Efficiency of Semi-Synchronous versus Asynchronous Systems: Atomic Shared Memory

Marios Mavronicolas[1],
Aiken Computation Laboratory,
Harvard University,
Cambridge, MA 02138.

May 1992

[1]Supported by ONR contract N00014-91-J-1981.

Abstract

The *s-session problem* is studied in *asynchronous* and *semi-synchronous* shared-memory systems, under a particular shared-memory communication primitive – *b-atomic registers*, – where $b > 1$ is an integer reflecting the communication bound in the model. A session is a part of an execution in which each of n processes takes at least one step; an algorithm for the s-session problem guarantees the existence of at least s disjoint sessions. The existence of many sessions guarantees a degree of interleaving which is necessary for certain computations. In the asynchronous model, it is assumed that the time between any two consecutive steps of any process is in the interval $[0, 1]$; in the semi-synchronous model, the time between any two consecutive steps of any process is in the interval $[c, 1]$ for some c such that $0 < c \leq 1$, the *synchronous* model being the special case where $c = 1$. All processes are initially synchronized and take a step at time 0.

Our main result is a tight (within a constant factor) lower bound of $1 + \min\{\lfloor \frac{1}{2c} \rfloor, \lfloor \log_b(n-1) - 1 \rfloor\}(s-2)$ for the time complexity of any semi-synchronous algorithm for the s-session problem. This result shows the inherent limitations on using timing information in shared–memory systems subject to communication bounds, and implies a time separation between semi-synchronous and asynchronous such systems.

1 Introduction

In shared-memory distributed systems, some number n of independent processes communicate by reading and writing to shared memory. Central to the programming of such systems are *synchronization* problems, where a process is required to guarantee that *all* processes have performed a particular set of steps. Naturally, the timing information available to processes has critical impact on the time complexity of synchronization.

Arjomandi, Fischer and Lynch ([1]) introduced the *session problem* to study the impact of timing information on the time complexity of synchronization. Roughly speaking, a *session* is a sequence of events that contains *at least* one step by each process. An algorithm for the *s-session problem* guarantees that each execution of the algorithm includes at least s disjoint sessions.

The session problem is an abstraction of the synchronization needed for the execution of some tasks that arise in a distributed system, where separate components are each responsible for performing a small part of a computation. Consider, for example, a system which solves a set of equations by successive relaxation, where every process holds part of the data (cf. [5]). Interleaving of steps by different processes is necessary in order to ensure that a correct value was computed, since it implies sufficient interaction among the intermediate values computed by the processes. Any algorithm which ensures that sufficient interleaving has occurred also solves the s-session problem. The session problem is also an abstraction of some problems in real-time computing which involve synchronization of several computer system components, in order that they cooperate in performing a task involving real-world components. For example, multiple robots might cooperate in moving machinery pieces around different sites of a large manufacturing system. This cooperation requires to synthesize a motion, through physical space, avoiding obstacles present therein while respecting certain dynamic constraints, such as given bounds on the velocity and acceleration. Interleaving of motion by different robots is necessary to ensure that pieces are delivered in the right order; a robot should deliver a particular machinery item early enough at a site before another robot arrives at the site to collect all machinery pieces delivered there. Clearly, any algorithm which ensures that sufficient motion interleaving has occurred also solves the s-session problem. Thus, the difficulty of solving the s-session problem reflects those of implementing the successive relaxation method and moving machinery pieces around in the manufacturing system.

Arjomandi, Fischer and Lynch ([1]) assumed that processes communicate via *shared variables* and studied the time complexity of the session problem in *synchronous* and *asynchronous* shared-memory systems. Informally, in a *synchronous* system, processes operate in lock-step,

taking steps simultaneously, while in an *asynchronous* system, processes work at completely independent rates and have no way to estimate time. The results of Arjomandi, Fischer and Lynch ([1]) show that there is a significant gap between the time complexities of solving the session problem in the synchronous and the asynchronous models.

In reality, however, there is an important middle ground between the synchronous and the asynchronous models of computation: in most distributed systems, processes operate neither at lock-step nor at a completely independent rate. For example, processes may have access to inaccurate clocks that operate at approximately, but not exactly, the same rate. Following [3], Attiya and Mavronicolas ([4]) modeled these *semi-synchronous* systems by assuming that there exist a lower and an upper bound on processes' step time that enable processes to estimate time; they addressed the cost of synchronization in semi-synchronous and asynchronous *networks* by presenting upper and lower bounds for the time complexity of solving the s-session problem. The results of Attiya and Mavronicolas imply a time separation between semi-synchronous and asynchronous networks. In this paper, we revisit the shared-memory model introduced by Arjomandi, Fischer and Lynch and address the effect of the timing assumptions in the semi-synchronous model on the time complexity of solving the s-session problem.

Informally, the *time complexity* of an algorithm is the maximal time, over all executions, until every process stops executing the algorithm. The following timing assumptions are made on the system. In the *asynchronous* model, processes' step time is in the range $[0, 1]$; in the *semi-synchronous* model, processes' step time is in the range $[c, 1]$, for some parameter c such that $0 < c \leq 1$[1]. Processes are initially synchronized and take a step at time 0. Following [1], we consider a particular shared-memory primitive, *b-atomic registers,* where the integer $b > 1$ is an upper bound on the number of processes that may instantaneously and indivisibly access (read and, possibly, modify) each of the registers. Thus, b reflects the communication bound in the model and captures communication limitations of existing distributed systems such as those of a message-passing system which accesses buffers of finite fan-in.

An algorithm sketched in [1] relies on explicit communication through shared memory to ensure that the needed steps have occurred and does not use any timing information. This algorithm achieves time complexity of $O(s \log_b n)$ in both the asynchronous and the semi-synchronous models. On the other hand, an algorithm presented in [4] (Theorem 3.8) does not use any communication, but relies on timing information and works only in the semi-synchronous model to achieve time complexity of $O(s\frac{1}{c})$. These two algorithms can be combined to yield a semi-synchronous algorithm for the s-session problem whose time complexity is

[1] The synchronous model is the special case of the semi-synchronous model where $c = 1$.

