On universal classes of extremely random constant-time hash functions and their time-space tradeoff

Alan Siegel[†]
Courant Institute
New York University
New York, NY 10012

[†]The work of the author was supported in part by ONR grant N00014-85-K-0046, NSF grants CCR-8906949, CCR-8902221, CCR-9204202, and CCR-9503793.

Abstract

A family of functions F that map $[0, m-1]$ into $[0, n-1]$ is said to be κ-wise independent if any tuple of κ distinct points in $[0, m-1]$ have a corresponding image, for a randomly selected $f \in F$, that is uniformly distributed in $[0, n-1]^\kappa$. This paper shows that for suitably fixed $\epsilon < 1$, and any $\kappa < m^\epsilon$, there are families of κ-wise independent functions that can be evaluated in constant time for the standard random access model of computation, and which require a provably necessary storage array of m^δ random seeds, for a suitable $\delta < 1$. These seeds can be pseudo-random values precomputed from an initial $O(\kappa)$ random seeds. A simple adaptation yields n^ϵ-wise independent functions that require n^δ storage in many cases where $m \gg n$. In addition, lower bounds are presented to show that neither storage requirement can be materially reduced. Previous constructions of random functions having constant evaluation time and sublinear storage exhibited only a constant degree of independence.

Unfortunately, the explicit randomized constructions, while requiring a constant number of operations, are far too slow for any practical application. However, non-constructive existence arguments are given, which suggest that this factor might be eliminated. The problem of eliminating this factor is shown to be equivalent to a fundamental open question in graph theory.

As a consequence of these constructions, many probabilistic algorithms can for the first time be shown to achieve, up to constant factors, their expected asymptotic performance for a programmable, albeit formal and rather impractical model of computation, and a research direction is now available that may eventually lead to implementations that are fast and provably sound.

Categories and Subject Descriptors: E.2 [**Data Storage Representation**]: Hash-table representation; F.1.2 [**Modes of Computation**]: Probabilistic Computation; F2.3 [**Tradeoffs among Computational Measures**]; F.2.1 [**Computation in finite fields**]; G.3 [**Probability and Statistics**]: Random number generation.

General terms: Algorithms, Theory.

Additional Key Words and Phrases: Hash functions, universal hash functions, hashing, limited independence, storage-time tradeoff.

1. Introduction

Many probabilistic algorithms and data structures have been proven to work well when fully random functions are used as unit-time subroutines. For example, it is well known that for uniform hashing, the expected cost to insert the $(\alpha n+1)$-st item into a table of size n is $\frac{1}{1-\alpha}-o(1)$ probes when fully random hash functions are used [11]. Moreover, Yao proved that in terms of expected retrieval cost for open-address (a.k.a. closed) hashing, uniform hashing is optimal: up to a factor of $(1+o(1))$, no family of hash functions gives a better average-case performance than the random functions of uniform hashing [31].

Yet the significance of these performance bounds for real computation is by no means clear. The difficulty is that they have been proven for hash functions that cannot be efficiently computed. Suppose, for example, we wish to select a random mapping from $[0, n^2-1]$ into $[0, n-1]$. Since there are n^{n^2} such mappings, it follows that the program length of such a function must be at least $n^2 \log n$ bits, on average, which is much larger than the hash table it is intended to service.

On the other hand, results based upon full randomness sometimes translate into average case performance guarantees for real computation. For example, double hashing uses the same insertion and retrieval strategies as uniform hashing, but defines the j-th probe for a key x to be $(d(x)+(j-1)f(x)) \mod p$, where p is prime, the table addresses range from 0 to $p-1$, the function d is a random mapping of the key space into $[0, p-1]$, and f is a comparable mapping into $[1, p-1]$. This scheme requires about $2\lceil \log p \rceil$ random bits per hash key, and we could take these bits to be any fixed portions of the key itself, provided it is at least $2\lceil \log p \rceil$ bits long. Then a probabilistic performance bound for the truly random case would hold as an average case performance bound for this deterministic case, where the averaging is over all possible sequences of data keys. For uniform hashing, which requires additional randomness, the question of how to interpret a probabilistic upper bound on performance is even more problematic.

Yet even where such average case results are meaningful, we would rather establish randomized performance bounds – which hold, on average, for any set of data – instead of a bound that cannot be applied to any fixed instance of data.

Many theoretical studies of large scale parallel and distributed computation have foundational assumptions and performance assurances based on efficient randomization. For example, storage is often randomized to distribute data, to balance access patterns to memory modules, and to balance load requirements across networks. While some forms of randomization can use precomputed and even adaptive methods to achieve the desired performance, other fundamental computation, and communication problems have yet to be solved by any provably efficient method that does not depend on high randomness. Valiant and Brebner were the first to show that randomization could be used to get high performance in various communication networks, and that deterministic schemes were bound to have communication bottlenecks [29]. The performance proofs for PRAM emulation as developed by Ranade [18] depend upon fairly significant amounts of randomized computation. In fact, his address computations require $\Theta(\log n)$ arithmetic operations to map a data reference into a physical memory address. But while these methods (or the associated performance proofs) might be viewed as, at present, somewhat restrictive, they represent a significant improvement over formulations based on the idealized mathematical axioms of fully random functions. Accordingly, it is to these notions of limited randomness that we now turn our attention.

1.1 Background and overview

Carter and Wegman introduced *universal classes of hash functions* [4] to provide a theoretical and pragmatic framework for methods that exploit computable hash functions with fixed degrees of freedom. Subsequent work on universal hashing (cf. [30, 15, 16]) have sometimes defined limited randomness with a few extra degrees of freedom, but all of the formulations are basically as follows:

Definition 1.

A family of hash functions F with domain D and range R is (κ, μ)-*wise independent* if it is finite and $\forall\ y_1, y_2, \ldots, y_\kappa \in R, \forall$ distinct $x_1, x_2, \ldots, x_\kappa \in D$:

$$|\{f \in F : f(x_i) = y_i,\ i = 1, 2, \ldots, \kappa\}| \leq \mu \frac{|F|}{|R|^\kappa}. \tag{1}$$

Consequently, if a function $f \in F$ is chosen at random with all elements equally likely to be selected, then f will map any fixed set of h distinct points from D into R^h with a probability that is nearly uniform.

In these definitions, D and R are always finite. It is worth pointing out that any function is a hash function, and from the perspective of universal hashing, any function is a bad hash function. What matters are the statistical characteristics of the family members as quantified in (1). (Of course, we are also concerned with the program size and operation count associated with evaluating the functions in such a family.) It is important to notice that (κ, μ)-wise independence implies (j, μ)-wise independence for $j < \kappa$. Indeed, if we sum both sides of equation (1) over all $y_\kappa \in R$, the constraint on $f(x_\kappa)$ becomes the trivial $f(x_\kappa) \in R$, and the bound reverts to the exact requirement for $(\kappa - 1, \mu)$-wise independence.

In brief, the various notions of (κ, μ)-wise independence are all applicable, more or less, to statements of the same form: the probability that κ or fewer items behave in any way is, within a factor of μ, bounded by the probability of the behavior for the fully independent case. Formally, a Boolean statement $\mathcal{E}$ about the hashing of $x_1, x_2, \ldots, x_\kappa$ can be decomposed into a disjoint sum of atomic κ-way events:

$$\mathcal{E} = \bigvee_{(y_1, y_2, \ldots, y_\kappa) \in \Omega} \left(\bigwedge_{1 \leq i \leq \kappa} f(x_i) = y_i \right),$$

for some set Ω of κ-tuples. Moreover, any such $\mathcal{E}$ has a probability that is, up to the factor μ, bounded by the analogous probability that results from full independence and a uniform distribution.

There are, of course, many events that do not satisfy these restrictions. For example, if $f : D \to R$ is selected from a family of (κ, μ)-wise independent hash functions, then the

probability that, say $f(x_i) = 1$, for $i = 1, 2, \ldots, n$ might be only $\frac{\mu}{|R|^\kappa}$, and not nearly as small as $\frac{1}{|R|^n}$, no matter how large $n > \kappa$ might be. Similarly, we cannot use $\mu(1 - \frac{1}{|R|})^n$ as an estimate for the probability that $f(x_i) \neq 1$, for $i = 1, 2, \ldots, n$. Nevertheless, there are standard methods that often give satisfactory estimates for the probability of events $\mathcal{E}$ that concern the behavior of f on large subdomains. These procedures typically use collections of κ-way events that are implied by $\mathcal{E}$: if $\mathcal{E}$ implies $\bigvee_i e_i$, then $Prob\{\mathcal{E}\} \leq \sum_i Prob\{e_i\}$, and each $Prob\{e_i\}$ can be readily estimated, if each e_i concerns κ or fewer items. While more complex formulations are possible, they are unnecessary for this work.

For expositional simplicity, we will, unless stated otherwise, set $\mu = 1$, and simply refer to (κ)-wise independence.

The limited randomness provided by such classes of hash functions is frequently sufficient to achieve an expected performance for many randomized algorithms that is equivalent to the use of fully random hash functions. For example, Carter and Wegman exhibited the following universal classes of (κ, μ)-wise independent hash functions that map $[0, p-1]$ into $[0, n-1]$, where p is prime:

$$F_{(\kappa)} = \{f \mid f(x) = (\sum_{j=0}^{\kappa-1} a_j x^j \mod p) \mod n,\ a_j \in [0, p-1]\}, \tag{2}$$

and showed that these functions give a performance for hashing with separate chaining that is effectively indistinguishable from that for fully random functions. In this hashing scheme, the members of a set $\aleph \subseteq [0, p-1]$ are mapped, by a randomly selected function $f \in F_{(\kappa)}$ (independent of $\aleph$), into linked lists that are accessed by the array $B[0, n-1]$. When multiple items are mapped to a common address location j, the items are chained together in a list with the head list item stored in $B[j]$. The remaining list members are kept in auxiliary storage outside of B. Carter and Wegman used the fact that the sum of the expected j-th moments of the list lengths, when fully random hash functions are used, is essentially the same as that resulting from the use of random functions in $F_{(\kappa)}$, provided $j \leq \kappa$. For hashing with separate chaining, the expected average of the second

moments of the list lengths determines the expected retrieval time, whence pairwise independence guarantees an expected performance that is equivalent to that resulting from fully independent hash functions.

Subsequently, Mehlhorn and Vishkin presented more comprehensive families of universal hash functions [16].

Ranade used $O(\log n)$-wise independent hash functions to attain optimal expected performance (up to constant factors) for randomized routing schemes on an $n \times \log n$ butterfly network [18]. In this application, the routing delays subsume the $O(\log n)$ operation costs that result from hashing each memory reference by a function in $F_{(\Theta(\log n))}$. However, for (theoretical) PRAM emulation on butterfly networks (cf. [9, 18]), there would seem to be limited opportunity to use pipelining to mask latency for read intensive algorithms, as long as each address computation requires more than constant time.

$O(\log n)$-wise independent hash functions have also been shown to give optimal expected probe performance for double hashing ([19]). But this efficiency is only in terms of probe counts; the cost to compute a single hash address would be $c \log n$, without the hash functions and developments that we now present. Thus, the formal results of [19] are that in various models of closed hashing, dictionaries can be implemented with traditional $(c \log n)$-wise independent hash functions, which result in an average case performance of $O(\log n)$ operations per data access. This result would have more theoretical interest if highly independent constant-time hash functions were availaible.

Accordingly, it is reasonable to ask,

Is there an inherent κ-time penalty for computing

(κ)-wise independent hash functions, or can we do better?

The answer will turn out to be that faster functions are indeed programmable. However, a complete solution is fairly technical and cannot be stated in terms of the standard formulations for limited randomness. The definitions must be extended to expose addi-

tional statistical characteristics that affect the computational resources in cases where the independence exceeds the number of random seeds used to hash an individual key.

Families of random hash functions are often defined as a three-tier mapping $g \circ f \circ h$, where $h : D \to D_1$, $f : D_1 \to D_1$, and $g : D_1 \to R$. The underlying domain D might be huge, in which case a preliminary mapping h is used to map the hash keys into a smaller domain that is better suited for efficient computation. Thus, D_1 might be a finite field with elements that can be represented by a machine word or so, and the actual target range R might be a table index that is less well suited for defining families of uniformly distributed hash functions. For example, the Carter-Wegman class (2) actually defines a (κ)-wise independent family $f \in F$ on the field of integers $\bmod p$, and then uses $g(x) \equiv x \mod n$ to project the hashing onto the intended range $[0, n-1]$. Consequently, the resulting composition fails to achieve $(\kappa, 1)$-wise independence because the first $p \mod n$ points in $[0, n-1]$ have $\lceil \frac{p}{n} \rceil$ preimage points, whereas the remaining $n - (p \mod n)$ points have $\lfloor \frac{p}{n} \rfloor$ preimage points. This nonuniform postprocessing was the reason for introducing the parameter μ in the definition of (κ, μ)-wise independence. However, for the problems we consider, the pathologies associated with g are insignificant, whereas the irregularities associated with h turn out to have a greater affect on the underlying computational resources.

Formally, we analyze the computational costs for computing (κ)-wise independent mappings from a domain D into a range R. The underlying parameters are: 1) the number σ of initializing random seeds, 2) the evaluation time $T < \kappa$ for the hash function, and 3) the size of the hashing program and a provably necessary auxiliary storage array M, which, it will turn out, we can presume to hold pseudo-random numbers that have been precomputed from σ initial seeds. Additional parameters are the domain and range sizes $|D|$ and $|R|$. This paper gives a lower bound to show that M must store at least $|D|^\epsilon$ words from R, for suitable fixed $\epsilon < 1$. We also show how to circumvent this lower bound. By allowing an asymptotically negligible chance (which is yet another

parameter) that the hash function will fail to achieve (κ)-wise independence, the lower bound will still apply, but for an effective domain size $|D_1|$, which might be far smaller than $|D|$.

The exact details include additional parameters, but they play a minor role in the resource requirements. In all cases, the requisite number of random seeds remains modest: $\sigma = \Theta(\kappa)$![†]

We also present two kinds of algorithmic formulations for such high performance hash functions. In particular, nonconstructive existence arguments are used to show that algorithms might, in principle, match the resource constraints predicted by the lower bound. Our (nonconstructive) hash functions require a precomputation phase where the σ random seeds are used to precompute $|D|^\epsilon$ pseudo-random values that are stored in an auxiliary array M. Then a suitable algorithm can compute highly random hash values by using its hash key deterministically to select a small number T of elements from M and by simply returning the sum of these elements $\bmod |R|$, or, say, their bitwise exclusive-or. The computation can be formulated to use, say, just twice as many elements from M as are proved necessary by our bound. Moreover, this count T turns out to be constant: $T = \Theta(1)$. On the other hand, the question of which T words should be selected from M lies at the heart of the nonconstructive existence argument.

While the question of how to select these T random words effectively is still open, we show that hash functions with very high independence are indeed programmable, provided we accept the less satisfactory evaluation time of $T^{\Theta(T)}$, where T is the $O(1)$ operation count predicted by nonconstructive methods.

[†]For very large domains, the prehashing in Appendix 1, which is based on [7,16] uses an additional $\log\log|D|$ bits, which are provably necessary (cf. [7, 16]). On the other hand, the convention that words can be processed in unit time is suspect for domains where such a term would matter.

1.2 Related Work

A preliminary version of this work appeared as an extended abstract in [21]. The current presentation includes more efficient existential formulations, better probabilistic constructions, a stronger and more general lower bound, and an improvement in the applications. In the interim, several works have appeared that use the properties of these high performance hash functions [8, 10, 14].

The theory of hashing has also progressed along several independent lines. In particular, Dietzfelbinger et al. pursued the perfect hashing methods begun by Fredman et al. [7] to develop hash functions that can locate data without collisions [6]. The processing is fully dynamic, with an average insertion time of $O(1)$ steps per insertion, and a resulting 1 table probe per data request. While these techniques have many useful properties, and might comprise the methods of choice for many applications, they do not construct functions with the statistical randomness that is the primary objective of this work. Further, the functions require auxiliary storage of $\Omega(n)$ bits, which is in excess of the storage we will use.

