Self-Simulation for the Passive Optical Star *†

Pascal Berthomé ‡ Torben Hagerup § Ilan Newman ¶ Assaf Schuster ||

Abstract

Optical technology offers simple interconnection schemes with straightforward layouts that support complex logical interconnection patterns. The Passive Optical Star (POS), in which communication takes place via optical wavelengths agreed upon between senders and receivers, is often suggested as a platform for implementing the optical network: Logically it offers an all-to-all broadcast capability.

We investigate the use of POS optical technology as the communication medium for parallel computing, focusing on the scalability or self-simulation issue, which is a fundamental concern of great importance to the simplicity of algorithm design and program portability. A family of parallel machines is *scalable* or *self-simulating* if any member of the family can simulate any larger member with k times as many processors with a slowdown close to k; i.e., scalability states that if a certain efficiency is achievable on a large machine, then it is achievable on any smaller machine as well.

A POS is *balanced* if the number of available wavelengths equals the number of processors. We show that the balanced POS is highly scalable by presenting a universal randomized simulation of a kn-processor balanced POS on an n-processor balanced POS, for arbitrary integers $n, k \geq 1$, with an expected slowdown of $O(k + \log^* n)$. We also describe a deterministic simulation with a slowdown of $O(\min\{k^2, k + \log n\})$ and show that a slowdown of $O(k)$ is achievable if the communication pattern of each step is known in advance and available for preprocessing. For a certain restricted class of self-simulations we establish matching upper and lower bounds of $\Theta(k^2)$. We also show that the balanced POS is equivalent to a balanced version of the well-known COLLISION CRCW PRAM, where we now take the balance condition to mean that the number of shared memory cells is the same as the number of processors.

*A preliminary version of this paper was presented at the 3rd European Symposium on Algorithms (ESA) in Corfu in September 1995.

†This work was supported in part by ANM, C3, the French-Israeli grant for cooperation in Computer Science, by a grant from the Israeli Ministry of Science, and by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (project ALCOM II).

‡Laboratoire de la Recherche en Informatique, CNRS URA 410, Bât. 490, Université Paris Sud, 91405 Orsay Cedex, France. E-mail: berthome@lri.fr

§Fachbereich Informatik, Johann Wolfgang Goethe-Universität Frankfurt, Robert-Mayer-Str. 11–15, D–60054 Frankfurt am Main, Germany. The work was done while this author was with the Max-Planck-Institut für Informatik, D–66123 Saarbrücken, Germany. E-mail: torben@mpi-sb.mpg.de

¶Mathematics and Computer Science, Haifa University, Haifa, Israel 31905. E-mail: ilan@mathcs.haifa.ac.il

||Department of Computer Science, Technion, Haifa, Israel 32000. E-mail: assafs@cs.technion.ac.il

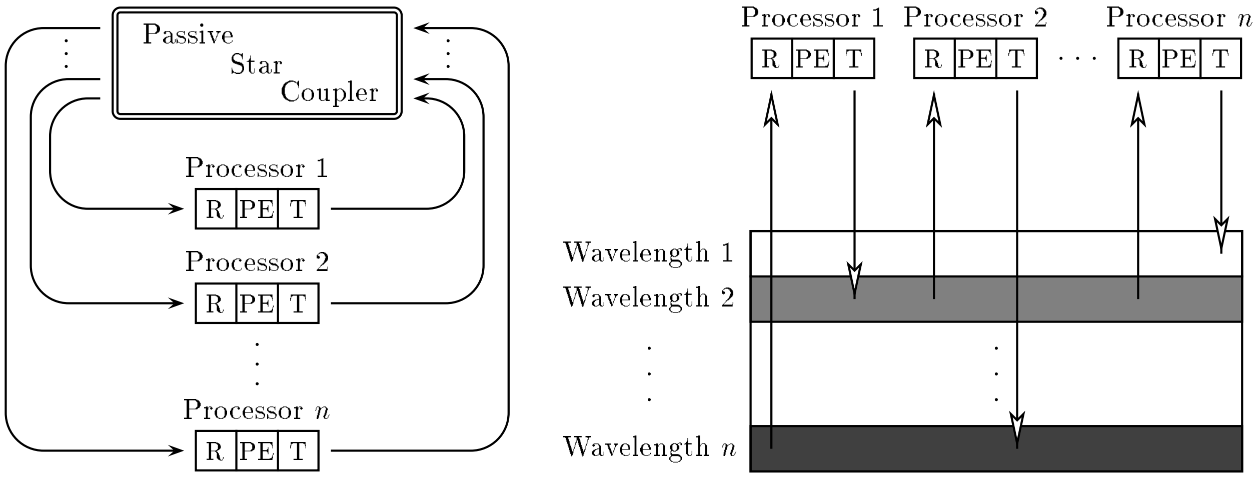

Figure 1: Left: The POS architecture. The Passive Star Coupler connects all incoming fibers to all outgoing fibers. Right: The POS computational model. All receivers tuned to a wavelength receive the message transmitted by a single transmitter tuned to this wavelength. Receivers, processing elements, and transmitters are denoted by 'R', 'PE', and 'T', respectively.

1 Introduction

The interest in optical communication systems and optical computers has recently increased dramatically in the community dealing with the theory of algorithms. Indeed, the emerging high-bandwidth applications, such as voice/video services, huge distributed data bases, and network super-computing are driving the use of optics as the communication medium for the future [OLL95]. This new technology suggests many new communication, routing, and simulation problems. Here we deal with the technology of the *passive optical star* (hereafter referred to as the POS), whose communication links consist of optical fibers capable of carrying orders of magnitude more information than what can be modulated electronically. The high bandwidth is utilized by the electronic interfaces by means of the *wavelength division multiplexing* (WDM) technique: Several transmissions may be carried by the same optical link, provided that the transmitters use different wavelengths [Bra90, Ram93]. A physical star topology is often suggested for implementing an optical network in which optical fibers are interconnected via an optical star coupler; see, e.g., [GK91, SGK91, Meh90, HKS90, WDD93, GMC+95].