$O(s \min\{\frac{1}{c}, \log_b n\})$. On the other hand, a lower bound of $\Omega(s \log_b n)$ shown in [1] holds for our asynchronous model as well and implies, for appropriate values of the various parameters, a time separation between semi-synchronous and asynchronous systems that use communication through atomic shared memory.

At this point, it is natural to ask whether communication and timing information can be combined to yield an upper bound that is significantly better than $O(s \min\{\frac{1}{c}, \log_b n\})$. Our main result, a lower bound of $1 + \min\{\lfloor \frac{1}{2c} \rfloor, \lfloor \log_b(n-1) - 1 \rfloor\}(s-2)$ for the time complexity of any semi-synchronous algorithm for the s-session problem[2], shows the inherent limitations on using timing information and implies that such a combination is impossible.

As in [4], our main lower bound result can also be used to derive a lower bound of $1 + \lfloor \log_b(n-1) - 1 \rfloor (s-2)$ for a shared-memory model in which processes' step time is in the range $(0, 1]$ (rather than in $[0, 1]$, as in the asynchronous model). This is equivalent to requiring that two steps by the same process do not occur at the same time[3]. Fix some $c' > 0$ such that $\lfloor \frac{1}{2c'} \rfloor \geq \lfloor \log_b(n-1) - 1 \rfloor$, and use the proof of the lower bound for the model where the rate of processes steps is in $[c', 1]$; since $[c', 1] \subset (0, 1]$, the claim follows. This implies a time separation between this model and the synchronous shared-memory model. (Note that the proof in [1] relies heavily on the ability to schedule many steps by the same process at the same time.)

Our lower bound uses the same general approach as in [1] and [4]. Specifically, our proof combines fan-in and causality arguments as in [1], along with information propagation and careful timing arguments as in [4] .

The rest of this paper is organized as follows. Section 2 presents the system model and defines the session problem. Section 3 discusses some simple bounds for both the semi-synchronous and asynchronous models. Section 4 includes our main lower bound for the semi-synchronous model. We conclude, in Section 5, with a discussion and some open problems.

2 Definitions

In this section, we present the definitions for the underlying formal model[4], and define what it means for an algorithm to solve the *s-session problem*.

[2] An essentially identical lower bound has been obtained independently by Rhee and Welch ([14]).

[3] We remark that this is the most common way of measuring time in an asynchronous system (see, e.g., [12]).

[4] These definitions could be expressed in terms of the general *timed automaton model* described in [11, 3, 6].

2.1 The System Model

In this subsection, we describe the system model and the time measure we will consider. Our definitions are standard and are similar to the ones in, e.g., [4, 3, 2, 1, 12, 7, 9].

A *system* consists of n processes $p_1, \ldots, p_n$. Each process p_i is modeled as a (possibly infinite) state machine with state set Q_i. The state set Q_i contains a distinguished initial state $q_{0,i}$. The state set Q_i also includes a subset I_i of *idle* states; we assume $q_{0,i} \notin I_i$.

Processes communicate through *b-atomic registers* (also called *shared variables*), $R_1, R_2, \ldots$. Each shared variable may attain values from a *domain*, a set $\mathcal{V}$ of *values*, which includes a special undefined value $\perp$. Each process p_i has a single *read-modify-write* atomic operation available to it that may read a shared variable R, return its value v, and modify R. Associated with each shared variable R is a set $Access(R)$ that includes the processes which may perform atomic operations on R; we assume that for each R, $|Access(R)| \leq b$.

A system configuration consists of the states of the processes and the values of the shared variables. Formally, a *configuration* is a vector $C = \langle q_1, \ldots, q_n, v_1, v_2, \ldots \rangle$ where q_i is the local state of p_i and v_k is the value of the shared variable R_k; denote $state_i(C) = q_i$ and $value_k(C) = v_k$. Each shared variable may attain values from some *domain* $\mathcal{V}$ of *values* which includes a special "undefined" value, $\perp$. The *initial configuration* is the configuration in which every local state is an initial state and all shared variables are set to $\perp$.

We consider an interleaving model of concurrency, in the style of Lynch and Tuttle (cf. [9]), where computations of the system are modeled as sequences of *atomic events*, or simply *events*. Each event is a *computation event* representing a computation step of a single process; it is specified by $comp(i, R)$ for some $i \in [n]$ and some shared variable R. In this computation step, the process p_i, based on its local state, performs an operation on the shared variable R, performs some local computation, and changes to its next state.

Each process p_i follows a deterministic local algorithm $\mathcal{A}_i$ that determines p_i's local computation, i.e., the register to be accessed and the state transition to be performed. More specifically, $\mathcal{A}_i$ determines:

- A shared variable R as a function of p_i's local state.
- Whether p_i is to modify R and, if so, the value v' to be written and p_i's next state, as a function of p_i's local state and the value v read from R.

We assume that when a proces enters an idle state, it will remain in an idle state. An *algorithm* (or a *protocol*) is a sequence $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ of local algorithms.

An *execution* is an infinite sequence of alternating configurations and steps

$$\alpha = C_0, \pi_1, C_1, \ldots, \pi_j, C_j, \ldots ,$$

satisfying the following conditions:

1. C_0 is the initial configuration;
2. If $\pi_j = comp(i, R_k)$, then R_k is obtained by applying $\mathcal{A}_i$ to $state_i(C_{j-1})$, and $state_i(C_j)$ and $value_k(C_j)$ are obtained by applying $\mathcal{A}_i$ to $state_i(C_{j-1})$ and $value_k(C_{j-1})$;
3. If π_j involves process p_i and shared variable R_k, then $state_l(C_{j-1}) = state_l(C_j)$ for every $l \neq i$ and $value_l(C_{j-1}) = value_l(C_j)$ for every $l \neq k$.

