registers, when $t \geq 2$.

3.2 Space

We prove a lower bound of $(t + 3)\min(t + 1, m) - \min(t + 1, m)^2 - 1$ shared registers on the number of shared registers necessary to solve the consensus problem. Recall thet the algorithm we presented after Proposition 1 requires $(t^2 + 3t + 2)/2$ shared registers.

Theorem 2 *For any n, m, and $1 \leq t < n$, in a system that supports atomic m-register operations, every t-resilient consensus protocol for n processes must use at least $(t + 3)\min(t + 1, m) - \min(t + 1, m)^2 - 1$ shared registers.*

We first prove the following lemma.

Lemma 6 *Let x be a run of a 1-resilient, m-register consensus protocol, let P_0 and P_1 be nonempty disjoint sets of processes. Assume that for every $p_0 \in P_0$ and $p_1 \in P_1$ run xp_0 is 0-valent and run xp_1 is 1-valent. Then the number of registers used is at least $|P_0| + |P_1| + |P_0||P_1|$.*

PROOF: Let $p_v \in P_v$ where $v \in \{0,1\}$, and let $P = P_0 \cup P_1$. We show first that $write\_set(x, p_0) \neq \emptyset$; by a similar argument, $write\_set(x, p_1) \neq \emptyset$. Assume that $write\_set(x, p_0) = \emptyset$. Then,

$$xP_1[\overline{p_0}]xp_0P_1.$$

But xP_1 is 1-valent, xp_0P_1 is 0-valent, and the values of all registers in these two runs are the same, contradicting Lemma 4. Thus, it follows that the next step by every process in P is a write.

Suppose that $write\_set(x, p_0) \subset write\_set(x, P - p_0)$. Then,

$$xP_1(P_0 - p_0)[\overline{p_0}]xp_0P_1(P_0 - p_0).$$

But $xP_1(P_0 - p_0)$ is 1-valent, $xp_0P_1(P_0 - p_0)$ is 0-valent, and the values of all registers in these two runs are the same, contradicting Lemma 4. Thus, it follows that $write\_set(x, p_0)$ contains at least one register not written next by any other process in P. By a similar argument, $write\_set(x, p_1)$ also contains at least one register not written next by any other process in P, and hence there are a minimum of $|P_0| + |P_1|$ such registers.

Suppose next that there exists $p_0 \in P_0$ and $p_1 \in P_1$ such that $write\_set(x, p_0) \cap write\_set(x, p_1) \subset write\_set(P - \{p_0, p_1\})$. Then,

$$xp_0p_1(P - \{p_0, p_1\})[N]xp_1p_0(P - \{p_0, p_1\}).$$

But $xp_0p_1(P - \{p_0, p_1\})$ is 0-valent, $xp_1p_0(P - \{p_0, p_1\})$ is 1-valent, and the values of all registers in these two runs are the same, again contradicting Lemma 4. Thus, it follows for every pair of processes $p_0 \in P_0$ and $p_1 \in P_1$, that $write\_set(x, p_0) \cap write\_set(x, p_1)$ contains at least one register both p_1 and p_0 write next but no other process in P writes next. There are a minimum of $|P_0||P_1|$ such registers. The result follows. □

To prove the lower bound of Theorem 2, we inductively construct a run x exactly as in the proof of the previous theorem. Hence, as in Theorem 1, there is some bivalent finite run x, and a set P of $t + 1$ processes such that any extension of x containing a step by any member of P is univalent. Suppose that for some $p \in P$, the run xp is v-valent. Since x is bivalent, there is a (shortest) extension x' of x which is $\overline{v}$-valent. As in Theorem 1 one of two cases holds, and for each one of them we calculate the number of registers necessary. Let S denote the number of registers used.

Case 1: For some $x \leq y \leq x'$ the set P is partitioned into nonempty sets P_0 and P_1, such that for all $p_0 \in P_0$, yp_0 is 0-valent, and for all $p_1 \in P_1$, yp_1 is 1-valent. Let $a = max(|P_0|, |P_1|)$ and $b = \min(|P_0|, |P_1|)$.

