Building Counting Networks from Larger Balancers *

Technical Report #93-04-09

Edward W. Felten Anthony LaMarca Richard Ladner

Dept. of Computer Science and Engineering
University of Washington
Seattle, WA 98195
U.S.A.

April 30, 1993

Abstract

We introduce a generalization of the counting networks of Aspnes, Herlihy, and Shavit [AHS91]. Our counting networks are constructed using k-balancers, rather than the 2-balancers of Aspnes *et al.* For reasonable values of k, k-balancers and 2-balancers can be implemented with equal efficiency on existing computers. Our *k-bitonic* networks have depths ranging from $O(1)$ to $O(\log^2 w)$, where w is the number of inputs to the network. The depth of our networks varies with the choice of k; choosing the optimal k depends on a tradeoff between the desire for a shallow network and the desire for low contention.

We present the k-bitonic construction, prove its correctness, and introduce some useful variations to the construction. We then compare the performance of our networks with the networks of Aspnes *et al.*, using simulation of idealized counting networks. Our networks perform at least as well in all cases, and typically have throughput about 25% higher than the networks of Aspnes *et al.*

1 Introduction

Recent years have seen the introduction of shared-memory parallel computers with both larger numbers of processors and relatively longer latencies to access memory. Algorithms for multiple processor coordination are especially important on machines with these characteristics, and as a result interest in coordination algorithms has increased.

*This material is based upon work supported by the National Science Foundation (Grants CCR-8619663, CCR-9108314, CCR-9123308, and CCR-9200832), the Washington Technology Center, Digital Equipment Corporation (the External Research Program and the Systems Research Center), and Apple Computer Company. Felten was supported by an AT&T Ph.D. Scholarship and a Mercury Seven Fellowship. LaMarca was supported by an AT&T Ph.D. Scholarship. Part of the research was done while Ladner was at Victoria University of Wellington, New Zealand.

The usual approach to coordination is to serialize access to shared data by using a lock. While much work has been done to optimize the performance of locking [And90, MCS91], these techniques still suffer from memory contention, since the processors are forced to serialize through a single memory location. The effect of contention grows as the number of processors increases, and as memory latencies become relatively longer.

The cost of contention has motivated the search for low-contention coordination algorithms, which require relatively few processors to access any one memory location. Counting networks, which were originally defined by Aspnes, Herlihy, and Shavit [AHS91], are one such mechanism. Counting networks are acyclic networks of shared objects called *balancers*. Counting networks can be used to build shared counters, producer/consumer buffers and distributed queues, none of which require synchronization on a single shared location. Due to their low contention, counting networks offer a potential advantage in performance over traditional synchronization techniques.

1.1 Balancers and Counting Networks

A counting network is a network of objects called *balancers*. Abstractly, the balancers cooperate to route *tokens* through the network. In practice, a counting network is a linked data structure made up of data objects representing balancers, and the tokens are represented by processors traversing the linked structure.

A balancer behaves like a two-input, two-output toggle. Tokens arrive arbitrarily on the balancer's two input wires and the balancer outputs tokens alternately on the first and then the second output wire. Let X_0 and X_1 denote the input wires and Y_0 and Y_1 denote the output wires. If i tokens have passed through a balancer, $\lceil \frac{i}{2} \rceil$ tokens have left the balancer on output wire Y_0 and $\lfloor \frac{i}{2} \rfloor$ have left on output wire Y_1.

A *balancing network of width* w is formed from a group of balancers by connecting the outputs of some balancers to the inputs of other balancers in an acyclic fashion. There are exactly w balancer input wires which remain unconnected. These w input wires are the input wires of the balancing network. Similarly, there are exactly w unconnected balancer output wires which form the output wires of the balancing network. Let x_i be the number of tokens that have entered on the counting network's ith input wire and let y_i be the number of tokens that have exited on the counting network's ith output wire. We say a balancing network is *quiescent* if no tokens are in the network. A *counting network* is a balancing network which has the *step property*. The step property says that in any quiescent state, $0 \leq y_i - y_j \leq 1$ for any $0 \leq i \leq j < w$.

Apnes, Herlihy and Shavit presented the bitonic counting network construction [AHS91] based on Batcher's bitonic sorting network [Bat68]. A bitonic network with w input wires has depth $O(\log^2 w)$ [1] ; that is, a token must pass through $O(\log^2 w)$ balancers to traverse the network. Recently Klugerman has presented alternative constructions with depth $O(\log w \log\log w)$ and $O(C^{\log^* w} \log w)$ [Klu91, KP92]. While these new constructions represent an asymptotic improvement in counting network depth, the constants are too large

[1] All logarithms are base 2 unless otherwise specified

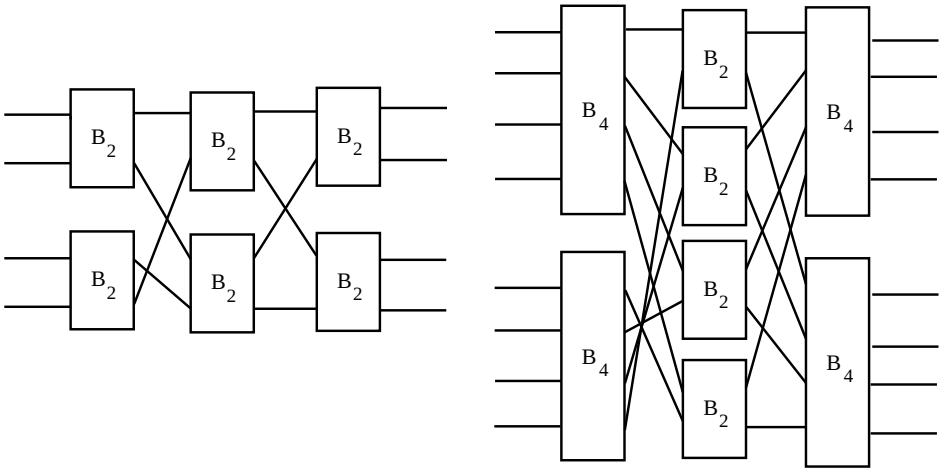

Figure 1: The bitonic network of width 4, and the k-bitonic network $C_4(8)$. B_k denotes a k-balancer.