Many other works use hash functions to ameliorate resource conflicts that arise in distributed computation. For example, Karp et al. use the highly random constant-time hash functions described in the preliminary version of this work to resolve various problems of contention at the memory module level [10]. Their model is designed to avoid some of the idealized characteristics of PRAMs, where contention only occurs at the level of memory cells. Additional machine emulation results that rely upon the hash functions of this paper include developments by Goldberg et al. [8] and Matias et al. [14]. On the other hand, some machine contention can be resolved with much simpler hash functions. For example, Kruskal et al. use such an approach to emulate an $(n^{1+\epsilon})$-processor PRAM on an n-processor machine [12].

1.3 The organization

This paper is organized as follows. Section 1 concludes with a brief description of the computational model and an introduction to the calculus of limited independence. Section 2 analyzes three random function formulations, which range from the nonconstructively defined but extremely efficient, to the programmable (with code). They all run in constant time and can be as much as $(|D|^{\delta})$-wise independent, for different, suitably small constants $\delta > 0$.

Section 3 presents lower bounds for the resource requirements of fast hash functions. Section 4 discusses the intrinsic resource tradeoffs for the general hashing formulations of Section 2, their lower bounds, and their equivalence to a fundamental problem in graph theory. Section 5 gives a few applications and Section 6 presents the conclusions.

1.4 Computational assumptions

The performance results are based on a variety of assumptions that characterically hold for most hashing problems, as well as some decisions to use problem descriptors aimed at keeping the parameters at hand as simple as possible. The first point of clarification is that the underlying problem size, in all hashing problems, will be n. We state this because n will sometimes be only implicitly represented by, say, a domain size parameter m, which will later be set to n^{ℓ}, for suitabe ℓ. Second, we assume that for some constant $r \geq 2$, the family of hash functions we construct need only be (κ, μ)-wise independent with a probability exceeding $1 - n^{-r}$. Third, there is an underlying set of data that is to be hashed, which has a size that is at most n. It is easy to adapt the results to larger data sets, but to avoid clutter, we will perform these adaptations only when necessary.

Fourth, either the domain size $|D|$ is polynomial in n, or there is "no computation charge" for premapping elements from a very large space D into, say, $[0, n^{\ell} - 1]$ for some constant $\ell \geq r + 2$. It will suffice to use a pairwise independent hash function (i.e., a randomly selected member of universal class of a pair-wise independent hash functions) for the premapping $h : D \to [0, n^{\ell} - 1]$.

Fifth, the range R is no larger than n^γ for some constant γ. In fact, we sometimes assume that $n \leq |R|$. This assumtion is harmless; if $|R|$ is much less than n, it will suffice to pick an n that is a multiple of $|R|$, and postprocess the hash functions mod$|R|$.

The underlying computational model (once the domain is reduced in size) is the standard Random Access Machine.

Lastly, we assume that there is a source of truly independent uniformly distributed random seeds in any interval $[0, \rho-1]$, and that modest quantites of them are available for use. In particular, if a hash function is to be (κ)-wise independent, then $\Theta(\kappa)$ random seeds can be supplied as initializating input to the hashing algorithm. Of course, if strings of $\theta(\kappa)$ seeds have relative probabilities of occurance that are not equal but instead differ by as much as a factor of μ, then the resulting hash functions will have statistics that have the same variability.

1.5 A primer on limited independence and the simplest derived inequalities

The probabilistic inequalities used in this paper all follow from a few closely related formulations. Let $\mathcal{X}(*) : (T, F) \rightarrow (1, 0)$ be the indicator function, which maps Boolean expressions into 1 when they are true and 0 otherwise. Let g be a nonnegative increasing function, and let X be a random variable. Then $\frac{g(X)}{g(a)}$ is at least 1 whenever $X > a$, and is nonnegative everywhere. Thus $Prob\{X > a\} = Prob\{g(X) > g(a)\} = \mathrm{E}[\mathcal{X}(\frac{g(X)}{g(a)} > 1)] \leq \mathrm{E}[\frac{g(X)}{g(a)}] = \frac{\mathrm{E}[g(X)]}{g(a)}$. There are several functions g commonly used for proving probabilistic inequalities of the form

$$Prob\{X > a\} \leq \frac{\mathrm{E}[g(x)]}{g(a)}. \tag{3}$$

When X is itself nonnegative, setting $g(x) = x$ gives what is sometimes referred to as Markov's inequality.

This scheme also works if g is an increasing multinomial or multivariate function that is never negative on its domain. Let, for example, the random variable Φ be the sum of n Bernoulli trials: $\Phi = z_1 + z_2 + \cdots + z_n$, where z_k is 1 if, say, the hash function

f maps x_k to memory module 1, and 0 otherwise. In this case z_k is itself the disjoint sum of many atomic events: $z_k = \bigvee_j (f(x_k) = j)$, where the j index ranges over all addresses in module 1. If f is selected from a (κ, μ)-wise independent family, then, as is straightforward to verify, the Bernoulli Trials will be (κ, μ)-wise independent. A suitable multinomial g would be $g(z_1, z_2, \ldots, z_n) = \binom{\Phi}{\kappa}$, which has a very natural interpretation. In this case, $\binom{\Phi}{\kappa}$ is the sum of all products of subsets of κ different z's: $\binom{\Phi}{\kappa} = \sum_{1 \le i_1 < i_2 < \cdots < i_\kappa \le n} z_{i_1} z_{i_2} \cdots z_{i_\kappa}$.

We recall that the expectation and multiplication commute for independent random variables: $\mathrm{E}[YX] = \mathrm{E}[X]\mathrm{E}[Y]$ if X and Y are independent. In the case of (κ, μ)-wise independence, $\mathrm{E}_\kappa[X_1 X_2 \cdots X_\kappa] \le \mu \prod_{1 \le j \le \kappa} \mathrm{E}_\infty[X_j]$, for nonnegative (κ, μ)-wise independent random variables $X_1, X_2, \ldots, X_\kappa$, where E_κ denotes the expectation under (κ, μ)-wise independence, and E_∞ denotes the expectation under full randomness. More generally, we note that if the X_i-s are (κ, μ)-independent, then so are $\{g_i(X_i)\}_{j=1}^{\kappa}$ for any set of functions g_i. (These facts can be verified by formulating events as the logical-or of κ-way atomic events that assign each X_i fixed values.) Under (κ, μ)-wise independence, it follows that

$$Prob\{\Phi > a\} \le \frac{\mathrm{E}_\kappa[\binom{\Phi}{\kappa}]}{\binom{a}{\kappa}} \le \frac{\mu}{\binom{a}{\kappa}} \sum_{1 \le i_1 < i_2 < \cdots < i_\kappa \le n} \mathrm{E}_\infty[z_{i_1}]\mathrm{E}_\infty[z_{i_2}] \cdots \mathrm{E}_\infty[z_{i_\kappa}].$$

The bound is useless if $a < \kappa$, since the denominator would be zero; in such a case, we might replace κ with a smaller value.

In many of our applications, $\mathrm{E}_\infty[z_k] = p$ for some expression p that is independent of k. In these cases, $\mathrm{E}_\infty[\binom{\Phi}{\kappa}] = \binom{n}{\kappa} p^\kappa$, and hence $Prob\{\Phi > a\} \le \mu \frac{\binom{n}{\kappa} p^\kappa}{\binom{a}{\kappa}}$.

These derivations suggest that we will need some basic combinatorial inequalities. Some use the notation $n^{\underline{k}} \equiv n(n-1)\ldots(n-k+1)$. For example, when $m > n$, then we can write $\frac{n^{\underline{\kappa}}}{m^{\underline{\kappa}}} < (\frac{n}{m})^\kappa$, and $\frac{m^{\underline{\kappa}}}{n^{\underline{\kappa}}} > (\frac{m}{n})^\kappa$ without appealing to any estimate for factorials.

A fairly elementary form of Stirling's Formula says that $n! > n^{n+1/2} e^{-n}$. Sometimes even more naive formulations such as $n! > n^n e^{-n}$ will suffice. For example, the last inequality implies that $n^{\underline{k}} > n^k e^{-k}$. (Indeed, to see that the former implies the latter,

observe that $n^{\underline{\kappa}} > \frac{\kappa^\kappa}{e^\kappa \kappa!} n^{\underline{\kappa}} = \frac{\kappa^\kappa}{e^\kappa} \cdot \frac{n^{\underline{\kappa}}}{\kappa^{\underline{\kappa}}} > (\frac{\kappa}{e})^\kappa \cdot (\frac{n}{\kappa})^\kappa = (\frac{n}{e})^\kappa$.) It now follows that

$$\frac{n^k e^{-k}}{k!} \leq \binom{n}{k} \leq \frac{n^k}{k!},$$

and we will bound many combinatorial expressions by replacing $\binom{n}{k}$ by $\frac{n^k}{k!}$, if the expression appears in a numerator, and by $\frac{n^k e^{-k}}{k!}$, if it is in a denominator. As an immediate application, we have:

$$\mu \frac{\binom{n}{\kappa} p^\kappa}{\binom{a}{\kappa}} \leq \mu \frac{(np)^\kappa}{a^{\underline{\kappa}}} \leq \mu (\frac{npe}{a})^\kappa,$$

which further simplifies the preceding estimates for $Prob\{\Phi > a\}$, when Φ is the sum of n (κ, μ)-wise independent Bernoulli trials with probability of success p. Although much better estimates can be derived, they will not be needed.

Many powerful inequalities come from setting $g(X) = e^{\lambda X}$, in (3), and then solving for the value of $\lambda > 0$ that gives the best bound. If X is the sum of n mutually independent identically distributed random variables z_i, then $\mathrm{E}[e^{\lambda X}] = \mathrm{E}[e^{\lambda z_1}]^n$, and $Prob\{X > a\} < e^{-\lambda a} \mathrm{E}[e^{\lambda z_1}]^n$. If $X = z_1 + z_2 + \ldots + z_\kappa$ where $\{z_i\}_{i=1}^\kappa$ are random variables that enjoy (κ, μ)-wise independence, then $\mathrm{E}_\kappa[e^{\lambda X}] \leq \mu \prod_{i=1}^\kappa \mathrm{E}_\infty[e^{\lambda z_i}]$. A more complete set of these estimates for cases with limited independence can be found in [20].

Sometimes an event $\mathcal{E}$ will have a corresponding predicate $Pred(*,*)$ and sets U and W that satisfy the following: if $\mathcal{E}$ occurs, then there exits $u, \subset U$ and $w \subset W$ such that $Pred(u, w)$ is true. This implies that $Prob\{\mathcal{E}\} \leq \sum_{u \subset U,\, w \subset W} \mathrm{E}[\mathcal{X}(Pred(u, w))]$, where $\mathcal{X}$ is the 0–1 indicator function that maps true,false into 1,0. We will take care to ensure that U and W are suitably restricted so that (κ, μ)-wise independence can be used to evaluate the expectations.

Armed with this set of counting arguments, we proceed to the problem of building fast computable hash functions.

2. The hash functions

The first step is to formalize a somewhat stronger and occasionally more applicable notion of restricted randomness.

Definition 2.

A family of hash functions F with domain D and range R is *r-practical* (κ, μ)*-wise independent* if for any subset $D_0 \subset D$, with $|D_0| \leq \sqrt{|R|^r}/2$, there is a subset of functions $\widehat{F} \subseteq F$ where $|\widehat{F}| \geq |F|(1 - 2\frac{\binom{|D_0|}{2}}{|R|^r})$ and the following holds:

$$\begin{aligned} &\forall\ y_1, y_2, \ldots, y_\kappa \in R,\ \forall \text{ distinct } x_1, x_2, \ldots, x_\kappa \in D_0 : \\ &\qquad \frac{|\widehat{F}|}{\mu |R|^\kappa} \leq |\{f \in \widehat{F} : h(x_i) = y_i,\ i = 1, 2, \ldots, \kappa\}| \leq \frac{\mu |\widehat{F}|}{|R|^\kappa}. \end{aligned} \tag{4}$$

We define F to be *uniformly r-practical* (κ, μ)*-wise independent* if (4) holds for some $|\widehat{F}| \geq |F|(1 - \frac{1}{|R|^r})$ with $D_0 = D$.

Besides accommodating small percentages of faulty hash functions, this definition differs from the usual formulations by specifying two-sided constraints. This formulation (which becomes an equality when $\mu = 1$) is intended to support inclusion-exclusion arguments based on (κ)-wise independent hash functions. (In [19], for example, such a two-sided bound is essential.) As a practical matter, this additional constraint is not particularly burdensome. While most constructions of (κ, μ)-wise independent function classes have focussed on one-sided constraints as typified by Definition 1, actual constructions, such as the polynomials $F_{(\kappa)}$ of equation (2) usually satisfy the more restrictive standard of Definition 2.

2.1 Existence arguments

We now examine the problem of constructing families of (κ)-wise independent hash functions that map a domain $D = [0, m-1]$ into a range $R = [0, \rho - 1]$. The basic motivation for the approach comes from considering the following question. Imagine, for the moment, that $D = R$, and is of prime size. Then we can use κ random elements from D as coefficients in equation (2) to construct a (κ)-wise independent hash function.

Evidently, evaluation requires $O(\kappa)$ time. If m random elements are provided, then table lookup gives an $O(1)$ time (m)-wise independent hash function. What sort of random functions can be constructed from m^ϵ random seeds?

For the purposes of this section, we will require that the underlying storage be sufficient to hold the random function code, including its random seeds. Our model of computation is the Random Access Machine, where both memory access and the basic arithmetic and logic operations can be executed on words in unit time (cf.[1]). Each word will comprise $\lceil \log_2 \rho \rceil$ or possibly $\lceil \log_2 m \rceil$ bits.

We temporarily suppress the issue of program size and prove the existence of families of fast highly independent hash functions that map $[0, m-1]$ into $[0, \rho-1]$ and use m^ϵ words of random input. We will also initially ignore all preprocessing and postprocessing steps as well as any the concern associated with huge domains to study the problem of constructing fully $(\kappa, 1)$-wise independent hash functions that map $D = [0, m-1]$ into a suitable $R = [0, \rho-1]$, given an auxiliary array of m^ϵ random words from R, for some[†] $\epsilon < 1$. Evidently, any random hash function must have a mechanism to associate each element in D with a few of these random words, as otherwise no random computation can result. If the association is not adaptive[‡] then it can be represented by a bipartite graph G on the vertex sets D and $D^\epsilon \equiv [0, m^\epsilon - 1]$. Such a bipartite graph must associate at least l random numbers with each set of l elements from D, for $l \leq \kappa$, as otherwise there are not enough degrees of freedom to achieve (κ)-wise independence. Suitable graphs are formalized as follows.

[†] We will avoid the clutter of floors and ceilings as long as possible, and assume that the exponent ϵ has been chosen to make the resulting expression such as m^ϵ an integer.

[‡] The lower bound will include adaptive probing, where the location of the next random key to retrieve can depend on the value of other such data that has already been read. The purpose of mentioning a nonadaptive approach is because it is more intuitive, and helps explain all of our constructions, which, it turns out, are nonadaptive.

Definition 3.

Let an (m, ϵ, d, κ) *local concentrator* be a bipartite graph on sets of vertices I (inputs) and O (outputs), where $|I| = m$, $|O| = m^\epsilon$, and the following hold. Each input has outdegree d. Every set of j inputs, for $j \leq \kappa$, has edges to some set of j or more outputs.

Our next observation is that such graphs exist, even with very small outdegree d. In the following lemma, the parameter r is extraneous because the existence result holds for $r = 0$. It is included to compress two proofs into one. By increasing r from 0 some positive quantity, a suitably structured random graph will be as described with a probability exceeding $1 - m^{-rd}$ (which can be with overwhelming likelihood) as opposed to a probability exceeding zero. The same issue applies to the parameter r in Lemma 4.

Lemma 1. Let d, κ, m and m^ϵ be positive integers with $\epsilon < 1$. Suppose that $r \geq 0$ and $d \geq \frac{1+\epsilon+r}{\epsilon - \frac{\log \kappa}{\log m}}$. Let $G = (V, E)$ be a random bipartite graph with input vertices I, and output vertices O, where $|I| = m$, and $|O| = m^\epsilon$, and where each vertex in I has edges to d distinct randomly selected neighbors in O. Then with probability exceeding $1 - m^{-rd}$, G is an (m, ϵ, d, κ) local concentrator.

