Combining Shared Coin Algorithms

James Aspnes* Hagit Attiya† Keren Censor ‡

Abstract

This paper shows that shared coin algorithms can be combined to optimize several complexity measures, even in the presence of a strong adversary. By combining shared coins of Bracha and Rachman [10] and of Aspnes and Waarts [7], this yields a shared coin algorithm, and hence, a randomized consensus algorithm, with $O(n \log^2 n)$ individual work and $O(n^2 \log n)$ total work, using single-writer registers. This improves upon each of the above shared coins (where the former has a high cost for individual work, while the latter reduces it but pays in the total work), and is currently the best for this model.

Another application is to prove a construction of Saks, Shavit, and Woll [16], which combines a shared coin algorithm that takes $O(1)$ time in failure-free executions, with one that takes $O(\log n)$ time in executions where at most $\sqrt{n}$ process fail, and another one that takes $O(\frac{n^3}{n-f})$ time in any other execution.

1 Introduction

Coordinating the actions of processes is crucial for virtually all distributed applications, especially in asynchronous systems. At the core of many coordination problems is the need to reach *consensus* among processes, despite the possibility of process failures. A consensus algorithm is a distributed algorithm where n processes collectively arrive at a common decision value starting from individual process inputs. It must satisfy *agreement* (all processes decide on the same value), *validity* (the decision value is an input to some process), and *termination* (all processes eventually decide). Unfortunately, there is no deterministic consensus algorithm in an asynchronous system, if one process may fail [11, 13, 14]. Luckily, reaching consensus becomes possible using randomization with the termination condition relaxed to hold with probability 1. (The agreement and validity properties remain the same.) The complexity of solving consensus is measured by the expected number of steps performed by *all* processes (*total work*) or by a single process (*per-process* or *individual work*).

*Supported in part by NSF grant CNS-0435201. Department of Computer Science, Yale University, New Haven, CT. aspnes@cs.yale.edu

†Supported in part by the *Israel Science Foundation* (grant number 953/06). Department of Computer Science, Technion, Haifa, Israel. hagit@cs.technion.ac.il

‡Supported in part by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities, and the *Israel Science Foundation* (grant number 953/06). Department of Computer Science, Technion, Haifa, Israel. ckeren@cs.technion.ac.il

Many randomized consensus algorithms have been suggested, in different communication models and under various assumptions about the adversary (see [3]). Virtually all of them rely on a *shared-coin* algorithm [6], in which each process outputs a value -1 or $+1$, and for every $v \in \{-1, +1\}$, there is a probability of at least δ that all processes output the same value v; δ is the *agreement parameter* of the algorithm. Aspnes and Herlihy [6] have shown that a shared-coin algorithm with agreement parameter δ, expected individual work I, and expected total work T, yields a randomized consensus algorithm with expected individual work $O(n + I/\delta)$ and expected total work $O(n^2 + T/\delta)$.

Various shared coin algorithms have been proposed, each using different communication primitives or optimizing a different complexity measure. Abrahamson [1] presented a randomized algorithm for solving consensus in asynchronous systems using shared memory, which has an exponential running time. The first polynomial algorithm for solving randomized consensus under the control of a strong adversary, which can observe the results of local coin-flips before scheduling the next process, was presented by Aspnes and Herlihy [6]. They described an algorithm that uses a shared coin in order to reach agreement and has a total work of $O(n^4)$. The amount of memory required by this algorithm was later bounded by Attiya, Dolev, and Shavit [9]. Aspnes [2] presented an algorithm for randomized consensus with $O(n^4)$ total work, which also uses bounded space. These algorithms were followed by an algorithm of Saks, Shavit and Woll [16] with $O(n^3)$ total work, and an algorithm of Bracha and Rachman [10] with $O(n^2 \log n)$ total work.

In all of the above algorithms a process running alone may have to do all the work by itself, which implies that the individual work is the same as the total work. Later, Aspnes and Waarts [7] presented a shared coin algorithm in which each process performs $O(n \log^2 n)$ expected number of steps, which is significantly lower than the $O(n^2 \log n)$ individual work of the algorithm by Bracha and Rachman [10]. However, by simply multiplying the individual work by n, their algorithm increases the total work to $O(n^2 \log^2 n)$. This led Aspnes and Waarts to ask whether this tradeoff is inherent.