Each processor in a POS-based network consists of a receiver, a transmitter, and a processing element with a memory unit (Fig. 1). The transmitters and receivers may tune to any one of a range of wavelengths. All the receivers and transmitters are connected by optical fibers to a Passive Star Coupler, which is an all-to-all communication device with broadcast capability. Thus broadcast is an elementary operation: Any message sent by a transmitter using a certain wavelength reaches all the receivers that tune to this wavelength. If several messages are transmitted simultaneously on the same wavelength, detectable "noise" is received.

The POS paradigm has been used for the design and realization of several experimental networks. The reader is referred to [Muk92a, Muk92b] for a comprehensive survey. To mention only a few that best coincide with the POS model as described above: Bellcore's LAMBDANET [GKV+90] provides an 18-wavelength system; the RAINBOW system [JRS93] has 32 wavelengths (the architects of this prototype mention a target network supporting 1000 wavelengths); and the NTT FDM-100 project

[NTIO93] consists of a 100-wavelength network.

The systems that were built vary in capabilities with respect to receiving and transmitting. Dowd [Dow91] proposed a system in which both transmitters and receivers are tunable. In contrast, only the receivers are tunable in the STAR-TRACK and RAINBOW systems [JRS93]. In the FOX system only the transmitters are tunable, and the system provides two optical stars of this type [ACG+88]. Other variations abound. For example, each node of the HYPASS system [AGKV88] is equipped with two transmitters and two receivers, only one of each being tunable.

1.1 The Abstract POS: A Parallel Computational Device

The POS technology as described above gives rise to viewing an abstract POS as a parallel computational device whose processors collaborate synchronously on some computational task. Below we briefly characterize an abstract model that shares salient features with existing POS devices and which we will also denote by the term POS.

The abstract POS model comprises a finite number of *processors* and a finite number of *wavelengths*, both indexed consecutively starting at 1. At every step, each of the processors may perform either a computation step or a communication step in which it does one or both of the following: *transmit* a message on one of the wavelengths, and *listen* to one of the wavelengths. Note that a processor may transmit on a certain wavelength and listen to a different wavelength concurrently in the same communication step. If a single processor transmits on a certain wavelength, then every processor that listens to this wavelength *receives* the transmitted message. If two or more processors transmit on the same wavelength, then each processor listening on this wavelength receives a special *collision message*, but no otherwise useful information.

Note that each step implicitly defines what we call the *communication pattern* of that step, namely the set of all ordered pairs of processors of the form (P_i, P_j), where P_i transmits on a wavelength and P_j listens to the same wavelength.

One may view the abstract POS model as a generalization of the Optical Communication Parallel Computer OCPC. This model, introduced by Anderson and Miller [AM88], has recently attracted increased attention of the theoretical community [GT92, GV94, Rao92, GJLR93, GMR94, GJM94]. Whereas the POS model is described in terms of light moving in fibers, the OCPC model is better described in terms of light beams moving in free space. Similar to the POS, an n-processor OCPC is composed of n processors, each having a transmitter and a receiver that serve as its opto-electronic network interface. Each processor may send a message to any other processor by directing a beam of light at it. However, the beam is directed to a single target; thus, in contrast to the POS, no broadcasting is possible.

Another important characteristic of the POS model is that the decisions of whether to transmit and/or receive, and using which wavelength, are taken locally by each processor depending on its local data. Thus a transmitting processor may not know which, and how many (if any), processors listen to it. This mode of operation, in which a message is sent corresponding to a certain wavelength, is very different from other models of communication (including that of the OCPC), which commonly assume that a message is always sent to a certain address. Both this feature and the ease of broadcasting influence the nature of algorithms designed for POS machines.

We call a POS *balanced* if it has as many available wavelengths as processors. POS machines that are (roughly) balanced seem to be the most interesting from both a feasibility and an algorithmic point of view, and in this paper we concentrate on such machines. It turns out that the balanced POS coincides with a restricted version of the well-known shared-memory PRAM model. The COLLISION CRCW PRAM allows concurrent reading from and writing to the same cells; whenever more than a single processor attempts to write to a given cell in a given step, a special *collision symbol* appears in the cell. Define a PRAM to be *balanced* if it has as many shared memory cells as processors. We show that the balanced POS is equivalent to the balanced COLLISION CRCW PRAM in the sense that machines of each kind with the same number of processors can simulate each other with only a constant slowdown. (The *slowdown* of a (randomized) simulation of a machine M on a machine M' is the (expected) ratio of the number of steps executed by M' to the number of steps executed by M.)

1.2 The Self-Simulation Problem

In this work we address one of the most fundamental aspects of the POS as a parallel computational model, namely the simulation of a large POS by a smaller one, called *self-simulation* (also known as *virtual parallelism* or *scalability*). The self-simulation capability of a parallel model is directly related to the efficiency and ease of algorithm design: It is desirable that the algorithm designer may assume that as many processors as required by his algorithm are simultaneously available for his program. Once the algorithm that is designed for kn processors is to be executed on a certain machine with only n processors, for $n, k \geq 1$, a simulation of the kn-processor program should be done by the actual n processors. Moreover, the set of wavelengths should also be reduced, since wavelengths are considered a scarce resource. Similarly, the communication carried out by the larger virtual machine has to be "scaled down" to the smaller machine. We assume that a POS with n processors and m wavelengths supports messages of $O(\log(n+m))$ bits and, in the context of self-simulation, that messages of the simulated machine can also be transmitted by the smaller simulating machine.

