Modeling Parallel Bandwidth: Local vs. Global Restrictions *

Micah Adler[†] Phillip B. Gibbons[‡] Yossi Matias[§] Vijaya Ramachandran[¶]

May 30, 1997

Abstract

Recently there has been an increasing interest in models of parallel computation that account for the bandwidth limitations in communication networks. Some models (e.g., BSP and LOGP) account for bandwidth limitations using a per-processor parameter $g > 1$, such that each processor can send/receive at most h messages in $g \cdot h$ time. Other models (e.g., PRAM(m)) account for bandwidth limitations as an aggregate parameter $m < p$, such that the p processors can send at most m messages in total at each step.

This paper provides the first detailed study of the algorithmic implications of modeling parallel bandwidth as a per-processor (local) limitation versus an aggregate (global) limitation. We consider a number of basic problems such as broadcasting, parity, summation and sorting, and give several new upper and lower time bounds that demonstrate the advantage of globally-limited models over locally-limited models given the same aggregate bandwidth (i.e., $p \cdot \frac{1}{g} = m$). In general, globally-limited models have a possible advantage whenever there is an imbalance in the number of messages sent/received by the processors. To exploit this advantage, the processors must schedule the sending of messages so as to respect the aggregate bandwidth limit. We present a new parallel scheduling algorithm for globally-limited models that enable an unknown, arbitrarily-unbalanced set of messages to be sent through the limited bandwidth within a $(1+\epsilon)$ factor of the optimal offline schedule w.h.p., even if the penalty for overloading the network is an exponential function of the overload. We also present a near-optimal algorithm for the case where long messages must be sent as flits in consecutive time steps, as well as for the case where new messages to be sent arrive dynamically over an infinite time line. These results consider both message passing (distributed memory) and shared memory scenarios, and improve upon the best locally-limited model results by a factor of $\Theta(g)$. Finally, we present results quantifying the power of concurrent reads in a globally-limited bandwidth setting, including showing an $\Omega(\frac{p \lg m}{m \lg p})$ time separation between the exclusive-read and the concurrent-read PRAM(m) models, which, when $m \ll p$, greatly improves upon the $2^{\Omega(\sqrt{\lg p})}$ separation known previously.

*Preliminary version to appear in *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, June 1997.

[†]Heinz Nixdorf Institute, Room F1-119, Fürstenallee 11, D-33102 Paderborn, Germany. email: micah@uni-paderborn.de. Part of the work was done while visiting Bell Laboratories.

[‡]Bell Laboratories (Lucent Technologies), 600 Mountain Ave, Murray Hill NJ 07974. email: gibbons@research.bell-labs.com

[§]Bell Laboratories (Lucent Technologies), 600 Mountain Ave, Murray Hill NJ 07974. email: matias@research.bell-labs.com

[¶]Dept. of Computer Sciences, University of Texas at Austin, Austin TX 78712. email: vlr@cs.utexas.edu. Supported in part by NSF grant CCR/GER-90-23059. Part of the work was done while visiting Bell Laboratories.

1 Introduction

Recently there has been an increasing interest in high-level models for general-purpose parallel computing that account for the bandwidth limitations in communication networks. Some models, such as the well-studied BSP [44] and LOGP [19] models, assume that the primary bandwidth limitation in the network is captured by a *local* restriction on the rate at which an individual processor can send or receive messages. In the BSP model, processors communicate through h-relations, in which each processor sends and receives at most h messages, at a cost of $g \cdot h$, where g is a bandwidth parameter. In the LOGP model, processors are charged o overhead to send or receive a message and can only send a message every g steps. Thus in both models, a large value for the parameter g models a *per-processor* restriction on network bandwidth. Other models, such as the PRAM(m) model [39], assume that the primary bandwidth limitation in the network is captured by a *global* restriction on the rate at which messages can traverse the network. In the PRAM(m) model, there are m memory cells that can be used to communicate between the processors. A value for the parameter m that is much smaller than the number of processors models an *aggregate* restriction on network bandwidth. The LOGP model also provides a capacity constraint on the network, but this is modeled as a per-processor restriction bounding the number of messages simultaneously in transit to or from any one processor.

Whether a local or a global bandwidth limitation is more suitable depends on the communication network of the machine being modeled. Local bandwidth limitations seem more suitable for networks in which each processor has access to its "share" of the network bandwidth and no more. Also, if the primary bandwidth bottleneck is in the processor-network interface, then bandwidth should be modeled on a per-processor basis. Global bandwidth limitations seem more suitable for networks in which processors can "steal" unused bandwidth, by routing along alternative paths. If the primary network bottleneck is the bisection bandwidth, and this bandwidth can be divided among any subset of the processors, then bandwidth should be modeled on an aggregate basis.

As an example of the impact of local versus global bandwidth restrictions, consider the problem of a single processor sending a distinct message to each of the $p-1$ other processors (*one-to-all personalized communication* [33]). Suppose that a processor can send at most one message per time step. Then with a per-processor bandwidth parameter $g > 1$, bandwidth restrictions impose a lower bound of $g(p-1)$ time. On the other hand, with an aggregate bandwidth parameter m, the bandwidth is not the bottleneck for any $m \geq 1$, and we have a lower bound of only $p - 1$.

Contributions of this paper. This paper provides the first detailed study of the algorithmic implications of modeling parallel bandwidth as a per-processor limitation (*locally-limited*) versus an aggregate limitation (*globally-limited*). For concreteness, we consider the following four models:

- The BSP model [44], a message-passing model with a per-processor bandwidth parameter g, denoted in this paper as the BSP(g) model.
- The QSM model [24], a shared-memory model with a per-processor bandwidth parameter g, denoted in this paper as the QSM(g) model.
- The BSP(m) model (defined in this paper), similar to the BSP(g) but with an aggregate bandwidth parameter m.
- The QSM(m) model (defined in this paper), similar to the QSM(g) but with an aggregate bandwidth parameter m.

Each of these models assumes that a processor can pipeline its messages or memory requests. Unlike, e.g., the capacity constraints of the PRAM(m) and the LOGP, the BSP(m) and the QSM(m) impose a penalty for overloading the network that grows with the amount of overload. For lower bounds, we assume that the penalty is only linear in the overload, whereas for upper bounds, we assume (pessimistically) that the penalty is exponential in the overload. The four models are bulk-synchronous models, although many of our results extend to more asynchronous models.

We identify situations where strictly better algorithmic results can be obtained if the bandwidth limitation is global, rather than local. First, we consider a number of basic problems such as broadcasting, parity, summation, list ranking and sorting, and give several new upper and lower time bounds that demonstrate and quantify the advantage that globally-limited models such as the BSP(m) and QSM(m) have over locally-limited models such as the BSP(g) and QSM(g). The comparison applies to a range of values of m and g, but for simplicity, we will assume in this paper that both types of models have the same aggregate bandwidth (i.e., $p \cdot \frac{1}{g} = m$). Some of our lower bound results are obtained using a new technique for converting lower bounds for the CRCW PRAM into lower bounds for the BSP(g) and the QSM(g).

Next, we consider the general (unbalanced) routing problem, in which each processor has an arbitrary number of messages to send to other processors. In locally-limited models, bandwidth restrictions impose a lower bound of $g \cdot h$, where h is the maximum number of messages sent or received by any one processor. In globally-limited models, bandwidth restrictions and the fact that each processor can send or receive only one message at a time impose a lower bound of $\max(\frac{n}{m}, h)$, where n is the total number of messages to be sent. Note that when $m = \frac{p}{g}$, we have that $\max(\frac{n}{m}, h) = \max(g \cdot \frac{n}{p}, h) \leq g \cdot h$. Thus the local bandwidth lower bound is at best equal to the global bandwidth lower bound and it is worse by a factor of g whenever there is a significant imbalance ($h \geq g \cdot \frac{n}{p}$).

To exploit this inherent advantage, the processors in globally-limited models must schedule (i.e., stagger) the sending of messages so as to respect the aggregate bandwidth limit. (Note that no special scheduling is needed for locally-limited models.) When the communication pattern is known in advance, careful offline scheduling can be done. However, in general, this may not be the case, and we present the following three parallel scheduling algorithms for globally-limited models. (The results are stated for the BSP(m), but similar results are shown for the QSM(m) in the full paper). Here, the parameter L is the periodicity parameter from the BSP model, which reflects message latency plus synchronization overheads. Let $\epsilon < 1$ be a small positive constant.

- A randomized algorithm for scheduling an unknown, arbitrarily-unbalanced h-relation on the BSP(m) that runs in $\max\{(1+\epsilon)n/m, h, L\} + \tau$ time, with high probability, where n is the total number of messages sent and $\tau = O(p/m + L + L \lg m / \lg L)$ is the time to compute and broadcast the value of n. For the important case where $n \gg p$ and $\max(n/m, h) \gg L$, τ is negligible, and hence this algorithm is roughly within a $(1+\epsilon)$ factor of the optimal offline schedule. Alternatively, if n is known by all the processors, then $\tau = 0$, and again, we are within a $(1+\epsilon)$ factor of optimal.

