Counting Networks

James Aspnes [1] Maurice Herlihy
Nir Shavit [2]

Digital Equipment Corporation
Cambridge Research Lab

CRL 93/11 August 6, 1993

[1]IBM Almaden Research Center. A large part of this work was performed while the author was at Carnegie-Mellon University.

[2]MIT Laboratory for Computer Science. Author's current address: Computer Science Department, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel. This work was supported by ONR contract N00014-91-J-1046, NSF grant CCR-8915206, DARPA contract N00014-89-J-1988, and by a Rothschild postdoctoral fellowship. A large part of this work was performed while the author was at IBM's Almaden Research Center.

Abstract

Many fundamental multi-processor coordination problems can be expressed as *counting problems:* processes must cooperate to assign successive values from a given range, such as addresses in memory or destinations on an interconnection network. Conventional solutions to these problems perform poorly because of synchronization bottlenecks and high memory contention.

Motivated by observations on the behavior of sorting networks, we offer a new approach to solving such problems, by introducing *counting networks*, a new class of networks that can be used to count.

We give two counting network constructions, one of depth $\log n(1 + \log n)/2$ using $n \log n(1+\log n)/4$ "gates," and a second of depth $\log^2 n$ using $n \log^2 n/2$ gates. These networks avoid the sequential bottlenecks inherent to earlier solutions, and substantially lower the memory contention.

Finally, to show that counting networks are not merely mathematical creatures, we provide experimental evidence that they outperform conventional synchronization techniques under a variety of circumstances.

This report supersedes CRL Tech Report 90/11. A preliminary version of this work appeared in the *Proceedings of the 23rd ACM Symposium on the Theory of Computing, New Orleans, May 1991.*

Keywords: Counting Networks, Parallel Processing, Hot-Spots, Network Routing.

1 Introduction

Many fundamental multi-processor coordination problems can be expressed as *counting problems:* processors collectively assign successive values from a given range, such as addresses in memory or destinations on an interconnection network. In this paper, we offer a new approach to solving such problems, by introducing *counting networks*, a new class of networks that can be used to count.

Counting networks, like sorting networks [4, 7, 8], are constructed from simple two-input two-output computing elements called *balancers*, connected to one another by wires. However, while an n input sorting network sorts a collection of n input values only if they arrive together, on separate wires, and propagate through the network in lockstep, a counting network can count any number $N \gg n$ of input tokens even if they arrive at arbitrary times, are distributed unevenly among the input wires, and propagate through the network asynchronously.

Figure 2 provides an example of an execution of a 4-input, 4-output, counting network. A balancer is represented by two dots and a vertical line (see Figure 1). Intuitively, a balancer is just a toggle mechanism [1], alternately forwarding inputs to its top and bottom output wires. It thus balances the number of tokens on its output wires. In the example of Figure 2, input tokens arrive on the network's input wires one after the other. For convenience we have numbered them by the order of their arrival (these numbers are *not* used by the network). As can be seen, the first input (numbered 1) enters on line 2 and leaves on line 1, the second leaves on line 2, and in general, the Nth token will leave on line $N \bmod 4$. (The reader is encouraged to try this for him/herself.) Thus, if on the ith output line the network assigns to consecutive outputs the numbers $i, i+4, i+2\cdot 4, ..$, it is *counting* the number of input tokens without ever passing them all through a shared computing element!

Counting networks achieve a high level of throughput by decomposing interactions among processes into pieces that can be performed in parallel. This decomposition has two performance benefits: It eliminates serial bottlenecks and reduces memory contention. In practice, the performance of many

[1]One can implement a balancer using a read-modify-write operation such as *Compare & Swap*, or a short critical section.

shared-memory algorithms is often limited by conflicts at certain widely-shared memory locations, often called *hot spots* [30]. Reducing hot-spot conflicts has been the focus of hardware architecture design [15, 16, 22, 29] and experimental work in software [5, 13, 14, 25, 27].

Counting networks are also *non-blocking:* processes that undergo halting failures or delays while using a counting network do not prevent other processes from making progress. This property is important because existing shared-memory architectures are themselves inherently asynchronous; process step times are subject to timing uncertainties due to variations in instruction complexity, page faults, cache misses, and operating system activities such as preemption or swapping.

Section 2 defines counting networks. In Sections 3 and 4, we give two distinct counting network constructions, each of depth less than or equal to $\log^2 n$, each using less than or equal to $(n \log^2 n)/2$ balancers. To illustrate that counting networks are useful we use counting networks to construct high-throughput shared-memory implementations of concurrent data structures such as shared counters, producer/consumer buffers, and barriers. A *shared counter* is simply an object that issues the numbers 0 to $m-1$ in response to m requests by processes. Shared counters are central to a number of shared-memory synchronization algorithms (e.g., [10, 12, 16, 31]). A *producer/consumer buffer* is a data structure in which items inserted by a pool of producer processes are removed by a pool of consumer processes. A *barrier* is a data structure that ensures that no process advances beyond a particular point in a computation until all processes have arrived at that point. Compared to conventional techniques such as spin locks or semaphores, our counting network implementations provide higher throughput, less memory contention, and better tolerance for failures and delays. The implementations can be found in Section 5.

Our analysis of the counting network construction is supported by experiment. In Section 6, we compare the performance of several implementations of shared counters, producer/consumer buffers, and barrier synchronization on a shared-memory multiprocessor. When the level of concurrency is sufficiently high, the counting network implementations outperform conventional implementations based on spin locks, sometimes dramatically. Finally, Section 7 describes how to mathematically verify that a given network counts.

In summary, counting networks represent a new class of concurrent algorithms. They have a rich mathematical structure, they provide effective

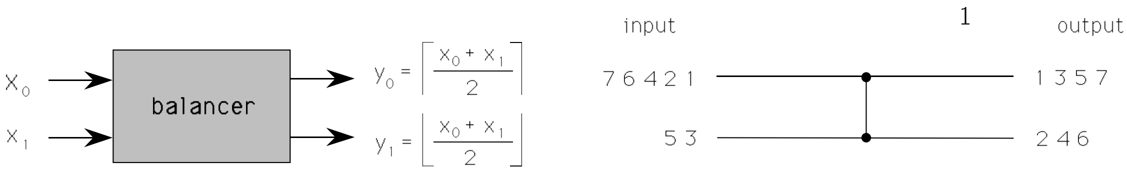


Figure 1: A Balancer.

solutions to important problems, and they perform well in practice. We believe that counting networks have other potential uses, for example as interconnection networks [32] or as load balancers[28], and that they deserve further attention.

2 Networks That Count

2.1 Counting Networks

Counting networks belong to a larger class of networks called balancing networks, constructed from wires and computing elements called balancers, in a manner similar to the way in which comparison networks [8] are constructed from wires and comparators. We begin by describing balancing networks.

A *balancer* is a computing element with two input wires and two output wires[2] (see Figure 1). Tokens arrive on the balancer's input wires at arbitrary times, and are output on its output wires. Intuitively, one may think of a balancer as a toggle mechanism, that given a stream of input tokens, repeatedly sends one token to the top output wire and one to the bottom, effectively balancing the number of tokens that have been output on its output wires. We denote by x_i, $i \in \{0,1\}$ the number of input tokens ever received on the balancer's ith input wire, and similarly by y_i, $i \in \{0,1\}$ the number of tokens ever output on its ith output wire. Throughout the paper we will abuse this notation and use x_i (y_i) both as the name of the ith input (output) wire and a count of the number of input tokens received on the wire.

Let the state of a balancer at a given time be defined as the collection of tokens on its input and output wires. For the sake of clarity we will assume

[2]In Figure 1 as well as in the sequel, we adopt the notation of [8] and and draw wires as horizontal lines with balancers stretched vertically.

that tokens are all distinct. We denote by the pair (t, b), the state *transition* in which the token t passes from an input wire to an output wire of the balancer b.

We can now formally state the safety and liveness properties of a balancer:

1. In any state $x_0 + x_1 \geq y_0 + y_1$ (i.e. a balancer never creates output tokens).

2. Given any finite number of input tokens $m = x_0 + x_1$ to the balancer, it is guaranteed that within a finite amount of time, it will reach a *quiescent* state, that is, one in which the sets of input and output tokens are the same. In any quiescent state, $x_0 + x_1 = y_0 + y_1 = m$.

3. In any quiescent state, $y_0 = \lceil m/2 \rceil$ and $y_1 = \lfloor m/2 \rfloor$.

A *balancing network* of width w is a collection of balancers, where output wires are connected to input wires, having w designated input wires $x_0, x_1, .., x_{w-1}$ (which are not connected to output wires of balancers), w designated output wires $y_0, y_1, .., y_{w-1}$ (similarly unconnected), and containing no cycles. Let the state of a network at a given time be defined as the union of the states of all its component balancers. The safety and liveness of the network follow naturally from the above network definition and the properties of balancers, namely, that it is always the case that $\sum_{i=0}^{w-1} x_i \geq \sum_{i=0}^{w-1} y_i$, and for any finite sequence of m input tokens, within finite time the network reaches a *quiescent* state, i.e. one in which $\sum_{i=0}^{w-1} y_i = m$.

It is important to note that we make no assumptions about the timing of token transitions from balancer to balancer in the network — the network's behavior is completely asynchronous. Although balancer transitions can occur concurrently, it is convenient to model them using an interleaving semantics in the style of Lynch and Tuttle [24]. An *execution* of a network is a finite sequence $s_0, e_1, s_1, \ldots e_n, s_n$ or infinite sequence $s_0, e_1, s_1, \ldots$ of alternating states and balancer transitions such that for each (s_i, e_{i+1}, s_{i+1}), the transition e_{i+1} carries state s_i to s_{i+1}. A *schedule* is the subsequence of transitions occurring in an execution. A schedule is *valid* if it is induced by some execution, and *complete* if it is induced by an execution which results in a quiescent state. A schedule s is *sequential* if for any two transitions $e_i = (t_i, b_i)$ and $e_j = (t_j, b_j)$, where t_i and t_j are the same token, then all transitions between them also involve that token.

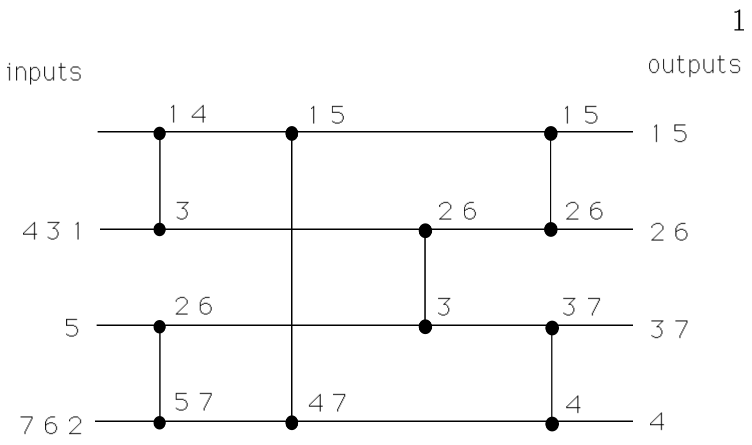


Figure 2: A sequential execution for a BITONIC[4] counting network.

On a shared memory multiprocessor, a balancing network is implemented as a shared data structure, where balancers are records, and wires are pointers from one record to another. Each of the machine's asynchronous processors runs a program that repeatedly traverses the data structure from some input pointer (either preassigned or chosen at random) to some output pointer, each time shepherding a new token through the network (see section 5).

