Distributed Cooperation During the Absence of Communication*

Grzegorz Greg Malewicz[1], Alexander Russell[1], and Alex A. Shvartsman[1,2]

[1] Department of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269, USA.
{greg,acr,alex}@cse.uconn.edu
[2] Laboratory for Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

Abstract. This paper presents a study of a distributed cooperation problem under the assumption that processors may not be able to communicate for a prolonged time. The problem for n processors is defined in terms of t tasks that need to be performed efficiently and that are known to all processors. The results of this study characterize the ability of the processors to schedule their work so that when some processors establish communication, the wasted (redundant) work these processors have collectively performed prior to that time is controlled. The lower bound for wasted work presented here shows that for any set of schedules there are two processors such that when they complete t_1 and t_2 tasks respectively the number of redundant tasks is $\Omega(t_1 t_2/t)$. For $n = t$ and for schedules longer than $\sqrt{n}$, the number of redundant tasks for two or more processors must be at least 2. The upper bound on pairwise waste for schedules of length $\sqrt{n}$ is shown to be 1. Our efficient deterministic schedule construction is motivated by design theory. To obtain *linear* length schedules, a novel deterministic and efficient construction is given. This construction has the property that pairwise wasted work increases gracefully as processors progress through their schedules. Finally our analysis of a random scheduling solution shows that with high probability pairwise waste is well behaved at all times: specifically, two processors having completed t_1 and t_2 tasks, respectively, are guaranteed to have no more than $t_1 t_2/t + \Delta$ redundant tasks, where $\Delta = O(\log n + \sqrt{t_1 t_2/t}\sqrt{\log n})$.

1 Introduction

The problem of cooperatively performing a set of tasks in a decentralized setting where the computing medium is subject to failures is a fundamental problem in distributed computing. Variations on this problem have been studied in in message-passing models [3, 5, 7], using group communications [6, 9], and in shared-memory computing using deterministic [12] and randomized [2, 13, 16] models.

* This work was supported by NSF Grant CCR-9988304 and a grant from AFOSR. The work of the third author was supported by a NSF Career Award.

We consider the abstract problem of performing t tasks in a distributed environment consisting of n processors. We refer to this as the DO-ALL problem. The problem has simple and efficient solutions in synchronous fault-free systems; however, when failures and delays are introduced the problem becomes very challenging. Dwork, Halpern and Waarts [7] consider the DO-ALL problem in message-passing systems and use a work measure W defined as the number of tasks executed, counting multiplicities, to assess the computational efficiency. A more conservative measure [5] includes any additional steps taken by the processors, for example steps taken for coordination and waiting for messages. Communication efficiency M is gauged using the message complexity, accounting for all messages sent during the computation. It is not difficult to formulate solutions for DO-ALL in which each processor performs each of the t tasks. Such solutions have $W = \Omega(t \cdot n)$, and they do not require any communication, i.e., $M = 0$. Another extreme is the synchronous model with fail-stop processors, where each processor can send 0-delay messages to inform their peers of the computation progress. In this case one can show that $W = O(t + n \log n / \log\log n)$. This work is efficient (there is a matching lower bound, cf. [12]), and the upper bound does not depend on the number of failures. However the number of messages is more than quadratic, and can be $\Omega(n^2 \log n / \log\log n)$ [3]. Thus satisfactory solutions for DO-ALL must incorporate trade-off between communication and computation.

In failure- and delay-prone settings it is difficult to precisely control the trade-off between communication and computation. In some cases [7] it is meaningful to attempt to optimize the overall *effort* defined as the sum of work and message complexities, in other cases [5] an attempt is made to optimize efficiency in a *lexicographic* fashion by first optimizing work, and then communication. For problems where the quality of distributed decision-making depends on communication and can be traded off for communication, the solution space needs to consider the possibility of *no communication*. Notably, this is the case in the load-balancing setting introduced by Papadimitriou and Yanakakis [18] and studied by Georgiades, Mavronicolas and Spirakis [8]. In this work we study the ability of n processors to perform efficient scheduling of t tasks (initially known to all processors) during prolonged periods of *absence of communication*.

This setting is interesting for several reasons. If the communication links are subject to failures, then each processor must be ready to execute all of the t tasks, whether or not it is able to communicate. In realistic settings the processors may not initially be aware of the network configuration, which would require expenditure of computation resources to establish communication, for example in radio networks. In distributed environments involving autonomous agents, processors may *choose* not to communicate either because they need to conserve power or because they must maintain radio silence. Regardless of the reasons, it is important to direct any available computation resources to performing the required tasks as soon as possible. In all such scenarios, the t tasks have to be scheduled for execution by all processors. The goal of such scheduling must be to control redundant task executions in the absence of communication and during the period of time when the communication channels are being (re)established.

For a variation of DO-ALL Dolev *et al.* [6] showed that for the case of dynamic changes in connectivity, the termination time of any on-line task assignment algorithm can be greater than the termination time of an off-line task assignment algorithm by a factor linear in n. This means that an on-line algorithm may not be able to do better than the trivial solution that incurs linear overhead by having each processor perform all the tasks. With this observation [6] develops an effective strategy for managing the task execution redundancy and prove that the strategy provides each of the n processors with a schedule of $\Theta(n^{1/3})$ tasks such that at most one task is performed redundantly by any two processors.

In this work we advance the state-of-the-art with the ultimate goal of developing a general scheduling theory that helps eliminate redundant task executions in scenarios where there are long periods of time during which processors work in isolation. We require that all tasks are performed even in the absence of communication. A processor may learn about task executions either by executing a task itself of by learning that the task was executed by some other processor. Since we assume initial lack of communication and the possibility that a processor may never be able to communicate, each processor must know the set of tasks to perform. We seek solutions where the isolated processors can execute tasks independently such that when any two processors are able to communicate, the number of tasks they have *both* executed is as small as possible. We model solutions to the problem as sets of n lists of distinct tasks from $\{1, \ldots, t\}$. We call such lists *schedules*.

Consider an example with two processors ($n = 2$). Let the schedule of the first processor be $\langle 1, 2, 3, \ldots, t \rangle$, and the schedule of the second processor be $\langle t, t-1, t-2, \ldots, 1 \rangle$. In the absence of communication each processor works without the knowledge of what the other is doing. If the processors are able to communicate after they have completed t_1 and t_2 tasks respectively and if $t_1 + t_2 \leq t$ then no work is wasted (no task is executed twice). If $t_1 + t_2 > t$, then the redundant work is $t_1 + t_2 - t$. In fact this is a lower bound on waste for any set of schedules. If some two processors have individually performed all tasks, then the wasted work is t.

Contributions. This paper presents new results that identify limits on bounded-redundancy scheduling of t tasks on n processors during the absence of communication, and gives efficient and effective constructions of bounded-redundancy schedules using deterministic and randomized techniques.