Proof: We use standard probabilistic arguments (cf. [24]) to estimate the probability that G fails to have j outputs for some set of j inputs, for $j \leq \kappa$. By construction, failure cannot occur for $j \leq d$. For larger aggregates of size at most κ, the probability of a failure is bounded by the expected number of pairs $(u \subset I,\ w \subset O)$, where $|u| = j$, $|w| = j - 1$, and all jd edges from u have destinations within w, for $d < j \leq \kappa$. Evidently, the probability that the jd edges are so selected, for any fixed $(u,\ w)$, is

$$\left(\frac{\binom{j-1}{d}}{\binom{m^\epsilon}{d}} \right)^j < \left(\frac{j-1}{m^\epsilon} \right)^{jd},$$

(where the inequality is strict because $d > 1$). The number of candidate (u, w) pairs is

just

$$\binom{m}{j}\binom{m^\epsilon}{j-1}.$$

Thus:

$$\begin{aligned} Prob\{failure\} &< \sum_{d<j\le\kappa} \binom{m}{j}\binom{m^\epsilon}{j-1}\left(\frac{j-1}{m^\epsilon}\right)^{jd} \\ &< \sum_{d<j\le\kappa} \frac{m^j m^{j\epsilon-\epsilon} j^{jd}}{j!(j-1)!m^{\epsilon jd}} \\ &< m^{-\epsilon} \sum_{d<j\le\kappa} \frac{1}{j!(j-1)!}\frac{m^{j+j\epsilon}\kappa^{jd}}{m^{\epsilon jd}}. \end{aligned}$$

Now, the constraint for d can be written as $m^{d\epsilon} \ge \kappa^d m^{1+\epsilon+r}$, whence $m^{-r} \ge \frac{\kappa^d m^{1+\epsilon}}{m^{d\epsilon}}$. Substituting gives:

$$\begin{aligned} Prob\{failure\} &< m^{-\epsilon} \sum_{d<j\le\kappa} \frac{m^{-rj}}{j!(j-1)!} \\ &< m^{-rd}. \quad \blacksquare \end{aligned}$$

Setting $r = 0$, we see that a randomly constructed graph fails to be an (m, ϵ, d, κ)-local concentrator with a probability that is less than 1. It follows that such a construction will succeed with positive probability and hence these graphs do indeed exist.

We have, as yet, no hash function; but each element, at least, is now associated with a few random values. The obvious use for these values is as coefficients of a hashing polynomial. By increasing the number of random values used in this calculation, we can turn a local concentrator into a calculation procedure for $(\kappa, 1)$-wise independent hash functions.

Let G be an (m, ϵ, d, κ) local concentrator. Let p be prime. For each input i in G, let i's d neighbors in G be stored in the set $Adj(i)$. Let M_w be an $m^\epsilon \times d$ array of words in $[0, p-1]$, whose concatenated contents is the very long word $w \in [0, p-1]^{m^\epsilon d}$.

Define the random hash function

$$f_w^G(i) = \left(\sum_{0\le l<d} \sum_{j\in Adj(i)} i^l M_w(j,l)\right) \mod p.$$

Thus, computing $f_w^G(i)$ requires $d^2 - 1$ additions and $d - 1$ multiplications plus a comparable number of modular divisions. The result turns out to be (κ)-wise independent

when the concatenated contents of the storage array M_w is equally likely to be any value in $[0, p-1]^{dm^\epsilon}$.

Theorem 1. Let G be an (m, ϵ, d, κ) local concentrator. Then $\{f_w^G\}_{w \in [0,p-1]^{dm^\epsilon}}$ is a $(\kappa, 1)$-wise independent family of hash functions mapping $[0, m-1]$ into $[0, p-1]$.

Proof: Let $x_1, x_2, \ldots, x_\kappa$ be any κ distinct values in $[0, m-1]$, and $y_1, y_2, \ldots, y_\kappa$ be κ arbitrary values in $[0, p-1]$. We must show that for all x and y assignments, there are exactly the same number of functions f_w that satisfy the system

$$f_w^G(x_i) = y_i \mod p, \text{ for } i = 1, 2, \ldots, \kappa. \tag{5}$$

Now, the unknowns, in this system, are the word assignments to the array M_w, and the system is linear in these variables. The equations in (5) comprise κ constraints in dm^ϵ unknowns. So if the system enjoys linear independence, then the set of κ equations in (5) will have exactly $p^{dm^\epsilon - \kappa}$ different w words that are solutions, which ensures the precise uniformity required of (κ)-wise independence. It follows that we need only establish the linear independence of all systems defined in (5).

We prove the linear independence by contradiction. In the context of Lemma 1, $[0, m-1]$ plays the role of I, and $[0, m^\epsilon - 1]$ is O. The linear system $f_w^G(\overline{x}) = \overline{y} \mod p$ of all m equations has the explicit formulation

$$\sum_{k=0}^{d} (\mathcal{N})^k \times \mathcal{A} \times (M_{j,k})_{j \in O} = (y_0, y_1, \ldots, y_{m-1})^T \mod p, \tag{6}$$

where $\mathcal{N}^k$ is the diagonal matrix with i^k in location (i, i), for $i = 0, 1, \ldots, m-1$ (with $\mathcal{N}^0$ is set to the identity matrix), and $\mathcal{A} = (a_{i,j})_{i \in I,\, j \in O,}$ is the adjacency matrix for G, which has has m rows, m^ϵ columns, and d 1-s in each row. The representation $(M_{j,k})_{j \in O}$ is intended to denote the k-th of d column vectors, which each have $|O| = m^\epsilon$ rows. Taken together, they comprise the entries in the auxiliary array M of random words.

The proof will exploit the structural characteristic of G to establish the linear independence of any subsystem of $|I_0| \leq \kappa$ equations, for $I_0 \subset [0, m-1]$:

$$b_i = \left(\sum_{0 \leq k < d} \sum_{j \in Adj(i)} i^k M_w(j, k) \right) \mod p, \text{ for } i \in I_0.$$

Accordingly, suppose that I_0 names the row indices in a subsystem of (6) is linearly dependent, and which has no proper subset that is dependent.

By definition, I_0, in the local concentrator, has $d|I_0|$ edges, which reach at least $|I_0|$ outputs. Let O_0 comprise the vertices adjacent to I_0 in G. Then the average number of edges received from I_0 by a vertex in O_0 is $d|I_0|/|O_0|$ which is at least 1 and at most d, since $|I_0| \leq O_0 \leq d|I_0|$. By the pigeonhole principle, there must be an output $o \in O_0$ having exactly q edges that originate in I_0, for some q where $1 \leq q \leq d$. Let I_1 be this set of neighbors of o in I_0: $I_1 = \{i \in I_0 : o \in Adj(i)\}$. Let $I_1 = \{i_1, i_2, \ldots, i_q\}$, where $1 \leq q \leq d$. Now consider the linear subsystem with rows indexed by I_1 and columns restricted to the variables $M(o,k)$, where $k = 0, 1, \ldots, d-1$. This system comprises the following submatrix of a Vandermonde matrix:

$$\begin{pmatrix} 1 & i_1 & i_1^2 & \ldots & i_1^{d-1} \\ 1 & i_2 & i_2^2 & \ldots & i_2^{d-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & i_q & i_q^2 & \ldots & i_q^{d-1} \end{pmatrix}.$$

As is well known, such a subsystem cannot be linearly dependent since no two rows are the same, and $q \leq d$. Since none of the rows with indices in $I_0 - I_1$ have any of the variables $M(o,0), M(o,1), \ldots, M(o,d-1)$ present, (that is, they are present with coefficients of zero) the only way a linear combination of the rows can add to the zero vector is if each row in I_1 has a coefficient of zero. Thus, the assumption that the system is dependent and minimal is contradicted. ∎

So far, we have a probabilistic fast hashing procedure that is (κ)-wise independent, uses dm^ϵ random words of $\log p$ bits, and requires $\Theta(d^2)$ operations. The construction gives a generic transformation from a graph rich in matchings to a family of highly random functions. We now give a more efficient construction that uses better random graph properties, with a constant factor degradation in the outdegree d, but where only one random value is stored per output destination. The construction uses a sparse bipartite graph where every set of κ rows of its adjacency matrix is not only linearly independent, but also exhibits linear independence over finite commutative groups such

as the integers modρ for any integer ρ.

Definition 4.

Let G be a bipartite graph on sets of vertices I (inputs) and O (outputs), where $|I| = m$, $|O| = m^\epsilon$. Let $Adj(i)$, for $i \in I$, be the set of i's neighbors in O. We say that G is (m, ϵ, d, κ) *locally peelable* if each input node has an outdegree of at most d and the following holds: if I_0 is an arbitrary set of κ or fewer input vertices, then there is an $i_0 \in I_0$ such that

$$Adj(i_0) - \bigcup_{i \in I_0 - \{i_0\}} Adj(i) \neq \emptyset.$$

(Some node in I_0 has a neighbor that is not a neighbor of any other node in I_0.)

Lemma 2. Let $D = [0, m-1]$, and $R = [0, \rho - 1]$. Let $\bigoplus$ be denote addition modρ or any other commutative group operator that is closed over R. Let G be (m, ϵ, d, κ) locally peelable, with $\epsilon < 1$. For each input i in G, let i's neighbors in G be stored in the set $Adj(i)$. Let M_w be an array of m^ϵ words from R, where the concatenated content of M_w comprises the very long word $w \in [0, \rho - 1]^{m^\epsilon}$.

Define the hash function

$$f_w^G(i) = \bigoplus_{j \in Adj(i)} M_w(j), \text{ for } i \in D.$$

Then $\{f_w^G\}_{w \in [0, \rho-1]^{m^\epsilon}}$ is a $(\kappa, 1)$-wise independent family of hash functions mapping D into R.

Proof: Consider the subsystem (in unknowns w) $f_w^G(x_i) = y_i$, for $i = 1, 2, \ldots, \kappa$, which assigns κ values in R to κ distinct input vertices in D. Since the system is peelable, it can be permuted into a system where the first κ columns comprise an upper triangular matrix with 1's along its diagonal. The first row corresponds to an input x_{i_0} reaching an output o_0 that is not a neighbor of any node in $\{x_i\}_{i=1}^{\kappa} - \{x_{i_0}\}$. The columns are permuted so that the first column corresponds to o_0. Then the same reordering is recursively applied to the remaining $\kappa - 1$ equations in unknowns $O - \{o_0\}$. Given such an upper triangular permutation of the system, there are $m^\epsilon - \kappa$ column variables that

are not among the columns that contain the κ diagonal entries, and we are free to assign arbitrary values from R to these variables. Then the remaining κ variables will have a unique solution that can be found by back-solving via the $\oplus$ operator. Thus, the number of solutions to any κ such equations in the m^ϵ unknowns is exactly $\rho^{m^\epsilon-\kappa}$, which ensures that the f_w^G are $(\kappa,1)$-wise independent. ∎

We now show that some random graphs are locally peelable.

Definition 5.

Let an (m,ϵ,d,κ) *local expander* be a bipartite graph on sets of vertices I (inputs) and O (outputs), where $|I|=m$, and $|O|=m^\epsilon$, and the following hold. Each input has a positive outdegree bounded by d. Any set of j inputs, for $1<j\le\kappa$, has edges to at least $\lfloor jd/2\rfloor+1$ different outputs.

Lemma 3. An (m,ϵ,d,κ) *local expander* is (m,ϵ,d,κ) locally peelable.

Proof: The proof is by contradiction. Suppose that I_0 is a smallest set of input variables that is not peelable. Obviously $|I_0|>1$. Suppose $|I_0|\le\kappa$. By definition, the input variables of I_0 have at least $\lfloor jd/2\rfloor+1$ different output variables, which have, on average, at most $jd/(\lfloor jd/2\rfloor+1)<2$ different input neighbors in I_0. By the pigeonhole principle, some output vertex o_0 must therefore have just one input neighbor i_0 in I_0. We peel off this input, and observe that the subgraph with $|I_0|-1$ input variables is peelable, whence the supposition that $|I_o|\le\kappa$ must be false. ∎

As with Lemma 1, the following existence argument includes an extraneous parameter r to increase the likelihood of success from something positive to a probability overwhelmingly close to 1.

Lemma 4. Let d, κ, m and m^ϵ be positive integers with $\epsilon<1$. Suppose that $r\ge 0$ and $\epsilon>\frac{2+r}{d}+\frac{1+\ln\frac{d}{2}+\ln\kappa}{\ln m}$. Let $G=(V,E)$ be a random bipartite graph with input vertices I, and output vertices O, where $|I|=m$, and $|O|=m^\epsilon$, and where each vertex in I has edges to d distinct randomly selected neighbors in O. Then with probability exceeding

$1 - m^{-r}$, G is an (m, ϵ, d, κ) local expander.

Proof: Rewriting the bound for ϵ gives $m^{\epsilon d/2} > m(\kappa de/2)^{d/2} m^{r/2}$, so that

$$\frac{m(jde/2)^{d/2}}{m^{\epsilon d/2}} < m^{-r/2} \text{ for } j \leq \kappa. \tag{7}$$

Proceeding as in Lemma 1 gives the following estimates for the probability that G is not a local expander.

$$\begin{aligned} Prob\{\text{failure}\} &< \sum_{1<j\leq\kappa} \binom{m}{j} \binom{m^\epsilon}{\lfloor jd/2 \rfloor} \left(\frac{\lfloor jd/2 \rfloor}{m^\epsilon}\right)^{jd} \\ &< \sum_{1<j\leq\kappa} \frac{m^j m^{\epsilon jd/2} (\lfloor jd/2 \rfloor)^{jd}}{j!(\lfloor jd/2 \rfloor)! m^{\epsilon jd}} \end{aligned}$$

whence approximating $(\lfloor jd/2 \rfloor)!$ via our two Stirling estimates, and simplifying gives

$$Prob\{\text{failure}\} < \sum_{2\leq j\leq\kappa} \left(\frac{m(jde/2)^{d/2}}{m^{\epsilon d/2}}\right)^j \frac{1}{j!}.$$

Applying (7) gives

$$Prob\{\text{failure}\} < \sum_{2\leq j\leq\kappa} m^{-rj/2}/j! < m^{-r} \sum_{2\leq j} \frac{1}{j!} < m^{-r}. \quad \blacksquare$$

Setting $r = 0$ shows that such a graph exists. Of course, actual use of this lemma will require that ϵ be less than 1, which requires that $d \geq 3$.

Combining Lemmas 2, 3 and 4 gives the following.

Theorem 2. Let d, κ, m and m^ϵ be positive integers with $\epsilon < 1$. Suppose that $1 > \epsilon \geq \frac{2}{d} + \frac{1+\ln d+\ln \kappa}{\ln m}$, there are fixed programs that, for each input from $[0, m-1]$, "$\oplus$" together d words from an array of m^ϵ random words from $[0, \rho-1]$ to compute an $(\kappa, 1)$-wise independent hash function mapping $[0, m-1]$ into $[0, \rho-1]$.

Proof: Let G be an (m, ϵ, d, κ) local expander. Lemma 4 ensures that such a graph exists, when $\epsilon \geq \frac{2}{d} + \frac{1+\ln d+\ln \kappa}{\ln m}$. Lemma 3 ensures that G is (m, ϵ, d, κ) locally peelable. Lemma 2 defines

$$f_w^G(i) = \bigoplus_{j \in Adj(i)} M_w(j),$$

and shows that $\{f_w^G\}_{w\in[0,\rho-1]^{m^\epsilon}}$ is an $(\kappa,1)$-wise independent family of hash functions mapping $[0,m-1]$ into $[0,\rho-1]$. ∎

In particular, $\oplus$ can be addition modρ, or $\oplus$ can be the bitwise exclusive-or operator, where ρ is power of 2 and the integers in $[0,\rho-1]$ are regarded as binary strings of length $\log\rho$.