We define the *depth* of a balancing network to be the maximal depth of any wire, where the depth of a wire is defined as 0 for a network input wire, and

$$\max(\mathit{depth}(x_0), \mathit{depth}(x_1)) + 1$$

for the output wires of a balancer having input wires x_0 and x_1. We can thus formulate the following straightforward yet useful lemma:

Lemma 2.1 *If the transition of a token from the input to the output by any balancer (including the time spent traversing the input wire) takes at most Δ time, then any input token will exit the network within time at most Δ times the network depth.*

A *counting network* of width w is a balancing network whose outputs $y_0, .., y_{w-1}$ satisfy the following *step property*:

In any quiescent state, $0 \leq y_i - y_j \leq 1$ for any $i < j$.

To illustrate this property, consider an execution in which tokens traverse the network sequentially, one completely after the other. Figure 2 shows such an execution on a BITONIC[4] counting network which we will define formally in Section 3. As can be seen, the network moves input tokens to output wires in increasing order modulo w. Balancing networks having this property are called counting networks because they can easily be adapted to count the total number of tokens that have entered the network. Counting is done by adding a "local counter" to each output wire i, so that tokens coming out of that wire are consecutively assigned numbers $i, i+w, \ldots, i+(y_i-1)w$. (This application is described in greater detail in Section 5.)

The step property can be defined in a number of ways which we will use interchangeably. The connection between them is stated in the following lemma:

Lemma 2.2 *If $y_0, \ldots, y_{w-1}$ is a sequence of non-negative integers, the following statements are all equivalent:*

1. *For any $i < j$, $0 \leq y_i - y_j \leq 1$.*
2. *Either $y_i = y_j$ for all i, j, or there exists some c such that for any $i < c$ and $j \geq c$, $y_i - y_j = 1$.*
3. *If $m = \sum_{i=0}^{w-1} y_i$, $y_i = \left\lceil \frac{m-i}{w} \right\rceil$.*

It is the third form of the step property that makes counting networks usable for counting.

Proof: We will prove that 3 implies 1, 1 implies 2, and 2 implies 3.

For any indexes $a < b$, since $0 < a < b < w$, it must be that $0 \leq \left\lceil \frac{m-a}{w} \right\rceil - \left\lceil \frac{m-b}{w} \right\rceil \leq 1$. Thus 3 implies 1.

Assume 1 holds for the sequence $y_0, \ldots, y_{w-1}$. If for every $0 \leq i < j \leq w$, $y_i - y_j = 0$, then 2 follows. Otherwise, there exists the largest a such that there is a b for which $a < b$ and $y_a - y_b = 1$. From a's being largest we get that $y_a - y_{a+1} = 1$, and from 1 we get $y_i = y_a$ for any $0 \leq i \leq a$ and $y_i = y_{a+1}$ for any $a+1 \leq i \leq w$. Choosing $c = a+1$ completes the proof. Thus 1 implies 2.

Assume by way of contradiction that 3 does not hold and 2 does. Without loss of generality, there thus exists the smallest a such that $m = \sum_{i=0}^{w-1} y_i$ and

$y_a \neq \left\lceil \frac{m-a}{w} \right\rceil$. If $y_a < \left\lceil \frac{m-a}{w} \right\rceil$, then since $\sum_{i=0}^{k-1} y_i = m$, by simple arithmetic there must exist a $b > a$ such that $y_b > \left\lceil \frac{m-b}{w} \right\rceil$. Since $0 \leq \left\lceil \frac{m-a}{w} \right\rceil - \left\lceil \frac{m-b}{w} \right\rceil \leq 1$, $y_b - y_a \geq 1$, and no c as in 2 exists, a contradiction. Similarly, if $y_a > \left\lceil \frac{m-a}{w} \right\rceil$, there exists a $b \neq a$ such that $y_b < \left\lceil \frac{m-b}{w} \right\rceil$, and $y_a - y_b \geq 2$. Again no c as in 2 exists, a contradiction. Thus 2 implies 3.

■

The requirement that a quiescent counting network's outputs have the step property might appear to tell us little about the behavior of a counting network during an asynchronous execution, but in fact it is surprisingly powerful. Even in a state in which many tokens are passing through the network, the network must eventually settle into a quiescent state if no new tokens enter the network. This constraint makes it possible to prove such important properties as the following:

Lemma 2.3 *Suppose that in a given execution a counting network with output sequence $y_0, \ldots, y_{w-1}$ is in a state where m tokens have entered the network and m' tokens have left it. Then there exist non-negative integers d_i, $0 \leq i < w$, such that $\sum_{i=0}^{w-1} d_i = m - m'$ and $y_i + d_i = \left\lceil \frac{m-i}{w} \right\rceil$.*

Proof: Suppose not. There is some execution e for which the non-negative integers d_i, $0 \leq i < w$ do not exist. If we extend e to a complete execution e' allowing no additional tokens to enter the network, then at the end of e' the network will be in a quiescent state where the step property does not hold, a contradiction. ■

In a sequential execution, where tokens traverse the network one at a time, the network is quiescent every time a token leaves. In this case the i-th token to enter will leave on output $i \bmod w$. The lemma shows that in a concurrent, asynchronous execution of any counting network, any "gap" in this sequence of mod w counts corresponds to tokens still traversing the network. This critical property holds in any execution, even if quiescent states never occur, and even though the definition makes no explicit reference to non-quiescent states.

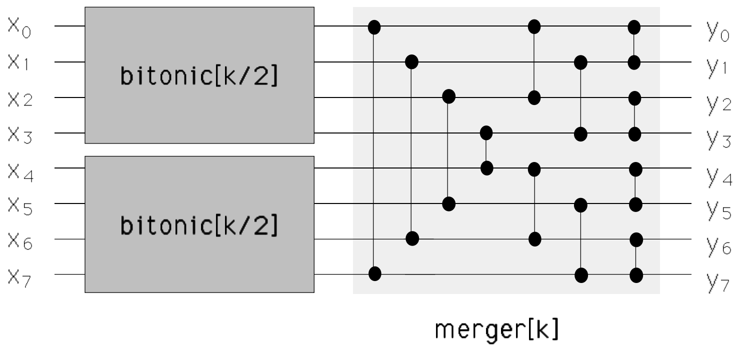


Figure 3: Recursive Structure of a BITONIC[8] Counting Network.

2.2 Counting vs. Sorting

A balancing network and a comparison network are *isomorphic* if one can be constructed from the other by replacing balancers by comparators or vice versa. The counting networks introduced in this paper are isomorphic to the Bitonic sorting network of Batcher [7] and to the Periodic Balanced sorting network of Dowd, Perl, Rudolph and Saks [9]. There is a sense in which constructing counting networks is "harder" than constructing sorting networks:

Theorem 2.4 *If a balancing network counts, then its isomorphic comparison network sorts, but not vice versa.*

Proof: It is easy to verify that balancing networks isomorphic to the EVEN-ODD or INSERTION sorting networks [8] are not counting networks.

For the other direction, we construct a mapping from the comparison network transitions to the isomorphic balancing network transitions.

By the 0-1 principle [8], a comparison network which sorts all sequences of 0's and 1's is a sorting network. Take any arbitrary sequence of 0's and 1's as inputs to the comparison network, and for the balancing network place a token on each 0 input wire and no token on each 1 input wire. We now show that if we run both networks in lockstep, the balancing network will simulate the comparison network, that is, the correspondence between tokens and 0's holds.

The proof is by induction on the depth of the network. For level 0 the claim holds by construction. Assuming it holds for wires of a given level k, let us prove it holds for level $k+1$. On every gate where two 0's meet in the comparison network, two tokens meet in the balancing network, so one 0 leaves on each wire in the comparison network on level $k+1$, and one token leaves on each line in the balancing network on level $k+1$. On every gate where two 1's meet in the comparison network, no tokens meet in the balancing network, so a 1 leaves on each level $k+1$ wire in the comparison network, and no tokens leave in the balancing network. On every gate where a 0 and 1 meet in the comparison network, the 0 leaves on the lower wire and the 1 on the upper wire on level $k+1$, while in the balancing network the token leaves on the lower wire, and no token leaves on the upper wire.

If the balancing network is a counting network, i.e., it has the step property on its output level wires, then the comparison network must have sorted the input sequence of 0's and 1's. ■

Corollary 2.5 *The depth of any counting network is at least* $\Omega(\log n)$.

Though in general a balancing network isomorphic to a sorting network is not guaranteed to count, its outputs will always have the step property if the input sequence satisfies the following *smoothness property*:

A sequence $x_0, ..., x_{w-1}$ is *smooth* if for all $i < j$, $|x_i - x_j| \leq 1$.

This observation is stated formally below:

Theorem 2.6 *If a balancing network is isomorphic to a sorting network, and its input sequence is smooth, then its output sequence in any quiescent state has the step property.*

Proof: The proof follows along the lines of Theorem 2.4. We will show the result by constructing a mapping, this time from the transitions of the balancing network to the transitions of the isomorphic sorting network. However, unlike in the proof of Theorem 2.4, we will map sets of transitions of the balancing network to single transitions of the isomorphic sorting network. We do this by considering the *number* of tokens that have passed along each wire of a balancing network in an execution ending in a quiescent state. From this perspective the transitions of a balancer gate can be mapped to those of a mathematical device that receives integers x_0 and x_1 (numbers of tokens) and outputs integers $\left\lceil \frac{x_0+x_1}{2} \right\rceil$ and $\left\lfloor \frac{x_0+x_1}{2} \right\rfloor$.

Given that the input sequence to the balancing network is smooth, there is a quantity x such that each input wire carries either x or $x+1$ tokens. By simple induction on the depth of the network, one can prove that the inputs and outputs of any balancer in a network with x or $x+1$ tokens on each input wire, will have as outputs x or $x+1$ tokens, and that for a given balancer:

1. If both input wires have x tokens, then both outputs will have x.
2. If one input has x and the other has $x+1$, then the output on the top wire will be $x+1$ tokens and on the bottom wire it will be x tokens.
3. If both input wires have $x+1$ tokens, then both output wires will have $x+1$ tokens.

This behavior, if one considers x and $x+1$ as integers, maps precisely to that of comparators of numeric values in a comparison network. Consequently, in a quiescent state of a balancing network isomorphic to a sorting network, if the network as a whole was given a smooth input sequence, its output sequence must map to a sorted sequence of integers x and $x+1$, implying that it has the step property. ■

3 A Bitonic Counting Network

Naturally, counting networks are interesting only if they can be constructed. In this section we describe how to construct a counting network whose width is any power of 2. The layout of this network is isomorphic to Batcher's famous Bitonic sorting network [7, 8], though its behavior and correctness arguments are completely different. We give an inductive construction, as this will later aid us in proving its correctness.

Define the width w balancing network MERGER$[w]$ as follows. It has two sequences of inputs of length $w/2$, x and x', and a single sequence of outputs y, of length w. MERGER$[w]$ will be constructed to guarantee that in a quiescent state where the sequences x and x' have the step property, y will also have the step property, a fact which will be proved in the next section.

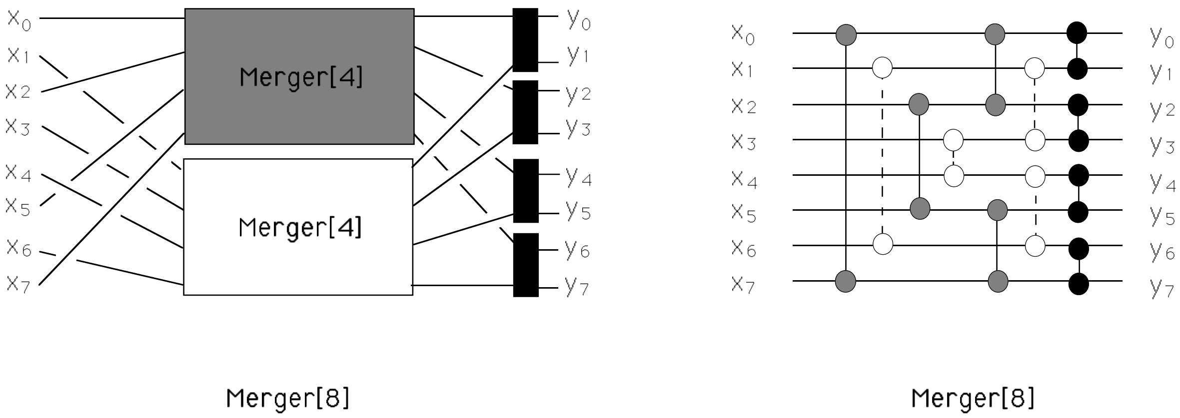


Figure 4: A MERGER [8] balancing network.