In this paper, we show that there is no such tradeoff, and in fact, each complexity measure can be optimized separately. We achieve this result by showing that shared-coin algorithms can be *interleaved* in a way that obtains the best of their complexity figures, without harming the agreement parameter.

Given the power of the adversary to observe both protocols and adjust their scheduling, it is not obvious that this is the case, and indeed, following [17], recent work of Lynch, Segala, and Vaandrager [15] shows that undesired effects may follow from the interaction of an adaptive adversary and composed probabilistic protocols. Nonetheless, we can show that in the particular case of weak shared-coin algorithms, two algorithms with agreement parameter δ_A and δ_B can be composed with sufficient independence such that the combined protocol terminates with the minimum of each of the protocols' complexities (e.g., in *both* individual and total work), while obtaining an agreement parameter of $\delta_A \cdot \delta_B$.

An immediate application of our result shows that the shared coin algorithm of Bracha and Rachman can be interleaved with the algorithm of Aspnes and Waarts, to get a protocol with both $O(n \log^2 n)$ individual work and $O(n^2 \log n)$ total work. This implies that wait-free randomized consensus can be solved with $O(n \log^2 n)$ expected individual work and $O(n^2 \log n)$ expected total work, using *single-writer* multi-reader registers. These are currently the best complexities known for this model.

Our result has other applications for combining shared coins in order to enjoy the best of more than one complexity measure. For example, Saks, Shavit, and Woll [16] presented three shared-coin algorithms, each having a good complexity for a different scenario. The complexity measure they consider is of *time units*: one time unit is defined to be a minimal interval in the execution of the algorithm during which each non-faulty processor executes at least one step. The first is a wait-free algorithm which takes $O(\frac{n^3}{n-f})$ time, where f is the actual number of faulty processes. In the second algorithm, the time is $O(\log n)$ in executions with at most $f = \sqrt{n}$ faulty processes, and in the third algorithm the time is $O(1)$ in failure-free executions. All three shared coin algorithms have constant agreement parameters, and Saks, Shavit, and Woll claim that they can be interleaved to yield one algorithm with a constant agreement parameter that enjoys all of the above complexities. However, they do not prove this claim, and our paper is the first to do so.

Our result holds also for the shared-memory model with *multi-writer* multi-reader registers. In this model, Attiya and Censor [8] have shown a shared-coin algorithm with $O(n^2)$ total work. Again, the individual work is also $O(n^2)$. Later, we have shown [4] that the individual work can be further reduced to $O(n \log n)$; again, this had a cost of increasing the total work to $O(n^2 \log n)$. A restricted version of the result presented here was first proven in [4] to obtain a shared-coin with $O(n^2)$ total work and $O(n \log n)$ individual work. This was superseded recently by Aspnes and Censor [5], who gave a shared-coin algorithm with $O(n)$ individual work and (immediately) $O(n^2)$ total work, which is optimal due to the $O(n^2)$ lower bound on total work presented by Attiya and Censor in [8].

2 Model of Computation

We consider a standard model of an asynchronous shared-memory system, where n processes communicate by reading and writing to shared registers. Each *step* consists of some local computation, including an arbitrary number of local coin flips (possibly biased) and one shared memory *event*, which is either a read or a write to some register. The interleaving of processes' events is governed by a *strong* adversary that observes the results of the local coin flips before scheduling the next event; in particular, it may observe a coin-flip and, based on its outcome, choose whether or not that process may proceed with its next shared-memory operation.

A *configuration* consists of the local states of all the processes, and the values of all the registers. Since we consider randomized algorithms with a strong adversary, we partition the configurations into two categories C_{alg} and C_{adv}. In configurations $C \in C_{alg}$ there is a process waiting to flip a local coin, and in configurations $C \in C_{adv}$ all processes are pending to access the shared memory.