Lower Bounds. In Section 3 we show that for any n schedules for t tasks the worst case pairwise redundancy when one processor performs t_1 and another t_2 tasks is $\Omega(t_1 t_2 / t)$, e.g., the pairwise wasted work grows quadratically with the schedule length, see Figure 1.(a). We also show that for $n = t$ and for schedules with length exceeding $\sqrt{n}$, the number of redundant tasks for two (or more) processors must be at least two.

When $t \gg n$ scheduling is relatively easy initially by assigning chunks of t/n tasks to each processor. Our deterministic construction focuses on the most challenging case when $t = n$.

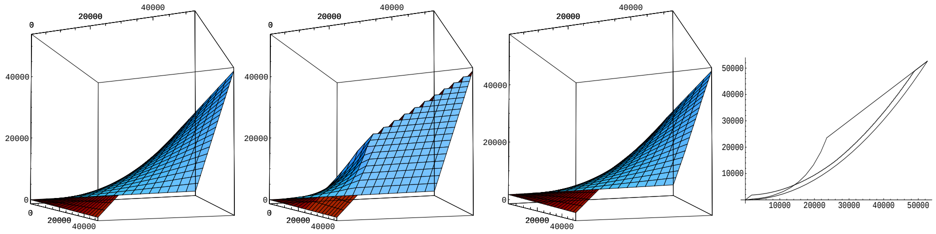

Fig. 1. Pairwise waste (redundancy) as a function of advancement through schedules for $n = t$: (a) lower bound, (b) deterministic construction (c) randomized construction, (d) diagonal vertical cut.

Deterministic Construction of Short Schedules. We show in Section 4 that it is in fact possible to construct schedules of length $\Theta(\sqrt{n})$ such that exactly *one* redundant task is performed for *any* pair of processors. This result exhibits a connection between design theory [10, 4] and the distributed problem we consider. Our design-theoretic construction is efficient and practical. The schedules are constructed by each processor independently in $O(\sqrt{n})$ time.

Deterministic Construction of Long Schedules. Design theory offers little insight on how to extend a set of schedules into longer schedules in which waste is increased in a controlled fashion. We show in Section 5 that longer schedules with controlled waste can be constructed in time linear in the length of the schedule. This deterministic construction yields schedules of length $\frac{4}{9}n$ such that pairwise wasted work increases gradually as processors progress through their schedules. For each pair of processors p_1 and p_2, the overlap of the first t_1 tasks of processor p_1 and the first t_2 tasks of processor p_2 is bounded by $O\left(\frac{t_1 t_2}{n} + \sqrt{n}\right)$. The upper bound on pairwise overlaps is illustrated in Figure 1(b). The quadratic growth in overlap is anticipated by our lower bound. The overall construction takes linear time and, except for the first $\sqrt{n}$ tasks, the cost of constructing the schedule is completely amortized.

Randomized Constructions. Finally, in Section 6, we explore the behavior of schedules selected at random. Specifically, we explore the waste incurred when each processor's schedule is selected uniformly among all permutations on $\{1, \ldots, t\}$. For the case of pairwise waste, we show that with high probability these random schedules enjoy two satisfying properties: (i) for each pair of processors p_1, p_2, the overlap of the first t_1 tasks of processor p_1 and the first t_2 tasks of processor p_2 is no more than $\frac{t_1 t_2}{t} + O\left(\log n + \sqrt{\frac{t_1 t_2}{t} \log n}\right)$, (ii) all but a vanishing fraction of the pairs of processors experience no more than a single redundant task in the first $\sqrt{t}$ tasks of their schedules. This is illustrated in Figure 1(c). As previously mentioned, the quadratic growth observed in property (i) above is unavoidable.

The results represented by the surfaces in Figures 1(a), (b) and (c) are compared along the vertical diagonal cut in Figure 1(d).

2 Definition and Models

We consider the abstract setting where n processors need to perform t independent tasks, where $n \leq t$. The processors have unique identifiers from the set $[n] = \{1, \dots, n\}$, and the tasks have unique identifiers from the set $[t] = \{1, \dots, t\}$. Initially each processor knows the tasks that need to be performed and their identifiers, which is necessary for solving the problem in absence of communication.

A *schedule* L is a list $L = \langle \tau^1, \dots, \tau^b \rangle$ of distinct tasks from $[t]$, where b is the *length of the schedule* ($b \geq 0$). A *system of schedules* $\mathcal{L}$ is a list of schedules for n processors $\mathcal{L} = \langle L_1, \dots, L_n \rangle$. When each schedule in the system of schedules $\mathcal{L}$ has the same length b, we say that $\mathcal{L}$ has length b. Given a schedule L of length b, and $c \geq 0$, we define the *prefix schedule* L^c to be: $L^c = \langle \tau^1, \dots, \tau^c \rangle$, if $c \leq b$, and $L^c = L$, if $c > b$. For a system of schedules $\mathcal{L}$ and a vector $\boldsymbol{a} = \langle a_1, \dots, a_n \rangle$ ($a_i \geq 0$) a system of schedules $\mathcal{L}^{\boldsymbol{a}} = \langle L_1^{a_1}, \dots, L_n^{a_n} \rangle$ is called a *prefix system of schedules*.

Sometimes we the order of tasks in a schedule is irrelevant, and we introduce the notion of *plan* as an unordered set of tasks. Given a schedule $L = \langle \tau^1, \dots, \tau^a \rangle$ we define the *plan* $P = P(L)$ to be the set $P = \{\tau^1, \dots, \tau^a\}$. Given a schedule L and $c \geq 0$, we write P^c to denote the plan corresponding to the schedule L^c (the set of the first c tasks from schedule L). For a system of schedules $\mathcal{L} = \langle L_1, \dots, L_n \rangle$, a *system of plans* is the list of plans $\mathcal{P} = \langle P_1, \dots, P_n \rangle$, where P_i is the plan for schedule L_i.

We can represent a system of plans as a matrix called a *scheme*. Specifically, given a system of plans $\mathcal{P}$ we define the *scheme* $\mathcal{S}$ to be the $n \times t$ matrix $(s_{i,j})$ such that $s_{i,j} = 1$ if $j \in P_i$, and $s_{i,j} = 0$ otherwise. Conversely, a scheme $\mathcal{S}$ yields a system of plans $\mathcal{P} = \langle P_1, \dots, P_n \rangle$, where $P_i = \{m : s_{i,m} = 1\}$; we say that $P_1, \dots, P_n$ are the plans of scheme $\mathcal{S}$. A scheme is called r-regular if each row has r ones, and k-uniform if each column has k ones. Since scheme and system of plans representations are equivalent, we choose the most convenient notation depending on the context. When the ordering of tasks is important, we use the schedule representation.

To assess the quality of scheme $\mathcal{S}$, we are interested in quantifying the "wasted" (redundant) work performed by a collection I of processors when each processor i ($i \in I$) performs all tasks assigned to it by the corresponding plan P_i of $\mathcal{S}$. We formalize the notion of *waste* as follows.