We define the network MERGER$[w]$ inductively (see example in Figure 4). Since w is a power of 2, we will repeatedly use the notation $2k$ in place of w. When k is equal to 1, the MERGER$[2k]$ network consists of a single balancer. For $k > 1$, we construct the MERGER$[2k]$ network with input sequences x and x' from two MERGER$[k]$ networks and k balancers. Using a MERGER$[k]$ network we merge the even subsequence $x_0, x_2, \ldots, x_{k-2}$ of x with the odd subsequence $x'_1, x'_3, \ldots, x'_{k-1}$ of x' (i.e., the sequence $x_0, \ldots, x_{k-2}, x'_1, \ldots, x'_{k-1}$ is the input to the MERGER$[k]$ network) while with a second MERGER$[k]$ network we merge the odd subsequence of x with the even subsequence of x'. Call the outputs of these two MERGER$[k]$ networks z and z'. The final stage of the network combines z and z' by sending each pair of wires z_i and z'_i into a balancer whose outputs yield y_{2i} and y_{2i+1}.

The MERGER$[w]$ network consists of $\log w$ layers of $w/2$ balancers each. MERGER$[w]$ guarantees the step property on its outputs only when its inputs also have the step property— but we can ensure this property by filtering these inputs through smaller counting networks. We define BITONIC$[w]$ to be the network constructed by passing the outputs from two BITONIC$[w/2]$ networks into a MERGER$[w]$ network, where the induction is grounded in the BITONIC$[1]$ network which contains no balancers and simply passes its input directly to its output. This construction gives us a network consisting of $\binom{\log w+1}{2}$ layers each consisting of $w/2$ balancers.

3.1 Proof of Correctness

In this section we show that BITONIC$[w]$ is a counting network. Before examining the network itself, we present some simple lemmas about sequences having the step property.

Lemma 3.1 *If a sequence has the step property, then so do all its subsequences.*

Lemma 3.2 *If $x_0, \ldots, x_{k-1}$ has the step property, then its even and odd subsequences satisfy:*

$$\sum_{i=0}^{k/2-1} x_{2i} = \left\lceil \sum_{i=0}^{k-1} x_i/2 \right\rceil \quad \textit{and} \quad \sum_{i=0}^{k/2-1} x_{2i+1} = \left\lfloor \sum_{i=0}^{k-1} x_i/2 \right\rfloor$$

Proof: Either $x_{2i} = x_{2i+1}$ for $0 \leq i < k/2$, or by Lemma 2.2 there exists a unique j such that $x_{2j} = x_{2j+1} + 1$ and $x_{2i} = x_{2i+1}$ for all $i \neq j$, $0 \leq i < k/2$. In the first case, $\sum x_{2i} = \sum x_{2i+1} = \sum x_i/2$, and in the second case $\sum x_{2i} = \lceil \sum x_i/2 \rceil$ and $\sum x_{2i+1} = \lfloor \sum x_i/2 \rfloor$. ■

Lemma 3.3 *Let $x_0, \ldots, x_{k-1}$ and $y_0, \ldots, y_{k-1}$ be arbitrary sequences having the step property. If $\sum_{i=0}^{k-1} x_i = \sum_{i=0}^{k-1} y_i$, then $x_i = y_i$ for all $0 \leq i < k$.*

Proof: Let $m = \sum x_i = \sum y_i$. By Lemma 2.2, $x_i = y_i = \left\lceil \frac{m-i}{k} \right\rceil$. ■

Lemma 3.4 *Let $x_0, \ldots, x_{k-1}$ and $y_0, \ldots, y_{k-1}$ be arbitrary sequences having the step property. If $\sum_{i=0}^{k-1} x_i = \sum_{i=0}^{k-1} y_i + 1$, then there exists a unique j, $0 \leq j < k$, such that $x_j = y_j + 1$, and $x_i = y_i$ for $i \neq j$, $0 \leq i < k$.*

Proof: Let $m = \sum x_i = \sum y_i + 1$. By Lemma 2.2, $x_i = \left\lceil \frac{m-i}{k} \right\rceil$ and $y_i = \left\lceil \frac{m-1-i}{k} \right\rceil$. These two terms agree for all i, $0 \leq i < k$, except for the unique i such that $i = m - 1 \pmod{k}$. ■

We now show that the MERGER$[w]$ networks preserves the step property.

Lemma 3.5 *If* MERGER$[2k]$ *is quiescent, and its inputs $x_0, \ldots, x_{k-1}$ and $x'_0, \ldots, x'_{k-1}$ both have the step property, then its outputs $y_0, \ldots, y_{2k-1}$ have the step property.*

Proof: We argue by induction on $\log k$.

If $2k = 2$, MERGER$[2k]$ is just a balancer, so its outputs are guaranteed to have the step property by the definition of a balancer.

If $2k > 2$, let $z_0, \ldots, z_{k-1}$ be the outputs of the first MERGER$[k]$ subnetwork, which merges the even subsequence of x with the odd subsequence of x', and let $z'_0, \ldots, z'_{k-1}$ be the outputs of the second. Since x and x' have the step property by assumption, so do their even and odd subsequences (Lemma 3.1), and hence so do z and z' (induction hypothesis). Furthermore, $\sum z_i = \lceil \sum x_i/2 \rceil + \lfloor \sum x'_i/2 \rfloor$ and $\sum z'_i = \lfloor \sum x_i/2 \rfloor + \lceil \sum x'_i/2 \rceil$ (Lemma 3.2). A straightforward case analysis shows that $\sum z_i$ and $\sum z'_i$ can differ by at most 1.

We claim that $0 \leq y_i - y_j \leq 1$ for any $i < j$. If $\sum z_i = \sum z'_i$, then Lemma 3.3 implies that $z_i = z'_i$ for $0 \leq i < k/2$. After the final layer of balancers,

$$y_i - y_j = z_{\lfloor i/2 \rfloor} - z_{\lfloor j/2 \rfloor},$$

and the result follows because z has the step property.

Similarly, if $\sum z_i$ and $\sum z'_i$ differ by one, Lemma 3.4 implies that $z_i = z'_i$ for $0 \leq i < k/2$, except for a unique ℓ such that z_ℓ and z'_ℓ differ by one. Let $max(z_\ell, z'_\ell) = x + 1$ and $min(z_\ell, z'_\ell) = x$ for some non-negative integer x. From the step property on z and z' we have, for all $i < \ell$, $z_i = z'_i = x + 1$ and for all $i > \ell$ $z_i = z'_i = x$. Since z_ℓ and z'_ℓ are joined by a balancer with outputs $y_{2\ell}$ and $y_{2\ell+1}$, it follows that $y_{2\ell} = x + 1$ and $y_{2\ell+1} = x$. Similarly, z_i and z'_i for $i \neq \ell$ are joined by the same balancer. Thus for any $i < \ell$, $y_{2i} = y_{2i+1} = x + 1$ and for any $i > \ell$, $y_{2i} = y_{2i+1} = x$. The step property follows by choosing $c = 2\ell + 1$ and applying Lemma 2.2.

∎

The proof of the following theorem is now immediate.

Theorem 3.6 *In any quiescent state, the outputs of* BITONIC$[w]$ *have the step property.*

4 A Periodic Counting Network

In this section we show that the bitonic network is not the only counting network with depth $O(log^2 n)$. We introduce a new counting network with the interesting property that it is *periodic*, consisting of a sequence of identical subnetworks. Each stage of this periodic network is interesting in its own right, since it can be used to achieve barrier synchronization with low contention. This counting network is isomorphic to the elegant *balanced periodic sorting network* of Dowd, Perl, Rudolph, and Saks [9]. However, its behavior, and therefore also our proof of correctness, are fundamentally different.

We start by defining chains and cochains, notions taken from [9]. Given a sequence $x = \{x_i | i = 0, \ldots, n-1\}$, it is convenient to represent each index (subscript) as a binary string. A *level i chain* of x is a subsequence of x whose indices have the same i low-order bits. For example, the subsequence x^E of entries with even indices is a level 1 chain, as is the subsequence x^O of entries with odd indices. The *A-cochain* of x, denoted x^A, is the subsequence whose indices have the two low-order bits 00 or 11. For example, the *A-cochain* of the sequence $x_0, \ldots, x_7$ is x_0, x_3, x_4, x_7. The *B-cochain* x^B is the subsequence whose low-order bits are 01 and 10.

Define the network BLOCK$[k]$ as follows. When k is equal to 2, the BLOCK$[k]$ network consists of a single balancer. The BLOCK$[2k]$ network for larger k is constructed recursively. We start with two BLOCK$[k]$ networks A and B. Given an input sequence x, the input to A is x^A, and the input to B is x^B. Let y be the output sequence for the two subnetworks, where y^A is the output sequence for A and y^B the output sequence for B. The final stage of the network combines each y_i^A and y_i^B in a single balancer, yielding final outputs z_{2i} and z_{2i+1}. Figure 5 describes the recursive construction of a BLOCK [8] network. The PERIODIC$[2k]$ network consists of $\log k$ BLOCK$[2k]$ networks joined so that the i^{th} output wire of one is the i^{th} wire of the next. Figure 6 is a PERIODIC[8] counting network [3]

This recursive construction is quite different from the one used by Dowd et al. We chose this construction because it yields a substantially simpler and shorter proof of correctness.

4.1 Proof of Correctness

In the proof we use the technical lemmas about input and output sequences presented in Section 3. The following lemma will serve a key role in the

[3] Despite the apparent similarities between the layouts of the BLOCK and MERGER networks, there is no permutation of wires that yields one from the other.

inductive proof of our construction:

Lemma 4.1 *For $i > 1$,*

1. *The level i chain of x is a level $i-1$ chain of one of x's cochains.*
2. *The level i chain of a cochain of x is a level $i+1$ chain of x.*

Proof: Follows immediately from the definitions of chains and cochains. ■

As will be seen, the price of modularity is redundancy, that is, balancers in lower level blocks will be applied to sub-sequences that already have the desired step property. We therefore present the following lemma that amounts to saying that applying balancers "evenly" to such sequences does not hurt:

Lemma 4.2 *If x and x' are sequences each having the step property, and pairs x_i and x'_i are routed through a balancer, yielding outputs y_i and y'_i, then the sequences y and y' each have the step property.*

Proof: For any $i < j$, given that x and x' have the step property, $0 \leq x_i - x_j \leq 1$ and $0 \leq x'_i - x'_j \leq 1$ and therefore the difference between any two wires is $0 \leq x_i + x'_i - (x_j + x'_j) \leq 2$. By definition, for any i, $y_i = \left\lceil \frac{x_i + x'_i}{2} \right\rceil$ and $y'_i = \left\lfloor \frac{x_i + x'_i}{2} \right\rfloor$, and so for any $i < j$, it is the case that $0 \leq y_i - y_j \leq 1$ and $0 \leq y'_i - y'_j \leq 1$, implying the step property. ■

To prove the correctness of our construction for Periodic$[k]$, we will show that if a block's level i input chains have the step property, then so do its level $i-1$ output chains, for i in $\{0, \ldots, \log k - 1\}$. This observation implies that a sequence of $\log k$ Block$[k]$ networks will count an arbitrary number of inputs.