For each configuration $C \in C_{alg}$ where p_i is about to flip a local coin there is a fixed probability space for the result of the coin, which we denote by X_i^C. An element $y \in X_i^C$ with probability $\Pr[y]$ represents a possible result of the local coin flip of p_i from the configuration C. If there is more than one process that is waiting to flip a coin in C, then we fix some arbitrary order of flipping the local coins, for example according to the process identifiers. In this case, p_i will be the process with the smallest identifier that is waiting to flip a coin in C. The next process will flip its coin only from the resulting configuration.

Algorithm 1 Interleaving shared coin algorithms A and B: code for process p_i.

```
1: while true do
2:     take a step in algorithm A
3:     if terminated in A by returning v then return v        // local computation
4:     take a step in algorithm B
5:     if terminated in B by returning v then return v        // local computation
```

We can now define the *strong adversary* formally, as follows. For every configuration $C \in C_{alg}$ the adversary lets a process p_i, with the smallest identifier, flip its local coin. For every configuration $C \in C_{adv}$, the adversary σ picks an arbitrary process to take a step which accesses the shared memory. Having the adversary wait until all processes flip their current coins does not restrict the adversary's power, since any adversary that makes a decision before scheduling some pending coin-flip, can be viewed as one that schedules the pending coin-flip but ignores its outcome until after making that decision.

3 Interleaving shared-coin algorithms

Let A and B be two shared-coin algorithms. Interleaving A and B is done by performing a loop in which the process executes one step of each algorithm (see Algorithm 1). When one of the algorithms terminates, returning some value v, the interleaved algorithm terminates as well, and returns the same value v.

We denote by δ_A and δ_B the agreement parameters of algorithm A and algorithm B, respectively.

We next show that the agreement parameter of the interleaved algorithm is the product of the agreement parameters of algorithms A and B. The idea behind the proof is that since different processes may choose a value for the shared coin based on either of the two algorithms, for all processes to agree on some value v we need all processes to agree on v in both algorithms. In order to deduce an agreement parameter which is the product of the two given agreement parameters, we need to show that the executions of the two algorithms are independent, in the sense that the adversary cannot gain any additional power out of running two interleaved algorithms.

In general, it is not obvious that the agreement parameter of the interleaved algorithm is the product of the two given agreement parameters. In each of the two algorithms it is only promised that there is a constant probability that the adversary cannot prevent a certain outcome, but in the interleaved algorithm the adversary does not have to decide in advance which outcome it tries to prevent from a certain algorithm, since it may depend on how the other algorithm proceeds. For example, it suffices for the adversary to have the processes in one algorithm agree on 0 and have the processes in the other algorithm agree on 1.

The first theorem assumes that one of the algorithms always terminates within some fixed bound on the number of steps, and not only with probability 1. We later extend this result to hold for any pair of shared coin algorithms.

Theorem 1 *If algorithm A always terminates within some fixed bound on the number of steps, then the interleaving of algorithms A and B has agreement parameter $\delta \geq \delta_A \cdot \delta_B$.*

Proof: Since algorithm A always terminates within some fixed bound on the number of steps, the interleaved algorithm also terminates within some fixed bound on the number of steps (at most twice as many). We define the probability of reaching agreement on the value v for every configuration C in one of the algorithms, by backwards induction, as follows.

With every configuration C, we associate a value s that is the maximal number of steps taken by all the processes from configuration C, over all possible adversaries and all results of the local coin flips, until they terminate in the interleaved algorithm (by terminating either in A or in B). Since algorithm A always terminates within some fixed bound on the number of steps, s is well defined.

We denote by $C|_A$ the projection of the configuration C on algorithm A. That is, $C|_A$ consists of the local states referring to algorithm A of all processes, and the values of the shared registers of algorithm A. Similarly we denote by $C|_B$ the projection of C on algorithm B.

We denote by S_C the set of adversaries possible from a configuration C. We consider a partition of S_C into S_C^A and S_C^B, which are the set of adversaries from configuration C whose next step is in algorithm A and B, respectively.

