Algorithmica (1994) 12: 170–181

Algorithmica

Locality-Preserving Hash Functions for General Purpose Parallel Computation[1]

A. Chin[2]

Abstract. Consider the problem of efficiently simulating the shared-memory parallel random access machine (PRAM) model on massively parallel architectures with physically distributed memory. To prevent network congestion and memory bank contention, it may be advantageous to hash the shared memory address space. The decision on whether or not to use hashing depends on (1) the communication latency in the network and (2) the locality of memory accesses in the algorithm.

We relate this decision directly to algorithmic issues by studying the complexity of hashing in the Block PRAM model of Aggarwal, Chandra, and Snir, a shared-memory model of parallel computation which accounts for communication locality. For this model, we exhibit a universal family of hash functions having optimal locality. The complexity of applying these hash functions to the shared address space of the Block PRAM (i.e., by permuting data elements) is asymptotically equivalent to the complexity of performing a square matrix transpose, and this result is best possible for all pairwise independent universal hash families. These complexity bounds provide theoretical evidence that hashing and randomized routing need not destroy communication locality, addressing an open question of Valiant.

Key Words. General-purpose parallel computation, Communication latency, Block PRAM, Locality, PRAM simulations, Universal hashing.

1. Introduction.

For many years the parallel random access machine (PRAM) model has provided a standard theoretical framework for discussing the complexity of problems and the performance of parallel algorithms [11], [18]. Because of the popularity of the model, PRAM algorithms represent a potentially rich software library for massively parallel computation.

However, the PRAM model has not been widely adopted by practitioners of parallel computing. Although both complexity theorists and practitioners view efficiency as the paramount concern in parallel computation, their definitions of efficiency are disparate. An asymptotic, worst-case PRAM complexity bound is meaningless to a software engineer trying to achieve constant factor speedups on real data. Further, several aspects of the PRAM model—namely, freedom from communication latency, asynchrony, and component failure—have never been realized on any massively parallel computer.

Recent efforts to close the gap between theory and practice have provided

[1] This work was started when the author was a student at Oxford University, supported by a National Science Foundation Graduate Fellowship and a Rhodes Scholarship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation or the Rhodes Trust.

[2] Department of Mathematics, Texas A&M University, College Station, TX 77843, USA.

Received June 19, 1991; revised November 23, 1992. Communicated by Jeffrey Scott Vitter.

encouraging results in two areas. First, simulation results by Ranade and others [9], [16], [17], [24], [27], [28] have shown how a network of physically distributed processing elements can perform general PRAM computations while restricting the effects of communication latency and asynchrony. Second, the PRAM theory has been extended to more realistic models of parallel computation which account for communication latency [1], [2], asynchrony [8], [12], [23], [26], [31] and component failure [6], [15], [19]. For surveys of this work, see [5], [22], [30], and [31].

However, even this enriched PRAM complexity theory continues to diverge from the practice of parallel computation in an important respect. Much of the practical effort of optimizing parallel software to run on a particular machine goes into achieving communication locality: i.e., ensuring that large blocks of data are used for communication whenever possible. On the other hand, complexity theorists have proposed PRAM simulation algorithms which prevent memory contention and/or network congestion by performing universal hashing on the shared address space, even though hashing has been believed to destroy communication locality [14], [31]. In what follows we refer to the procedure of moving the entire contents of the address space into the hashed address space for a given hash function as *performing the hash function*; the complexity of this procedure is referred to as the *complexity of the hash function.*

In this paper we address this gap in the complexity theory by proposing the notion of *locality-preserving hash functions* for general-purpose parallel computation. We exhibit a universal family of hash functions that can be performed in asymptotically optimal time on the Block PRAM, a variant of the PRAM model which accounts for communication locality. We believe that such hash functions will be useful in the development of general-purpose parallel computers which exploit communication locality, through both current techniques for supporting virtual shared memory, and future application of recent efficient algorithms for *bit-serial routing.*

Specifically, we foresee the development of parallel architectures which simulate the performance of the Block PRAM model on physically distributed memory. Such architectures would allow long blocks of data to be pipelined to exploit communication locality, but would be able to avoid contention through universal hashing.