Lemma 4.3 *Let* Block$[2k]$ *be quiescent with input sequence x and output sequence y. If x^E and x^O both have the step property, so does y.*

Proof: We argue by induction on $\log k$. The proof is similar to that of Lemma 3.5.

For the base case, when $2k = 2$, Block$[2k]$ is just a balancer, so its outputs are guaranteed to have the step property by the definition of a balancer.

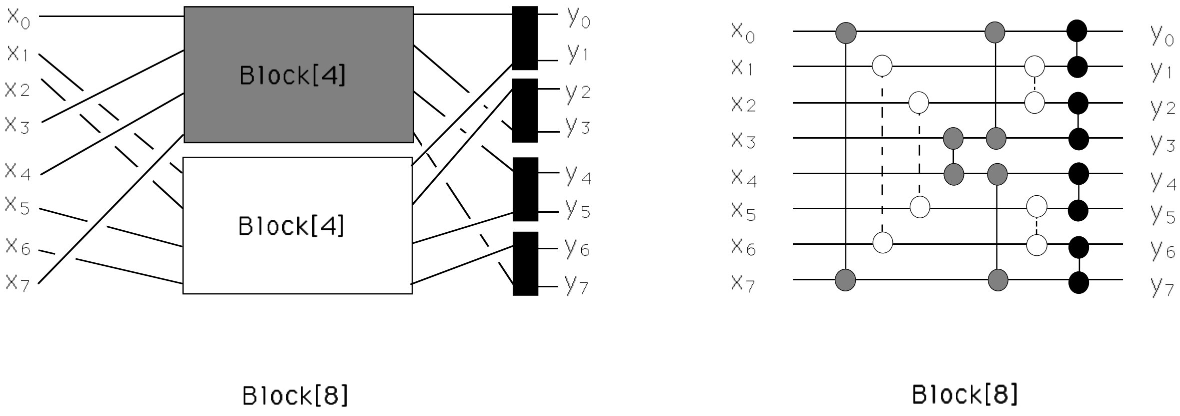


Figure 5: A BLOCK [8] balancing network.

For the induction step, assume the result for BLOCK$[k]$ and consider a BLOCK$[2k]$. Let x be the input sequence to the block, z the output sequence of the nested blocks A and B, and y the block's final output sequence. The inputs to A are the level 2 chains x^{EE} and x^{OO}, and the inputs to B are x^{EO} and x^{OE}. By Lemma 4.1, each of these is a level 1 chain of x^A or x^B. These sequences are the inputs to A and B, themselves of size k, so the induction hypothesis implies that the outputs z^A and z^B of A and B each has the step property.

Lemma 3.2 implies that $0 \leq \sum x_i^{EE} - \sum x_i^{EO} \leq 1$ and $0 \leq \sum x_i^{OE} - \sum x_i^{OO} \leq 1$. It follows that the sum of A's inputs, $\sum x_i^{EE} + \sum x_i^{OO}$, and the sum of B's inputs, $\sum x_i^{EO} + \sum x_i^{OO}$, differ by at most 1. Since balancers do not swallow or create tokens, $\sum z^A$ and $\sum z^B$ also differ by at most 1. If they are equal, then Lemma 3.3 implies that $z_i^A = z_i^B = z_{2i} = z_{2i+1}$. For $i < j$,

$$y_i - y_j = z^A_{\lfloor i/2 \rfloor} - z^A_{\lfloor j/2 \rfloor}$$

and the result follows because z^A has the step property.

Similarly, if $\sum z_i^A$ and $\sum z_i^B$ differ by one, Lemma 3.4 implies that $z_i^A = z_i^B$ for $0 \leq i < k$, except for a unique ℓ such that z_ℓ^A and z_ℓ^B differ by one. Let $max(z_\ell^A, z_\ell^B) = x + 1$ and $min(z_\ell^A, z_\ell^B) = x$ for some non-negative integer x. From the step property on z^A and z^B we have, for all $i < \ell$, $z_i^A = z_i^B = x + 1$ and for all $i > \ell$ $z_i^A = z_i^B = x$. Since z_ℓ^A and z_ℓ^B are joined by a balancer with outputs $y_{2\ell}$ and $y_{2\ell+1}$, it follows that $y_{2\ell} = x + 1$ and $y_{2\ell+1} = x$. Similarly, z_i^A and z_i^B for $i \neq \ell$ are joined by the same balancer. Thus for any $i < \ell$, $y_{2i} = y_{2i+1} = x + 1$ and for any $i > \ell$, $y_{2i} = y_{2i+1} = x$. The step property follows by choosing $c = 2\ell + 1$ and applying Lemma 2.2. ∎

Theorem 4.4 *Let* $\text{BLOCK}[2k]$ *be quiescent with input sequence* x *and output sequence* y*. If all the level* i *input chains to a block have the step property, then so do all the level* $i - 1$ *output chains.*

Proof: We argue by induction on i. Lemma 4.3 provides the base case, when i is 1.

For the induction step, assume the result for chains up to $i - 1$. Let x be the input sequence to the block, z the output sequence of the nested blocks A and B, and y the block's final output sequence. If $i > 1$, Lemma 4.1 implies that every level i chain of x is entirely contained in one cochain or the other. Each level i chain of x contained in x^A (x^B) is a level $i - 1$ chain of x^A (x^B), each has the step property, and each is an input to A (B). The induction hypothesis applied to A and B implies that the level $i - 2$ chains of z^A and z^B have the step property. But Lemma 4.1 implies that the level $i - 2$ chains of z^A and z^B are the level $i - 1$ chains of z. By Lemma 4.2, if the level $i - 1$ chains of z have the step property, so do the level $i - 1$ chains of y. ∎

By Theorem 2.4, the proof of Theorem 4.4 constitutes a simple alternative proof that the *balanced periodic comparison network* of [9] is a sorting network.

5 Implementation and Applications

In a MIMD shared-memory architecture, a balancer can be represented as a record with two fields: *toggle* is a boolean value that alternates between 0 and 1, and *next* is a 2-element array of pointers to successor balancers. A balancer is a *leaf* if it has no successors. A process shepherds a token through the network by executing the procedure shown in Figure 7. In our implementations, we preassigned processes to input lines so that they were evenly distributed. Thus, a given process always started shepherding tokens from the same preassigned line. It toggles the balancer's state, and visits the next balancer, halting when it reaches a leaf. The network's wiring

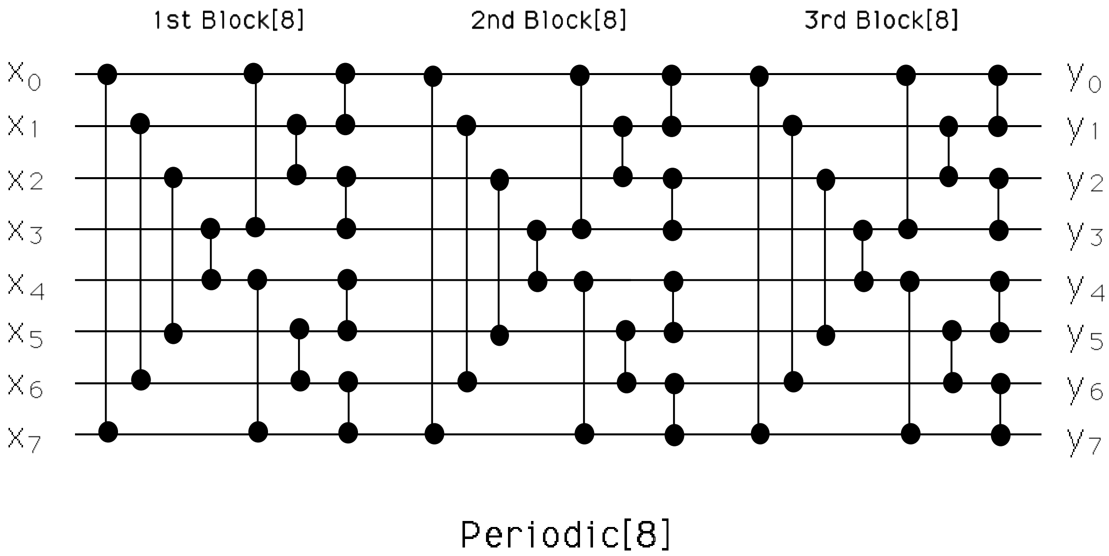


Figure 6: A PERIODIC [8] counting network.

```
balancer = [toggle: boolean, next: array [0..1] of ptr to balancer]
traverse(b: balancer)
   loop until leaf(b)
         i := rmw(b.toggle := ¬ b.toggle)
         b := b.next[i]
         end loop
   end traverse
```

Figure 7: Code for Traversing a Balancing Network

information can be cached by each process, and so the transition time of a balancer will be almost entirely a function of the efficiency of the toggle implementation. Advancing the *toggle* state can be accomplished either by a short critical section guarded by a spin lock[4], or by a *read-modify-write* operation (*rmw* for short) if the hardware supports it. Note that all values are bounded.

We illustrate the utility of counting networks by constructing highly concurrent implementations of three common data structures: shared counters, producer/consumer buffers, and barriers. In Section 6 we give some experimental evidence that counting network implementations have higher throughput than conventional implementations when contention is sufficiently high.

5.1 Shared Counter

A *shared counter* [12, 10, 16, 31] is a data structure that issues consecutive integers in response to *increment* requests. More formally, in any quiescent state in which m increment requests have been received, the values 0 to $m-1$ have been issued in response. To construct the counter, start with an arbitrary width-w counting network. Associate an integer cell c_i with the i^{th} output wire. Initially, c_i holds the value i. A process requests a number by traversing the counting network. When it exits the network on wire i, it atomically adds w to the value of c_i and returns c_i's previous value.

Lemmas 2.1 and 2.3 imply that:

[4]A spin lock is just a shared boolean flag that is raised and lowered by at most one processor at a time, while the other processors wait.

Lemma 5.1 *Let x be the largest number yet returned by any increment request on the counter. Let R be the set of numbers less than x which have not been issued to any increment request. Then*

1. *The size of R is no greater than the number of operations still in progress.*
2. *If $y \in R$, then $y \geq x - w|R|$.*
3. *Each number in R will be returned by some operation in time $\Delta \cdot d + \Delta_c$, where d is the depth of the network, Δ is the maximum balancer delay, and Δ_c is the maximum time to update a cell on an output wire.*

5.2 Producer/Consumer Buffer

A *producer/consumer buffer* is a data structure in which items inserted by a pool of m producer processes are removed by a pool of m consumer processes. The buffer algorithm used here is essentially that of Gottlieb, Lubachevsky, and Rudolph [16]. The buffer is a w-element array *buff*$[0..w-1]$. There are two w-width counting networks, a *producer* network, and a *consumer* network. A producer starts by traversing the producer network, leaving the network on wire i. It then atomically inspects *buff*$[i]$, and, if it is $\perp$, replaces it with the produced item. If that position is full, then the producer waits for the item to be consumed (or returns an exception). Similarly, a consumer traverses the consumer network, exits on wire j, and if *buff*$[j]$ holds an item, atomically replaces it with $\perp$. If there is no item to consume, the consumer waits for an item to be produced (or returns an exception).