---

to make these networks practical. The original bitonic construction still performs best in all practical situations.

The focus of our work has been to improve the depth of counting networks, while retaining their practical viability. We begin by presenting *k-balancers*, a generalization of the original balancers. Intuitively, a k-balancer behaves like a k input, k output toggle. On existing machines, k-balancers are no more expensive in time or space than the traditional 2-balancers, unless k is so large that it cannot be represented in a single machine word.

We then present a construction for building counting networks from k-balancers. Our k-bitonic construction is a generalization of the original bitonic construction. This new construction allows us to trade off depth for contention, independent of the width of the network. Depending on the choice of k, our networks have depth ranging from $O(1)$ to $O(\log^2 w)$. Large values of k yield shallow networks with high contention; small values of k yield deep networks with low contention. Optimally balancing these two factors yields the network with the highest throughput.

In addition, we use simulation to explore the effects of width, depth and contention on the throughput of counting networks. Our simulations show that the extra degree of freedom available in the k-bitonic construction provides approximately a 25% performance improvement over the bitonic construction, for simulations of several hundred processors.

2 K-Balancers

A k-balancer is a device with k input wires and k output wires. Intuitively, a k-balancer behaves like a k-input, k-output toggle. Tokens arrive at arbitrary input wires and the

k-balancer outputs the tokens on the output wires in a cyclic fashion from the first output to the kth. Let x_i denote the number of tokens that have arrived on the k-balancer's i^{th} input wire, and y_i denote the number of tokens output on the k-balancer's i^{th} output wire. The formal properties of k-balancers are:

1. In any state $\sum_{i=0}^{k-1} x_i \geq \sum_{i=0}^{k-1} y_i$. (A k-balancer never creates tokens.)
2. Given that any finite number of tokens $\sum_{i=0}^{k-1} x_i$ have arrived at the k-balancer, the k-balancer reaches a *quiescent* state (a state in which $\sum_{i=0}^{k-1} x_i = \sum_{i=0}^{k-1} y_i$) within a finite amount of time. (A k-balancer never consumes tokens.)
3. In any quiescent state, $0 \leq y_i - y_j \leq 1$ for any $0 \leq i \leq j < k$ (The output has the step property.)

In actual implementations of balancers, the toggle state is represented with a word of memory and the output wires are represented with pointers. Balancers can be manipulated in two different ways. If the architecture being used provides synchronization primatives such as *Fetch&Increment* or *Compare&Swap*, these instructions can be used to directly manipulate the balancer's state. For example, one can traverse a balancer by doing a *Fetch&Increment* on the balancer's toggle memory, and exiting on the wire corresponding to the fetched value modulo 2. In the absence of such powerful primitives, the balancer must contain a lock, which processors hold while manipulating the balancer's state.

In theory, the original balancer requires only a single bit of state to be toggled, while the k-balancer requires a $\log k$ bit operation. Actual machines, however, operate on entire words in a single instruction. Thus, if our machine has b bit words, the original balancer requires one operation to toggle the state, while the k-balancer requires $\lceil \frac{\log k}{b} \rceil$ operations to toggle the state. Therefore, if $k < 2^b$, our k-balancer requires no more operations than the original balancer. On a typical 32-bit machine, k can be as large as 2^{32}, without losing any performance over traditional balancers.

It should be noted that we are not the first to generalize the balancer. Ahoronson and Attiya use the *b-balancer* with a finite number of inputs and b outputs [AA92]. However, they do not use these larger balancers to construct practical networks. They present an impossibility result showing that networks of certain sizes cannot be built with some sizes of balancers. They also present a construction for counting networks of width $p2^i$, but since the resulting networks have large depth, their construction serves primarily as an existence proof rather than a practical tool. Their networks have depth $O(log^3 w)$, or alternatively $O(log^2 w)$ with a very large constant if the AKS sorting network [AKS83] is used in the construction. Finally, they show how to build counting networks of arbitrary width by adding cycles to the network.

3 The K-bitonic Counting Network Construction

We now present a construction for building counting networks of width 2^i from k-balancers for $k = 2^j$, $i, j \geq 1$. We call our construction the k-bitonic construction since it is a gen-

eralization of the original bitonic counting network construction [AHS91]. The k-bitonic construction attempts to build the network exclusively from balancers of size k. Our construction, however, is recursive and at times the subproblems are too small to be built from balancers of size k. In these situations, the construction uses balancers as close to size k as possible.

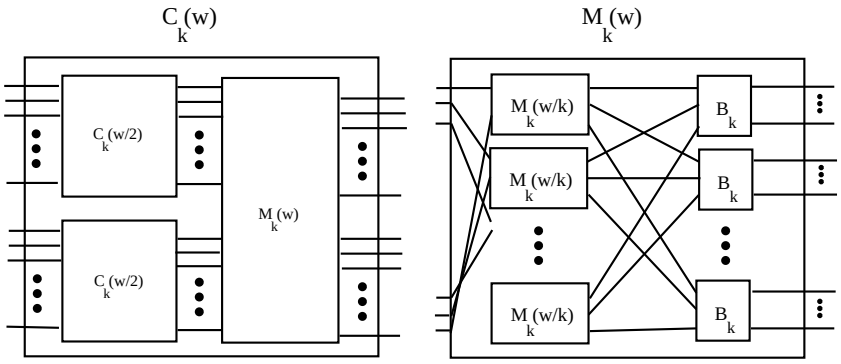

Figure 2: The k-bitonic construction: building a $C_k(w)$ and a $M_k(w)$ from smaller parts. B_k denotes a k-balancer.

Let $C_k(w)$ be a k-bitonic counting network of width w. $C_k(w)$ has w inputs $x_0, x_1, ..., x_{w-1}$ and w outputs $y_0, y_1, ..., y_{w-1}$. $C_k(w)$ is constructed so that in a quiescent state, the output y has the step property. In our construction, counting networks are built recursively from smaller counting networks and merging networks.

