Accounting for Memory Bank Contention and Delay in High-Bandwidth Multiprocessors

Guy E. Blelloch * Phillip B. Gibbons † Yossi Matias † Marco Zagha *

* School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{blelloch,marcoz}@cs.cmu.edu

† AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{gibbons,matias}@research.att.com

Abstract

This paper considers issues of memory performance in shared memory multiprocessors that provide a high-bandwidth network and in which the memory banks are slower than the processors. We are concerned with the effects of memory bank contention, memory bank delay, and the bank expansion factor (the ratio of number of banks to number of processors) on performance, particularly for irregular memory access patterns. This work was motivated by observed discrepancies between predicted and actual performance in a number of irregular algorithms implemented for the CRAY C90 when the memory contention at a particular location is high.

We develop a formal framework for studying memory bank contention and delay, and show several results, both experimental and theoretical. We first show experimentally that our framework is a good predictor of performance on the CRAY C90 and J90, providing a good accounting of bank contention and delay. Second, we show that it often improves performance to have additional memory banks, even beyond the natural choice of d banks per processor to compensate for a bank delay of d. Third, we explore scenarios under which high-level models, the EREW PRAM and QRQW PRAM, can be efficiently mapped onto high-bandwidth machines. We provide a work-preserving QRQW PRAM emulation, whose slowdown is a nonlinear function of the bank delay and the number of banks per processor. Finally, we evaluate the impact of contention on performance for several algorithms.

1 Introduction

Over the years processor speeds have been increasing much more rapidly than memory speeds. This divergence has motivated several computer manufacturers to design memory systems with many more memory banks than processors (see Table 1) and also memory manufacturers to design memory modules with multiple internal banks [TTT+94]. When used in conjunction with latency hiding techniques (*e.g.* vectorization, multithreading, prefetching) and a high-bandwidth network, supplying more banks than processors provides an effective way to support shared memory access with relatively slow inexpensive (commodity) memory parts.

In this paper we are interested in systematically studying the performance effects of having multiple slow memory banks in shared memory machines.

Our goal is to use a simple model for such machines that accounts for the slow access rate of memory banks on the one hand, and for the expanded number of memory banks on the other hand. For this purpose we extend Valiant's bulk-synchronous parallel (BSP) model [Val90a] with two additional parameters: (i) the *bank delay*, d, which is the number of cycles between accesses to a memory bank, and (ii) the *expansion factor*, $\mathbf{x}$, which is the ratio of the number of memory banks to the number of processors. We denote this model as the $(d, \mathbf{x})$-BSP (the "deluxe" BSP). The research presented in this paper was motivated by the need to explain observed discrepancies between predicted performance based on previous models and actual performance in a number of programs written for the CRAY C90 [BHZ93, Zag95].

Machine	Procs (p)	Banks (B)	Expansion ($\mathbf{x}$)
NEC SX-3	4	1024	256
Tera MTA	256	512×64	128
Cray C90	16	1024	64
Cray J90	16	512	32
Convex C4	4	128	32
Meiko CS-2 node	2	16	8
SGI Power Challenge	18	8×8	3.5

Table 1: Expansion in the number of memory banks for various machines. In the current Tera design, memory banks are organized into memory modules containing 64 memory banks. The SGI Power Challenge memory is arranged as up to 8 modules with 8 banks each. The Meiko CS-2 node contains two Fujitsu μ-VP vector processors.

Based on the $(d, \mathbf{x})$-BSP model we show a number of results, both experimental and theoretical. We are particularly interested in cases where the best sustainable gap, g, between memory requests issued by each individual processor is less than the best sustainable delay, d, between accesses to each individual memory bank.

We first show experimentally that the $(d, \mathbf{x})$-BSP quite accurately models the CRAY vector multiprocessors in many situations, even though the model ignores details of the machines. We show this both for the CRAY C90, which uses static RAM (SRAM) and has a bank delay of 6 clock cycles, and for the more modestly priced CRAY J90, which uses dy-

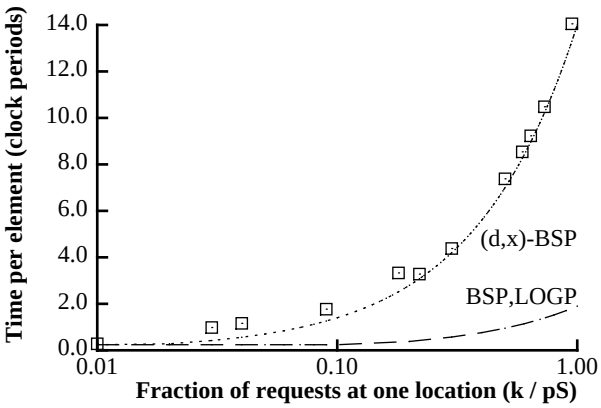

Figure 1: Predicted and measured performance on a set of memory access patterns extracted from a trace of Greiner's algorithm for finding the connected components of a graph [Gre94]. Measured times on an 8 processor CRAY J90 for several patterns are shown with squares. Predicted times are given both for the BSP and the $(d, \mathbf{x})$-BSP as a function of contention.

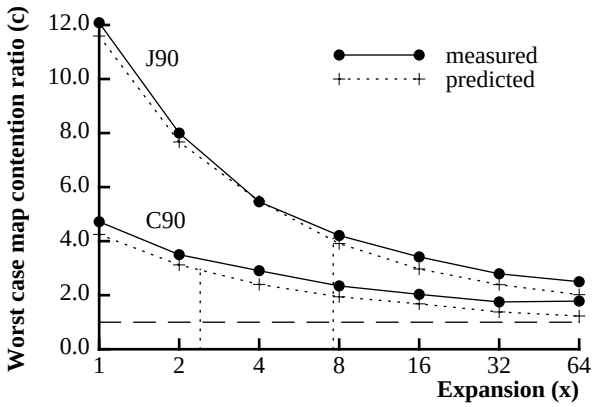

Figure 2: The ratio of the time that includes the effect of multiple memory locations being mapped to the same bank to the time that excludes the effect, when using random mapping. This is given as a function of expansion and is for a worst-case reference pattern. For both the CRAY C90 and J90 the actual expansion is 64 and the other data points are taken by using a subset of the banks. This shows the importance of having an expansion greater than the d/g that is needed to match the servicing bandwidth at the banks with the issuing bandwidth at the processors (shown with the dotted vertical lines).

namic RAM (DRAM) and has a bank delay of 14 clock cycles. The $(d, \mathbf{x})$-BSP has made it easier to analyze several algorithms and has allowed us to predict various effects that cannot be predicted without taking into account the bank delay. For example, Figure 1 and later Figure 12 compare predicted and measured times for an implementation of a graph-connectivity algorithm [Gre94] and a sparse-matrix vector multiplication [BHZ93] algorithm. These times are compared with the predictions based on models that do not account for the bank delay (*e.g.* the BSP or LOGP). Although the $(d, \mathbf{x})$-BSP does not model the time exactly, due mostly to ignoring effects of the network, it more accurately captures the effect of high contention. Furthermore, the discrepancy between the BSP and $(d, \mathbf{x})$-BSP becomes larger as either the bank delay or the number of processors increases (the experiment shown is for only 8 processors). All predictions for the CRAY assume that the code is fully vectorized so that memory traffic is high.

Second, we study to what extent we can ignore the effects of multiple memory locations residing in a single bank when using pseudo-random mappings of memory locations to memory banks. Many researchers have studied the effect of randomly mapping memory to banks (*e.g.* [MV84, KU86, Ran91, Val90a, ACC+90, RH90, Rau91, IC93]). If there is sufficient parallel "slackness" (extra parallelism) so that each bank is receiving multiple requests, it has been shown [MV84, KU86, Ran91, Val90a] that with high probability the memory references will be reasonably balanced across the banks. These results, however, assume that there is no contention at individual memory locations.[1] If we allow for contention at memory locations then ignoring the mapping of memory locations to banks can significantly underpredict the running time even for unbounded slackness. This is because even when the number of memory requests is large, the number of accessed locations could be small and not well balanced across the memory banks. We show that having a high expansion factor can alleviate this problem to some extent. For the CRAY C90, which has a high expansion factor, we show both experimentally and analytically that ignoring the mapping will underpredict running time by a factor of about 2.0 for the worst case reference pattern and typically by much less. The effect of expansion is shown in Figure 2.