In addition to being motivated by ease of programming, the self-simulation of larger machines on smaller ones was also suggested by several researchers [MV84, Val90] as a latency-hiding technique. In this context, the size ratio of the larger to the smaller machine is called the *slackness* of the computation. The idea may be applied when the communication latency is higher than a single step, so that pipelining may be exploited in sending messages. In such cases the higher granularity of the simulating processors compensates for the higher communication latency. Similarly as other models such as the PRAM, our POS model does not incorporate latency. Nevertheless, as for the PRAM, slackness may provide the latency-hiding means for actual implementations of the model.

If an efficient self-simulation algorithm exists, then the model is said to exhibit virtual parallelism or self-simulation. For example, the well-known Brent principle (in the constructive, local sense) guarantees virtual parallelism for shared-memory models such as the PRAM. Several researchers have suggested "virtual embeddings" for the POS [BFG90, CG90, Goo89, HK88, LA90, WDD93, WD91]. Most of that work considers the emulation of a restricted set of topologies by the POS. To our knowledge, general efficient self-simulation for POS machines was not previously achieved.

In this work we consider self-simulations of a balanced kn-processor POS on a balanced n-processor POS, for $n, k \geq 1$. We focus on step-by-step simulations, said to be *normal*, in which each processor of the smaller POS simulates the operation of a fixed set of processors of the larger POS. A normal simulation *realizes* a communication pattern S of the simulated machine if, for each $(P_i, P_j) \in S$, a message is transmitted from the (simulating) processor in charge of P_i to the processor in charge of P_j. If all such messages are transmitted directly between the two processors concerned, with no intermediate stop-overs, the simulation is said to be *direct*. A simulation takes place in an *offline* setting, finally, if the communication pattern of each step of the simulated machine is revealed to the simulating machine before the simulation and available for arbitrary preprocessing.

1.3 Results

We show that the POS exhibits scalability properties. Namely, a balanced kn-processor POS can be simulated by a balanced n-processor POS in a step-by-step manner with a slowdown of $O(k+\log^* n)$ with high probability, for arbitrary integers $n, k \geq 1$. The algorithm consists essentially of a randomized solution to a distributed load-balancing problem together with a deterministic self-simulation algorithm that takes $O(k^2)$ steps in the worst case. We show that once a certain load-balancing problem has been solved by a randomized algorithm in $O(k+\log^* n)$ expected time, then the deterministic algorithm can realize the simulated communication pattern in $O(k)$ steps. Our randomized solution implies a deterministic offline algorithm that can carry out the self-simulation in $O(k)$ steps.

The rest of the paper is organized as follows. We first establish in Sec. 2 the equivalence between the balanced POS and the balanced PRAM of the same size. Mutual simulations of these models imply that they are equivalent in computational power up to constant factors. In Sec. 3 we present our main result, namely the randomized self-simulation algorithm for balanced POS machines. For reasons of convenience, we describe this self-simulation in terms of the equivalent PRAM model. Finally, in Sec. 4 we consider *direct* algorithms. We show that $\Omega(\min\{k^2, k + \log n/\log\log n\})$ is a lower bound for the slowdown of any direct algorithm, even in an offline setting, and even when the number of wavelengths of the simulating machine is unbounded. The corresponding upper bound that we exhibit is not tight: We give a direct self-simulation algorithm with a slowdown of $O(k^3)$ for balanced POS machines.

2 The Equivalence Between POS and COLLISION PRAM

As already mentioned, the balanced POS is equivalent to a variant of the PRAM with a global memory of linear size. This simple observation, expressed in the theorem below, allows a certain simplification in our description of self-simulation algorithms for the POS. We will measure time in terms of *rounds*, where a round is a constant number of elementary machine steps sufficient to carry out some specified constant-time computation.

Theorem 2.1 *For all integers n and m with $1 \leq m \leq n$, the n-processor* POS *with m wavelengths and the n-processor* COLLISION PRAM *with m global memory cells are equivalent; i.e., each machine can simulate the other one in a step-by-step fashion with a constant slowdown. In particular, the balanced n-processor* POS *and the balanced n-processor* COLLISION PRAM *are equivalent.*

PROOF Let M and M' be the POS and the PRAM of the theorem, respectively. For $i = 1, \ldots, n$, denote by P_i and P_i' the ith processors of M and M', respectively, and for $j = 1, \ldots, m$, let λ_j and C_j be the jth wavelength of M and the jth global memory cell of M', respectively.

A step of M is simulated as follows on M': For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, if P_i transmits a value on wavelength λ_j, P'_i writes the same value to C_j. If P_i listens on wavelength λ_j, P'_i, in a round following the simulated transmission, reads the value stored in C_j. Finally, P'_l stores a special *null* value in C_l, for $l = 1, \ldots, m$, in order to prevent false reads in later simulated steps.

Conversely, a step of M' is simulated as follows on M: For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, if P'_i writes a value to C_j, P_i transmits the same value on wavelength λ_j; simultaneously, P_l listens on wavelength λ_l, for $l = 1, \ldots, m$, and saves any value received. If P'_i reads the value of C_j, P_i listens on wavelength λ_j; simultaneously, P_l transmits the last value received on wavelength λ_l, for $l = 1, \ldots, m$.

The correctness of the simulations and the fact that the slowdown is constant are obvious. ∎

3 Self-simulation Algorithms

In this section we show the upper bounds for the self-simulation of the POS expressed in the following theorem.

Theorem 3.1 *For all integers $n, k \geq 1$, one step of a balanced kn-processor* POS *can be simulated on a balanced n-processor* POS

(a) in $O(\min\{k^2, k + \log n\})$ time;

(b) in $O(k + \log^* n)$ time with probability at least $1 - 2^{-n^{\epsilon}}$, for some constant $\epsilon > 0$;

(c) in $O(k)$ time in an offline setting.