Merging networks are balancing networks which guarantee that if both halves of their inputs have the step property, the output has the step property. Let $M_k(w)$ be a merging network of width w. $M_k(w)$ has w inputs and w outputs. Let $x_0, x_1, ..., x_{\frac{w}{2}-1}$ be the first half of the merger's inputs and let $x'_0, x'_1, ..., x'_{\frac{w}{2}-1}$ be the second half of the merger's inputs. Let $y_0, y_1, ..., y_{w-1}$ be the merger's outputs. $M_k(w)$ is constructed so that in a quiescent state in which the inputs x and x' each have the step property, the output y is guaranteed to have the step property.

3.1 The Merging Network

The base case for the merger construction $M_k(w)$ is when $w \leq k$. In this case, our merger $M_k(w)$ is simply a w-balancer.

If $w > k$, we build the $M_k(w)$ merging network from k $M_k(\frac{w}{k})$ merging networks and $\frac{w}{k}$ k-balancers. For clarity, we refer to the recursively-built mergers as "sub-mergers." In order for these k sub-mergers to function properly, we must ensure that both halves of their inputs have the step property. The merger's inputs serve as inputs to the sub-mergers according to the following rules:

- Merger input wire x_i serves as input wire $x_{\lfloor \frac{i}{k} \rfloor}$ of sub-merger $i \bmod k$.
- Merger input wire x'_i serves as input wire $x'_{\lfloor \frac{i}{k} \rfloor}$ of sub-merger $(k-1)-(i \bmod k)$.

This input wire distribution guarantees that if the two halves of the merger's input have the step property, the halves of the sub-merger's inputs will also have the step property. The remaining task is to merge the sub-mergers' outputs into a sequence of length w with the step property. This is achieved with a set of $\frac{w}{k}$ k-balancers. Formally:

- The output wire y_i of sub-merger j is connected to input wire x_j of k-balancer i.
- The output wire y_j of k-balancer i serves as output wire y_{ki+j} of the $M_k(w)$.

Figure 2 provides a graphic illustration of the construction.

3.2 The Counting Network

The construction for $C_k(w)$ is simpler than that of the merger. In the base case that $w \leq k$, $C_k(w)$ is a w-balancer. If $w > k$, our $C_k(w)$ is built from 2 sub-counting networks $C_k(\frac{w}{2})$ and a merging network $M_k(w)$.

The components are connected in a divide and conquer fashion. The counting network's inputs are divided in two and sent through the sub-counters. The output of the sub-counters are then sent to the merger for recombination. Formally:

- The counting network inputs $x_0, x_1, ..., x_{\frac{w}{2}-1}$ serve as inputs $x_0, x_1, ..., x_{\frac{w}{2}-1}$ of sub-counter 0, and the counting network inputs $x_{\frac{w}{2}}, x_{\frac{w}{2}+1}, ..., x_{w-1}$ serve as inputs $x_0, x_1, ..., x_{\frac{w}{2}-1}$ of sub-counter 1.
- The outputs $y_0, y_1, ..., y_{\frac{w}{2}-1}$ of sub-counter 0 serve as merger inputs $x_0, x_1, ..., x_{\frac{w}{2}-1}$, and the outputs $y_0, y_1, ..., y_{\frac{w}{2}-1}$ of sub-counter 1 serve as merger inputs $x'_0, x'_1, ..., x'_{\frac{w}{2}-1}$.
- The merger outputs $y_0, y_1, ..., y_{w-1}$ serve as outputs $y_0, y_1, ..., y_{w-1}$ of the counting network.

Figure 2 provides a graphic illustration of the construction.

3.3 Proof of Correctness

We now show that $C_k(w)$ is a counting network. We begin by giving a useful equivalent form of the step property. Recall that a sequence $x_0, x_1, x_2, ..., x_{w-1}$ of non-negative integers has the step property if $0 \leq x_i - x_j \leq 1$ whenever $i < j$.

Lemma 3.1 *A sequence $x_0, x_1, x_2, ..., x_{w-1}$ of non-negative integers satisfies the step property if and only if*

$$x_i = \lfloor \frac{X + w - 1 - i}{w} \rfloor$$

for $0 \leq i < w$, where $X = \sum_{k=0}^{w-1} x_k$.

Proof: Let $x_0, x_1, x_2, ..., x_{w-1}$ satisfy the step property and let $X = \sum_{k=0}^{w-1} x_k$. If $x_i = x_{i+1}$ for $0 \leq i < w-1$ then $X = wx_i$ for $0 \leq i < w$. Thus,

$$\lfloor \frac{X+w-1-i}{w} \rfloor = \lfloor \frac{wx_i+w-1-i}{w} \rfloor = x_i + \lfloor \frac{w-1-i}{w} \rfloor = x_i.$$

Suppose $x_i \neq x_{i+1}$ for some i where $0 \leq i < w-1$. By the step property, choose c such that $x_c - x_{c+1} = 1$. By the step property, it must be the case that if $i \leq c$ then $x_i = x_c$ and if $i > c$ then $x_i = x_{c+1} = x_c - 1$. Thus,

$$X = \sum_{k=0}^{w-1} x_k = (c+1)x_c + (w-c-1)(x_c-1) = w(x_c-1) + c + 1.$$

We conclude that

$$\lfloor \frac{X+w-1-i}{w} \rfloor = \lfloor \frac{wx_c+c-i}{w} \rfloor = x_c + \lfloor \frac{c-i}{w} \rfloor.$$

This quantity equals x_c if $i \leq c$, and $x_c - 1$ if $i > c$. In any case, this quantity equals x_i.