Lemmas 2.1 and 2.3 imply that:

Lemma 5.2 *Suppose m producers and m' consumers have entered a producer/consumer buffer built out of counting networks of depth d. Assume that the time to update each buff$[i]$ once a process has left the counting network is negligible. Then if $m \leq m'$, every producer leaves the network in time $d\Delta$. Similarly, if $m \geq m'$, every consumer leaves the network in time $d\Delta$.*

5.3 Barrier Synchronization

A *barrier* is a data structure that ensures that no process advances beyond a particular point in a computation until all processes have arrived at that point. Barriers are often used in highly-concurrent numerical computations to divide the work into disjoint *phases* with the property that no process executes phase i while another process concurrently executes phase $i+1$.

A simple way to construct an n-process barrier is by exploiting the following key observation: Lemma 2.3 implies that as soon as some process exits with value n, the last phase must be complete, since the other $n-1$ processes must already have entered the network.

We present a stronger result: one does not need a full counting network to achieve barrier synchronization. A *threshold network* of width w is a balancing network with input sequence x_i and output sequence y_i, such that the following holds:

> In any quiescent state, $y_{w-1} = m$ if and only if $mw \leq \sum x_i < (m+1)w$.

Informally, a threshold network can "detect" each time w tokens have passed through it. A counting network is a threshold network, but not vice-versa.

Both the BLOCK$[w]$ network used in the periodic construction and the MERGER$[w]$ network used in the bitonic construction are threshold networks, provided the input sequence satisfies the smoothness property. Recall that a sequence $x_0, ..., x_{w-1}$ is *smooth* if for all $i < j$, $|x_i - x_j| \leq 1$. Every sequence with the step property is smooth, but not vice-versa. The following two lemmas state that smoothness is "stable" under partitioning into subsequences or application of additional balancers.

Lemma 5.3 *Any subsequence of a smooth sequence is smooth.*

Lemma 5.4 *If the input sequence to a balancing network is smooth, so is the output sequence.*

Proof: Observe that if the inputs to a balancer differ by at most one, then so do its outputs. By a simple induction on the depth of the network, the output sequence from the balancers at any level of a balancing network with a smooth input sequence, is a permutation of its input sequence, hence, the network's output sequence is smooth. ■

Theorem 5.5 *If the input sequence to* BLOCK$[w]$ *is smooth, then* BLOCK$[w]$ *is a threshold network.*

Proof: Let x_i be the block's input sequence, z_i the output sequence of nested blocks A and B, and y_i the block's output sequence.

We first show that if $y_{w-1} = m$, then $m\,w \leq \sum x_i < (m+1)w$. We argue by induction on w, the block's width. If $w = 2$, the result is immediate. Assume the result for $w = k$ and consider $\textsc{Block}[2k]$ in a quiescent state where $y_{2k-1} = m$. Since x is smooth by hypothesis, by Lemma 5.4 so are z and y. Since y_{2k-1} and y_{2k-2} are outputs of a common balancer, y_{2k-2} is either m or $m+1$. The rest is a case analysis.

If $y_{2k-1} = y_{2k-2} = m$, then $z_{2k-1} = z_{2k-2} = m$. By the induction hypothesis and Lemma 5.3 applied to A and B, $mk \leq \sum x_i^A < (m+1)k$ and $mk \leq \sum x_i^B < (m+1)k$, and therefore $2mk \leq \sum x_i^A + \sum x_i^B < 2(m+1)k$.

If $y_{2k-2} = m+1$, then one of z_i^A and z_i^B is m, and the other is $m+1$. Without loss of generality suppose $z_i^A = m+1$ and $z_i^B = m$. By the induction hypothesis, $(m+1)k \leq \sum x_i^A < (m+2)k$ and $mk \leq \sum x_i^B < (m+1)k$. Since x is smooth, by Lemma 5.3 x^B is smooth and some element of x^B must be equal m, which in turn implies that no element of x^A exceeds $m+1$. This bound implies that $(m+1)k = \sum x_i^A$. It follows that $2mk + k \leq \sum x_i^A + \sum x_i^B < 2(m+1)k$, yielding the desired result.

We now show that if $m\,w \leq \sum x_i < (m+1)w$, then $y_{w-1} = m$. We again argue by induction on w, the block's width. If $w = 2$, the result is immediate. Assume the result for $w = k$ and consider $\textsc{Block}[2k]$ in a quiescent state where $2mk \leq \sum x_i < 2(m+1)k$. Since x is smooth, by Lemma 5.4 $m \leq y_{2i-1}$. Furthermore, since x is smooth, by Lemma 5.3, either $mk \leq \sum x_i^A \leq (m+1)k$ and $mk \leq \sum x_i^B < (m+1)k$ or vice versa, which by the induction hypothesis implies that $z_{k-1}^A + z_{k-1}^B \leq 2m+1$. It follows that $y_{2k-1} < m+1$, which completes our claim. ∎

The proof that the $\textsc{Merger}[w]$ network is also a threshold network if its inputs are smooth is omitted because it is almost identical to that of Theorem 5.5. A *threshold counter* is constructed by associating a local counter c_i with each output wire i, just as in the counter construction.

We construct a barrier for n processes, where $n = 0 \bmod w$, using a width-w threshold counter. The construction is an adaptation of the "sense-reversing" barrier construction of [18] as follows. Just as for the counter construction, we associate a local counter c_i with each output wire i. Let F be a boolean flag, initially *false*. Let a process's phase at a given point in the execution of the barrier algorithm be defined as 0 initially, and incremented by 1 every time the process begins traversing the network. With each phase the algorithm will associate a *sense*, a boolean value reflecting the phase's parity: *true* for the first phase, *false* for the second, and so on. As illustrated in Figure 8, the token for process P, after a phase with sense s, enters the network on wire $P \bmod w$. If it emerges with a value not equal to $n-1 \bmod n$, then it waits until F agrees with s before starting the next phase. If it emerges with value $n-1 \bmod n$, it sets F to s, and starts the next phase.

As an aside, we note that a threshold counter implemented from a BLOCK$[k]$ network can be optimized in several additional ways. For example, it is only necessary to associate a local counter with wire $w-1$, and that counter can be modulo n rather than unbounded. Moreover, all balancers that are not on a path from some input wire to exit wire $w-1$ can be deleted.

Theorem 5.6 *If P exits the network with value n after completing phase ϕ, then every other process has completed phase ϕ, and no process has started phase $\phi+1$.*

Proof: We first observe that the input to BLOCK$[w]$ is smooth, and therefore it is a threshold network. We argue by induction. When P receives value $v = n-1$ at the end of the first phase, exactly n tokens must have entered BLOCK$[w]$, and all processes must therefore have completed the first phase. Since the boolean F is still *false*, no process has started the second phase. Assume the result for phase ϕ. If Q is the process that received value $n-1$ at the end of that phase, then exactly ϕn tokens had entered the network when Q performed the reset of F. If P receives value $v = n-1$ at the end of phase $\phi+1$, then exactly $(\phi+1)n$ tokens have entered the network, implying that an additional n tokens have entered, and all n processes have finished the phase. No process will start the next phase until F is reset. ∎

6 Performance

6.1 Overview

In this section, we analyze counting network throughput for computations in which tokens are eventually spread evenly through the network. As mentioned before, to ensure that tokens are evenly spread across the input wires,

```
barrier()
   v := exit wire of traverse(wire P mod w)
   if v = n − 1   (mod w)
      then F := s
      else  wait until F = s
      end if
   s := ¬s
   end barrier
```

Figure 8: Barrier Synchronization Code

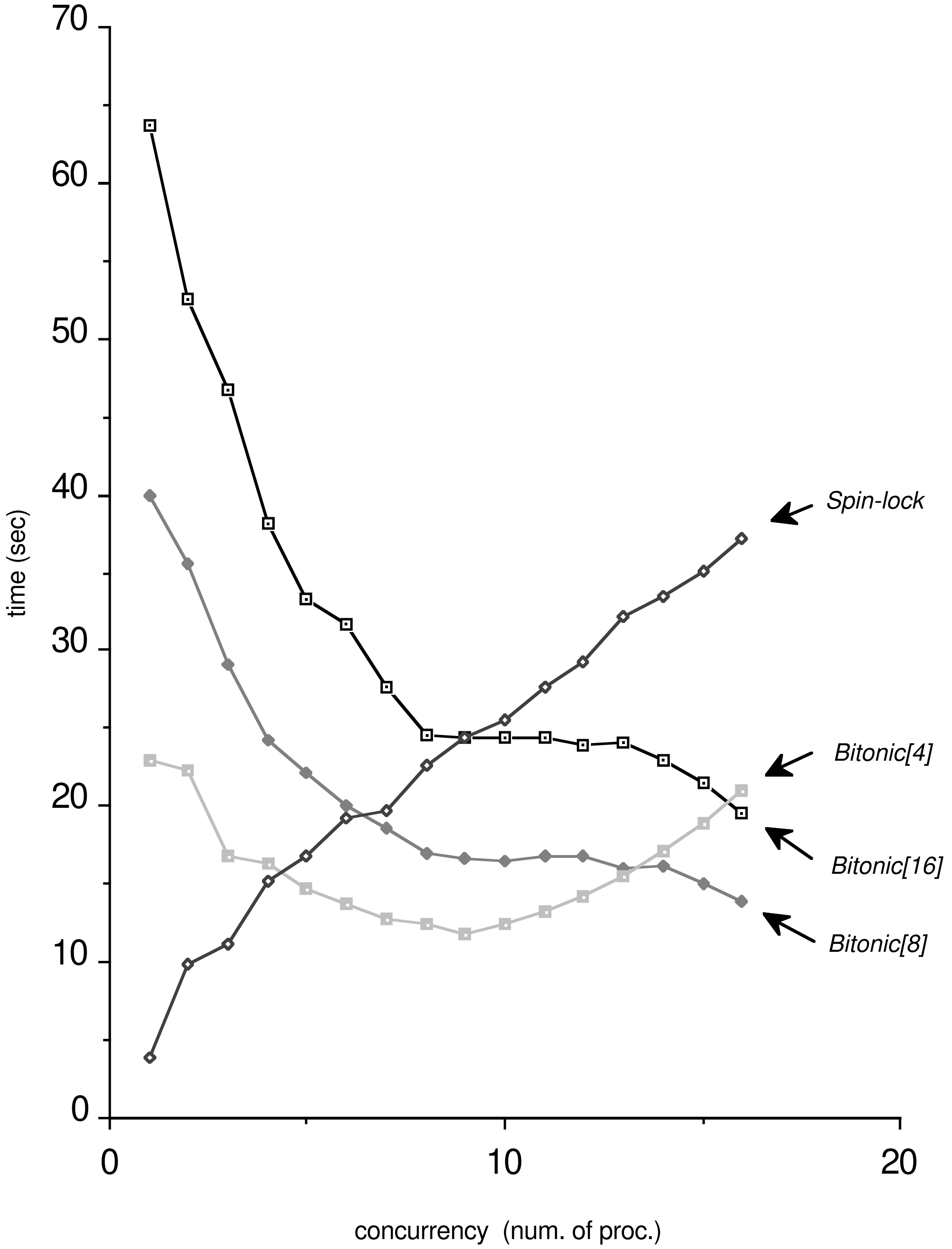
each processor could be assigned a fixed input wire. Alternatively, processors could choose input wires at random.

The network *saturation* S at a given time is defined to be the ratio of the number of tokens n present in the network (i.e. the number of processors shepherding tokens through it) to the number of balancers. If tokens are spread evenly through the network, then the saturation is just the expected number of tokens at each balancer. For the BITONIC and PERIODIC networks, $S = 2n/wd$. The network is *oversaturated* if $S > 1$, and *undersaturated* if $S < 1$.