We first develop two subroutines used by the simulations. When speaking about a sequence of length kp stored on a (balanced) p-processor COLLISION PRAM, we intend the first k elements of the sequence to be stored locally in the first processor, the next k elements to be stored in the second processor, etc. It will be convenient to view the sequence as stored in a $p \times k$ matrix in row-major order. By *unshuffling* the sequence we mean permuting its elements so that the original sequence can be obtained by reading the resulting matrix in column-major order instead, and *shuffling* a sequence means executing the inverse permutation. In the offline setting, any fixed permutation of kp elements can be executed in $O(k)$ time on a balanced p-processor COLLISION PRAM; this is an immediate consequence of König's Line Coloring Theorem (see [LP86, Theorem 1.4.18]), which states that an arbitrary bipartite graph with maximum degree k can be edge-colored with k colors (corresponding to k rounds). In order to avoid the nonuniformity of this argument, we provide an explicit algorithm for the two permutations of interest to us.

Lemma 3.2 *For all integers $p, k \geq 1$, a sequence of length kp can be shuffled or unshuffled in $O(k)$ time on a balanced p-processor* COLLISION PRAM.

PROOF We consider first the unshuffling operation and plan to execute it in $2k-1$ rounds numbered $1, \ldots, 2k-1$. Assume that the processors are numbered $0, \ldots, p-1$, let d and D be the greatest common divisor and the least common multiple of k and p, respectively, and consider the following algorithm: For $i = 0, \ldots, p-1$ and $j = 1, \ldots, k$, processor i sends the jth element in its row to processor $(ik + j) \bmod p$ in round number $\lfloor id/p \rfloor + j$. It is easy to see that $2k - 1$ rounds suffice, and that each element is sent to the correct processor. If we additionally show that no processor receives two or more elements in the same round, it will be clear that the algorithm can be executed in $O(k)$ time on a balanced PRAM. For $i \in I = \{0, \ldots, p-1\}$, let $f(i) = (ik - \lfloor id/p \rfloor) \bmod p$. It can be seen that it suffices to show that f is a bijection of I. Hence let $i, j \in I$ and suppose that $f(i) = f(j)$. Since $|\lfloor id/p \rfloor - \lfloor jd/p \rfloor| < d$ while $(ik - jk) \bmod p$ is a multiple of d, we must have $\lfloor id/p \rfloor = \lfloor jd/p \rfloor$ and $(i-j)k \bmod p = 0$. The first relation implies that $|i - j| < p/d$, while the second relation implies that $(i-j)k$ divides $D = kp/d$ and therefore that $i - j$ divides p/d. Thus $i = j$, and f is injective.

The shuffle permutation can be carried out by first unshuffling a set of dummy items as described above, with each processor logging the arrival times and sender identities of the items that it receives. Then the entire communication is reversed, as regards both time and direction, and applied to the real data, which is therefore permuted according to the inverse permutation. ∎

Lemma 3.3 *Let p and k be powers of 4 with $1 \le p \le k$. Then a sequence of kp elements with keys in the range $1, \ldots, k$ can be sorted in $O(k)$ time (in row-major order) on a balanced p-processor* COLLISION PRAM.

PROOF [MG88] describes a sorting algorithm for the $p \times k$ mesh that works in a constant number of stages, each of which either (1) sorts each row independently within disjoint segments, (2) rotates each row cyclically within disjoint segments, or (3) sorts each column independently within disjoint segments (the algorithm actually described in [MG88] assumes that $k \le p^2$ but, as mentioned by the authors, it can be generalized to the case $k > p^2$). Moreover, the segment boundaries and rotation distances of each stage can easily be computed sequentially in $O(k)$ time. We emulate the algorithm as follows on the balanced p-processor COLLISION PRAM: Since rows are stored entirely within a single processor, stages of type (1) can be executed in $O(k)$ time by radix-sorting pairs consisting of a segment identifier and an actual key within each row. Stages of type (2) are trivial to handle in a similar manner, and stages of type (3) can be executed by first shuffling the element matrix (Lemma 3.2), which turns each column into a contiguous segment of some row, then sorting each row within (possibly smaller) segments, and finally unshuffling the element matrix. Since each stage takes $O(k)$ time, this also holds for the entire sorting. ∎

We now prove a series of lemmas whose conjunction implies Theorem 3.1.

Lemma 3.4 *For all integers $n, k \ge 1$, one step of a balanced kn-processor* POS *can be simulated in $O(k^2)$ time on a balanced n-processor* POS.

PROOF In view of the equivalence between the balanced POS and the balanced COLLISION PRAM shown in the previous section, we may instead consider the simulation of a balanced kn-processor COLLISION PRAM M on a balanced n-processor COLLISION PRAM M'. As is natural, we divide the processors of M into n *clusters*, each containing exactly k consecutively numbered processors, and dedicate one processor of M' to each cluster. A basic component of our simulations is a *phase*, in which each processor of M' steps through the processors of M in its cluster, performing some specified action for each. It will be convenient to describe phases as though the relevant actions were carried out by the processors of M themselves; one may think of the processors in a cluster as being "animated" one by one by the processor of M' dedicated to the cluster.

Assume first that M' is not required to be balanced; instead we allow M' to use the same number of global memory cells as M. We describe a simple simulation that achieves a slowdown of $O(k)$ in this situation. Between simulated steps, the state of the global memory of M' is precisely the same as that of M. Thus it is trivial to simulate a read step of M by a single phase in which each processor of M' simply inspects the at most k global cells to be read by the processors in its cluster; recall our convention to express this by saying that each processor of M goes ahead and reads its cell. The simulation of a write step of M is similar, except for the problem that a write conflict may go undetected because what was concurrent writing in the original computation is spread out over k steps in the simulation. We counter this by means of a slightly more complicated algorithm consisting of four phases. In the first phase each processor of M attempts to store the value 0 in its *target cell*, the global cell to which it wants to write. In the second phase the processor inspects its target cell to see if it contains the collision symbol; if not, it attempts to increment its value by 1. In the third phase the processor again inspects its target cell and, if its value is not 1, stores the collision symbol in the cell and omits the fourth phase, which carries out the actual writing to the target cell of the true value to be stored there.