To complete the proof, assume $x_i = \lfloor \frac{X+w-1-i}{w} \rfloor$. Let $i < j$, then

$$0 \leq \lfloor \frac{X+w-1-i}{w} \rfloor - \lfloor \frac{X+w-1-j}{w} \rfloor = x_i - x_j.$$

Furthermore, we have

$$\begin{aligned} x_i - x_j &= \lfloor \tfrac{X+w-1-i}{w} \rfloor - \lfloor \tfrac{X+w-1-j}{w} \rfloor \\ &\leq \lfloor \tfrac{X+w-1-i}{w} - \tfrac{X+w-1-j}{w} \rfloor + 1 \\ &= \lfloor \tfrac{j-i}{w} \rfloor + 1 \\ &= 1 \end{aligned}$$

The inequality follows from the general fact that $\lfloor x \rfloor - \lfloor y \rfloor \leq \lfloor x-y \rfloor + 1$. The final equality follows from the fact that $0 \leq j - i < w$. Thus, we have $0 \leq x_i - x_j \leq 1$ whenever $i < j$.

The most difficult part of the proof that $C_k(w)$ is a counting network is demonstrating that the merging network $M_k(w)$ behaves correctly. The correctness of $M_k(w)$ is shown in the following theorem.

Theorem 3.1 *If the merging network $M_k(w)$ is quiescent and its inputs $x_0, x_1, ..., x_{\frac{w}{2}-1}$ and $x'_0, x'_1, ..., x'_{\frac{w}{2}-1}$ both have the step property, then its outputs $y_0, y_1, ..., y_{w-1}$ have the step property.*

Proof: The theorem is proved by induction on w.

Base Case: $w \leq k$.

In the base case where $w \leq k$, the merger $M_k(w)$ is simply a w-balancer. By the definition of a k-balancer, the output will have the step property.

Inductive Step: Assume the theorem holds for $w' < w$ and that $w > k$.

Let x, x', and y be the sequences $x_0, x_1, ..., x_{\frac{w}{2}-1}$, $x'_0, x'_1, ..., x'_{\frac{w}{2}-1}$ and $y_0, y_1, ..., y_{w-1}$, respectively. Assume x and x' have the step property. In the construction of the merger $M_k(w)$ the top half of the inputs of any one of the submergers $M_k(w/k)$ is a subsequence of x and the bottom half of the inputs is a subsequence of x'. Since a subsequence of any sequence with the step property also has the step property,the top and bottom halves of the inputs to each submerger have the step property. For $0 \leq a < k$, let z^a be the output sequence of the a-th submerger with z^a_i the output of the i-th output wire. By the induction hypothesis, for each a, z^a has the step property.

Let $X = \sum_{i=0}^{\frac{w}{2}-1} x_i$, $X' = \sum_{i=0}^{\frac{w}{2}-1} x'_i$, and $Z^a = \sum_{i=0}^{\frac{w}{k}-1} z^a_i$ for $0 \leq a < k$. By the constuction, the total contribution of the input x to submerger a is $\lfloor \frac{X+k-1-a}{k} \rfloor$. Similarly, the total contribution of the input x' to submerger a is $\lfloor \frac{X'+a}{k} \rfloor$. The difference in the contributions of x and x' comes from the fact that the inputs from x enter in the order of the submergers, while the inputs from x' enter in the reverse order of the submergers. Thus,

$$Z^a = \lfloor \frac{X+k-1-a}{k} \rfloor + \lfloor \frac{X'+a}{k} \rfloor.$$

We next show that for $0 \leq a, b < k$, $-1 \leq Z^a - Z^b \leq 1$. Without loss of generality, assume $0 \leq a < b < k$.

$$Z^a - Z^b = \lfloor \frac{X+k-1-a}{k} \rfloor - \lfloor \frac{X+k-1-b}{k} \rfloor + \lfloor \frac{X'+a}{k} \rfloor - \lfloor \frac{X'+b}{k} \rfloor$$

Since $a < b < k$, $0 \leq \lfloor \frac{X+k-1-a}{k} \rfloor - \lfloor \frac{X+k-1-b}{k} \rfloor \leq 1$ and $-1 \leq \lfloor \frac{X'+a}{k} \rfloor - \lfloor \frac{X'_i+b}{k} \rfloor \leq 0$. Thus, $-1 \leq Z^a - Z^b \leq 1$.

Let $e = \min\{Z^a \bmod w/k : 0 \leq a < k\}$. We will show that for $0 \leq a, b < k$ and for $i \neq e$, $z^a_i = z^b_i$. If $Z^a = Z^b$, then by lemma 3.1 $z^a_i = z^b_i$ for all i. If $Z^a \neq Z^b$, then, since for all c and d, $-1 \leq Z^c - Z^d \leq 1$, it must be the case that one of $Z^a \bmod w/k$ or $Z^b \bmod w/k$ equals e and the other is $e+1$. Without loss of generality, assume $Z^b \bmod w/k = e$ and $Z^a \bmod w/k = e+1$. Thus, for some Q, $Z^b = Qw/k + e$ and $Z^a = Qw/k + e + 1$. By lemma 3.1

$$\begin{aligned} z^a_i - z^b_i &= \lfloor \frac{Z^a + w/k - 1 - i}{w/k} \rfloor - \lfloor \frac{Z^b + w/k - 1 - i}{w/k} \rfloor \\ &= \lfloor \frac{Qw/k + e + w/k - i}{w/k} \rfloor - \lfloor \frac{Qw/k + e + w/k - 1 - i}{w/k} \rfloor \\ &= \lfloor \frac{e-i}{w/k} \rfloor - \lfloor \frac{e-1-i}{w/k} \rfloor \end{aligned}$$

Thus, $z^a_i = z^b_i$ for $i \neq e$.

In the construction there are exactly w/k distinct k-balancers. The i-th k-balancer receives the i-th input from each of the submergers. Let S^i be the total of the inputs to k-balancer i, that is, $S^i = \sum_{a=0}^{k-1} z^a_i$. Hence, $S^i = kz^0_i$ for all $i \neq e$. Furthermore, if $i < j$ then because each sequence z^a, for $0 \leq a < k$, satisfies the step property we must have $0 \leq S^i - S^j \leq k$.

Let $ik + i' < jk + j'$ where $0 \leq i, j < w/k$ and $0 \leq i', j' < k$. To complete the proof we show that $0 \leq y_{ik+i'} - y_{jk+j'} \leq 1$. There are four cases to consider.

