An Optical Simulation of Shared Memory [*]

Leslie Ann Goldberg [†]
Department of Computer Science
University of Warwick
Coventry CV4 7AL United Kingdom
leslie@dcs.warwick.ac.uk

Yossi Matias
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
matias@research.att.com

Satish Rao
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
satish@research.nec.com

Abstract

We present a work-optimal randomized algorithm for simulating a shared memory machine (PRAM) on an optical communication parallel computer (OCPC). The OCPC model is motivated by the potential of optical communication for parallel computation. The memory of an OCPC is divided into modules, one module per processor. Each memory module only services a request on a timestep if it receives exactly one memory request.

Our algorithm simulates each step of an $n \lg\lg n$-processor EREW PRAM on an n-processor OCPC in $O(\lg\lg n)$ expected delay. (The probability that the delay is longer than this is at most $n^{-\alpha}$ for any constant α.) The best previous simulation, due to Valiant, required $\Theta(\lg n)$ expected delay.

1 Introduction

The huge bandwidth of the optical medium makes it possible to use optics to build communication networks of very high degree. Eshaghian [8, 9] first studied the computational aspects of parallel architectures with complete optical interconnection networks. The OCPC model is an abstract

[*]A preliminary version of this paper appeared in *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, June 1994.

[†]Part of this work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76AL85000. Part of this work was supported by ESPRIT LTR Project no. 20244 — ALCOM-IT and ESPRIT Project no. 21726 — RAND-II.

model of computation which formalizes important properties of such architectures. It was first introduced by Anderson and Miller [2] and Eshaghian and Kumar [10]. In an n-processor *completely connected Optical Communication Parallel Computer* (n-OCPC) n processors with local memory are connected by a complete network. A computation on this computer consists of a sequence of communication steps. During each communication step each processor can perform some local computation and then send one message to any other processor. If a processor is sent a single message during a communication step then it receives this message successfully, but if it is sent more than one message then the transmissions are garbled and it receives none of them.

While the OCPC seems a reasonable model for optical computers, it has not been used as a programming model to date. The PRAM model, on the other hand, has been extensively used for parallel algorithmic design (e.g., [19, 22, 34]). The convenience of programming on the PRAM is largely due to the fact that the programmer does not have to specify interprocessor communication or to allocate storage in a distributed memory. For the very same reason, the PRAM is considered as highly theoretical, and the task of emulating the PRAM on more realistic models has attracted considerable attention; emulations may enable automatic mapping of PRAM algorithms to weaker models, as well as a better understanding of the relative power of different models. Indeed, many emulations of the PRAM on bounded degree networks were introduced (see, e.g., [1, 21, 23, 31, 32, 36, 37] or [24] for a survey).

In this paper, we present a simulation of an EREW PRAM on the OCPC. In particular, we present a randomized simulation of an $n \lg\lg n$ processor EREW PRAM on an n processor OCPC in which, with high probability, each step of the PRAM requires $O(\lg\lg n)$ steps on the OCPC.[1] Our simulation is work optimal, to within a constant factor.

Our results are closely related to previous work on the well studied *distributed memory machine* (DMM) which consists of n processors and n memory modules connected via a complete network of communication. Each processor can access any module in constant time, and each module can service at most one memory request (read or write) at any time. The DMM is thus a weaker model than the shared memory PRAM, in that the memory address space is partitioned into modules with a restricted access imposed on them. We remark that there are several variants of DMM models differing in their contention rules.

Several papers have studied the emulation of a PRAM on various DMM models [31, 21, 38, 35, 6, 20, 7]. Karp *et al.* [20] present $O(\lg\lg n)$ expected delay simulations of various types of PRAM on a CRCW DMM in which each memory module allows concurrent read or write access to at most one of its memory locations during any step. Dietzfelbinger and Meyer auf der Heide [7] improve upon this paper by presenting an $O(\lg\lg n)$ expected delay simulation of an EREW PRAM on the (weaker) c-collision DMM in which any memory module that receives c or fewer read or write requests serves all of them. Although Dietzfelbinger and Meyer auf der Heide require $c \geq 3$ for their analysis

[1]We will refer to the time required to simulate one PRAM step as the *delay* of the simulation.

to work, they report that experiments show that $c = 2$ works as well. The 1-collision DMM is equivalent to the OCPC.

Our result improves on the result of [7] in two ways. First, it is work-optimal. Second, it works for the OCPC (or 1-collision DMM). The previous best known work-optimal simulation of a PRAM on the OCPC is an $O(\lg n)$ delay simulation of Valiant [39]. In addition, unlike [39, 7] we explicitly consider the construction and evaluation of the hash functions used in our simulation algorithm.

1.1 Related work

The OCPC model The OCPC model was first introduced by Anderson and Miller [2] and Eshaghian and Kumar [10], and has been studied by Valiant [39], Geréb-Graus and Tsantilas [12], Gerbessiotis and Valiant [11], Rao [33], Goldberg, Jerrum, Leighton and Rao [17], and Goldberg, Jerrum and MacKenzie [18]. The feasibility of the OCPC from an engineering point of view is discussed in [2, 12]. See also the survey paper of McColl [30] and the references therein.

Computing h-relation on the OCPC A fundamental problem that deals with contention resolution on the OCPC is that of realizing an *h-relation*. In this problem, each processor has at most h messages to send and at most h messages to receive. Following Anderson and Miller [2], Valiant [39], and Geréb-Graus and Tsantilas [12], Goldberg *et al.* [17] solved the problem in time $O(h + \lg\lg n)$ for an n-processor OCPC. A lower bound of $\Omega(\sqrt{\lg\lg n})$ expected time was recently obtained by Goldberg, Jerrum, and MacKenzie [18].

Simulating PRAMs on OCPCs Valiant described a simulation of an EREW PRAM on an OCPC in [39]. More specifically, Valiant gave a constant delay simulation of a Bulk Synchronous Parallel (BSP) computer on the OCPC (there called the S*PRAM), and also gave an $O(\lg n)$ randomized simulation of an $n \lg n$-processor EREW PRAM on an n-processor BSP computer. A simpler simulation with delay $O(\lg n \lg\lg n)$ was given by Geréb-Graus and Tsantilas [12]. Valiant's result is the best previously known simulation of a PRAM on the OCPC.

Independently of our work, MacKenzie, Plaxton and Rajaraman [28], and Meyer auf der Heide, Scheideler and Stemann [27] have shown how to simulate a n processor EREW PRAM on an n-processor OCPC. Both simulations have $\Theta(\lg\lg n)$ expected delay. However, neither simulation is work-optimal, and both simulations require $n^{\Omega(1)}$ storage at each processor.

Simulating PRAMs on DMMs Mehlhorn and Vishkin [31] used a $(\lg n/\lg\lg n)$-universal class of hash functions to achieve a simple simulation of a CRCW PRAM on a CRCW DMM with expected delay $O(\lg n/\lg\lg n)$. An n-processor CRCW PRAM can be simulated on an n-processor EREW DMM in $O(\lg n)$ expected delay using techniques from [39]. The work of this simulation is thus a $\Theta(\lg n)$

factor away from optimality. The best work-optimal simulation of a PRAM on an EREW DMM has delay $O(n^\epsilon)$ [23].

Recently, Karp, Luby and Meyer auf der Heide [20] presented a simulation of an n-processor CRCW PRAM on an n-processor CRCW DMM with $O(\lg\lg n)$ delay. They also presented a work-optimal simulation of an $(n \lg\lg n \lg^* n)$-processor EREW PRAM on an n-processor CRCW DMM in $O(\lg\lg n \lg^* n)$ expected delay, and a nearly work-optimal simulation of an $n \lg\lg n$ processor CRCW PRAM on an n-processor CRCW DMM with the same delay. Subsequently, Dietzfelbinger and Meyer auf der Heide [7] presented a simplified (non-optimal) simulation of an n-processor EREW PRAM on an n-processor DMM with $O(\lg\lg n)$ expected delay. The simulation in [20] introduces a powerful technique that incorporates the use of two or three hash functions to map the memory address space into the memory modules, combined with the use of a CRCW PRAM algorithm for perfect hashing (see [16] and references therein). It heavily uses the concurrent read capability of the CRCW DMM. The simulation in [7] circumvents the need for using the CRCW PRAM perfect hashing by an elegant use of an idea from Upfal and Wigderson [38].

1.2 Overview of the algorithm

Our simulation algorithm incorporates techniques and ideas from the simulation algorithms of [20, 7], as well as from the h-relation routing algorithm of [17], as follows.