That is, in an execution the changes in processes' states and shared variables' values are according to the local algorithms, only a process which takes a step changes its state, and only a shared variable on which an operation is performed changes its value. We adopt the convention that finite prefixes of an execution end with a configuration, and denote the last configuration in a finite execution prefix α by $last(\alpha)$. We say that $\pi_j = comp(i, R)$ is a *non-idle step* of the execution if $state_i(C_{j-1}) \notin I_i$, i.e., it is taken from a non-idle state.

A *timed event* is a pair (t, π), where t, the "time", is a nonnegative real number, and π is an event. A *timed sequence* is an infinite sequence of alternating configurations and timed events

$$\alpha = C_0, (t_1, \pi_1), C_1, \ldots, (t_j, \pi_j), C_j, \ldots ,$$

where the times are nondecreasing and unbounded.

In our model, timed executions are defined as follows. Fix a real number c such that $0 \leq c \leq 1$. Letting α be a timed sequence as above, we say that α is a *timed execution* of $\mathcal{A}$ provided that the following all hold:

1. $C_0, \pi_1, C_1, \ldots, \pi_j, C_j, \ldots$ is an execution of $\mathcal{A}$;
2. (Synchronous start) There are computation steps for all processes with time 0.
3. (Upper bound on step time) If the jth timed event is $(t_j, comp(i_j, R))$, then there exists a $k > j$ with $t_k \leq t_j + 1$ such that the kth timed event is $(t_k, comp(i_j, R'))$;
4. (Lower bound on step time) If the jth timed event is $(t_j, comp(i_j, R))$, then there does not exist a $k > j$ with $t_k < t_j + c$ such that the kth timed event is $(t_k, comp(i_j, R'))$.

We say that α is an *execution fragment* of $\mathcal{A}$ if there is an execution α' of $\mathcal{A}$ of the form $\alpha' = \beta\alpha\beta'$. This definition is extended to apply to timed executions in the obvious way. For a finite execution fragment $\alpha = C_0, (t_1, \pi_1), C_1, \ldots, (t_k, \pi_k), C_k$, we define $t_{start}(\alpha) = t_1$ and $t_{end}(\alpha) = t_k$.

The *asynchronous* model is defined by taking $c = 0$, while the *semi-synchronous* model is defined by taking $0 < c \leq 1$; the *synchronous model* is the special case of the latter where $c = 1$. Note that the asynchronous model allows, as defined above, two computation steps of the same process to occur at the same time (Condition 4 is vacuous when $c = 0$). If we want to define the more common asynchronous model where a process can have at most one computation step at each time, we have to replace Condition 4 above with:

> (Lower bound on step time) If the jth timed event is $(t_j, comp(i_j, R))$, then there does not exist a $k > j$ with $t_k = t_j$ such that the kth timed event is $(t_k, comp(i_j, R'))$.

In both models, we say that a process p_i *enters an idle state by time* t' (in a timed execution α) if there exists a timed event (t_{j-1}, π_{j-1}) in α such that $t_{j-1} \leq t'$, $\pi_{j-1} = comp(i, R)$ and $state_i(C_j) \in I_i$.

2.2 The Session Problem

An execution fragment $C_1, \pi_1, C_2 \ldots, \pi_m, C_m$ is a *session* if for each i, $i \in [n]$, there exists at least one event $\pi_j = comp(i)$, for some $j \in [m]$, which is a non-idle step of the underlying execution. Intuitively, a session is an execution fragment in which each process takes at least one non-idle step. An execution α *contains* s *sessions* if it can be partitioned into at least s disjoint execution fragments such that each of them is a session. These definitions are extended to apply to timed executions in the obvious way.

An algorithm *solves the* s*-session problem within time* t if each of its timed executions α satisfies the following: α contains s sessions and all processes enter an idle state no later than time t in α.

3 Simple Bounds

In this section, we briefly mention some simple algorithms and lower bounds for the s-session problem from previous work ([1, 4]) that also hold for the asynchronous and semi-synchronous models considered in this paper.

For the asynchronous model where there is no lower bound on processes' step time, the lower bound proof in [1], relying on the ability to schedule many steps by the same process at the same time, still works to yield a lower bound of $\Omega(s \log_b n)$[5]. Also, the "tree network" algorithm sketched in [1] (Section 4) still works in our model. The "tree network" algorithm relies entirely on explicit communication between processes to ensure that the needed steps have occurred and does not use any timing information. Roughly speaking, this algorithm consists of building up a "tree"out of b-atomic registers, whose leaves are the n processes. Neglecting roundoffs, this network has depth $\log_b n$. Processes communicate through this network in order to learn about completion of a session before advancing to the next session. Thus, the necessary communication for one session can be accomplished in time $O(\log_b n)$ and the total time for all processes to enter an idle state after performing s sessions is $O(s \log_b n)$ in both the asynchronous and the semi-synchronous models.

On the other hand, an algorithm which relies entirely on timing information and does not use any communication[6] is one presented for the semi-synchronous network model in [4] (Theorem 3.8) which still works for the semi-synchronous shared-memory model considered in this paper. This algorithm exploits the timing information available in the semi-synchronous model to obtain a bound which is sometimes better than the bound of the "tree network" algorithm. Roughly speaking, in this algorithm each process takes about $s\lfloor \frac{1}{c} \rfloor$ computation steps before entering an idle state.

It is possible to run the two previous algorithms "side by side," halting when the first of them does, and get a bound of $O(s \min\{\frac{1}{c}, \log_b n\})$ for the s-session problem in the semi-synchronous shared-memory model. Note that, by an appropriate choice of the various parameters, this upper bound and the $\Theta(s \log_b n)$ tight bound for the asynchronous model together imply a time separation between semi-synchronous and asynchronous shared-memory models.

4 Main Result

We show that communication and timing information *cannot* be combined to yield an upper bound that is significantly better than the $O(s \min\{\frac{1}{c}, \log_b n\})$ upper bound discussed in Section 3.

[5]Note that in [1], the asynchronous model is defined in a slightly different way than ours, more specifically by having all infinite admissible computations be allowable, and puts no restriction on the number of steps a process takes at a time.

[6]This means that no state transition can result in an operation on a shared variable.