Definition 1. *For a collection $I \subseteq [n]$ of processors and a scheme $\mathcal{S}$ the* I-waste *of $\mathcal{S}$, denoted $w_I(\mathcal{S})$, is defined as $w_I(\mathcal{S}) = \sum_{i \in I} |P_i| - \left|\bigcup_{i \in I} P_i\right|$, where $P_1, \dots, P_n$ are the plans of $\mathcal{S}$.*

In general, we are interested in bounding the worst case redundant work of any set of k processors that may (re)establish communication after they perform all tasks assigned to them. Hence we introduce *k-waste* by ranging *I-waste* over all subsets I of size k:

Definition 2. *For a scheme $\mathcal{S}$ the* k-waste *of $\mathcal{S}$ is the quantity $w_k(\mathcal{S}) = \max_{I \subseteq [n], |I| = k} w_I(\mathcal{S})$.*

For a system of schedules $\mathcal{L}$ we write $w_k(\mathcal{L})$ to stand for $w_k(\mathcal{S})$, where $\mathcal{S}$ is the scheme induced by $\mathcal{L}$. In our work we are mostly interested in bounding k-waste for the case when $k = 2$. Observe that $w_{\{i,j\}}(\mathcal{S})$ is exactly $|P_i \cap P_j|$, so that in this case we are interested in controlling *overlaps*:

Definition 3. *We say that a scheme $\mathcal{S}$ is λ-bounded if $|P_i \cap P_j| \le \lambda$ for all $i \neq j$. More generally, $\mathcal{S}$ is $[\lambda, u]$-bounded if for all sets $U \subseteq [n]$ of cardinality u we have $\left|\bigcap_{j \in U} P_j\right| \le \lambda$. We say that $\mathcal{S}$ has λ-overlap (or is λ-overlapping) if there exists $i \neq j$ so that $|P_i \cap P_j| \ge \lambda$. More generally, $\mathcal{S}$ has $[\lambda, u]$-overlap if there is a set $U \subseteq [n]$ of cardinality u such that $\left|\bigcap_{j \in U} P_j\right| \ge \lambda$.*

In this work we assume that it takes unit time to add, multiply or divide two $\log(\max\{n, t\})$-bit numbers.

3 Lower Bounds on Processor-Pairs Overlaps

In this section we show lower bounds for 2-waste. We prove that 2-waste has to grow *quadratically* with the length of system of schedules, and is inversely proportional to t. This is intuitive; if $t \gg n$ then it is easy to construct n schedules of at least $\lfloor t/n \rfloor$ tasks such that the resulting scheme is 0-bounded, i.e., the 2-waste of the scheme is 0. On the other hand if $n = t$ then any system of schedules of length at least 2 must be 1-overlapping. A system of 1-bounded schedules of length $\Theta(\sqrt[3]{n})$ for $t = n$ tasks was designed by Dolev *et al.* [6]. We show that for $n = t$ no schedules can have the length greater than $\sqrt{n}$ and still be 1-bounded.

We first show a key lemma that uses a probabilistic argument (see [1] for other proofs with this flavor). Recall that given a schedule L_i, the plan P_i^a is the set of the first a tasks in L_i.

Lemma 1. *Let $\mathcal{L} = \langle L_1, \ldots, L_n \rangle$ be a system of schedules of length t, let $0 \le a \le t$, $0 \le b \le t$, and $\lambda = \max_{i \neq j} |P_i^a \cap P_j^b|$. Then $(n-1)\lambda \ge \frac{n}{t}ab - \min\{a, b\}$.*

Proof. We select i and j independently at random among $[n]$ and bound the expected value of $|P_i^a \cap P_j^b|$ in two ways. First observe that we have the total of n^2 pairs for i and j. If $i \neq j$ then the cardinality of the intersection is bounded by λ. If $i = j$ then the cardinality is obviously $\min\{a, b\}$. Hence

$$\mathbf{E}[|P_i^a \cap P_j^b|] \le \frac{n(n-1)\lambda + n \cdot \min\{a,b\}}{n^2}$$

For the second bound we consider t random variables X_τ, indexed by $\tau \in [t]$, defined as follows: $X_\tau = 1$ if $\tau \in P_i^a \cap P_j^b$, 0 otherwise. Observe that $\mathbf{E}[|P_i^a \cap P_j^b|] = \mathbf{E}[\sum_{\tau \in [t]} X_\tau]$. By linearity of expectation, and the fact that the events are independent, we may recompute this expectation

$$\mathbf{E}[|P_i^a \cap P_j^b|] = \sum_{\tau \in [t]} \mathbf{E}[X_\tau] = \sum_{\tau \in [t]} \mathsf{Pr}\left[\tau \in P_i^a\right] \cdot \mathsf{Pr}\left[\tau \in P_j^b\right]$$

Now we introduce the function $x^m(\tau)$, equal to the number of the prefixes of schedules of length m to which τ belongs, i.e., $x^m(\tau) = |\{i : \tau \in P_i^m\}|$. Using the fact that $\mathsf{Pr}\left[\tau \in P_i^m\right] = \frac{x^m(\tau)}{n}$, and twice the Cauchy-Schwartz inequality, we can rewrite the expectation as follows.

$$\mathbf{E}[|P_i^a \cap P_j^b|] = \frac{1}{n^2} \sum_{\tau \in [t]} x^a(\tau) x^b(\tau) \geq$$
$$\frac{1}{n^2} \sqrt{\sum_{\tau \in [t]} x^a(\tau)^2} \sqrt{\sum_{\tau \in [t]} x^b(\tau)^2} \;\geq\; \frac{1}{tn^2} \sqrt{\left(\sum_{\tau \in [t]} x^a(\tau)\right)^2} \sqrt{\left(\sum_{\tau \in [t]} x^b(\tau)\right)^2}$$

Finally, since $|P_i^m| = m$, we have that $\sum_{\tau \in [t]} x^m(\tau) = m \cdot n$. Hence $\mathbf{E}[|P_i^a \cap P_j^b|] \geq \frac{ab}{t}$, and the result follows.

For any given system of schedules $\mathcal{L}$, Lemma 1 leads to a lower bound on the pairwise overlap for any two processors i and j when i performs the tasks in P_i^a and j performs the tasks in P_j^b. The lower bound in the next theorem states that the pairwise overlap must be proportional to $a \cdot b$ (see Figure 1(a) for the case when $n = t$).

Theorem 1. *Let $\mathcal{L} = \langle L_1, \ldots, L_n \rangle$ be a system of schedules of length t, and let $0 \leq a \leq t$, $0 \leq b \leq t$. Then* $\max_{i \neq j} |P_i^a \cap P_j^b| \geq \lceil \frac{n}{t(n-1)} ab - \frac{\min\{a,b\}}{n-1} \rceil = \Omega(\frac{ab}{t})$.

Immediate consequence of Theorem 1 is that 2-waste must grow quadratically with the length of the schedule. Observe that k-waste, for $k \geq 2$, must be at least as big as 2-waste, because additional processors can only increase the number of tasks executed redundantly. Hence our next result is that k-waste must grow quadratically with the length of the schedule.