It is worth observing that the m^ϵ-word seeds w as used in Theorem 2 can be produced by random members of a universal class H of $(d\kappa,\mu)$-wise independent hash functions. That is, let $h\in H$ map $[0,m^\epsilon-1]$ into $[0,\rho-1]$, and comprise a universal class of $(d\kappa,\mu)$-wise independent has functions. Let w_h be the concatenation of $h(0),h(1),\ldots,h(m^\epsilon-1)$. Then the family $\{f_{w_h}^G\}_{h\in H}$ as used in Theorem 2 is (κ,μ)-wise independent. To see that this is so, observe that each set of assignments to a j-tuple of elements in D induces (via a peelable linear system of j constraints) a collection of solution values in at most $d\kappa$ (not necessarily identical) index locations of the auxiliary array M. These assignments induce a disjoint partitioning of the set of all $w\in[0,\rho-1]^{|O|}$, when all possible seedings are available, (and where one of the partitions corresponds to assignments that do not produce the desired h-tuple of values). The sets are disjoint because they correspond to M-locations and M-location-values read in the process of computing the desired h-tuple of hash values, and this process is deterministic. So if these different assignments occur with the weighting that corresponds to all $|O|$ locations being equally likely to have any seed value in $[0,\rho-1]$, the resulting class of hash functions $F=\{f_w^G\}$ will be $(\kappa,1)$-wise independent by Theorem 2. If each individual partition subset of $d\kappa$ or fewer assignments has a probability of occurrence that is reweighted by a factor of $\mu\in[\mu_1,\mu_2]$ then the resulting aggregate probability that F computes the h-tuple is likewise reweighted by a factor that is within this interval.

Consequently, the space-time tradeoff, for families of fast highly independent hash functions, is not a function of the number of random seeds that must be specified (which is just $\Theta(\kappa)$) but is really a matter of intrinsic storage requirements for the auxiliary

storage array M.

While the second construction gives a more efficient family of hash functions, and also provides a generic procedure that turns a good graph into a family of hash functions, it does not quite supersede the first construction because there are no known explicit graphs of either type. Should a short (deterministic or effective probabilistic) algorithm be found, which builds local concentrators where an input's adjacency list can be generated in constant time, then fast highly independent hash functions will follow. Similarly, effective procedures for constructing local expanders will yield even better hash functions.

2.2 Asymptotically compact constructions and their randomization

From a positive perspective, we have proven that one good graph is all we need: the contents of the auxiliary random seed array M defines the different members in the associated family of hash functions. Furthermore, we have shown that good graphs not only exist, but can even be formulated so that randomized constructions can build them with very high likelihood.

Unfortunately, no deterministically defined graph has been proven, as yet, to satisfy either expander-like formulation, and the problem seems to be quite difficult. Worse still, any such graph would be useless, since its size would be prohibitively large. And nobody knows how to design an algorithm that can construct, in constant time, the adjacency list for the input vertices of such graphs.

We now show that from a theoretical perspective (where impractical constants might be tolerated), there are asymptotically spatially compact programmable formulations of constant-time (n^ϵ)-wise independent hash functions. In particular, we will show that Cartesian products can be used to attain compact representations of (wildly) less efficient hash functions, where we forgo some randomness, increase the $O(1)$ operation count by an exponentially larger constant, but reduce the overall storage requirements to n^ϵ, for suitable $\epsilon < 1$. These variations can be applied to either of our nonconstructive

formulations, but we will focus on those built from peelable graphs, since they appear to be more efficient.

The first step is to show that such compact representations exist. The second is to observe that simple changes in the random constructions yield such graphs with a high probability, so that the graph itself can be part of the random seed specification. The result is a uniform r-practical family where a good graph need not be found. Instead, a randomized graph formulation can be used, which will comprise a pool of $O(d\kappa)$ random seeds, a finite initialization program to decompress the seeds into, for some $\beta < 1$, n^β pseudorandom seeds that (are stored in an auxiliary graph array G and) represent a compact formulation of the graph, which can be processed by a finite program to find, in constant time, the neighbors of a given input vertex, and the pseudorandom value associated with each neighbor (and stored in the auxiliary array M.)

Definition 6.

Let the *Cartesian product* $G \bigotimes H$ of two bipartite graphs $G = (I, O; E)$ and $H = (J, Q; F)$ be the graph $\mathcal{G} = (\mathcal{I}, \mathcal{O}; \mathcal{E})$, with input vertex set $\mathcal{I} = I \times J$, output set $\mathcal{O} = O \times Q$, and edge set $\mathcal{E}$, which contains the edge from $(i, j) \in \mathcal{I}$ to $(o, q) \in \mathcal{O}$ if and only if $edge(i, o) \in E$ and $edge(j, q) \in F$.

Lemma 5. Let $G = (I, O; E)$ be (m, ϵ, d, κ) locally peelable, and $H = (J, Q; F)$ be (n, ϵ, c, κ) locally peelable. Then the Cartesian product $G \otimes H$ is $(mn, \epsilon, cd, \kappa)$ locally peelable.

Proof: We need only verify the local peelability property for $G \bigotimes H$. The proof is by contradiction. Let X be a smallest set of input variables for $G \otimes H$ that is not peelable. Obviously $|X| > 1$. Suppose $|X| \leq \kappa$. Let $I_G = \{i \,|\, \exists j : (i, j) \in X\}$. Now $|I_G| \leq \kappa$, and hence there is an $\hat{\imath} \in I_G$ with a neighbor o_1 that is not a neighbor of any node in $I_G - \{\hat{\imath}\}$. Now let $J_G = \{j | \exists j : (\hat{\imath}, j) \in X\}$, and let $\ell = |J_G|$. By definition, J_G is peelable. So let $q_1, q_2, \ldots, q_\ell$ be a sequence of outputs peeled one-by-one in order from J_G. It is easy to see that $(o_1, q_1), (o_1, q_2), \ldots, (o_1, q_\ell)$ now gives a comparable peeling for

$\{o_1 \times J_G\}$ in X. But then the remaining $|X| - \ell$ variables (if any) are by assumption peelable, whence the supposition that there is a nonpeelable subgraph with at most κ input vertices must be false. ∎

Consequently, if G is an $(m^\epsilon, \epsilon_1, d, \kappa)$ local expander, then applying Lemma 5 a total of $c = \lceil \frac{1}{\epsilon_1} \rceil - 1$ times shows that the Cartesian product $\times_{j=1}^{\lceil 1/\epsilon \rceil} G$ is $(m^{c\epsilon}, \epsilon_1, d^c, \kappa)$ locally peelable.

Lemma 5 plus the constructions of Theorem 2 establish the following.

Theorem 3. Let the positive integers m, m^ϵ, κ, d and ρ be specified with $m > \rho$, and suppose that $\frac{1}{2} \le \frac{\epsilon}{2} \ge \frac{2}{d} + \frac{1+\ln d + \ln \kappa}{\epsilon \ln m}$, and $m^{\epsilon^2/2} > \frac{1}{\epsilon}$. Let $c = \lceil \frac{1}{\epsilon} \rceil$ and set $\epsilon_1 = \frac{\log \lceil m^{\epsilon/c} \rceil}{\epsilon \log m}$. Let $\mathcal{G}$ be the set of all graphs $G = (V, E)$ that are bipartite with m^ϵ input vertices, $m^{\epsilon_1 \epsilon}$ output vertices and where each input vertex has edges to d distinct neighbors among the outputs. Then there is a $G \in \mathcal{G}$ and a fixed programs $f_w^{\widehat{G}}$, for $w \in [0, \rho - 1]^{m^{\epsilon_1 \epsilon}}$ of size $O(\epsilon^2 d) m^\epsilon \log m$ bits that, for each input from $[0, m-1]$, use $\widehat{G} = \bigotimes_{i=1}^{c} G$ to retrieve no more than $d^{1+1/\epsilon}$ words from an auxiliary storage array of $O(m^\epsilon)$ random words in $[0, \rho - 1]$ "$\oplus$" the words together to compute a $(\kappa, 1)$-wise independent hash function that maps $[0, m-1]$ into $[0, \rho - 1]$.

Proof: Lemma 4 will be used to establish the existence of $(m^\epsilon, \epsilon_1, d, \kappa)$ local expanders. Programmatically, G will be stored and used as part of the hash function. A value $i \in [0, m-1]$ is hashed by computing the adjacency list for i in the implicitly defined $\widehat{G} \equiv \bigotimes_{j=1}^{c} G$. Given the peelability of G, Lemma 5 shows that $\widehat{G}$ is $(m^{c\epsilon}, \epsilon_1, d^c, \kappa)$ locally peelable, which is adequate for use in Theorem 2, since $c\epsilon > 1$.

It is not difficult to verify that all parameter sizes are as stated. In particular, there are three issues: the peelability of the G, the size of the auxiliary array and the outdegree of each vertex in $\widehat{G}$.

The parameter ϵ_1 satisfies $\frac{1}{c} \le \epsilon_1$. By construction, $c < \frac{1}{\epsilon} + 1$, whence $\frac{1}{c} > \frac{\epsilon}{1+\epsilon} > \frac{\epsilon}{2}$. Hence $\frac{\epsilon}{2} < \frac{\epsilon}{1+\epsilon} < \epsilon_1$ so ϵ_1 satisfies the peelability inequality of Lemma 4 since $\frac{\epsilon}{2}$ does.

By definition,

$$\epsilon_1 = \frac{\ln\lceil m^{\epsilon/c}\rceil}{\epsilon \ln m} < \frac{\ln(m^{\epsilon/c}+1)}{\epsilon \ln m} = \frac{1}{c} + \frac{\ln(1+m^{-\epsilon/c})}{\epsilon \ln m} < \frac{1}{c} + \frac{m^{-\epsilon/c}}{\epsilon \ln m} \le \frac{1}{c} + \frac{m^{-\epsilon^2/2}}{\epsilon \ln m},$$

whence, since $m^{\epsilon^2/2} \ge \frac{1}{\epsilon}$:

$$\epsilon_1 < \frac{1}{c} + \frac{1}{\ln m}.$$

So

$$\epsilon_1 - \frac{1}{c} < \frac{1}{\ln m}.$$

The word count for the auxiliary array is

$$m^{\epsilon\epsilon_1 c} = m^{\epsilon(\epsilon_1 - 1/c + 1/c)c} \le m^{\epsilon} m^{\frac{\epsilon c}{\ln m}} = m^{\epsilon}\left(m^{\frac{1}{\ln m}}\right)^{c\epsilon} = m^{\epsilon} e^{\epsilon c} < m^{\epsilon} e^2.$$

The outdegree is $d^c < d^{\frac{1}{\epsilon}+1}$.

Thus, the hash function has three parts: one part comprises the array M of $m^{c\epsilon\epsilon_1} = O(m^{\epsilon})$ random words from $[0, p-1]$, which requires $O(m^{\epsilon}\log p)$ bits. Another stores the graph G, which requires m^{ϵ} strings of d words that are each $O(\log(m^{\epsilon^2}))$ bits long. The requisite storage for G (i.e., part 2) is, therefore, $O(\epsilon^2 d m^{\epsilon}\log m)$ bits. The third part of the function is the finite program that uses G and M to evaluate the hash function for an element in $[0, m-1]$. ∎

Remarks. Theorem 3 can be used with m replaced by n^k, ϵ replaced by ϵ/k, κ replaced by n^{δ}, and d unchanged. The inequality for ϵ then becomes

$$\frac{\epsilon}{2k} \ge \frac{2}{d} + \frac{1 + \ln d + \delta \ln n}{\epsilon \ln n}, \tag{8}$$

which we can write as $\epsilon \ge \frac{4k}{d} + 2k\frac{1+\ln d+\delta\ln n}{\epsilon \ln n}$. Upon applying the simplification $\epsilon > a + \sqrt{b}$ implies $\epsilon \ge a + b/\epsilon$, we see that the requirement for ϵ is met if $\epsilon \ge \frac{4k}{d} + \sqrt{2k(\delta + \frac{1+\ln d}{\ln n})}$. The corresponding hash functions can be evaluated with $O(d^{k/\epsilon})$ operations, and have a program size of $O(n^{\epsilon})$ words. It is worth pointing out that (8) can permit ϵ to be any sufficiently small positive constant provided, say, $d > 12k/\epsilon$, $\delta < \epsilon/6$, $n > d^{6/\epsilon}$, and $n^{k\epsilon^2/2} > \frac{1}{\epsilon}$. The resulting functions are $(n^{\epsilon/6}, 1)$-wise independent.

For completeness, we state without proof the analogous compaction/expansion formulation for local concentrators.

Lemma 6. Let G be an $(m^\epsilon, \epsilon, d, \kappa)$ local concentrator. Then the Cartesian product $\bigotimes_{j=1}^{1/\epsilon} G$ is an $(m, \epsilon, d^{1/\epsilon}, \kappa)$ local concentrator. ∎

So far, families of hash functions mapping D into R have been "demonstrated" only in a probabilistic sense; no explicit constructions have been given. However, by increasing, slightly, the degrees of freedom in our probabilistic constructions, the same counting arguments ensure that with probability $1 - \frac{1}{|R|^r}$, a randomly selected graph is a local concentrator or expander. Consequently, we can include, in the initial seeding, enough random data to build a local expander as well as the array of random words it will be used to access. The graph will not be prohibitively large if it defines a compact version as formulated in Theorem 3, and adapted as in (8). The only necessary accommodation is to keep the probability of failure appropriately tiny in the rescaled parameters, which can be done by increasing r by a factor of $\frac{1}{\epsilon}$. The resulting randomized construction $F_M^{G(m,\epsilon,d)}$ is an explicit family of $O(1)$ time hash functions that is uniformly r-practical (κ)-wise independent as characterized by Definition 2. Lastly, it is worth pointing out that the graph structure suggests that the number of initializing random seeds need not change by more than a constant factor; it is sufficient to build G from pseudo-random numbers generated from a traditional $(\kappa d^{k/\epsilon})$-wise independent hash function. To see that this is so, it suffices to observe that the proof of Lemma 4 was based on expectations of disjunctions of atomic events comprising the conjunction of $d\kappa$ or fewer random edge assignments.

These observations are formalized in the following theorem, where the bipartite graph G is just a collection of $(\kappa d^{k/\epsilon})$-wise independent pseudorandom numbers. A simple algorithm to process this graph and compute the actual hash function it encodes is presented after Theorem 4. The construction is randomized and asymptotically compact.

Theorem 4. Let the positive integers n, k, n^ϵ, κ, d, ρ and r and be specified, and suppose that $\frac{\epsilon}{k+1} \geq \frac{2}{d} + \frac{r}{d\epsilon} + \frac{1+\ln d+\ln \kappa}{\epsilon \ln n}$. Let $c = \lceil \frac{k}{\epsilon} \rceil$ and set $\epsilon_1 = \frac{\log \lceil n^{\epsilon/c} \rceil}{\epsilon \log n}$. Suppose

that $\epsilon < \frac{k}{k+1}$ and $n^{\epsilon^2/(k+1)} > \frac{1}{\epsilon}$. Let $\mathcal{G}(d, \kappa d^{kc})$ be a set of pseudorandomly generated bipartite graphs with n^{ϵ} input vertices, $n^{\epsilon\epsilon_1}$ output vertices and where each vertex has edges to d distinct randomly selected neighbors, and these d-tuple assignments of neighbors are (κd^{kc-1})-wise independent. Let M be a collection of long words that are the concatenation of (κd^{kc})-wise independent sequences of $n^{\epsilon\epsilon_1 c}$ words that individually belong to $[0, \rho - 1]$. For any $G \in \mathcal{G}(d, \kappa d^{kc})$, let $\widehat{G} = \bigotimes_{i=1}^{ck} G$. Given $w \in M$, and $G \in \mathcal{G}$, let f_w^G be the hash function defined by $f_w^{\widehat{G}}$ as in Lemma 2. Let $F_M^{\mathcal{G}} = \{f_w^G\}_{w \in M,\, G \in \mathcal{G}}$.