- Second, we consider the case where processors send messages of arbitrary lengths, and the flits of a long message must be sent in consecutive time steps. We provide a randomized algorithm for scheduling an unknown, arbitrarily-unbalanced h-relation on the BSP(m) that runs in $\max\{(2+\epsilon)n/m, h, L\} + O(p/m + L + L \lg m / \lg L)$ time, with high probability, where h is the maximum total length of messages sent or received by any one processor and n is the total length of all the messages sent.

- Third, we consider a dynamic scenario in which new messages to be sent by the processors are generated according to an Adversarial Queuing Theory model over an infinite time line. We present a randomized algorithm that can successfully schedule and send messages through the limited bandwidth up to an arrival rate that is within a constant factor of optimal.

We also show how the second algorithm given above can be employed in situations where a processor incurs a non-unit start-up cost in initiating or receiving a message (e.g., as modeled by the overhead parameter, o, in the LOGP model). Such start-up costs are often the motivation for sending long messages, and we show that the second algorithm can be adapted to handle such costs with a running time within a $(2+\epsilon)$ factor of optimal in many cases. One implication of the efficient running times of these routing protocols is that we can, in many scenarios, consider models with global bandwidth restrictions that do not force the algorithm designer to explicitly schedule message transmission times.

Finally, we present results quantifying the power of concurrent-read versus exclusive-read and queue-read in a globally-limited bandwidth setting. Among our results is an $\Omega(\frac{p \lg m}{m \lg p})$ time separation between the exclusive-read and the concurrent-read PRAM(m) models, which, when $m \ll p$, greatly improves upon the best previously known separation of $2^{\Omega(\sqrt{\lg p})}$ from [1].

Outline. The rest of the paper is organized as follows. Section 2 presents the definitions of the models we consider. Section 3 describes the related work in detail. Section 4 considers a number of basic problems for which we quantify the gap in time between locally-limited and globally-limited models. Section 5 presents our results relating concurrent-read and exclusive- or queue-read. Our results for sending unbalanced h-relations are given in Section 6.

2 Modeling parallel bandwidth

The BSP model. The Bulk-Synchronous Parallel (BSP) model [44, 45] consists of p processor/memory components communicating by sending point-to-point messages. The interconnection network supporting this communication is characterized only by a per-processor throughput parameter g and a latency parameter L. The particular topology of the network is ignored and the cost to communicate among processors is assumed to be uniform, independent of the identity of the processors. A BSP computation consists of a sequence of "supersteps" separated by bulk synchronizations (typically, a barrier synchronization among all the processors). In each superstep the processors can perform local computations and send and receive a set of messages. Messages are sent in a pipelined fashion (i.e., each processor may issue messages and continue with its computation prior to the receipt of those messages). Messages sent in one superstep will arrive prior to the start of the next superstep. The time charged for a superstep is calculated as follows. Let w_i be the amount of local work performed by processor i in a given superstep. Let s_i (r_i) be the number of messages sent (received) by processor i. Let $w = \max_{i=1}^{p} w_i$, and $h = \max_{i=1}^{p} \max(s_i, r_i)$. Then the cost, T, of a superstep is defined to be

$$T = \max(w,\ g \cdot h,\ L)\ .$$

Intuitively the communication throughput parameter, g, is the best sustainable gap between message sends issued by each individual processor; therefore $1/g$ represents the available bandwidth per processor. Intuitively the communication latency parameter, L, called the "periodicity factor" in [44], is the worst case time to deliver a message between two processors in an otherwise unloaded network plus the time to perform a barrier synchronization.

The model assumes that all messages are the same size; to extend to variable length messages, one may replace "messages" with "flits/packets" in the definition above such that a long message is viewed as a sequence of flits to be sent between a pair of processors. (The same g is used for both short and long messages; see [3, 9] for examples of models that use distinct throughput parameters for short and long messages.) In this paper, we denote the BSP model as the BSP(g) model, to make explicit its per-processor bandwidth parameter g.

The BSP(m) model. We now consider a variant of the BSP model that replaces the per-processor bandwidth parameter g by an aggregate bandwidth parameter m. We denote this globally-limited BSP model as the BSP(m) model. Let w and h be defined as in the BSP(g). At each step t of a superstep, each processor may initiate at most one message send. The interconnection network is assumed to handle up to m message sends in a step; exceeding this limit results in a larger charge for the step. Let m_t be the number of message sends initiated in step t of a given superstep. Let f_m be a function of m_t that equals 0 when $m_t = 0$, equals 1 when $1 \leq m_t \leq m$, and such that when $m_t > m$, $f_m(m_t) \geq m_t/m$ is an increasing function of m_t. Let τ be the number of steps in a given superstep, and let $c_m = \sum_{t=1}^{\tau} f_m(m_t)$. Then the time cost, T, for the superstep is defined to be

$$T = \max(w,\ h,\ c_m,\ L)\ .$$

The definition allows for a choice of functions f_m when $m_t > m$. For lower bounds, we will assume the minimum (linear) charge $f_m(m_t) = f_m^{\ell}(m_t) = m_t/m$; intuitively, this models a network that can absorb arbitrary message injection rates and sustain a throughput of m messages at a time: there is no penalty for overloading the network. For upper bounds, we will assume $f_m(m_t) = f_m^u(m_t) = e^{\frac{m_t}{m}-1}$ when $m_t > m$; intuitively, this models a network that suffers an exponential penalty for injecting more messages than the network's aggregate bandwidth limit. We view this as representative of the maximum charge likely to be incurred in a network.[1] Note that $f_m^u(m_t) \geq f_m^{\ell}(m_t)$ for all m and all $m_t \geq m$.

Both the BSP(g) and the BSP(m) models can be extended to variable length messages by replacing "messages" with "flits/packets" in the definition above, and viewing a long message as a sequence of flits to be sent between a pair of processors. We consider both the case where flits of a long message must be injected in consecutive time steps and the case where they need not be. The latter case thus reduces to the fixed-sized message case. The same g is used for both short and long messages; see [3, 9, 32] for examples of models that use distinct throughput parameters for short and long messages. Note that the BSP(g) model does not insist that flits of long messages be injected in consecutive time slots.

A simplified cost metric. We also briefly mention a variant of the BSP(m) model where we ignore the exact sending times within a superstep and charge

$$T = \max(w, h, n/m, L)$$

for any superstep that transmits n messages (where w, h and L are defined as before).[2] We refer to this as the *self-scheduling* BSP(m) model. By using the results we prove in Section 6, we can show that any algorithm $\mathcal{A}$ that has running time t on the p-processor self-scheduling BSP(m) can be implemented on a p-processor BSP(m) in time $(1+\epsilon)t$ with very high probability. Thus, we shall see that in most

[1] With this pessimistic measure, m represents the breaking point at which the performance of the network deteriorates drastically.

[2] This is similar to a model where the cost of a superstep is $g_1 n/p + g_2 h$, as proposed in the conclusion of [35].

situations, it is sufficient to consider the self-scheduling variant of the BSP(m) model. The simplicity of this model eases the task of algorithm design.

The QSM model. The *Queuing Shared Memory* (QSM) model [24] consists of p processors, each with its own private memory, communicating by reading and writing locations in a shared memory. The interconnection network supporting this communication is characterized by a per-processor throughput parameter g. Processors execute a sequence of bulk-synchronous phases (supersteps), each consisting of an arbitrary interleaving of the following operations: (i) *Shared-memory reads*: Each processor i copies the contents of r_i shared memory locations into its private memory. The value returned by a shared-memory read can only be used in a subsequent phase. (ii) *Shared-memory writes*: Each processor i writes to w_i shared-memory locations. (iii) *Local computation*: Each processor i performs c_i RAM operations, involving only its private state and private memory.

Concurrent reads or writes (but not both) to the same shared-memory location are permitted in a phase. In the case of multiple writers to a location x, an arbitrary write to x succeeds in writing the value present in x at the end of the phase. Each shared memory location can be read or written by any number of processors. Let the *maximum contention* of a QSM phase be the maximum, over all locations x, of the number of processors reading x or the number of processors writing x.

Consider a QSM phase with maximum contention κ. Let $w = \max_i\{c_i\}$ for the phase, i.e. the maximum over all processors i of its number of local operations, and likewise let $h = \max_i\{r_i, w_i\}$ for the phase. Then the time cost, T, of the superstep is defined to be

$$T = \max(w,\ g \cdot h,\ \kappa)\ .$$

Note that although the model charges g per shared memory request at a given processor (the $g \cdot h$ term in the cost metric), it only charges 1 per shared memory request at a given location (the κ term in the cost metric). In this paper, we denote the QSM model as the QSM(g) model, to make explicit its per-processor bandwidth parameter g.