Corollary 1. *If $\mathcal{L}$ is a n-processor system of schedules of length r for $t = n$ tasks, where $t \geq r$, then $w_k(\mathcal{L}) \geq \lceil \frac{r \cdot (r-1)}{n-1} \rceil$.*

Finally we show that no 1-bounded schedules exist of length greater than $\sqrt{n - 3/4} + \frac{1}{2} > \sqrt{n}$.

Corollary 2. *If $r > \sqrt{n - 3/4} + \frac{1}{2}$ then any n-processor schedule of length r for n tasks is 2-overlapping.*

This result is tight: in Section 4 we construct an infinite family of 1-bounded schedules of length $\sqrt{n - 3/4} + \frac{1}{2}$.

4 Construction of Deterministic "Square-root" Plans

We now present an efficient construction of deterministic 1-bounded schedules with maximal $\Theta(\sqrt{n})$ length, for $n = t$. In the rest of this section we assume that $n = t$.

We briefly introduce the concept of *design*, the major object of interest in design theory. A reader interested in this subject is referred to, e.g., [10]. A design is a set of n points and t blocks (subsets of points) with the following properties. Each block contains exactly k points, each point is contained in (is on) exactly r blocks, number of blocks any subset of σ points intersects (is on) is exactly λ. An object with such properties is called σ-(n, k, λ) design. A design can be represented by an *incidence matrix* $(a_{i,j})$ of zeros and ones. Numbering points and blocks, an element $a_{i,j}$ of the matrix is 1 if point i is on block j and otherwise 0. Designs have many interesting properties. One fact is that a

σ-(n,k,λ) design is also a u-(n,k,λ) design for $0 \le u \le \sigma$. Not surprisingly for smaller u the number of blocks a subset of u points is on increases. This number is given by[1]: $\lambda \cdot \frac{(n-u)^{\underline{\sigma-u}}}{(k-u)^{\underline{\sigma-u}}}$, (see [10] Theorem 1.2).

We now give the result linking design theory to our setting.

Theorem 2. *The incidence matrix of any σ-(n,k,λ) design with t blocks yields a $[\Lambda,u]$-bounded scheme ($0 \le u \le \sigma$) for n processors and t tasks, where each processor executes $r = \frac{t}{n}k$ tasks, each task is executed k times, and $\Lambda = \lambda \cdot \frac{(n-u)^{\underline{\sigma-u}}}{(k-u)^{\underline{\sigma-u}}}$.*

Proof. Take any σ distinct points of the design. By the definition of σ-(n,k,λ) design the number of blocks on these σ points is equal to λ. Hence the number of tasks executed in common by any σ processors is exactly λ. The formula for Λ results from Theorem 1.2 [10]. This is because the design is a $(\sigma-(\sigma-u))$-(n,k,Λ) design, i.e., u-(n,k,Λ) design, for Λ as in that theorem. Moreover, since $t \cdot k = n \cdot r$ (see Corollary 1.4 [10]), each processor executes $r = \frac{t}{n}k$ tasks.

Theorem 2 makes it clear that we need to look for designs with large k and small λ because such designs yield long plans (large r) with small overlap (small Λ). We will consider a special case of this theorem for $\sigma = 2$. In this case we want to guarantee that 2-waste is exactly λ (note that when $u = \sigma = 2$, we have $\lambda = \Lambda$).

We use a well-known construction of a $2-(q^2+q+1, q+1, 1)$ design, for a prime q. The algorithm is presented in Figure 2. It has the following properties: (1) For a given number $i \in \{0,\ldots,q^2+q\}$, the value of a function $\mathsf{blocksOnPoint}(i)$ is a set of $q+1$ distinct integers from $\{0,\ldots,q^2+q\}$. (2) For $i \ne j$ the intersection of the set $\mathsf{blocksOnPoint}(i)$ with the set $\mathsf{blocksOnPoint}(j)$ is a singleton from $\{0,\ldots,q^2+q\}$. For a proof these two standard facts from design theory see e.g. [10, 15]. Invoking the function $\mathsf{blocksOnPoint}(i)$ for any i requires finding at most two multiplicative inverses b^{-1} and c^{-1} in $\mathbb{Z}_q$. We can do this in $O(\log q)$ by using the Extended Euclid's Algorithm (see [14], page 325). The worst case time of finding inverses is bounded, by the Lamé theorem, by $O(\log q)$, see [14], page 343. This cost is subsumed by q iterations of the loop. Hence the total time cost of the function is $O(q)$.

Theorem 3. *If $r \cdot (r-1) = n-1$ and $r-1 = q$ is prime then it is possible to construct a r-regular r-uniform 1-bounded scheme for n processors and n tasks. Each plan is constructed independently in $O(\sqrt{n})$ time.*

Using our construction we can quickly compute schedules of size approximately $\sqrt{n}$ for n processors and $t = n$ tasks, provided we have a prime q such that $q(q+1) = n-1$. Of course in general, for a given n there may not be a prime q that satisfies $q(q+1) = n-1$. This however does not limit our construction. We discuss this in more detail in Section 5

[1] The expression $y^{\underline{x}}$ is the "falling power" defined as $y(y-1)(y-2)\ldots(y-x+1)$, with $y^{\underline{0}} = y^0 = 1$.

```
vectorToIndex( a, b, c )                    indexToVector( i )
  if a = 1 then return b · q + c              if i = q · q + q then return (0, 0, 1)
  else if b = 1 then return q · q + c         else if i ≥ q · q then return (0, 1, i − q · q)
       else return q · q + q                       else return (1, i div q, i mod q)

blocksOnPoint( i )
  (a, b, c) = indexToVector( i )
  block = ∅
  if a = 1 ∧ b ≠ 0 ∧ c ≠ 0 then block ∪= {vectorToIndex( 0, 1, − b · c^-1 )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 1, (−1 − c · d) · b^-1, d )}
  if a = 1 ∧ b = 0 ∧ c ≠ 0 then block ∪= {vectorToIndex( 0, 1, 0 )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 1, d, −c^-1, d )}
  if a = 1 ∧ b ≠ 0 ∧ c = 0 then block ∪= {vectorToIndex( 0, 0, 1 )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 1, −b^-1, d )}
  if a = 1 ∧ b = 0 ∧ c = 0 then block ∪= {vectorToIndex( (0, 0, 1) )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 0, 1, d )}
  if a = 0 ∧ b = 1 ∧ c ≠ 0 then block ∪= {vectorToIndex( 0, 1, −c^-1 )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 1, d, −d · c^-1 )}
  if a = 0 ∧ b = 1 ∧ c = 0 then block ∪= {vectorToIndex( (0, 0, 1) )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 1, 0, d )}
  if a = 0 ∧ b = 0 ∧ c = 1 then block ∪= {vectorToIndex( (0, 1, 0) )}
        for d = 0 to q − 1 do block ∪= {vectorToIndex( 1, d, 0 )}
  return block
```

Fig. 2. Algorithm for finding $q+1$ blocks on a point of a 2-$(q^2+q+1, q+1, 1)$ design. The notation $x \cup= \ y$ stands for $x = x \cup y$. Boldface font denotes arithmetic in $\mathbb{Z}_q$.