The simulation in [7] uses three hash functions to map each memory cell of the EREW PRAM to three processors (and memory cells) in the DMM. A write on an EREW memory cell is implemented by writing a value and a time stamp to at least two out of the three associated DMM memory cells. A read of an EREW memory cell is implemented by reading two out of three of the memory cells and choosing the value with the most recent time stamp. Dietzfelbinger and Meyer auf der Heide's proof that their simulation requires only $O(\lg\lg n)$ delay on a 3-collision DMM relies on the fact that, given a randomly generated tripartite hypergraph on $3n$ nodes with ϵn edges, one can, with high probability, remove all the nodes in the hypergraph using the following process.
Repeat $O(\lg\lg n)$ times:

1. Remove all of the nodes with degree at most 3.

2. Remove all resulting trivial hyperedges (hyperedges in which only one incident node remains.)

Each hyperedge corresponds to a read or write of a PRAM memory location: The three vertices correspond to the three processors in the DMM associated with that memory location. Thus, one step of an ϵn node EREW PRAM is implemented by using the process above to deliver at least two out of three of the messages associated with each memory request.

Since we are simulating an $n \lg\lg n$ processor PRAM on an n-node OCPC, we must simultaneously implement the process above for $O(\lg\lg n)$ $3n$-node hypergraphs using only n processors. To do this, we start by sparsifying all of the hypergraphs using ideas from the $(\lg\lg n)$-relation routing algorithm in [17]. That is, we route all but $O(n/\lg^c n)$ messages and we ensure that at most one undelivered message remains at any processor. Even so, implementing the process above in parallel could still require $\Omega(\lg\lg n)$ time steps per iteration since each destination may participate in as many as $(\lg\lg n/\epsilon)$ different hypergraphs. Thus, we must also "copy" each destination in such a manner that each message can locate the appropriate copy of its destination. We then perform the process in each hypergraph, ensuring that the process delivers at most a constant number of messages to each copy of a destination. After that, the messages can be sequentially forwarded to their true destinations in $O(\lg\lg n)$ time.

We remark that, in fact, we cannot directly perform the process above on any of the $O(\lg\lg n)$ hypergraphs since our processors can only receive one message in a time step whereas the processors in [7] can receive three messages in a time step. The details of our solution to this problem can be found in the technical sections.

1.3 Paper outline

We proceed in Section 2 with a high level description of our simulation. In Section 3, we present our algorithm in detail and prove correctness. In Section 4 we deal with the evaluation of the hash function that maps the virtual shared memory to the memory modules.

2 The Simulation

Our objective is to show how to simulate one step of an $n\lg\lg n$ processor EREW PRAM in $O(\lg\lg n)$ time-steps on an n processor OCPC. Our simulation follows [7] in using the following idea from [38]. The memory of the PRAM is hashed using three hash functions, h_1, h_2, and h_3. Thus, each memory cell of the PRAM is stored in three memory cells of the OCPC. To write memory cell x, a processor of the OCPC sends a message to at least two of the processors in $\{h_1(x), h_2(x), h_3(x)\}$. The message contains the new value for cell x and also a time stamp. To read memory cell x, a processor p of the OCPC sends a message to at least two of the processors in $\{h_1(x), h_2(x), h_3(x)\}$. Each of these two processors sends p the value that it has for cell x and also its time stamp for cell x. Processor p uses the value with the later time step. The hash functions h_1, h_2, and h_3 are chosen from the "highly" universal family $\overline{R}^{d,j}_{m,n}$ from [20], which guarantees random-like behavior.

Each OCPC processor will simulate $\lg\lg n$ PRAM processors. Thus, at the start of a PRAM step, each of the OCPC processors will wish to access up to $\lg\lg n$ cells of the PRAM memory. Each processor uses h_1, h_2 and h_3 to obtain the three destinations where each memory cell is stored. Thus, each OCPC processor wants to send messages to up to $3\lg\lg n$ destinations. Our objective is to deliver at least two of the messages associated with every request.

As in [17], we will divide the processors of the OCPC into target groups of size $k = \lg^c n$. We will also divide the $n \lg\lg n$ memory requests into $\lg\lg n/\epsilon$ groups of ϵn requests each for a sufficiently small constant ϵ. We will refer to the set of messages associated with a particular group of memory requests as a "group of messages". The messages will be delivered using the following procedures:

- **Thinning and deliver to target groups.** Initially, the number of messages destined for any given target group may be as high as $4k \lg\lg n$. (We will show that, with high probability, it is no larger than this.) We will use techniques from [17] to route the messages to their target groups. With high probability when this procedure is finished every message will be in the target group of its destination. Furthermore, each processor will have at most one message left to send. For a sufficiently large constant c_2, we will allocate a contiguous block of c_2 processors from the target group to each unfinished message for that destination. All senders will know which processors are allocated for their destination. For a sufficiently large constant c_1, we will ensure that for any of the $\lg\lg n/\epsilon$ groups of ϵn messages, with high probability, all but $O(n/(\lg n)^{c_1})$ of the messages in the group will be delivered to their final destinations.

- **Divide into sub-problems and duplicate.** We now divide the OCPC into $\lg\lg n/\epsilon$ sub-OCPCs, each with $n' = n\epsilon/\lg\lg n$ processors. Each sub-OCPC will work on the sub-problem of delivering the messages corresponding to a particular group of messages. For each sub-OCPC we now make $\lg^2 n'$ copies of the relevant sub-problem, all of which will reside in its processors $1, \ldots, n'/2$. We will also allocate its processors $n'/2+1, \ldots, n'$, as follows. For each outstanding memory request (i.e., for each memory request which has the property that at most one of its three messages was delivered during the previous procedure), we will allocate $\lg^2 n'$ processors. These $\lg^2 n'$ processors will do the book-keeping concerning the request in the $\lg^2 n'$ copies of the sub-problem. Each message will know the identity of the processors responsible for the book-keeping concerning its memory request.

- **Route messages for each sub-problem.** In each copy of each sub-problem we route messages according to the c_2-collision access schedule from Section 3 of [7]. Dietzfelbinger and Meyer auf der Heide prove that with high probability each sub-problem is "good" (this term will be defined later on). We will prove that if a sub-problem is good then for any particular memory request in any particular copy of the sub-problem, the probability that the memory request is satisfied in the c_2-collision access schedule routing is at least 1/2. Also, no destination in any copy of any sub-problem receives more than a constant number ($3c_2$) of messages during the c_2-collision access schedule routing.

- **Combining problem copies and combining sub-problems.** In this procedure we identify a subset S of the set of messages that were delivered by the various copies of the

c_2-collision access schedule routing procedure. The messages in S are chosen in such a way that every processor is the destination of $O(\lg\lg n)$ messages in S. We show that with high probability every memory request in every sub-problem that was created in the "divide into sub-problems and duplicate" procedure will be satisfied if the messages in S are delivered. We deliver the messages in S using the routing algorithm in [17].

3 Simulation details and analysis

Before giving the details and analysis we define the class of hash functions $\overline{R}^{d,j}_{m,n}$ being used and describe its properties that are used in the analysis. In the subsequent subsections we will give the details of each of the procedures described in the previous section.

3.1 The hash functions

The class $\overline{R}^{d,j}_{m,n}$ is taken from [20] and is defined as follows.
Definition of $\overline{R}^{d,j}_{m,n}$: A function from $\overline{R}^{d,j}_{m,n}$ is a combination of functions taken from several classes. Carter and Wegman [4] introduced $H^d_{m,n} \subseteq \{g : [1,\dots,m] \to [1,\dots,n]\}$, the class of universal functions $P(x) \bmod n$ where P is a polynomial of degree $d-1$ over $[1,\dots,m]$. Siegel [35] introduced a class of functions $\overline{H}_{n^j,n} \subseteq \{h : [1,\dots,n^j] \to [1,\dots,n]\}$. (More details on this class are given in Section 4.1.) To choose a random hash function $h : [1,\dots,m] \to [1,\dots,n]$ from $\overline{R}^{d,j}_{m,n}$, one first chooses

- A function f, chosen uniformly at random from $H^d_{m,\sqrt{n}}$
- A function r, chosen uniformly at random from $\overline{H}_{n^j,n}$
- A function s, chosen uniformly at random from H^1_{m,n^j}
- $\sqrt{n}$ integers $a_1,\dots,a_{\sqrt{n}}$, each chosen uniformly at random from the range $[1,\dots,n]$.

The function h is defined by $h(x) = (r(s(x)) + a_{f(x)}) \bmod n$.