In our lower bound proof, we use an infinite timed execution in which processes take steps in round-robin order, starting with p_1, with step time equal to 1. It is a called a *slow, synchronous* timed execution. We have:

Theorem 4.1 *There does not exist a semi-synchronous algorithm which solves the s-session problem within time strictly less than* $1 + \min\{\lfloor \frac{1}{2c} \rfloor, \lfloor \log_b(n-1) - 1 \rfloor\}(s-2)$.

Proof: Assume, by way of contradiction, that there exists a semi-synchronous algorithm, $\mathcal{A}$, which solves the s-session problem within time strictly less than $1 + \min\{\lfloor \frac{1}{2c} \rfloor, \lfloor \log_k(n-1) - 1 \rfloor\}(s-2)$. We construct a timed execution of $\mathcal{A}$ which does not include s sessions.

We start with a slow, synchronous timed execution of $\mathcal{A}$ and partition it into an execution fragment containing the events at time 0 and at most $s-2$ other execution fragments each of which is completed within time $< \min\{\lfloor \frac{1}{2c} \rfloor, \lfloor \log_b(n-1) - 1 \rfloor\}$. We use causality and fan-out arguments to argue that there is no communication through shared memory between a certain pair of processes within each fragment. Furthermore, since the execution is slow, a process takes, roughly, at most $\frac{1}{2c}$ steps in each fragment, so it is possible to have all these steps occur while another process takes only one step. By "retiming", we will perturb each fragment to get a new one in which there is a "fast" process which takes all of its steps before a "slow" process takes any of its steps. The part of the proof that shows that the "retimed" execution preserves the timing constraints of the semi-synchronous model requires to choose the execution fragments to take time $< \lfloor \frac{1}{2c} \rfloor$, so that it will be possible for a process to not take a computation step during a large part of the execution. Our construction will have the "fast" process of each execution fragment be identical to the "slow" process of the next execution fragment. This will guarantee that at most one session is completed in each execution fragment. Thus, the total number of sessions in the "retimed" execution is at most $s-1$, contradicting the correctness of $\mathcal{A}$.

We now present the details of the formal proof.

Denote $e = \min\{\lfloor \frac{1}{2c} \rfloor, \lfloor \log_b(n-1) - 1 \rfloor\}$.

If $e \leq 1$, then the lower bound we are trying to prove is $\leq 1 + 1(s-2) = s-1$. Since s steps of each process are necessary if s sessions are to occur and they can occur 1 time unit apart, it follows that $s-1$ is a lower bound. Thus, we assume, without loss of generality, that $e > 1$.

Let γ be a slow, synchronous timed execution of $\mathcal{A}$. Assume $\gamma = \alpha_0 \alpha \alpha'$, where α_0 contains only events that occur at time < 1, $\alpha_0 \alpha$ is the shortest prefix of γ such that all processes are in

Figure 1 should appear here.

Figure 1: The timed execution $\alpha_0\alpha\alpha'$

an idle state in $last(\alpha_0\alpha)$, and α' is the remaining part of γ. Denote $T = t_{end}(\alpha_0\alpha)$. Since γ is slow and s steps of each process are necessary to guarantee s sessions, $T \geq s-1$. Since $\mathcal{A}$ solves the s-session problem within time strictly less than $1+e(s-2)$, it follows that $T < 1+e(s-2)$. Note that, by construction, $t_{start}(\alpha) = 1$. Thus, $t_{end}(\alpha) - t_{start}(\alpha) = T - 1 < e(s-2)$. Denote $s' = \lceil \frac{T-1}{e} \rceil$; it follows that $s' \leq s-2$.

We write $\alpha = \alpha_1\alpha_2 \ldots \alpha_{s'}$, where:

- For each k, $1 \leq k < s'$, α_k contains all events that occur at time t, where $1+(k-1)e \leq t < 1+ke$, and
- $\alpha_{s'}$ contains all events occurring at time t, where $1+(s'-1)e \leq t \leq T$.

That is, we partition α into execution fragments, each taking time $< e$.

Figure 1 depicts the timed execution $\alpha_0\alpha\alpha'$. Each horizontal line represents events happening at one process. We use the symbol $\bullet$ to mark non-idle process steps; similarly, we use the symbol $\times$ to mark idle process steps. Dashed vertical lines mark time points that are used in the proof.

We reorder and retime events in α to obtain a timed sequence β and reorder and retime events in α' to obtain a timed sequence β', such that $\alpha_0\beta\beta'$ is a timed execution of $\mathcal{A}$ that does not include s sessions.

In our construction, we will use a partial order $\leq_\alpha$, representing "causality", on the computation steps that processes take in α. We start by defining $\leq_\alpha$. For every pair of steps π_1, π_2 in α, we let $\pi_1 \leq_\alpha \pi_2$ if $\pi_1 = \pi_2$ or if π_1 precedes π_2 in α and either π_1 and π_2 are steps taken by the same process or by different processes, but on the same shared variable. Close $\leq_\alpha$ under transitivity. $\leq_\alpha$ is a partial order, and every total order of computation steps in α consistent with $\leq_\alpha$ represents a computation which leaves the system in the same configuration as α. (Clearly, α itself provides such a total order.)

We first show how to modify α to obtain an execution fragment $\beta = \beta_1\beta_2 \ldots \beta_{s'}$ that includes at most $s' \leq s-2$ sessions. For some sequence $p_{i_0}, \ldots, p_{i_{s'}}$ of processes, we construct from each execution fragment α_k an execution fragment $\beta_k = \rho_k\sigma_k$, such that:

(1) ρ_k contains no computation step of $p_{i_{k-1}}$, and

(2) σ_k contains no computation step of p_{i_k}.

In this construction, p_{i_k} is the "fast" process which takes all its steps in ρ_k, before the "slow" process $p_{i_{k-1}}$ takes any of its steps. (All the steps of $p_{i_{k-1}}$ are in σ_k.)

For each k, $1 \leq k \leq s'$, we show how to construct β_k inductively. For the base case, let p_{i_0} be an arbitrary process.