According to Lemma 6,

$$S \geq a + b + ab. \tag{1}$$

Since $a + b = |P|$ and by construction $|P| = t + 1$,

$$a + b = t + 1. \tag{2}$$

From (the first part of) Lemma 5, and the fact that $b \geq 1$,

$$a + 1 \leq m. \tag{3}$$

Hence, using (2) and (3), and the fact that $b \geq 1$,

$$a \leq \min(t + 1, m) - 1. \tag{4}$$

From (2), $b = t + 1 - a$ and by substituting in (1),

$$S \geq (t + 1) + a(t + 1 - a) = -a^2 + (t + 1)(a + 1). \tag{5}$$

The strongest possible lower bound on S is obtained when $a = (t + 1)/2$.
Hence, from (4) and (5),

$$S \geq -(\min(t + 1, m) - 1)^2 + (t + 1)(\min(t + 1, m)), \tag{6}$$

which implies

$$S \geq -\min(t + 1, m)^2 + (t + 3)\min(t + 1, m) - 1. \tag{7}$$

This complete the proof of the first case.

Case 2: For some $x \leq yq \leq x'$, it is the case that $q \notin P$, for all $p \in P$ the run yp is v-valent and the run yqp is $\overline{v}$-valent.

From the second part of Lemma 5, and the fact that $|P| = t + 1$,

$$S \geq 2|P| + 1 = 2t + 3, \tag{8}$$

and from the first part of Lemma 5,

$$m \geq |P| + 1 = t + 2. \tag{9}$$

But when $m \geq t + 1$, the theorem requires a weaker bound than that in (8),

$$S \geq (t+3)\min(t+1, m) - \min(t+1, m)^2 - 1 = (t+1)^2 + (t+3)(t+1) - 1 = 2t + 1. \tag{10}$$

□

4 Mutual Exclusion

The mutual exclusion problem, which is one of the most studied problems in concurrent computing, is to design a protocol that guarantees mutually exclusive access to a critical section among a number of competing processes. Such a protocol is starvation-free if any correct process that attempts to enter the critical section eventually succeeds. In this section we use atomic m-register operations to design a fast solution to the mutual exclusion problem. The time complexity is defined as the number of accesses to the shared memory in

order to enter the critical section in the absence of contention [13]. For us, as for Lamport, this time dominates the time in the exit region.

Lamport notes that all the previously published starvation-free mutual exclusion algorithms (for $m = 1$) require a process to execute at least $O(n)$ operations to shared memory in the absence of contention [13]. By slightly modifying the solution in [16], it is possible to design, for $m = 1$, a starvation-free solution to the mutual exclusion problem with time complexity $O(\log_2 n)$. By designing a solution for m processes, replacing the binary tree in [16] with an m-ary tree, and assuming a fault-free environment, we obtain the following theorem.

Theorem 3 *For any n and $m \geq 1$, in a system with n processes that supports atomic m-register operations, there is a starvation-free solution to the mutual exclusion problem with time complexity $3\lceil \log_m n \rceil$.*

PROOF: We first describe a simple starvation-free solution to the mutual exclusion problem for m processes, with a time complexity of 3 m-register operations. In this solution every process p_i has a unique register r_{ii}, and every two processes p_i and p_j have a single register, denoted by both r_{ij} and r_{ji}, that is shared only by them. All $m + m(m-1)/2$ registers are initially set to zero. We say that process p_i is a *winner* at some point if $r_{ii} = i$ and for every $j \neq i$ either $r_{jj} = 0$ or $r_{ij} = j$. To enter its critical section, process p_i first writes i into all the registers r_{ik} where $1 \leq k \leq m$. Then, p_i reads all the $2m - 1$ registers r_{ik} and r_{kk} where $1 \leq k \leq m$. If the values read indicate p_i is the winner, then p_i enters its critical section; otherwise p_i busy-waits on these $2m - 1$ registers until it observes that p_i is the winner. To exit its critical section process p_i simply sets r_{ii} to zero. The protocol actually implements a queue and it is not difficult to see that it is a starvation-free solution to the mutual exclusion problem.

The above solution can be easily generalized to n processes by constructing an m-ary tree where each node is a copy of the solution for m processes, using a separate set of registers. To enter its critical section a process i starts participating in the protocol at leaf $\lceil i/m \rceil$ (level 1) and advances towards the root of the tree. A process advances to level i if it is a winner at level $i - 1$. A process can enter its critical section when it is the winner at the root.