As in [20], we say that a family $H_{p,n}$ of hash functions is (μ,k)-universal, if for each $x_1 < \dots < x_j \in \{1,\dots,p\}$, $\ell_1,\dots,\ell_j \in \{1,\dots,n\}$, $j \le k$, it holds that, if the hash function h is drawn uniformly at random from $H_{p,n}$, then $\Pr[h(x_1) = \ell_1,\dots,h(x_j) = \ell_j] \le \mu/n^j$.

Let ℓ be an arbitrary constant and let j and d be large enough relative to ℓ. Let ϵ' be a sufficiently small positive constant. We will use the following properties of the hash functions with respect to a set $S \subseteq [1,\dots,m]$, $n \le |S| \le n^{11/10}$. The first two properties are proven in [20].

Property 3.1 *Let $\overline{R}^{d,j}_{m,n}(s)$ be the restriction of $\overline{R}^{d,j}_{m,n}$ induced by fixing $s \in H^1_{m,n^j}$. If s is chosen uniformly at random from H^1_{m,n^j} then s is "1-perfect" on S with probability at least $1 - n^{-\ell}$. If*

s is "1-perfect" on S then $\overline{R}^{d,j}_{m,n}(s)$ is $(2, n^{\epsilon'})$-universal. (Hence, $\overline{R}^{d,j}_{m,n}(s)$ is $(2, n^{\epsilon'})$-universal with probability at least $1 - n^{-\ell}$.)

Remark: Karp et al. actually prove a stronger version of Property 3.1 which states that $\overline{R}^{d,j}_{m,n}(s)$ is $(1, \sqrt{n})$-universal with probability at least $1 - n^{-\ell}$. We use the weaker version because we will later (in Section 4.1) use the space-efficient implementation of the class $\overline{H}_{n^j,n}$ from [35] which is $(2, n^{\epsilon'})$-universal (in fact, it is $((1+\mathrm{o}(1)), n^{\epsilon'})$-universal) but is not necessarily $(1, \sqrt{n})$-universal (see Section 2 of [35]). Thus, with the space-efficient implementation, we only get the weaker version of Property 3.1.

Property 3.2 *Let f be drawn randomly from $H^d_{m,\sqrt{n}}$. Then with probability at least $1 - n^{-\ell}$ every set $f^{-1}(i) \cap S$ has size at most $2|S|/\sqrt{n}$.*

We can now derive:

Property 3.3 *Let Z be a subset of $[1, \ldots, n]$ and let i be an integer in $[1, \ldots, \sqrt{n}]$. Suppose that $\beta \leq n^{\epsilon'}$. Let h be chosen randomly from $\overline{R}^{d,j}_{m,n}$. (That is, let f, r, s, and $a_1, \ldots, a_{\sqrt{n}}$ be chosen as described above.) The probability that β or more members of $S \cap f^{-1}(i)$ are mapped to Z by h is at most $2n^{-\ell} + \binom{2|S|/\sqrt{n}}{\beta} 2\left(\frac{|Z|}{n}\right)^{\beta}$.*

Proof. By Property 3.2, with probability at least $1 - n^{-\ell}$ every set $f^{-1}(i) \cap S$ has size at most $2|S|/\sqrt{n}$. By Property 3.1, with probability at least $1 - n^{-\ell}$, the hash destinations are $(2, n^{\epsilon'})$-universal. ∎

3.2 Thinning and deliver to target groups

We start out by running the "thinning" procedure from [17], which is based on the algorithm of Anderson and Miller [2]. The procedure runs for $\mathrm{O}(\lg \lg n)$ steps. During each step each sender chooses a message uniformly at random from the set of messages that it has not yet sent successfully and it sends the message to its destination with a certain probability. Let $h = 32e \lg \lg n$. We prove further below the following lemma.

Lemma 3.1 *With probability at least $1 - 2n^{-\alpha}$ (for any constant α), after the thinning procedure from [17] terminates, there are at most $k/h\lceil c_3 \lg \lg n\rceil$ undelivered messages destined for any particular target group. (c_3 is a constant which must be sufficiently large; it is the constant c_2 from [17].)*

The proof of Lemma 3.1 will use the following lemma.

Lemma 3.2 *With probability at least $1 - n^{-\alpha}$ (for any constant α), each target group of size k is the destination of at most $9k \lg \lg n$ messages.*

Proof. Consider a target group T. By Property 3.1 of the the hash functions, $\overline{R}^{d,j}_{m,n}(s)$ is $(2, n^{\epsilon'})$-universal with high probability. If $\overline{R}^{d,j}_{m,n}(s)$ is $(2, n^{\epsilon'})$-universal then the probability that at least $9k \lg\lg n$ messages have destinations in T is at most

$$\binom{3n \lg\lg n}{9k \lg\lg n} 2 \left(\frac{k}{n}\right)^{9k \lg\lg n}$$

which is at most $2(e/3)^{9k \lg\lg n}$ by Stirling's approximation. ∎

In order to continue with the proof of Lemma 3.1 we need some notation. For every target group T let $S(T)$ denote the set containing all senders that have messages destined for target group T. We will say that a sender is *bad* if it has some message that has the same destination as at least h other messages. We will use the following lemma.

Lemma 3.3 *With probability at least* $1 - n^{-\alpha}$ *(for any constant* α*) every set* $S(T)$ *contains at most* $k/(2h^2\lceil c_3 \lg\lg n\rceil)$ *bad senders.*

Proof. This proof is similar to the proof of Claim 2 in [17]. We include it here for completeness and also to demonstrate how the limited independence is handled. Let $h' = h/2$. For a given target group T let $M(S(T))$ denote the set of messages that are sent by senders in $S(T)$. We will say that a message is *externally bad* with respect to a target group T if the message has the same destination as at least h' other messages that are not sent from senders in $S(T)$. We will say that a message is *internally bad* with respect to a target group T if it has the same destination as at least h' other messages that are sent from senders in $S(T)$. We wish to prove that with probability at least $1 - n^{-\alpha}$ at most $k/(2h^2\lceil c_3 \lg\lg n\rceil)$ of the messages in $M(S(T))$ are either externally or internally bad.

First we consider externally bad messages. We will say that a processor P is *externally crowded* with respect to a target group T if there are at least h' messages which are not in $M(S(T))$ and have destination P. A set of b members of a target group are all externally crowded only if at least bh' messages have destinations in the set. Property 3.1 of the hash functions tells us that with high probability, the destinations are chosen from a $(2, n^{\epsilon'})$-universal family of hash functions. In this case, as long as $b \le n^{\epsilon'}/h'$ the probability that there is a set of b members of a target group that are all externally crowded is at most $n^{-\alpha}$ (for any constant α)[2], plus

$$\left(\frac{n}{k}\right)\binom{k}{b}\binom{9k \lg\lg n}{bh'} 2 \left(\frac{b}{k}\right)^{bh'}.$$

We can use Stirling's approximation to show that for $b = k/h'^6$ this quantity is at most $(2n/k)(3/5)^{k/h'^5}$. Therefore, with probability at least $1 - n^{-\alpha} - (2n/k)(3/5)^{k/h'^5}$ every target group has at most k/h'^6

[2]By Lemma 3.2, $n^{-\alpha}$ is an upper bound on the probability that more than $9k \lg\lg n$ messages are destined for any target group.

processors which are externally crowded with respect the T. Suppose that this is the case. Since the family of hash functions is $(2, n^{\epsilon'})$-universal, the probability that $3\,|M(S(T))|/h'^6$ messages in $M(S(T))$ choose a destination which is externally crowded with respect to T is at most

$$2\binom{|M(S(T))|}{3\,|M(S(T))|/h'^6}\left(\frac{1}{h'^6}\right)^{3\,|M(S(T))|/h'^6},$$

which is at most $2(e/3)^{3\,|M(S(T))|/h'^6}$ by Stirling's approximation. Note that as long as n is sufficiently large then $3\,|M(S(T))|/h'^6 \le k/(4h^2\lceil c_3 \lg\lg n\rceil)$. Also, as long as $|M(S(T))| \ge k/(4h^2\lceil c_3 \lg\lg n\rceil)$ and the constant c (in the definition of k) is sufficiently large, the sum of $(2n/k)(3/5)^{k/h'^5}$ and $(2n/k)(e/3)^{3\,|M(S(T))|/h'^6}$ is at most $n^{-\alpha}$.