Third, we explore scenarios under which high-level models for algorithm design, the EREW PRAM (*e.g.* [JáJ92]) and the stronger QRQW PRAM [GMR94b] can be effectively mapped onto high-bandwidth machines (small g) when memory bank delay is accounted for. For the case $\mathbf{x} \leq d$ we observe that $(d/\mathbf{x})$ is an inevitable work overhead, and provide an emulation of the QRQW PRAM on the $(d, \mathbf{x})$-BSP in which the overhead matches this factor; the emulation, as well as its analysis, follow and generalize the QRQW PRAM emulation onto the BSP given in [GMR94a]. For the case $\mathbf{x} \geq d$, we present a work-preserving emulation of the QRQW PRAM on the $(d, \mathbf{x})$-BSP, assuming high bandwidth, where the effect of d on the slowdown is partially compensated for by the expansion factor $\mathbf{x}$; the analysis for this emulation requires a somewhat different approach. The emulation assumes a random mapping of memory locations to memory banks, but it is often good enough to randomize the order of the data at the beginning of an algorithm.

Finally, we experiment with four algorithms: a QRQW binary search algorithm, a QRQW random permutation algorithm, a sparse matrix multiply, and a CRCW connected components algorithm. The $(d, \mathbf{x})$-BSP model is used to predict the running time of each algorithm.

An important part of this research is to question the assumption made by most of the existing abstract models of parallel machines (*e.g.* BSP, LOGP, DMM, Hypercube, Butterfly) that it is adequate to assume that the number of memory modules is equal to the number of processors; this paper studies the effect of relaxing this assumption.

Related Work: There have been several studies of memory bank contention on vector multiprocessors [OL85, CS86,

[1] In the case of Ranade's work [Ran91] it is assumed that references to a single location are combined in the network.

Cal88, Cal89]. These studies include more detailed models of the memory system and concentrate on constant stride access patterns. We are interested in having as simple a model as possible that captures the important effects but is independent of most machine-specific details. We are also interested in arbitrary access patterns. In particular we are interested in patterns that appear in irregular (unstructured) applications such as sparse matrix or pointer-based algorithms.

Bailey [Bai87] studied the number of additional memory banks needed to compensate for increases in the bank delay. His analysis is in the context of a lightly-loaded system where a processor may have at most one request outstanding and at most one request is ever waiting at a bank. In contrast, we consider scenarios closer to peak utilization.

2 Accounting for memory bank contention and delay

Figure 3 shows the basic model we assume. As with the Valiant's BSP model a computation consists of a sequence of supersteps with bulk synchronization between them. In each superstep the processors can perform local computations and make a set of global memory requests. Unlike the BSP model, the global memory requests are serviced by a separate set of memory banks, which are used as a shared memory by the processors. In practice these memory banks might be physically located next to the processor, but it is important that the processors not be involved in handling the memory requests. Processors also have local memory for use with local operations. We include the same parameters as included in the BSP model but add two more: the *memory delay* and the *bank expansion factor*. The parameters of the $(d, \mathbf{x})$-BSP are summarized as follows:

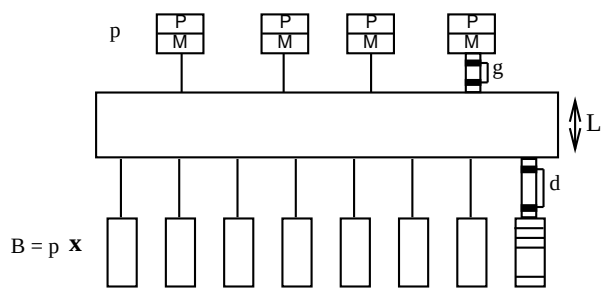

Figure 3: The $(d, \mathbf{x})$-BSP model.

p	number of processors
g	cycles between memory requests at a proc. (gap)
L	the periodicity factor (latency + synchronization)
d	cycles between memory accesses at a bank (delay)
$\mathbf{x}$	memory bank expansion factor

where the number of memory banks, denoted as B, is $\mathbf{x} \cdot p$.

The time of a superstep depends on three factors: the local work, the maximum number of memory requests made by a processor, and the maximum number of requests handled by a bank. To account for these we use the following definitions. W_i is the amount of local work performed by processor i, S_i is the number of global memory requests made by processor i, and R_j is the number of requests handled by memory bank j. Let $W = \max_{i=1}^{p} W_i$ (maximum local work), $S = \max_{i=1}^{p} S_i$ (maximum memory request load at the processors) and $R = \max_{j=1}^{\mathbf{x} \cdot p} R_j$ (maximum load at the memory banks). Based on the definitions, the cost of a step in the model is

$$T = \max(W,\ g \cdot S,\ d \cdot R,\ L)$$

We note that the cost model is basically the same as the BSP when $\mathbf{x} = 1$ and $d = g$. We refer to a machine as $(d, \mathbf{x})$-*balanced* if $\mathbf{x} = d/g$. This is the point where the total bandwidth available at the processors and network matches the total bandwidth available at the memories. Although we have chosen the BSP model to extend it should be straightforward to extend other related models, such as the LOGP [CKP+93] or DMM [MV84] models, with the d and $\mathbf{x}$ parameters. To extend the LOGP it is assumed that the banks are separate modules from the processors.

The contention at a memory bank can be caused both by contention at a particular location in the bank and by writing to multiple locations within one bank. In the basic model we make no assumptions about how the memory locations are mapped onto the memory banks. This allows us to consider both scenarios where the mapping is under user control and where the mapping is random. To separate the two types of contentions we make the following definitions, each of which is defined relative to a single superstep. Let the *memory request contention* (k_i) be the number of requests to location i, and $k = \max_i k_i$. Let M_j be the set of locations mapped to bank j. The size of this set is *module map contention* ($\mu_j = |M_j|$) of a given bank, and $\mu = \max_j \mu_j$ (see Figure 4). Then

$$R_j = \sum_{i \in M_j} k_i \ .$$

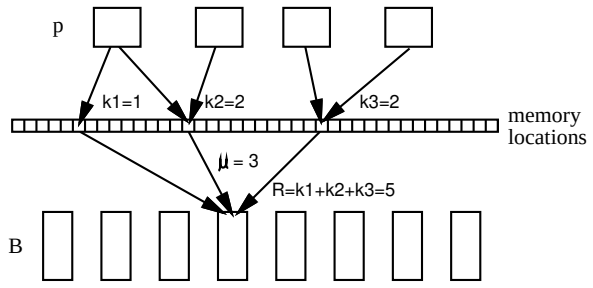

Figure 4: Memory contention (k), module map contention (μ), and module load contention (R).

3 Case study: modeling the Cray

This section presents a qualitative and quantitative comparison of the CRAY C90 and CRAY J90 to the $(d, \mathbf{x})$-BSP. The experiments in this section provide evidence that the abstract model can produce realistic predictions of running time despite ignoring many architectural details, such as the use of vector processors, the topology of the interconnection network, the flow control for the network, and the priority scheme on the banks. The experiments also show some cases where the model breaks down. The features of the CRAY vector machines that make them suitable for the model are

1. A high bandwidth multistage interconnection network between the processors and memory that supports fine grained memory references.

	C90	C90*	J90
Processors	16	16	16
Banks	1024	512	1024
Memory	SRAM	SRAM	DRAM
Clock period	4.2 nsec	4.2 nsec	10.0 nsec
g	1.2	2.5	1.8
d	6	6	14

Table 2: The parameters for the CRAY C90 and CRAY J90. The gap and delay are measured in clock periods. The C90* is the configuration of the C90 available to us at the Pittsburgh Supercomputing Center, which has only half the memory banks and network of a full configuration.