Assume we have picked $p_{i_0}, \ldots, p_{i_{k-1}}$ and constructed $\beta_1, \ldots, \beta_{k-1}$. We first show that there exists some process such that a communication between it and $p_{i_{k-1}}$ cannot be established in α_k.

Lemma 4.2 *Let π_1 be the first step of $p_{i_{k-1}}$ in α_k. There is some process of which there is no computation step σ in α_k such that $\pi_1 \leq_\alpha \sigma$.*

Proof: Clearly, it suffices to show that the number of steps τ in α_k such that $\pi \leq_\alpha \tau$, where π is any step of $p_{i_{k-1}}$ in α_k, is at most $n-1$. We proceed to count the number of such steps.

By construction,

$$t_{end}(\alpha_k) - t_{start}(\alpha_k) < 1 + ke - 1 - (k-1)e = e.$$

Let m be the maximum number of steps over all processes that some process takes within α_k. Since α is a slow execution,

$$m \leq \lceil t_{end}(\alpha_k) - t_{start}(\alpha_k) \rceil \leq \lceil e \rceil \leq \lceil \lfloor \log_b(n-1) - 1 \rfloor \rceil = \lfloor \log_b(n-1) - 1 \rfloor.$$

Clearly, the number of steps τ taken by any process in α_k such that $\pi_i \leq_\alpha \tau$, where π_i is the ith step of $p_{i_{k-1}}$ in α is at most b^{m-i+1}. Thus, the number of steps τ in α_k such that $\pi \leq_\alpha \tau$, where π is any step of $\pi_{i_{k-1}}$ is at most:

$$\sum_{i=1}^{m} b^{m-i+1} = b \sum_{i=0}^{m-1} b^i = b\frac{b^m - 1}{b-1} \leq b^{m+1} \leq b^{\lfloor \log_b(n-1)-1 \rfloor + 1} \leq b^{\log_b(n-1)} = n-1.$$

The claim follows. ∎

Fix p_{i_k} to be any process such that a communication between $p_{i_{k-1}}$ and p_{i_k} is not established in α_k. We now show how to construct β_k. For any process u, σ_k includes all steps τ of u in

Figure 2 should appear here.

Figure 2: The timed execution $\alpha_0\beta\beta'$

α_k such that $\pi \leq_\alpha \tau$, where π is any step of $\pi_{i_{k-1}}$ in α_k; ρ_k includes all remaining steps of u in α_k. Steps at each process occur in the same order as in α_k and all occur at step time of c, in both ρ_k and σ_k. In addition, ordering of steps by different processes that occur at the same time in α_k is preserved within each of ρ_k and σ_k. By Lemma 4.2, there is no step σ of p_{i_k} in α_k such that, for some step π of $p_{i_{k-1}}$ in α_k, $\pi \leq_\alpha \sigma$. This implies that all steps of p_{i_k} in α_k will appear in ρ_k. On the other hand, since $\pi \leq_\alpha \pi$ for any step π of $p_{i_{k-1}}$ in α_k, all steps of $p_{i_{k-1}}$ in α_k will appear in σ_k. Thus, $\beta_k = \rho_k\sigma_k$ has properties (1) and (2) above.

To complete our construction, we assign times to steps in β_k. Let $t_{start}(\rho_1) = c$. The first and last steps of p_{i_k} in ρ_k occur at times $t_{start}(\rho_k) = t_{end}(\sigma_{k-1}) + c$ and $t_{end}(\rho_k)$, respectively. Similarly, the first and last steps of $p_{i_{k-1}}$ in σ_k occur at times $t_{start}(\sigma_k) = t_{end}(\rho_k)$ and $t_{end}(\sigma_k)$, respectively. Steps are taken c time units apart. For each process p_j, we schedule each step π_j of p_j in ρ_k to occur simultaneously with a step, π_{i_k}, of p_{i_k} which is such that π_j and π_{i_k} occurred at the same time in α_k. Similarly, for each process p_j, we schedule each step π_j of p_j in σ_k to occur simultaneously with a step, $\pi_{i_{k-1}}$, of $p_{i_{k-1}}$ which is such that π_j and $\pi_{i_{k-1}}$ occurred at the same time in α_k. We will shortly show that assigning times in this manner is consistent with the requirements for a timed execution.

We now modify α' to obtain β'. The first computation step of any process in β' will occur at time c after its last computation step in β and all later computation steps of it will occur c time units apart in β'.

Figure 2 depicts the timed execution $\alpha_0\beta\beta'$ using the same conventions as in Figure 1.

We remark that what allowed us to "separate" the steps of $p_{i_{k-1}}$ from those of p_{i_k} in each of the execution fragments was the assumption that the length of each execution fragment is less than $\lfloor \log_b(n-1) - 1 \rfloor$ which, due to the communication limitations of the model, is not enough to guarantee that a process can "affect" at least one step of every other process.

We next establish that $\alpha_0\beta\beta'$ is a timed execution of $\mathcal{A}$. We start by showing:

Lemma 4.3 *Ordering of computation steps operating on the same shared variable is preserved in* $\alpha_0\beta\beta'$.

Proof: Let π_1 and π_2 be computation steps operating on the same shared variable in α_k, such that $\pi_1 \leq_\alpha \pi_2$. The only non-trivial case is when π_1 and π_2 occur in the same α_k, for some k, $1 \leq k \leq s'$. We show that the ordering of π_1 and π_2 is the same in β_k as in α_k.