Case 1: $i = j$. In this case both outputs $y_{ik+i'}$ and $y_{jk+j'}$ are outputs of the same k-balancer. Hence, $0 \leq y_{ik+i'} - y_{jk+j'} \leq 1$.
Case 2: $i \neq j$, $i \neq e$, and $j \neq e$. We have:

$$\begin{aligned} y_{ik+i'} - y_{jk+j'} &= \lfloor \tfrac{S^i+k-1-i'}{k} \rfloor - \lfloor \tfrac{S^j+k-1-j'}{k} \rfloor \\ &= \lfloor \tfrac{kz_i^0+k-1-i'}{k} \rfloor - \lfloor \tfrac{kz_j^0+k-1-j'}{k} \rfloor \\ &= z_i^0 - z_j^0 + \lfloor \tfrac{k-1-i'}{k} \rfloor - \lfloor \tfrac{k-1-j'}{k} \rfloor \\ &= z_i^0 - z_j^0. \end{aligned}$$

Since the sequence z^0 has the step property, then $0 \leq y_{ik+i'} - y_{jk+j'} \leq 1$.
Case 3: $i = e$ and $j \neq e$. Let $S^e = S^j + m$ where $0 \leq m \leq k$. We have $S^j = kz_j^0$ and $S^e = kz_j^0 + m$. Thus,

$$\begin{aligned} y_{ek+i'} - y_{jk+j'} &= \lfloor \tfrac{S^e+k-1-i'}{k} \rfloor - \lfloor \tfrac{S^j+k-1-j'}{k} \rfloor \\ &= \lfloor \tfrac{kz_j^0+m+k-1-i'}{k} \rfloor - \lfloor \tfrac{kz_j^0+k-1-j'}{k} \rfloor \\ &= \lfloor \tfrac{m+k-1-i'}{k} \rfloor - \lfloor \tfrac{k-1-j'}{k} \rfloor \\ &= \lfloor \tfrac{m+k-1-i'}{k} \rfloor \end{aligned}$$

Since, $0 \leq m \leq k$ and $0 \leq i' < k$ then $0 \leq y_{ek+i'} - y_{jk+j'} \leq 1$.
Case 4: $i \neq e$ and $j = e$. This case is proved in a similar way to case 3 above.

Theorem 3.2 *In any quiescent state the output of the counting network $C_k(w)$ has the step property.*

Proof: The theorem is proved by induction on w.
Base Case: $w \leq k$.

In the base case where $w \leq k$, the counter $C_k(w)$ is simply a w-balancer. By the definition of a k-balancer, we know the output will have the step property.
Inductive Step: Assume the theorem holds for $w' < w$ and that $w > k$.

By the inductive hypothesis, both of the sub-counters produce a sequence which has the step property. The output of the first sub-counter serves as the first half of the input wires to the merger. The output of the second sub-counter serves as the second half of the input wires to the merger. Since the two halves of the merger's inputs have the step property, by theorem 3.1, the merger's output has the step property. Since the merger's outputs serve as the outputs for the counter, the counter's output has the step property.

3.4 Size and Depth

The size of a balancing network is defined to be the number of balancers contained in the network. The depth of a balancing network is defined to be the length of the longest path a token can follow through the network.

We know the depth of our merger to be equal to the depth of the sub-merger plus one balancer. We know the depth of our counter to be the depth of a sub-counter plus the depth

of the merger. (Recall that w and k are assumed to be positive powers of two.) This yields the following recurrences:

$$\mathrm{Depth}(M_k(w)) = \begin{cases} 1 & \text{if } w \le k \\ \mathrm{Depth}(M_k(\frac{w}{k})) + 1 & \text{otherwise} \end{cases}$$

$$\mathrm{Depth}(C_k(w)) = \begin{cases} 1 & \text{if } w \le k \\ \mathrm{Depth}(C_k(\frac{w}{2})) + \mathrm{Depth}(M_k(w)) & \text{otherwise} \end{cases}$$

Solving these recurrences, we find for all w and k

$$\mathrm{Depth}(M_k(w)) = \lceil \log_k w \rceil,$$

and for $w \ge k$,

$$\frac{\log^2 w}{2\log k} + \frac{\log w}{2\log k} + \frac{1}{2}\log\frac{2}{k} \le \mathrm{Depth}(C_k(w)) \le \frac{\log^2 w}{2\log k} + \frac{\log w}{2\log k} + \frac{1}{2}\log\frac{2}{k} + \log\frac{w}{k}.$$

The size of our merger is equal to the size of the k sub-mergers plus the $\frac{w}{k}$ balancers. The size of the counter is equal to the size of the two sub-counters plus the size of the merger. This yields the following two recurrences:

$$\mathrm{Size}(M_k(w)) = \begin{cases} 1 & \text{if } w \le k \\ k \cdot \mathrm{Size}(M_k(\frac{w}{k})) + \frac{w}{k} & \text{otherwise} \end{cases}$$

$$\mathrm{Size}(C_k(w)) = \begin{cases} 1 & \text{if } w \le k \\ 2 \cdot \mathrm{Size}(C_k(\frac{w}{2})) + \mathrm{Size}(M_k(w)) & \text{otherwise} \end{cases}$$

which yield these upper bounds: for all w and k,

$$\mathrm{Size}(M_k(w)) = \frac{w}{k}(\lceil \log_k w \rceil - 1) + k^{\lceil \log_k w \rceil - 1},$$

and for $w \ge k$,

$$\frac{w\log^2 w}{2k\log k} + \frac{w\log w}{2k\log k} + \frac{w}{2k}\log\frac{2}{k} \le \mathrm{Size}(C_k(w)) \le \frac{w\log^2 w}{2k\log k} + \frac{w\log w}{2k\log k} + \frac{w}{2k}\log\frac{2}{k} + w\log\frac{w}{k}.$$

We see that the size and depth of our k-bitonic counting networks depend on the choice of k. Note that the size of a counting network is not necessarily proportional to the amount of memory space required to represent it. A k-balancer requires $\Omega(k)$ space, since it contains one pointer for each of its k outputs. Thus, the space required by a counting network is proportional to the product of k and the size. Since the number of wires connecting each level of the counting network to the next level is exactly the width w, the space required for the counting network is also proportional to the product of the width and the depth.