2. Memory banks that are significantly slower than the rate at which processors can issue memory requests. The ratio d/g is 5 on the C90 and over 7 on the J90.

3. A memory latency hiding technique for arbitrary memory access patterns.

Here we discuss these features in more detail. The instruction set supports vector load and store instructions, which load or store up to 128 words per instruction (64 on the CRAY J90). These instructions can either be strided for regular access patters, or can be based on a vector of addresses for indirect addressing. In the indirect loads and stores, often called *gather* and *scatter*, each address specifies the location of a single 64-bit word allowing for fine-grained memory accesses. The processors are connected to the memory banks with a multistage network. In the largest memory configuration of the CRAY C90, banks are divided into 8 "sections" at the first level, which are further subdivided into 8 "subsections", each of which contains 16 memory banks. Each processor has independent paths through the section and subsection, but processors can interfere with each other due to bank conflicts. Sequential memory locations are interleaved across memory banks, making regular, strided memory access fast (except on strides which are large powers of two).

The latency through the network is about 9 clock cycles each way. When added to the access time at the memory bank, the total latency for a load is between 23 and 35 clock cycles, depending on the model and assuming no contention at the memory bank. This latency is usually hidden when using vector loads and stores since each load or store requests multiple locations which are pipelined through the network. Furthermore, vector memory references can overlap so that one batch of 128 loads or stores can start before the previous has finished. For strided accesses patterns it is possible to get a throughput of better than one memory access per clock cycle since there are actually multiple pipelines that operate in parallel (2 on the J90 and up to 3 dual pipelines on the C90). Only one of these, however, can be used for a gather or scatter instruction.

Table 2 shows the $(d, \mathbf{x})$-BSP parameters for the CRAY C90 and CRAY J90. The gap g is measured experimentally based on the time for a gather to random locations. The other values are available from the machine specifications.

We have performed several experiments of memory access patterns to quantitatively compare the predictions given by the $(d, \mathbf{x})$-BSP with running times on the CRAY C90 and CRAY J90. The experiments were selected to test various aspects and extremes of the model. Figure 5 summarizes the experiments. For most experiments the measured times closely match the predicted times. For one of the experiments, 4c, the numbers are off by up to a factor of 2.5 due to the effects of the network, which are discussed. In all our experiments we assume that the work term W can be ignored since on the CRAY it is typically subsumed by the $g \cdot S$ term (each processor has a very small local memory, just a set of registers, making it hard to take account of locality). For all experiments we randomize the injection of memory requests within the processors. All the experiments are based on using the scatter operation although experiments with the gather operation give almost identical results. The patterns we are interested in cannot be created with strided access—timings for various strided access patterns can be found elsewhere [CS86, Soh93]. All experiments were run on a dedicated 8 processor system. All graphs are for the CRAY J90 except where noted—CRAY C90 results are qualitatively similar. For all experiments $S = 64K$ and L is negligible.

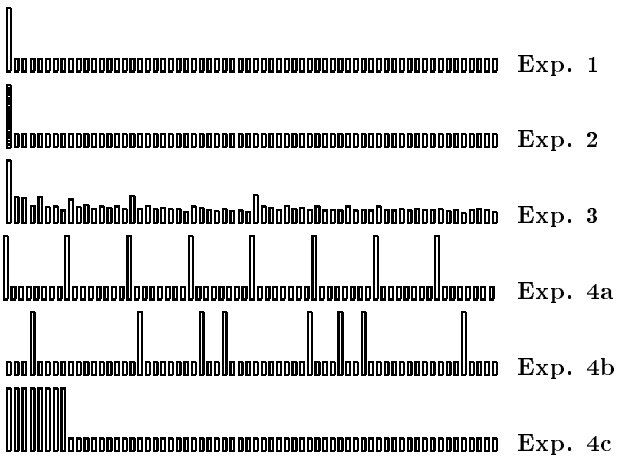

Figure 5: Summary of the experiments. Each bar represents the load on one memory bank. A shaded bar represents multiple different locations being written to the bank while a clear bar represents a single location being written.

Experiment 1: The first of the experiments is used to verify the $(d, \mathbf{x})$-BSP time equation $T = \max(g \cdot S, d \cdot R, L)$ over a range of R. One aspect we were interested in checking was whether the time would be better modeled by adding the memory and processor terms (*i.e.* $T = g \cdot S + d \cdot R + L$) rather than taking the maximum. The experiment consisted of writing one location with load R; the remaining work is spread across $B - 1$ memory locations, one memory location per remaining bank. R is varied and S is kept constant. The result of the experiment is shown in Figure 6, and it verifies that for the CRAY using max is the better model.

Experiment 2: The second experiment is used to verify that the time is determined by the maximum contention at a bank, independent of the distribution of requests within a bank. The experiment consisted of sending a single request to R different locations within a single bank; the remaining work is spread across $B - 1$ memory locations, one memory location per remaining bank. Again, R is varied and S is kept constant. This differs from the previous experiment

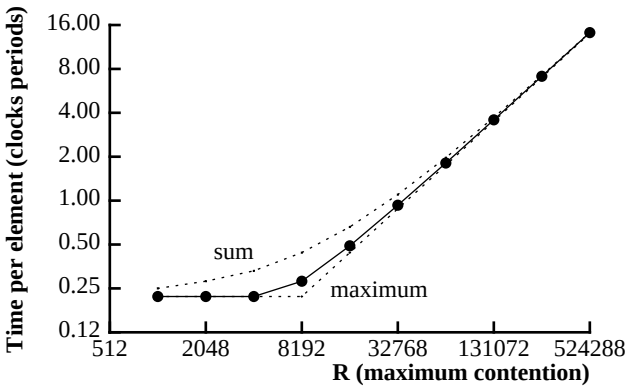

Figure 6: Experiment 1: Measured and predicted times on the CRAY J90 over a range of contentions. The measured time (shown with a solid curve) is very close to the maximum of the time spent at the processors and memory. The knee in the curve is where the dominant term switches.

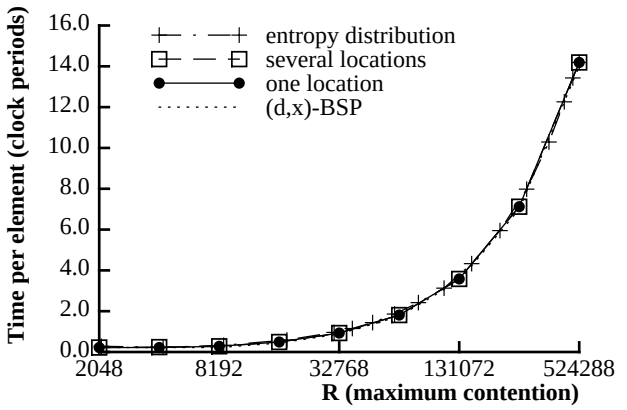

Figure 7: Comparison on the CRAY J90 of experiment 1, experiment 2, and experiment 3: one measuring the time where each bank contains at most one active memory location, one where the accesses to each bank are spread over many memory locations, and one using successive ANDings of random keys. As expected, the curves are nearly identical.

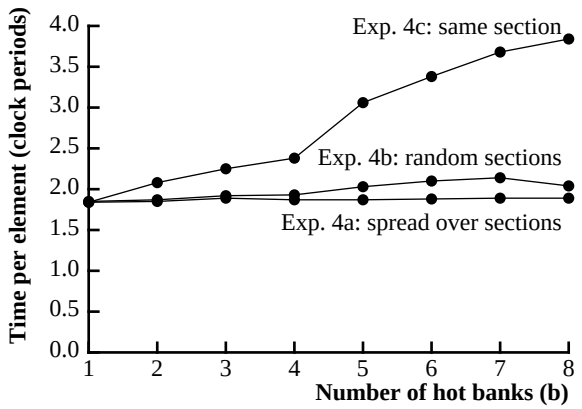

Figure 8: Time as a function of the number of hot banks when R is constant. The three curves are for different distributions of hot banks across the network and show the effect of the network contention.

only in that the load on the "hot" bank is due to multiple memory locations rather than multiple elements to the same location. The results shown in Figure 7 verify that the performance is not affected by whether the R term is dominated by location contention or module map contention.