We define the probability $\Pr_v[C]$ for agreeing in the interleaved algorithm by induction on s. In a configuration C for which $s = 0$, all processes terminate in the interleaved algorithm, by terminating either in A or in B. We define $\Pr_v[C]$ to be 1 if all the processes decide v, and 0 otherwise. Let C be any other configuration. If $C \in C_{adv}$, then:

$$\Pr_v[C] = \min_{\sigma \in S_C} \Pr_v[C^\sigma],$$

where C^σ is the resulting configuration after scheduling one process, according to the adversary σ, to access the shared memory[1]. If $C \in C_{alg}$, then:

$$\Pr_v[C] = \sum_{y \in X_i^C} \Pr[y] \cdot \Pr_v[C^y],$$

where p_i is the process waiting to flip a local coin in the configuration C and C^y is the resulting configuration after the coin is flipped. Notice that for the initial configuration C_I, we have that $\delta = \min_v \Pr_v[C_I]$.

In order to prove the theorem, we use $\Pr_v^A[C]$ (and similarly $\Pr_v^B[C]$) for a configuration C in the interleaved algorithm, which is the probability that starting from C, all the processes that terminate by terminating in algorithm A (algorithm B) agree on the value v.

We define these probabilities formally by induction on s. We only state the definition of $\Pr_v^A[C]$; the definition of $\Pr_v^B[C]$ is analogous. Notice that $\Pr_v^A[C]$ depends only on the projection $C|_A$ of configuration C on algorithm A, and on the possible adversaries in S_C^A. For a configuration C in which $s = 0$, all the processes have terminated in the interleaved algorithm. We define $\Pr_v^A[C]$ to be 1 if all the processes that

[1]Notice that the minimum of the probabilities over all adversaries in S_C is well defined, since S_C is always finite because there is a fixed bound on the number of steps in algorithm A and a finite number of processes.

terminated by terminating in algorithm A agree on the value v, and 0 otherwise (in the latter case there is at least one process that terminated by terminating in algorithm A, but did not decide on the value v).

Let C be any other configuration, i.e., with $s > 0$. If $C \in C_{adv}$, then:

$$\Pr_v^A[C] = \Pr_v^A[C|_A, S_C^A] = \min_{\sigma \in S_C^A} \Pr_v^A[C^\sigma|_A, S_{C^\sigma}^A],$$

where C^σ is the resulting configuration after scheduling one process, according to the adversary σ, to access the shared memory. If $C \in C_{alg}$, then:

$$\Pr_v^A[C] = \Pr_v^A[C|_A, S_C^A] = \sum_{y \in X_i^C} \Pr[y] \cdot \Pr_v^A[C^y|_A, S_{C^y}^A],$$

where p_i is the process waiting to flip a local coin in the configuration C and C^y is the resulting configuration after the coin is flipped.

We now claim that for every configuration C, $\Pr_v[C] \geq \Pr_v^A[C] \cdot \Pr_v^B[C]$; the proof is by induction on s.

Base case: If $s = 0$, then all processes have terminated in the interleaved algorithm. Processes agree on v if and only if every process decides v, whether it terminates in A or in B, therefore

$$\Pr_v[C] = \Pr_v^A[C] \cdot \Pr_v^B[C].$$

Induction step: Assume the claim holds for any configuration C' with at most $s-1$ steps remaining to termination under any adversary. Let C be a configuration with at most s steps until termination under any adversary. We consider two cases according to the type of C.

If $C \in C_{adv}$, then:

$$\begin{aligned} \Pr_v[C] &= \min_{\sigma \in S_C} \Pr_v[C^\sigma] \\ &= \min\left\{ \min_{\sigma \in S_C^A} \Pr_v[C^\sigma], \min_{\sigma \in S_C^B} \Pr_v[C^\sigma] \right\}, \end{aligned}$$