An oversaturated network represents a full pipeline, hence its throughput is dominated by the per-balancer contention, not by the network depth. If a balancer with S tokens makes a transition in time $\Delta(S)$, then approximately $w/2$ tokens emerge from the network every $\Delta(S)$ time units, yielding a throughput of $w/2\Delta(S)$. Δ is an increasing function whose exact form depends on the particular architecture, but similar measures of degradation have been observed in practice to grow linearly [5, 25]. The throughput of an oversaturated network is therefore maximized by choosing w and d to minimize S, bringing it as close as possible to 1.

The throughput of an undersaturated network is dominated by the network depth, not by the per-balancer contention, since the network pipeline is partially empty. Every $1/S$ time units, $w/2$ tokens leave the network, yielding throughput $\frac{wS}{2}$. The throughput of an undersaturated network is therefore maximized by choosing w and d to increase S, bringing it as close as possible to 1.

This analysis is necessarily approximate, but it is supported by exper-



1

Figure 9: Bitonic Shared Counter Implementations

imental evidence. In the remainder of this section, we present the results of timing experiments for several data structures implemented using counting networks. As a control, we compare these figures to those produced by more conventional implementations using spin locks These implementations were done on an Encore Multimax, using Mul-T [21], a parallel dialect of Lisp. The spin lock is a simple "test-and-test-and-set" loop [26] written in assembly language, and provided by the Mul-T run-time system. In our implementations, each balancer is protected by a spin lock.

6.2 The Shared Counter

We compare seven shared counter implementations: bitonic and periodic counting networks of widths 16, 8, and 4, and a conventional spin lock implementation (which can be considered a degenerate counting network of width 2). For each network, we measured the elapsed time necessary for a 2^{20} (approximately a million) tokens to traverse the network, controlling the level of concurrency.

For the bitonic network, the width-16 network has 80 balancers, the width-8 network has 24 balancers, and the width-4 network has 6 balancers. In Figure 9, the horizontal axis represents the number of processes executing concurrently. When concurrency is 1, each process runs to completion before the next one starts. The number of concurrent processes increases until all sixteen processes execute concurrently. The vertical axis represents the elapsed time (in seconds) until all 2^{20} tokens had traversed the network. With no concurrency, the networks are heavily undersaturated, and the spin lock's throughput is the highest by far. As saturation increases, however, so does the throughput for each of the networks. The width-4 network is undersaturated at concurrency levels less than 6. As the level of concurrency increases from 1 to 6, saturation approaches 1, and the elapsed time decreases. Beyond 6, saturation increases beyond 1, and the elapsed time eventually starts to grow. The other networks remain undersaturated for the range of the experiment; their elapsed times continue to decrease. Each of the networks begins to outperform the spin lock at concurrency levels between 8 and 12. When concurrency is maximal, all three networks have throughputs at least twice the spin lock's. Notice that as the level of concurrency increases, the spin lock's performance degrades in an approximately linear fashion (because of increasing contention).

1

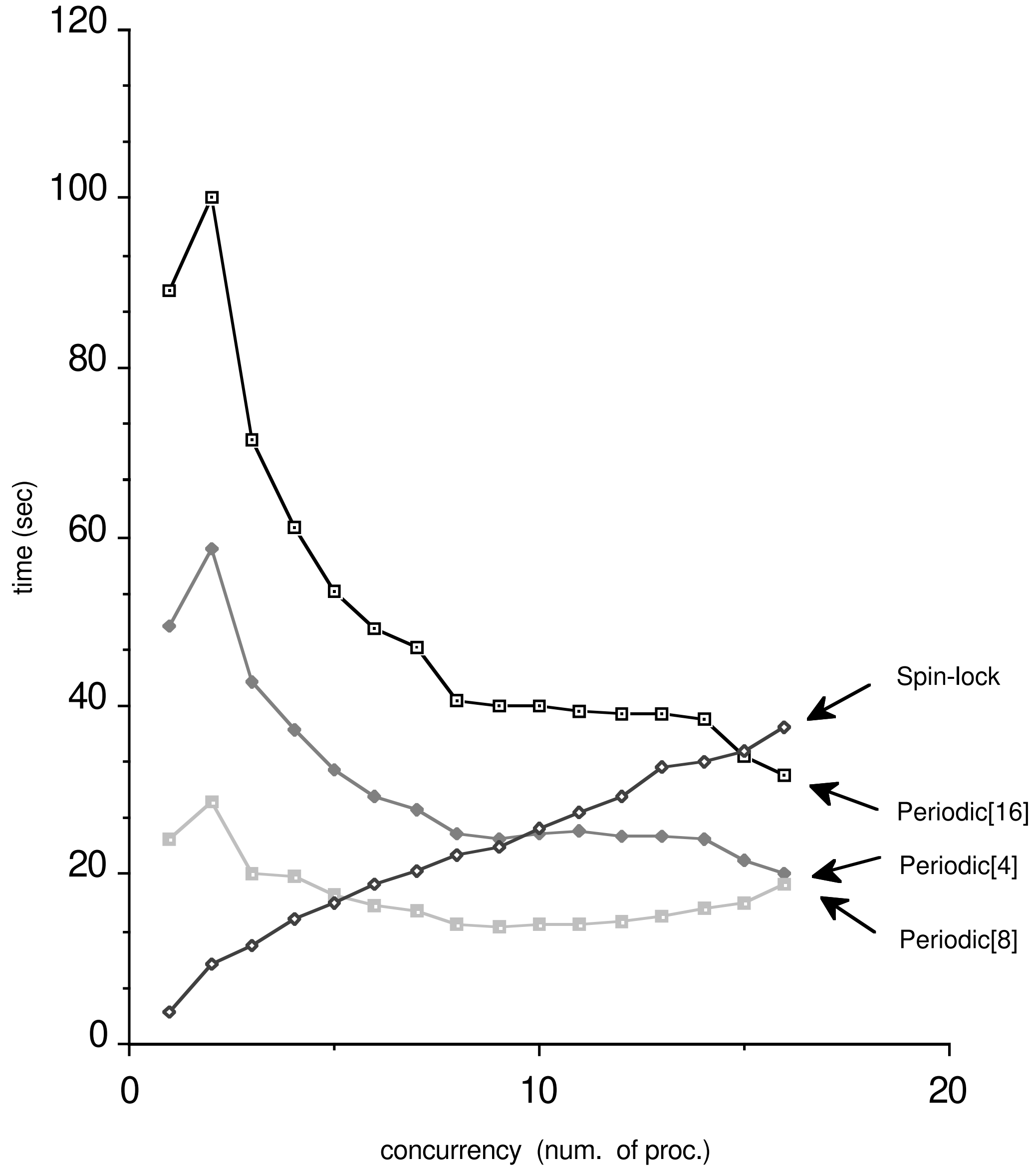


Figure 10: Periodic Shared Counter Implementations

	spin	width 2	width 4	width 8
bitonic	57.74	17.51	10.44	14.25
periodic		17.90	12.03	19.99

Figure 11: Producer/Consumer Buffer Implementations

The performance of the periodic network (Figure 10) is similar. The width-4 network reaches saturation 1 at 8 processes; its throughput then declines slightly as it becomes oversaturated. The other networks remain undersaturated, and their throughputs continue to increase. Each of the counting networks outperforms the spin lock at sufficiently high levels of contention. At 16 processes, the width-4 and width-8 networks have almost twice the throughput of the single spin-lock implementation. Each bitonic network has a slightly higher throughput than its periodic counterpart.

6.3 Producer/Consumer Buffers

We compare the performance of several producer/consumer buffers implemented using the algorithm of Gottlieb, Lubachevsky, and Rudolph [16] discussed in Section 5. Each implementation has 8 producer processes, which continually produce items, and 8 consumer processes, which continually consume items. If a producer (consumer) process finds its buffer slot full (empty), it spins until the slot becomes empty (full).

We consider buffers with bitonic and periodic networks of width 2, 4, and 8. As a final control, we tested a circular buffer protected by a single spin lock, a structure that permits no concurrency between producers and consumers. Figure 11 shows the time in seconds needed to produce and consume 2^{20} tokens. Not surprisingly, the single spin-lock implementation is much slower than any of the others. The width-2 network is heavily oversaturated, the bitonic width-4 network is slightly oversaturated, while the others are undersaturated.

	Spin lock	Barrier 4	Barrier 8	Barrier 16
time (seconds)	62.05	43.53	41.27	42.32

Figure 12: Barrier Implementations

6.4 Barrier Synchronization

Figure 12 shows the time (in seconds) taken by 16 processes to perform 2^{16} barrier synchronizations. The remaining columns show BLOCK$[k]$ networks of width 4, 8, and 16. The last column shows a simple sense-reversing barrier in which the BLOCK network is replaced by a single counter protected by a spin lock. The three network barriers are equally fast, and each takes about two-thirds the time of the spin-lock implementation.

7 Verifying That a Network Counts

The "0-1 law" states that a comparison network is a sorting network if (and only if) it sorts input sequences consisting entirely of zeroes and ones, a property that greatly simplifies the task of reasoning about sorting networks. In this section, we present an analogous result: a balancing network having m balancers is a counting network if (and only if) it satisfies the step property for all *sequential* executions in which up to 2^m tokens have traversed the network. This result simplifies reasoning about counting networks, since it is not necessary to consider all concurrent executions. However, as we show, the number of tokens passed through the network in the longest of these sequential executions cannot be less than exponential in the network depth.

We begin by proving that it suffices to consider only sequential executions.

Lemma 7.1 *Let s be a valid schedule of a given balancing network. Then there exists a valid sequential schedule s' such that the number of tokens which pass through each balancer in s and s' is equal.*

Proof: Let $s = s_0 \cdot p \cdot q \cdot s_1$, where s_0, s_1 are sequences of transitions, p and q are individual transitions involving distinct tokens P and Q, and where "$\cdot$" is the concatenation operator. If p and q do not occur at the same balancer, then $s_0 \cdot q \cdot p \cdot s_1$ is a valid schedule. If p and q do occur at the same balancer, then $s_0 \cdot q \cdot p \cdot s_1'$ is a valid schedule where s_1' is constructed from s_1 by swapping the identities of P and Q. In each case we can swap p and q without changing the preceding sequence of transitions s_0 and without changing the number of tokens that pass through any balancer during the execution.

Now suppose that s is a complete schedule. We will transform it into a sequential schedule by a process similar to selection sorting. Choose some total ordering of the tokens in s. Split s into $s_0 \cdot t_0$ where s_0 is the empty sequence and $t_0 = s$. Now repeatedly carry out the following procedure which constructs $s_{i+1} \cdot t_{i+1}$ from $s_i \cdot t_i$: while t_i is nonempty let p be the earliest transition in t_i whose token is ordered as less than or equal to all tokens in t_i. Move p to the beginning of t_i by swapping it with each earlier token in t_i as described above, and let $s_{i+1} = s_i \cdot p$ and t_{i+1} be the suffix of the resulting schedule after p. This procedure is easily seen to maintain the following invariant:

1. After stage i, $s_i \cdot t_i$ is a valid schedule in which each balancer passes the same number of tokens as in s.

2. After stage i, s_i is sorted by token.

Thus when the procedure terminates, we have a valid sequential schedule s' in which each balancer passes the same number of tokens as in s. ∎

Theorem 7.2 *A balancing network with m balancers satisfies the step property in all executions if (and only if) it satisfies it in all sequential executions in which at most 2^m tokens traverse the network.*

Proof: Since the step property depends only on the number of tokens that pass through the network's output wires, it follows from Lemma 7.1 that a balancing network satisfies the step property in all executions if (and only if) it satisfies it in all *sequential* executions.