The QSM(m) model. Lastly, we consider a variant of the QSM(g) model that replaces the per-processor bandwidth parameter g by an aggregate bandwidth parameter m. We denote this globally-limited QSM model as the QSM(m) model. Let w, κ and h be defined as in the QSM(g). Let m_t be the number of shared memory read or write requests initiated in step t of the given phase (superstep), and let the function f_m and the quantity c_m be defined as in the BSP(m). Then the cost, T, of the superstep is defined to be

$$T = \max(w,\ h,\ \kappa,\ c_m)\ .$$

3 Related work

Several authors have pointed out that the BSP(g) model does not accurately reflect the cost of sending unbalanced communication patterns, and experimental evidence measuring this inaccuracy is provided in [47]. Modifications to the BSP(g) model to more accurately reflect communication costs have been proposed, for example the E-BSP of [35] as well as the Y-BSP of [21]. An early work that proposed and studied an abstract model that addressed bandwidth considerations is the DRAM model of Leiserson and Maggs [37], which assesses communication costs in terms of the congestion of messages across every cut in the underlying network. The DRAM, as well the Y-BSP and the E-BSP incorporate cost measures which vary greatly depending on the underlying network architecture, and both the Y-BSP and the

E-BSP incorporate cost measures designed to capture network proximity. Although this approach can provide a more accurate estimation of running time, the complexity of the resulting models makes it difficult to isolate the relative effects of local and global bandwidth restrictions, and thus these models are not well suited for the study presented here. These models also differ from the globally-limited models we study here in that they do not provide an advantage to algorithms that schedule communication within a superstep so as to avoid exceeding the capacity of the network. However, as we shall see in Section 6, there is a simple algorithm for scheduling messages to avoid exceeding the network capacity, and thus this is a less significant difference.

Many previous works have studied the *total-exchange* (also called *complete exchange* and *all-to-all personalized communication*) primitive in which each processor has a distinct message to send to every other processor (see, e.g., [8, 13, 16, 28, 30, 31, 33, 38, 42, 43]). The total-exchange primitive has been incorporated into communication libraries such as the Collective Communication Library provided with the IBM SP-2 [7]. It is used in matrix transposition, two-dimensional Fourier Transform, conversion between storage schemes (remapping of arrays in HPF compilers), shuffle permutation, N-body problems, matrix-vector multiplication, Ascend and Descend algorithms, and routing h-relations. Efficient total-exchange algorithms have been studied for complete networks, hypercubes and tori, d-dimensional mesh of busses, circuit-switched butterflies, the OCPC, multi-port fully-connected message-passing models, etc. Unlike previous work, this paper considers the total-exchange on an abstract, but bandwidth-limited model, the BSP(m). Moreover, whereas most previous work has considered only the case in which the messages sent between processors are the same length, we consider the more general *unbalanced total-exchange* problem (unbalanced h-relation), in which each message may be of a different nonnegative length.

Bhatt *et al.* [13] have studied the unbalanced total-exchange problem (which they call "chatting") on leveled networks. They consider a scenario in which (1) communication is performed in rounds, so that all messages in an unbalanced total-exchange are routed before any new messages are generated, (2) each message travels through the network as a contiguous stream of flits, (3) the routing is oblivious, and (4) there are no buffers or queues in the network. In this paper, we consider essentially the same scenario, except that we consider a model, the BSP(m), which abstracts away the topology and buffering considerations of the network. Specifically, we consider rounds as in (1), and present a schedule of injecting messages such that a message of length x_{ij} consumes a unit of (bisection) bandwidth for x_{ij} time steps, in the spirit of (2) and (4). The algorithm of Bhatt *et al.* first combines all p^2 (source, destination, length) triples in a single processor that computes an efficient schedule and then broadcasts the schedule to all the processors; collecting these triples takes $\Theta(p^2 + L)$ time on the BSP(m). This contrasts with our algorithm, which only computes and broadcasts n, the sum of the lengths of all the messages; this can be done in $O(p/m + L + L \lg m / \lg L)$ time on the BSP(m).

Routing the dynamic case of the total-exchange problem (where the messages to be sent arrive dynamically over an infinite time line) has been considered in the context of routing messages through multiple Ethernet-like channels by Raghavan and Upfal [41] and Goldberg and MacKenzie [25]. These works study the problem of scheduling the transmission of messages from p processors to m communication channels. The arrival times of the messages are typically determined by a simple random process, such as a Bernoulli distribution, and the objective is to successfully route the messages over the requested channels. Different routing protocols are analyzed, and the primary concern is to show that the protocols are stable for sufficiently low arrival rates of messages. These results can be used to provide dynamic unbalanced total-exchange algorithms for the case where the message arrival times are chosen randomly, and at most m channels are utilized at any time step. We here consider the case where a malicious adversary chooses the message arrival times, instead of an oblivious random process.

Also, the models we consider here are incomparable to the multiple channel model. In the multiple channel model, if more than one processor tries to send simultaneously using the same channel, no processor is successful, but the channel is clear at the next time step. Thus, algorithms designed for the multiple channel model have the advantage over our model that a single bad step does not cost more than one unit of time, and can provide feedback for use at the next step. In the BSP(m) model we consider, on the other hand, a single bad step can require time $e^{\frac{p}{m}-1}$. On the other hand, algorithms designed for the models considered here have the advantage that processors are not required to choose a specific channel and that any attempted transmission is guaranteed to eventually be successful. For example, consider the algorithm where every processor attempts to send a message at every time step until it is successful. In the multiple channel model, if more than m processors have messages to send, this algorithm never terminates. In our model, the algorithm is successful after one (possibly very slow) step.

Algorithms for the PRAM(m) and closely related models have appeared in [46, 5, 39, 2, 1]. The most significant difference between the PRAM(m) and the BSP(m) and the QSM(m) is that the PRAM(m) model includes a separate, concurrently readable Read Only Memory that contains the input to any problem being solved. This means that in this model, distributing the entire input to the processor occurs without charge. Also, all the work in the PRAM(m) model has been in a CRCW framework, with the exception of [2], which assumed that reading was either exclusive or queued. Due to these features, this model does not seem to have an efficient emulation on lower level models such as the BSP, or BSP(m). On the other hand, algorithm design for the PRAM(m) is complicated by the fact that there are only m shared memory locations.

We finally mention that several groups have been recently involved in implementations and experimentations of parallel algorithms, using bandwidth-limited general purpose models (see, e.g., [6, 14, 17, 19, 20, 23, 27, 29, 40]).

♣ Should we keep the entire list?, part of it?, add on this?

4 Separation results between locally-limited and globally-limited models

In this section we present some algorithmic results and lower bounds for certain problems that establish various time separations of the QSM(m) and BSP(m) model with latency L from the QSM(g) model and BSP(g) model with latency L respectively. In the following, we assume that the gap and the aggregate bandwidth are related by $g = p/m$, where p is the number of processors. We note that any QSM(g) algorithm can be emulated on the QSM(m) with the same asymptotic time bound, as can a BSP(g) algorithm on a BSP(m). ♣ Provide a proof in the appendix. Our results are tabulated in Table 1 for the case when $p = n$, the size of the input. ♣ We should consider in the appendix the issue of self-simulations for the various models (not urgent).

All of the upper bounds for QSM(m) and BSP(m) with the exception of the results for sorting follow from the following observation. Given an EREW PRAM or QRQW PRAM algorithm that runs in time $t(n)$ and work $w(n)$ it can be converted into a QSM(m) algorithm that runs in time $O(n/m+t(n)+w(n)/m)$ as follows. We distribute the input elements evenly into the first m processors in time n/m, and then

Separation Results			
problem	**stronger model**	**weaker model**	**time separation (for $n = p$)**
One-to-all communication	QSM(m): $\Theta(p)$ BSP(m): $\Theta(p + L)$	QSM(g): $\Theta(gp)$ BSP(g): $\Theta(gp + L)$	$\Theta(g)$ $\Theta(g)$
Broadcasting	QSM(m): $\Theta(\lg m + p/m)$ BSP(m): $O(L \lg m / \lg L + p/m + L)$	QSM(g): $\Theta(g \lg p / \lg g)$ BSP(g): $\Theta(L \lg p / \lg(L/g))$	$\Theta(\lg p / \lg g)$ $\Theta(\lg L \lg p / (\lg(L/g) \lg m))$
Parity, Summation	QSM(m): $\Theta(\lg m + n/m)$ BSP(m): $O(L \lg m / \lg L + n/m + L)$	QSM(g): $\Omega(g \lg n / \lg\lg n)$ BSP(g): $\Theta(L \lg n / \lg(L/g))$	$\Omega(\lg n / \lg\lg n)$ $\Theta(\lg L \lg n / (\lg(L/g) \lg m))$
List ranking	QSM(m): $O(\lg m + n/m)$ BSP(m): $O(L \lg m + n/m)$	QSM(g): $\Omega(g \lg n / \lg\lg n)$ BSP(g): $\Omega(g \lg n / \lg\lg n + L)$	$\Omega(\lg n / \lg\lg n)$ $\Omega(\lg n / \lg\lg n)$
Sorting ($m = O(n^{1-\epsilon})$)	QSM(m): $\Theta(n/m)$ BSP(m): $\Theta(n/m + L)$	QSM(g): $\Omega(g \lg n / \lg\lg n)$ BSP(g): $\Omega(g \lg n / \lg\lg n + L)$	$\Theta(\lg n / \lg\lg n)$ $\Theta(\lg n / \lg\lg n)$

Table 1: *Some results separating the globally-limited models from the corresponding locally-limited models.* Here, g is the gap parameter, L is the latency, p is the number of processors, n is the size of the input, and $m = p/g$ is the bandwidth parameter in QSM(m) and BSP(m). The separation results are for $n = p$, and for suitable values of L and g.

simulate the PRAM algorithm on m processors in time $O(w(n)/m + t(n))$ by a naive simulation of the PRAM algorithm on the QSM(m). This is possible since the simulation will generate at most m memory accesses per step. ♣ Provide a proof in the appendix. In particular, the issue of contention needs to be addressed. This method will usually generate a bound of $O(t(n) + w(n)/m)$ since $w(n) \geq n$ for most nontrivial problems. We can map this onto the BSP(m) to run in time $O(L \cdot t(n) + w(n)/m)$ by pipelining the computations in each of the $t(n)$ steps.