By the induction hypothesis on the configuration C^σ, with one step less to termination, we get:

$$\begin{aligned} \Pr_v[C] &= \min\left\{ \min_{\sigma \in S_C^A} \left(\Pr_v^A[C^\sigma] \cdot \Pr_v^B[C^\sigma]\right), \min_{\sigma \in S_C^B} \left(\Pr_v^A[C^\sigma] \cdot \Pr_v^B[C^\sigma]\right) \right\} \\ &= \min\left\{ \min_{\sigma \in S_C^A} \left(\Pr_v^A[C^\sigma|_A, S_{C^\sigma}^A] \cdot \Pr_v^B[C^\sigma|_B, S_{C^\sigma}^B]\right), \min_{\sigma \in S_C^B} \left(\Pr_v^A[C^\sigma|_A, S_{C^\sigma}^A] \cdot \Pr_v^B[C^\sigma|_B, S_{C^\sigma}^B]\right) \right\} \end{aligned}$$

where the second equality follows by the definition of $\Pr_v^A[C^\sigma]$ and $\Pr_v^B[C^\sigma]$. If the step taken from C by σ is in algorithm A then $C^\sigma|_B = C|_B$. Moreover, $S_{C^\sigma}^B \subseteq S_C^B$, because a process may terminate in algorithm A and be unavailable for scheduling. Therefore we have

$$\Pr_v^B[C^\sigma|_B, S_{C^\sigma}^B] \geq \Pr_v^B[C|_B, S_C^B].$$

Similarly, if the step taken from C by σ is in algorithm B then $\Pr_v^A[C^\sigma|_A, S_{C^\sigma}^A] \geq \Pr_v^A[C|_A, S_C^A]$. Thus,

$$\begin{aligned}
\Pr_v[C] &\geq \min\left\{\min_{\sigma\in S_C^A}\left(\Pr_v^A[C^\sigma|_A, S_{C^\sigma}^A]\cdot\Pr_v^B[C|_B, S_C^B]\right), \min_{\sigma\in S_C^B}\left(\Pr_v^A[C|_A, S_C^A]\cdot\Pr_v^B[C^\sigma|_B, S_{C^\sigma}^B]\right)\right\} \\
&= \min\left\{\Pr_v^B[C|_B, S_C^B]\left(\min_{\sigma\in S_C^A}\Pr_v^A[C^\sigma|_A, S_{C^\sigma}^A]\right), \Pr_v^A[C|_A, S_C^A]\left(\min_{\sigma\in S_C^B}\Pr_v^B[C^\sigma|_B, S_{C^\sigma}^B]\right)\right\} \\
&= \min\left\{\Pr_v^B[C]\cdot\Pr_v^A[C], \Pr_v^A[C]\cdot\Pr_v^B[C]\right\} \\
&= \Pr_v^A[C]\cdot\Pr_v^B[C],
\end{aligned}$$

which completes the proof of the claim that $\Pr_v[C] \geq \Pr_v^A[C]\cdot\Pr_v^B[C]$ for a configuration $C \in C_{adv}$.

If $C \in C_{alg}$, with process p_i waiting to flip a local coin, then:

$$\begin{aligned}
\Pr_v[C] &= \sum_{y\in X_i^C}\Pr[y]\cdot\Pr_v[C^y] \\
&= \sum_{y\in X_i^C}\Pr[y]\cdot\Pr_v^A[C^y]\cdot\Pr_v^B[C^y] \\
&= \sum_{y\in X_i^C}\Pr[y]\cdot\Pr_v^A[C^y|_A, S_{C^y}^A]\cdot\Pr_v^B[C^y|_B, S_{C^y}^B],
\end{aligned}$$

by the induction hypothesis on the configuration C^y, with one step less to termination. If the coin of p_i is in algorithm A, then $C^y|_B = C|_B$. Moreover, $S_{C^y}^B \subseteq S_C^B$, because a process may terminate in algorithm A and be unavailable for scheduling. Therefore we have $\Pr_v^B[C^y|_B, S_{C^y}^B] \geq \Pr_v^B[C|_B, S_C^B]$. Similarly, if the coin of p_i is in algorithm B then $\Pr_v^A[C^y|_A, S_{C^y}^A] \geq \Pr_v^A[C|_A, S_C^A]$. Assume, without loss of generality, that the coin is in algorithm A, thus,

$$\begin{aligned}
\Pr_v[C] &\geq \sum_{y\in X_i^C}\Pr[y]\cdot\Pr_v^A[C^y|_A, S_{C^y}^A]\cdot\Pr_v^B[C|_B, S_C^B] \\
&= \Pr_v^B[C|_B, S_C^B]\left(\sum_{y\in X_i^C}\Pr[y]\cdot\Pr_v^A[C^y|_A, S_{C^y}^A]\right) \\
&= \Pr_v^B[C]\cdot\Pr_v^A[C],
\end{aligned}$$

which completes the proof of the claim that $\Pr_v[C] \geq \Pr_v^A[C]\cdot\Pr_v^B[C]$ for a configuration $C \in C_{alg}$.