We now consider internally bad messages. We start by calculating an upper bound on the probability that a message is internally bad. Lemma 3.2 tells us that with high probability at most $9k \lg\lg n$ messages are destined for any target group. Thus, with high probability, at most $9k \lg\lg n$ messages in $M(S(T))$ are destined for the same target group as the given message. Property 3.1 of the hash functions tells us that with high probability, the destinations are chosen from a $(2, n^{\epsilon'})$-universal family of hash functions. Therefore, the probability that the given message is internally bad is at most

$$2\binom{9k \lg\lg n}{h'}\left(\frac{1}{k}\right)^{h'} \le (2/3)^h .$$

So the expected number of messages in $M(S(T))$ which are internally bad is at most $|M(S(T))|(2/3)^h$.

In order to prove that with high probability the number of internally bad messages is not far from the expectation we will use the following theorem of McDiarmid [29]. (The inequality is a development of the "Azuma martingale inequality"; a similar formulation was also derived by Bollobás in [3].)

Theorem 3.1 [McDiarmid] *Let $x_1, \ldots, x_n$ be independent random variables, with x_i taking values in a set A_i for each i. Suppose that the (measurable) function $f : \prod A_i \to \mathbb{R}$ satisfies $|f(\overline{x}) - f(\overline{x}')| \le c_i$ whenever the vectors $\overline{x}$ and $\overline{x}'$ differ only in the ith coordinate. Let Y be the random variable $f(x_1, \ldots, x_n)$. Then for any $t > 0$,*

$$\Pr(|Y - \mathrm{E}(Y)| \ge t) \le 2\exp\left(-2t^2/\textstyle\sum_{i=1}^n c_i^2\right).$$

If the hash functions h_1, h_2, and h_3 were chosen uniformly at random from the set of functions from $[1, \ldots, m]$ to $[1, \ldots, n]$, the application of the bounded differences inequality would be straightforward. We would take as the random variable x_i the destination of the ith message in $M(S(T))$. We would let Y be the random variable denoting the number of internally bad messages in $M(S(T))$. If we change the value of one of the x_is the value of Y would change by at most $h' + 1$. Plugging these values into the inequality, we would get a sufficiently small failure probability.

However, since h_1, h_2, and h_3 are in fact drawn from the family $\overline{R}^{d,j}_{m,n}$, the x_is are not independent so we cannot apply Theorem 3.1 to them. Instead, we follow the approach used in the proof of Lemma 6.1 in [20]. Consider the independent random variables $a_1, \ldots, a_{\sqrt{n}}$. As before, let Y be a random variable denoting the number of internally bad messages in $M(S(T))$. Let Z be the set of all destinations of messages in $M(S(T))$. (The size of Z is at most $|M(S(T))|$, which is at most $27k(\lg\lg n)^2$ (with high probability), by Lemma 3.2.) Suppose that we change one of the a_is. By Property 3.3 of the hash functions, the probability that β or more members of $M(S(T))$ change destination is at most $2n^{-\alpha} + 2\binom{6n\lg\lg n/\sqrt{n}}{\beta}\left(\frac{27k(\lg\lg n)^2}{n}\right)^{\beta}$. This probability is sufficiently small as long as the constant β is sufficiently large. So suppose that at most β members of $M(S(T))$ change destination. Each of those may make at most $h'+1$ members of $M(S(T))$ become internally bad. Therefore, if we change one a_i we change Y by at most $\beta(h'+1)$. Therefore, by Theorem 3.1 the probability that $Y \geq k/(4h^2\lceil c_3 \lg\lg n\rceil)$ is at most

$$2\exp\left(\frac{-2\left(\frac{k}{4h^2\lceil c_3\lg\lg n\rceil} - \mathrm{E}(Y)\right)^2}{(|M(S(T))|\,\beta^2(h'+1)^2)}\right).$$

(The $|M(S(T))|$ appears in the denominator because there are $|M(S(T))|$ random variables in $\{a_1, \ldots, a_{\sqrt{n}}\}$ that affect the destinations of messages in $M(S(T))$. Changing any of these random variables could change Y by at most $\beta(h'+1)$. Changing any of the other random variables in $\{a_1, \ldots, a_{\sqrt{n}}\}$ does not change Y.) Since $\mathrm{E}(Y) \leq \frac{k}{8h^2\lceil c_3\lg\lg n\rceil}$ (for big enough n) and, with high probability (by Lemma 3.2), $|M(S(T))| \leq 27k(\lg\lg n)^2$, the probability is at most

$$2\exp(-k/(32h^4\lceil c_3\lg\lg n\rceil^2 27(\lg\lg n)^2\beta^2(h'+1)^2)).$$

This quantity is at most $\frac{1}{2}n^{-\alpha}(k/n)$ as long as c is sufficiently large. This concludes the proof of Lemma 3.3. ∎

The following lemma is proved in [17] (just after Lemma 3'). (The proof of the lemma uses the fact that $|S(T)| \leq 9k\lg\lg n$, which is true with high probability, according to Lemma 3.2.)

Lemma 3.4 *With probability at least $1 - n^{-\alpha}$ the number of messages destined for any target group that start at good senders but are not delivered during the thinning procedure from [17] is at most $k/(2h\lceil c_3\lg\lg n\rceil)$.*

Proof of Lemma 3.1. We conclude that with probability at least $1-2n^{-\alpha}$ the number of undelivered messages destined for any given target group after the thinning procedure terminates is at most $k/(h\lceil c_3\lg\lg n\rceil)$. ∎

After the "thinning" procedure from [17] terminates we will use the "spreading" procedure from [17] to spread out the unfinished requests so that each processor has at most one unfinished message to deliver. As part of the spreading procedure we will allocate one processor to do the book-keeping associated with each memory request and we will ensure that all messages associated with the request know the identity of this processor. During this procedure of our simulation the three messages associated with a request may be sent to various processors but they will keep the book-keeping processor informed about their whereabouts.

After the "spreading", we will use the "deliver to target groups" procedure from [17] to deliver the rest of the messages to their target groups in $O(\lg\lg n)$ steps. With probability at least $1-n^{-\alpha}$ (for any constant α) every message will be in its target group at the end of the "deliver to target group" procedure. Furthermore, each sender will have at most 2 undelivered messages to send and (by Lemma 3.1), the number of unfinished messages in a target group will be less than k. At this point we can sort the messages in the target groups by destination. After the sorting, each sender will have at most one message to send.

We now wish to allocate a contiguous block of c_2 processors from the appropriate target group to each unfinished destination (for a sufficiently large constant c_2). We wish to do the allocation in such a way that all senders know which processors are allocated for their destination. We do this as follows. If a destination is the destination of fewer than c_2 requests we simply deliver them. Otherwise, we allocate c_2 processors for the destination. The processors allocated will be the first c_2 processors with requests for that destination.

At this point we wish to send all but $O(n2^{-c_1 \lg\lg n})$ of the messages in any group to their final destinations. We will say that a message is *bad* if its destination is also the destination of at least $c_1 \lg\lg n$ other messages. We will use the following lemma.

Lemma 3.5 *With probability at least $1-n^{-\alpha}$ (for any constant α) at most $O(n2^{-c_1 \lg\lg n})$ of the messages in any group of messages are bad.*

Proof. This proof is similar to the second part of the proof of Lemma 3.3. By Property 3.1 of the hash functions, the destinations are chosen from a $(2, n^{\epsilon'})$-universal family of hash functions with high probability. In this case, the probability that a given message is bad is at most $2\binom{3n\lg\lg n}{c_1\lg\lg n} n^{-c_1 \lg\lg n}$. By Stirling's approximation, this is at most $2(3e/c_1)^{c_1 \lg\lg n}$ which is at most $2^{-c_1 \lg\lg n}$ for $c_1 \geq 7e$. Therefore, the expected number of bad messages in a group is at most $\epsilon n 2^{-c_1 \lg\lg n}$.

We now use Theorem 3.1 (the bounded differences inequality) to prove that with high probability the number of bad messages in a group is not much more than the expectation.