Experiment 3: To verify that the running time can be accurately predicted for less regular distributions of memory accesses, we constructed an experiment using the entropy distributions suggested by Thearling and Smith [TS92]. The distributions are generated by starting with a set of random keys and then bitwise ANDing together each key with another key selected at random. Iterating this process generates a family of distributions each with a higher contention than the previous and eventually all keys become 0. The experiment was run over the whole family. Figure 7 shows the time as a function of this maximum contention. The predictions are slightly less accurate than those in Experiments 1 and 2, but are still within 30% of the measurements.

Experiment 4: To verify that the running time is determined by the maximum contention R, independent of the average contention and distribution of contention, we send R requests to one location within each of b banks and send an equal portion of the remaining work to $B - b$ locations all in different banks. R and S are kept constant, while b is varied. We tried three versions of this experiments differing in how the high-contention banks are distributed across the network: (a) evenly distributed across the network, (b) randomly distributed across the banks, and (c) all within the same section of the network. Figure 8 shows the results of the experiment. Versions (a) and (b) are quite close to the predicted performance. Version (c), however, is up to a factor of 2.5 off from the prediction because of congestion at one of the *subsections* of the network. A more refined model would be needed to take account of this [ST91], but the experiment shows that even in what we expect to be the worst case the predictions are not catastrophic. Note that when memory is mapped at random into banks, an issue that is discussed in the next section, the situation described in version (c) is unlikely.

4 Using random memory mappings

Randomly mapping memory locations into banks is a standard technique to reduce module map contention (contention due to multiple memory locations being mapped to the same bank) in simulations of shared memory on machines with a fixed set of memory modules (see, e.g., [MV84, KU86, Ran91, Val90a, ACC+90, RH90, Rau91, IC93]). The primary advantage is that it ensures that concurrently requested memory locations will likely be distributed evenly across the banks. In this section we study to what extent we can ignore the module map contention (μ) when randomly mapping memory to banks. In particular we consider the ratio of the time including module map contention to the time excluding it. We call this ratio the *map contention ratio* (c), and in the $(d, \mathbf{x})$-BSP it can be expressed as

$$c = \frac{\max(W,\ g \cdot S,\ d \cdot R,\ L)}{\max(W,\ g \cdot S,\ d \cdot k,\ L)} \qquad (1)$$

We are interested in how bad this ratio is for various machine parameters (p, g, d, $\mathbf{x}$) and memory access patterns. We show that for the CRAY, which has a reasonably high

expansion factor, c is small. The results in this section are generalized in the context of the QRQW simulation in the next section.

To derive an equation that bounds the map contention ratio we consider the worst case memory access pattern. We claim that for a given k (maximum contention at any memory location) the worst case pattern is when all memory locations that are being accessed have the same contention. We call this a *uniform distribution* of requests and assuming each processor is making S requests, then a total of $m = Sp/k$ locations are being accessed. We are interested in the map contention ratio as a function of m—the worst ratio over m will give us the worst overall ratio. If we assume the cost is dominated by the send and contention terms we can simplify equation 1 to

$$c(m) = \frac{\max(mz, \mu)}{\max(mz, 1)}$$

where $z = g/dp$; z is a constant of the machine. The module map contention μ can be expressed in terms of the function $Urn(m, n)$, which is the expected maximum sized bucket when throwing m balls into n buckets at random. More specifically, let m balls be thrown independently at random into n urns, and let x be the random variable representing the maximum number of balls in any one urn; then, $Urn(m, n) = \mathbf{E}(x)$. We can estimate:

$$c(m) \approx \frac{\max(mz, Urn(m, B))}{\max(mz, 1)} \qquad (2)$$

Figure 9 shows both the predicted and measured values of c as a function of m for the CRAY C90 and the CRAY J90. The measured ratio is based on the average over 20 trials. The results show that for all m, c is at most 2.0 on the C90 and 2.5 on the J90, and for most patterns it is close to 1. This argues that for many practical purposes we can ignore the module map contention when using random mappings on the CRAY—inaccuracies in predictions from other sources are likely to dominate. Intuitively, the reason for the peak in the graph is that as we increase the number of "hot" locations (m) past the peak the load at each location decreases and eventually the first term in the equation $\max(g \cdot S,\ d \cdot k)$ dominates. As we decrease m below the peak it becomes less likely that multiple locations will be mapped to the same bank. In particular when $m < \sqrt{B}$ it becomes less likely that more than one location will be mapped to a single bank. The slight difference between the predicted and measured times is due to effects in the network as discussed in the previous section. In particular, at these small m it is reasonably likely that the locations are not only imbalanced across the banks, but are also imbalanced across the *sections* in the network causing backup at the source due to section conflicts.

What is more interesting is to see how the the worst case c is affected by the machine parameters. Equation 2 is maximized when $mz = 1$ (*i.e.*, $m = dp/g$), giving

$$c_{\max} = Urn\left(\frac{1}{z}, B\right) = Urn\left(\frac{dp}{g}, p\mathbf{x}\right) \quad .$$

Figure 2 shows $c_{\max}$ as a function of $\mathbf{x}$ for d and g set to the parameters of the CRAY C90 and CRAY J90 and p set to 16. As can be seen, it is helpful to have an expansion factor beyond the $(d, \mathbf{x})$-balanced ratio ($\mathbf{x} = d/g$).

The TERA design is planned to provide hardware support for hash functions to be used for pseudo random mapping of memory locations to memory banks. The CRAY does not supply hardware to perform the random mapping of the memory locations to banks. For some algorithms, however, it is possible to get the same effect by randomly permuting the input and some of the intermediate results. In others, the nature of the algorithm results in random mapping without any additional steps (see examples in Section 6). For other algorithms computing a pseudorandom hash in software is not prohibitive.

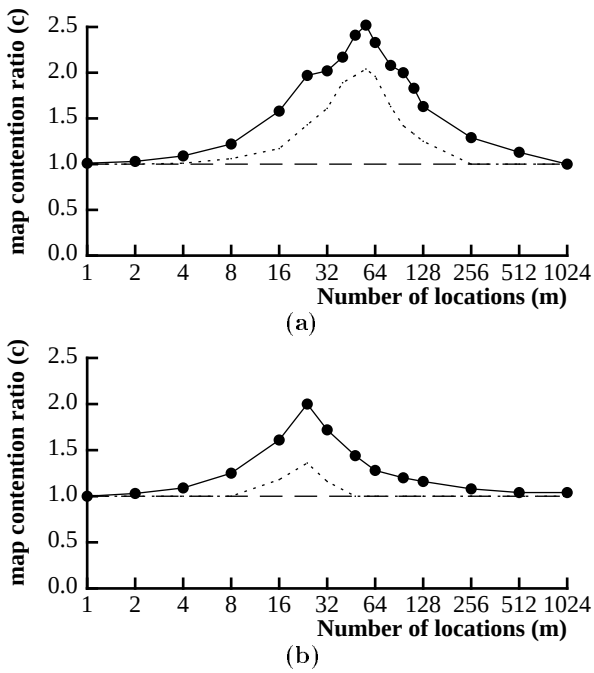

Figure 9: The measured and predicted map contention ratio c for J90 (a) and C90 (b). The measured is taken as the ratio of the measured running time on the CRAY to the equation $\max(g \cdot S,\ d \cdot k)$, which ignores module map contention (solid curves). The predicted is given by Equation 2. This is for a uniform distribution to m locations using random mappings from locations to banks. S is kept constant. $p = 8$.