We now turn to the more difficult situation in which M' is required to be balanced. First note that although in actual fact M' has exactly n processors, we can assume that it has cn processors for any convenient constant $c \geq 1$. This is because cn virtual processors can be simulated with constant slowdown by the available n processors, with collision detection handled precisely as described above. We can also assume that M' has cn rather than n global memory cells, for any convenient constant $c \geq 1$. To see this, imagine a simulated global memory of size cn as laid out in the shape of a matrix with n rows and c columns and designate n processors (from among our cn processors) as *memory managers*. Each memory manager is in charge of one row of the memory matrix. What this means is that the processor will store the values of the (simulated) global cells belonging to the row in its local memory and cyclically "expose" them one after the other in one cell of (true) global memory. Thus the n memory managers together cyclically expose the columns of the memory matrix one by one. At the same time n "real" processors carry out the actual computation, reading from and writing to (simulated) global memory cells in the time slots during which they are exposed. Since a complete "memory cycle" takes $O(c)$ time and c is a constant, the slowdown is constant.

The scheme just described, with c equal to k, can be used to simulate an n-processor COLLISION PRAM with kn global memory cells on an n-processor COLLISION PRAM with n global memory cells with a slowdown of $O(k)$. Recall, however, that our task is to reduce *both* the number of processors and the number of global memory cells by a factor of k. A simple way to do this is to slow down the "memory cycle" so that each cell is exposed for $\Theta(k)$ steps, during which an entire phase can be executed. This provides the desired simulation with a slowdown of $O(k^2)$. ∎

Lemma 3.5 *For all integers $n, k \geq 1$, one step of a balanced kn-processor* POS *can be simulated in $O(k + \log n)$ time on a balanced n-processor* POS.

PROOF We refine the basic simulation described in the previous proof. Recall that we view the global memory of the simulated machine as an $n \times k$ matrix. A central notion for our algorithms is the column number of a simulated global memory cell, which will be called its *color*. All memory cells of a particular color are exposed simultaneously, and in each simulated step the memory managers step through the colors in the order $1, \ldots, k$. Each of the n simulating processors in charge of a cluster (call these *cluster processors*) wants to access up to k memory cells of various colors. We will assume that each cluster processor creates a *token* for each such memory request. Each token adopts the color of the relevant memory cell and lists its address, the value to be stored there in the case of a write request, and the identity of the creator of the token in the case of a read request.

Before each simulated memory access, each cluster processor sorts its tokens by color using bucket sorting. Since colors are integers in the range $1, \ldots, k$, this takes $O(k)$ time. The rest of the simulated step is one "memory cycle", in which each cluster processor processes all its tokens of a particular color during the time slot in which the memory cells of that color are exposed. In contrast with the algorithm with slowdown $O(k^2)$, the time slot of a particular color is not chosen in a worst-case manner (namely as $\Theta(k)$), but instead according to the actual "demand" for that color, namely the maximum number of tokens of that color present in any single cluster processor. This can be implemented using concurrent writing to a single global cell after each round of memory accesses to determine whether some cluster processor is still busy with tokens of the current color; when this no longer the case, the memory managers proceed to the next color.

It is not difficult to see that even the modified simulation may still have a slowdown of $\Theta(k^2)$. This happens, e.g., if some cluster processor has k tokens of color 1, another cluster processor has k tokens of color 2, etc. The key to improving the efficiency is to redistribute the tokens such that the tokens of each color are approximately evenly distributed among the cluster processors; we call this *balancing* the tokens. In particular, we will say that the tokens are 1-*balanced* if, for each color, the numbers of tokens of that color in any two cluster processors differ by at most 1. If the tokens are 1-balanced, the modified simulation algorithm will finish in $O(k)$ time, since no cluster processor will ever have to wait more than constant time for the next color to be exposed. What remains is to show how to 1-balance the tokens.

We distinguish between the cases $n \le k$ and $n > k$. Assume first that $n \le k$. Since increasing k simply corresponds to simulating a bigger machine, while we have already seen that n can be increased by any constant factor, we can assume without loss of generality that n (in the role of p) and k are of the form required by Lemma 3.3. We process the tokens by first sorting them by color (in row-major order) (Lemma 3.3) and subsequently unshuffling the sorted sequence of tokens (Lemma 3.2). This takes $O(k)$ time, and it is not difficult to see that the resulting distribution of tokens is 1-balanced; this connection between sorting and balancing has been observed often before. We then finish the simulation of the step under consideration by carrying out the relevant memory accesses. In the case of a simulated write step, this is easy, since the action associated with a token (namely storing a certain value in a global memory cell) can be carried out by any processor. In the case of a simulated read step, the action associated with a token marks the token with the value read, after which the token must be returned to its creator by reversing the communication of all steps that moved tokens around, similarly as described at the end of the proof of Lemma 3.2. All of this takes $O(k)$ steps.

Assume now instead that $n > k$, so that the algorithm of Lemma 3.3 cannot be used directly. We would like to appeal to the existence of logarithmic-time sorting algorithms for the PRAM [AKS83, Col88]. These algorithms, however, need as many processors and shared memory cells (up to a constant factor) as there are elements to be sorted. In our context this means that we cannot apply the algorithms to the full set of up to kn tokens, but only to a collection of $O(n)$ objects.

We solve this problem by first combining tokens of the same color into *supertokens* in such a way as to create only $O(n)$ supertokens, then sorting supertokens rather than simple tokens, and finally again replacing each supertoken by the set of simple tokens that it represents. The details follow.