As in the case of Lemma 3.3, the bounded differences inequality would be straightforward if the hash functions h_1, h_2, and h_3 were chosen uniformly at random from the set of functions from $[1,\ldots,m]$ to $[1,\ldots,n]$. We would take as the random variable x_i the destination of the ith message and we would let Y be the random variable denoting the number of bad messages. If we change the value of one of the x_is the value of Y would change by at most $c_1 \lg\lg n + 1$. Therefore, we would obtain the following inequality.

$$\Pr(Y \geq 2E) \leq 2\exp(-2E^2/(\epsilon n(c_1 \lg\lg n + 1)^2)).$$

However, since h_1, h_2, and h_3 are in fact drawn from the family $\overline{R}^{d,j}_{m,n}$, we again follow the approach used in the proof of Lemma 6.1 in [20]. Consider the independent random variables $a_1, \ldots, a_{\sqrt{n}}$. Let Y be a random variable denoting the number of bad messages. If we change the value of one of the a_is then, with high probability at most $6n \lg\lg n/\sqrt{n}$ messages get new destinations. (This follows from Property 3.2 of the hash functions.) Each new destination could cause at most $c_1 \lg\lg n + 1$ messages to become bad. Thus, changing one of the a_is could change Y by at most $6\sqrt{n} \lg\lg n(c_1 \lg\lg n + 1)$. So, by the bounded differences inequality,

$$\begin{aligned} &\Pr(Y \geq 2E) \leq \\ &2\exp(-2E^2/(\sqrt{n}36n(\lg\lg n)^2(c_1 \lg\lg n + 1)^2))\ , \end{aligned}$$

which is sufficiently small. ■

Given Lemma 3.5, it suffices to route $c_1 \lg\lg n$ messages to each destination. This can be done in $O(\lg\lg n)$ steps since the messages are sorted by destination. At this point we have finished the "thinning and deliver to target groups" procedure. The book-keeping processor associated with every memory request now cancels the request if at least two of its messages were delivered. If the request is canceled then the third message is deleted.

3.3 Divide into sub-problems and duplicate

Our goal is to divide the OCPC into $\lg\lg n/\epsilon$ sub-OCPCs, each of which has $n' = n\epsilon/\lg\lg n$ processors. Each sub-OCPC will work on the sub-problem of delivering the messages corresponding to a particular group of messages. For each sub-OCPC we wish to make $\lg^2(n')$ copies of the relevant sub-problem, all of which will reside in its processors $1, \ldots, n'/2$.

We will use an approximate compaction tool to divide the problem into sub-problems and to make copies of the problem. (For similar tools see [5, 15, 25, 26].) Given

- an n-OCPC in which at most s senders each have one message to send,
- a set of βs receivers which is known to all of the senders,

the (s, β) *approximate compaction problem* is to deliver all of the messages to the set of receivers in such a way that each receiver receives at most one message.

The following lemma is from [17].

Lemma 3.6 *For any positive constant α there is a positive constant c_3 such that the $(s, \lceil c_3 \lg\lg n \rceil)$ approximate compaction problem can be solved in $O(\lg\lg n)$ communication steps with failure probability at most $\alpha^{-\sqrt{s}} + s^{-\alpha}$*

We proved in the previous subsection that, with high probability, when the "thinning and deliver to target groups" procedure terminates, the number of undelivered messages is at most $3n \lg\lg n 2^{-c_1 \lg\lg n}$. Furthermore, every message is in the target group of its destination and each processor will have at most one message left to send.

The number of unfinished target groups is at most the number of unfinished messages, which is at most

$$3n \lg\lg n 2^{-c_1 \lg\lg n} \leq n'/(2\lg^2(n')k^2\lceil c_3 \lg\lg n\rceil)$$

for a sufficiently large c_1. Therefore, with high probability (by Lemma 3.6), we can compact one message from the first processor in each unfinished target group to the first $n'/(2\lg^2(n')k^2)$ processors in the n-OCPC. Having done that, we can copy each of the unfinished target groups to one of the first $n'/(2\lg^2(n')k)$ target groups in the n-OCPC. Next, we can use doubling to make $\lg^2(n')$ copies of each unfinished target group. All of these copies will reside in the first $n'/(2k)$ target groups in the n-OCPC.

At this point, the entire problem is copied $\lg^2(n')$ times into the first $n'/(2k)$ target groups in the n-OCPC. These $n'/(2k)$ target groups will form the first half of the processors in the first n'-processor sub-OCPC. Our objective is to use the first sub-OCPC to solve the sub-problem of delivering the messages in the first group of messages. The sub-OCPC will do this by simply ignoring all messages that are not in the first group of messages.

The $\lg^2(n')$ copies of the entire problem can now be copied into the remaining $\lg\lg n/\epsilon - 1$ sub-OCPCs. The jth sub-OCPC will ignore all messages that are not in the jth group of messages.

Our next goal is to allocate the processors $n'/2, \ldots, n'$ of each sub-OCPC such that for each outstanding memory request (i.e., for each memory request which has the property that at most one of its three messages was delivered during the previous procedure), we allocate $\lg^2(n')$ processors. (These $\lg^2(n')$ processors will do the book-keeping concerning the request in the $\lg^2(n')$ copies of the sub-problem.)

The allocation can be done in the same way that the problem was split and copied because the number of remaining requests is at most $3n \lg\lg n 2^{-c_1 \lg\lg n}$.

3.4 Route messages for each sub-problem

Consider a particular copy of a particular sub-problem. Lemma 3.5 tells us that with high probability at most $O(n2^{-c_1 \lg\lg n})$ of the memory requests from the ϵn memory requests associated with this sub-problem remain. Although each processor has at most one message to send, there is a book-keeping processor allocated to each memory request and each message knows the identity of its book-keeping processor. Furthermore, there is a block of c_2 contiguous processors allocated to each unfinished destination and each sender knows which processors are allocated to its destination. For $i \in \{1, 2, 3\}$ we will say that a message is an "i-message" if it obtained its destination using hash function i.

We now route messages according to the c_2-collision access schedule from Section 3 of Dietzfelbinger and Meyer auf der Heide's paper [7]. Each round of the access schedule is defined as follows.
For $i = 1, 2, 3$:

a. For all destinations d in parallel, repeat $\lceil c_2 \lg(2c_2) \rceil$ times: Each i-message with destination d that is not already waiting at one of the c_2 processors allocated to d picks a random processor from those allocated to d and sends there. Each of the allocated processors will only accept one message.

b. Each destination d now checks whether there are any other i-messages destined for d (that is, whether there are any i-messages with destination d that are not at the allocated processors). To do this, the first of the c_2 processors allocated to d sends to d. Also, any i-messages with destination d that have not yet been successful in reaching one of the c_2 processors allocated to d send to d. Then the first of the c_2 processors allocated to d tells d whether or not it had a collision.

c. For each destination d, if all of the i-messages destined for d are at the processors allocated to d then these messages are delivered. Otherwise, no requests are delivered.

d. The book-keeping processor associated with each memory request checks which of the messages associated with the requests were delivered. If at least 2 of the messages associated with the request have been delivered then the request is canceled and the third message is deleted.

Note that no destination receives more than $3c_2$ messages during the c_2-collision access schedule routing. We use the following lemma:

Lemma 3.7 *During one round of the c_2-collision access schedule routing procedure any processor that is the destination of at most c_2 i-messages gets* all *of the i messages with probability at least* $1/2$ *(and none of them with the remaining probability). Any processor that is the destination of more than c_2 i-messages receives none of them.*

Proof. If d is the destination of at most c_2 i-messages then the probability that one of them fails to reach the allocated processors in $\ell = \lceil c_2 \lg(2c_2) \rceil$ attempts is at most $c_2(1 - 1/c_2)^\ell \leq 1/2$. ∎

In their analysis of the c_2-collision access schedule routing procedure (as implemented on a c_2-collision DMM), Dietzfelbinger and Meyer auf der Heide define a hypergraph $H = (V, E)$ for a set of memory requests $x_1, ..., x_{\epsilon n}$ with vertex set $V = \{v_{rt} \mid 1 \leq r \leq 3, 1 \leq t \leq n\}$ and hyperedge set $E = \{\{v_{1,h_1(x_i)}, v_{2,h_2(x_i)}, v_{3,h_3(x_i)}\} \mid 1 \leq i \leq \epsilon n\}$.

In light of Lemma 3.7, we can view the c_2-collision access schedule routing as a process on H. In each round, the process removes each node with degree at most c_2 (i.e., the i-messages destined for the processor are delivered) with probability at least $1/2$. Then the process removes each hyperedge that consists of only one node (i.e., memory requests are canceled if at least two of the messages associated with the request are delivered).

Following Dietzfelbinger and Meyer auf der Heide, we will say that H is s-good if

1. The largest connected component in H has at most $\alpha = \alpha(s) \lg n$ nodes.

2. Every set $A \subseteq V$ intersects fewer than $|A| + s$ hyperedges from E in at least 2 points.

Dietzfelbinger and Meyer auf der Heide prove the following lemma. (The proof presented in [7] is based on the assumption that h_1, h_2, and h_3 are chosen uniformly at random from the set of functions from $[1, \ldots, m]$ to $[1, \ldots, n]$. However, the lemma is also true if h_1, h_2, and h_3 are chosen randomly from $\overline{R}^{d,j}_{m,n}$.)

Lemma 3.8 *The probability that H is s-good is $1 - \mathrm{O}(n^{-s})$.*

We will prove the following lemma.