Then $F_M^{\mathcal{G}}$ is an explicit family of uniformly r-practical $(\kappa, 1)$-wise independent hash functions that map $[0, n^k - 1]$ into $[0, \rho - 1]$. Furthermore, each $w \in M$ comprises $|w| = n^{\epsilon\epsilon_1 c} \leq e^{k+1} n^{\epsilon}$ from $[0, \rho - 1]$. Each w string can be generated from κd^{kc} random seeds that belong to $[0, \rho - 1]$. Each $G \in \mathcal{G}$ can be stored as a sequence of n^{ϵ} words of $O(\frac{\epsilon^2}{k} \log n)$ bits, and can be generated from (κd^{kc}) random seeds that are evenly distributed among the ranges $[0, n^{\epsilon\epsilon_1} - 1], [0, n^{\epsilon\epsilon_1} - 2], \ldots, [0, n^{\epsilon\epsilon_1} - d]$. The probability that specific $F_*^{\widehat{G}}$ fails to be $(\kappa, 1)$-wise independent is bounded by n^{-r}.

Proof: The arguments are just a combination of the reasoning given in Lemma 4, the rescaling listed in (8), and, mutatis-mutandis, the calculations given in the proof of Theorem 3.

However, a few comments should be made about how to generate the two tables of pseudorandom data. The potential difficulty is that we need, at initialization time, a small (traditional) hash function to expand some count σ of truly random seeds into a much larger collection of n^{δ} pseudorandom seeds belonging to $[0, \rho - 1]$, even when $[0, \rho - 1]$ cannot define a finite field.

Perhaps the mathematically cleanest solution is to factor ρ into its product of powers of distinct primes, and use a (κd^{ck})-wise independent hash function over each of these fields. Then the Chinese Remainder Theorem can be used to reconstruct the comparably random $(\kappa d^{kc}, 1)$-wise independent pseudorandom seeds as needed.

Lastly, we need a (pseudo)randomized construction of, say, a local expander graph,

which comprises $|I|$ (κd^{kc-1})-wise independent selections of d distinct random destinations from the output set $|O| = [0, \rho - 1]$. One solution to the distinctness problem is to use an adaptive process to remap the range of the hash function to exclude, when the j-th destination, $0 \leq j \leq d-1$, is to be computed for input vertex i, the previously computed $j-1$ destinations. Let $h_0, h_1, \ldots, h_{d-1}$ be (κd^{kc-1})-wise independent, and let h_j map I into $[0, |O| - j - 1]$. The j-th output destination for vertex i, for $j = 0, 1, \ldots, d-1$, $0 \leq j \leq |I| - 1$, is assigned $Adj(i, j) = s$, where $s = h_j(i) + r(i, j)$, and $r(i, j)$ has the property that $r(i, j)$ equals the number of vertices among $Adj(i, 0), Adj(i, 1), \ldots, Adj(i, j-1)$ that are less than s, and none of the vertices in the list equals s. ∎

The first requirement for ϵ is satisfied if $\epsilon \geq \frac{2(k+1)}{d} + \sqrt{(k+1)(\frac{r}{d} + \frac{1+\ln d + \ln \kappa}{\ln n})}$.

For applications of Theorem 4, a new hash function family (which might have to be found in case $F_*^{\widehat{G}}$ fails to be $(\kappa, 1)$-wise independent) would include new seeds to select a new pseudorandom graph $G \in \mathcal{G}$ as well as new pseudorandom seeds to select a new $w \in M$.

For completeness, this subsection closes with a rather crudely transparent iterative version of the hashing algorithm, which is presented below, where the use of $\frac{1}{\epsilon}$ is for conceptual transparency and lack of clutter.

function Hash(i: in $[0, n^k - 1]$): in $[0, n^k - 1]$;

Global M: array of n^ϵ words in $[0, \rho - 1]$;

Global G: $n^\epsilon \times d$ array of words in $[0, n^{\epsilon^2/k} - 1]$;

Local $l_1, l_2, \ldots, l_{k/\epsilon}$: in $[0, d - 1]$;

Local $i_1, i_2, \ldots, i_{k/\epsilon}$: in $[0, n^\epsilon - 1]$;

Local j: in $[0, n^\epsilon - 1]$;

Local val: in $[0, n^k - 1]$;

$val \leftarrow 0$;

$(i_1, i_2, \ldots, i_{k/\epsilon}) \leftarrow i$; {Distribute i as $\frac{k}{\epsilon}$ different packets of $\epsilon \log n$ bits.}

for $l_1 \leftarrow 0$ **to** $d - 1$ **do**

for $l_2 \leftarrow 0$ **to** $d-1$ **do**

$\vdots$

for $l_{k/\epsilon} \leftarrow 0$ **to** $d-1$ **do**

$j \leftarrow (G[i_1, l_1], G[i_2, l_2], \ldots, G[i_{k/\epsilon}, l_{k/\epsilon}])$;

$val \leftarrow (M[j] \oplus val)$

endallfors; {Altogether, $d^{k/\epsilon}$ $\oplus$-s take place.}

return(val).

3. Lower bounds

We now show that the size of our random word array cannot be materially reduced without affecting the running time of the hash function. A family of (κ, μ)-wise independent hash functions $F_M = \{f_m(x)\}_{m \in M}$ where $f_m : D \rightarrow R$ will be modeled as follows. Each f_m is defined by the same algorithm, which inputs x and then reads d locations in an array $A[1..z]$ that contains z values belonging to R. The index m is a very long word comprising the concatenated data contained in A. We can suppose that each computation of f examines exactly d entries in the array A. The randomness in the system comes from the choice of input seed strings $m \in M \subseteq R^z$. The set M need not contain all possible sequences of z words from R, and can be a multiset, which means that some strings could have several copies present in M. More generally, it is worth remarking that the bounds we present would apply equally well if the strings were assigned real weights, and each $m \in M$ were then selected with a probability that equals m's fraction of the total weight of elements in M, but we will not address this issue any further. The algorithm can even be viewed as adaptive since we allow a key x to be hashed by a scheme that, for $j = 0, 1, \ldots, d-1$, uses x and the first j values found in A to determine which A location to read for the $(j+1)$-st value. These values and x are then used deterministically to compute the random function value in R.

The forthcoming lower bound argument will exploit the following independence constraint. Let $X = \{x_1, x_2, \ldots, x_\kappa\}$ be a set of κ keys in D, and let, for $j = 1, 2, \ldots, \kappa$, the

sequence of seed values read to compute $f_M(x_j)$ be $s_{j,1}, s_{j,2}, \ldots, s_{j,d}$. Then the hashing of X, over all possible seed strings in M, cannot cause the same fixed sequence of κd seed values to be read with a probability that exceeds $\frac{\mu}{|R|^\kappa}$ as otherwise X will be mapped into the same κ-tuple with a probability that is too high. Since the determination of the sequence of seed locations can be adaptive, we are obliged to formulate the proof with respect to the triples $(\zeta, M_i^\zeta, D(\zeta, i))$, where ζ is a subset of $\kappa - 1$ locations in the seed array A, i is a string that can result from concatenating the seed values in ζ, M_i^ζ is the subset of seed data sets $m \in M$ for which the concatenation of the seed values on ζ is i, and $D(\zeta, i)$ is the subset of keys in D that are hashed by seeds read solely from ζ, when the seed data is restricted to M_i^ζ. Given these triples, the proof proceeds by summing their respective weights, and bounding the respective sums. Bounds are attained by selecting the "good" triples, where $|M_i^\zeta|$ is large enough to ensure that $|D(\zeta, i)| < \kappa$. The lower bound quantifies the intuition that if most $|D(\zeta, i)|$ are small, then A must be big.

Theorem 5. Let $F_M = \{f_m\}_{m \in M}$ denote a family of (κ, μ)-wise independent hash functions mapping D into R, where $M \subseteq R^z$. Then the number of probes T that must be used to evaluate $f \in F_M$ satisfies either $T \geq \kappa$ or

$$z^{\frac{T}{T-1}} > (\kappa - 2)|D|(1 - \frac{\mu}{|R|}).$$

Proof: Let every evaluation of f examine exactly d entries in the array A. We show that d satisfies the constraint for T. We can also suppose that $\mu < |R|$ as otherwise there is nothing to prove.

For each set ζ of $\kappa - 1$ locations in the z-element array A, let the locations in ζ be sorted by the value of their indices in A, so that each ζ imparts a fixed sequencing to the values stored in its locations. Then any string $m \in M$ will have, on its restriction to ζ, $\kappa - 1$ values in a fixed order. We can view the concatenation of this subsequence as a number in $[0, |R|^{\kappa-1} - 1]$, so that any such ζ defines an implicit projection of any $m \in M$ into $[0, |R|^{\kappa-1} - 1]$. Given ζ, let M be partitioned into $M^\zeta = \left\langle M_1^\zeta, M_2^\zeta, \ldots, M_{|R|^{\kappa-1}}^\zeta \right\rangle$,

where M_i^ζ is the set of strings in M that have projections equal, on ζ, to the value $i \in [0, |R|^{\kappa-1}-1]$. Let $D(\zeta, i)$ be the set of domain elements $x \in D$ that, when computing $f_m(x)$ for $m \in M_i^\zeta$, have their d A-locations read from within ζ. Let

$$D_0(\zeta, i) = \begin{cases} D(\zeta, i), & \text{provided } |M_i^\zeta| > \mu|M|/|R|^\kappa; \\ \emptyset, & \text{if } |M_i^\zeta| \le \mu|M|/|R|^\kappa. \end{cases} \tag{9}$$

Given an $x \in D(\zeta, i)$, $f_m(x)$ will be computed by probing the same d-tuple of locations within ζ for all $m \in M_i^\zeta$. It follows that $|D_0(\zeta, i)| < \kappa$ since otherwise there are κ elements in D that hash to some κ-tuple of values in R with a probability that exceeds $\mu/|R|^\kappa$.

There are $\binom{z}{\kappa-1}$ subsets ζ, and each subset induces a partition of M indexed by the m-values restricted to ζ. Let $\Sigma = \sum_\zeta \sum_i |M_i^\zeta|\,|D(\zeta, i)|$. Let $\Sigma_0 = \sum_\zeta \sum_i |M_i^\zeta|\,|D_0(\zeta, i)|$. Since $|D_0(\zeta, i)| < \kappa$, $\Sigma_0 \le \sum_\zeta \sum_i (\kappa-1)|M_i^\zeta| = \sum_\zeta (\kappa-1)|M|$, whence

$$\Sigma_0 \le (\kappa-1)\binom{z}{\kappa-1}|M|. \tag{10}$$

On the other hand, Σ has an alternative description, which gives a precise combinatorial formulation. Let $D^+(\zeta, i)$ be the set of pairs (m, x), with $x \in D$ and $m \in M$, such that, when computing $f_m(x)$ for $m \in M_i^\zeta$, all d probes to A-locations lie within ζ. By definition, $|D^+(\zeta, i)| = |M_i^\zeta|\,|D(\zeta, i)|$. Furthermore, since the probing for x uses exactly d locations, is clear that each pair (m, x) is counted in exactly $\binom{z-d}{\kappa-1-d}$ different $D^+(\zeta, i)$. Hence

$$\Sigma = \binom{z-d}{\kappa-1-d}|D|\,|M|. \tag{11}$$

Finally, the hashing of an $x \in D$ uses d probes that retrieve d specific values in a fixed order from d locations in A. There are at most $|R|^d$ different sequences that can be retrieved, and each sequence corresponds to d distinct locations (since the program need not reread a location when hashing an individual key). Consequently, for a given key x, each possible probe data tuple in R^d could correspond to a specific sequence of d probe locations in A, which can belong to at most $\binom{z-d}{\kappa-1-d}$ different ζ sets, which would have $\kappa-1-d$ unprobed locations that could have up to $|R|^{\kappa-1-d}$ different assignments.

Thus, any $x \in D$ belongs to at most $|R|^d\binom{z-d}{\kappa-1-d}|R|^{\kappa-1-d}$ different $D(\zeta, i)$. Hence

$$\Sigma_\zeta \Sigma_i |D(\zeta, i)| \leq |D| \binom{z-d}{\kappa-1-d} |R|^{\kappa-1}. \tag{12}$$

From (9) and the definitions for Σ and Σ_0, we can count that

$$\Sigma - \Sigma_0 \leq \Sigma_\zeta \Sigma_i \frac{\mu|M|}{|R|^\kappa} |D(\zeta, i)| = \frac{\mu|M|}{|R|^\kappa} \Sigma_\zeta \Sigma_i |D(\zeta, i)|. \tag{13}$$

Applying (12) to (13) shows that $\Sigma - \Sigma_0 \leq \frac{\mu|M|}{|R|^\kappa}|D|\binom{z-d}{\kappa-1-d}|R|^{\kappa-1}$, whence

$$\Sigma - \Sigma_0 \leq \binom{z-d}{\kappa-1-d} \mu|M| \frac{|D|}{|R|}. \tag{14}$$

Combining equation (11) and inequality (14) gives

$$\Sigma_0 \geq \binom{z-d}{\kappa-1-d} |M|\,|D|(1 - \frac{\mu}{|R|}). \tag{15}$$

We remark that (15) cannot hold as an equality. Indeed, suppose that it did. Then (14) and the first inequality in (13) would hold as equalities. But if (13) were an equality, then $D_0(\zeta, i)$ would have to be the empty set for all i and ζ, since the definitions of $D(\zeta, i)$ and $D_0(\zeta, i)$ ensure that $|M_i^\zeta|(|D(\zeta, i)| - |D_0(\zeta, i)|) \leq \frac{\mu|M|}{|R|^k}|D(\zeta, i)|$, and equality can occur only if $|D_0(\zeta, i)| = 0$. So if (13) is an equality then then Σ_0 must be zero, which contradicts (15), since $\mu < |R|$, $d < \kappa$, and $z \geq \kappa$. Hence

$$\Sigma_0 > \binom{z-d}{\kappa-1-d} |M|\,|D|(1 - \frac{\mu}{|R|}). \tag{16}$$

Combining inequalities (10) and (16) gives

$$(\kappa - 1)\binom{z}{\kappa-1} |M| > \binom{z-d}{\kappa-1-d} |M|\,|D|(1 - \frac{\mu}{|R|}).$$

Eliminating common factors establishes that

$$\frac{z^{\underline{d}}}{(\kappa-2)^{\underline{d-1}}} > |D|(1 - \frac{\mu}{|R|}). \quad \blacksquare \tag{17}$$

Now, the derivation of (17) is only valid if $d < \kappa$, but it is nevertheless reassuring to observe that if we (unjustifiably) set d to κ in the inequality, $z^{\underline{d}} > (\kappa-2)^{\underline{d-1}}|D|(1-\frac{\mu}{|R|})$,

then the resulting expression collapses to the requirement $z^{\underline{\kappa}} > 0$, which is to say that $z \geq \kappa$. Of course κ random numbers are necessary and sufficient, in this trivial case.

There are more significant conclusions to be drawn from Theorem 5 and its relationship to Theorem 2. These observations are deferred to Section 4. Meanwhile, we close this section with two immediate corollaries.

The counting argument for Theorem 5 also gives an average case time bound.

Corollary 1. Let z, D and R be fixed. Let $F_M = \{f_m\}_{m \in M}$ denote a family of (κ, μ)-wise independent hash functions mapping D into R, where $M \subseteq R^z$ is the collection of z-word auxiliary array data used to evaluate functions in F_M. Suppose that $\mu \leq |R|/2$. Let T be a lower bound for the worst-case count of the number of probes to the array data that must be used to evaluate some function in any such F_M. Let $\overline{T}$ be the average probe count for any such family, which is averaged over all of D and all of F_M.

Then $\overline{T} \geq T - \frac{2}{|D|}(\frac{z^{\underline{T-1}}}{(\kappa-2)^{\underline{T-2}}} + \frac{z^{\underline{T-2}}}{(\kappa-2)^{\underline{T-3}}} + \cdots + z)$, which can be expressed as $\overline{T} \geq T - o(1)$ when $z > c\kappa$ for fixed $c > 1$.

Proof: The proof is generic. Let Δ_h be the size of the largest domain that can be serviced by a $(\kappa, 1)$ wise independent hash function that maps Δ_h into R and uses h probes or less. Let T be a lower bound for the number of probes that are necessary, in the worst case, for a $(\kappa, 1)$-wise independent function that services D. Then a lower bound for the total number of probes necessary to service all of D is $|D|T - |\Delta_{T-1}| - |\Delta_{T-2}| - \cdots - |\Delta_1|$, since the negative terms comprise overcorrections for the numbers of probes that are miscounted by the principal term $|D|T$. The result now follows from Theorem 5, which says that $\frac{1}{2}\Delta_d < \frac{z^{\underline{d}}}{(\kappa-2)^{\underline{d-1}}}$, and division by $|D|$. ∎

Of course Theorem 5 also gives an admissible estimate to use for T.