Without loss of generality assume that k divides n and partition the cluster processors into n/k *groups* of k consecutively numbered processors each. Begin by using the algorithm of Lemma 3.3 to sort the tokens within each group (i.e., in processors in the group) into row-major order; this takes $O(k)$ time. Subsequently each cluster processor, for each color present in its row, creates a supertoken marked with the relevant color and the identity of its creator. Since each sorted sequence of tokens contains at most $k-1$ "color changes", while a processor with m "color changes" in its row generates $m+1$ supertokens, the processors within each group generate a total of at most $2k-1$ supertokens. Thus the total number of supertokens is bounded by $2n$, and we can store them in an array of $2n$ cells (with some cells left empty) in $O(k)$ time and sort them by color in $O(\log n)$ time using one of the algorithms of [AKS83, Col88]. Subsequently the resulting sorted sequence of supertokens is stored in constant time with at most two supertokens per processor. Then each processor P, for each supertoken that it holds, sends a message to the creator P' of the supertoken. By using the color of the supertoken to designate a time slot to use for the communication, all messages of this kind can be exchanged in $O(k)$ time. Subsequently the up to k (simple) tokens represented by the supertoken are transmitted from P' to P, which again can be done in $O(k)$ overall time. This stores the tokens in a correctly sorted sequence of length $O(kn)$, except that there may be "holes" between consecutive tokens. All that remains is to unshuffle this sequence (Lemma 3.2) and proceed with the memory accesses as in the case $n \le k$, simply skipping the holes. ∎

Lemma 3.6 *There is a constant $\epsilon > 0$ such that for all integers $n, k \ge 1$, one step of a balanced kn-processor* POS *can be simulated in $O(k + \log^* n)$ time on a balanced n-processor* POS *with probability at least $1 - 2^{-n^\epsilon}$.*

PROOF Observe that the sorting operation used in the proof of Lemma 3.5 is actually stronger than what we need: It suffices to arrange the tokens into a sequence in which the tokens of each color appear in a contiguous subsequence, an operation known as *semisorting*. We can again allow the semisorting to create holes between consecutive tokens, provided that the total number of holes is bounded by a constant factor times the number of tokens, in which case the operation applied to the sequence of tokens is called *padded semisorting*. Padded semisorting has very fast randomized parallel algorithms. In particular, Lemma 3.6 is implied by a result of Bast and Hagerup [BH95, Thm. 9.21], which states that for some constant $\epsilon > 0$, a sequence of n elements with colors in the range $1, \ldots, n$ can be padded-sorted in $O(\log^* n)$ time on a balanced n-processor COLLISION PRAM with probability at least $1 - 2^{-n^\epsilon}$. ∎

Lemma 3.7 *For all integers $n, k \ge 1$, one step of a balanced kn-processor* POS *can be simulated in $O(k)$ time on a balanced n-processor* POS*, provided that the communication pattern is known in advance and available for precomputation.*

PROOF Simply observe that the instances of sorting problems encountered in the algorithm of Lemma 3.5 depend only on the communication pattern of the simulated step and not on the actual values transmitted. Thus the permutation sorting each instance can be computed offline and later applied in constant time, while all other operations take only $O(k)$ time. ∎

4 Direct Algorithms

A step-by-step simulation of a POS M on another POS M' is called *normal* if there is a fixed mapping π from the set of processors of M to the set of processors of M' such that for each message transmitted between two processors P_1 and P_2 of M in a particular step of its computation, a message is transmitted between $\pi(P_1)$ and $\pi(P_2)$ in the simulation of that step on M' (informally, each processor of M' is in charge of the processors of M mapped to it under π). If, additionally, the message is always transmitted directly between $\pi(P_1)$ and $\pi(P_2)$ without intermediate stopovers at other processors (in particular, $\pi(P_1)$ and $\pi(P_2)$ must then be tuned to the same wavelength at some point during the simulated step), the simulation is *direct*. While all of the algorithms described in the previous section are normal, none of them is direct; this is because of the indirectness introduced by the simulations between the POS and the PRAM. Nonetheless, we can easily devise direct simulations of a kn-processor POS M on an n-processor POS M' based on the idea of the simple deterministic simulation with slowdown $O(k^2)$ (Lemma 3.4).

If M' is allowed to use the same wavelengths as M, we achieve $O(k^2)$ slowdown even with a direct algorithm. This is done by simulating a step in k stages, each of which is subdivided into k rounds. Each processor of M (more precisely, the processor of M' in charge of it) activates its transmitter in each round of a single stage and its receiver in a single round of each stage. Thus each processor of M is certain to receive either a collision indication or all messages transmitted on the wavelength to which it listens; in the latter case, it can clearly determine whether there is a collision on that wavelength in the original computation.

If M' is required to be balanced, we can partition the kn wavelengths of M into k groups of n wavelengths each and modify the simulation given above to treat the groups one by one, mapping the wavelengths of each group bijectively to the n wavelengths of M'. This contributes an extra factor of k to the slowdown; i.e., we obtain a direct simulation of a balanced kn-processor POS on a balanced n-processor POS with a slowdown of $O(k^3)$.

We now prove a lower bound for direct self-simulations. The bound applies even to the offline setting and holds regardless of whether or not we allow simultaneous transmit and receive operations. We assume that the simulation is normal, but do not require all simulating processors to be in charge of the same number of simulated processors.

Theorem 4.1 *For all integers $n \geq 4$ and $k \geq 1$, every normal and direct simulation of a balanced kn-processor* POS *M on a (not necessarily balanced) n-processor* POS *M' has a slowdown of $\Omega(\min\{k^2, k + \log n/\log\log n\})$.*

PROOF We will construct a single step of M whose simulation on M' needs $\Omega(\min\{k^2, k+\log n/\log\log n\})$ time. Let π be the function mapping each processor of M to the processor of M' in charge of it and, as in the simulations in Section 3, define a *cluster* to be a maximal set of processors of M with the same image under π. Since the maximal size of a cluster, which is at least k, is an obvious lower bound on the slowdown, we can assume without loss of generality that $2e \leq k \leq \log n/\log\log n$ and that no cluster contains more than k^2 processors.