The actual evaluation costs for a variety of hash functions on the CRAY C90 are given in Table 3. When hash functions are used for pseudo-random mapping of memory locations to memory banks, it is important that they exhibit favorable properties for any given input (*i.e.* that they are "universal"). The function h_a^1, which is called the *multiplicative hashing scheme* in [Knu73, p. 509], was recently shown by Dietzfelbinger *et al.* [DHKP93] to be 2-universal in the sense of Carter and Wegman [CW79]: for any two distinct numbers $x, y \in [0..2^u - 1]$, $\mathbf{Prob}\left(h_a^1(x) = h_a^1(y)\right) \leq 1/2^{m-1}$; *i.e.*, the collision probability is approximately the same as for a random mapping. We note that the study of map contention ratio discussed above assumed random mapping, and should be re-examined when selecting a hash function, taking into account the properties of the particular candidate function. Thus, the choice of a hash function may be influenced by several parameters, including its degree of universality, its evaluation cost, and its congestion behavior, both theoretically (see [DGMP92]) and experimentally (see [EK93]).

Hash Function	T/n
Linear	
$h^1_{a,b}(x) = ((ax+b) \bmod 2^u) \operatorname{div} 2^{u-m}$	1.8
$h^1_a(x) = (ax \bmod 2^u) \operatorname{div} 2^{u-m}$	1.8
Quadratic	
$h^2_{a,b,c}(x) = ((ax^2+bx+c) \bmod 2^u) \operatorname{div} 2^{u-m}$	3.4
$h^2_a(x) = ((ax^2) \bmod 2^u) \operatorname{div} 2^{u-m}$	3.4
Cubic	
$h^3_{a,b,c,d}(x) = ((ax^3+bx^2+cx+d) \bmod 2^u) \operatorname{div} 2^{u-m}$	6.7
$h^3_a(x) = ((ax^3) \bmod 2^u) \operatorname{div} 2^{u-m}$	4.9

Table 3: The evaluation cost of various hash functions in terms of clock cycles per element (for each CRAY C90 processor). The functions map items x from the domain $[0..2^u]$ into the range $[0..2^m]$, and a, b, c, and d are odd numbers selected at random from $[1..2^u-1]$.

5 High-level programming model

In this section, we explore scenarios under which a high-level model for algorithm design, the QRQW PRAM [GMR94b], can be effectively mapped onto high-bandwidth machines. In the context of this section, we assume that high-bandwidth machines have a small g. When $\mathbf{x} \geq d$, we provide a work-preserving emulation of the QRQW PRAM on the $(d, \mathbf{x})$-BSP, whose slowdown is a nonlinear function of d and $\mathbf{x}$. For example, the slowdown is minimized when $\mathbf{x} \geq d \cdot 2^d$, in which case d is only an additive term in the slowdown. On the other hand, when $\mathbf{x} < d$, no work-preserving emulation exists, and we provide a emulation whose work bounds match the lower bound that we prove. All of these results apply as well to the EREW PRAM, as a special case of the theorems.

The QRQW PRAM allows for concurrent reading and writing to shared memory locations, but at cost proportional to the number of readers/writers to any one memory location. In particular the charge is based on the maximum contention at any location. It was argued in [GMR94b] that neither the exclusive nor the concurrent rules for memory access accurately reflect the contention capabilities of most commercial and research machines, and that the *queue* rule, used in the QRQW cost metric, is more appropriate. The QRQW PRAM model allows processors to send multiple memory requests at each parallel step; this enables processors to pipeline memory requests while they are delayed at memory locations due to contention. The charge for a step is the maximum of the number of requests sent by any one processor and the number of requests to any one location, *i.e.* $\max\{S, k\}$.

As with other PRAM models (e.g., CRCW, EREW), the QRQW PRAM abstracts away many features of real machines, including the latency or delay in accessing the shared memory, the cost of synchronizing the processors, and the layout of memory locations at memory banks. In the next section, we discuss some experimental results regarding the implementation of two QRQW PRAM algorithms from [GMR94a] on the CRAY. The experimental results supplement the general simulating results presented in this section in demonstrating that in many cases, the two metrics of the high-level QRQW PRAM model are sufficient to accurately predict performance on the CRAY.

5.1 Simulating EREW and QRQW PRAM on $(d, \mathbf{x})$-BSP

We present efficient emulations of the QRQW PRAM model on the $(d, \mathbf{x})$-BSP. These results clearly apply also for the weaker EREW PRAM. First, we observe an inherent overhead in work of the simulation algorithm, when $\mathbf{x} < d$. Theorem 5.1 provides a simulation in which the overhead is minimized. Finally, we provide in Theorem 5.2 an optimal work simulation, for the case that $\mathbf{x} \geq d$. Recall that one step of a QRQW PRAM with contention k takes k units of time.

Observation 1 *Let $p' \geq p$. Any simulation of one step of a p'-processor* QRQW PRAM *with maximum memory request contention k on a p-processor high bandwidth $(d, \mathbf{x})$-*BSP *requires $\max\{k(p'/p) \cdot (d/\mathbf{x}), kd\}$ time.*

Proof. Consider a uniform memory request contention of k requests to each of $m = p'$ locations. The average module map contention is $\overline{\mu_j} = m/B = \frac{p'}{\mathbf{x}p}$. (Note that $\overline{\mu_j}$ may be less than 1.) Since we consider uniform memory request contention, $R_j = \mu_j k$ and therefore $R \geq \overline{\mu_j} k = \frac{p'k}{\mathbf{x}p}$. Also, by definition, we have $R \geq k$. Therefore, the time on the $(d, \mathbf{x})$-BSP is at least $Rd \geq \max\left\{\frac{p'dk}{\mathbf{x}p}, kd\right\}$. ∎

The following two theorems are generalizations of the QRQW PRAM emulation on a BSP given in [GMR94b]. Theorem 5.1 handles the case where $\mathbf{x} \leq d$; the proof follows the one given for the BSP simulation, and is given in the appendix of this extended abstract. Theorem 5.2 handles the case of larger $\mathbf{x}$; its proof requires a somewhat different approach.

Theorem 5.1 (QRQW simulation with small x) *Let $1 \leq \mathbf{x} \leq d$, and let $\mathbf{x} \leq p^{\bar{c}}$ for some constant $\bar{c} > 0$. Let $\tau = \mathbf{x} \lg p + d + L$. One step of a τp-processor* QRQW PRAM *with maximum memory request contention k can be emulated on a p processor high bandwidth $(d, \mathbf{x})$-*BSP *in time $O(k\tau d/\mathbf{x})$ w.h.p.*

Theorem 5.2 (work-optimal QRQW simulation) *Let $2d/\gamma \leq \mathbf{x} \leq p^{\bar{c}}$, where $\bar{c}, \gamma > 0$ are some constants; let $\delta = \max\{d/\lg(\gamma\mathbf{x}/d), 1\}$; and let $\tau = \delta \lg p + d + L$. One step of a τp-processor* QRQW PRAM *with maximum memory request contention k can be emulated in a work-efficient manner on a p-processor high bandwidth $(d, \mathbf{x})$-*BSP *in time $O(k\tau)$ w.h.p.*

Proof. The memory is randomly hashed onto the $(d, \mathbf{x})$-BSP memory banks.

In this sketch of the proof of Theorem 5.2, we assume that $L \leq \delta \lg p + d$. We also focus entirely on the memory bank contention issue.

For the sake of analysis we re-assign $\mathbf{x}$ to be $\mathbf{x}\gamma$. (The result of the analysis will be the same up to a constant factor.) Thus, $\mathbf{x} \geq 2d$. We assume first that $\mathbf{x} \leq d2^d$, and therefore $\delta = d/\lg(\mathbf{x}/d)$.

Let $c > 0$ be some arbitrary constant, and let $\alpha = \max\{c + \bar{c} + 1, e\}$. We show that the emulation running time is at most $\alpha k\tau$ with probability $1 - p^{-c}$.

Since the QRQW PRAM step is assumed to take time k, each QRQW PRAM processor can send at most k memory requests. For the sake of simplicity in the analysis, we assume here that each processor actually sends exactly k memory requests, and thus the total number of requests is $Z = p\tau k$.