We now show that any failure to satisfy the step property can be detected in some execution involving at most 2^m tokens. Consider sequential executions of a balancing network with m balancers. Any quiescent state is characterized by specifying for each balancer the output wire to which it will send the next token, yielding a maximum of 2^m distinct quiescent states. In a sequential execution, each time a token traverses the network, it carries the network from one quiescent state to another. Thus, in any execution, after at most 2^m traversals, the network must reenter its initial state. Let H be the shortest sequential execution needed to detect a violation of the step property. If H involves more than 2^m tokens, then H can be split into a prefix H_0 and a suffix H_1 such that H_0 involves at most 2^m tokens and leaves the network in its initial state. If H_0 sends "illegal" numbers of tokens through two output wires, then H_0 alone suffices to detect the violation, and otherwise H_1 alone suffices. ∎

How tight is this bound? We now construct a balancing network that is not a counting network, yet satisfies the step property for any execution in which the number of tokens is less than exponential in the network depth. Through the remainder of this section we will only consider networks in quiescent states, so that we can ignore issues of timing and concentrate solely on the total number of tokens that have passed along each wire.

First, consider the following balancing network STAGE $[2w]$. Take two counting networks A and B of width w having outputs wires a_0 through a_{w-1} and b_0 through b_{w-1} respectively. Add a layer of w balancers such that the i-th balancer has inputs a_i and b_{w-1-i} and outputs a'_i and b'_{w-1-i}. The resulting network STAGE $[2w]$ is not a counting network; however, it is easily extended to one by virtue of the following lemma.

Lemma 7.3 *For any input to* STAGE $[2w]$, *there exists a permutation* π_a *of the output sequence* $a'_0, \ldots, a'_{w-1}$ *and a permutation* π_b *of the output sequence* $b'_0, \ldots, b'_{w-1}$ *such that the sequence* $\pi_a(a'_0, \ldots, a'_{w-1}) \cdot \pi_b(b'_0, \ldots, b'_{w-1})$ *has the step property.*

Proof: Observe that the total inputs to any two balancers in the last layer differ by at most 1.

Thus there is always a k such that every balancer in the last layer outputs either k or $k+1$ tokens. If k is even, then $b'_i = k/2$ for all i and $a'_i = a_i + b_{w-1-i} - k/2$, which is either $k/2$ or $k/2+1$. One can obtain a sequence with the step property by setting π_a to sort the values in a'. If k is odd, then each a'_i is $(k+1)/2$ and each b'_i is $a_{w-1-i} + b_i - (k+1)/2$, which will be either $(k+1)/2$ or $(k+1)/2 - 1$. In this case having π_b sort the values in b' produces the desired result. ∎

By Lemma 2.2 it follows that

Corollary 7.4 *For any m tokens input to* STAGE $[2w]$, $\sum_{i=0}^{w-1} a'_i = \sum_{i=0}^{w-1} \lceil m - i/2w \rceil$ *and* $\sum_{i=0}^{w-1} b'_i = \sum_{i=w}^{2w-1} \lceil m - i/2w \rceil$.

In other words, the total number of tokens that end up on the $a'_0, \ldots, a'_{w-1}$ and $b'_0, \ldots, b'_{w-1}$ outputs wires is the same as in a proper counting network. In fact, Lemma 7.3 guarantees an even stronger property: the actual number of tokens on each wire correspond to the number of tokens that occur on some wire in the output sequence of a proper counting network. However, there is no guarantee that these numbers appear in the correct order (or even the same order given different inputs). Because of Theorem 2.6, we can extend the STAGE$[2w]$ network into a (not very efficient) counting network by passing the outputs $a'_0, \ldots, a'_{w-1}$ and $b'_0, \ldots, b'_{w-1}$ to two separate balancing networks isomorphic to sorting networks. But we are not interested in getting a working counting network; instead we will use a modified version of STAGE$[2w]$ to construct a balancing network which counts all input sequences with up to some bounded number of tokens, but fails on sequences with more tokens.

We construct such a balancing network (denoted ALMOST $[2w]$) as follows. Take a STAGE $[2w]$ network and modify it by picking some x other than 0 or $w-1$ and deleting the final balancer between a_x and b_{w-1-x}. Denote this balancing network as STAGE$^x[2w]$. Let ALMOST $[2w]$ be the periodic network constructed from k stages, for some $k > 0$, each a STAGE$^x[2w]$ network, with the outputs of each stage connected to the inputs of the next.

Let A_t and B_t be the sums of the number of tokens input to each of the two subnetworks A and B in the t-th stage of ALMOST $[2w]$. A_0 and B_0 are thus the numbers of tokens input to A and B respectively. Let $y = \{y_0, \ldots, y_{2w-1}\}$ be the sequence given by $y_i = \lceil (A_0 + B_0 - i)/2w \rceil$. Thus, y_i counts the number of tokens that would exit on output wire i if ALMOST $[2k]$ were a counting network.

We now define the quantities A_∞ and B_∞ used in the proofs below. They measure the number of tokens that would have come out of the respective parts of network in the last stage ($t = \infty$) if it were a counting network. Formally, let $A_\infty = \sum_{i=0}^{w-1} y_i$, and $B_\infty = \sum_{i=w}^{2w-1} y_i$. Note that $A_t + B_t = A_0 + B_0 = A_\infty + B_\infty$ for all t and that by Lemma 2.2, $\lceil (A_\infty - i)/w \rceil = y_i$ and $\lceil (B_\infty - i)/w \rceil = y_{w+i}$ for all i.

Finally, let the *imbalance* $\delta_t = A_t - A_\infty = -(B_t - B_\infty)$; this quantity represents "how far" the network is from balancing the tokens between the

A and B subnetworks in stage t, in other words, how many excess tokens must be moved from the A part of the network to the B part (or, if the quantity is negative, how many tokens should be moved from B to A).

The following lemma follows from arguments almost identical to those of Lemma 5.4.

Lemma 7.5 *If the input sequence to a balancing network has the step property, then so does the output sequence.*

Lemma 7.6 *In the output sequence of stage t of* ALMOST $[2w]$*, each a_i is equal to $y_i + e_i$, where $e_i \leq 0$ when $\delta_t \leq 0$, and $e_i \geq 0$ when $\delta_t \geq 0$; and each b_i is equal to $y_{w+i} + e_{w+i}$, where $e_i \leq 0$ when $\delta_t \geq 0$, and $e_i \geq 0$ when $\delta_t \leq 0$.*

Proof: For $i < w$ we have

$$\begin{aligned} e_i &= a_i - y_i \\ &= \lceil (A_t - i)/w \rceil - \lceil (A_\infty - i)/w \rceil \\ &= \lceil (\delta_t + A_\infty - i)/w \rceil - \lceil (A_\infty - i)/w \rceil \end{aligned}$$

which is at least zero when $\delta \geq 0$ and at most zero when $\delta \leq 0$.

The claim for $e_{w+i} = b_i - y_{w+i}$ follows by a similar argument. ■

Corollary 7.7 *If $\delta_t = 0$ then the output sequences of stage t of* ALMOST $[2w]$ *have the step property.*

Proof: If $\delta_t = 0$ then by the preceding lemma each $a_i = y_i$ and $b_i = y_{w+i}$, so the output sequences of stage t form the sequence y. Since y has the step property it is left unchanged by the final layer of balancers (Lemma 7.5). ■

Lemma 7.8 $\delta_{t+1} = \left\lfloor \frac{\lceil (A_t - x)/w \rceil - \lceil (B_t - (w-1-x))/w \rceil}{2} \right\rfloor$.

Proof: If a balancer were placed between a'_x and b'_{w-1-x} after stage t, then the STAGE$^x[2w]$ network would become a STAGE $[2w]$ counting network, and by Corollary 7.4, exactly A_∞ tokens would emerge from the A half of the network after stage $t+1$, giving an imbalance would be 0. The above quantity δ_{t+1} is simply the number of tokens that this balancer would move from the A part of the network to the B part in order to bring the parts into balance, and is thus the actual imbalance that results from deleting the balancer. ■

The following lemmas show that the imbalance tends toward zero as more stages are added:

Lemma 7.9 *If $\delta_t \geq 0$ then $\delta_{t+1} \geq 0$. If $\delta_t \leq 0$ then $\delta_{t+1} \leq 0$.*

Proof: Suppose $\delta_t \geq 0$. Then $A_t \geq A_\infty$ and $B_t \leq B_\infty$, and so

$$\begin{aligned}\delta_{t+1} &= \left\lfloor \frac{\lceil (A_t - x)/w \rceil - \lceil (B_t - (w-1-x))/w \rceil}{2} \right\rfloor \\ &\geq \left\lfloor \frac{\lceil (A_\infty - x)/w \rceil - \lceil (B_\infty - (w-1-x))/w \rceil}{2} \right\rfloor \\ &= 0.\end{aligned}$$

(The last equality holds because when the two parts of the network hold A_∞ and B_∞ tokens there is no imbalance.)

Reversing the inequalities gives the corresponding result for $\delta_t \leq 0$. ■

Lemma 7.10 *If $|\delta_t| > 0$ then $|\delta_{t+1}| \leq |\delta_t| - 1$.*

Proof: By virtue of Lemma 7.9 we need only show that δ decreases when positive and increases when negative.

Let $a_0, \ldots, a_{w-1}, b_0, \ldots, b_{w-1}$ be the outputs of the A and B subnetworks of the $(t+1)$-th stage before the last layer of balancers. Because $\delta_t \neq 0$, this sequence does not have the step property; however, each of the two subsequences $a_0, \ldots a_{w-1}$ and $b_0, \ldots, b_{w-1}$ is the output of a counting network and so has the step property. Thus the step property of the whole sequence must be violated by some a_i, b_j such that $a_i - b_j$ is either less than 0 or greater than 1.

We will consider two cases, depending on the sign of δ_t:

Case 1. $\delta_t < 0$. Then by Lemma 7.6 each $a_i \leq y_i$ and each $b_j \geq y_{w+j}$. (Recall that y_i is the number of tokens that would exit from the i-th output of a counting network with the same input sequence.) So for each a_i and each b_j we have, using the step property of the y sequence, $a_i \leq y_i \leq y_{w+j} + 1 \leq b_j + 1$. Thus:

1. For each a_i and b_{w-1-i}, $a_i \leq b_{w-1-i} + 1$, so the balancer between these outputs moves no tokens from the A side to the B side.

2. Given some a_i and b_j that violate the step property, it cannot be the case that $a_i > b_j + 1$ and thus it must be the case that $a_i < b_j$. But then $a_{w-1} \leq a_i < b_j \leq b_0$, and since a_{w-1} and b_0 are connected by a balancer, that balancer moves at least one token from the B side to the A side.

Hence at least one token moves from the B side to the A side and $\delta_{t+1} > \delta_t$.

Case 2. $\delta_t > 0$. Then each $a_i \geq y_i$ and each $b_i \leq y_{w+i}$. So $a_i \geq y_i \geq y_{w+1} \geq b_i$. Thus:

1. For each a_i and b_{w-1-i}, $a_i \geq b_{w-1-i}$, so no final-stage balancer moves tokens from the B side to the A side.
2. Given some a_i and b_j that violate the step property, it must be the case that $a_i \geq b_j + 2$. But $a_0 \geq a_i \geq b_j + 2 \geq b_{w-1} + 2$; so the balancer between a_0 and b_{w-1} moves at least one token from the A side to the B side.

Hence at least one token moves from the A side to the B side and $\delta_{t+1} < \delta_t$.

■

Lemma 7.11 $\delta_{t+1} = \delta_t/w + c$ *where* $-3/2 \leq c < 3/2$.