Let C be the set of clusters containing at least k processors. Then

$$|C| \cdot k^2 + n(k-1) \geq kn$$

and hence $|C| \geq n/k^2$, since $|C|$ clusters contain at most k^2 processors each, the remaining at most n clusters contain at most $k-1$ processors each, and the total number of processors in all clusters is exactly kn.

Let m be the largest integer with $2(em)^k \leq |C|$ and consider the communication pattern characterized as follows: km processors of M, distributed over m clusters of C with exactly k such processors in each cluster, transmit distinct messages on km different wavelengths. In the same step and for each set A of exactly k among the km transmitting processors, there is a cluster in C whose processors listen on the wavelengths used by the processors in A. Moreover, no cluster contains both a transmitting and a receiving processor; this ensures that each message must in fact be communicated between two distinct simulating processors. Since there are $\binom{km}{k} \leq (em)^k$ different sets A, the relation $2(em)^k \leq |C|$ shows that such a communication pattern is indeed possible.

Assume that the simulation of the step defined by the communication pattern above is completed in exactly $t < k^2$ steps; obviously, $t \geq k$. Each transmitting cluster sends k different messages, so each transmitting cluster contains a processor whose message is transmitted in at most $\lfloor t/k \rfloor \leq k-1$ steps; we will call such a processor *unlucky*. Define a *slot* as any subset of exactly $\lfloor t/k \rfloor$ of the t steps of the simulation. We will say that a slot S is *hit* by an unlucky processor P if all transmissions of the message of P take place during steps in S. Each of the at least m unlucky processors hits at least one of the

$$\binom{t}{\lfloor t/k \rfloor} \leq \left(\frac{et}{\lfloor t/k \rfloor}\right)^{\lfloor t/k \rfloor} \leq (2ek)^{t/k} \leq k^{2t/k}$$

slots, so some slot is hit by at least $m/k^{2t/k}$ unlucky processors. On the other hand, no k unlucky processors hit the same slot, since some receiving cluster is supposed to receive all k messages of such k unlucky processors, for which one slot is not sufficient. Thus $m \leq k \cdot k^{2t/k} \leq k^{3t/k}$. On the other hand, the inequalities $k \leq \log n / \log\log n$ and $2(e(m+1))^k > |C| \geq n/k^2$ show that for n larger than some constant, we will have $m^{2k} \geq n$. Combining the two bounds for m yields $k^{6t} \geq m^{2k} \geq n$, which proves that $t = \Omega(\log n / \log k) = \Omega(\log n / \log\log n)$. ∎

For sufficiently large values of n relative to k such as $n = 2^{k^3}$, Theorem 4.1 yields a lower bound of $\Omega(k^2)$. Coupled with the upper bound developed above, this shows the exact complexity of direct self-simulation without any balance requirement to be $\Theta(k^2)$ for sufficiently large values of n. We do not have any lower bound corresponding to the upper bound of $O(k^3)$ for self-simulation on balanced machines.

5 Conclusions

We presented a randomized indirect self-simulation algorithm with a slowdown of $O(k + \log^* n)$, showing that the balanced POS and the balanced COLLISION CRCW PRAM are essentially scalable. We also analyzed the offline case and direct and deterministic algorithms.

One may question our focus on balanced POS machines. Since our randomized self-simulation already achieves an expected slowdown of $O(k + \log^* n)$, while $\Omega(k)$ is a lower bound for any simulation of a kn-processor machine on an n-processor machine, it makes little sense to make the simulating machine more powerful or the simulated machine less powerful by changing the number of wavelengths. If we go in the opposite direction by allowing the simulating machine only $m < n$ wavelengths, we can obtain a simulation of a balanced kn-processor POS with expected slowdown $O(kn/m + \log^* m)$ simply by ignoring $n - m$ simulating processors. On the other hand, a consideration of the maximum number of messages that can be transmitted in one step on the two machines shows $\Omega(kn/m)$ to be a lower bound on the slowdown, so that again we are within an additive $O(\log^* n)$ of the optimum. If instead we allow the simulated kn-processor machine to have an arbitrary number of wavelengths, rather than just kn, an efficient randomized self-simulation can be obtained by first mapping the wavelengths used in a particular step injectively to an integer range of size $O(kn)$ by means of known hashing techniques and then applying the algorithm of Theorem 3.1(b).

Certain technological constraints, while not taken into account in this paper, are of considerable practical relevance and may be addressed in future work. These include:

Broadcast: Limitations on the broadcast capabilities of the simulating POS.

Wavelengths: Limitations on the number of wavelengths available, especially for broadcasts to large numbers of processors.

Tuning: The tuning time may be considerably longer than the transmission time. During tuning a transmitter or receiver is not operational.

Other models: Some systems would supply powerful processors with several independent transmitters and/or receivers. Sometimes tuning is restricted to (some of the) receivers and/or transmitters.

References

[ACG+88] E. Arthurs, J.M. Cooper, M.S. Goodman, H. Kobrinski, M. Tur, and M.P. Vecchi. Multiwavelength optical crossconnect for parallel processing computers. *Electronic Letters*, 24:119–120, 1988.

[AGKV88] E. Arthurs, M.S. Goodman, H. Kobrinski, and M.P. Vecchi. HYPASS: An optoelectronic hybrid packet-switching system. *IEEE Journal on Selected Areas in Communications*, 6:1500–1510, 1988.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[AM88] 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, LA, 1988.

[BFG90] J. A. Bannister, L. Fratta, and M. Gerla. Topological design of the wavelength-division optical network. In *IEEE Infocom*, volume 3, pages 1005–1013, 1990.

[BH95] H. Bast and T. Hagerup. Fast parallel space allocation, estimation and integer sorting. *Information and Computation*, 123(1):72–110, November 1995.