This assumption can be removed by adding dummy memory requests, similarly to the proof of Theorem 5.1.

Let $i_1, i_2, \ldots, i_m$ be the different memory locations accessed in this step, and let k_j be the number of accesses to location i_j, $1 \leq j \leq m$. Note that $\sum_{j=1}^{m} k_j = Z$. Consider a memory bank β. For $j = 1, \ldots, m$, let x_j be an indicator binary random variable which is 1 if memory location i_j is mapped onto the memory bank β, and is 0 otherwise. Thus, $\mathbf{Prob}(x_j = 1) = 1/B$. Let $a_j = k_j/k$; clearly, $a_j \in (0, 1]$. Let $\Psi_\beta = \sum_{j=1}^{m} a_j x_j$. Then,

$$\mathbf{E}(\Psi_\beta) = \sum_{j=1}^{m} \frac{a_j}{B} = \frac{1}{\mathrm{x}p} \sum_{j=1}^{m} \frac{k_j}{k} = \frac{1}{\mathrm{x}p} \cdot \frac{Z}{k} = \frac{\tau\, p\, k}{\mathrm{x}\, p\, k} = \frac{\tau}{\mathrm{x}}.$$

By a theorem of Raghavan and Spencer [Rag88], which provides a tail inequality for the weighted sum of Bernoulli trials, for any $\nu > 0$, $\mathbf{Prob}(\Psi_\beta > (1+\nu)\mathbf{E}(\Psi_\beta))$

$$< \left(\frac{e^\nu}{(1+\nu)^{(1+\nu)}} \right)^{\mathbf{E}(\Psi_\beta)} . \qquad (3)$$

We set $\nu = \alpha \frac{\mathrm{x}}{d} - 1$, implying

$$(1+\nu)\mathbf{E}(\Psi_\beta) = \alpha \frac{\mathrm{x}}{d} \cdot \frac{\tau}{\mathrm{x}} = \frac{\alpha\tau}{d} . \qquad (4)$$

Therefore, $\mathbf{Prob}\left(\Psi_\beta > \frac{\alpha\tau}{d}\right)$

$$\begin{aligned}
&\overset{(3),(4)}{<} \left(\frac{e}{(1+\nu)} \right)^{(1+\nu)\mathbf{E}(\Psi_\beta)} \overset{(4)}{=} \left(\frac{\alpha \mathrm{x}}{ed} \right)^{\frac{-\alpha\tau}{d}} \\
&\overset{\alpha \geq e}{\leq} \left(\frac{\mathrm{x}}{d} \right)^{-\alpha\frac{\tau}{d}} = 2^{-\frac{\lg(\mathrm{x}/d)}{d} \cdot \alpha\tau} = 2^{-\frac{\alpha}{\delta}(\delta \lg p + d)} \\
&= p^{-\alpha} \cdot 2^{-\alpha d/\delta} = p^{-\alpha} \cdot \left(\frac{d}{\mathrm{x}} \right)^{\alpha} \overset{\mathrm{x} \geq 2d}{\leq} p^{-\alpha} \\
&\leq p^{-(c+\bar{c}+1)} = \frac{p^{-(c+1)}}{p^{\bar{c}}} \overset{\mathrm{x} \leq p^{\bar{c}}}{\leq} \frac{p^{-(c+1)}}{\mathrm{x}} .
\end{aligned}$$

Note that

$$R_\beta = \sum_{j=1}^{m} x_j k_j = \Psi_\beta \cdot k .$$

Therefore,

$$\mathbf{Prob}\left(R_\beta > \frac{\alpha\,\tau\,k}{d} \right) < \frac{p^{-(c+1)}}{\mathrm{x}} ,$$

and hence

$$\begin{aligned}
\mathbf{Prob}\left(R > \frac{\alpha\,\tau\,k}{d} \right) &\leq B \cdot \mathbf{Prob}\left(R_\beta > \frac{\alpha\,\tau\,k}{d} \right) \\
&< B \cdot \frac{p^{-(c+1)}}{\mathrm{x}} = p^{-c} .
\end{aligned}$$

The time of the (d, x)-BSP step is $T = dR$, and therefore

$$\mathbf{Prob}(T \leq \alpha\,\tau\,k) \geq 1 - p^{-c} .$$

It remains to consider the case where $\mathrm{x} \geq d2^d$. Consider a partition of the memory banks into $\mathrm{x}' = d2^d$ sets, each denoted as a *memory super-bank*. The module map contention and module load contention are defined with respect to the memory super-banks analogously to their original definitions, and denoted as μ' and R', respectively. The above analysis for μ and R clearly holds for μ' and R'. Since the $\mu \leq \mu'$ and $R \leq R'$, the theorem follows. ∎

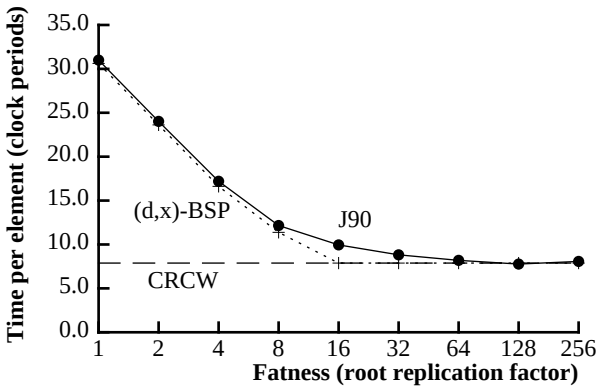

Figure 10: Predicted and measured times for a binary search fat-tree algorithms, as a function of the "fatness" of the tree for a table of size 256. The solid curve shows measured times on 8 processors of the CRAY J90 and the dotted curve shows predicted times.

6 Algorithm experiments

In this section, we present results of experiments on four algorithms: QRQW algorithms for binary search and random permutation, and CRCW algorithms for sparse matrix multiplication and finding connected components of a graph. The (d, x)-BSP model can be used for accurately predicting the running time of each algorithm, and in the case of the binary search, for determining a good setting for the "fatness" parameter.

Binary search: The first QRQW algorithm is a simple parallel binary search to look up n keys in a balanced binary search tree of size m [GMR94a]. Such binary searching is an important substep in several algorithms for sorting and merging (*e.g.* [RV87]). The algorithm replicates nodes of the search tree to avoid contention, and at each level selects one of the replicated nodes at random. This is an interesting problem from the point of view of the QRQW PRAM and the (d, x)-BSP since the amount of replication needed will depend on the contribution of the contention term to the running time, and in general will present a tradeoff. On the CRCW PRAM there is no need for replication.

To design an optimized algorithm that uses binary searching we would like to use the (d, x)-BSP model to predict the running time of the binary search for different amounts of replication. In the experiment, the root is replicated f times (the "fatness"), and each level below the root is replicated half as many times as the level above. Assuming an equal number of lookups to each key, the expected time per lookup is:

$$\sum_{i=0}^{\lceil \lg m \rceil - 1} \max\left(\frac{cg}{p}, \frac{d}{\max(2^i, f)} \right) ,$$

where c is approximately 3.9 on the CRAY J90 and 1.4 on the C90*. Figure 10 shows that the predictions are very accurate.

Random permutation: The second QRQW algorithm generates random permutations using a "dart throwing " algorithm. The algorithm first generates n random indices in the range $[0..cn]$ for some constant c (2 in our experiments).

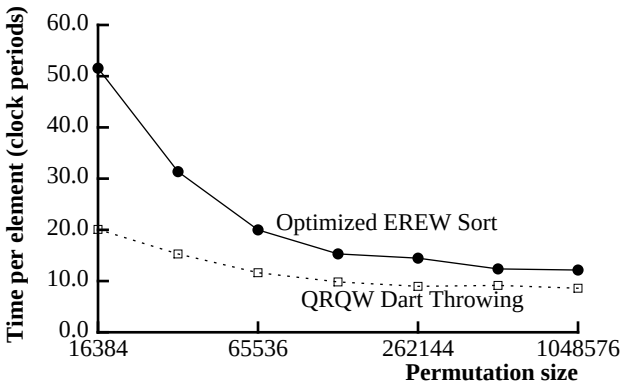

Figure 11: Results on an 8 processor CRAY J90 comparing two algorithms for generating a random permutation: a QRQW dart throwing algorithm and an EREW sorting-based algorithm. The QRQW algorithm performs better over a wider range of problem sizes, and even a simple C implementation outperforms the EREW version, which is based on a highly-optimized radix sort [ZB91]. (The radix sort is currently the fastest implementation of the NAS sorting benchmark [BBDS94].)