The only case of interest is when π_1 occurs in σ_k, while π_2 occurs in ρ_k. By construction, there is some step π_1' of $p_{i_{k-1}}$ in α_k such that $\pi_1' \leq_\alpha \pi_1$, while there is no step π_2' of $p_{i_{k-1}}$ in α_k such that $\pi_2' \leq_\alpha \pi_2$. But, from $\pi_1' \leq_\alpha \pi_1$ and $\pi_1 \leq_\alpha \pi_2$, it follows, by transitivity, that $\pi_1' \leq_\alpha \pi_2$. A contradiction. ■

Before showing that the timing constraints are preserved in $\alpha_0\beta\beta'$, we prove the following simple fact:

Claim 4.4 *For any k, $1 \leq k \leq s'-1$, $t_{end}(\rho_{k+1}) - t_{end}(\rho_k) \leq 1-c$.*

Proof: We first show that for any k, $1 \leq k \leq s'-1$, $t_{end}(\rho_{k+1}) - t_{end}(\beta_k) \leq \frac{1}{2}$, and for any k, $1 \leq k \leq s'$, $t_{end}(\beta_k) - t_{end}(\rho_k) \leq \frac{1}{2} - c$.

Fix some k, $1 \leq k \leq s'$. Recall that, by construction,

$$t_{end}(\alpha_k) - t_{start}(\alpha_k) < 1 + ke - 1 - (k-1)e = e \leq \lfloor \frac{1}{2c} \rfloor .$$

Let m be the maximum number of steps over all processes that some process takes within α_k.

Since both $t_{start}(\alpha_k)$ and $t_{end}(\alpha_k)$ are integral, $t_{end}(\alpha_k) - t_{start}(\alpha_k) \leq \lfloor \frac{1}{2c} \rfloor - 1$; then, since α is a slow execution,

$$m \leq t_{end}(\alpha_k) - t_{start}(\alpha_k) + 1 \leq \lfloor \frac{1}{2c} \rfloor \leq \frac{1}{2c}.$$

Let n_k be the number of steps of process $p_{i_{k-1}}$ in α_k and n_{k+1} be the number of computation steps of process $p_{i_{k+1}}$ in α_{k+1}. (Recall that, by construction, in β_k, $p_{i_{k-1}}$ will have all of its steps in σ_k, while in β_{k+1}, $p_{i_{k+1}}$ will have all of its steps in ρ_{k+1}.) Thus,

$$t_{end}(\rho_{k+1}) - t_{end}(\beta_k) = n_{k+1}c \leq mc \leq \frac{1}{2c}c = \frac{1}{2}.$$

Also, since $p_{i_{k-1}}$ takes n_k steps in σ_k with the first and last occurring at times $t_{start}(\sigma_k) = t_{end}(\rho_k)$ and $t_{end}(\sigma_k) = t_{end}(\beta_k)$, respectively, we have:

$$t_{end}(\beta_k) - t_{end}(\rho_k) = (n_k - 1)c \leq (m-1)c \leq (\frac{1}{2c} - 1)c = \frac{1}{2} - c \ .$$

Now, we have

$$t_{end}(\rho_{k+1}) - t_{end}(\rho_k) = t_{end}(\rho_{k+1}) - t_{end}(\beta_k) + t_{end}(\beta_k) - t_{end}(\rho_k) \leq \frac{1}{2} + \frac{1}{2} - c = 1 - c,$$

as needed. ■

We next show:

Lemma 4.5 *Lower and upper bounds on step time are preserved in $\alpha_0\beta\beta'$.*

Proof: By construction, no two computation steps are closer than c in $\alpha_0\beta\beta'$; so, the lower bound on step time is preserved. Note also that the time difference between consecutive computation steps of a process is maximized when the process is some p_{i_k}, for some k such that $1 \leq k \leq s' - 1$, that has no computation steps in either σ_k or ρ_{k+1}. By Claim 4.4, this time difference is less than or equal to 1. ■

This completes the proof that $\alpha_0\beta\beta'$ is a timed execution. To derive a contradiction, we finally prove:

Lemma 4.6 *There are at most s' sessions in β.*

Proof: We show, by induction on k, that $\beta_0 \ldots \beta_{k-1}\rho_k$ does not contain k sessions, for $1 \leq k \leq s'$. (By convention, β_0 denotes the *empty execution.*)

For the base case, note that, by construction, ρ_1 does not include a computation step of p_{i_0}. Thus, $\beta_0\rho_1$ cannot contain one session.

For the induction step, assume that the claim holds for $k-1$, i.e., $\beta_0 \ldots \beta_{k-2}\rho_{k-1}$ does not contain $k-1$ sessions for $1 \leq k \leq s'$. Hence, the kth session does not start within $\beta_0 \ldots \beta_{k-2}\rho_{k-1}$. Since neither σ_{k-1} nor ρ_k contains a computation step of $p_{i_{k-1}}$, $\sigma_{k-1}\rho_k$ does not contain a session. Thus, $\beta_0 \ldots \beta_{k-1}$ does not contain k sessions.

To complete the proof, note that $\sigma_{s'}$ does not contain a session since, by construction, it does not contain a computation step of $p_{i_{s'}}$. ■

Lemma 4.6 implies that β contains at most $s' \leq s - 2$ sessions; also, α_0 contains exactly one session. Therefore, there are at most $s - 1$ sessions in $\alpha_0\beta$. Since in β' no process takes a non-idle step, there is no additional session in β'. Thus, there are strictly less than s sessions in $\alpha_0\beta\beta'$. A contradiction. ■

We remark that the general structure of our proof closely follows [1, 4]. It uses causality arguments as in [1] to reorder the steps in the execution and presents an explicit retiming of them as in [4].

5 Discussion and Future Research

We showed a lower bound of $1+\min\{\lfloor\frac{1}{2c}\rfloor, \lfloor\log_b(n-1)-1\rfloor\}(s-1)$ on the time complexity of the s-session problem in a realistic semi-synchronous, shared-memory model. Neglecting round-offs, this lower bound is no less than 1/2 of the simple (combined) upper bound described in Section 3. This lower bound shows the inherent limitations on using timing information in systems where communication is achieved through atomic shared memory.

This work continues the study of time bounds in the presence of timing uncertainty within the framework of the semi-synchronous model ([2, 3, 4, 13]). Our results give a time separation between semi-synchronous and asynchronous shared-memory systems. Like the results in [4] (and unlike the separation results in [1]), our results do not rely on the ability to schedule several steps by the same process at the same real time.