[Bra90] C. Brackett. Dense wavelength division multiplexing networks: Principles and applications. *IEEE Journal on Selected Areas in Communications*, 8:373–380, August 1990.

[CG90] I. Chlamtac and A. Ganz. Toward alternative high-speed network concepts: The SWIFT architecture. *IEEE Transactions on Communications*, 38(4):431–439, April 1990.

[Col88] R. Cole. Parallel merge sort. *SIAM Journal of Computing*, 17(4):770–785, August 1988.

[Dow91] P.W. Dowd. Random access protocols for high-speed interprocessor communication based on an optical passive star topology. *IEEE/OSA Journal of Lightwave Technology*, 9(6):799–808, June 1991.

[GJLR93] L.A. Goldberg, M. Jerrum, T. Leighton, and S. Rao. A doubly logarithmic communication algorithm for the completely connected optical communication parallel computer. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 300–309, Velen, Germany, June 1993. SIGACT and SIGARCH.

[GJM94] L.A. Goldberg, M. Jerrum, and P.D. MacKenzie. An $\Omega(\sqrt{\log\log n})$ lower bound for routing in optical networks. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 147–156, Cape May, NJ, USA, June 1994. SIGACT and SIGARCH, ACM Press.

[GK91] A. Ganz and Z. Koren. WDM passive star - protocols and performance analysis. In *IEEE Infocom*, pages 991–1000, 1991.

[GKV+90] M. S. Goodman, H. Kobrinski, M. Vecchi, R. M. Bulley, and J. M. Gimlett. The LAMBDANET multiwavelength network: Architecture, applications and demonstrations. *IEEE Journal on Selected Areas in Communications*, 8(6):995–1004, August 1990.

[GMC+95] G. Gravenstreter, R.G. Melhem, D.M. Chiarulli, S.P. Levitan, and J.P. Teza. The partitioned optical passive stars (POPS) topology. In *IEEE International Parallel Processing Symposium*, pages 4–10, 1995.

[GMR94] L. A. Goldberg, Y. Matias, and S. Rao. An optical shared memory simulation. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 257–267, Cape May, NJ, USA, June 1994. SIGACT and SIGARCH, ACM Press.

[Goo89] M. S. Goodman. Multiwavelength networks and new approaches to packet switching. *IEEE Communication Magazine*, 27(10):27–35, October 1989.

[GT92] M. Geréb-Grauss and T. Tsantilas. Efficient optical communication in parallel computers. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 41–48, June 1992.

[GV94] Gerbessiotis and Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22(2):251–267, 1994.

[HK88] M.G. Hluchyj and M.J. Karol. ShuffleNet: An application of generalized perfect shuffles to multihop lightwave networks. *IEEE Transactions on Communications*, 37:379–390, 1988.

[HKS90] I.M.I. Habbab, M. Kavehrad, and C.E.W. Sundberg. Protocol for very high speed optical fiber local area networks using a passive star topology. *IEEE/OSA Journal of Lightwave Technology*, 15(12), December 1990.

[JRS93] F. J. Janniello, R. Ramaswami, and D. G. Steinberg. A prototype circuit-switched multi-wavelength optical metropolitan-area network. *IEEE/OSA Journal of Lightwave Technology*, 11:777–782, May/June 1993.

[LA90] J.-F. Labourdette and A.S. Acampora. Wavelength agility in multihop lightwave networks. In *IEEE Infocom*, volume 3, pages 1022–1029, June 1990.

[LP86] L. Lovász and M.D. Plummer. *Matching theory.* North Holland, Amsterdam, 1986.

[Meh90] N. Mehravari. Performance and protocol improvements for very high speed optical fiber local area networks using a passive star topology. *IEEE/OSA Journal of Lightwave Technology*, 8(4), April 1990.

[MG88] J.M. Marberg and E. Gafni. Sorting in constant number of row and column phases on a mesh. *Algorithmica*, 3(4):561–572, 1988.

[Muk92a] B. Mukherjee. WDM-based local lightwave networks Part I: Single-hop systems. *IEEE Networks*, pages 12–27, May 1992.

[Muk92b] B. Mukherjee. WDM-based local lightwave networks Part II: Multi-hop systems. *IEEE Networks*, pages 20–32, July 1992.

[MV84] 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.

[NTIO93] K. Nosu, H. Toba, K. Inoue, and K. Oda. 100 channel optical FDM technology and its applications to optical FDM channel-based networks. *IEEE/OSA Journal of Lightwave Technology*, 11:764–776, May/June 1993.

[OLL95] N. Otha, H.U. Lemke, and J.-C. Lehureau, editors. *Advanced Image and Video Communications and Storage Technologies*, volume 2451 of *SPIE*, Amsterdam, The Netherlands, March 1995. SPIE. EurOpto Series.

[Ram93] R. Ramaswami. Multi-wavelength lightwave networks for computer communication. *IEEE Communication Magazine*, 31:78–88, 1993.

[Rao92] S.B. Rao. Properties of an interconnection architecture based on wavelength division multiplexing. Technical Report TR-92-009-3-0054-2, NEC Research Institute, January 1992.

[SGK91] G.N.M. Sudhakar, N.D. Georganas, and M. Kavehrad. Slotted aloha and reservation aloha protocols for very high-speed optical fiber local area networks using passive star topology. *IEEE/OSA Journal of Lightwave Technology*, 9(10):1411–1422, October 1991.

[Val90] L.G. Valiant. A bridging model for parallel computation. *Communication of the ACM*, 33(8):103–111, 1990.

[WD91] K. Williams and D. Du. Efficient embedding of a hypercube in an irregular WDM network. Technical Report TR 91-16, U. of Minnesota, May 1991.

[WDD93] K. A. Williams, T. Q. Dam, and D. H. C. Du. A media-access protocol for time- and wavelength-division multiplexed passive star networks. *IEEE Journal on Selected Areas in Communications*, 11(4):560–567, 1993.