The theorem follows by applying the claim to the initial configuration C_I, to get that $\Pr_v[C_I] \geq \Pr_v^A[C_I]\cdot\Pr_v^B[C_I]$. Notice again that in the interleaved algorithm, the adversary is slightly more restricted in scheduling processes to take steps in algorithm A, than in algorithm A itself, since a process might terminate in algorithm B and be unavailable for scheduling. This only reduces the power of the adversary, implying that $\Pr_v^A[C_I] \geq \delta_A$. The same applies for algorithm B and hence $\Pr_v^B[C_I] \geq \delta_B$. Therefore we have that $\Pr_v[C_I] \geq \Pr_v^A[C_I]\cdot\Pr_v^B[C_I] \geq \delta_A\cdot\delta_B$, for every $v \in \{0,1\}$, which completes the proof since $\delta = \min_v \Pr_v[C_I]$. ■

When neither algorithm A nor algorithm B have a bound on the number of steps until termination, the same result is obtained by considering *truncated* algorithms. In a truncated algorithm A_h, we stop the original algorithm A after at most h steps, for some finite number h, and if not all processes have terminated then we regard this execution as one that does not agree on any value. This only restricts the algorithm, so the agreement parameter of a truncated algorithm is at most the agreement parameter of the original algorithm, i.e., $\delta_{A_h} \leq \delta_A$.

For any shared coin algorithm, we define $\Pr_v^h[C_I]$ to be the probability that all the processes terminate and decide v within at most h steps from the initial configuration C_I. This is exactly the probability $\Pr_v[C_I]$ of the truncated algorithm A_h. The next lemma proves that the probabilities $\Pr_v^h[C_I]$ tend to the probability $\Pr_v[C_I]$, as h goes to infinity.

Lemma 2 *$Pr_v[C_I] = \lim_{h\to\infty} Pr_v^h[C_I]$.*

Proof: For every h, let Y_h be the event that all the processes terminate and decide v within at most h steps from the initial configuration C_I. By this definition, we have $\Pr[Y_h] = \Pr_v^h[C_I]$, and $\Pr[\cup_{h=1}^{\infty} Y_h] = \Pr_v[C_I]$.

It is easy to see that the sequence of events $Y_1, Y_2, \ldots$ is a monotone increasing sequence of events, i.e., $Y_1 \subseteq Y_2 \subseteq \cdots$, therefore the limit $\lim_{h\to\infty} \Pr[Y_h]$ exists, and by [12, Lemma 5, p. 7] we have $\Pr[\cup_{h=1}^{\infty} Y_h] = \lim_{h\to\infty} \Pr[Y_h]$. This implies that $\Pr_v[C_I] = \lim_{h\to\infty} \Pr_v^h[C_I]$. ■

We use the above limit to show that we can truncate an algorithm to get as close as desired to the agreement parameter by a bounded algorithm.

Lemma 3 *In a shared coin algorithm with agreement parameter δ, for every $\epsilon > 0$ there is an integer h_ϵ such that $Pr_v^{h_\epsilon}[C_I] \geq \delta - \epsilon$.*

Proof: Assume, towards a contradiction, that for every h we have $\Pr_v^h[C_I] < \delta - \epsilon$. Since $\Pr_v[C_I]$ is the probability that all the processes terminate and decide v (without a bound on the number of steps), this implies that

$$\Pr_v[C_I] = \lim_{h\to\infty} \Pr_v^h[C_I] \leq \delta - \epsilon < \delta,$$

which completes the proof. ■

We can now truncate the shared coin algorithm A after a finite number of steps as a function of ϵ, and use Theorem 1 to get an interleaved algorithm with agreement parameter $\delta_{A_{h_\epsilon}} \cdot \delta_B \geq (\delta_A - \epsilon) \cdot \delta_B$.