To exit its critical section process p_i simply sets all the r_{ii}'s in its path from the leaf to the root to zero. Since the depth of the tree is $\lceil \log_m n \rceil$, this can be done in $\lceil (\lceil \log_m n \rceil / m) \rceil$ m-register operation. □

We notice that in the above solution the register r_{ii} used at level $i - 2$ can be reset as soon as a process advances from level $i - 1$ to level i. This allows competing processes to advance up the tree behind the winner. However, this increases the worst-case time complexity to $4\lceil \log_m n \rceil - 1$, with a single assignment in the exit region. Resetting these $\lceil \log_m n \rceil - 1$ registers could also be done in parallel with the critical section execution.

Finally, for systems that support only atomic read and write operations (i.e., $m = 1$) Burns and Lynch proved that any deadlock-free solution to the mutual exclusion problem must use at least as many shared registers as processes [5]. They also showed that this lower bound is tight. Their lower bound proof generalizes trivially to the case $m > 1$. That is, for every integer $m \geq 1$, in a system which supports atomic m-register operations, a

deadlock-free solution to the mutual exclusion problem must use at least as many shared registers as processes.

5 Mixed m-register operations

A *mixed m-register* operation allows a mixture of reads and writes of m (distinct) registers in a single atomic operation. The results below explore the minimum number of writes that must occur in mixed m-register operations to solve the consensus and mutual exclusion problems. We denote a mixed m-register operation by

$$(write_p(r_1, ..., r_W, v_1, ..., v_W), read_p(r_1^{'}, ..., r_R^{'}, v_1^{'}, ..., v_R^{'}))$$

and require that $\{r_1, ..., r_W\}$ and $\{r_1^{'}, ..., r_R^{'}\}$ be disjoint sets, containing together at most m elements. (That is, $W + R \leq m$.)

DEFINITION **2** *An asynchronous protocol that supports atomic mixed m-register operations is a protocol whose runs satisfy axioms MA1 – MA2.*

AXIOMS FOR ATOMIC MIXED m-REGISTER OPERATIONS

MA1 Let $\langle x; (write_p(r_1, ..., r_W, v_1, ..., v_W), read_p(r_1^{'}, ..., r_R^{'}, v_1^{'}, ..., v_R^{'}))\rangle$
and y be finite runs where $x[p]y$.
Then $\langle y; (write_p(r_1, ..., r_W, v_1, ..., v_W), read_p(r_1^{'}, ..., r_R^{'}, u_1, ..., u_R))\rangle$
is a run for some values $u_1, ..., u_R$.

MA2 Let $\langle x; (write_p(r_1, ..., r_W, v_1, ..., v_W), read_p(r_1^{'}, ..., r_R^{'}, v_1^{'}, ..., v_R^{'}))\rangle$ be a run.
Then $v_i^{'} = value(r_i^{'}, x)$ for all $1 \leq i \leq R$.

Similar proofs as those given for m-register operations suffice to show that Lemmas 1, 2, 3 and 4, hold also for protocols that support atomic mixed m-register operations.

5.1 Fault-tolerance

Except for the case $t = 1$, a variant of Theorem 1 holds for mixed m-registers, restricting the number of registers which are written simultaneously. Hence, solving consensus is not affected by adding any number of reads to a fixed number of writes, except for the case $t = 1$. (Theorem 5 shows that mixed 2-registers, with operations containing only single writes, suffice to solve consensus for $t = 1$.)

Theorem 4 *For any n, $1 \leq m$, and $2 \leq t < n$, in a system that supports atomic mixed m-register operations, any t-resilient consensus protocol for n processes must contain operations which write a value into at least $(t + 3)/2$ registers in a single step, no matter how many other registers are read.*

By Theorem 1, this bound is tight. The proof of the theorem is the same as that of Theorem 1, substituting the following variant of Lemma 5.

Lemma 7 *Let xp be a run of a t-resilient, mixed m-register consensus protocol and let Q be a set of processes, where $p \notin Q$ and $2 \leq |Q| \leq t$. Assume that for every $q \in Q$, run xq is v-valent and run xpq is $\overline{v}$-valent. Then the protocol must contain operations that atomically write to at least $|Q|+1$ registers (regardless of the number of registers read simultaneously), and the number of registers used is at least $2|Q|+1$.*