Figure 3 shows upper bounds on the size and depth of our counting networks for a few likely choices of k.

k	*Size*	*Depth*
2	$\frac{1}{4}w\log^2 w + \frac{1}{4}w\log w$	$\frac{1}{2}\log^2 w + \frac{1}{2}\log w$
4	$\frac{1}{16}w\log^2 w + O(w\log w)$	$\frac{1}{4}\log^2 w + O(\log w)$
$\sqrt{w}$	$w + \frac{1}{2}\sqrt{w}\log w$	$\log w + 1$
w^ϵ	$(1-\epsilon)w\log w + O(w^{1-\epsilon}\log w)$	$\frac{1+2\epsilon-3\epsilon^2}{2\epsilon}\log w + O(1)$
w	1	1

Figure 3: Summary of the size and depth of k-bitonic counting networks of width w, for various values of k. The bounds for $k = 2, w^{\frac{1}{2}}$, and w are exact, while the remaining are upper bounds. The formulas for $k = w^\epsilon$ holds for constant ϵ where $0 < \epsilon < 1/2$.

4 Variations

The construction we have presented is the straightforward generalization of the original bitonic construction. We have developed a number of variations to our k-bitonic counting network construction which result in more interesting and/or efficient networks.

4.1 Using Balancers of Arbitrary Size

As presented, our construction allows for counting networks of size 2^i to be built from balancers of size 2^j. We now relax this restriction, and use our construction to build networks from balancers of arbitrary size, with a single restriction. Our construction can be used to build counting networks of size $p2^i$ from balancers of size p, for any $p \geq 2$. This allows us more choices when tuning the balancer size k.

With this change, however, the recursive construction may need to build $M_a(b)$ where a does not evenly divide b. $M_3(12)$, for example, will attempt to build $M_3(4)$ in the recursive step. In these cases, we build $M_a(b)$ as an $M_{a'}(b)$ where a' is the largest number such that $a' \leq a$ and a' evenly divides b.

4.2 Layer Sorting

The merger's recursive construction results in scatter/gather token traffic patterns. The tokens are first scattered to the sub-mergers and are then gathered into the back-row balancers. Under light load, this token pattern makes contention more likely at the back-row balancers. Since at most one token can pass through a balancer at the same time, the use of smaller balancers in the back row will reduce contention.

Consider the merger $M_4(8)$, which forms the last two layers of the $C_4(8)$ shown in figure 1. The merger is built from two layers of balancers. The tokens are first scattered into a layer of 2-balancers, and then gathered into a layer of 4-balancers. By our previous observation, we can reduce contention by instead scattering into a layer of 4-balancers and

gathering into a layer of 2-balancers. While this change lessens the congestion in the merger, it does not affect the correctness, size or depth of the merger.

The performance of our counting networks can be improved by always applying this technique within our merger construction. Our construction is unconcerned about how the sub-mergers are built and always builds $M_k(w)$ from $\frac{w}{k}$ k-balancers and k sub-mergers of width $\frac{w}{k}$. Our first change is to have the merger obtain some information about the size of the layers to be built by the sub-mergers. We define $s_k(w)$ to be the smallest balancer size that will be needed in the construction of $M_k(w)$. For our construction $s_k(w) = w$ if $w \leq k$, otherwise $s_k(w) = s_k(\frac{w}{k})$. We now change the merger construction to build $M_k(w)$ from $\frac{w}{s_k(w)}$ $s_k(w)$-balancers and $s_k(w)$ sub-mergers of width $\frac{w}{s_k(w)}$. This change results in the construction always using the smallest remaining balancers in the back row. After the construction has finished, the resulting merger will have its layers sorted in order of ascending balancer size from back to front.

4.3 The Pyramid Construction

While our k-bitonic construction can use different size balancers in different layers, it tries to build as much of the network as it can from balancers of size k. From our second variant, we learned that under light loads the scatter/gather wire pattern results in congestion that can be reduced by using smaller balancers in the back of the network. The pyramid construction uses small balancers near the back of the network, and progressively larger balancers toward the front of the network.

In order to reduce this contention, we propose a modification to the construction in which we use small balancers near the back of the network, and larger balancers toward the front. This is achieved by changing the subnetworks that are recursively built. In our construction, $C_k(w)$ recursively builds two $C_k(\frac{w}{2})$ sub-counters. We now change $C_k(w)$ so that it instead builds two $C_{2k}(\frac{w}{2})$ sub-counters. Similarly, we change $M_k(w)$ so that it builds k $M_{2k}(\frac{w}{k})$ sub-mergers. The resulting networks have succesive layers that double in size. We call this the pyramid construction. While our previous variations have not changed the size and depth of the network, this variant does. Applying this variant, our counting networks have $depth \leq 2 \log wk \log \frac{\log w}{\log k} + \log k$.

Our choice to double the network layers is arbitrary. Many functions could be used and all would have different space/depth/performance tradeoffs. Both our original construction and our pyramid construction are examples of a larger class of heterogeneous counting networks that can be generalized from our k-bitonic construction.

5 Performance

Our performance results were obtained by simulation. We chose to simulate our counting networks for a number of reasons. First, an analytical solution proved to be too complex. Second, for an implementation study we did not have access to a machine with as many processors as we wanted. Lastly, we were interested in the performance of counting networks

in general, rather than counting networks on a specific machine. By simulating an abstract architecture, we hoped to obtain results of more universal value.

In our simulations, a fixed number of processors repeatedly traverse the counting network. When a processor exits the network, it immediately reenters. A balancer services one processor at a time; the service time is random with exponential distribution. We start with all processors entering the network on randomly chosen wires. Before collecting statistics, we simulate the network long enough for it to reach an equilibrium state. We then measure the network's throughput: the number of tokens per unit time that exit the network. Note that maximizing the throughput is equivalent to minimizing the average time it takes for a processor to traverse the network.