Lemma 3.9 *Suppose that H is s-good for some positive constant s. Then the probability that any particular memory request is satisfied after $\mathrm{O}(\lg \lg n)$ rounds of routing according to the c_2-collision access schedule is at least $1/2$.*

Proof. Let H_t denote the hypergraph obtained by applying t rounds of the c_2-collision access schedule routing process to H. Dietzfelbinger and Meyer auf der Heide have made the following observation [7].

Observation 3.1 *If H is s-good and $A \subseteq V$ is a component of H_t for some $t \geq 0$, then A contains at most $3|A|/(c_2 + 1) + 3s/(c_2 + 1)$ nodes of degree larger than c_t in H_t.*

We will use the following lemma.

Lemma 3.10 *Suppose that H is s-good. Let r be an edge in a component of size $\ell \geq s$ of H_t for some $t \geq 0$. If $c_2 \geq 23$ then with probability at least $1 - \exp(-\ell/54)$ the component of r in H_{t+1} has size at most $5\ell/6$.*

Proof. Let $b = 3(\ell + s)/(c_2 + 1)$. By Observation 3.1 and Lemma 3.7, the expected number of nodes in the component of r in H_{t+1} is at most $\ell/2 + b/2$. Using a Chernoff bound, we see that the probability that there are at most $4/3(\ell/2 + b/2) \leq 5\ell/6$ nodes is at least $1 - \exp(-(\ell/2 + b/2)/27)$. ■

Using Lemma 3.10, we conclude that for some constant $c_4 \geq s$, with probability at least 3/4, $O(\lg\lg n)$ rounds of the c_2-collision access schedule routing procedure reduce the size of the component of a given memory request r to at most c_4. We conclude the proof of Lemma 3.9 by observing that as long as $c_2 > 3s + 2$, $O(1)$ rounds will, with probability at least 3/4, further reduce the component to size 1. ■

3.5 Combining problem copies and combining sub-problems

Let us focus our attention on the jth sub-problem. Let S_j be the set of messages that were in the sub-problem when it was created. Let S'_j be the subset containing all messages in S_j that are delivered in at least $\lg^2(n')/9$ copies of the c_2-collision access schedule routing procedure.

Note that when the c_2-collision access schedule routing procedure terminates the $\lg^2(n')$ processors per memory request that were allocated in the "divide and copy" procedure to do book-keeping can inform all of the the messages in S_j (in the first copy of the sub-problem) whether or not they are in S'_j.

We will prove the following lemma.

Lemma 3.11 *With probability at least $1 - n^{-\alpha}$ (for any positive constant α) each set S'_j has the following properties.*

1. *Each processor is the destination of at most $27c_2$ messages in S'_j.*
2. *Each memory request in the jth sub-problem will be satisfied if the messages in S'_j are delivered.*

If each set S'_j has the properties described in Lemma 3.11 (as it will, with high probability), then we can satisfy all of the memory requests in $O(\lg\lg n)$ steps by routing the messages in $S = \bigcup_j S'_j$. These messages form a $27c_2 \lg\lg n/\epsilon$-relation, so we can use the routing algorithm in [17] to route the messages.

To prove Lemma 3.11 we use the following lemma and the following observation.

Lemma 3.12 *With probability at least $1 - n^{-\alpha}$ (for any constant α) every memory request in every sub-problem is satisfied in at least $\lg^2(n')/3$ of the $\lg^2(n')$ copies of the c_2-collision access schedule routing procedure.*

Proof. Suppose that every sub-problem is such that the corresponding hypergraph is s-good. (Lemma 3.8 shows that this is so with high probability, as long as s is chosen to be sufficiently large.) Consider a particular memory request in a particular sub-problem. Lemma 3.9 shows that the probability that this request is satisfied in any given copy of the sub-problem is at least $1/2$. A Chernoff bound shows that with probability at least $1 - ne^{-\lg^2(n')/54}$ the request is satisfied in at least $\lg^2(n')/3$ copies. The lemma follows by summing the failure probabilities over particular memory requests. ∎

Observation 3.2 *If x_1, x_2 and x_3 are the three messages in a memory request that is satisfied in at least ℓ copies of the c_2-collision access schedule routing procedure then there is a pair of messages from $\{x_1, x_2, x_3\}$ such that both of the messages in the pair are satisfied in at least $\ell/3$ copies of the procedure. Similarly, if x_1 and x_2 are the two messages in a memory request that is satisfied in at least ℓ copies of the c_2-collision access schedule routing procedure then at least one of x_1 and x_2 is satisfied in at least $\ell/2$ copies of the procedure.*

Proof of Lemma 3.11. The fact that (with high probability) each memory request in the jth sub-problem will be satisfied if the messages in S'_j are delivered follows from Lemma 3.12 and from Observation 3.2. To see that each processor is the destination of at most $27c_2$ messages in S'_j note that a message is a member of S'_j only if it is delivered in at least $\lg^2(n')/9$ copies of the c_2-collision access schedule routing procedure. However, we proved in the previous section that each destination will receive at most $3c_2$ messages in each copy of the procedure. Therefore, at most $27c_2$ messages that have the same destination will be included in S'_j. This completes the proof of Lemma 3.11. ∎

4 Construction and evaluation of the hash function

In the simulation algorithm we have assumed that a hash function h was chosen uniformly at random from the family $\overline{R}^{d,j}_{m,n}$ and is available to every processor for constant time evaluation. When concurrent-read is available in the simulating model, a hash function in use can be kept in the shared memory, and be read as necessary in constant time. The exclusive-read nature of the OCPC model, together with the fact that the function $h \in \overline{R}^{d,j}_{m,n}$ is represented by a polynomial number of memory words, imply a more subtle situation. A straightforward implementation is to keep a copy of the function h at each processor. However, this implies polynomial overheads in both the time of preprocessing for distributing all copies, and in the space dedicated for this function at each processor. In the remainder of this section we describe an efficient implementation in which the function requires only a total of linear space, and its evaluation increases the simulation delay by at most a constant factor.

4.1 The family of hash functions

Our basic approach is: (i) replace the class $\overline{R}^{d,j}_{m,n}$ with a class whose functions h have similar properties, but can be represented in $O(n^{\epsilon})$ space, where $1/2 \leq \epsilon < 1$; the modified class will still satisfy Properties 3.1–3.3. (ii) make $O(n^{1-\epsilon})$ copies of the selected function h; and (iii) make sure that at each simulation step the number of processors that need to read a component of h is bounded by $O(n^{1-\epsilon} \lg\lg n)$, an average of $O(\lg\lg n)$ per copy, thereby enable the use of an efficient $\lg\lg n$-relation algorithm for the read operation. (A similar approach of making duplicates to reduce contention was used in [14], in implementing a perfect hash function on the QRQW PRAM.) To implement the approach sketched above we first modify the definition of $\overline{R}^{d,j}_{m,n}$ from Section 3.1 as follows. To choose a random hash function $h : [1, \ldots, m] \rightarrow [1, \ldots, n]$ from the modified class, one first chooses a function f uniformly at random from $H^{d}_{m,\sqrt{n}}$ and integers $a_1, \ldots, a_{\sqrt{n}}$ uniformly at random from $[1, \ldots, n]$ as before. Similarly, the function r will be chosen uniformly at random from $\overline{H}_{n^j,n}$ as before. However, we will use Siegel's space-efficient implementation of the class $\overline{H}_{n^j,n}$ from [35] which we will explain below and we will make sure that the implementation satisfies an additional property (see Lemma 4.2 below). The function s will no longer be chosen uniformly at random from H^{1}_{m,n^j}. Instead, it will be chosen as follows. Let $t = j/\epsilon$ and let d be a sufficiently large constant. We will choose functions $s_1, s_2, \ldots, s_t$ uniformly at random from the class $H^{d}_{m,n^{\epsilon}}$ and we will define $s(x)$ to be the tuple $\langle s_1(x), \ldots, s_t(x) \rangle$. Finally, we will define $h(x) = (r(s(x)) + a_{f(x)}) \bmod n$ as before. The following lemma shows that Property 3.1 still holds for the new family of hash functions.

Lemma 4.1 *Let $\ell \geq 1$ be arbitrary and let d and j be large enough relative to ℓ. Let $S \subseteq [1, \ldots, m]$, $n \leq |S| \leq n^{11/10}$. If s is chosen randomly as described above then* $\Pr[s$ *is* 1*-perfect on* $S]$ *is at least* $1 - n^{-\ell}$.