PROOF: **Case 1:** Suppose first that $write\_set(x,p) \subset write\_set(x,Q)$. Let $q \in Q$. Then,

$$x(Q-q)q[\overline{Q-q+p}]xp(Q-q)q.$$

(Because they may be simultaneously reading and writing, the processes in $Q-q+p$ may observe a difference between $x(Q-q)q$ and $xp(Q-q)q$. But since $write\_set(x,p) \subset write\_set(x,Q) = write\_set(x,Q-q) \cup write\_set(x,q)$, it follows that $write\_set(x,p) - write\_set(x,Q-q) \subset write\_set(x,q)$. Hence, q overwrites any remaining distinct registers that distinguish $x(Q-q)$ from $xp(Q-q)$, and so q will not observe any difference.) We have that $x(Q-q)q$ is v-valent, $xp(Q-q)q$ is $\overline{v}$-valent, $|Q-q+p| \leq t$, and the values of all registers in these two runs are the same, contradicting Lemma 4. Thus, $write\_set(x,p)$ contains at least one register not written next by any process in Q.

Case 2: Suppose next that there exists $q \in Q$ such that $write\_set(x,p) \cap write\_set(x,q) \subset write\_set(Q-q)$. Let $q' \in Q$ where $q' \neq q$, and note that $\overline{Q-q'+p}$ contains at least the process q'. Then,

$$xqp(Q-\{q,q'\})q'[\overline{Q-q'+p}]xpq(Q-\{q,q'\})q'.$$

But $xqp(Q-\{q,q'\})q'$ is v-valent, $xpq(Q-\{q,q'\})q'$ is $\overline{v}$-valent, $|Q-q'+p| \leq t$, and the values of all registers in these two runs are the same, again contradicting Lemma 4. Thus, $write\_set(x,q)$, contains for every process $q \in Q$, at least one register both p and q write to but no other process in Q writes to. Cases 1 and 2 imply that $write\_set(x,q) \geq |Q|+1$, and hence the first part of the lemma is proven.

Case 3: Finally, suppose that there exists $q \in Q$ such that $write\_set(x,q) \subset write\_set(x,Q-q) \cup write\_set(x,p)$. Let $q' \in Q$ where $q' \neq q$, and note again that $\overline{Q-q'+p}$ contains at least the process q'. Then,

$$xqp(Q-\{q,q'\})q'[\overline{Q-q'+p}]xp(Q-\{q,q'\})q'.$$

Then $xqp(Q-\{q,q'\})q'$ is v-valent, $xp(Q-\{q,q'\})q'$ is $\overline{v}$-valent, $|Q-q'+p| \leq t$, and the values of all registers in these two runs are the same, once again contradicting Lemma 4. Thus, $write\_set(x,q)$ contains at least one register not written next by any other process in $Q+p$.

The bound, $2|Q|+1$, on the number of shared registers follows: By Case 1, at least one register is written next by p and by no process in Q, by Case 2, $|Q|$ registers are each written next by p and a distinct process in Q, and by Case 3, $|Q|$ additional registers are each written next by a distinct process in Q and no other process in $Q+p$. □

Note that a mixed 1-register operation is simply a write or a read of a single register, hence the lower bound of Theorem 1 holds in this case. That is, there is no 1-resilient consensus protocol for this case. But what happens when $t = 1$ and in addition to writing a single register it is also possible to read other registers?

Theorem 5 *There is a 1-resilient consensus protocol for any number of processes using atomic mixed 2-registers operations, with a single write and a single read.*

PROOF: Pick processes p_0 and p_1 and three three-valued registers r_0, r_1 and r_{decide} which are initially $\perp$. Processor p_0 writes its initial value to r_0 and simultaneously reads r_1, while process p_1 writes its initial value to r_1 and simultaneously reads r_0. If either reads $\perp$, it chooses its own initial value, otherwise choosing the other's initial value. Upon reaching a decision, each writes the decision to r_{decide}. The other processes each read r_{decide} repeatedly, until the first non-$\perp$ is returned. This value is decided upon. □

5.2 Space

Next we prove a variant of Theorem 2.

Theorem 6 *Let w be the maximum number of registers a process may write in one atomic operation in a system that supports atomic mixed m-register operations (it may read other registers in that same operation). In such a system, for every n, m, and $2 \leq t < n$, every t-resilient consensus protocol for n processes must use at least $(t+3)\min(t+1, w) - \min(t+1, w)^2 - 1$ shared registers,*

The proof of this theorem is again similar to that of Theorem 2, substituting Lemma 7 for Lemma 5, and using the following variant of Lemma 6.