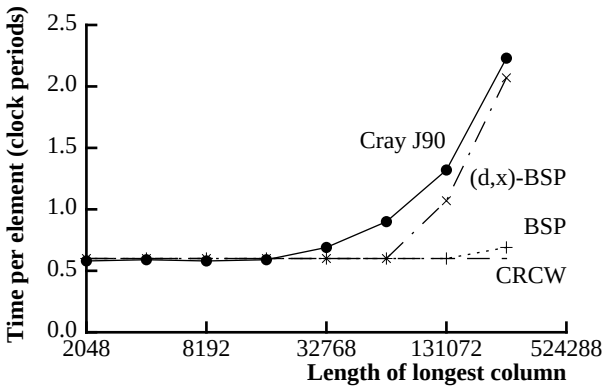

Figure 12: Time per nonzero element for sparse matrix vector multiplication on a matrix with one dense column and an average row size of 7. Measured times are given for an 8 processor CRAY J90, and predicted times are given for the $(d, \mathbf{x})$-BSP, BSP, and CRCW.

Each element i then writes its self-index into a destination array at the location specified by the ith random-index. Elements for which there are no collisions are considered done and drop out. Elements for which there are collisions repeat another round. The rounds continue until there are no elements left. At this point the values written into the destination are packed into contiguous locations, producing the index for the random permutation. The algorithm runs in $O(n/p + \lg n)$ time on a QRQW PRAM [GMR94a].

In our experiments we compare the running time of the algorithm to an algorithm designed for the EREW model (a sorting based algorithm, which is the most practical EREW algorithm in the literature, to the best of our knowledge). The EREW algorithm is based on a radix sort that ranks $2.5 \cdot \lg n$ bits and checks for duplicate keys. This experiment illustrates that by allowing some well-accounted memory contention, we can get a more practical algorithm than in a scenario where we do not allow any contention. (A similar experiment, on the MasPar MP-1, was reported in [GMR94a]; for the EREW algorithm, the system sort was used.) Results of the experiments are shown in Figure 11. (The timings do not include the time for random number generation; however, the two algorithms require approximately the same number of random bits.)

Sparse matrix multiplication: In our third algorithm experiment, we measured the time to multiply a sparse matrix by a dense vector. Our implementation uses a compressed row format containing the number of non-zero elements in each row, and the values of each non-zero matrix element along with its column index. The computation is vectorized using "segmented scan" operations [BHZ93], a technique that allows the latency to be hidden regardless of the structure of the matrix. For the purposes of analyzing contention, the most important characteristic of our implementation is that elements from the input vector are gathered based on the column indices of the nonzero matrix element. Elements of the input vector are typically read multiple times. Thus, our formulation of sparse matrix multiplication can be viewed as a CRCW algorithm or as a QRQW algorithm where the contention is equal to the number of elements in the densest column. A dense column can arise in practice from having a global constraint or bias, such as a circuit with many connections to ground.

In the experiment, we constructed a set of test matrices with one dense column and an average row length of 7. The number of rows and total amount of work are held constant, while the number of elements in the dense column is varied. Except in the dense column, column indices are selected at random. Figure 12 shows measured and predicted time as a function of the length of the dense column.

Connected components: Our final algorithm experiment measures the contention in Greiner's algorithm for finding the connected components of a graph [Gre94]. The algorithm consists of several phases: hooking nodes together to form a forest, performing repeated shortcutting operations to contract each tree to a single node, contracting the graph to form a new graph that is processed recursively, and expanding the graph to propagate the new labels. Contention can occur in each step of the algorithm and varies considerably depending on characteristics of the input graph and the method used for implementing the contraction and expansion phases. In our experiments, we used a variant of Greiner's suite of test graphs consisting of subsets of 2d and 3d toroidal graphs, random graphs (*i.e.* subsets of complete graphs), and "tertiary" graphs in which each node has three random neighbors. In order to reduce module map contention without hashing the data structures, nodes and edges are randomly permuted and renumbered in a preprocessing step. To simplify the comparison of different phases of the algorithm, we extracted memory access patterns used and timed them on a simple "scatter" operation. Note that as discussed in Section 4, under the circumstances of random memory mapping (in this case, due to the nature of the algorithm) and sufficiently high expansion factor, the module map contention can be ignored without substantial error in the prediction. Figure 1 compares timings on the CRAY J90 to predictions generated using the $(d, \mathbf{x})$-BSP model, but ignoring module map contention. These results have helped to clarify some of the observed performance problems with scaling the CRCW algorithm to a large number of processors. We are currently evaluating a number of heuristics for re-

ducing contention in the CRCW algorithm, and we are also working on developing a practical QRQW algorithm.

7 Discussion

This paper studies the effect of bank delay and expansion on performance. We are interested in modeling the effects with as few machine specific details as possible. We therefore described a simple model and verified that it reasonably explains the memory performance of the CRAY C90 and J90. Our results can be summarized as follows:

- For low memory contention, high parallel slackness, and random mapping of memory locations to banks, we can typically ignore the effect of the bank delay.
- When memory contention is high we cannot ignore the effects of bank delay. In particular modeling the delay is quite important in analyzing the performance of algorithms with high contention.
- An expansion factor beyond $\mathbf{x} = d/g$ can be used to better balance the load when using random mapping of memory locations to banks. This is particularly important in conjunction with high memory contention. The expansion on the CRAY is such that the load is not a serious problem.
- For work-efficient simulations of the EREW or QRQW PRAM on a $(d, \mathbf{x})$-BSP an additional d factor is needed in the slackness beyond what is needed for the BSP. This can be alleviated by having additional banks.
- The $(d, \mathbf{x})$-BSP can serve as a bridging model between the QRQW PRAM and high-bandwidth multiprocessors. Our results show that the QRQW model is adequate for designing algorithms and generating rough predictions of running time. When necessary, predictions and implementations can be refined using the $(d, \mathbf{x})$-BSP.

There are several issues that the $(d, \mathbf{x})$-BSP does not capture, which would be needed for a more refined model of the memory system. These include the effects of the network, the effects of caching at the memory banks (available on the TERA and discussed by Hsu and Smith [HS93]), and the effects of the order of injecting messages into the network.

We are currently looking into analyzing the contention properties of other practical parallel algorithms, *e.g.* for merging, multiprefix [She93], and list ranking [RM94].

Acknowledgements. The experiments were run on the CRAY C90 at the Pittsburgh Supercomputing Center (PSC) and on a CRAY J90 at Cray Research. We are grateful to Charles Grassl at Cray and Raghurama Reddy at the PSC for their help running the experiments and getting us exclusive access to the machines. Thanks to Max Dechantsreiter at Tera and Patrick McGehearty at Convex for useful information on the Tera and Convex. Thanks to J. for the papers.

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant number F33615-93-1-1330 and in part by an NSF Young Investigator Award.

References

[ACC+90] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In *Proceedings 1990 International Conference on Supercomputing*, pages 1–6, June 1990.

[Bai87] D. H. Bailey. Vector computer memory bank contention. *IEEE Transactions on Computers*, C-36:293–298, March 1987.

[BBDS94] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. NAS parallel benchmark results 10-94. Technical Report NAS-94-001, NASA Ames Research Center, October 1994.

[BHZ93] G. E. Blelloch, M. A. Heroux, and M. Zagha. Segmented operations for sparse matrix computation on vector multiprocessors. Technical Report CMU-CS-93-173, School of Computer Science, Carnegie Mellon University, August 1993.