Proof: From Lemma 7.8 we have:

$$\delta_{t+1} = \left\lfloor \frac{\lceil (A_t - x)/w \rceil - \lceil (B_t - (w-1-x))/w \rceil}{2} \right\rfloor$$

Looking more closely at the B_t term, notice that $\left\lceil \frac{B-(w-1-x)}{w} \right\rceil = \left\lceil \frac{B+x+1}{w} \right\rceil - 1$. If $\frac{(B+x+1)}{w}$ is not an integer then this is just $\left\lfloor \frac{B+x+1}{w} \right\rfloor$, which is equal to $\left\lfloor \frac{B+x}{w} \right\rfloor$ since subtracting 1 from the numerator cannot bring it below the next integral multiple of w. Now if $\frac{(B+x+1)}{w}$ *is* an integer then this is $\left\lfloor \frac{B+x+1}{w} \right\rfloor - 1$ which in this case is equal to $\left\lfloor \frac{B+x}{w} \right\rfloor$ since subtracting 1 from the numerator *does* bring it below an integral multiple of w. So in either case we have $\left\lceil \frac{B-(w-1-x)}{w} \right\rceil = \left\lfloor \frac{B+x}{w} \right\rfloor$, and we can rewrite the original expression as:

$$\begin{aligned}
\delta_{t+1} &= \left\lfloor \frac{\lceil (A_t - x)/w \rceil - \lfloor (B_t + x)/w \rfloor}{2} \right\rfloor \\
&= \left\lfloor \frac{(A_t - x)/w - (B_t + x)/w + c_1}{2} \right\rfloor \\
&= \frac{A_t - B_t}{2w} - \frac{x}{w} + \frac{c_1}{2} - c_2 \\
&= \frac{2\delta_t + (A_\infty - B_\infty)}{2w} - \frac{x}{w} + \frac{c_1}{2} - c_2
\end{aligned}$$

where $0 \leq c_1 < 2$ and $0 \leq c_2 < 1$. Using the fact that $0 \leq A_\infty - B_\infty \leq w$ (hence $0 \leq (A_\infty - B_\infty)/2w \leq 1/2$), and that $0 < x \leq w - 1$ (hence $1/2 < -x/w \leq 0$), we can rewrite all of the terms not containing δ as a single value c and get

$$\delta_{t+1} = \frac{\delta_t}{w} + c$$

where the bound $-3/2 < c < 3/2$ is obtained by summing the bounds on the individual terms. ∎

Theorem 7.12 *Let w be a power of 2 greater than 1. Then there exists a width-$2w$ balancing network that has the step property in all executions with up to $w^{(k-4)}$ tokens, yet is not a counting network.*

Proof: From Lemma 7.11 we have $|\delta_{t+1}| < |\delta_t|/w + 3/2$. Let $U(t)$ be defined by the recurrence $U(0) = |\delta_0|$, $U(t+1) = U(t)/w + 3/2$; then $U(t)$ is a strict upper bound on $|\delta_t|$ for $t > 0$. Solving the recurrence using standard methods yields $U(t) = |\delta_0| w^{-t} + \frac{(3/2)}{1-1/w} - \left(\frac{(3/2)}{w-1}\right) w^{-t}$.

Now suppose the network is given an input involving at most w^t tokens. Then $|\delta_0|$ cannot possibly exceed w^t, and after t stages $|\delta_t| < U(t) \leq 1 + \frac{(3/2)}{1-1/w} - \left(\frac{(3/2)}{w-1}\right) w^{-t}$, which is at most 4 if $w \geq 2$ and $t \geq 1$. So by Lemma 7.10, $|\delta_{t+4}| = 0$ and thus by Corollary 7.7 the outputs of stage $t+4$ have the step property. Thus a network with $k = t + 4$ stages will count up to $w^{(k-4)}$ tokens.

To see that this k-stage network is not a counting network, suppose $|\delta_0| > 4w^{(k+1)}$. From Lemma 7.11 we have $|\delta_{t+1}| > |\delta_t|/w - 3/2$. Let $L(t)$ be defined by $L(0) = |\delta_0|$ and $L(t+1) = L(t)/w - 2$; $L(t)$is a strict lower bound on $|\delta_t|$

for $t > 0$. Solving the recurrence gives $L(t) = |\delta_0|w^{-t} - \frac{(3/2)}{1-1/w} + \left(\frac{(3/2)}{w-1}\right) w^{-t}$. Dropping the last term and setting $|\delta_0| > 4w^{(k+1)}$ gives $|\delta_{k+1}| > L(k+1) > 4 - \frac{(3/2)}{1-1/w} \geq 1$. Since $\delta_{k+1} \neq 0$, the outputs of stage k (and hence the entire network) cannot have the step property. ∎

8 Discussion

Counting networks deserve further study. We believe that they represent a start toward a general theory of low-contention data structures. Work is needed to develop other primitives, to derive upper and lower bounds and new performance measures. We have made a start in this direction by deriving constructions and lower bounds for *linearizable* counting networks [20], networks which guarantee that the values assigned to tokens reflect the real-time order of their traversals. Aharonson and Attiya [3], Felton, LaMarca, and Ladner [11], and Hardavellas, Karakos, and Mavronicolas [17] have investigated the structure of counting networks with fan-in greater than two. Klugerman and Plaxton [23] have shown an explicit network construction of depth $O(c^{\log^* n} \log n)$ for some small constant c, and an existential proof of a network of depth $O(\log n)$.

Work is also needed in experimental directions, comparing counting networks to other techniques, for example those based on exponential backoff [1], and for understanding their behavior in architectures other than the single-bus architecture provided by the Encore. We have made a start in this direction by comparing the performance of counting networks to that of known methods using the ASIM simulator of the MIT Alewife machine [19]. Preliminary results show that there is a substantial gain in performance due to parallelism on such distributed memory machines.

Finally, we point out that smoothing networks, balancing networks that smooth but do not necessarily count, are interesting in their own right since they can be used as hardware solutions to problems such as load balancing (cf. [28]).

9 Acknowledgments

Orli Waarts made many important remarks. The serialization lemma and the observation that *smoothing + sorting = counting*, are products of our cooperation with her and with Eli Gafni, to whom we are also in debt. Our thanks to Heather Woll, and Shanghua Teng for several helpful discussions, to Cynthia Dwork for her comments, and to David Kranz and Randy Osborne for Mul-T support, and to the helpful yet anonymous referees. Finally, the first and third authors wish to thank David Michael Herlihy for remaining quiet during phone calls.

References

[1] A. Agarwal and M. Cherian. Adaptive Backoff Synchronization Techniques *16th Symposium on Computer Architecture*, June 1989.

[2] A. Agarwal et al. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. In *Proceedings of Workshop on Scalable Shared Memory Multiprocessors*. Kluwer Academic Publishers, 1991. An extended version of this paper has been submitted for publication, and appears as MIT/LCS Memo TM-454, 1991.

[3] E. Aharonson and H. Attiya. Counting Network with Arbitrary Fan-Out. In *3rd Symposium on Discrete Algorithms*, pages 104–113. ACM-SIAM, January 1992.

[4] M. Ajtai, J. Komlos and E. Szemeredi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, 1-9, 1983.

[5] T.E. Anderson. The performance implications of spin-waiting alternatives for shared-memory multiprocessors. Technical Report 89-04-03, University of Washington, Seattle, WA 98195, April 1989.

[6] J. Aspnes, M.P. Herlihy, and N. Shavit. Counting Networks and Multi-Processor Coordination In *Proceedings of the 23rd Annual Symposium on Theory of Computing*, May 1991, New Orleans, Louisiana.

[7] K.E. Batcher. Sorting networks and their applications. In *Proceedings of AFIPS Joint Computer Conference*, 32:338-334, 1968.

[8] T.H. Cormen, C.E. Leiserson, and R. L. Rivest. Introduction to Algorithms MIT Press, Cambridge MA, 1990.

[9] M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The Periodic Balanced Sorting Network *Journal of the ACM*, 36(4):738–757, October 1989.

[10] C.S. Ellis and T.J. Olson. Algorithms for parallel memory allocation. *Journal of Parallel Programming*, 17(4):303–345, August 1988.

[11] E.W. Felton, A. LaMarca, and R. Ladner. Building Counting Networks from Larger Balancers. Technical Report 93-04-09, University of Washington, Seattle, WA 98195, April 1993.

[12] E. Freudenthal and A. Gottlieb Process Coordination with Fetch-and-Increment In *Proceedings of the 4th International Conference on Architecture Support for Programming Languages and Operating Systems*, April 1991, Santa Clara, California.

[13] D. Gawlick. Processing 'hot spots' in high performance systems. In *Proceedings COMPCON'85*, 1985.

[14] J. Goodman, M. Vernon, and P. Woest. A set of efficient synchronization primitives for a large-scale shared-memory multiprocessor. In *3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.

[15] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir. The NYU ultracomputer – designing an mimd parallel computer. *IEEE Transactions on Computers*, C-32(2):175–189, February 1984.

[16] A. Gottlieb, B.D. Lubachevsky, and L. Rudolph. Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *ACM Transactions on Programming Languages and Systems*, 5(2):164–189, April 1983.

[17] N. Hardavellas, D. Karakos, and M. Mavronicolas. Notes on Sorting and Counting Networks. in *Proceedings of WDAG'93*, to appear.

[18] D. Hensgen and R.Finkel and U. Manber. Two algorithms for barrier synchronization. *International Journal of Parallel Programming*, 17(1):1-17, 1988.

[19] M.P. Herlihy, B.H. Lim, and N. Shavit. Low Contention Load Balancing on Large Scale Multiprocessors. In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1992, pp. 219–227.

[20] M.P. Herlihy, N. Shavit, and O. Waarts. Low-Contention Linearizable Counting. In *32th IEEE Symposium on Foundations of Computer Science*, October 1991, pp. 526–535.

[21] D. Kranz, R. Halstead, and E. Mohr. "Mul-T, A High-Performance Parallel Lisp", *ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, Portland, OR, June 1989, pp. 81–90.

[22] C.P. Kruskal, L. Rudolph, and M. Snir. Efficient synchronization on multiprocessors with shared memory. In *Fifth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1986.

[23] M. Klugerman and C.G. Plaxton. Small-depth Counting Networks. In *ACM Symposium on the Theory of Computing*???.

[24] N.A. Lynch and M.R. Tuttle. Hierarchical Correctness Proofs for Distributed Algorithms. In *Sixth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1987, pp. 137–151. Full version available as MIT Technical Report MIT/LCS/TR–387.

[25] J.M. Mellor-Crummey and M.L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. Technical Report Technical Report 342, University of Rochester, Rochester, NY 14627, April 1990.

[26] L. Rudolph, Decentralized cache scheme for an MIMD parallel processor. In *11th Annual Computing Architecture Conference,* 1983, pp. 340-347.

[27] J.M. Mellor-Crummey and M.L. Scott Synchronization without Contention In *Proceedings of the 4th International Conference on Architecture Support for Programming Languages and Operating Systems*, April 1991, Santa Clara, California. ???

[28] D. Peleg and E. Upfal. The token distribution problem. In *27th IEEE Symposium on Foundations of Computer Science*, October 1986.

[29] G.H. Pfister et al. The IBM research parallel processor prototype (RP3): introduction and architecture. In *International Conference on Parallel Processing*, 1985.

[30] G.H. Pfister and A. Norton. 'hot spot' contention and combining in multistage interconnection networks. *IEEE Transactions on Computers*, C-34(11):933–938, November 1985.

[31] H.S. Stone. Database applications of the fetch-and-add instruction. *IEEE Transactions on Computers*, C-33(7):604–612, July 1984.

[32] U. Vishkin. A parallel-design distributed-implementation (PDDI) general purpose computer. *Theoretical Computer Science*, 32:157–172, 1984.