In our simulations, activity at one balancer does not affect the performance of other balancers; as a result, our performance graphs look different than those generated by Aspnes *et al.*, in which accesses to all balancers are made across a single memory bus.

For bitonic networks, the width w is the only parameter that can be adjusted to optimize performance. In k-bitonic networks, the balancer size k is an additional degree of freedom.

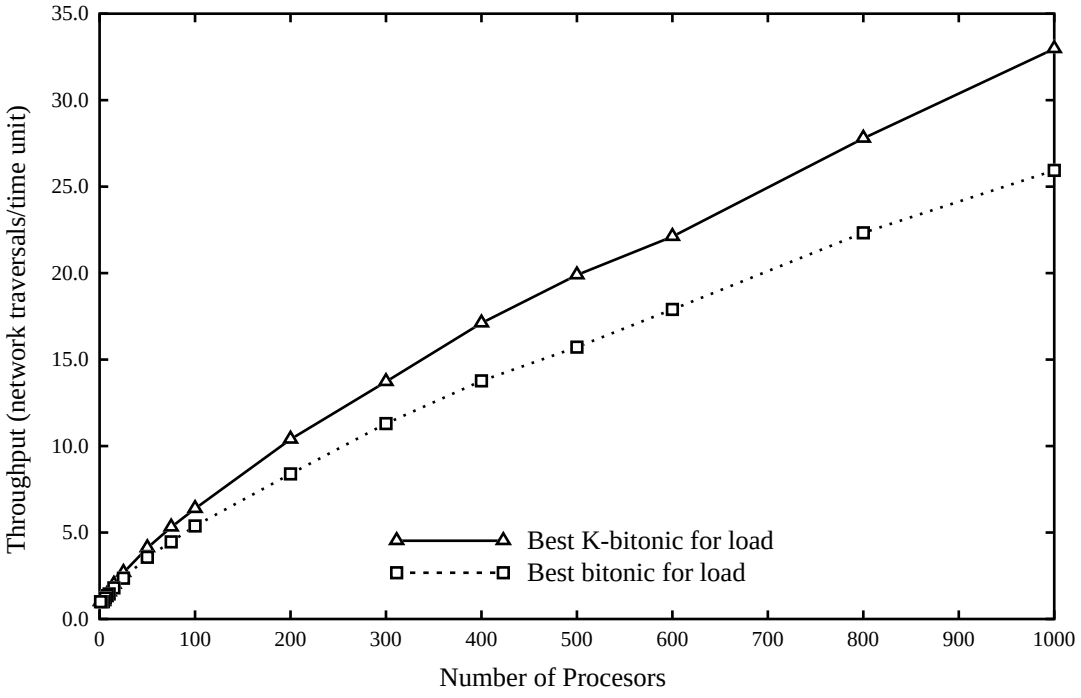

Figure 4: Performance of the best bitonic and k-bitonic networks.

5.1 Performance of K-bitonic vs. Bitonic Networks

Figure 4 compares the performance of bitonic and k-bitonic networks. For each number of processors, we varied the width of the bitonic network, and the width and balancer size of

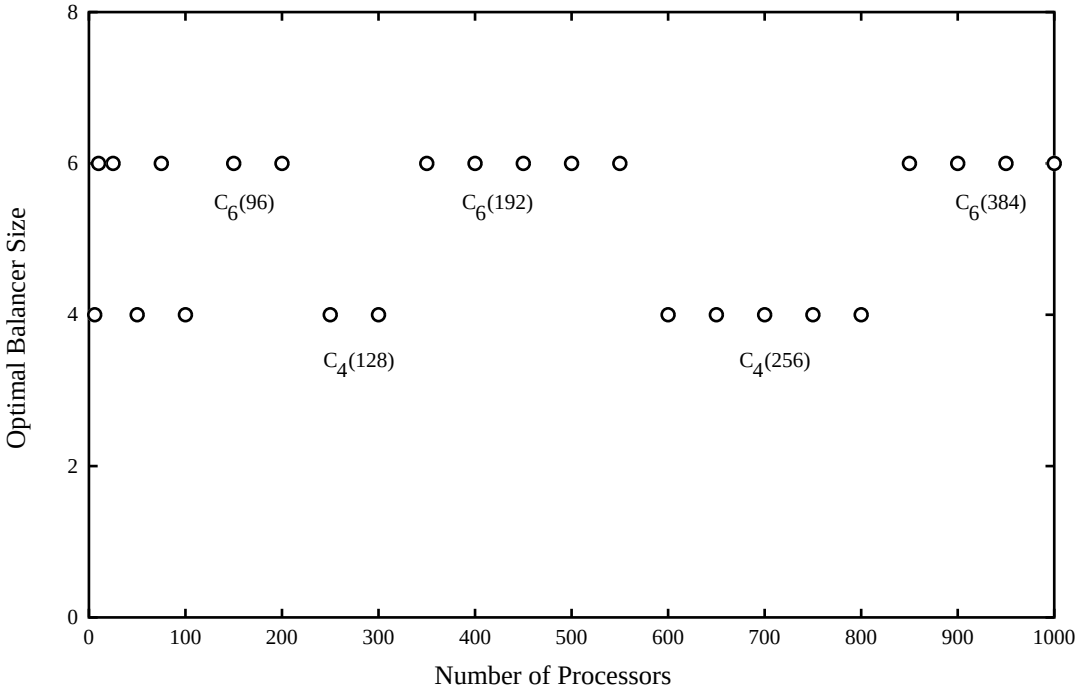

Figure 5: Optimal balancer size vs. network load for k-bitonic networks.

the k-bitonic network, to find the network of each type with the best performance. The figure shows that the k-bitonic network consistently outperforms the bitonic network; the improvement in throughput is about 25%.

5.2 Choosing the Optimal k

When choosing k, we are balancing two competing factors. If k is too small, the network's depth will be too large; it will take too long to traverse all the layers. If k is too large, the contention for each balancer will be too high; it will take too long to pass through each layer. The optimal k balances these factors perfectly.