Proof. The probability that two given distinct points $x, y \in S$ will collide under s, i.e., that $s(x) = s(y)$, is at most $(2/n^{\epsilon})^t$, since the s_i's are $(2, d)$-universal. The probability that any pair of points from S will collide is therefore at most

$$\binom{|S|}{2} (2/n^{\epsilon})^t \leq n^{22/10-j} 2^{j-1}$$

The lemma follows by taking $j > \ell + 22/10$. ∎

4.2 The implementation of $\overline{H}_{n^j,n}$

We now describe Siegel's space-efficient implementation of the class $\overline{H}_{n^j,n}$ from [35].

Siegel defines a (p, ϵ, d, h)-*weak concentrator* H as a bipartite graph on the sets of vertices I (inputs) and O (outputs), where $|I| = p$, and $|O| = p^{\epsilon}$, that has outdegree d for each node in $|I|$, and that has, for any h inputs, edges matching them one-by-one with some h outputs.

A (p, ϵ, d, h)-weak concentrator H is used to construct a function F by storing d random numbers from $[0, \ldots, p-1]$ at each node of O. On input i, $F(i)$ is computed by evaluating a polynomial hash function of degree $d-1$ whose coefficients are determined by the numbers stored at the neighbors of i in O. Siegel showed that the family of hash functions F so defined is a $(1, h)$-universal family of hash functions mapping $[0, p-1] \mapsto [0, p-1]$.

Let H be a $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator. Siegel showed that the Cartesian product $G = H^t$ is a $(n^j, \epsilon, d^t, n^{\epsilon'})$-weak concentrator. The graph G can therefore be used to construct a $(1, n^{\epsilon'})$-universal family of hash functions mapping $[1, ..., n^j]$ to $[1, ..., n^j]$. One obtains a $(2, n^{\epsilon'})$-universal family of hash functions mapping $[1, \ldots, n^j]$ to $[1, \ldots, n]$ by taking the results modulo n.

In the following lemma, we observe that Siegel's implementation of $\overline{H}_{n^j,n}$ only requiresa graph H with small out-degree. This property will be useful in our OCPC implementation.

Lemma 4.2 *There exists a graph H that is $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator, and which also has the property that every output of H has degree at most $2dn^{\epsilon-\epsilon^2}$.*

Proof. We use a probabilistic construction, as given in [35] for finding an $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator. Suppose that each input of H chooses its d (distinct) neighbors uniformly at random. Siegel proves that the probability that H is not a $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator is at most $n^{-(\epsilon^2-\epsilon')}$. (As long as ϵ' is sufficiently small.) We can now use a Chernoff bound to show that the degree of each output of H is sufficiently small as required. ∎

4.3 Constructing the hash functions

The graph H from Lemma 4.2 can be constructed and be built into the machine when the machine is built. Each of the n^{ϵ} inputs has d neighbors. A set of $n^{1-\epsilon}$ processors is selected and each processor in the set is given the name of these dn^{ϵ} neighbors.

A new hash function h from the modified family is constructed in $O(\lg n)$ steps as follows:

(1) Select (appropriately at random) $s_1, .., s_t$ and f and distribute to all processors.

(2) Each of the $n^{j\epsilon}$ output nodes of $G = H^t$ chooses d^t values in $[0, .., n^j - 1]$. A set of $n^{1-j\epsilon}$ processors is selected for each given output node and each processor in the set is given the d^t values associated with the output node.

(3) The values $a_1, ..., a_{\sqrt{n}}$ are generated. $\sqrt{n}$ sets of $\sqrt{n}$ processors are selected and each processor in a set i is given the value of a_i.

Recall that it may be the case that a new function needs to be constructed (a "re-hash" operation), when the selected one does not satisfy the required properties. (This occurs with polynomially small probability for each parallel step, and with high probability after a polynomial number of steps.)

4.4 Evaluating the hash function

At each simulation step, the hash function is computed for all memory addresses in $O(\lg \lg n)$ time, as described next. Let S be the set of $3n \lg \lg n$ requests from $[1, .., m]$. Recall from Section 4.1 that $h(x) = (r(s(x)) + a_{f(x)}) \bmod n$.

Each processor executes the following steps for each request x:

(1) Compute $s_1(x), ..., s_t(x)$.

(2) Compute the names of the neighbors of $\langle s_1(x), ..., s_t(x) \rangle$ in G.

(3) Read the values corresponding to the neighbors of $\langle s_1(x), ..., s_t(x) \rangle$ in G.

(4) Apply r to $\langle s_1(x), ..., s_t(x) \rangle$.

(5) Compute $f(x)$.

(6) Read $a_{f(x)}$.

(7) Compute $r(s(x)) + a_{f(x)}$.

The executions of Steps 1,4,5, and 7 are in constant time. The following lemma of Dietzfelbinger, given in [23], is central to the analysis of the other steps.

Lemma 4.3 *Let $X_1, ..., X_n$ be $0-1$ valued, d-independent, equidistributed random variables. Let $\mu = E(X_i)$. Then, for $n \geq d/(2\mu)$,*

$$\Pr\left(\sum_{i=1}^{n}(X_i - \mu) \geq \lambda\right) \leq \frac{\alpha(n\mu)^{d/2}}{\lambda^d}$$

where α is a constant that depends on d but not on n.

Claim 4.4 *In Step 2, with high probability, for every y in $[1, ..., n^\epsilon]$ (i.e., for every input of H) there are at most*
$O(n^{1-\epsilon} \lg \lg n)$ pairs (i, x) such that $x \in S$ and $s_i(x) = y$.

Proof. Note that the set of values $s_i(x) : 1 \leq x \leq m$ is d-independent. Following Kruskal, Rudolph, and Snir [23] we use Lemma 4.3. Fix a y and i and let X_b be a 0-1 random variable which is 1 if and only if s_i maps the b'th member of S to y. μ is $1/n^\epsilon$. Let λ be $|S|/n^\epsilon$. Then the probability that s_i maps more than 2λ to y is $O(n^{-d/2(1-\epsilon)})$. Choose d large enough to sum over all i and y. ∎

We conclude that at most $O(n^{1-\epsilon} \lg\lg n)$ processors want to read the information about input y, and so we have a "target group $O(\lg\lg n)$ relation". The requests can be routed using [17].

Claim 4.5 *In Step 3, with high probability, for every output y of G there are at most $O(n^{1-j\epsilon} \lg\lg n)$ values x in S such that $\langle s_1(x), ..., s_t(x)\rangle$ is a neighbor of y in G.*

Proof. Fix $y = \langle y_1, .., y_t\rangle$. Let L_i denote the neighbors of y_i in H. Note that $|L_i| \leq 2dn^{\epsilon-\epsilon^2}$. If $s(x)$ has a neighbor y in G then $s_i(x)$ is in L_i, for $1 \leq i \leq t$.

The probability of this event is at most $(2d/n^{\epsilon^2})^t$. Let X_b be a 0-1 random variable which is 1 if and only if the b-th member x of S has $s(x)$ mapped to y in G. Apply Lemma 4.3: μ is at most $(2d/n^{\epsilon^2})^t$ by Lemma 4.2; let λ be $|S|(2d/n^{\epsilon^2})^t$. The probability that there are more than λ such values x is at most $\alpha n^{-(d/2)(1-j\epsilon)}$. ∎

Given the claim, we have a "target group $O(\lg\lg n)$ relation". The requests can be routed using [17].

It remains to analyze Step 6. By Property 3.2, with probability at least $1 - n^{-\alpha}$ each group needs to be read by at most $6\sqrt{n} \lg\lg n$ of the requests, so we have a "target group $6 \lg\lg n$ relation". The requests can be routed using [17].

5 Conclusions

In this paper we have described a work-optimal algorithm which simulates an $n \lg\lg n$-processor EREW PRAM on an n-processor OCPC with O($\lg\lg n$) expected delay. The probability that the delay is longer than this is at most $n^{-\alpha}$ for any constant α.

It would be interesting to determine whether this is the fastest possible work-optimal simulation. It would also be interesting to discover how much delay is required in order to simulate a CRCW PRAM. We have recently derived an algorithm that simulates an n-processor CRCW PRAM step on an n-processor OCPC in time $O(\lg k + \lg\lg n)$ with high probability, where k is the maximum memory contention of the CRCW step.

The simulation algorithm assumes that k is known. This assumption can be removed by augmenting the OCPC model to include a single bus which can be used to synchronize all of the processors: each processor can broadcast a '1' bit and every processor can determine whether or not any processor is broadcasting a '1' at any given time.

We note that the $\lg k$ term in the simulation algorithm is provably necessary, as implied by an $\Omega(\lg k)$ expected time lower bound for broadcasting the value of a bit to k processors on a QRCW PRAM (and hence on an ERCW), by Gibbons, Matias and Ramachandran (see [14]).