The remainder of this paper is organized as follows. Section 2 defines the problem and the complexity model. Section 3 proves a lower bound on the expected time complexity of universal hashing. Section 4 gives an optimal algorithm for performing the hash functions in a particular universal family. Section 5 describes several areas in the practice of parallel computation where the complexity of hashing is of interest. Section 6 concludes.

2. Preliminaries.

Throughout this paper all logarithms are base two.

2.1. *The Block PRAM Model.*

The results in this paper are based on the Block PRAM model of parallel computation as introduced by Aggarwal *et al.* [1].

DEFINITION. A *Block PRAM* is a collection of p processors, each with a local memory of unbounded size, together with a shared memory of unbounded size. All processors execute the same program, although a processor may wait instead of executing a given instruction. Each arithmetic operation and access to a local memory location can be performed in unit time. Accesses to shared memory are subject to a delay of l time units due to communication latency. A processor may access a block of b consecutive locations in the shared memory in time $l + b$. No read or write conflicts are allowed: concurrent requests for overlapping blocks are serialized in some arbitrary order.

Note that the Block PRAM is essentially an exclusive-read, exclusive-write PRAM [11], [18] with a time charge for the communication latency incurred in accessing blocks of shared memory. The effect of this charge on the running time of an algorithm depends on the extent to which large blocks are used for communication. With its two-level hierarchical memory, the Block PRAM is able to account for communication latency and locality while remaining independent of any specific network topology. In particular, the Block PRAM time complexity of permuting data in shared memory varies according to the permutation; e.g., see [1] and [7].

2.2. Universal Hashing. Hash functions are used in PRAM simulation to compute physical addresses for all logical memory accesses, and therefore they should be easy to specify and evaluate. Universal families of hash functions were introduced by Carter and Wegman [4] as a computationally feasible alternative to random hashing. Universal hashing has recently emerged as a key component of many PRAM simulations on physically distributed memory [9], [16], [17], [24], [27], [28].

DEFINITION. A family $\mathcal{H}$ of hash functions with domain D and range R is said to be h_μ-*wise independent* if, for all $y_1, \ldots, y_h \in R$, all distinct $x_1, \ldots, x_h \in D$, and some constant μ, $|\{f \in \mathcal{H}: f(x_i) = y_i, i = 1, \ldots, h\}| \leq \mu|\mathcal{H}|/|R|^h$. (Often the subscript μ will be omitted.)

Families with 2-wise (pairwise) independence have useful analytic properties in PRAM simulation [2], [24], [28], [31] and may be adequate for all practical purposes. This paper focuses on proving bounds for these pairwise independent hash families.

It is an open question whether highly ($\omega(1)$-wise) independent hash families are required for PRAM simulations in practice. If so, such families are available. Recently, Siegel [28] presented $n^{\Omega(1)}$-wise independent families of hash functions which can be defined in sublinear space and computed in constant time. These families, based on expander graphs, are widely applicable in theoretical PRAM simulations, including those for the Block PRAM.

2.3. Hashing on the Block PRAM. We now formalize what it means to perform universal hashing on the Block PRAM. We assume that some finite contiguous

block in the Block PRAM shared memory is initially unhashed: the logical addresses and the physical addresses are one and the same. Without loss of generality, we number these addresses $[0 \cdots s-1]$. (In an implementation on distributed memory, the physical addresses would be partitioned into modules [24]; in this paper, however, we study the movement of data at the address level rather than the modular level.)

To perform a permutation Π on this block of the shared memory, it is sufficient to perform a sequence of copying instructions on memory elements so that the element initially in physical address i finishes in physical address $\Pi(i)$. Such an algorithm consisting of copying instructions only is called a *conservative* algorithm. Focusing on the problem of locality in the presence of communication latency, Aggarwal *et al.* [1] restrict the Block PRAM complexity theory of data movement to conservative algorithms for performing permutations. For consistency and simplicity, we apply the same restrictions to our study of hashing in this paper.