Figure 5 shows the best balancer size for varying numbers of processors. The graph shows that the optimal size varies with the load, but is not two. In all cases, the best k is either 4 or 6, but there is no apparent pattern to guide our choice between the two alternatives. The difference seems to lie in roundoff effects due to the floors and ceilings in the expression for the network's depth.

Intuition, and approximate performance models[2], suggest that the optimal k should grow as the load increases. The experimental evidence does not support this hypothesis. The lack of a pattern in figure 5 does not allow us to firmly reject it either. More evidence is needed to decide.

[2]not discussed in this version of the paper

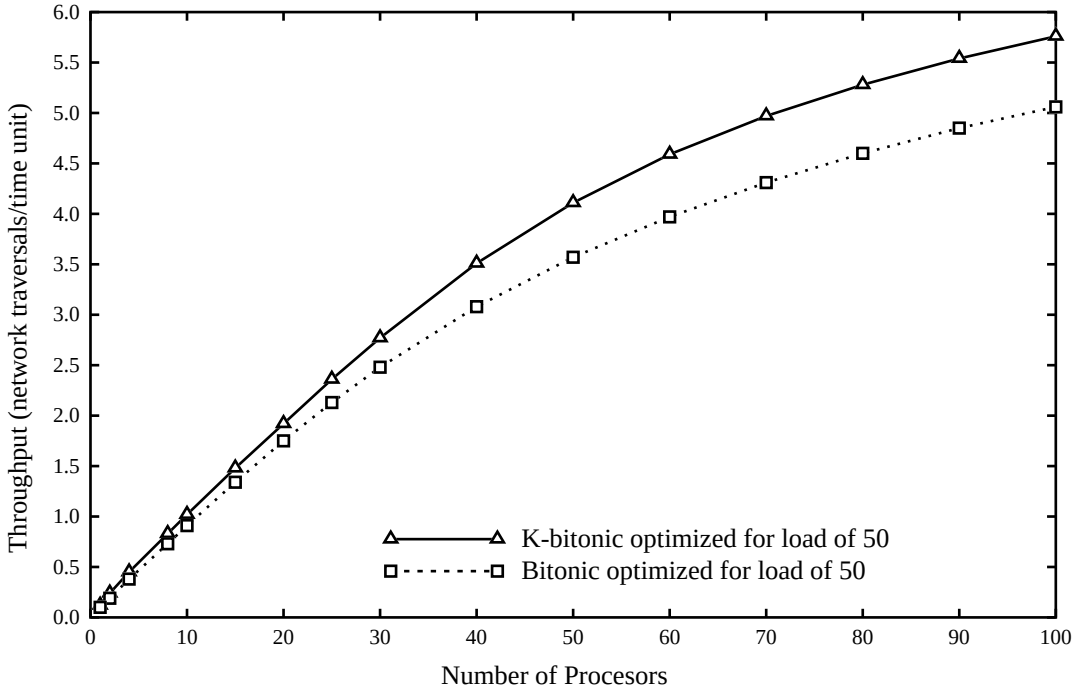

Figure 6: Robustness of bitonic and k-bitonic networks against changes in load.

5.3 Robustness

In practice, the load on a counting network might not be known in advance. Although the number of processors might be known, the intensity with which they use the counting network could be unpredictable, or could even vary over the course of a run. As a result, we would like our networks to perform reasonably for unexpected loads.

Figure 6 considers the robustness of our networks under variations in load. We found the optimal bitonic and k-bitonic networks for a load of 50 processors. We then observed the performance of these two networks for loads between 1 and 100 processors. Figure 6 shows the performance of the k-bitonic $C_4(32)$ versus the bitonic $C(16)$. As before the k-bitonic counting network has higher throughput at all loads than the bitonic counting network. Both networks appear to be equally robust; the performance difference merely reflects the inherent advantage of k-bitonic networks.

5.4 Performance Summary

Our k-bitonic networks clearly outperform the original bitonic networks; the difference is about 25%. Both networks adapt reasonably well to changes in load.

Unfortunately, we have no definite procedure for choosing the balancer size k. Choosing either $k = 4$ or $k = 6$ works well in all the cases we tried. We are unable to predict whether this will hold true for larger values of k.

6 Conclusions

We have introduced counting networks built from k-balancers. The ability to vary k allows us to smoothly trade off network depth against contention. At one extreme, $k = w$, our networks reduce to the traditional single-lock solution, with minimum depth but maximum contention. At the other extreme, $k = 2$, our networks reduce to the counting networks of Aspnes *et al.*, with large depth but low contention. Our networks span the continuum between these two extreme solutions.

In practice, the ability to vary the balancer size k allows a significant performance advantage over either of the extreme solutions. With several hundred processors, our networks typically have throughput about 25% higher than those of Aspnes *et al.* This performance gap appears to be growing as the number of processors increases, but we cannot be sure that it continues to do so asymptotically.

References

[AA92] E. Ahoronson and H. Attiya. Counting Networks with Arbitrary Fan-Out. In *3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 104–113, January 1992.

[AHS91] J. Aspnes, M. Herlihy, and N. Shavit. Counting Networks and Multi-processor Coordination. In *23st Annual ACM Symposium on Theory of Computing*, pages 348–358, 1991.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. *Combinatorica*, 3:1–19, 1983.

[And90] T. E. Anderson. The Performance of Spin Lock Alternatives for Shared Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):6–16, January 1990.

[Bat68] K. E. Batcher. Sorting Networks and Their Applications. In *Proceedings of AFIPS Joint Computer Conference*, volume 32, pages 307–314, 1968.

[Klu91] M. R. Klugerman. Lecture 17: Counting Networks. In F.T. Leighton, C.E. Leiserson, and N. Kahale, editors, *Research Seminar Series 15: Advanced Parallel and VLSI Computation,* pages 153-161. MIT Press, 1991.

[KP92] M. R. Klugerman and C. G. Plaxton. Small-Depth Counting Networks. In *24st Annual ACM Symposium on Theory of Computing*, pages 417–428, 1992.

[MCS91] John Mellor-Crummey and Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, February 1991.