Theorem 5 suggests that T has a very mild dependence on the parameter μ, and our constructions have typically allowed μ to remain at one. On the other hand, it is worth noticing that the lower bound exhibits a far greater dependence on $|D|$, which could be dramatically larger than $|R|$. In this case, we can use a simple, randomly chosen linear

congruence (cf. F_0^k in Appendix I) to premap $|D|$ into, say, R^4, whence the resource requirements dictated by Theorem 5 would depend on $|R|^4$ rather than D. Thus there is a significant benefit, in some cases, in allowing asymptotically negligible fractions of hash functions to be defective (due to distinct data values colliding at the premapping stage), and our constructions reflect this benefit.

Finally, it is worth observing that Theorem 5 can be used quite simply to give a moderately satisfactory lower bound for the running time of r-practical hash functions.

Corollary 2. Let F be a family of hash functions F with domain D and range R that is r-practical (κ, μ)-wise independent. Then

$$z^T > (\kappa - 2)^{\frac{T-1}{2}} \frac{|R|^{r/2}}{2} (1 - \frac{\mu}{|R|}).$$

We omit the proof because it is a straightforward application of Theorem 5.

While the exponent $r/2$ in this bound is surely too small by a factor of 2, its influence on the probe count T is still fairly acceptable. If the data can be examined during the function selection stage, then probabilistic constructions as presented in Appendix 1 allow, with high probability, the prehashing step to succeed without any collisions, so that the resulting function, once a successful linear congruence is found, is fully $(\kappa, 1)$-wise independent.

4. A space–time tradeoff and problem equivalence

It is interesting to observe the dramatic change in the requirements for z when d drops from κ to $\kappa - 1$. Suppose that $\mu < |R|/2$, and $\kappa > 3$. We can weaken (17) to state $z^d > D$, so that z must be at least $|D|^\epsilon$ for some positive ϵ. Setting $|D| = n^k$ gives $d > \frac{k}{\epsilon}$. In view of Theorem 2 (with m set to n^k), we conclude that the number of probes to the auxiliary array A of $z = n^\epsilon$ random $\ell \log n$-bit words that is necessary per evaluation of a (κ)-wise independent hash function that maps $[0, n^k - 1]$ into $[0, n^\ell - 1]$ satisfies

$$d = \Theta(k/\epsilon), \quad \text{for } d < \kappa.$$

Restated, we have a time-log(space) tradeoff: $T\log(CacheSize) \geq \log(DomainSize)$, where $CacheSize$ is the dimension z of the random word array A, which contains words from R, and $DomainSize = |D|$. This lower bound and tradeoff applies to any algorithm including those with random precomputation (that is oblivious to the data) as long as any internal storage and precomputed values are counted as part of the array as measured by z. Furthermore, the program need not be uniform, and the storage resulting from any deterministic precomputation that precedes the reading of the random seeds need not be charged to the storage measured by z. (The point of these remarks is that the tradeoff need not include the generation of G, which could be probabilistic or based on exhaustive search, which would entail an extravagant use of time and space, but result in an optimally structured graph.)

From a more abstract perspective, we have exposed a very close equivalence between the true space-time computational complexity for evaluating (κ)-wise independent hash functions that map, say $[0, n-1]$ into $[0, n-1]$, and the operation count needed to compute the neighbors of an input vertex of bipartite graphs on $[0, n-1] \times [0, n^\epsilon - 1]$ that have low outdegree d and have good expansion properties for small vertex sets. A spatially compact graph representation that can be used to compute the adjacency list of an input vertex in time $T_G = cd$ gives a time $T_f \approx T_G$ hash function with a high degree of independence, when augmented with an array of n^ϵ random numbers. Similarly, a set of d random functions that are independent and enjoy $(\epsilon\kappa)$-wise independence can be used to build such a graph with $T_G \approx \epsilon d T_f$, albeit with an additive spatial cost of $\epsilon d n^\epsilon$ for the random number arrays: The equivalence holds in this direction because our probability estimates in Section 2 were calculated from κ-way expectations, and never used full independence. The resource blowup is the modest factor ϵd because a random function value in $[0, n-1]$ gives $\frac{1}{\epsilon}$ points in $[0, n^\epsilon - 1]$ (where all expressions, for expositional simplicity, are assumed to be integers). A crude application of our lower bound imposes the requirement that $d > 1/\epsilon$, while our existential Theorem 2

establishes sufficiency for $d = 2/\epsilon + 1$.

Taken together, these upper and lower bounds show that up to a fairly fine granularity of instruction block counting, the computational complexity of these two algorithms problems are within a factor of 2 of each other.

4.1 Independence in retrospect

The derivations and proofs, which were little more than a study of locally 1 to 1 mappings, exhibit a kind of probabilistic transparancy; if the initial seeding for the hash function are $(\Theta(\kappa), \mu)$-wise independent, then the resulting hash functions are likewise $(\Theta(\kappa), \mu)$-wise independent, albeit for smaller multiple of κ. The more demanding application for the seeds, of course, is as input for the actual hash function; the seeding (if any) for the (presumably) random graph does not seem to be especially sensitive to the parameter μ, and, in theory, does not require any randomness whatsoever. The graph existance arguments, of course, would need minor adjustment. When $\mu = 1$, the existance of suitable functions (or structures) will follow when probability estimates for failure are less than 1. With $\mu > 1$, we will need to read just the parameters to get comparable combinatorial expressions for failure that are less than $\frac{1}{\mu}$.

Evidently, the (κ, μ)-wise independence of the resulting functions depends on the quality of the input seeds. Some proofs (cf. [19]) seem to depend on $(\kappa, 1)$-wise independence. It is worth observing that in many such circumstances, seedings of ℓ words that meet the two-sided formulation of $(\ell, 1 + O(\frac{1}{n^2}))$-wise independence may well suffice. After all, if all seedings are, within a factor of $1 + O(\frac{1}{n^2})$, equally likely to occur, then the family of seeds can viewed as $(\ell, 1)$-wise independent with probability $1 - O(\frac{1}{n^2})$, and as faulty with probability $O(\frac{1}{n^2})$. This simple observation would seem to simplify some performance analyses, and might well be the the most compelling reason to seek formulations that depend on small initial seed counts.

5. Formal applications

The constructions in Section 2 show that (κ)-wise independent hash functions, for nonconstant $\kappa < n^\delta$ and sufficiently small constant $\delta > 0$, can actually be programmed as constant time subroutines that require only a moderate size array of random numbers as input. Thus we have established a formal feasibility of any probabilistic algorithm that has a performance bound based exclusively upon the use of such functions.

The two examples cited in this section are by no means self-contained. The first, which concerns the performance of double hashing, follows from an elaborate proof based on $O(\log n)$-wise independence [19]. Consequently, Corollary 3 follows trivially.

The second application, which concerns the pipelined emulation of an idealized $n \log n$ processor parallel machine on an n processor real machine, requires modifications of the original construction [18], which is based on universal classes of $O(\log n)$-wise independent hash functions. The original algorithm, though elegant, is sufficiently elaborate that we only present the changes. In both applications, the original references are necessary for a complete understanding of the results. For additional applications, see [8, 10, 14] and. http://citeseer.nj.nec.com/context/122560/0 as well.

Corollary 3. For fixed load factor $\alpha < 1$, $O(\log n)$-wise independent hash functions can be used for double hashing with constant time per probe and an expected cost of $\frac{1}{1-\alpha} + o(\frac{1}{n})$ probes for unsuccessful search. ∎

Randomized routing schemes and PRAM emulation have had a substantial and fruitful literature [29, 25, 2, 17, 26, 27, 9, 18, 10, 8]. In particular, Karlin and Upfal ([9]) and Ranade ([18]) show formally (and perhaps plausibly) how $n \log n$-processor Butterfly networks with bounded queues can, with very high probability, emulate an $n \log n$-processor PRAM with an optimal performance penalty that is "only" a multiplicative factor of $\log n$.

Both Karlin and Upfal [9] and Ranade [18] presented schemes for an $n \log n$-processor emulation of $n \log n$-processor PRAM algorithms. The processors are interconnected by

an $n \times \log n$ butterfly network. For this configuration, the opportunity to exploit pipelining is limited for models featuring 100% randomized memory references, because each PRAM emulation step causes the network to be effectively saturated for $O(\log n)$ time. Thus, their feasibility results, which were based on hash functions comprising $\log n$ degree polynomials, sustain essentially no performance penalty for evaluating such a polynomial for each memory reference. Given the $\log n$ performance cost for referencing, Karlin and Upfal did not need to address the much less significant issue of what to do about hashing collisions at the memory cell level; it simply cannot be a problem when $O(\log n)$ time is available to locate each item. Ranade [18] uses the hardware capability, which can support up to $O(\log n)$ reads per fetch, to solve the problem.

Now that highly independent hash functions can be evaluated in constant time, it is natural to reexamine these models to see if optimal speed-up can be achieved by pipelining these algorithms on machines that feature a reduced ratio of processor density to routing capacity. The idea of pipelining that exploits large scale parallel slackness to mask network latency can be traced to Smith [23], and has also been a subject of theoretical study [12, 16, 28].

We adapt the constructions in [9] and [18] to emulate an $n \log n$ PRAM machine on a machine having one column of n processors interconnected by an $n \times \log n$ butterfly network. A PRAM step of $n \log n$ parallel instructions is emulated by assigning $\log n$ concurrent PRAM instructions to each processor. The machine would also have n memory modules, say, one per processor.

Ranade's Common PRAM emulation scheme can still be used, but a few aspects must be modified to adapt to the higher performance requirements. A brief high-level outline of his scheme will serve to identify the key issues. The system uses $O(\log n)$-wise independent hash functions (where the constant factor is about 8) to map the virtual address space of $[0, \log n - 1] \times [0, n - 1] \times [0, m - 1]$ into itself.

Ranade's scheme is roughly as follows.

1 Each of the $n \log n$ processors constructs a hashed memory request.

2 Each row of $\log n$ processors cooperate to sort their $\log n$ requests. The sorting is done by a parallel systolic bubblesort.

3 The first processor in each row injects its row's $\log n$ requests into the routing network, smallest address first. The messages are routed to the correct row, whence they pass along the row from module to module and eventually access the true location of the referenced data.

4 Each return message backs up along its path to its home processor.

A minor issue, in his scheme, is that as many as $O(\log n)$ data items might hash to (i.e., collide at) a single location. Ranade resolves this problem by distributing the data for such a collision set fairly evenly across a row of $\log n$ memory modules in the butterfly network. The actual hash address might hold a pointer to the distributed list of items that hash to the computed location. Then a memory request, which includes the unhashed reference address as a disambiguating identifier, can access each module in sequential order to access the correct data location. As only $O(\log n)$ references will, with high probability, be issued to each row, the $\log n$ processors are adequate to execute up to $(\log n)^2$ evenly distributed memory accesses to execute the $O(\log n)$ references in $O(\log n)$ time.

The subtle part of his scheme, which is responsible for its remarkable performance, is the routing sequencing and its analysis. Inasmuch as we exploit this part of the algorithm without any modification whatsoever, we omit its description despite its essential characteristics and critical content.

In a pipelined version, there would only be one processor in each row of the network, and it might typically issue as many as $\log n$ memory references for a single emulation step of Ranade's algorithm. Since the hash addresses for each memory reference can be computed in constant time, a single processor can execute these $\log n$ emulation steps in $O(\log n)$ time.

In this adaptation of his scheme, the hash function h would be used to map a key address $x \in [0, nm-1]$ into $h(x) \in [0, \log n - 1] \times [0, n-1] \times [0, \frac{m}{\log n} - 1]$. As in the original scheme, data packets are still kept locally lexicographically sorted (with the value x used to break and disambiguate ties). The first address field, which corresponds, in Ranade's scheme, to the column number of the destination module (where a pointer to the data could be found), is still used to control the timing and sequencing of packets in the routing switch buffers, but does not specify a column in the network, since all storage is in column 0. Instead, these bits, along with the third field, determine the local hash address for the destination memory module (which has m words of storage). The local process number (in $[1, \log n]$) for each packet would be explicitly listed in a separate field.

There are three modifications to his scheme that require explanation and proof.

Each of the n processors should sort their (locally generated) $\log n$ remote address keys in $\log n$ time. Ranade's scheme can afford to use $\log n$ processors and $\log^2 n$ work to sort $\log n$ addresses.

Each of the n memory modules, which have a logical address space of size m, should receive, with overwhelming probability, only $O(m)$ preimage addresses as computed by the hash function. This constraint is essentially the same as that in Ranade's scheme, and is included for completeness.

Finally, each module should be able to process its incoming memory access requests in $O(\log n)$ time with very high probability. Thus, each processor should receive (with high probability) only $O(\log n)$ memory references per PRAM step, and $O(\log n)$ time should be adequate for these references to be disambiguated from the other preimage data that hash to this same address set.

The local sorting is accomplished in three steps. Lemmas 7 and 8 will show that for any fixed c, there is a fixed d adequate to guarantee that each (i.e. the slowest) of the n processors can sort its $\log n$ messages in $d \log n$ time with a probability that is at

least $1 - \frac{1}{n^c}$. The worst case time will be $\log n \log\log n$. Since the use of ghost messages in Ranade's scheme ensures that the routing phase of the algorithm will synchronize with the processor that completes its sorting step last, the fact that some steps might take $\log n \log\log n$ time to sort is irrelevant to the overall asymptotic performance of this scheme. Thus, we can even take $c = 1$, and conclude that the expected time for the sorting phase is $\Theta(\log n)$.

The sorting begins with a hashing step to collect common memory references. Its purpose is to accommodate the Common PRAM model, which permits, in a single parallel step, a memory location to be referenced by any subset of the individual processors. Without this hashing step, the sorting phase (in step 3 below) might have too high a probability of performing poorly, even if a splay tree is used to instantiate the sorting.

Step 1) Bucket partitioning (hashing) is used to group together the memory references for identical locations. The common references, as generated within each pipeline, are then combined. This step comprises n parallel threads wherein each processor processes its local set of $\log n$ or fewer references.

The bucket partitioning is straightforward. Suppose we have a multiset $\mathcal{M}$ of $\log n$ items comprising $k \leq \log n$ different values in $[0, m-1]$. The partitioning proceeds by sequencing through $\mathcal{M}$ and, for each $h(x) \in \mathcal{M}$, inserting $h(x)$ into bucket $B[\lfloor \frac{h(x)}{m} n \rfloor]$. A list recording the buckets that are actually referenced is maintained during the insertion process. Then the referenced buckets are traversed and a fast sorting algorithm is applied to sort the contents of each occupied bucket into lists of identical values. The k lists are then sorted as a set of k distinct random values by steps two and three.

Lemma 7. For any fixed $c > 0$, there is a fixed d such that step 1 completes in $d \log n$ time with a probability that exceeds $1 - \frac{1}{n^c}$.

Proof: The probability that, for a single processor, the partitioning places at least $r > 0$ different key values in one bucket is bounded by the sum of the expected number of distinct r-tuples contained in each bucket. This latter count is at most $\mu \frac{\binom{\log n}{r}}{n^{r-1}}$, if the

multiset is derived from a family of (κ, μ)-wise independent hash functions, for $r \leq \kappa$. Since there are n processors, the probability that no processor encounters as many as r different key values in a single bucket is at least $1 - \mu\frac{\binom{\log n}{r}}{n^{r-2}}$.

When no bucket has as many as r different values, for some $r > 2$, the total time to sort each nonempty bucket is bounded by the worst case bound $O(\frac{\log n}{r} r \log r) = O(\log n \log r)$, which corresponds to the sorting of multisets comprising n items with at most r values. A suitable sorting scheme might use, for example, a balanced search tree of lists, with one list per value.