It should be noted that the hash functions in many universal hash functions are not necessarily permutations. It is possible to define a scheme for non-injective hashing on the Block PRAM. Suppose we are hashing array $X[0 \cdots s-1]$ using function $f: [0 \cdots s-1] \to [0 \cdots s-1]$. The array will be hashed into an (uninitialized) $s \times (s+1)$ matrix Y, stored in row-major order, such that, for $0 \le j \le s-1$, the first $|f^{-1}(j)| + 1$ entries in the column vector Y_j are the elements of $\{X(i): i \in f^{-1}(j)\}$ (possibly none) in some order, followed by an end marker. The rows of the matrix correspond to successive probes in the hash table, which must take place in distinct accesses to the shared memory.

This scheme generalizes Block PRAM permutation while preserving locality considerations: the complexity of performing permutations is at most doubled, and the contiguity of blocks at each probe level is preserved. However, the additional time and space complexities of resolving collisions will pass to any PRAM simulation which uses the scheme. For simplicity, we do not attempt to characterize the Block PRAM complexity of noninjective hashing further in this paper.

2.4. Locality-Preserving Hashing. As we are considering communication locality in a topology-independent context, we use the Block PRAM model, with a hashed shared memory space, to represent a general-purpose parallel computer simulating the PRAM model. If a Block PRAM algorithm has asymptotically the same running time in hashed and unhashed shared memory, we can say that the hashing has preserved the communication locality in the algorithm.

That is, a Block PRAM algorithm can be performed in hashed shared memory by

(i) unhashing the shared memory,
(ii) running the algorithm, and
(iii) rehashing the shared memory.

If there is no increase in the asymptotic running time of the algorithm, then the exploitation of locality by the algorithm has been preserved. This motivates the following definition.

DEFINITION. Let $\mathscr{A}$ be a Block PRAM algorithm with time bound $T(n, l, p)$ which uses at most $S(n, l, p)$ consecutive shared memory locations. Let $\mathscr{H}$ be a universal family of bijective hash functions on $[0 \cdots S(n, l, p) - 1]$. Then $\mathscr{H}$ is *locality-preserving* for $\mathscr{A}$ if, for any $f \in \mathscr{H}$, f can be performed and inverted by a Block PRAM on $S(n, l, p)$ consecutive locations in shared memory in time $O(T(n, l, p))$.

2.5. Bit-Serial Randomized Routing. Recent bit-serial randomized routing algorithms [3], [20] provide a theoretical basis for designing massively parallel architectures which allow block pipelining and, therfore, exploitation of locality as described by the Block PRAM model. These algorithms allow any permutation of n message packets of size m to be routed in time $O(m + \log n)$ with high probability on hypercube and butterfly networks. (Previous randomized routing algorithms [13], [21], [27], [29], [32] are designed for fixed-size packets and run in time $O(m \log n)$ when implemented on real machines [3].) The performance of these bit-serial algorithms can be stated in the following theorem.

THEOREM 2.1. *Any permutation of n packets of size m can be routed in* $O(m + \log p)$ *time*

- *with probability* $1 - n^{-\Omega(1)}$ *on a hypercube network with* $p = n$ [3], *and*
- *with probability* $1 - 2^{-2^{\Omega(\sqrt{\log n})}}$ *on a butterfly network with* $p = n \log n$ [20].

3. The Lower Bound. We prove a lower bound for implementing pairwise independent univeral families of bijective hash functions on the Block PRAM. The proof depends on a potential function argument first developed in [9a] and [2a] and later applied in Theorem 3.3 of [1] to prove a lower bound for transposing a square matrix on the Block PRAM.

Consider any conservative Block PRAM algorithm for performing a permutation in shared memory. We can assume the following, while multiplying the running time by at most a constant factor:

(1) Only one copy of each memory element is in use ("live") at any time during the execution of the algorithm.
(2) The shared memory can be subdivided into *segment* each consisting of $\min(l, p, n/l, n/p)$ memory locations, and execution of the algorithm can be subdivided into *read rounds* and *write rounds* each taking time $\Theta(l)$, such that during each round, each processor accesses a block of at most l memory locations.

DEFINITION. For $x > 0$, define the *entropy function* $H(x) = x \log x$; by convention we take $H(0) = 0$.