Evidently, the performance of the CRCW simulation depends on the maximum contention. A model that accounts for memory contention was recently proposed in [13]. In this model the run time of each step is a function of the memory contention encountered at this step. Thus, in the sub-model of SIMD-QRQW(log) PRAM, a step in which the maximum memory contention is k is assumed to take $\lg k$ time units.

The CRCW simulation implies that an n-processor SIMD-QRQW(log) PRAM algorithm can be simulated on an n-processor OCPC, augmented with a bus, with delay $O(\lg \lg n)$ with high probability. We note that the SIMD-QRQW(log) PRAM is strictly stronger than the EREW PRAM.

References

[1] H. Alt, T. Hagerup, K. Mehlhorn and F.P. Preparata, Deterministic Simulation of Idealized Parallel Computers on More Realistic Ones. *SIAM Journal of Computing* **16** (1987) 808–835.

[2] R.J. Anderson and G.L. Miller, Optical Communication for Pointer Based Algorithms, Technical Report CRI 88-14, Computer Science Department, University of Southern California, Los Angeles, CA 90089-0782 USA, 1988.

[3] B. Bollobás, Martingales, Isoperimetric Inequalities and Random Graphs, in *Combinatorics* (eds A. Hajnal, L. Lovász, and V. T. Sós), *Colloq. Math. Soc. János Bolyai* **52** (North Holland 1988) 113–139.

[4] J.L. Carter and M.N. Wegman, Universal Classes of Hash Functions, *Journal of Computer and Systems Sciences* **18** (1979) 143–154.

[5] B.S. Chlebus, K. Diks, T. Hagerup, and T. Radzik, New Simulations between CRCW PRAMs, *Proc. Foundations of Computation Theory* **7** , Lecture Notes in Computer Science **380** (Springer-Verlag 1989) 95–104.

[6] M. Dietzfelbinger and F. Meyer auf der Heide, How to Distribute a Dictionary in a Complete Network, *Proceedings of the ACM Symposium On Theory of Computing* **22** (1990) 117–127.

[7] M. Dietzfelbinger and F. Meyer auf der Heide, Simple, Efficient Shared Memory Simulations, *Proceedings of the ACM Symposium On Parallel Algorithms and Architectures* **5** (1993) 110–119.

[8] M.M. Eshaghian, Parallel Computing with Optical Interconnects, PhD thesis, USC 1988.

[9] M.M. Eshaghian, Parallel Algorithms for Image Processing on OMC, *IEEE Transactions on Computers,* **40(7)** (1991) 827–833.

[10] M.M. Eshaghian and V.K.P. Kumar, Optical Arrays for Parallel Processing, Proc. Second Annual Parallel Processing Symposium (1988) 58–71.

[11] A.V. Gerbessiotis and L.G. Valiant, Direct Bulk-Synchronous Parallel Algorithms, *Proceedings of the Scandinavian Workshop on Algorithm Theory* **3** (1992).

[12] M. Geréb-Graus and T. Tsantilas, Efficient Optical Communication in Parallel Computers, *Proceedings of the ACM Symposium On Parallel Algorithms and Architectures* **4** (1992) 41–48.

[13] P.B. Gibbons, Y. Matias, and V. L. Ramachandran. The QRQW PRAM: Accounting for contention in parallel algorithms. *Proceedings of the ACM-SIAM Symposium On Discrete Algorithms* **5** (1994) 638–648.

[14] P.B. Gibbons, Y. Matias, and V. L. Ramachandran. Efficient Low-Contention Parallel Algorithms, *Proceedings of the ACM Symposium On Parallel Algorithms and Architectures* **6** (1994) 236–247.

[15] J. Gil and Y. Matias, Fast Hashing on a PRAM – Designing by Expectation, *Proceedings of the ACM-SIAM Symposium On Discrete Algorithms* **2** (1991) 271–280.

[16] J. Gil, Y. Matias, and U. Vishkin. Towards a Theory of Nearly Constant Time Parallel Algorithms, *Proceedings of the IEEE Symposium on Foundations of Computer Science* **32** (1991) 698–710.

[17] L.A. Goldberg, M. Jerrum, T. Leighton and S. Rao, Doubly Logarithmic Communication Algorithms for Optical Communication Parallel Computers, To appear in this journal. (A preliminary version appeared in *Proceedings of the ACM Symposium On Parallel Algorithms and Architectures* **5** (1993) 300–309.)

[18] L.A. Goldberg, M. Jerrum and P.D. MacKenzie, An $\Omega(\sqrt{\lg\lg n})$ Lower Bound for Routing in Optical Networks, To appear in this journal. (A preliminary version appeared in *Proceedings of the ACM Symposium On Parallel Algorithms and Architectures* **6** (1994) 147–156.)

[19] J. JáJá. *An Introduction to Parallel Algorithms.* (Addison-Wesley, 1992).

[20] R.M. Karp, M. Luby and F. Meyer auf der Heide, Efficient PRAM Simulation on a Distributed Memory Machine, Pre-print, 1994. (A preliminary version of this paper appeared in *Proceedings of the ACM Symposium On Theory of Computing* **24** (1992) 318–326.)

[21] A.R. Karlin and E. Upfal, Parallel Hashing — an Efficient Implementation of Shared Memory, *Proceedings of the ACM Symposium On Theory of Computing* **18** (1986) 160–168.

[22] R.M. Karp and V. Ramachandran, Parallel Algorithms for Shared-Memory Machines, *Handbook of Theoretical Computer Science, Volume A*, (J. van Leeuwen, editor, Elsevier, 1990) 869–941.

[23] C.P. Kruskal, L. Rudolph, and M. Snir, A Complexity Theory of Efficient Parallel Algorithms, *Theoretical Computer Science*, **71** (1990) 95–132.

[24] F.T. Leighton, Methods for Message Routing in Parallel Machines, *Proceedings of the ACM Symposium On Theory of Computing* **24** (1992) 77–96.

[25] Y. Matias, *Highly Parallel Randomized Algorithmics.* PhD thesis, Tel Aviv University, Israel, 1992.

[26] Y. Matias and U. Vishkin, Converting High Probability into Nearly-Constant Time — with Applications to Parallel Hashing, *Proceedings of the ACM Symposium On Theory of Computing* **23** (1991) 307–316.

[27] F. Meyer auf der Heide, C. Scheideler, and V. Stemann, Fast simple dictionaries and shared memory simulation on distributed memory machines; upper and lower bounds, Pre-print 1994.

[28] P.D. MacKenzie, C.G. Plaxton, and R. Rajaraman, On Contention Resolution Protocols and Associated Probabilistic Phenomena, *Proceedings of the ACM Symposium On Theory of Computing* **26** (1994) 153–162. To appear.

[29] C. McDiarmid, On the Method of Bounded Differences, Surveys in Combinatorics, London Math. Soc. Lecture Notes Series **141** (Cambridge Univ. Press, 1989) 148–188.

[30] W.F. McColl. General Purpose Parallel Computing, in Lectures on Parallel Computation, Proc. 1991 ALCOM Spring School on Parallel Computation, Edited by A.M. Gibbons and P. Spirakis, (Cambridge University Press 1993) 337–391.

[31] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21:339–374, 1984.

[32] A.G. Ranade, How to Emulate Shared Memory, *Journal of Computer and Systems Sciences* **42** (1991) 307–326.

[33] S.B. Rao, Properties of an Interconnection Architecture Based on Wavelength Division Multiplexing, Technical Report TR-92-009-3-0054-2, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 USA, 1992.

[34] J.H. Reif, editor, *A Synthesis of Parallel Algorithms* (Morgan-Kaufmann, 1993).

[35] A. Siegel, On Universal Classes of Fast High Performance Hash Functions, Their Time-Space Tradeoff, and Their Applications, *Proceedings of the IEEE Symposium on Foundations of Computer Science* **30** (1989) 20–25.

[36] E. Upfal, Efficient Schemes for Parallel Communications, *Journal of the ACM* **31** (1984) 507–517.

[37] E. Upfal, A Probabilistic Relation Between Desirable and Feasible Models of Parallel Computation, *Proceedings of the ACM Symposium On Theory of Computing* **16** (1984) 258–265.

[38] E. Upfal, A. Wigderson, How to Share Memory in a Distributed System, *Journal of the ACM* **34** (1987) 116–127.

[39] L.G. Valiant, General Purpose Parallel Architectures, Chapter 18 of Handbook of Theoretical Computer Science, Edited by J. van Leeuwen (Elsevier 1990).