On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications
(Extended Abstract)

Alan Siegel[†]

Department of Computer Science
Stanford University
Stanford, CA 94305

and

Courant Institute
New York University
New York, NY 10012

Summary

A growing number of probabilistic algorithms have been shown to work well under the assumption of limited randomness. For example, recent randomized routing schemes for size n Omega networks have been proven to give optimal expected performance, given a random $\log n$-wise independent hash function. Similar hash functions have been shown to support near optimal expected probe performance for double hashing, under suitably moderate load factors.

This paper provides a mechanism for constructing $\log n$-wise independent hash functions that can be evaluated in $O(1)$ time. In particular,

- A probabilistic argument shows that for fixed $\epsilon < 1$, a table of n^{ϵ} random words can be accessed by a *small* $O(1)$-time program to compute a family of $n^{\epsilon^3/(2k^2)}$-wise independent hash functions mapping $[0,\ n^k] \mapsto [0,\ n^k]$, for fixed k.
- An explicit algorithm for such a family is also given, which achieves comparable performance, for all practical purposes.
- A lower bound shows that such a program must take $\Omega(k/\epsilon)$ time, and a probabilistic argument shows that programs (which may not necessarily be short) can run in $O(k^2/\epsilon^2)$ time.
- An immediate consequence of these constructions is that double hashing using these universal functions has (constant factor) optimal performance in time, for suitably moderate loads.
- Another consequence is that a T-time PRAM algorithm for $n \log n$ processors (and n^k memory) can be emulated on an n-processor machine interconnected by an $n \times \log n$ Omega network with a multiplicative penalty for total work that, with high probability, is only $O(1)$, rather than $O(\log n)$. (The running time is, w.h.p., $O(T \log n)$ rather than $O(T \log^2 n)$.)

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

1. Introduction

L. Carter and M. Wegman introduced universal hash functions [CW-79] and thereby provided a theoretical framework to formalize methods to exploit actual hash functions exhibiting fixed degrees of freedom. Related works [WC-79], [MV-84] have sometimes required a little more limited randomness, which is often formalized as in

Definition 1.

We say that a family of hash functions F with domain D and range R is *$(h)_\mu$-wise independent* if $\forall\ y_1, \dots y_h \in R, \forall$ distinct $x_1, x_2, \dots x_h \in D$:

$$|\{f \in F : f(x_i) = y_i,\ i = 1, 2, \dots, h\}| \le \mu \frac{|F|}{|R|^h}.$$

Thus the distribution of a random $f \in F$ on any h points is nearly uniform, and $(h)_\mu$-wise independence implies $(j)_\mu$-wise independence for $j < h$. For expositional simplicity, we will frequently suppress the μ dependence and simply refer to h-wise independence.

The limited randomness provided by such classes 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, recent randomized routing schemes for size n Omega networks have been proven to give optimal expected performance (up to constant factors), given a random $O(\log n)$-wise independent hash function ([KU-86], [Ra-87]). The hash functions used to date have been typically congruential polynomials of degree $\beta \log n$. In the case of randomized routing on n-node bounded degree graphs, the $\log n$ cost to compute each address is

CH2806-8/89/0000/0020/$01.00 © 1989 IEEE

readily subsumed by the $\Omega(\log n)$ delay in routing the data.

Recently, $O(\log n)$-wise independent hash functions have also been shown to give near optimal expected probe performance for double hashing, under suitably moderate load factors ([SS-89]). But this efficiency is only in terms of probe counts; the cost to compute a single hash address is $c \log n$, given the hash functions developed to date. Even in the case of PRAM emulation, the possibility of exploiting pipelining to mask latency is precluded for read intensive algorithms, if each read requires a $\log n$-time address computation as in [KU-86] and [Ra-87].

Is there an inherent $\log n$ penalty for computing such hash functions, or can we do better?

This paper shows how to trade the time complexity of the hash function for the number of random bits provided to it, and gives a mechanism for computing $\log n$-wise independent functions in $O(1)$ time from n^ϵ random words, for any fixed $\epsilon < 1$.

An immediate consequence of these constructions is that double hashing using these universal functions has (constant factor) optimal performance in time, for suitably moderate loads. Another consequence is that a T-time PRAM algorithm for $n \log n$ processors (and n^k memory) can be emulated on an n-processor machine interconnected by an $n \times \log n$ Omega network with a multiplicative penalty for total work that, with high probability, is only $O(1)$, rather than $O(\log n)$.

The hash function

We first suppress the issue of program size and construct a family of fast $\beta \log n$-wise independent hash functions that map $[0, n^k - 1]$ onto $[0, n^k - 1]$ and use n^ϵ words of random input. So suppose we have a resource of about n^ϵ random words, for some $\epsilon < 1$. It would seem appropriate to have a mechanism that associates each element in $[0, n^k - 1]$ with a few of these words. We use a bipartite graph G on the vertex sets $[0, n^k - 1]$ and $[0, n^\epsilon - 1]$) to achieve this mapping. Accordingly, we formalize the suitability of such graphs in

Definition 2.

An (n, k, ϵ, d, h)-*weak concentrator* is a bipartite graph on the sets of vertices I (inputs) and O (outputs), where $|I| = n^k$, and $|O| = n^\epsilon$, that has degree d for each node in I, and that has, for any h inputs, edges matching them with some h outputs.