Let m be a positive integer and let f be a permutation on $[0 \cdots s - 1]$. For integers j, k, $0 \leq j$, $k \leq \lfloor (s-1)/m \rfloor$, let

$$A_{j,k}(f, m) = |\{a \in [0 \cdots s-1] : \lfloor a/m \rfloor = j \quad \text{and} \quad \lfloor f(a)/m \rfloor = k\}|.$$

The *m-wise entropy* of f is given by $H(f, m) = \sum_{j,k} H(A_{j,k}(f, m))$.

Consider a conservative algorithm for performing f on the shared memory. We may define the *potential functions* $x_{j,k}(f, m, t)$, $y_{i,k}(f, m, t)$ as follows. For $j \geq 0$, $0 \leq k \leq \lfloor (s-1)/m \rfloor$ and $1 \leq i \leq p$, let

$$x_{j,k}(f, m, t) = |\{a \in \mathbf{N} \cup \{0\}, b \in [0 \cdots s-1]: \lfloor a/m \rfloor = j, \lfloor b/m \rfloor = k \text{ and there is a live element at shared memory address } a \text{ at the beginning of the } t\text{th round having destination address } b\}|,$$

and let

$$y_{i,k}(f, m, t) = |\{b \in [0 \cdots s-1]: \lfloor b/m \rfloor = k \text{ and there is a live element in the local memory of processor } i \text{ having destination address } b\}|.$$

Note in particular that $\sum_{j,k} x_{j,k}(f, m, 1) = H(f, m)$ and $\sum_{i,k} y_{i,k}(f, m, 1) = 0$; also, if the algorithm is finished after r rounds, then $\sum_{j,k} x_{j,k}(f, m, r+1) = s \log m$ and $\sum_{i,k} y_{i,k}(f, m, r+1) = 0$.

LEMMA 3.1. *Let f be a permutation on $[0 \cdots n-1]$. Any conservative Block PRAM algorithm for performing f on n consecutive locations in shared memory requires time*

- $\Omega((n \log m - H(f, m))/(p \log(2n/(lp))))$ *for* $lp \leq n$, *where* $m = \min(l, p)$, *and*
- $\Omega(l(n \log m - H(f, m))/(n \log(2lp/n)))$ *for* $lp > n$, *where* $m = \min(n/l, n/p)$.

PROOF. Denote $\Phi(f, m, t) = \sum_{j,k} x_{j,k}(f, m, t) + \sum_{i,k} y_{i,k}(f, m, t)$ and $\Delta\Phi(r) = \Phi(f, m, r+1) - \Phi(f, m, 1)$. From the above discussion, we have

$$\Delta\Phi(t) = n \log m - H(f, m).$$

Further, it is shown in the proof of Theorem 3.3 of [1] that $\Delta\Phi(r) \leq rpl \log(2n/(lp))$ for $lp \leq n$, $m = \min(l, p)$, and $\Delta\Phi(r) \leq rn(\log(2lp/n))$ for $lp > n$, $m = \min(n/l, n/p)$. Since each round takes time $\Theta(l)$, the stated bounds follow. □

THEOREM 3.2. *Let $\mathcal{H}$ be a 2_μ-wise independent universal family of bijective hash functions on $[0 \cdots n-1]$ and let f be chosen randomly from $\mathcal{H}$. Then any conservative Block PRAM algorithm performing f on n consecutive locations in shared memory requires expected time*

- $\Omega(n/p + (n \log \min(l, p))/(p \log(2n/(lp))))$ *for* $lp \leq n$, *and*
- $\Omega(l + l \log \min(n/l, n/p)/\log(2lp/n))$ *for* $lp > n$.

PROOF. For each of the cases $l = O(1)$, $l = \Omega(n)$, $p = O(1)$, $p = \Omega(n)$, the above bounds are just $\Omega(n/p + l)$, the trivial lower bound.