Lemma 8 *Let x be a run of a t-resilient, mixed m-register consensus protocol, let P_0 and P_1 be nonempty disjoint sets of processes, where $2 \leq |P_0 \cup P_1| \leq t+1$. Assume that for every $p_0 \in P_0$ and $p_1 \in P_1$ run xp_0 is 0-valent and run xp_1 is 1-valent. Then the number of registers used is at least $|P_0| + |P_1| + |P_0||P_1|$.*

PROOF: Let $p_v \in P_v$ where $v \in \{0, 1\}$, and let $P = P_0 \cup P_1$.

Suppose that $write\_set(x, p_0) \subset write\_set(x, P - p_0)$. Let p' be the last process scheduled in the run $xp_0P_1(P_0 - p_0)$. (Note that $\overline{P - p'}$ contains at least p'.) Then,

$$xp_0P_1(P_0 - p_0)[\overline{P - p'}]xP_1(P_0 - p_0).$$

That is, the other processes in P overwrite all registers written by p_0, and p' which is in $\overline{P - p'}$, is unable to distinguish whether p_0 took a step. (Note that p' is also the last process which is scheduled in the run $xP_1(P_0 - p_0)$.) We have that $xp_0P_1(P_0 - p_0)$ is 0-valent, $xP_1(P_0 - p_0)$ is 1-valent, $|P - p'| \leq t$, and the values of all registers in these two runs are the same, contradicting Lemma 4. Thus, $write\_set(x, p_0)$ contains at least one register that is not written next by any other process in P. By a similar argument, $write\_set(x, p_1)$ satisfies the same property, and there are a minimum of $|P_0| + |P_1|$ such registers.

Suppose next that there exists $p_0 \in P_0$ and $p_1 \in P_1$ such that $write\_set(x, p_0) \cap write\_set(x, p_1) \subset write\_set(P - \{p_0, p_1\})$. Let p' be the last process scheduled in the run $xp_0p_1(P - \{p_0, p_1\})$. (Once again, note that $\overline{P - p'}$ contains at least p'.) Then,

$$xp_0p_1(P - \{p_0, p_1\})[\overline{P - p'}]xp_1p_0(P - \{p_0, p_1\}).$$

Here, the processes in P overwrite all registers written by both p_0 and p_1, and p' which is in $\overline{P - p'}$, is unable to distinguish which of p_0 or p_1 took a step first. (As above, p' is also scheduled last in $xp_1p_0(P - \{p_0, p_1\})$.) We have that $xp_0p_1(P - \{p_0, p_1\})$ is 0-valent, $xp_1p_0(P - \{p_0, p_1\})$ is 1-valent, $|P - p'| \leq t$, and the values of all registers in these two runs are the same, again contradicting Lemma 4. Thus, it follows for every pair of processes $p_0 \in P_0$ and $p_1 \in P_1$ that $write\_set(x, p_0) \cap write\_set(x, p_1)$ contains at least one register both p_1 and p_0 write next but no other process in P writes next. There are a minimum of $|P_0||P_1|$ such registers. The result follows. □

5.3 Mutual Exclusion

We show that for systems that support mixed m-register operations there is a solution to the mutual exclusion which is more efficient in time and space than the solution presented in Theorem 3.

Theorem 7 *For any n and $m \geq 1$, in a system with n processes that supports atomic mixed m-register operations with a single write and $m-1$ reads, there is a starvation-free solution to the mutual exclusion problem with time complexity $\lceil \log_m n \rceil$.*

PROOF: We start with a simple starvation-free solution for m processes. There are m registers $r_1, ..., r_m$ taking on values from $\{0, 1, 2\}$, with values initially 0. On entering the trying region, each process p_i writes a 1 to r_i and reads the other $m-1$ registers. If all other registers are 0, then p_i enters the critical section. Otherwise, p_i reads r_i repeatedly until it reads the value 2, and then p_i enters the critical section. In the absence of contention, the cost is a single mixed m-register operation.

In the exit region, p_i writes a 0 to r_i and reads the other registers. If all other registers are 0, p_i is done. Otherwise, it picks a process p_j whose register is set to 1 and writes 2 to it. Each process keeps enough local information (outside the shared memory) to ensure that it will schedule its successors fairly, ensuring starvation-freedom. (A counter *mod* m will do—p_i need only pick p_j, where j is the smallest index greater than the counter such that r_j is 1, and p_i increments the counter each time through the critical section.) In the absence of contention, the cost is again a single mixed m-register operation. (In the presence of contention, the cost is 2 such operations.)