For the problem of *sorting* n keys, upper bounds of $O(n/m)$ in the QSM(m) and $O(n/m+L)$ for the BSP(m) hold whenever $m = O(n^{1-\epsilon})$ for some $\epsilon > 0$. The algorithm that achieves these bounds routes the input keys to a subset of $m \lg n$ processors. We can then sort these keys using the deterministic sorting algorithm from [2], which is an adaptation of columnsort. When $m = O(n^{1-\epsilon})$, the running time of this algorithm is within a constant of the time required to route a permutation of the n keys that is balanced on the subset of $m \lg n$ processors. This requires time $O(n/m)$ and $O(n/m + L)$ on the QSM(m) models and BSP(m) models respectively. The keys are then routed to the processors that need to output them. For larger values of m, we can get a bound of $O((n \lg n)/m + \lg n)$ on the QSM(m) (valid for all values for m) by using the general strategy described above of mapping regular PRAM algorithms onto the QSM(m), and on the BSP(m) a bound of $O((n/m + L) \lg n)$ derivable either from the QSM(m) algorithm or the BSP(g) algorithm.

4.1 Converting CRCW PRAM lower bounds into BSP(g) lower bounds

We note that a time lower bound of $\Omega(t(n))$ for p processors with unlimited local computational powers on the CRCW PRAM implies a time lower bound for the same problem of $\Omega(g \cdot t(n))$ for p processors with unlimited local computational powers on the QSM(g). However on a QSM(m) it translates into a lower bound of only $\Omega(t(n))$. Thus the CRCW lower bound result of Beame and Hastad [10] gives a lower bound for the n-element *parity*, *summation*, *list ranking* and *sorting* problems of $\Omega(g \cdot \lg n / \lg\lg n)$ time on the QSM(g) for deterministic and randomized algorithms when the number of processors is polynomial in n.

The result of [10] is also used to derive the lower bounds for the BSP(g) model. Any lower bound of the form $t(n)$ for the number of steps required on the CRCW PRAM (with unlimited local computation, and a polynomial number of processors) gives a lower bound of the form $\Omega(gt(n))$ for the BSP(g). To prove this, it is sufficient to show that we can realize an h-relation on the CRCW in time $O(h)$. Given this, we can simulate any BSP(g) superstep requiring time $T(n)$ for communication using time $O(T(n)/g)$ on the CRCW PRAM. Thus, any algorithm that requires time $g \cdot t(n)$ for communication on the BSP(g) gives an algorithm with $t(n)$ steps on the CRCW PRAM.

An h-relation can be realized on a CRCW PRAM with a simple deterministic algorithm that runs in O(h) time using polynomial space and polynomial number of processors:

- Compute m, the max number of messages to be sent by any processor (a simple constant time computation with p^2 processors).
- Use a p by $m \cdot p$ array, where the ith row is for messages to be sent to the ith processor. Partition each row into p blocks of m elements each. The jth processor will write the messages destined for the ith processor in the jth block of row i. This write is done in $\leq h$ steps.
- Repeat:
  - Find the leftmost nonzero entry in each row in parallel (a constant time computation with polynomial number of processors).
  - Transmit this information to the corresponding destination processor.
  - Zero out this leftmost nonzero entry in each row until no row has any nonzero entry

Each iteration of the repeat loop can be done in constant time, so the repeat loop takes O(h) time.

Furthermore, any randomized lower bound of the form $t(n)$ for the number of steps required on a p processor CRCW PRAM (with unlimited local computation) gives a randomized lower bound of the form $g \cdot t(n) \cdot \min(\frac{L+g}{g \lg^* p}, 1)$ for the BSP(g). This follows from the fact that the h-relation problem can be solved on the CRCW PRAM in $O(h + \lg^* p)$ time and linear work with high probability, as follows. The elements are first placed in an array of size $O(hn)$, sorted by the index of their destination processor; this can be done in $O(\lg^*(nh))$ time and $O(nh)$ work w.h.p., using an algorithm for approximate integer sorting [26]. Then, for each element a pointer to its nearest element on its right in the array is found; this can be done in $O(\alpha(nh))$ time (i.e., $o(\lg^*(nh))$) and $O(nh)$ work, using a nearest-zero algorithm [11]. The last step creates a list of elements, consisting of sub-lists of elements with the same destination processors. The elements that are first in their sub-lists are identified, and their location is notified to their destination processor; this can be done in constant time and $O(hn)$ work. Each destination processor can now scan its list in $O(h)$ time. In the case that $L \geq g \lg^* p$, a bound of $t(n)$ for the CRCW PRAM becomes a bound of $g \cdot t(n)$, and for all values of L and g, we get a lower bound of at least $g \cdot t(n)/\lg^* p$.

Also, any deterministic lower bound on time of the form $t(n)$ for a ($p \lg\lg p$)-processor CRCW PRAM (with unlimited local computation and the Arbitrary rule for resolving concurrent writes) provides a deterministic lower bound of the form $g \cdot t(n)$ for the p-processor BSP(g). This follows since we can realize an h-relation on a $(p \lg\lg p)$-processor Arbitrary CRCW PRAM with a deterministic algorithm that runs in $O(h)$ time, as follows.

- Compute and broadcast $\bar{x}$, the maximum number of messages to be sent by any processor. This can be done in $O(\bar{x})$ time. Note that $\bar{x} \leq h$.
- If $\bar{x} \geq \lg\lg p$ then processor i writes its x_i messages to locations $(i-1)\bar{x}+1$ through $i\bar{x}$ in an array of size $\bar{x}p$ (if $x_i < \bar{x}$, then processor i writes null values to the remaining locations). This requires time $O(\bar{x})$ and work $O(p\bar{x})$. This array is then integer chain sorted by destination (the universe is of size p). This requires time $O(\lg\lg p)$ and work $O(p\bar{x}\lg\lg p)$ [12]. Each processor keeps track of the location of every key that it wrote to the sorted chain, and after the chain is sorted, uses this information to find if any of these keys are the first message destined for any processor. Each processor i can then be informed of the first message in the sorted list destined for i, and then each processor reads the messages destined for it in time $O(h)$.
- If $\bar{x} < \lg\lg p$, then each processor, in parallel, performs a concurrent write to a specified location for each of the destination processors for its messages (this requires a team of $\bar{x}$ processors for each of the p processors). For each write that succeeds, the corresponding message is read by the destination processor. If a write does not succeed the processor repeats the write until its write succeeds. This step can be performed in $O(h)$ time and $O(ph\lg\lg p)$ work.

We note that the time is optimal, and the work is a $\lg\lg p$ factor from optimal.

4.2 A lower bound for broadcasting

The lower bound listed in the table for broadcasting in the BSP(g) has been shown in [34] for a model where processors are not allowed to obtain information from non-receipt of messages. We can use a sensitivity argument, as developed in [18], to derive a lower bound for broadcasting both on the BSP(g) and the QSM(g) that accounts for non-receipt of messages. We here present the lower bound for the broadcast problem in the BSP(g) model. In this model, the importance of considering non-receipt of messages is demonstrated by the following algorithm, that uses non-receipt of messages to broadcast a single bit in time $g\lceil \lg_3 p\rceil$, provided $L \leq g$.

Initially, processor 1 has a single bit b that is to be broadcast to processors 2 through p. We maintain the invariant that at the start of step $i \geq 1$, processors 1 through 3^{i-1} know the value of b. During step i, processor $j \leq 3^{i-1}$ sends a message to processor $j + 3^{i-1}$ if $b = 0$ and to processor $j + 2 \cdot 3^{i-1}$ if $b = 1$. After each step, processors synchronize. This is enough information to maintain the invariant, and thus after $\lceil \lg_3 p\rceil$ such steps, every processor knows the value of b.

Theorem 4.1 *Any deterministic algorithm for the broadcast problem on the BSP(g) model requires at least time* $\frac{L \lg p}{2\lg(2L/g+1)}$.

Proof. We consider algorithms that broadcast a single bit; this provides a lower bound for the more general problem of an arbitrary number of bits. Let $S(i,j)$ be the set of possible states that processor i can be in during step j of any possible execution of a given broadcast algorithm. Since we are considering deterministic algorithms, and the input to the problem consists of only one bit, there are exactly two possible program executions, and thus for any i, j $|S(i,j)| \leq 2$. We say that processor i is *sensitive* at step j if $|S(i,j)| = 2$. Let $\mathcal{S}(t)$ be the set of processors that are sensitive during any step of superstep t. Note that any algorithm for the single bit broadcast problem cannot complete before there exists some t for which $|\mathcal{S}(t)| = p$.

Let x_t be the maximum number of messages sent by any processor during the tth superstep of the broadcast algorithm when the input is a 1. Let $\bar{x}_t$ be the maximum number of messages sent by any processor during the tth superstep of the broadcast algorithm when the input is a 0. Both x_t and $\bar{x}_t$ are defined to be 0 for t larger than the respective running times of the algorithm.