It follows that step 1 completes in time $O(\log n \log r)$ with a probability that exceeds $1 - \mu\frac{\binom{\log n}{r}}{n^{r-2}}$. ∎

If r or more values are encountered, we can use $\log n \log\log n$ as the time bound. The expected contribution, from this case, is a modest $O(\frac{(\log n)^{r+1} \log\log n}{n^{r-2}})$.

It suffices to set, say, $r = 3$ or thereabouts, and observe that the probability that some processor has three or more keys are mapped to the some single bucket is bounded by $\mu\frac{(\log n)^3}{6n}$.

For completeness, we note that while it is reasonable to expect that real parallel machines will have processors with free storage that in fact exceeds n words per machine, step 1 can achieve an expected time bound of $O(\log n)$ with fewer resources. First, n^ϵ storage, for any fixed positive ϵ will suffice. In this case, a suitable time bound can be established by bounding the expected number of r-tuples per bucket where $r \approx \frac{3}{\epsilon}$, or by using $\frac{1}{\epsilon}$ passes of the partitioning step in a procedure that is analogous to radix sorting. Actual sorting would be performed only for sequences of keys that have identical partitioning histories over all $\frac{1}{\epsilon}$ passes. Second, the storage need not be empty. A free store of $O(\log n)$ words is sufficient to implement a partitioning scheme that works by conditionally swapping its data into a possibly full local memory of size n^ϵ or more. Swapped-out data would be placed in the free store, and fully restored† upon completion

†The basic organization for such a scheme can be found in [1] p71, prob 2.12.

of partitioning step 1.

Step 2) Each processor's set of memory references, with one (locally combined) representative for each of the $k \leq \log n$ true (disambiguated) address references, is partitioned among $\log n$ bins. Simply stated, a record with the hashed address (t, x), with $t \in [0, \log n - 1]$ and $x \in [0, nm - 1]$, is placed in bin t.

This step clearly runs in $O(\log n)$ time.

Step 3) The contents of each bin are then sorted with an efficient sorting algorithm. Lastly, the $\log n$ sorted sequences are concatenated to produce the desired result.

Step 3 requires a performance proof, since we must assert that all n processors can complete this sorting task in $O(\log n)$ time.

Technically, each processor has up to $\log n$ values derived from a (κ)-wise independent hash function. But since $\kappa > \log n$, we can take these values to be fully random. The results of this paper also show that we can set $\mu = 1$, which gives the uniform distribution. We prove the performance bound for this case; it is straightforward to extend the conclusion to larger μ.

For convenience, we rescale the sorting problem with n replacing $\log n$, so that we now have 2^n processors that each sort n random numbers in a fixed interval. Let algorithm A sort a set D of n random real numbers in $[0, m - 1)$ as follows:

i) Partially sort the data by placing each $x \in D$ into $Bucket[\lfloor n\frac{x}{m-1} \rfloor]$.

ii) For each bucket, use a fast sorting algorithm to sort the bucket contents.

iii) Concatenate the sorted contents of the buckets.

The mean performance of Algorithm A is analyzed as follows.

Lemma 8. Let $[0, m - 1)^n$ comprise all sequences of n real values selected uniformly from $[0, m - 1)$. Then for any fixed $c > 0$, there is a fixed d such that algorithm A sorts a fraction exceeding $1 - e^{-cn}$ of all $D \in [0, m-1)^n$ in dn steps, where $c = d - 2 - \ln(d-1)$.

Proof: See [22]. ∎

Our sorting problem (as rescaled) concerns the slowest of 2^n processors. Let t_i

be the time that the i'th processor uses to sort its n random numbers. Then we can conclude that $Prob\{\max_{1\leq t\leq 2^n}(t_i) - n > (d-1)n\} < 2^n((d-1)e^{2-d})^n$, whence setting $d = 4$ gives the value $(\frac{6}{e^2})^n$, which is exponentially small. In the rescaled problem comprising n processors, each with $\log n$ numbers, the probability is polynomially small in n and parameter d.†

Ranade's combining scheme ensures that common references are merged, so that each memory bank executes just one memory access for each distinct memory reference. To conclude that this emulation scheme can complete a PRAM step of $n\log n$ threads in an expected time of $O(\log n)$, we establish, in Lemmas 9 10, 11, and 12, four distributional characteristics of the underlying hashing scheme. Their proofs and formulations are not especially sharp; rather, they were written to be simple.

Lemma 9. Let D_0 be a set of $n\log n$ distinct values belonging to a domain D, and let f be a (κ,μ)-wise independent hash function mapping D into $[0, nm-1]$. Let d_i, for $i = 0, 1, \ldots, n-1$, be the number of keys in D_0 that are mapped into $[im, (i+1)m)$, so that $d_i = |\{x \in D_0 : im \leq f(x) < (i+1)m\}|$. Let $\kappa = \beta\log n$, for any fixed $\beta > 0$. Then for any fixed $c > 0$, there is a fixed d such that $Prob\{d_i > d\log n\} \leq \frac{1}{n^c}$.

Proof: We can suppose that $d\log n \geq \kappa$. If not, the analogous calculations are even simpler. Alternatively, it suffices to use the following proof, but with $d\log n$ replacing κ. We take, d_1 to be the number of distinct memory references, among the $n\log n$ references of a single PRAM step, that are hashed to locations in memory module 1. We use (κ,μ)-wise independence to compute expectations that bound $Prob\{d_1 \geq d\log n\}$:

$$Prob\{d_1 \geq d\log n\} \leq \frac{\mathrm{E}[\binom{d_1}{\kappa}]}{\binom{d\log n}{\kappa}} \leq \frac{\mu\binom{n\log n}{\kappa}\frac{1}{n^\kappa}}{\binom{d\log n}{\kappa}} \leq \mu(\frac{e}{d})^\kappa.$$

This probability is polynomially small in n, for $\kappa = \beta\log n$ with fixed $\beta > 0$. ∎

†A value is polynomially small in n if it depends on parameters that can be set so that it is less than $\frac{1}{n^c}$ for any fixed c and sufficiently large n.

Lemma 10. Let D_0 be a set of $n \log n$ values in a domain $D = [0, nm-1]$, and let f be a (κ, μ)-wise independent hash function that maps D into D. Let Δ_i be the size of the preimage of $f[D_0] \cap [im, (i+1)m-1]$: $\Delta_i = |\{x \in D : \exists y \in D_0 \ s.t. \ f(x) = f(y) \text{ and } im \leq f(y) < (i+1)m\}|$. Let $\kappa = \beta \log n$, for any fixed $\beta > 0$. Then for any fixed $c > 0$, there is a fixed d such $Prob\{\Delta_i > d \log n\} \leq \frac{1}{n^c}$.

Proof: We can assume that $d \log n > \kappa$. Let d_i be the number of values in D_0 that are mapped into $[im, (i+1)m-1]$. By definition, Δ_i includes the count for d_i. We observe that $Prob\{\Delta_i > d \log n\} \leq Prob\{(\Delta_i - d_i > \frac{d}{2} \log n) \bigvee (d_i < \kappa/2)\} + Prob\{(\Delta_i - d_i > \frac{d}{2} \log n) \bigvee (\kappa/2 \leq d_i \leq \frac{d}{2} \log n)\} + Prob\{d_i > \frac{d}{2} \log n\}$. Lemma 9 ensures that $Prob\{d_i > \frac{d}{2} \log n\}$ is polynomially small. We use the (κ, μ)-wise independence to bound, in turn, each of $Prob\{(\Delta_i - d_i > \frac{d}{2} \log n) \bigvee (d_i < \kappa/2)\}$ and $Prob\{(\Delta_i - d_i > \frac{d}{2} \log n) \bigvee (\kappa/2 \leq d_i \leq \frac{d}{2} \log n)\}$.

The limited independence allows us to bound these probabilities by counting the expected number of pairs $(\mathcal{S}_1, \mathcal{S}_2)$ where $\mathcal{S}_1$ is a set of $j \leq \frac{\kappa}{2}$ references in D_0 that hash into module 1, and and $\mathcal{S}_2$ comprises $\kappa - j$ elements in $D - D_0$ that hash into the image of $\mathcal{S}_1$.

More precisely, we replace d by $2d$ so that the probabilities we seek to estimate are $p_1 = Prob\{(\Delta_1 - d_1 > d \log n) \bigvee (d_1 < \frac{\kappa}{2})\}$; $q_1 = Prob\{(\Delta_1 - d_1 > d \log n) \bigvee (d_1 \geq \frac{\kappa}{2})\}$.

Let f be the hash function. Then q_1 is overestimated by the following expectation

$$\frac{\mathrm{E}[|\{(\mathcal{S}_1, \mathcal{S}_2) \mid \mathcal{S}_1 \subset D_0, \mathcal{S}_2 \subset D - D_0, f(\mathcal{S}_1) \subset \text{module}_1, f(\mathcal{S}_2) \subseteq f(\mathcal{S}_1), |\mathcal{S}_1| = |\mathcal{S}_2| = \frac{\kappa}{2}\}|]}{\binom{d \log n}{\kappa/2}},$$

since there must be $\binom{d \log n}{\kappa/2}$ suitable subsets $\mathcal{S}_2$ among $d \log n$ items that collide with D_0 in the hashing range restricted to module 1. Similarly, p_1 is bounded in the same way, except that we set $|\mathcal{S}_1| = j$ and $|\mathcal{S}_2| = \kappa - j$ and sum over $j = 1, 2, \ldots, \kappa/2 - 1$. The j'th term in the summation is divided by $\binom{d \log n}{\kappa - j}$, rather than $\binom{d \log n}{\kappa/2}$.

In the following calculations, we make an additional over-count by treating $|D - D_0|$ as mn. We can also suppose that $(\frac{\kappa}{2})^2 \leq d(\log n)^2$, which ensures that, in the summation below, $\frac{(\log n)^j j^{\kappa - 2j}}{(d \log n)^{\kappa - j}}$ is maximized at $j = \frac{\kappa}{2}$.

$$
\begin{aligned}
p_1 &\le \sum_{j=1}^{\kappa/2} \mu \binom{n \log n}{j} (\frac{1}{n})^j \frac{(\frac{mn}{\kappa-j})(\frac{j}{mn})^{\kappa-j}}{\binom{d \log n}{\kappa-j}} \\
&\le \mu \sum_{j=1}^{\kappa/2} \frac{(\log n)^j}{j!} \frac{j^{\kappa-j}}{(d \log n)^{\frac{\kappa-j}{}}} \\
&\le \mu \sum_{j=1}^{\kappa/2} e^{j+\kappa-j} \frac{(\log n)^j j^{\kappa-j-j}}{(d \log n)^{\kappa-j}} \\
&\le \mu \sum_{j=1}^{\kappa/2} e^{\kappa} \frac{(\log n)^{\kappa/2}}{(d \log n)^{\kappa/2}} \\
&\le \mu \frac{\kappa}{2} (\frac{e^2}{d})^{\kappa/2}.
\end{aligned}
$$

As for q_1, the resulting expression is just the term for p_1 with $j = \kappa/2$, which is bounded by $\mu(\frac{e^2}{d})^{\kappa/2}$. Thus, q_1 is already counted in our estimate for p_1, which could have limited the summation range to $1 \le j < \kappa/2$.

These bounds are all polynomially small in n. It follows that for suitable fixed d, $Prob\{\Delta_1 > d \log n\} \le \frac{1}{n^c}$. As all modules have the same statistical performance, the bound is established. ∎

Consequently, $Prob\{\max_i \Delta_i > d \log n\} \le \frac{1}{n^{c-1}}$.

We now observe that the hashing of D distributes the data quite evenly among the n modules; there is only a polynomially small chance that any module will have more than dm records hashed into its address range, and we can choose, say, $d = 3$.

Lemma 11. Let f be a $(\kappa, \mu,)$-wise independent hash function that maps $[0, nm-1]$ into $[0, nm-1]$, and let d_i be the size of the preimage of $[im, (i+1)m-1]|$, so that $d_i = |\{x : f(x) \in [im, (i+1)m-1]\}$. Let $\kappa = \beta \log n$, for any fixed $\beta > 0$. Then for any fixed $c > 0$, there is a fixed d such that $Prob\{d_i > dm\} \le \frac{1}{n^c}$.

Proof: Let m_i be the number of items, among the nm data, that are mapped to module i by the hash function f, for $i = 1, 2, \ldots, n$. Under (κ, μ)-wise independence, we can compute that

$$
Prob\{\binom{m_i}{\kappa} > \binom{\gamma m + \kappa}{\kappa}\} \le \frac{\mathrm{E}[\binom{m_i}{\kappa}]}{\binom{\gamma m+\kappa}{\kappa}} \le \mu \frac{\binom{nm}{\kappa}\frac{1}{n^\kappa}}{\binom{\gamma m+\kappa}{\kappa}} < \mu \frac{(nm)^\kappa \frac{1}{n^\kappa}}{(\gamma m)^\kappa} = \mu(\frac{1}{\gamma})^\kappa,
$$

where $\kappa = \beta \log n$. Taking $\gamma = 2^{c/\beta}$, gives the bound n^{-c} as required. ∎

Since there are n modules with identical hashing statistics, the probability that a hashing failure occurs by allocating more than dm data to some module is the polynomially small $p < \sum_{j=1}^{n} \frac{1}{n^c} = \frac{1}{n^{c-1}}$.

It should be noted that good performance does not require a failure probability q to be astronomically small. Indeed, suppose that the PRAM program can be emulated in time T, where T includes the time for a successful hashing and loading of the data (without any failures to balance the data). Then we can expect that an hashing failure will run in a time that is also bounded by T. It even suffices to declare that a failure has occurred if the time exceeds $2T$, and begin afresh at that point with new hashing seeds. Even in this case, the degradation in T due to failures to balance the data gives an expected time of $T(1 + 2q + 2q^2 + 2q^3 + \ldots)$.

The only issue that remains to be shown before the emulation bound can be established is a bound for the expected time for a PRAM emulation step in the exceptional case where some processor must encounter too many keys while executing the references routed to its memory module. This event, according to Lemma 10, occurs with polynomially small probability, but consumes a time (which would be proportional to $\max_i \Delta_i$) that is not necessarily bounded by a small polynomial in n. (The reason that the time can be so high is that a hashed memory reference can collide with huge numbers of other keys that are not being referenced by other active processes.)

There are several ways to bound the expected service time for this exception.

One way is to declare the hashing a failure if any location is the image of n or more keys in D. It is easy to show that this event has a probability that is superpolynomially small in n, provided $m = O(n^k)$, for some fixed size k. If, in this slightly restrictive case, we restart the emulation afresh in this circumstance, then a successful hashing might have steps where an individual processor receives at most $n \log n$ references that each access hashing-based chains of length n, in the worst case. Such an $O(n^2 \log n)$

time delay would occur with a probability that is bounded by n^{-c}, so that the expected cost from this exception is $o(1)$ per time step.

However, we will adhere to the convention that a failure occurs only if too many keys are hashed to a memory module as described in Lemma 11. Other kinds of exceptional events can, upon rare occasion, cause long delays, We will charge each time step with a time penalty to account for the expected delay from these exceptional instances. One of these instance types is covered by Lemma 12, which accounts for the lost time due to too many data elements colliding with the hash address of a referenced datum.

Lemma 12. Let the domain $[0, nm-1]$ be mapped into itself by a (κ, μ)-wise independent hash function f, where $\kappa = \beta \log n$, for any fixed $\beta > 0$. Let D_t be the set of data referenced at the t-th time step of the PRAM algorithm, and let Δ_t be the number of elements in $D - D_t$ that collide with elements in D_t:

$$\Delta_t = |\{x \in D : (x \notin D_t) \bigvee (f(x) = f(y) \text{ for some } y \in D_t)\}.$$

Then for any fixed c, there is a fixed d so that $\mathrm{E}[\Delta_t \cdot \mathcal{X}(\Delta_t > dn \log n)] < \frac{1}{n^c}$.