[Cal88] D. A. Calahan. Characterization of memory conflict loading on the CRAY-2. In *International Conference on Parallel Processing*, pages 299–302, August 1988.

[Cal89] D. A. Calahan. Some results in memory conflict analysis. In *Proceedings Supercomputing '89*, pages 775–778, November 1989.

[CKP+93] 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.

[CS86] T. Cheung and J. E. Smith. A simulation study of the CRAY X-MP memory system. *IEEE Transactions on Computers*, C-35(7):613–622, July 1986.

[CW79] L.J. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[DGMP92] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable. In *Proc. 19th Int. Colloquium on Automata Languages and Programming, Springer LNCS 623*, pages 235–246, July 1992.

[DHKP93] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. Technical Report Research Report 513, Universitat Dortmund, December 1993.

[EK93] C. Engelmann and J. Keller. Simulation-based comparison of hash functions for emulated shared memory. In *Proc. Parallel architectures and languages Europe, Springer LNCS 694*, pages 1–11, June 1993.

[GMR94a] P. B. Gibbons, Y. Matias, and V. Ramachandran. Efficient low-contention parallel algorithms. In *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, pages 236–247, June 1994.

[GMR94b] P. B. Gibbons, Y. Matias, and V. Ramachandran. The QRQW PRAM: Accounting for contention in parallel algorithms. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 638–648, January 1994.

[GMR94c] P. B. Gibbons, Y. Matias, and V. Ramachandran. The queue-read queue-write PRAM model: Accounting for contention in parallel algorithms. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, September 1994. Submitted for publication.

[Gre94] J. Greiner. A comparison of data-parallel algorithms for connected components. In *Proceedings Symposium on Parallel Algorithms and Architectures*, pages 16–25, Cape May, NJ, June 1994.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[HS93] W.-C. Hsu and J. E. Smith. Performance of cached DRAM organizations in vector supercomputers. In *Proc. 20th International Symp. on Computer Architecture*, pages 327–336, San Diego, CA, May 1993.

[IC93] D. T. Harper III and Y. Costa. Analytical estimation of vector access performance in parallel memory architectures. *IEEE Transactions on Computers*, 42(5):616–624, May 1993.

[JáJ92] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, MA, 1992.

[Knu73] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1973.

[KU86] A.R. Karlin and E. Upfal. Parallel hashing—an efficient implementation of shared memory. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 160–168, May 1986.

[MV84] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 21:339–374, 1984.

[OL85] W. Oed and O. Lange. On the effective bandwidth of interleaved memories in vector processor systems. *IEEE Transactions on Computers*, pages 949–957, October 1985.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[Ran91] A.G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.

[Rau91] B. Rau. Pseudo-randomly interleaved memory. In *Proceedings Int. Symp. Computer Architecture*, pages 74–83, 1991.

[RH90] R. Raghavan and J. P. Hayes. On randomly interleaved memories. In *Proceedings Supercomputing '90*, pages 49–58, November 1990.

[RM94] M. Reid-Miller. List ranking and list scan on the Cray C-90. In *Proceedings Symposium on Parallel Algorithms and Architectures*, pages 104–113, Cape May, NJ, June 1994.

[RV87] J. H. Reif and L. G. Valiant. A logarithmic time sort for linear size networks. *Journal of the ACM*, 34(1):60–76, 1987.

[She93] T. J. Sheffler. Implementing the multiprefix operation on parallel and vector computers. In *Proceedings Symposium on Parallel Algorithms and Architectures*, pages 377–386, Velen, Germany, June 1993.

[Soh93] G. S. Sohi. High-bandwidth interleaved memories for vector processors — a simulation study. *IEEE Transactions on Computers*, 42(1):34–44, January 1993.

[ST91] J. E. Smith and W. R. Taylor. Accurate modeling of interconnection networks in vector supercomputers. In *Proc. International Conference on Supercomputing*, pages 264–273, Cologne, Germany, June 1991.

[TS92] K. Thearling and S. Smith. An improved supercomputer sorting benchmark. In *Proceedings Supercomputing '92*, pages 14–19, November 1992.

[TTT+94] S. Tanoi, Y. Tanaka, T. Tanabe, A. Kita, T. Inada, R. Hamazaki, Y. Ohtsuki, and M. Uesugi. A 32-bank 256-Mb DRAM with cache and tag. *IEEE Journal of Solid-State Circuits*, 29(11):1330–1335, Nov. 1994.

[Val90a] L.G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[Val90b] L.G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 18, pages 944–971. Elsevier Science Publishers B.V., 1990.

[Zag95] M. Zagha. Efficient irregular computation on pipelined-memory multiprocessors. Ph.D. Thesis (In Preparation), 1995.

[ZB91] M. Zagha and G. E. Blelloch. Radix sort for vector multiprocessors. In *Proceedings Supercomputing '91*, pages 712–721, November 1991.

A Proof of Theorem 5.1

Proof. (sketch) The theorem follows the lines of the theorem given in [GMR94b, GMR94c] for the BSP.

For simplicity of exposition, we assume that $\tau = \mathbf{x} \lg p$, and that each processor has exactly k shared memory accesses, where an access is either a read or a write. (The details of removing these two assumptions are omitted in this proof sketch.) Let $i_1, i_2, \ldots, i_m$ be the different memory locations accessed in this step, and let k_j be the number of accesses to location i_j, $1 \le j \le m$. For the purpose of this analysis we add $\alpha p \tau k$ memory accesses to this step, for a constant $\alpha \ge 23$, consisting of accesses with contention k, to locations $i_{m+1}, \ldots, i_{m'}$, where $m' = m + \alpha p \tau$. With this addition, this step has $Z = (\alpha+1) p \tau k$ concurrent accesses to m' different memory locations, and the maximum contention is k. We set $k_j = k$ for $m+1 \le j \le m'$, and note that $Z = \sum_{j=1}^{m'} k_j$. We now show that the bound stated in the theorem holds for this augmented problem. Clearly, this implies that the bound holds for the original problem.

We assume that the memory has been randomly hashed onto the B memory banks of the $(d, \mathbf{x})$-BSP. Consider a fixed bank β. As in [Val90b], we define a random variable x_j, $1 \le j \le m'$, where $x_j = k_j/k$ if memory location i_j is hashed onto memory bank β, and zero otherwise. Let $X_\beta = \sum_{j=1}^{m'} x_j$. We note that $x_j = k_j/k$ with probability $1/B$, and $R_\beta = k \cdot X_\beta$. Then,

$$\mathbf{E}(x_j) = \frac{k_j}{kB}, \quad 1 \le j \le m' \ .$$

Let $\overline{x}$ be the mean of the expectations of the x_j:

$$\begin{aligned} \overline{x} &= \sum_{j=1}^{m'} \frac{\mathbf{E}(x_j)}{m'} = \sum_{j=1}^{m'} \frac{k_j}{k\,B\,m'} = \frac{Z}{k\,B\,m'} \\ &= \frac{(\alpha+1) k\,\tau\,p}{k\,\mathbf{x}\,p\,m'} = (\alpha+1)\frac{\lg p}{m'} \ . \end{aligned}$$

By Hoeffding's inequality [Hoe63],

$$\mathbf{Prob}\left(X_\beta > (\overline{x}+z)m'\right) \le e^{-z^2 m'/3\overline{x}} \ ,$$

provided $z < \min\{\overline{x}, 1-\overline{x}\}$. Let $z = \overline{x}/2$. Then

$$\begin{aligned} \mathbf{Prob}\left(X_\beta > 3\overline{x}m'/2\right) &\le e^{-\overline{x}m'/12} = e^{-(\alpha+1)\lg p/12} \\ &= p^{-\ln 2(\alpha+1)/12} \ . \end{aligned}$$

Since $\mathbf{x} \le p^{\bar{c}}$, it can be shown that $R \in O(k\tau/\mathbf{x})$ w.h.p., and the theorem will follow. ∎