Claim 4.2 $|\mathcal{S}(t+1)| \leq (x_t + \bar{x}_t + 1)|\mathcal{S}(t)|$.

Proof. A processor i can only be sensitive during superstep $t+1$ if either it was sensitive during superstep t, or some sensitive processor k sends a message to processor i during some possible execution of the algorithm. However, since there are only two possible executions of the algorithm, the total number of processors that a sensitive processor k can possibly send a message to during any execution of the tth super step is at most $x_t + \bar{x}_t$. ∎

Thus, in order for a broadcast algorithm to successfully terminate on both inputs in n steps, we require that

$$\prod_{t=1}^{n}(x_t + \bar{x}_t + 1) \geq p.$$

The stated lower bound follows by minimizing the following expression for the worst case running time of the algorithm, subject to the requirement given above.

$$\max\left(\sum_{t=1}^{n}\max(L, gx_t), \sum_{t=1}^{n}\max(L, g\bar{x}_t)\right).$$

Let $y_t = \max(x_t, \bar{x}_t)$. We see that

$$2T \geq \sum_{t=1}^{n}\max(L, gy_t) = Y$$

where a necessary condition is that

$$Z = \prod_{t=1}^{n}(2y_t + 1) \geq p.$$

A convexity argument gives us that for any fixed value of Z, Y is minimized by setting all the $y_t = y$ for some value y. Thus, we wish to find the minimum value of $n\max(L, gy)$ subject to $(2y+1)^n \geq p$. We see that $n \geq \frac{\lg p}{\lg(2y+1)}$, and thus we instead minimize $\frac{\lg p}{\lg(2y+1)}\max(L, gy)$. This is minimized by setting $y = \frac{L}{g}$, which gives us that $Y \geq \frac{L\lg p}{\lg(2L/g+1)}$. The stated bound on T follows directly. ∎

5 The power of concurrent read in limited bandwidth

In this section, we discuss the power provided by concurrent access to data in a limited bandwidth setting. When bandwidth is not limited, there is a separation of $\Theta(\lg p)$ between concurrent reading and exclusive reading; this section discusses the analogous separation when bandwidth is limited. This question is relevant when the primary bandwidth bottleneck is either a global restriction, or a local restriction. Since the ability to sort p keys efficiently is used in most simulations of concurrent read using exclusive or queued read, the results on the separation between globally-limited and locally-limited bandwidth restrictions for the problem of sorting (discussed in Section 4) have an important impact on this problem.

An example of a setting where processors have concurrent read access to limited bandwidth is a set of processors that communicate over a shared broadcast bus with insufficient bandwidth to handle communication by every processor at every clock cycle. When the processors are able to access the bus at every time step, this is an environment with concurrent read and a global bandwidth restriction. If, on the other hand, there is a high local cost for accessing the bus, this is an environment with concurrent read and a local bandwidth restriction.

We here consider the problem of simulating concurrent read with exclusive or queued read in a globally-limited bandwidth setting. To model concurrent read limited bandwidth, we use the CRCW PRAM(m) of [39]. In the CRCW PRAM(m), the processors communicate through a shared memory consisting of m shared memory cells that can be read and written concurrently. The input to the problem resides in a separate Read Only Memory which processors can also read concurrently.

Theorem 5.1 *One read step of the* CRCW PRAM*(m) can be simulated on the* QSM*(m) in time $O(\frac{p}{m} + \lg m)$, provided $m = O(p^{1-\epsilon}), \epsilon > 0$.*

Proof. We describe here how to perform a read, but writing is similar. Each processor i writes the pair (j, i) into an array A of size p, where j is the address of the memory location that processor i reads. The array A is sorted using the sorting algorithm of the previous section, and the sorted values are stored in array B. Every processor i reads the ith location of B; processor i determines the value stored in the memory location which has an address stored in $B[i]$. Once this has been accomplished, the processor that appears in $B[i]$ can informed of this value by using another $\frac{2p}{m}$ steps.

Each of the m processors i, where i is congruent to 0 mod $\frac{p}{m}$ reads the memory location whose address appears in $B[i]$ by using the standard EREW PRAM simulation of a step of a CRCW PRAM algorithm. This value is written to location $i/\frac{p}{m}$ of a third array C, along with the address of the memory location that was read to produce this value. For the next $\frac{p}{m}$ steps, called *crucial read steps* at step j, processor i reads location $\lfloor i/\frac{p}{m} \rfloor$ of C when i is congruent to j mod $\frac{p}{m}$. If the memory location address which appears in C is not the same address that processor i read from the array B, then processor i reads that memory location directly. After n/p steps, every processor i knows the value appearing in the memory location with the address that it read from the array B.

This algorithm requires $O(\frac{p}{m} + \lg m)$ steps, and requires at most m memory accesses at any step. Thus, it suffices to show that no more than 1 processor accesses any memory location at any step. To see that this is the case, note that both the EREW PRAM algorithm and the sorting algorithm consist of only exclusive accesses. Furthermore, during the crucial read steps, processor i only reads from the shared memory outside the array C if processor i is reading a different memory location than processor $\lfloor i/\frac{p}{m} \rfloor$ read. Since the addresses of the memory locations read are sorted, there can be no conflict on such a step either. ∎

This can be seen to be optimal by comparing the time to broadcast in the two models. The broadcast problem also shows that there is a gap between the QSM(g) and the CRCW PRAM(m) of at least $g \lg p / \lg g$. Also note that in the QSM(g) model, any simulation of the CRCW PRAM(m) that requires sorting the memory locations accessed requires time at least $\Omega(g \lg p / \lg \lg p)$. Note that directly using the EREW PRAM simulation of a CRCW PRAM leads to a QSM(m) simulation with slowdown $\frac{p}{m} \lg m$.

In the PRAM(m), limited bandwidth does not affect the cost of distributing the input to the processors. However, we show that the difference between exclusive and concurrent read is not just a function of efficiently distributing the input to the processors. We show that if we add to the QSM(m) model a Read Only Memory containing the input, the simulation with slowdown $O(\frac{p}{m})$ is still close to optimal. The same result can be used to show a gap between the ER PRAM(m) and the CR PRAM(m) of $\Omega(\frac{p \lg m}{m \lg p})$, which, when $p \gg m$, greatly improves the best previous gap shown between the two models of $2^{\Omega(\sqrt{\lg p})}$ by [1].

Theorem 5.2 *The worst case time to simulate one step of the* CRCW PRAM*(m) on the* QSM*(m) is* $\Omega(\frac{p \lg m}{m \lg p})$.

We show that for the following problem, even if every processor in the QSM(m) model is given the entire input in advance, the CRCW PRAM(m) is faster than the QSM(m) by a factor of $\Omega(\frac{p \lg m}{m \lg p})$. The proof uses lower bound techniques developed in [2].

Definition 5.1 *The* **Leader Recognition** *problem.*

- *Input: p memory locations, one contains the value 1, and the rest contain the value 0.*
- *Output at each processor: the address of the memory location that contains the value 1.*

Let w be the number of bits contained in any memory cell. Note that the previously described simulation assumes that $w = \Omega(\lg p + \lg M)$, where M is an upper bound on the size of the shared memory. The theorem follows from the following lemma, and the fact that the leader recognition problem can be solved in the CRCW PRAM(m) in time $\lg p / w$, by every processor reading a distinct input cell, after which the one processor that finds a 1 broadcast its processor number to the remainder of the processors.

Lemma 5.3 *Any algorithm that solves the leader recognition problem in the* QSM*(m) requires average case time at least* $\Omega(\frac{p \lg m}{m w})$*, even if every processor knows the entire input in advance.*

The proof of the lemma, which uses techniques developed in [2], follows from the following claim:

Claim 5.4 *Any processor that solves the leader recognition problem either examines at least* $p \lg m / 2mw$ *inputs on average, or on average receives at least* $\lg m / 2$ *bits of communication from the other processors.*

♣ Make the sketch below a full proof.

Proof. (of claim – sketch) Consider any processor i that examines less than $\frac{p \lg m}{2m}$ inputs to the problem. We can model the actions of a processor by a decision tree, where all the initial branching in the decision tree occurs as a result of reading bits of communication (call this the *communication branching* portion of the tree) and the remainder of the branching occurs as a result of reading inputs to the problem (call this *input branching*). Call each maximal subtree that contains only input branching a *maximal input subtree*. In order to always respond correctly to the problem, processor i must produce at least p distinct results, and thus must have at least p leaves in this decision tree.