It would be very interesting to study the relative differences in efficiency between asynchronous and semi-synchronous shared-memory systems supporting different, possibly weaker, primitives such as *regular* or *safe* registers (cf. [8], Lecture 12), still subject to communication bounds. Preliminary steps in this direction already appear in [10], where an upper bound l is assumed on the worst-case response time of performing an operation on such registers. These results suggest that the time bounds of performing tasks in semi-synchronous shared-memory models critically depend on the strength of the available primitives of communication through shared memory.

A more general direction is to study similar problems in the semi-synchronous, shared-memory model. The *consensus problem* (cf. [2]) is a good such candidate; a first step would naturally be to design a "timeout" strategy for detecting faulty processes (see Section 3 in [2]), assuming that communication is done through b-atomic registers.

As in [4], our results show that there are some timing-based algorithms for which any asynchronous simulation incurs a (non-constant) time overhead dependent on communication parameters of the model like, e.g., b or d, the message delay uncertainty of the semi-synchronous network model in [4]. It would be of extreme importance to characterize the timing-based algorithms whose overhead cost of asynchronous simulation is independent of particular communication parameters.

Acknowledgements:

I am grateful to Professor Harry R. Lewis for his helpful comments and criticisms, and to Jennifer Welch for some insightful comments.

References

[1] E. Arjomandi, M. Fischer and N. Lynch, "Efficiency of Synchronous versus Asynchronous Distributed Systems," *Journal of the ACM*, Vol. 30, No. 3 (1983), pp. 449–456.

[2] H. Attiya, C. Dwork, N. Lynch and L. Stockmeyer, "Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty," in *Proceedings of the 23rd ACM Symp. on Theory of Computing*, 1991, pp. 359–369. Also: Technical Memo MIT/LCS/TM-435, Laboratory for Computer Science, MIT, November 1990.

[3] H. Attiya and N. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," in *Proceedings of the 10th Real-Time Systems Symp.*, pp. 268–284, December 1989. Also: Technical Memo MIT/LCS/TM-403, Laboratory for Computer Science, MIT, July 1989.

[4] H. Attiya and M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Networks," in *Proceedings of the 28th annual Allerton Conference on Communication, Control and Computing*, October 1990, pp. 578–587. Expanded version: Technical Report TR-21-90, Aiken Computation Laboratory, Harvard University, September 1990.

[5] G. M. Baudet, "Asynchronous Iterative Methods for Multi-Processors," *Journal of the ACM*, Vol. 25, No. 2 (April 1978), pp. 226-244.

[6] N. Lynch and H. Attiya, "Using Mappings to Prove Timing Properties," in *Proceedings of the 9th Annual ACM Symp. on Principles of Distributed Computing*, pp. 265-280, August 1990. Also: Technical Memo MIT/LCS/TM-412.d, Laboratory for Computer Science, MIT, October 1991.

[7] N. Lynch and M. Fischer, "On Describing the Behavior and Implementation of Distributed Systems," *Theoretical Computer Science*, Vol. 13, pp. 17–43, 1981.

[8] N. Lynch and K. Goldman, *"Distributed Algorithms (Lecture Notes for 6.852),"* Research Seminar Series MIT/LCS/RSS 5, Laboratory for Computer Science, MIT, May 1989.

[9] N. Lynch and M. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," in *Proceedings of the 6th Annual ACM Symp. on Principles of Distributed Computing*, pp. 137–151, August 1987. Also: Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, MIT.

[10] M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Systems: Non-Atomic Shared Memory," in preparation, Harvard University, 1992.

[11] M. Merritt, F. Modugno and M. Tuttle, "Time Constrained Automata," in *Proceedings of the 2nd International Conference on Concurrency,* Amsterdam, Lecture Notes in Computer Science (Vol. 527), pp. 408–423, Springer-Verlag, 1991.

[12] G. Peterson and M. Fischer, "Economical Solutions for the Critical Section Problem in a Distributed System," in *Proceedings of the 9th ACM Symp. on Theory of Computing,* 1977, pp. 91–97.

[13] S. Ponzio, *"The Real-Time Cost of Timing Uncertainty: Consensus and Failure Detection,"* Technical Report MIT/LCS/TR-518, Laboratory for Computer Science, MIT, October 1991.

[14] I. Rhee and J. Welch, "The Impact of Time on the Session Problem," manuscript.