The m-process solution generalizes as in the proof of Theorem 3. Since only a single register assignment can be done in each operation, the exit region requires $\lceil \log_m n \rceil$ operations in the absence of contention (and $2\lceil \log_m n \rceil$ operations in the presence of contention). As before, all but one of these assignments (at the root) can be done in the trying region, the critical section, or the exit region. □

6 Discussion

We have presented several results for the consensus and the mutual exclusion problems in systems where it is possible to atomically access m shared registers. For mutual exclusion, we presented a speed-up of only $\log_2 m$, in a model which groups m register operations together into a single step. Is it possible to find a speed-up linear in m? It would also

be interesting to know whether these results generalize to related problems, such as set consensus [7] and ℓ-exclusion.

In [12], Lamport defined three general classes of shared read/write registers—safe, regular, and atomic—depending on their properties when several reads and/or writes are executed concurrently. The weakest possibility is *safe* register, in which it is assumed only that a read not concurrent with any writes obtains the correct value. A *regular* register is a safe register in which a read that overlaps a write obtains either the old or new value. An *atomic* register, is a safe register in which the reads and writes behave as if they occur in some definite order. Lamport showed that there are wait-free constructions of atomic registers from safe registers. In this paper we have considered only the atomic m-register case. It would be interesting to study regular or safe m-registers, to determine, for example, whether there are wait-free constructions of atomic m-registers from safe m-registers.

6.1 Acknowledgements

The authors would like to thank the anonymous referees. Their extremely detailed comments were of great help.

References

[1] Y. Afek, H. Attiya, D. Dolev, M. Gafni, M. Merritt, and N. Shavit. Atomic snapshots. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–13, August 1990. To appear in Journal of the ACM.

[2] J.H. Anderson. Composite registers. Technical Report TR-89-25, Department of Computer Science, The University of Texas at Austin, September 1989.

[3] J.H. Anderson. Multiple-writer composite registers. Technical Report TR-89-26, Department of Computer Science, The University of Texas at Austin, September 1989.

[4] B. Bloom. Constructing two-writer atomic registers. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 249–259, 1987.

[5] J. E. Burns and A. N. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*. To appear. An early version appeared in the Proceedings of 18th Annual Allerton Conference on Communication, Control and Computing, 1980, pages 833–842.

[6] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[7] S. Chaudhuri. More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.

[8] B. Chor, A. Israeli, and M. Li. On processor coordination using asynchronous hardware. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 86–97, 1987.

[9] R. Cole and Ofer Zajicek. The APRAM: Incorporating asynchrony into the PRAM model. In *Proc. of the 1989 ACM Symp. on Parallel algorithms and Architectures*, pages 169–178, June 1989.

[10] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[11] M. Herlihy. Wait-free synchronization. *ACM Trans. on Programming Languages and Systems*, 11(1):124–149, January 1991.

[12] L. Lamport. On interprocess communication, parts I and II. *Distributed Computing*, 1(2):77–101, 1986.

[13] L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. on Computer Systems*, 5(1):1–11, 1987.

[14] M. C. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[15] G. L. Peterson and J. E. Burns. Concurrent reading while writing ii: the multiwriter case. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pages 383–392, 1987.

[16] G. L. Peterson and M. J. Fischer. Economical solutions for the critical section problem in a distributed system. In *Proc. 9th ACM Symp. on Theory of Computing*, pages 91–97, 1977.

[17] G. Taubenfeld. On the nonexistence of resilient consensus protocols. *Information Processing Letters*, 37:285–289, 1991.

[18] G. Taubenfeld and S. Moran. Possibility and impossibility results in a shared memory environment. In *3rd International Workshop on Distributed Algorithms*, 1989. Lecture Notes in Computer Science, vol. 392 (eds.: J.C. Bermond and M. Raynal), Springer-Verlag 1989, pages 254–267.

[19] P. M. B. Vitanyi and B. Awerbuch. Atomic shared register access by asynchronous hardware. In *Proc. 27th IEEE Symp. on Foundations of Computer Science*, pages 223–243, 1986. Errata, Ibid., 1987.