By truncating algorithm A we only restrict the interleaved algorithm (as we only restrict algorithm A), and therefore we have that by interleaving algorithms A and B we obtain an agreement parameter δ that is at least $(\delta_A - \epsilon) \cdot \delta_B$ for every $\epsilon > 0$, which implies that $\delta \geq \delta_A \cdot \delta_B$, and gives the following theorem.

Theorem 4 *The interleaving of algorithms A and B has agreement parameter $\delta \geq \delta_A \cdot \delta_B$.*

4 Applications

4.1 Shared Coin Using Single-Writer Registers

We obtain a shared-coin algorithm using only single-writer registers that has both $O(n^2 \log n)$ total work and $O(n \log^2 n)$ individual work, by interleaving the algorithm from [10] and the algorithm from [7].

For two shared-coin algorithms A and B, we denote by $T_A(n)$ and $I_A(n)$ the total and individual work, respectively, of algorithm A, and similarly denote $T_B(n)$ and $I_B(n)$ for algorithm B. We now argue that the total and individual step complexities of the interleaved algorithm are the minima of the respective complexities of algorithms A and B.

Lemma 5 *The interleaving of algorithms A and B has an expected total work of*

$$2\min\{T_A(n), T_B(n)\} + n,$$

and an expected individual work of

$$2\min\{I_A(n), I_B(n)\} + 1.$$

Proof: We begin by proving the total work. After at most $2T_A(n) + n$ total steps are executed by the adversary, at least $T_A(n)$ of them are in algorithm A, and hence all the processes have terminated in Algorithm A, and have therefore terminated in the interleaved algorithm. The same applies to Algorithm B. Therefore the interleaved algorithm has a total work of $2\min\{T_A(n), T_B(n)\} + n$.

We now prove the bound on the individual work. Consider any process p_i. After at most $2I_A(n) + 1$ total steps of p_i are executed by the adversary, at least $I_A(n)$ of them are in algorithm A, and hence the process p_i has terminated in Algorithm A, and has therefore terminated in the interleaved algorithm. The same applies to Algorithm B. This is true for all the processes, therefore the interleaved algorithm has an individual work of $2\min\{T_A(n), T_B(n)\} + 1$. ■

Applying Lemma 5 and Theorem 1 to an interleaving of the algorithms of [10] and [7], yields:

Theorem 6 *There is a shared-coin algorithm with a constant agreement parameter, with $O(n^2 \log n)$ total work and $O(n \log^2 n)$ individual work, using single-writer multi-reader registers.*

4.2 Shared Coin for Different Levels of Resilience

In this section we discuss the interleaving done by Saks, Shavit, and Woll [16]. They presented the following three shared-coin algorithms, all having a constant agreement parameter. The complexity measure they consider is of *time units*: one time unit is defined to be a minimal interval in the execution of the algorithm during which each non-faulty processor executes at least one step.

The first algorithm takes $O(\frac{n^3}{n-f})$ time, where f is the number of faulty processes. It is wait-free, i.e., it can tolerate $f = n - 1$ faulty processes. This is done by having each process repeatedly flip a local coin

and write it into an array, then collect the array to see if at least n^2 coins were already flipped. Once a process observes that enough coins were flipped, it terminates and decides on the majority of all the coins it collected. The individual work of the first algorithm is in $O(n^3)$, since in the worst case, the process does all the work by itself.

The second algorithm takes $O(\log n)$ time in executions with at most $f = \sqrt{n}$ faulty processes, but may not terminate otherwise. This is done by having each process flip one local coin and write it into an array, then repeatedly scan the array until it observes that at least $n - \sqrt{n}$ coins were already flipped. It then terminates and decides on the majority of all the coins it collected.

In the third algorithm there is one predetermined process that flips one local coin and writes the result into a shared memory location. The other processes repeatedly read that location until they see it has been written into, and then decide that value. This takes $O(1)$ time in failure-free executions, but may not terminate otherwise.