However, any input to this problem contains exactly one 1, and thus if a processor examines k inputs to the problem, the number of distinct leaves in any maximal input subtree is also at most k, since there is at most one leaf for each of the k different locations where the 1 in the input could appear. Therefore, the communication branching portion of the tree must partition the p inputs to the problem into sets of size at most k, and thus must contain at least p/k leaves. Thus, the number of communication bits must on average be at least $\lg(\frac{p}{k})$. The claim follows by using $k = p \lg m/2mw$. ∎

Proof. (of lemma) If there exists some processor that on average examines at least $p \lg m/2mw$ inputs to the problem, then the average running time of the algorithm must also be at least $p \lg m/2mw$. Otherwise, the average total number of communication bits read is at least $p \lg m/2$. Since at most wm bits can be read at any time step, at least $p \lg m/2mw$ steps are required. ∎

6 Sending unbalanced h-relations

In this section, we consider the general (unbalanced) routing problem, in which each processor has an arbitrary number of messages to send to other processors. We present three randomized parallel algorithms for scheduling an unknown, arbitrarily-unbalanced h-relation on a BSP(m) that are quite close to optimal; these algorithms handle the short messages, the varying-length messages and the dynamically-arriving messages cases. In irregular applications, processors can have varying amounts of messages to send due to skew in the inputs, skew in the fraction of data that is already local to the processor (e.g., sorting a nearly-sorted list or list-ranking a nearly-ordered list), skew in the amount of new values produced by the processors (e.g., an intermediate result of a join operation), skew in the number of new tasks spawned by the processors (e.g., in a nested parallel language), etc. Moreover, when $m \ll p$, the limited bandwidth of the BSP(m) makes the standard PRAM techniques for fast balancing of this skew unacceptably slow.

6.1 The Static Problem

We consider the following routing problem: Each processor i, $i = 1, \ldots, p$, has x_i messages to send. Let $n = \sum_{i=1}^{p} x_i$ and $\bar{x} = \max_{i=1}^{p} x_i$. Let y_i be the number of messages destined for processor i, and let $\bar{y} = \max_{i=1}^{p} y_i$. Each processor i knows x_i, but n, $\bar{x}$, y_i and $\bar{y}$ are unknown.

A lower bound for the problem on the BSP(g) is obtained directly from the definition of the problem, and a matching upper bound is obtained by a straightforward execution:

Proposition 6.1 *On the* BSP(g, L), *the routing problem takes* $\Theta(g(\bar{x} + \bar{y}) + L)$.

We devise simple algorithms for the BSP(m, L) that take $O(n/m + (\bar{x} + \bar{y}) + L(1 + \lg m/\lg L))$ with very high probability in m. We devise simple algorithms for the BSP(m, L) that take $O(n/m + (\bar{x} + \bar{y}) + L(1 + \lg m/\lg L))$ with very high probability in m. This algorithm is similar to techniques used for wormhole routing (see for example [22]).

Algorithm **Unbalanced-Send**

- Processors perform a prefix sum and a broadcast to inform every processor of the value n.
- For each processor i:
  - If $x_i \leq (1+\epsilon)n/m$ for some small ϵ, then processor i selects, uniformly at random, some j_i from the interval $[1, \ldots, (1+\epsilon)n/m]$. Let $j'_i = (1+\epsilon)n/m - j_i + 1$. Processor i sends $\min(x_i, j'_i)$ messages consecutively starting at time step j_i, and $\max(0, x_i - j'_i)$ messages consecutively starting at time step 1 (i.e., the messages are sent consecutively (mod $(1+\epsilon)n/m$) in the x_i locations starting at j_i).
  - If $x_i > (1+\epsilon)n/m$ then processor i sends all messages consecutively starting at time 1.

Theorem 6.2 *Let* $\sigma = \max((1+\epsilon)n/m, \bar{x}, \bar{y}) + O(L + L\lg m/\lg L)$. *Algorithm* **Unbalanced-Send** *completes in time* σ *with probability at least* $1 - e^{-\Omega(\epsilon^2 m)}$, *provided* $n < e^{\alpha m}$, *where* α *is a constant determined by the analysis. Also, for any* k, *the probability that algorithm* **Unbalanced-Send** *requires more than time* $k\sigma$ *is at most* $\frac{1}{k^4} \cdot e^{-\Omega(\epsilon^2 m)}$.

Proof. The prefix sum and broadcast require time $O(L + L\lg m/\lg L)$. The total number of sending steps required by the remainder of the algorithm is at most $\max((1+\epsilon)n/m, \bar{x})$, and thus it suffices to show that with probability at least $1 - e^{-\Omega(\epsilon^2 m)}$, no more than m messages are sent at every step of the algorithm. Since the number of processors with more than n/m messages to send can be at most m, this must be the case for any time slot larger than $(1+\epsilon)n/m$.

Consider a time slot τ, $1 \leq \tau \leq (1+\epsilon)n/m$. Let r_i be an indicator random variable that is a 1 if a message is being sent by processor i at time slot τ, and a 0 otherwise. The probability that a message from processor i is being sent at time slot τ is $\Pr[r_i = 1] = \rho_i = \min(1, x_i/((1+\epsilon)n/m))$. The expected number of messages being sent at time τ is hence $\sum_i \rho_i \leq n/((1+\epsilon)n/m) = m/(1+\epsilon)$. The r_is are mutually independent. By Chernoff Bounds, the number of messages at time τ exceeds m with probability at most $\exp(-\epsilon^2 m/3)$, and hence the number of messages at all times is at most m with probability $1 - (1+\epsilon)n/m \cdot \exp(-\epsilon^2 m/3)$.

We also show that for any k, the probability that algorithm **Unbalanced-Send** requires more than time $k\sigma$ is at most $\frac{1}{k^4} e^{-\Omega(\epsilon^2 m)}$. We see from the Chernoff Bound $\Pr\left[\sum r_i > (1+\delta)\mu\right] \leq (e/(1+e))^{\delta\mu}$, which holds whenever $\delta \geq e$, that for any l, and any time step τ, the probability that the number of messages sent does not exceed lm is at least $1 - e^{-\Omega(l\epsilon^2 m)}$. Such a time step requires time at most e^{l-1}. Thus, the probability that the algorithm requires more time than σe^l is at most $\sigma e^{-\Omega(l\epsilon^2 m)}$, which when $n < e^{\alpha m}$, is at most $(e^l)^{-4} \cdot e^{-\Omega(\epsilon^2 m)}$. ■

The exponential bound on the probability that the running time of the algorithm exceeds $k\sigma$ assures us that the expected running time is also $O(\sigma)$. The bound will also be used in our analysis of the dynamic problem. Note that the **Unbalanced-Send** algorithm can be easily adapted to any other sending pattern, such as if we insist on having a certain separation between every two messages sent by the same processor. We can use the same algorithm on any sending pattern "template", where the sending times are chosen by cyclically shifting the template by j slots. We elaborate on this further below.

We next consider a slight modification of the previous algorithm, called **Unbalanced-Consecutive-Send**. This algorithm has the advantage that it can be used in scenarios where a processor must send long messages using consecutive time steps. This is useful in the case where there are large message startup costs. This algorithm is the same as **Unbalanced-Send** with the modification that if $x_i \leq (1+\epsilon)n/m$, then processor i sends all its messages starting at time j_i.

Algorithm **Unbalanced-Consecutive-Send**

- Processors perform a prefix sum and a broadcast to inform every processor of the value n.
- For each processor i:
  - If $x_i \leq (1+\epsilon)n/m$ for some small ϵ, then processor i selects, uniformly random, some j from the interval $[1, \ldots, (1+\epsilon)n/m]$, and sends all its messages consecutively starting at time step j.
  - If $x_i > (1+\epsilon)n/m$ then processor i sends all messages consecutively starting at time slot 1.

Let X be the set of processors that have no more than $(1+\epsilon)n/m$ messages to send. Let $\bar{x'}$ be the maximum number of messages any processor in X has to send. The proof of the following theorem proceeds along the same lines as the analysis of the algorithm **Unbalanced-Send**.

Theorem 6.3 *Algorithm* **Unbalanced-Consecutive-Send** *completes in time* $\max((1+\epsilon)n/m + \bar{x'}, \bar{x}, \bar{y}) + O(L + L \lg m / \lg L)$ *with probability at least* $1 - e^{-\Omega(\epsilon^2 m)}$*, provided that* $n < e^{\alpha m}$*, where* α *is a constant determined by the analysis.*

The restriction about n being limited by $e^{\alpha m}$ can be replaced by a restriction on p instead, which may be more reasonable. To obtain this, the algorithm is slightly modified as follows. Let t' be the average over the x_is, that is, $t' = n/p$. For analysis purposes, let us "pad" each x_i that is smaller than t' to become equal to t'. This results with getting n at most twice its original value. Let c be some constant.

Algorithm **Unbalanced-Granular-Send**

- Processors perform a prefix sum and a broadcast to inform every processor of the value n.
- For each processor i:
  - If $x_i \leq n/m$ then processor i selects at random some j from the interval $0, \ldots, (cn/m - x_i)/t' - 1$, and sends all messages consecutively starting at time slot $j \cdot t' + 1$.
  - If $x_i > n/m$ then processor i sends all messages consecutively starting at time 1.