In the nontrivial case let $m = \min\{l, p, n/l, n/p\}$. Then $m = \omega(1)$ and $m = O(\sqrt{n})$. We estimate $E(H(f, m))$ as follows:

$$E\left(\sum_{j,k}(A_{j,k}(f,m)\cdot(A_{j,k}(f,m)-1))\right)$$
$$= E\left(\sum_{j,k}|\{(x_1,x_2,y_1,y_2): x_1 \neq x_2, \lfloor x_i/m\rfloor = j, \lfloor y_i/m\rfloor = k, f(x_i)=y_i\}|\right)$$
$$\leq \mu m^2$$

(by 2_μ-wise independence); and therefore

$$\begin{aligned}E(H(f,m)) &= E\left(\sum_{j,k} H(A_{j,k}(f,m))\right)\\ &\leq E\left(\sum_{j,k}(A_{j,k}(f,m))^2\right)\\ &= E\left(\sum_{j,k}(A_{j,k}(f,m)\cdot(A_{j,k}(f,m)-1))\right) + E\left(\sum_{j,k}A_{j,k}(f,m)\right)\\ &\leq \mu m^2 + n\\ &= O(n)\\ &= o(n\log m),\end{aligned}$$

and the theorem follows from Lemma 3.1. □

4. The Algorithm. In this section we present $\mathcal{F}$, a pairwise independent universal family of bijective hash functions which can be performed on Block PRAM arrays in asymptotically optimal time. Together with Theorem 3.2, this demonstrates that the locality-preserving properties of $\mathcal{F}$ are best possible for a universal family of hash functions.

Throughout this section we consider the problem of performing permutations on $n = 2^k$ shared memory locations on the Block PRAM. For any $k \in \mathbf{N}$, it will be useful to identify shared memory addresses $\{0, 1, \ldots, 2^k - 1\}$ with their binary representations $\{0,1\}^k$ as 0–1 column vectors of length k (highest-order bits first). For $x \in \{0,1\}^k$, let x_i denote the ith element of x.

LEMMA 4.1. *For $x \in \{0,1\}^k$, denote $\bar{x} = (x_1, \ldots, x_{k-\log l})$ and $\underline{x} = (x_{1+\log p}, \ldots, x_k)$. Let Π be a* basic permutation *on $\{0,1\}^k$: that is, one that can be expressed in one of the following two forms:*

(a) $\Pi(x) = (f_1(\bar{x}), \ldots, f_{k-\log l}(\bar{x}), x_{1+k-\log l}, \ldots, x_k)$, *or*
(b) $\Pi(x) = (x_1, \ldots, x_{\log p}, g_{1+\log p}(\underline{x}), \ldots, g_k(\underline{x}))$,

where f_i, g_j are 0–1 valued functions from $\{0,1\}^{k-\log l}$, $\{0,1\}^{k-\log p}$, respectively. Then a Block PRAM with p processors and communication latency l can perform Π conservatively on $n = 2^k$ consecutive locations in shared memory in time $O(n/p + l)$.

PROOF. We assume l and p are integral powers of two: l can be increased and p can be decreased to the next power of two while multiplying the running time by at most a factor of four.

To perform a permutation of type (a), each processor reads $n/(lp)$ blocks of length l in shared memory and writes them into their new locations in $O(n/p + l)$ time. (If $lp > n$, some of the processors will be idle.)

To perform a permutation of type (b), each processor reads one block of length n/p in shared memory, permutes it, and writes it back in $O(n/p + l)$ time. (If $p > n$, some of the processors will be idle.) □

LEMMA 4.2 [1, Theorem 3.1]. *Let Π be a* rational permutation *on $\{0,1\}^k$: that is, one that can be defined by a permutation π on $[1 \cdots k]$ as follows: $\Pi(x) = (x_{\pi(1)}, \ldots, x_{\pi(k)})$. Then a Block PRAM with p processors and communication latency l can perform Π conservatively on $n = 2^k$ consecutive locations in shared memory in time*

- $O(n/p + (n \log \min(l, p))/(p \log(2n/(lp))))$ *for* $lp \leq n$, *and*
- $O(l + l \log \min(n/l, n/p)/\log(2lp/n))$ *for* $lp > n$.

DEFINITION. Let M be the set of nonsingular $k \times k$ 0–1 matrices. For each $A \in M$ denote the permutation f_A on $\{0,1\}^k$ by $f_A(x) = Ax \bmod 2$. Define the family of functions $\mathcal{F} = \{f_A : A \in M\}$.