5 Constructing Long Deterministic Schedules

Applying design theory principles to constructing longer schedules is *not* necessarily a good idea. If we took a design with blocks of size $k > \sqrt{n}$ we could build a corresponding system of schedules using Theorem 2. Observe that Theorem 1 guarantees that such system would have overlap $\Omega(\frac{k^2}{n})$. Unfortunately there would be no guarantee that the overlap would increase *gradually* as processors progress through their schedules. In particular, $\Omega(\frac{k^2}{n})$ overlap may be incurred even if two processors "meet" only after executing $O(\frac{k^2}{n})$ tasks.

In this section we present a construction for longer schedules with the goal of maintaining a graceful degradation of overlap. Our novel construction extends the $\sqrt{n}$-length system of plans obtained in Theorem 3 so that the increase of overlap is controlled as the number of tasks executed by each processor grows. In the following sections we construct *raw schedules*, and then show how to use them to produce schedules with graceful degradation of overlap for arbitrary value of n.

Raw Schedules. In this section we build long raw schedules that have repeated tasks. We assume that $n = r^2 - r + 1$ and $r = q + 1$ for a prime q and use the construction from Theorem 3. Let $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ be the resulting 1-bounded system of n plans of length r, where P_u is the plan for each $u \in \{1, \ldots, n\}$. For a processor u ($1 \le u \le n$) let $L_u = \langle t_u^1, \dots, t_u^r \rangle$ be the sequence of tasks, in some order, from the plan P_u constructed as in Theorem 3. We introduce the term *raw schedule* to denote a sequence of task identifiers where some tasks may be repeated.

We now present and analyze a system $\mathcal{R}(\mathcal{P})$ of raw schedules. For each processor u, we construct the raw schedule R_u of length $r^2 \ge n$ by concatenating ($\circ$) distinct L_i, where $i \in P_u$. Specifically, we let $R_u = L_{t_u^1} \circ L_{t_u^2} \circ \dots \circ L_{t_u^r}$. Thus the raw schedule for processor u is $\langle t_{t_u^1}^1, .., t_{t_u^1}^r,\ t_{t_u^2}^1, .., t_{t_u^2}^r,\ .....,\ t_{t_u^r}^1, .., t_{t_u^r}^r \rangle$. Given $R_u = \langle \tau_u^1, \dots, \tau_u^{r^2} \rangle$ we define $R_u^a = \langle \tau_u^1, \dots, \tau_u^a \rangle$ to be the the prefix of R_u of length a, and $T_u^a = \{\tau_u^1, \dots, \tau_u^a\}$ for $0 \le a \le r^2$.

A direct consequence of Theorem 3 is that raw schedules can be constructed efficiently.

Theorem 4. *Each raw schedule in $\mathcal{R}(\mathcal{P})$ can be constructed in $O(n)$ time.*

Note that it is not necessary to precompute the entire raw schedule, instead it can be computed in r-size segments as needed. Some of the tasks in a raw schedule may be repeated and consequently the number of distinct tasks in a raw schedule of length r^2 may be smaller than r^2 – naturally processors do not execute repeated instances of tasks. For the proof of graceful increase of pairwise redundancy it is important to show that the number of distinct tasks in our raw schedules increases gracefully.

Theorem 5. *For any $R_u = L_{t_u^1} \circ L_{t_u^2} \circ \dots \circ L_{t_u^r} = \langle \tau^1, \dots, \tau^{r^2} \rangle$ and $1 \le a \le r^2$: $|T_u^a| = |\{\tau^1, \dots, \tau^a\}| \ge (\lceil \frac{a}{r} \rceil - 1)(r - \frac{1}{2}(\lceil \frac{a}{r} \rceil - 2)) + \max\{0,\ a - (\lceil \frac{a}{r} \rceil - 1)(r+1)\}$.*

Proof. Consider the task τ^a. It appears in $L_{t_u^i}$, where $i = \lceil \frac{a}{r} \rceil$. For tasks that appear in plans $P_{t_u^1}, \dots, P_{t_u^{i-1}}$ the number of repeated tasks is at most $1 + \dots + (i-2) = (i-1)(i-2)/2$ because $\mathcal{P}$ is a 1-bounded system of plans (any two of these plans intersect by exactly one, see Theorems 3). Hence there are at least $(i-1)r - (i-1)(i-2)/2$ distinct tasks in the raw schedule $L_{t_u^1} \circ \dots \circ L_{t_u^{i-1}}$.

We now assess any additional distinct tasks appearing in $P_{t_u^i}$. Task τ^a is the task number $a - (i-1)r$ in $L_{t_u^i}$. Since $\mathcal{P}$ is 1-bounded, up to $i-1$ tasks in $P_{t_u^i}$ may already be contained $P_{t_u^1}, \dots, P_{t_u^{i-1}}$. Of course in no case may the number of redundant tasks exceed $a - (i-1)r$. Hence the number of additional distinct tasks from $P_{t_u^i}$ is at least $\max\{0,\ a-(i-1)r-(i-1)\} = \max\{0,\ a-(i-1)(r+1)\}$.

Corollary 3. *Any R_u contains at least $\frac{1}{2}(r^2 + r) = \frac{1}{2}n + r - \frac{1}{2}$ distinct tasks.*

Together with Theorem 4, this result also shows that the schedule computation is fully amortized, since it takes $O(n)$ time to compute a schedule that includes more than $n/2$ distinct tasks.

For any processors u and w we wish to determine $\{u, w\}$-waste as u and w progress through the raw schedules R_u and R_w. We now show that for $1 \le a, b \le r^2$ the size of $T_u^a \cap T_w^b$ grows gracefully as a and b increase.

Theorem 6. *For any R_u, R_w and $0 \le a, b \le r^2$: $|T_u^a \cap T_w^b| \le \min\{\ a,\ b,\ r - 1 + \lceil \frac{a}{r} \rceil \cdot \lceil \frac{b}{r} \rceil\ \}$.*

Proof. By the definition of $\mathcal{P}$ and the raw schedules R_u and R_w, $\tau_u^a \in P_{t_u^i}$, where $i = \lceil \frac{a}{r} \rceil$, and $\tau_w^b \in P_{t_w^j}$, where $j = \lceil \frac{b}{r} \rceil$. Therefore, $T_u^a \subseteq P_{t_u^1} \cup \ldots \cup P_{t_u^i}$ and $T_w^b \subseteq P_{t_w^1} \cup \ldots \cup P_{t_w^j}$. Consequently,

$$T_u^a \cap T_w^b \subseteq (P_{t_u^1} \cup \ldots \cup P_{t_u^i}) \cap (P_{t_w^1} \cup \ldots \cup P_{t_w^j}) = \bigcup_{1 \le x \le i,\ 1 \le y \le j} (P_{t_u^x} \cap P_{t_w^y}).$$

Since the system of plans $\mathcal{P}$ is 1-bounded, R_u and R_w contain at most one common P_z for some $1 \le z \le r$. In the worst case, for the corresponding P_z, this contributes $|P_z \cap P_z| = |P_z| = r$ tasks to the intersection of T_u^a and T_w^b. On the other hand, $|P_{t_u^x} \cap P_{t_w^y}| \le 1$ when both t_u^x and t_w^y are not z, again because $\mathcal{P}$ is 1-bounded. Thus, $|T_u^a \cap T_w^b| \le r + |\bigcup_{1 \le x \le i,\ 1 \le y \le j,\ x \ne y \ne z} (P_{t_u^x} \cap P_{t_w^y})| \le r + i \cdot j - 1$. Finally, the overlap cannot be greater than $\min\{a, b\}$.