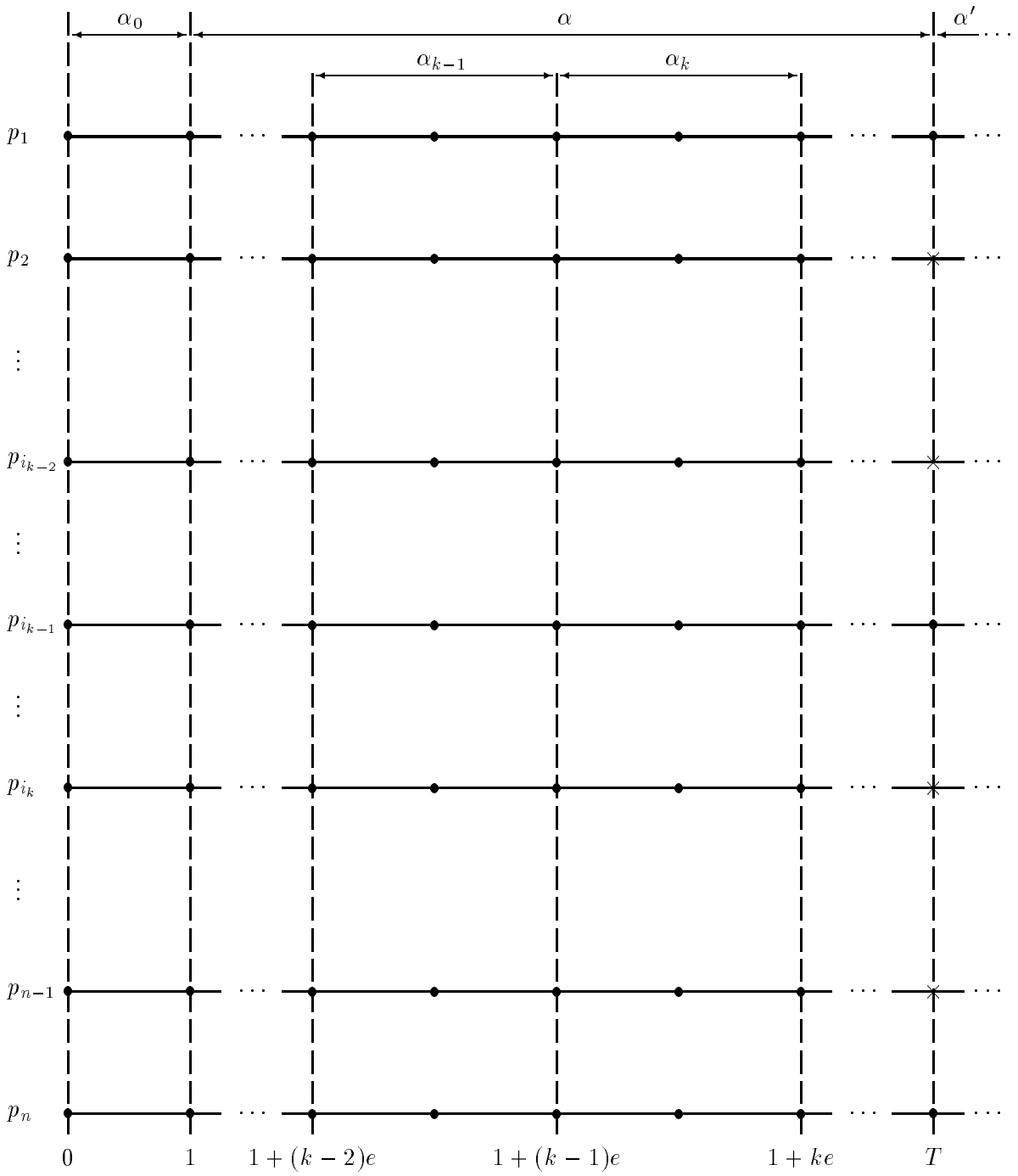
α_0
α
α'
α_{k-1}
α_k
p_1
p_2
$p_{i_{k-2}}$
$p_{i_{k-1}}$
p_{i_k}
p_{n-1}
p_n
0
1
$1+(k-2)e$
$1+(k-1)e$
$1+ke$
T

Figure 1

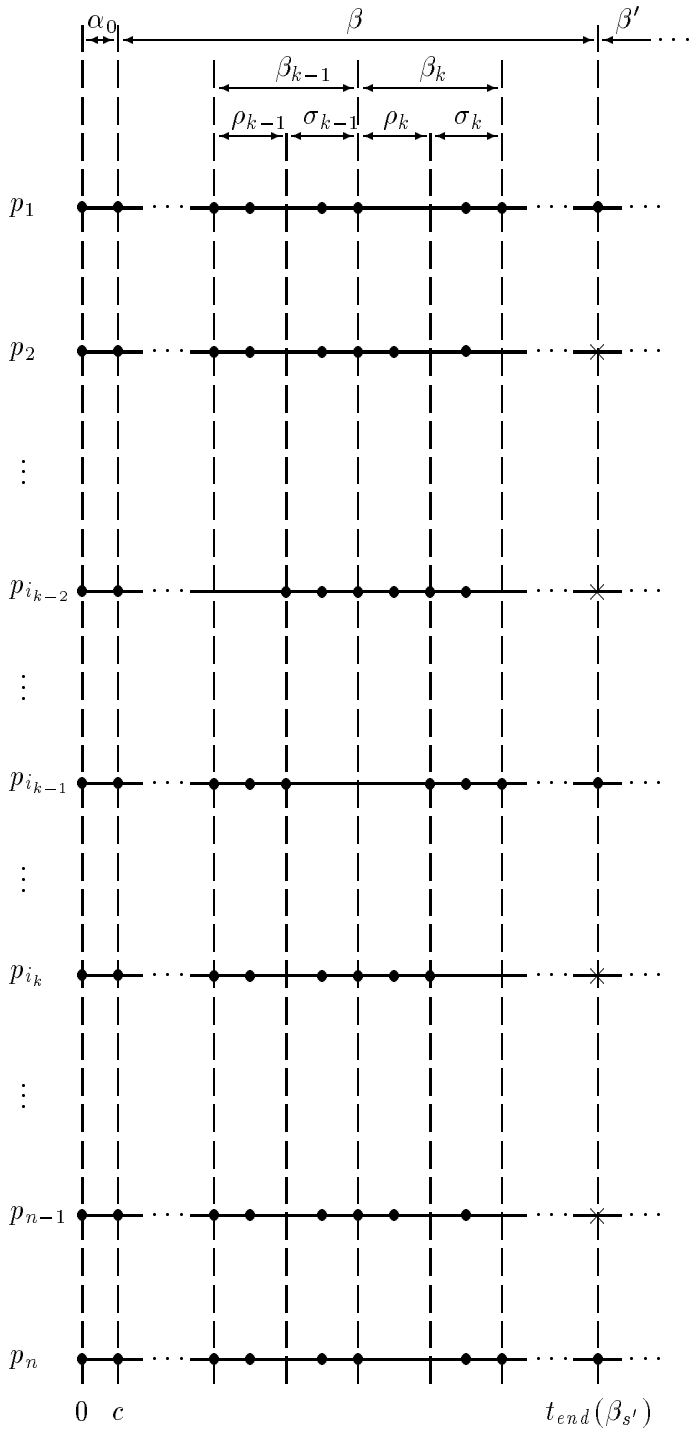
α_0
β
β'
β_{k-1}
β_k
ρ_{k-1}
σ_{k-1}
ρ_k
σ_k
p_1
p_2
$p_{i_{k-2}}$
$p_{i_{k-1}}$
p_{i_k}
p_{n-1}
p_n
0
c
$t_{end}(\beta_{s'})$

Figure 2