In [24] Mehlhorn and Vishkin observed that $\mathcal{F}$ is a 2_μ-wise independent universal family of bijective hash functions, where $\mu = \prod_{j=1}^{k}(1 - 2^{-j})^{-1} < e^{7/5}$. We now show that the hash functions in $\mathcal{F}$ can be performed in asymptotically optimal time.

THEOREM 4.3. *Let $f \in \mathcal{F}$. Then a Block PRAM with p processors and communication latency l can perform f conservatively on $n = 2^k$ consecutive locations in shared memory in time*

- $O(n/p + (n \log \min(l, p))/(p \log(2n/(lp))))$ *for* $lp \leq n$, *and*
- $O(l + l \log \min(n/l, n/p)/\log(2lp/n))$ *for* $lp > n$.

PROOF. We give the proof for the case $lp \leq n$, $l \leq p$; the other cases are analogous. Let $f_A \in \mathcal{F}$. We recall that any nonsingular square matrix A can be factored into the form LUP, where L is a lower triangular matrix, U is an upper triangular matrix, and P is a permutation matrix [10]. We can perform the permutation Ax by successively applying P, U, and L to x, Px, and UPx, respectively. We show that each of these permutations can be performed within the stated time bounds.

An application of P is just a rational permutation, which can be performed within the time bounds in Lemma 4.2.

Let I denote the $k \times k$ identity matrix. We claim that U can be factored into a product of nine matrices, each of which represents a basic permutation or a rational permutation as described in Lemmas 4.1 and 4.2. We demonstrate this by presenting a row reduction of U^{-1} to the identity matrix I in nine stages, such that each stage of the row reduction corresponds to one matrix in the factorization of U.

Note that U^{-1}, like U, is also an upper triangular matrix with all 1's along the diagonal. Let r_i denote the ith row of U^{-1} (even as rows are permuted during the reduction, r_i will continue to refer to the row that was originally the ith row of U^{-1}). Then, for subsets $S \subseteq [1 \cdots k]$, it is possible to perform successive row reductions involving only the rows in $R(S) = \{r_i : i \in S\}$ such that each entry $(r_i)_i$ remains one and each entry in $\{(r_i)_j : i, j \in S, i \neq j\}$ becomes zero.

We also note that a row reduction involving only the first $k - \log l$ rows corresponds to a basic permutation of type (a), while a row reduction involving only the last $k - \log p$ rows corresponds to a basic permutation of type (b).

Let

$$S_1 = [1 \cdots k - (3 \log l)/2],$$
$$S_2 = [1 + k - (3 \log l)/2 \cdots k - \log l],$$
$$S_3 = [1 + k - \log l \cdots k - (\log l)/2],$$

and

$$S_4 = [1 + k - (\log l)/2 \cdots k].$$

The reduction steps may be described as follows:

1. Row-reduce among the first $k - \log l$ rows: i.e., $R(S_1 \cup S_2)$. This corresponds to a basic permutation of type (a).
2. Row-reduce among the last $\log l$ rows: i.e., $R(S_3 \cup S_4)$. Since $\log l \leq k - \log p$, this corresponds to a basic permutation of type (b).
3. Permute the rows by exchanging the submatrices $R(S_2)$ and $R(S_3)$. This corresponds to a rational permutation.
4. The first $k - \log l$ rows are now $R(S_1 \cup S_3)$. Row-reduce among these rows. This corresponds to a basic permutation of type (a).
5. The last $\log l$ rows are now $R(S_2 \cup S_4)$. Row-reduce among these rows. This corresponds to a basic permutation of type (b).
6. Permute the rows by exchanging the submatrices $R(S_3)$ and $R(S_4)$. This corresponds to a rational permutation.
7. The first $k - \log l$ rows are now $R(S_1 \cup S_4)$. Row-reduce among these rows.
8. The last $\log l$ rows are now $R(S_2 \cup S_3)$. Row-reduce among these rows.
9. Permute the rows into the original order.

The factorization of L is analogous and omitted.