In the following theorem we show how the useful work (not redundant) grows as processors progress through their schedules.

Theorem 7. *For any processors u and w:*
(a) If $i + j \le r$ then $|T_u^{(i \cdot r)} \cup T_w^{(j \cdot r)}| \ge r(i + j) - r + 1 - \frac{1}{2}((i + j)^2 + i + j)$,
(b) If $i + j > r$ then $|T_u^{(i \cdot r)} \cup T_w^{(j \cdot r)}| \ge \frac{r^2}{2} - \frac{r}{2} + \frac{9}{8}$.

Proof. By Theorem 5 $|T_u^{(i \cdot r)}| \ge i \cdot r - i(i - 1)/2$ and $|T_w^{(j \cdot r)}| \ge j \cdot r - j(j - 1)/2$, and by Theorem 6 $|T_u^{(i \cdot r)} \cap T_w^{(j \cdot r)}| \le r - 1 + i \cdot j$. Thus:

$$|T_u^{(i \cdot r)} \cup T_w^{(j \cdot r)}| = |T_u^{(i \cdot r)}| + |T_w^{(j \cdot r)}| \ge (i + j)(r - \tfrac{i + j - 1}{2}) - r + 1$$

Consider the function $f(i + j) = f(x) = x \cdot (r - \frac{x - 1}{2}) - r + 1 = -\frac{1}{2}x^2 + (r + \frac{1}{2})x + (1 - r)$. It is nonnegative for $2 \le x \le 2r$. Additionally $f(x)$ grows from r, for $x = 2$, to a global maximum of $\frac{r^2}{2} - \frac{r}{2} + \frac{9}{8}$, for $x = r + \frac{1}{2}$, and then decreases to 1, for $x = 2r$. Because $|T_u^{(i \cdot r)}|$ and $|T_w^{(j \cdot r)}|$ are monotone nondecreasing in i and j respectively (the number of tasks already performed by processors cannot decrease), we have that $|T_u^{(i \cdot r)} \cup T_w^{(j \cdot r)}| \ge \frac{r^2}{2} - \frac{r}{2} + \frac{9}{8}$ for $i + j > r$.

Deterministic Construction for Arbitrary n. We now discuss practical aspects of using the system of raw schedules $\mathcal{R}(\mathcal{P})$. Recall that a raw schedule for a processor contains repeated tasks. When a schedule is *compacted* by removing all repeated tasks, the result may contain about half of all tasks (Corollary 3). To construct a *full* schedule that has all $t = n$ distinct tasks, we append the remaining tasks at the end of a compacted schedule (in arbitrary order). For the system $\mathcal{R}(\mathcal{P})$ we call such a system of schedules $\mathcal{F}(\mathcal{P}) = \langle F_1, \ldots, F_n \rangle$. For a schedule F_i we write N_i to denote the corresponding plan. In this section we use our results obtained for raw schedules to establish a bound on pairwise overlap for $\mathcal{F}(\mathcal{P})$. Recall that by construction, the length of $\mathcal{F}(\mathcal{P})$ is $q^2 + 1 + 1$, where q is a prime. We show that common padding techniques can be used to construct schedules for arbitrary $n = t$ such that the pairwise overlap is similarly bounded.

First we analyze overlaps for a system of schedules $\mathcal{F}(\mathcal{P})$. Assume that a processor u advanced to task number $i \cdot r$ in its raw schedule R_u ($1 \le i \le r$). Then, by Theorem 5, it has executed at least $i(r - \frac{i-1}{2})$ distinct tasks. Conversely, for a given x we can define $g(x, r)$ to be the number of segments of the raw schedules R_u that are sufficient to include x distinct tasks, i.e., $|T_u^{r \cdot g(x,r)}| \ge x$. Solving the quadratic equation $g(x,r)(r-\frac{g(x,r)-1}{2})=x$ yields $g(x,r)=\lceil\frac{1+2r-\sqrt{(1+2r)^2-8x}}{2}\rceil$, for $x=0,\ldots,\frac{1}{2}(r^2+r)$ (observe that $g(0,r)=0, g(1,r)=1, g(r,r)=1, g(r+1,r)=2, g(\frac{1}{2}(r^2+r),r)=r$). In the next theorem we use the definition of g and the result from Theorem 6 to construct a system of schedules with bounded overlaps (see Figure 1.b for the plot of the upper bound).

Theorem 8. *For $n=q^2+q+1$, $q=r-1$ prime, the system of schedules $\mathcal{F}(\mathcal{P})$ can be constructed deterministically in time $O(n)$ independently for each processor. Pairwise overlaps are bounded by:*

$$|N_u^a \cap N_w^b| \leq \begin{cases} \min\{a,b,r-1+g(a,r)\cdot g(b,r)\}\ , & a,b\leq \frac{1}{2}(r^2+r), \\ \min\{a,b\}, & \textit{otherwise.} \end{cases}$$

We next show that for long lengths pairwise overlap is strictly less than $\min\{a,b\}$ (the trivial part of the upper bound shown in Theorem 8). Assume that processors u and w have advanced to task number $i\cdot r$ in R_u and R_w respectively ($1\leq i\leq r$). By Theorem 5 the number of distinct tasks executed by each processor is at least $i(r-\frac{i-1}{2})$. By Theorem 6 the overlap is at most $r-1+i^2$. Equating the two expressions yields an equation, solutions to which tell us for which i the overlap does not exceed the number of distinct tasks in the schedule. The first (trivial) solution $i=1$ simply describes the possibility of two processors executing the same r tasks when the first task identifier in P_u is the same as that of P_w. The second solution $i=\frac{2}{3}(r-1)$, with Theorem 5, gives the number of distinct tasks in each schedule, which is no less than $\frac{4}{9}r^2+\frac{1}{9}(r-5)$. This gives guarantees that, using $\mathcal{R}(\mathcal{P})$, there are no two processors that execute the same subsets of tasks when each executes up to $\frac{4}{9}r^2+\frac{1}{9}(r-5)$ tasks. Hence as long as processors have not executed more that $\frac{4}{9}n-\Theta(\sqrt{n})$ tasks, the nontrivial part of the upper bound in Theorem 8 applies. The remaining tasks (approximately $\frac{5}{9}$ of the tasks) can be chosen by the processors arbitrarily (for example using a permutation) since our approach does not provide non-trivial overlap guarantees in that region. Note however, that for schedules longer than $\frac{4}{9}n$ the lower bound on 2-waste, by Theorem 1, is approximately $\frac{16}{81}n$, which is already linear in n.