Theorem 6.4 *There exists a constant c such that algorithm* **Unbalanced-Granular-Send** *completes in time cn/m with probability at least* $1 - e^{-\Omega(\epsilon^2 m)}$*, provided that* $p < e^{\alpha m}$*, where* α *is a constant determined by the analysis.*

Proof. It is easy to verify that due to the new "granularity", for each processor i there are at most t' time slots (the last ones) that may not be used for sending messages. Therefore, for any time slot τ, $\tau = j\cdot t'+1$ for some j, the probability to have a message sent at τ by processor i is $\rho \leq x_i/(cn/m-t') \leq x_i/((1+\epsilon)n/m)$. This probability is the same for all τ's in $\tau, \ldots, \tau+t'$ (unlike the situation in the previous algorithm). We continue as before, showing that the expected number of messages is at most $m/(1+\epsilon)$ for τ, and are therefore more than m with probability at most $\exp(-\Omega(m))$. The number of events we now have to sum up over is $(c'n/m)/t' = (c'n/m)/(n/p) = c'p/m$, where $c' = (1+\epsilon)$. Therefore, the number of messages is at most m for all steps with high probability, provided that p is at most $e^{\alpha m}$. ∎

Finally, we mention (without providing details) two other settings in which a variant of **Unbalanced-Send** algorithm may be useful. First, we consider the situation in which a processor has messages of various sizes to be sent. The **Unbalanced-Send** algorithm is used with the following modification. If a certain long message of length ℓ is allocated with time $j, j+1, \ldots, (1+\epsilon)n/m, 1, \ldots, (1+\epsilon)n/m-1$, then instead we send it in time slots $j, j+1, \ldots, j+\ell-1$. The additive time factor due to this change is at most $\hat{\ell}$, which is an upper bound over the length of the various messages (some further improvements are possible here if the messages are of different lengths). This is a better bound than the $\bar{x}'$ additive bound obtained for the **Unbalanced-Consecutive-Send** algorithm.

A second situation, is where a certain gap, o, is required between two consecutive messages being sent. (Such gap may be associated with the overhead of initiating a message, as in the LOGP model.) Let $\bar{\ell}$ be the average size of a message over all messages, and let $n' = (1+o/\bar{\ell})$. A simple approach is to apply the variant described above for long messages where each message is now prepended with a dummy message of length o, and n is replaced by n'. The $(1+\epsilon)n/m+\hat{\ell}$ component is then replaced by $(1+\epsilon)(1+o/\bar{\ell})n/m+\hat{\ell}+o$. A better result may be obtained by a more careful refinement, which is not included here.

♣ Do we want to leave the last sentence?

6.2 The Dynamic Problem

We consider here the case of the unbalanced routing problem where the messages to be sent are introduced to the system dynamically over an infinite time line. We consider the case where the introduction times of the messages, as well as their destinations are determined by an Adversarial Queuing Theory model, as introduced in [15], and studied further in [4]. In this model, there is an adversary that injects messages into the system, but there is a set of *restrictions*, rules that the adversary must adhere to. We wish to find a protocol, if one exists, for sending the messages in the system such that for any adversary that adheres to the restrictions, the system is *stable*. By stable we mean that there is a value v, (which is allowed to depend on any parameters in the system except the time t), such that as t approaches ∞, the expected number of messages in the system at time t is bounded above by v.

We consider the following restrictions on the adversary. There is a parameter w, called the *window size*, a parameter α, called the *global arrival rate*, as well as a parameter β, called the *local arrival rate*. For any set of $L \geq w$ consecutive time steps, the adversary is allowed to inject up to $\lceil \alpha L \rceil$ point-to-point messages into the system. Furthermore, the adversary can request at most $\lceil \beta L \rceil$ of these messages to be sent from any given processor, and at most $\lceil \beta L \rceil$ messages can be destined for any given processor. Note that the adversary only informs the message source of the existence of a message. Also, note that the adversary is non-adaptive: the strategy taken by the adversary is allowed to depend on the algorithm, but not on any random choices made by the algorithm.

Consider for a moment the BSP(g) model.

Theorem 6.5 *For the* BSP *model where $g > 1$, if $\alpha \geq \beta > \frac{1}{g}$, then there exists an adversary where the system is not stable for any algorithm. However, if $\beta \leq \frac{1}{g}$, there is an algorithm that is stable for any adversary.*

Proof. For any locally-limited model, if $\alpha \geq \beta > \frac{1}{g}$, then the adversary can inject one message for some processor i to send every $\max(1, \frac{1}{\beta})$ time steps. If $g > 1$, then at time t, processor i has sent at most $\lceil \frac{t}{g} \rceil$ messages, the adversary has given processor i at least $\lceil \beta t \rceil$ to send. Thus, the number of messages that processor i has waiting to be sent at time t is at least $t(\beta - \frac{1}{g}) - 1$. Since $\beta - \frac{1}{g} > 0$, the system is not stable.

When $\beta \leq \frac{1}{g}$ we use the following algorithm for the BSP(g) model. We partition the time line into consecutive intervals of size $\max(g \cdot \lceil w/g \rceil, L)$. During the first interval, no sending occurs. The messages received during the ith interval are sent during the $i + 1$st interval in time simply by routing all the messages as a single h-relation. The adversary is constrained such that at most $(1/g)\max(g \cdot \lceil w/g \rceil, L)$ messages can arrive in any interval, and thus we can always route any messages that the adversary is allowed to request during a single interval in the time we have available. ∎

Corollary 6.6 *For any adversary that requests messages at a total rate higher than $\frac{p}{g}$, there is no stable algorithm.*

in the BSP(g) model, the global arrival rate only effects the algorithm if it is smaller than the local arrival rate. When we have a globally-limited model, on the other hand, both arrival rates become interesting. We show that in the globally-limited models, we can handle a much higher local arrival rate, while maintaining a global arrival rate that is within a constant factor of the optimal described above. For simplicity, we assume here that $w \geq L$. It is straightforward to incorporate the parameter L into the analysis for the more general case.

Theorem 6.7 *Let A be an algorithm that always solves the synchronous unbalanced routing problem when n is known, where A completes in time $\sigma = \max(a\frac{n}{m}, b\bar{x}, b\bar{y})$ with probability at least $1 - r$ for any $a, b \geq 1$, and any $r \leq 1$. Furthermore, for any k, the probability that A requires more than time $k\sigma$ is at most $\frac{1}{k^4}r$. There is an algorithm B for the* BSP*(m) which is stable for the dynamic unbalanced routing problem provided that $\alpha \leq \frac{m}{a} - \frac{mu}{wa}$ and $\beta \leq \frac{1}{b} - \frac{u}{wb}$, where $u \geq \lfloor 1.21rw \rfloor + 1$. Furthermore, the expected service time of any arrival is $O(w^2/u)$.*

Proof. We use the following algorithm B. We partition the time line into consecutive intervals of size w. During the first interval, no sending occurs. The messages that arrive during the ith interval are sent using algorithm A using the value $n = \lceil \alpha w \rceil$, starting at time $\max(t_1, t_2)$, where t_1 is the start of the $i + 1$st interval, and t_2 is the first time step after algorithm A completes sending the $i - 1$st interval.

We call an interval i *successful* if algorithm A, when sending the messages that arrive during i, requires time at most $w-u$. In each interval, the adversary is constrained so that for the routing problem that A must solve, $n \leq \frac{w-u}{a}$, $\bar{x} \leq \frac{w-u}{b}$, and $\bar{y} \leq \frac{w-u}{b}$. Thus, each interval i, regardless of the outcomes of previous intervals, is successful with probability at least $1-r$. The adversary has some control over the amount of time required to run algorithm A (and can thus create some dependence on previous outcomes), but is not able to cause the algorithm to run longer than time $w-u$ with probability more than r. We need to show that even when some of the intervals are not successful, the system is stable.

We show that the following equivalent system S is stable, where each arrival to a FIFO server corresponds to one set of messages received during an interval. There is one arrival to this queue every w time steps. The service time of this arrival may depend on both previous arrivals, as well as the arrival time, but given all previous arrivals, with probability at least $1-r$, any arrival has service time at most $w-u$. Furthermore, given all previous arrivals, the service time of any arrival is at most $k(w-u)$ with probability at least $1-r/k^4$.

Let S_1 be the distribution of the service time for the first arrival. Let S_i be the distribution of the service time for the ith arrival, given the service times of the 1st through $i-1$st arrivals. Let S_0' represent the distribution that takes on the value $w-u$ with probability exactly $1-r$, and, for any integer $k>1$, takes on the value $k(w-u)$ with probability exactly $r/(k-1)^4 - r/k^4$. We see that for any i, $S_0' \geq_{st} S_i$, where by $\geq_{st}$ we mean stochastically dominates. Thus, we can analyze the system S', which is identical to S, except that every service time is determined by drawing an independent sample from the distribution S_0'. The expected queue length at any time t in the system S' is at least as large as the expected queue length at time t in the system S. Thus, if S' is stable, then so is S. The theorem follows from the following claim. ∎

Claim 6.8 *S' is stable. Furthermore, the expected service of any arrival to S' is $O(w^2/2)$.*

Proof. We see that the expected number of intervals (of size w) that an arrival will be in the system is equal to the expected service time of an arrival in the following system S'': there is an arrival to a FIFO server at each step independently of all previous steps with probability r. The service time of this arrival is drawn from the distribution S_0'', which, for all integers $k \geq 1$ takes on value kw/u with probability $1/k^4 - 1/(k+1)^4$. To see that the two systems are equivalent, note that in system S' we do not change the amount of work at the FIFO queue by giving priority to any arrival with service time $w-u$. Thus, S' can be viewed as a system where the only arrivals are the ones which have service time at least $2(w-u)$, and each of these only receives u units of service out of every w time steps. The system S'' is equivalent to a scaled version of this system.

However, we see that the system S'' is just an $M/G/1$ queue, which is well known to be stable whenever the product of the service time and the arrival rate is strictly less than 1. We see that the arrival rate is r, and the expected service time is $\leq w/u \sum_{i=1}^{\infty} \frac{1}{i^3} < 1.21w/u$. For our choice of u, we have that $1.21wr/u < 1$.