Theorem 1 shows that the interleaving of these three algorithm gives a shared coin algorithm with a constant agreement parameter. Technically, we apply the theorem twice, first to interleave the first algorithm (which is bounded) and the second algorithm; then we interleave the resulting algorithm (which is bounded) with the third algorithm.

The interleaving of all three algorithms enjoys all of the above complexities, by an argument similar to Lemma 5. This can of course be further improved by replacing the first algorithm with the algorithm of Section 4.1 or with the algorithm of Aspnes and Censor [5], if multi-writer registers can be employed.

5 Summary

We prove that interleaving shared coin algorithms yields a shared-coin algorithm whose agreement parameter is the product of the agreement parameters of the two algorithms, and its complexity is the minimum of their complexity figures. In particular, this yields a shared-coin algorithm with a constant agreement parameter, with $O(n^2 \log n)$ total work and $O(n \log^2 n)$ individual work, using single-writer multi-reader registers. Using the scheme of Aspnes and Herlihy [6] this gives a randomized consensus algorithm with $O(n^2 \log n)$ total work and $O(n \log^2 n)$ individual work, using single-writer multi-reader registers. We also give the first proof of the interleaving done by Saks, Shavit, and Woll [16], for algorithms that provide different levels of resilience.

An important open question is whether the complexities of Theorem 6 can be improved in the model of single-writer registers. The lower bounds of $\Omega(n^2)$ for total work and hence $\Omega(n)$ for individual work, which were obtained in [8] for multi-writer registers and are tight in that case, apply also to single-writer registers. This still leaves polylogarithmic gaps for both complexity measures. Resolving whether there is an inherent difference in the complexity of randomized consensus between the two models of registers is an intriguing direction for further research.

The result about interleaving is fairly general and can be extended to other communication models, and to other complexity measures as well.

Acknowledgements: We would like to thank Roberto Segala for useful discussions.

References

[1] K. Abrahamson. On achieving consensus using a shared memory. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 291–302, 1988.

[2] J. Aspnes. Time- and space-efficient randomized consensus. *Journal of Algorithms*, 14(3):414–431, May 1993.

[3] J. Aspnes. Randomized protocols for aynchronous consensus. *Distributed Computing*, 16(2-3):165–176, Sept. 2003.

[4] J. Aspnes, H. Attiya, and K. Censor. Randomized consensus in expected $O(n \log n)$ individual work. In *Proceedings of the 27th ACM symposium on Principles of distributed computing (PODC)*, pages 325–334, 2008.

[5] J. Aspnes and K. Censor. Approximate shared-memory counting despite a strong adversary. In *Proceedings of the 20th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 441–450, 2009 (Full version to appear in *Transactions on Algorithms, SODA 2009 special issue*).

[6] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, 1990.

[7] J. Aspnes and O. Waarts. Randomized consensus in expected $O(n \log^2 n)$ operations per processor. *SIAM J. Comput.*, 25(5):1024–1044, 1996.

[8] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. *J. ACM*, 55(5):1–26, 2008.

[9] H. Attiya, D. Dolev, and N. Shavit. Bounded polynomial randomized consensus. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 281–293, 1989.

[10] G. Bracha and O. Rachman. Randomized consensus in expected $O(n^2 \log n)$ operations. In *Proceedings of the 5th International Workshop on Distributed Algorithms (WDAG)*, pages 143–150, 1991.

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

[12] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford Science Publications, 2nd edition, 1992.

[13] M. Herlihy. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst.*, 13(1):124–149, January 1991.

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

[15] N. A. Lynch, R. Segala, and F. W. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM J. Comput.*, 37(4):977–1013, 2007.

[16] M. Saks, N. Shavit, and H. Woll. Optimal time randomized consensus—making resilient algorithms fast in practice. In *Proceedings of the 2nd annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 351–362, 1991.

[17] R. Segala. A compositional trace-based semantics for probabilistic automata. In *CONCUR '95: Proceedings of the 6th International Conference on Concurrency Theory*, pages 234–248, London, UK, 1995. Springer-Verlag.