We now discuss the case when the number of processors n is not of the form q^2+q+1, for some prime q. Since primes are dense, for any fixed $\epsilon>0$ and sufficiently large n, we can choose[2] a prime p in $O(n)$ time such that $n-1\leq p(p+1)\leq(1+\epsilon)n-1$. Using standard padding techniques we can construct a system of schedules of length n with overlap bounded similarly to Theorem 8. An easy analysis yields that the upper bound is strictly lower than the trivial bound as long as processors advance at most $\frac{4}{9}n-\Theta(\sqrt{n})-\Theta(n\sqrt{\epsilon})$ through their schedules.

In our presentation we assume that a suitable prime is available. The prime can be computed as follows: Find an integer $p\in[\sqrt{n},\sqrt{n}(1+\epsilon)]$ that satisfies: 1) $n-1\leq p(p+1)\leq(1+\epsilon)n-1$, and 2) p is not divisible by any of $2,3,4,5,\ldots,\lceil n^{1/4}(1+\epsilon)\rceil$. This gives $O(\epsilon n^{3/4})$ time algorithm. Alternatively, if

[2] This results from the Prime Number Theorem. Due to lack of space we show this in the technical report [15].

we assume the Extended Riemann Hypothesis, we can use an algorithm from [17] to find the prime in $O(\epsilon\sqrt{n}\log^4 n \log\log\log n)$. In any case the cost is expended once at the beginning of the construction, and this prime can be used multiple times so that this cost can be amortized over long-lived computations. Moreover, this cost does not distort the linear complexity of schedule construction. Finally observe that the schedules are produced in segments of size $\Theta(\sqrt{n})$. Thus if processors become able to communicate prior to the completion of all tasks then at most $\sqrt{n}$ tasks would have been scheduled unnecessarily.

6 Randomized Schedules

In this section we examine randomized schedules that, with high probability, allow us to control waste for the complete range of schedule lengths.

When the processors are endowed with a reasonable source of randomness, a natural candidate scheduling algorithm is RANDOM, where processors select tasks by choosing them uniformly among all tasks they have not yet completed. This amounts to the selection, by each processor i, of a random permutation $\pi_i \in S_{[t]}$ after which the processor proceeds with the tasks in the order given by π_i: $\pi_i(1), \pi_i(2), \ldots$. ($S_{[t]}$ denotes the collection of all permutations of the set $[t]$.)

These permutations $\{\pi_i \mid i \in [n]\}$ induce a system of schemes: specifically, coupled with a length $\ell_i \leq t$ for each processor i, such a family of permutations induces the plans $\mathcal{S}^i_{n,t} = \pi_i([\ell_i])$ which together comprise the scheme $\mathcal{S}[\boldsymbol{\ell}]$. Our goal is to show that these schemes are well behaved for each $\boldsymbol{\ell}$, guaranteeing that waste will be controlled. For 2-waste this amounts to bounding, for each pair i, j and each pair of lengths ℓ_i, ℓ_j, the overlap $|\pi_i([\ell_i]) \cap \pi_j([\ell_j])|$. Observe that when these π_i are selected at random, the expected size of this intersection is $\ell_i\ell_j/t$, and our goal will be to show that with high probability, each such intersection size is near this expected value. This is the subject of Theorem 9 below:

Theorem 9. *Let π_i be a family of n permutations of $[t]$, chosen independently and uniformly at random. Then there is a constant c so that with probability at least $1 - 1/n$, the following is satisfied:*

1. *$\forall i, j \leq n$ and $\forall \ell_i, \ell_j \leq t$, $|\pi_i([\ell_i]) \cap \pi_j([\ell_j])| \leq \frac{\ell_i\ell_j}{t} + \Delta$, for $\Delta = \Delta(\ell_i, \ell_j) = c\max\left(\log n, \sqrt{\frac{\ell_i\ell_j}{t}\log n}\right)$*
2. *$\forall z \leq c\log n$, the number of pairs i, j for which $\left|\pi_i([\sqrt{t}]) \cap \pi_j([\sqrt{t}])\right| > z$ is at most $\frac{n^{3/2}}{z-1}$.*

In particular, for each $\boldsymbol{\ell}$, $w_2(\mathcal{S}[\boldsymbol{\ell}]) \leq \max_{i,j} \frac{\ell_i\ell_j}{t} + \Delta(\ell_i, \ell_j)$.

Observe that Theorem 1 shows that schemes with plans of size ℓ must have ℓ^2/t overlap; hence these randomized schemes, for long regular schedules (i.e., where the plans considered have the same size), offer nearly optimal waste.

The following part of the section is devoted to proving Theorem 9. The analysis is divided into two sections, the first focusing on arbitrary pairs of lengths ℓ_i, ℓ_j, and the second focusing on specifically on "small" lengths $\ell < \sqrt{t}$.

Behavior for arbitrary lengths.
Consider two sets $A \subset [t]$ and $B \subset [t]$, A being selected at random among all sets of size $d_A\sqrt{t}$ and B at random among all sets of size $d_B\sqrt{t}$. Then $\mathsf{Exp}\left[|A \cap B|\right] = d_A d_B$. Judicious application of standard Chernoff bounds coupled with an approximation argument yields the following theorem:

Theorem 10. *Let A and B be chosen randomly as above. Then there exists a constant $c > 0$ so that for all n and $t \leq n$, $\mathsf{Pr}\left[|A \cap B| \geq d_A d_B + \Delta(d_A, d_B)\right] \leq \frac{1}{2n^5}$ where $\Delta(d_A, d_B) = c\sqrt{\log n}\left(\sqrt{d_A d_B} + \sqrt{\log n}\right)$. (The constant c is independent of t and n.)*

A proof of this fact can be found in a technical report [15]. Let c be the constant guaranteed by the above corollary and let $\mathcal{B}_{i,j}^{\ell_i,\ell_j}$ be the (bad) event that $|\pi_i([\ell_i]) \cap \pi_j([\ell_j])| \geq d_i d_j + \Delta(d_i, d_j)$, where $\ell_i = d_i\sqrt{t}$ and $\ell_j = d_j\sqrt{t}$. Let an event $\mathcal{B}_1$ be defined as disjunction $\bigvee_{i,j,\ell_i,\ell_j} \mathcal{B}_{i,j}^{\ell_i,\ell_j}$. Considering that $\mathsf{Pr}\left[\mathcal{B}_{i,j}^{\ell_i,\ell_j}\right] \leq \frac{1}{2n^5}$, we have $\mathsf{Pr}\left[\mathcal{B}_1\right] \leq n^4 \times \max_{i,j,\ell_i,\ell_j} \mathsf{Pr}\left[\mathcal{B}_{i,j}^{\ell_i,\ell_j}\right] \leq \frac{1}{2n}$. Hence

$$\mathsf{Pr}\left[\forall i, j, \ell_i, \ell_j, |\pi_i([\ell_i]) \cap \pi_j([\ell_j])| \leq d_i d_j + \Delta(d_i, d_j)\right] \geq 1 - \frac{1}{2n}$$

We now concentrate on the behavior of these schedules for lengths $\ell < \sqrt{t}$.