We have factored f into a constant number of basic and rational permutations, so that the complexity bounds follow immediately from Lemmas 4.1 and 4.2. □

The following observation follows immediately from Theorems 3.2 and 4.3:

COROLLARY 4.4. *Let $\mathcal{F}'$ be a universal family of hash functions, and let $\mathcal{A}$ be a Block PRAM algorithm. If $\mathcal{F}'$ is locality-preserving for $\mathcal{A}$, then so is $\mathcal{F}$.*

5. Applications. Our results suggest that hashing is not necessarily an obstacle to exploiting locality in general-purpose parallel computation. For example, a parallel architecture may be designed specifically to simulate the performance of the Block PRAM model on physically distributed memory using bit-serial randomized routing. When it is desirable to exploit locality as in many special-purpose applications, the machine will allow long blocks to be pipelined as in the Block PRAM model. When more finely grained parallelism is required, the machine can use hashing to prevent contention in order to support an efficient PRAM simulation. By using a locality-preserving hash family such as $\mathcal{F}$, such an architecture would be able to switch quickly between the two modes of operation. The unhashed mode would be preferred whenever possible to maximize the effective parallelism, since the general PRAM simulation requires a high degree (lp^{ε}) of parallel slackness [1, Theorem 6.1].

The complexity of the data movement required to perform hashing is of independent interest in other contexts.

- *Automatic hashing*: Even if the shared memory is always hashed, it will still be necessary to change the hash function from time to time [25], [27]. Rehashing is required, for example, if a particular hash function proves ineffective in preventing contention during a given computation.
- *Partially hashed shared memory*: In certain shared-memory designs, some of the memory address space will be hashed and some left unhashed [12]. The complexity of hashing is important in determining the possible advantages of this approach, and the cost of changing the partition.
- *Input/output*: Files will be read into (and out from) the shared memory in unhashed form and will need to be hashed (and unhashed).

6. Conclusion. In this paper we have studied the complexity of performing universal hash functions using conservative Block PRAM algorithms, demonstrating in particular that hashing need not destroy communication locality. We have focused on the complexity of data movement so as to address directly the issue of locality. As a result, we have left many issues open for future study.

The PRAM simulation on the Block PRAM described in Theorem 6.1 of [1] relies on a conservative Block PRAM algorithm for performing general permutations. It remains unclear to what extent the necessary off-line computations would affect the simulation's performance in practice.

The Block PRAM complexity of universal hash families with higher degrees of independence remains open. For these families, can the lower bound of Theorem 3.2 be improved, or the upper bound of Theorem 4.3 be extended? The discovery of locality-preserving, highly independent hash families—together with improved bit-serial routing algorithms, incorporating fault tolerance—will constitute significant progress toward providing for the exploitation of locality in general purpose parallel computation. The precise relationship between the independence and locality-preserving properties of hash families will be a challenging and rewarding study.

Acknowledgments. I wish to thank my supervisor, Dr. William F. McColl, for introducing me to many of these issues and for continual encouragement, and to the referees for suggesting improvements in the presentation of the paper.

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir, On communication latency in PRAM computations, *Proc. First ACM Symp. on Parallel Algorithms and Architectures*, 1989, pp. 11–21.

[2] A. Aggarwal, A. K. Chandra, and M. Snir, Communication complexity of PRAMs, *Theoret. Comput. Sci.* **71** (1990), 3–28.

[2a] A. Aggarwal and J. S. Vitter, The input/output complexily of sorting and related problems, *Comm. ACM* **31**(9) (1988), 1116–1127.

[3] W. Aiello, T. Leighton, B. Maggs, and M. Newman, Fast algorithms for bit-serial routing on a hypercube, *Proc. 2nd Annual ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 55–64.

[4] J. L. Carter and M. N. Wegman, Universal classes of hash functions, *J. Comput. System Sci.* **18** (1979), 143–154.

[5] A. Chin, Complexity issues in general-purpose parallel computation, D.Phil. thesis, Oxford University, 1991.

[6] A. Chin, Latency hiding for fault-tolerant PRAM computations, *Proc. Internat. Conf. on Sets, Graphs and Numbers*, D. Miklos, ed., North-Holland, Amsterdam, 1992.

[7] A. Chin, Permutations on the Block PRAM, *Inform. Process. Lett.* **45** (1993), 69–73.