Proof: We can use (κ)-wise independence to overestimate the expected sum of all chains for the $n \log n$ references in the case that $\Delta_t > dn \log n$, say. We suppress the t notation since the bound will apply for all t. Then $\mathrm{E}[\Delta \cdot \mathcal{X}(\Delta \geq dn \log n + 1)] < \mathrm{E}[\frac{\kappa\binom{\Delta}{\kappa}}{\binom{dn \log n}{\kappa - 1}}]$, since the ratio will be at least Δ, provided $\Delta > dn \log n$. Continuing the estimation gives: $\mathrm{E}[\frac{\kappa\binom{\Delta}{\kappa}}{\binom{dn \log n}{\kappa - 1}}] < \frac{\mu\kappa\binom{mn}{\kappa}(\frac{n \log n}{mn})^{\kappa}}{\binom{dn \log n}{\kappa - 1}} < \mu(\frac{e}{d})^{\kappa - 1} n \log n$. This expression is polynomially small in n. ∎

With one more definition, we will be able to establish an optimal speedup in emulation mode.

Let Ω be a network. We say that a no-load memory reference takes time T on Ω if, in the worst case, a single write (with all other process requests suspended) can be executed on Ω in a worst-case time of T.

Corollary 4. Suppose that a no-load distributed memory reference can be processed by

an n-processor $n \times \log n$ butterfly network in $\log n$ time. Then a T-time $n \log n$-processor PRAM algorithm with m words of shared memory can be emulated on a pipelined n-processor $n \times \log n$ butterfly network in expected time $O(T \log n)$. Furthermore, for any fixed c, there is a fixed d such that the algorithm runs in time $dT \log n$ with probability exceeding $1 - \frac{1}{n^c}$. The hashing is performed with a (κ, μ)-wise independent hash function where $\kappa = \beta \log n$, for suitable fixed β.

Proof: We can estimate the performance of the emulation procedure as follows. Lemma 11 ensures that the hashing succeeds in distributing the data adequately well among the n memory modules with probability $1 - \frac{1}{n^c}$. Moreover, the probability of a hashing failure has a nominal impact on the expected running time of the algorithm. Lemmas 7 and 8 ensure that each of the n processors will, when jointly emulating a PRAM step of $n \log n$ threads, be able to sort their locally derived $\log n$ memory references in $d \log n$ time with probability $1 - \frac{1}{n^c}$. Furthermore, the sorting time has a worst-case time of $O((\log n) \log \log n)$, whence the expected time of the sorting step is bounded by $d \log n + \frac{O(\log n) \log \log n}{n^{c-1}} = O(\log n)$.

Lemma 9 can be interpreted (with trivial adjustments of c and d) to state that with probability $1 - \frac{1}{n^c}$, each memory module will receive at most $d \log n$ references in a single PRAM time-step. We can take the reference count to a single module to be a worst case $n \log n$ with probability $\frac{1}{n^c}$, which will give a negligible expected time penalty.

Lemma 10 can be interpreted to ensure that each memory module will, with probability exceeding $1 - \frac{1}{n^c}$, be able to service the incoming memory references by following at most $\frac{\kappa}{2}$ chains with a total of $d \log n$ members in $O(d \log n)$ time during an individual parallel emulation step of an $n \log n$-thread PRAM step.

Lemma 12 says that the expected time used during an exception to the typical case covered by Lemma 10 is polynomially small.

Given an algorithm that takes T PRAM steps, let T_R be the time it takes in our emulation mode of Ranade's scheme. These Lemmas show that if a hashing failure does

not occur, then $\mathrm{E}[T_R] < dT\log n$, for a suitable constant d. As explained following the proof of Lemma 11, the possibility of a hashing failure increases the expected running time by a nominal factor.

To show that T_R satisfies such a time bound with high probability, we bound $Prob\{T_R > 2dT\log n\}$. Suppose, for the moment, that no hashing failure occurs. Let T_N be the running time for the algorithm in this case. Then there will be T PRAM emulation steps, and each step has five potential exceptions where slow processing could occur, (We count one exception for each of Lemmas 7, 8, 9, and 12, plus one for Ranade's routing scheme.) Let X be the sum of the time consumed by each of these $5T$ exceptional events. Most, of course, will be zero, since they will not occur. It follows that:

$$\begin{aligned} Prob\{T_N > 2dT\log n\} &< Prob\{\text{a hashing failure occurs}\} + Prob\{X > dT\log n\} \\ &< n^{-c} + \frac{\mathrm{E}[X]}{dT\log n} \\ &< n^{-c} + \frac{5Tn^{-c}}{dT\log n}. \end{aligned}$$

This probability is polynomially small as claimed. A careful modeling of T_R would be as follows. Let $b_1, b_2, \ldots$ be an infinite family of independent Bernoulli trials that model the probability of a hashing failure: $Prob\{b_i = 1\} = n^{-c}$. Let $T_{N,1}, T_{N.2}, \ldots$ be an infinite family of independent random variables distributed according to T_N. Then

$$T_R = T_{N,1} + b_1 T_{N,2} + b_1 b_2 T_{N_3} \cdots,$$

It now follows that the expected time will exceed $4dT\log n$ with a probability that is polynomially small, provided 2 independent initializing seed sets are available. That is, either b_1 is 1, or $T_{n,1}$ is at least $2dT\log n$. But both events are polynomially small. ∎

While the principal results of this paper suggest that there is little additional cost in creating constant time hash functions that are far more random than $O(\log n)$-wise independent, our analysis of PRAM emulation has followed a more parsimonious path by restricting the randomness allocated to these functions to be $\beta\log n$, for a fixed value

of β. The reasons for this restriction are two-fold. First, the sufficiency of $O(\log n)$-wise independence gives hope that even less randomness might suffice for real implementations. Second, the realization of fast random functions really awaits the discovery of suitable graphs; it is conceivable that good graphs might be found first for more moderate degrees of randomness, such as those which give $O(\log n)$-wise independence.

We conclude this section by suggesting that these PRAM emulation results are really feasibility arguments. While methods that are provably sound provide assurances based upon mathematical argument, and offer the best hope that techniques and architectures might scale without unanticipated performance losses, they are frequently no substitute for alternatives that might be difficult or even impossible to analyze completely, but which achieve much better performance in practice. Thus PRAM algorithms in general, and PRAM emulation schemes in particular do not necessarily tell us the best way to implement highly parallel algorithms or architectures. Rather, they suggest implementation and development paths, and serve to show that there might not be any intrinsic technology barrier for such massively parallel computation. At the same time, we are obliged to accept these kinds of results with a degree of skepticism, for the real barriers might well lie in the marketplace, where backwards compatibility, cost, demand and market share take precedence over feasibility results.

For completeness, we note that Kruskal, Rudolph and Snir use pipelines of depth $n^{1/\epsilon}$ to get a parallel emulation time that is optimal up to a factor of $\frac{1}{\epsilon}$ [12]. Versions of some of the basic counting estimates can also be found elsewhere [12, 16, 18, 28]. For example, a formulations comparable to Lemmas 9 and 11 can be found in [18]. Ranade uses the fact that degree h polynomials functions over a finite field have at most h roots, to bound the preimage size to $O(\log n^2)$ for the fetches that must be executed by a single row of $\log n$ processors. Valiant uses the root bound plus secondary hashing to control the fetch time. Lemma 13 establishes the necessary adaptations from the statistical properties of (κ)-wise independence, plus the implicit synchronization of

Ranade's emulation scheme (with ghost messages), which ensures that no processor can begin a new computation phase before all data accesses to remote memory have been successfully completed.

It should also be noted that the Fast Fourier Transform can be used to compute k evaluations of a degree k polynomial in $k \log k$ time (cf. [1]). Thus it is possible to use the above pipeline strategy on n processors with $\log n$ degree hash functions to attain a performance cost of $O(\log \log n)$ operations per memory reference rather than a naive $\log n$. We have shown that this multiplicative performance penalty can, in theory, be reduced to an asymptotic $O(1)$.

6. Conclusions

The high independence exhibited by our hash functions enriches the class of probabilistic algorithms that can be shown to achieve their expected performance in real computation. Proofs need not be restricted to (κ)-wise independence for constant κ, and probability estimates can use the probabilistic method to calculate the expected number of κ-tuples satisfying various behavior criteria.

On the other hand, it is worth noting that the fast hash functions described in this paper are not really necessary for pure routing problems. After all, if an adequately random assignment of intermediate destinations provides, with very high probability, nearly optimal performance in a Valiant-Brebner style of routing [29], then the same destinations could be used for many consecutive routings.

What these fast hash functions really provide is nearly uniform random mappings of data to modules and cell locations and a convenient way to assert that with high probability, no step in an n^k emulation sequence takes more than $O(\log n)$ time to complete. Thus, fast hash functions are even important for fast deterministic routing schemes, if large amounts of data have to be stored in a randomized manner. In addition, hash functions computed from destination addresses provide a way for common memory references to be fully combined en route in Ranade's simple queue management scheme,

and this might be important if combining is required to avoid hot spot contention.

But while the the asymptotic characteristics of fast hash functions are now well understood, the feasibility question remains wide open. Without doubt, the most significant open problem is to find good local expander-like graphs that are defined by short efficient programs. The discovery of such an object might have a very beneficial effect on the practicality of such a class of functions.

Appendix 1

Fact 1: Let $P_k = \{p \mid p \text{ is prime and } p \in (n^k \log m, (2+\beta)n^k \log m)\}$, for some small suitably fixed $\beta > 0$. Then

$$\forall x \neq y \in [0, m-1]: \ Prob_{p \in P_k}\{x = y \mod p\} < n^{-k}.$$

Proof: [15],[7]. The Prime Number Theorem says that $|P_k| = \frac{(1+\beta)n^k \ln m}{k \ln n + \ln \ln m}(1 - o(1))$, whence fewer than $1/n^k$ of the elements of P_k can divide $|x-y|$, since $\prod_{p \in P_k} p > (n^k \ln m)^{|P_k|} > (m)^{n^k}$. ∎

Fact 2: Let $F_0(p) = \{h \mid h(x) = (ax+b \mod p) \mod n^k, \ a \neq 0, b \in [0, p-1]\}$, where $p > n^k$ is prime. Then

$$\forall x \neq y \in [0, p-1]: \ Prob_{f \in F_0(p)}\{f(x) = f(y)\} \leq n^{-k}.$$

Proof: [4]. Given x and y, $x, y \in [0, p-1]$, $x \neq y$, the number of different $f \in F_0(p)$ where $f(x) = f(y)$, is precisely the number of 2×2 linear systems in a and b:

$$\begin{cases} ax + b = c + dn^k \mod p \\ ay + b = c + en^k \mod p \end{cases} \qquad c, d, e \geq 0; \ c + dn^k < p; \ c < n^k; \ e \neq d; \ c + en^k < p.$$

This system is designed so that $c + dn^k$ can have p different values: c ranges from 0 to $n^k - 1$, and dn^k gives increments of n^k. The same is true of $c + en^k$, since it has the same format. Both expressions equal c when taken modn^k, and all possible values are captured by this representation. The parameters e and d cannot be equal because the solution to the system would then give $a = 0$. Furthermore, a cannot be zero if $e \neq d$,

which is required for x and y to be distinct. Straightforward counting shows that there are at most $\lceil p/n^k - 1 \rceil$ different values available for e. Since there are $p(p-1)$ different functions in F_0, and $f(x) = f(y)$ for at most $p\lceil p/n^k - 1 \rceil \leq p\frac{p-1}{n^k}$ of them, the result follows. ∎

Combining Facts 1 and 2 shows that a hash function selected at random from $F_0^k = \cup_{p \in P_k} F_0(p)$ will, with probability exceeding $1 - 2\binom{n}{2}n^{-k}$, map n items from $[0, m-1]$ into $[0, n^k - 1]$ with no collisions at all among its $\binom{n}{2}$ pairs. We could take $k = 4$, so that the probability of a collision is below $1/n^2$, and assume the functions $F_0(p)$ are defined as in Fact 2 for $p \approx n^4 \ln m$.

References

[**1**] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[**2**] R. Aleliunas. Randomized parallel communication, 13*th PODC*, Aug., 1982, pp. 60-72.

[**3**] C. H. Bennett, G. Brassard, L. Salvail, and J. Smolin. Experimental Quantum Cryptography, *Journal of Cryptology*, Vol. 5(1), 1992, pp. 3–28.

[**4**] J.L. Carter and M.N. Wegman Universal Classes of Hash Functions, *Journal of Computer and System Sciences* 18, 1979, pp. 143–154.

[**5**] A. Chin. Locality-Preserving Hash Functions for General Purpose Parallel Computation, *Algorithmica*, Vol. 12, 1994, pp. 170–181.

[**6**] M. Dietzfelbinger, A. R. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, R. E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. *SIAM J. Comput.*, Vol. 23(4), 1994, pp. 738–761.

[**7**] M.L. Fredman, J. Komlós and E. Szemerédi. Storing a Sparse Table with $O(1)$ Worst Case Access Time, *Journal of the Association for Computing Machinery*, Vol. 31, No. 3, July 1984, pp. 538–544.

[**8**] L.A. Goldberg, Y. Matias, and S. Rao. An optical simulation of shared memory, *SIAM J. Comput.*, Vol. 28(5), 1999, pp. 1829-1847.

[**9**] A. Karlin and E. Upfal. Parallel hashing: An efficient implementation of shared memory, *JACM*, Vol. 35, 1988, pp. 876–892.

[**10**] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM Simulation on a Distributed Memory Machine, *Algorithmica*, Vol. 16, 1996, pp. 517–542.

[**11**] D.E. Knuth. *The Art of Computer Programming*, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

[**12**] C.P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, Vol. 71(1), 1990, pp. 95–132.

[**13**] G. Lueker and M. Molodowitch. More Analysis of Double Hashing, *Combinatorica*, Vol. 13, No. 1, 1993, pp. 83-96.

[**14**] Y. Matias and A. Schuster. Fast, efficient mutual and self simulations for shared memory and reconfigurable mesh, *Parallel Algorithms and Applications*, Special Issue on Algorithms for Enhanced Mesh Architectures, Vol. 8, 1996, pp. 195–221.

[**15**] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, Chap 3, Sect 2.3, Springer-Verlag, Berlin Heidelberg, 1984.

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

[**17**] N. Pippenger. Parallel communication with limited buffers, 25*th Annual Symposium on Theory of Computing*, May, 1984, pp. 127–136.

[**18**] A.G. Ranade. How To Emulate Shared Memory, *JCSS*, Vol. 42, No 3, 1991. pp. 307–326. See also the preliminary version, which appeared in 28$^{\text{th}}$ *Annual Symposium on Foundations of Computer Science*, pp. 185–194.

[**19**] J.P. Schmidt and A. Siegel. The analysis of closed hashing under limited randomness, 22*nd Annual Symposium on Theory of Computing*, 1990, pp. 224–234.

[**20**] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding Bounds for Applications with Limited Independence, *SIAM J. Discrete Math.*, Vol.8 (2) 1995.

[**21**] A. Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications, 30$^{\text{th}}$ *Annual Symposium on Foundations of Computer Science*, 1989, pp. 20–25.

[**22**] A. Siegel. Median Bounds and their Application. To appear, Journal of Algorithms.

[**23**] B. Smith. A pipelined, shared resource MIMD computer, *Proceedings 1978 International Conference on Parallel Processing*, 1978, pp. 6–8.

[**24**] J. Spencer. *Ten Lectures on the Probabilistic Method*, SIAM, 1987.

[**25**] E. Upfal. Efficient schemes for parallel computation, *JACM*, Vol. 31, 1984, pp. 507–517.

[**26**] E. Upfal. A probabilistic relation between desirable and feasible models of parallel computation, 16*th Annual Symposium on Theory of Computing*, May, 1984, pp. 258–265.

[**27**] E. Upfal and A. Wigderson. How to share memory in a distributed system, *JACM*, Vol. 34, 1987, pp. 116–127.

[**28**] L.G. Valiant. *General Purpose Parallel Parallel Architectures*, TR-07-89, Center for Research in Computing Technology, Harvard University, Cambridge, MA, 1989.*Handbook of Theoretical Computer Science*, Vol. A.J. van Leeuwen, ed., Elsevier, 1990, pp. 943–971.

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

[**30**] M.N. Wegman and J.L. Carter. New hash functions and their use in authentication and set equality, *JCSS*, Vol. 22, 1981, pp. 265–274.

[**31**] A.C. Yao. Uniform Hashing Is Optimal, *Journal of the Association for Computing*

Machinery, Vol. 32, No. 3, July, 1985, pp. 687–693.