Behavior for short lengths.
Observe that for any pair (i, j) of schedules, $\mathsf{Exp}\left[\left|\pi_i([\sqrt{t}]) \cap \pi_j([\sqrt{t}])\right|\right] = 1$. We would like to see such behavior for each pair (i, j). Let an event $\mathcal{B}_2$ be defined as $\exists i, j \left|\pi_i([\sqrt{t}]) \cap \pi_j([\sqrt{t}])\right| \geq c_0 \log n$. From the previous argument, there is a constant c_0 so that $\mathsf{Pr}\left[\mathcal{B}_2\right] \leq \frac{1}{2n}$. Considering that the expected value of this intersection is 1, we would like to insure some degree of palatable collective behavior: specifically, we would like to see that few of these overlaps are actually larger than a constant, say. To this end, let $I_{i,j} = \left|\pi_i([\sqrt{t}]) \cap \pi_j([\sqrt{t}])\right|$, and observe that $\mathsf{Exp}\left[\sum_{i<j} I_{i,j}\right] = \binom{n}{2}$. We may write $I_{i,j} = \sum_{m=1}^{\ell} X_m$ where X_m is the indicator variable for the event $\pi_i(m) \in \pi_j([\sqrt{t}])$. Observe that these variables are negatively correlated (i.e., $\mathsf{Cov}\left[X_m, X_{m'}\right] < 0$ for each pair) so that $\mathsf{Var}\left[I_{i,j}\right] \leq \sum_{m=1}^{\ell} \mathsf{Var}\left[X_m\right] \leq \sum_{m=1}^{\ell} \mathsf{Exp}\left[X_m\right] \leq \mathsf{Exp}\left[I_{i,j}\right]$. (Recall that for any indicator variable X, $\mathsf{Var}\left[X\right] \leq \mathsf{Exp}\left[X\right]$.) Observe now that the variables $I_{i,j}$ are pairwise independent so that $\mathsf{Var}\left[\sum_{i<j} I_{i,j}\right] = \sum_{i<j} \mathsf{Var}\left[I_{i,j}\right] \leq \binom{n}{2}$, and an application of Chebyshev's inequality to the quantity $\sum_{i<j} I_{i,j}$, yields $\mathsf{Pr}\left[\sum_{i<j} I_{i,j} - \binom{n}{2} > \lambda\sqrt{\mathsf{Var}\left[\sum_{i<j} I_{i,j}\right]}\right] < \frac{1}{\lambda^2}$, so that

$$\mathsf{Pr}\left[\sum_{i<j} I_{i,j} - \binom{n}{2} > \sqrt{2}n^{1.5}\right] < \frac{1}{2n}.$$

Collecting the pieces yields Theorem 9 above, since $\mathsf{Pr}\left[\mathcal{B}_1\right] \leq \frac{1}{2n}$ and $\mathsf{Pr}\left[\mathcal{B}_2\right] \leq \frac{1}{2n}$, $\mathsf{Pr}\left[\mathcal{B}_1 \vee \mathcal{B}_2\right] \leq \frac{1}{n}$, as desired.

Acknowledgements. We thank Shmuel Zaks for motivating parts of our research, and Ibrahim Matta and Eugene Spiegel for their comments.

References

1. Alon, N., Spencer, J. H.: The probabilistic method. John Wiley & Sons Inc., New York (1992). With an appendix by Paul Erdős, A Wiley-Interscience Publication
2. Aumann, Y., Rabin, M.O.: Clock Construction in Fully Asynchronous Parallel Systems and PRAM Simulation. Foundations of Comp. Sc. (1993) 147–156
3. Chlebus, B.S., De Prisco, R., Shvartsman, A.A.: Performing Tasks on Restartable Message-Passing Processors. Intl Workshop on Distributed Algorithms. Lecture Notes in Computer Science, Vol. 1320. (1997) 96–110
4. Colbourn, C. J., van Oorschot, P. C.: Applications of Combinatorial Designs in Computer Science. ACM Computing Surveys, Vol. 21. **2** (1989)
5. De Prisco, R., Mayer, A., Yung, M.: Time-Optimal Message-Efficient Work Performance in the Presence of Faults. ACM Symposium on Principles of Distributed Computing. (1994) 161–172
6. Dolev, S., Segala, R., Shvartsman, A.A.: Dynamic Load Balancing with Group Communication. Intl Colloquium on Structural Information and Communication Complexity. (1999) 111–125
7. Dwork, C., Halpern, J., Waarts, O.: Performing Work Efficiently in the Presence of Faults. SIAM J. on Computing, Vol. 27 **5**. (1998) 1457–1491
8. Georgiades, S., Mavronicolas, M., Spirakis, P.: Optimal, Distributed Decision-Making: The Case of No Communication. Intl Symposium on Fundamentals of Computation Theory. (1999) 293–303
9. Georgiou, Ch., Shvartsman A: Cooperative Computing with Fragmentable and Mergeable Groups. International Colloquium on Structure of Information and Communication Complexity. (2000) 141–156
10. Hughes, D.R., Piper, F.C.: Design Theory. Cambridge University Press (1985)
11. Ireland, K., Rosen, M.: A Classical Introdiction to Modern Number Theory. 2nd edn. Springer-Verlag (1990)
12. Kanellakis, P.C., Shvartsman, A.A.: Fault-Tolerant Parallel Computation. Kluwer Academic Publishers (1997)
13. Kedem, Z.M., Palem, K.V., Rabin, M.O., Raghunathan, A.: Efficient Program Transformations for Resilient Parallel Computation via Randomization. ACM Symp. on Theory of Comp. (1992) 306–318
14. Knuth, D.E.: The Art of Computer Programming. 2nd edn. Addison-Wesley Publishing Company, Vol. 2. (1981)
15. Malewicz, G., Russell, A., Shvartsman, A.A.: Distributed Cooperation in the Absence of Communication. Technical Report MIT-LCS-TR-804 available at http://theory.lcs.mit.edu/˜alex/mrsTR.ps. (Also: Brief announcement. ACM Symposium on Principles of Distributed Computing. (2000))
16. Martel, C., Park, A., Subramonian, R.: Work-optimal asynchronous algorithms for shared memory parallel computer. SIAM J. on Computing, Vol. 21 **6** (1992) 1070–1099
17. Miller, G.L.: Riemann's Hypothesis and Tests for Primality. Journal of Computer and Systems Sciences, Vol. 13 (1976) 300–317
18. Papadimitriou, C.H., Yannakakis, M.: On the value of information in distributed decision-making. ACM Symp. on Principles of Dist. Computing. (1991) 61–64