[8] R. Cole and O. Zajicek, The APRAM: incorporating asynchrony into the PRAM model, *Proc. First Annual ACM Symp. on Parallel Algorithms and Architectures*, 1989, pp. 169–178.

[9] M. Dietzfelberger and F. Meyer auf der Heide, How to distribute a dictionary in a complete network, *Proc. 22nd Annual ACM Symp. on Theory Computing*, 1990, pp. 117–127.

[9a] R. W. Floyd, Permuting information in idealized two-level storage, in *Complexity of Computer Calculations*, R. Miller and J. Thatcher, eds., Plenum, New York, 1972, pp. 105–109.

[10] G. E. Forsythe and C. B. Moler, *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1967.

[11] A. M. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, 1988.

[12] P. Gibbons, The asynchronous PRAM: a semi-synchronous model for shared memory MIMD machines, Ph.D. thesis, University of California at Berkeley, 1989.

[13] J. Håstad, T. Leighton, and M. Newman, Fast computation using faulty hypercubes, *Proc. 21st Annual ACM Symp. on Theory of Computing*, 1989, pp. 251–263.

[14] T. Heywood and S. Ranka, A practical hierarchical model of parallel computation: the model, Technical Report SU-CIS-91-06, Syracuse University, Syracuse, NY 10991.

[15] P. Kanellakis and A. Shvartsman, Efficient parallel algorithms can be made robust, *Proc. 8th Annual ACM Symp. on Principles of Distributed Computing*, 1989, pp. 211–222.

[16] A. Karlin and E. Upfal, Parallel hashing: an efficient implementation of shared memory, *J. Assoc. Comput. Mach.* **35** (1988), 876–892.

[17] R. M. Karp, M. Luby, and F. Meyer auf der Heide, Efficient PRAM simulation on a distributed memory machine, *Proc. 24th Annual ACM Symp. on Theory of Computing*, 1992, pp. 318–326.

[18] R. M. Karp and V. Ramachandran, Parallel algorithms for shared-memory machines, *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), North-Holland, Amsterdam, 1990, pp. 869–942.

[19] Z. M. Kedem, K. V. Palem, and P. G. Spirakis, Efficient robust parallel computations, *Proc. 22nd Annual ACM Symp. on Theory of Computing*, 1990, pp. 138–148.

[20] F. T. Leighton and C. G. Plaxton, A (fairly) simple circuit that (usually) sorts, *Proc. 31st Annual IEEE Symp. on Foundations of Computer Science*, 1990, pp. 264–274.

[21] Y.-D. Lyuu, Fast fault-tolerant parallel communication and on-line maintenance using information dispersal, *Proc. Second ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 378–387.

[22] W. F. McColl, General purpose parallel computing, in *Lectures on Parallel Computation*, A. Gibbons and P. Spirakis, eds., Cambridge University Press, Cambridge, 1993, pp. 337–391.

[23] C. Martel, A. Park, and R. Subramonian, Optimal asynchronous algorithms for shared-memory parallel computers, Technical Report CSE-89-8, University of California at Davis, 1989.

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

[25] J. K. Mullin, A caution on universal classes of hash functions, *Inform. Process. Lett.* **37** (1991), 247–256.

[26] N. Nishimura, Asynchronous shared memory parallel computation, *Proc. Second Annual ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 76–84.

[27] A. G. Ranade, How to emulate shared memory, *Proc. 28th Annual IEEE Symp. on Foundations of Computer Science*, 1987, pp. 185–194.

[28] A. Siegel, On universal classes of fast high-performance hash functions, their time–space tradeoff, and their applications, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 20–25.

[29] E. Upfal and A. Wigderson, How to share memory in a distributed system, *Proc. 25th Annual IEEE Symp. on Foundations of Computer Science*, 1984, pp. 171–180.

[30] L. G. Valiant, A bridging model for parallel computation, *Comm. ACM* **33** (1990), 103–111.

[31] L. G. Valiant, General purpose parallel architectures, *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), North-Holland, Amsterdam, 1990, pp. 103–110.

[32] L. G. Valiant and G. J. Brebner, Universal schemes for parallel communication, *Proc. 13th Annual ACM Symp. on Theory of Computing*, 1981, pp. 263–277.