Furthermore, we can now bound the expected service time of any arrival. The average queue size at customer departure instants is $r\bar{\mu} + \frac{r^2\bar{\mu^2}}{2(1-r\bar{\mu})}$, where r is the arrival rate, $\bar{\mu}$ is the expected service time, and $\bar{\mu^2}$ is the second moment of the service-time distribution (see for example [36]). In the system S'', this evaluates to approximately $\frac{2.42wru - 0.18w^2r^2}{2u^2 - 2.42wru}$. Thus, the expected time in the system of an arrival in the system S' is at most $2.42w^2/u + \frac{2.42w^2ru - 0.18w^3r^2}{2u^2 - 2.42wru}$. The expected service time in the system S is no higher. ∎

7 Conclusions

We have shown that for a number of basic problems, models that impose aggregate restrictions on bandwidth enjoy a considerable advantage over models that impose per-processor restrictions on bandwidth. In addition, we have demonstrated randomized on-line algorithms that schedule the transmission times of an arbitrarily-imbalanced set of messages in an aggregate bandwidth setting. These algorithms considerably improve upon the best possible results that can be achieved in models with per-processor restrictions. These results imply that it is important to use models that impose the type of restriction on bandwidth that most accurately reflects the machine in question. The routing results also indicate that the models with an aggregate bandwidth parameter, in most situations, can be replaced by the analogous model with the simplified cost metric discussed in Section 2.

Open problems include developing a further understanding of bandwidth as a resource. For example, it is interesting to note that in the QSM(m) model, the time bound that is analogous to linear time in sequential computation is $\frac{n}{m}$, the amount of time required to access entire input. Thus, for example, sorting an arbitrary set of real numbers can easily be performed in the QSM(m) model in "linear time." This may be much larger than the time bound analogous to "linear time" in classical PRAM algorithms: $\lg n$. An interesting open problem is to see if there exist problems which are considered hard for PRAM, in that no known $O(\lg n)$ PRAM algorithm exists, but turn out to be easy for QSM(m) in that they can be solved in time $O(\frac{n}{m})$.

Acknowledgements

We thank Torsten Suel for helpful comments.

References

[1] M. Adler. New coding techniques for improved bandwidth utilization. In *Proc. 37th IEEE Symp. on Foundations of Computer Science*, pages 173–182, October 1996.

[2] M. Adler, J. W. Byers, and R. M. Karp. Parallel sorting with limited bandwidth. In *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, pages 129–136, July 1995.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Sheiman. LogGP: Incorporating long messages into the LogP model — one step closer towards a realistic model for parallel computation. In *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, pages 95–105, July 1995.

[4] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proc. 37th IEEE Symp. on Foundations of Computer Science*, October 1996.

[5] Y. Azar. Lower bounds for threshold and symmetric functions in parallel computation. In *SIAM Journal of Computing*, volume 21(2), pages 329 – 338, 1992.

[6] D. A. Bader and J. JàJà. Practical parallel algorithms for dynamic data redistribution, median finding, and selection. In *Proc. 10th International Parallel Processing Symposium*, pages 292–301, April 1996.

[7] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. *IEEE Trans. on Parallel and Distributed Systems*, 6(2):154–164, 1995.

[8] A. Bar-Noy, J. Bruck, C.-T. Ho, S. Kipnis, and B. Schieber. Computing global combine operations in the multiport postal model. *IEEE Trans. on Parallel and Distributed Systems*, 6(8):896–900, 1995.

[9] A. Baumker and W. Dittrich. Fully dynamic search trees for an extension of the BSP model. In *Proc. 8th ACM Symp. on Parallel Algorithms and Architectures*, pages 233–242, June 1996.

[10] P. Beame and J. Håstad. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*, 36(3):643–670, July 1989.

[11] O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993.

[12] P. C.P. Bhatt, K. Diks, T. Hagerup, V. C. Prasad, T. Radzik, and S. Saxena. Improved deterministic parallel integer sorting. *Information and Computation*, 94:29–47, November 1991.

[13] S. N. Bhatt, G. Bilardi, G. Pucci, A. Ranade, A. L. Rosenberg, and E. J. Schwabe. On bufferless routing of variable length messages in leveled networks. *IEEE Trans. on Computers*, 45(6):714–729, 1996.

[14] R.H. Bisseling and W.F. McColl. Scientific computing on bulk synchronous parallel architectures. In *Proc. 133th IFIP World Computer Congress*, pages 509–514, 1994.

[15] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 376–385, May 1996.

[16] J. Bruck, C.-T. Ho, S. Kipnis, and D. Weathersby. Efficient algorithms for all-to-all communications in multi-port message-passing systems. In *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, pages 298–309, June 1994.

[17] T. Cheatham, A. Fahmy, D.C. Stefanescu, and L.G. Valiant. Bulk synchronous parallel computing – a paradigm for transportable software. In *Proc. IEEE 28th Hawaii Int. Conf. on System Science*, January 1995.

[18] S. Cook, C. Dwork, and R. Reischuk. Upper and lower bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 15:87–97, 1985.

[19] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proc. 4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming*, pages 1–12, May 1993.

[20] D. E. Culler, A. Dusseau, R. Martin, and K. E. Schauser. Fast parallel sorting under LogP: from theory to practice. In *Proc. Workshop on Portability and Performance for Parallel Processing*, Southhampton, England, July 1993.

[21] P. de la Torre and C. P. Kruskal. Submachine locality in the bulk synchronous setting. In *Proc. Euro-Par'96*, pages 352–358, August 1996.

[22] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992.

[23] A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized median finding on the BSP model. In *Proc. Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 223–232, June 1996.

[24] P. B. Gibbons, Y. Matias, and V. Ramachandran. Can a shared-memory model serve as a bridging model for parallel computation? In *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, June 1997. To appear.

[25] L. A. Goldberg and P. D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, pages 554–563, January 1996.

[26] M.T. Goodrich, Y. Matias, and U. Vishkin. Optimal parallel approximation algorithms for prefix sums. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 241–250, 1994.

[27] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards efficiency and portability: Programming with the BSP model. In *Proc. Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, June 1996.

[28] H. Gupta and P. Sadayappan. Communication efficient matrix multiplication on hypercubes. In *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, pages 320–329, June 1994.

[29] D. R. Helman, D. A. Bader, and J. JàJà. Parallel algorithms for personalized communication and sorting with an experimental study. In *Proc. Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 211–222, June 1996.

[30] S. Hinrichs, C. Kosak, D. R. O'Hallaron, T. M. Stricker, and Riichiro Take. An architecture for optimal all-to-all personalized communication. In *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, pages 310–319, June 1994.

[31] A. Jagota. A near-optimal algorithm for gossiping in a d-dimensional mesh bus interconnection network. In *Proc. 9th International Parallel Processing Symposium*, pages 331–337, April 1995.

[32] J. JáJá and K. W. Ryu. The Block Distributed Memory model. Technical Report UMIACS-TR-94-5, University of Maryland Institute for Advanced Computer Studies, College Park, MD, January 1994.

[33] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. on Computers*, 38(9):1249–1268, 1989.

[34] B H. H. Juurlink. Ph.D. Thesis, Leiden University, 1996.

[35] B. H. H. Juurlink and H. A. G. Wijshoff. The E-BSP Model: Incorporating general locality and unbalanced communication into the BSP Model. In *Proc. Euro-Par'96*, pages 339–347, August 1996.

[36] L. Kleinrock. *Queueing Systems Volume I: Theory.* John Wiley & Sons, 1975.

[37] C. E. Leiserson and B. M. Maggs. Communication-efficient parallel algorithms for distributed random-access machines. *Algorithmica*, 3(1):53–77, 1988.

[38] Y.-D. Lyuu and E. Schenfeld. Total exchange on a reconfigurable parallel architecture. In *Proc. 5th IEEE SYmp. on Parallel and Distributed Processing*, pages 2–10, December 1993.

[39] Y. Mansour, N. Nisan, and U. Vishkin. Trade-offs between communication throughput and parallel time. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 372–381, 1994.

[40] R. Miller. A library for bulk-synchronous parallel programming. In *Proc. of the British Computer Society Parallel Processsing, Specialist Group Workshop on General Purpose Parallel Computing*, December 1993.

[41] P. Raghavan and E. Upfal. Stochastic contention resolution with short delays. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 229–237, May-June 1995.

[42] S. Rao, T. Suel, T. Tsantilas, and M. Goudreau. Efficient communication using total-exchange. In *Proc. 9th International Parallel Processing Symposium*, pages 544–550, April 1995.

[43] Y.-C. Tseng and S. K. S. Gupta. All-to-all personalized communication in a wormhole-routed torus. *IEEE Trans. on Parallel and Distributed Systems*, 7(5):498–505, 1996.

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

[45] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A*, pages 943–972. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1990.

[46] U. Vishkin and A. Wigderson. Trade-offs between depth and width in parallel computation. In *SIAM Journal of Computing*, volume 14(2), pages 303 – 314, 1985.

[47] H. A. G. Wijshoff and B. H. H. Juurlink. A quantitative comparison of parallel computation models. In *Proc. 8th ACM Symp. on Parallel Algorithms and Architectures*, pages 13–24, June 1996.