Lemma 1.

Suppose $h < \frac{n^{\epsilon^2/k}}{2e^2}$ and $d = 2 + k/\epsilon$. Then (n, k, ϵ, d, h)-weak concentrators exist.

Proof: A straightforward counting argument shows that the probability is less than 1 that a randomly generated (n, k, ϵ, d, d)-weak concentrator fails to be a (n, k, ϵ, d, h)-weak concentrator:

$$\begin{aligned}\Pr\{\text{failure}\} &< \sum_{d<j\leq h} \binom{n^k}{j}\binom{n^\epsilon}{j-1}\left(\frac{j-1}{n^\epsilon}\right)^{jd} \\ &< \sum \frac{n^{kj+j\epsilon} j^{2j+jk/\epsilon}}{j!(j-1)!n^{2j\epsilon+jk}} \\ &< \sum_{j\leq h} (e^2 j^{k/\epsilon}/n^\epsilon)^j < 1. \qquad \blacksquare\end{aligned}$$

Let G be an (n, k, ϵ, d, h)-weak concentrator. For each input i in G, let i's d neighbors in G be stored in the set $Adj(i)$. Let M_m be an $n^\epsilon \times d$ array whose concatenated contents is $m \in [0, p-1]^{n^\epsilon d}$, for some prime $p > n^k$.

Define the random hash function

$$f_G^m(i) = \sum_{j \in Adj(i), 0 \leq k < d} M_m(j, k) i^k \pmod{p}.$$

Lemma 2.

Let G be an (n, k, ϵ, d, h)-weak concentrator. Then $\{f_G^m\}_{m \in [0,p-1]^{dn^\epsilon}}$ is a $(h)_1$-wise independent family of hash functions mapping I into $[0, p-1]$.

Proof: It is not difficult to see that we need only establish the linear independence of the systems of equations that constrain the m values to yield arbitrarily specified values for f, on any h inputs. So suppose that specifying values for some input set I_0, where $|I_0| \leq h$, induces a minimal dependent linear system. That is, a linear combination of the rows in the linear system with row indices in I_0 sums to the zero vector, and no row has a coefficient of zero in the linear combination. Now I_0 sources $d|I_0|$ edges, which reach at least $|I_0|$ outputs, so there must be an output $y \in O$ having exactly q edges that originate in I_0, for some q where $0 < q \leq d$. Consider the linear subsystem with rows indexed by $I_1 = \{i \in I_0 : y \in Adj(i)\}$. By construction, $1 \leq |I_1| \leq d$. This subsystem in the variables $M_m(y, k)$, where $k = 0, 1, \ldots d-1$ comprises a $|I_1| \times d$ Vandermonde submatrix, which cannot be linearly dependent. Since no other rows with indices in I_0 have these variables present, the assumption that the system is minimal and dependent is contradicted. $\blacksquare$

So far, we have a probabilistic construction of fast hash functions that uses dn^ϵ random words of $k \log n$-bits each, and requires d^2 additions and d multiplications per evaluation. However, the graph G requires a large description for its dn^k edges. Obviously, a short (deterministic or effective probabilistic) algorithm for building weak concentrators, where an input's adjacency list could be generated in constant time would resolve the problem of the function's size. Unfortunately, the problem of finding such a representation seems to be quite difficult.

Fortunately, compact representations of less efficient hash functions can be easily attained. The price of simple compaction is formalized in

Lemma 3.

Let G be an $(n^{\epsilon/k}, k, \epsilon, d, h)$-weak concentrator. Then the Cartesian product $G^{k/\epsilon}$ is an $(n, k, \epsilon, d^{k/\epsilon}, h)$-weak concentrator.

Proof: (Sketch) Straightforward. ▮

Combining Lemmas 1,2, and 3 gives

Theorem 1.

$(\beta \log n)_1$-wise hash functions mapping $[0, p-1] \mapsto [0, p-1]$ can be defined in $(2+k/\epsilon)^{2k(2+k/\epsilon)/\epsilon} n^\epsilon$ space, and evaluated in $(2+k/\epsilon)^{2k(2+k/\epsilon)/\epsilon}$ time, where $p \approx n^k$. ▮

The h-wise independent hash functions sought are attained by computing these fast hash functions modulo n^k.

It is worth observing that the database of random numbers will be sufficient for computing an h-wise independent hash function if it is dh-wise independent; the numbers can be precomputed from dh random seeds.

It is also not difficult to establish marginal improvements in the hash functions. For example, if the weak concentrator G is also an expander for size h input sets, then the number of variables per output node can be reduced in proportion to the expansion factor. The fundamental problem, however, is to find good descriptive weak concentrators.

As presented, the families of hash functions were only constructed probabilistically; no explicit constructions were given. Formally, (that is, up to constant factors) this distinction is moot. By increasing the degree d of our weak concentrators by $O(r/\epsilon)$, we can ensure that with probability $1 - 1/n^r$, a randomly selected graph is a weak concentrator. Accordingly, we may simply increase the size of the hash family by indexing it over all graphs satisfying the size and (modified) degree parameters of Lemma 3. The resulting family $F_{M,G}$ is an explicit family of $O(1)$ time hash functions that is almost $O(\log n)$-wise independent. We may formalize the good behavior of such a class via

Definition 3.

We say that a family of hash functions F with domain D and range R is ***strongly r-practical*** $(h)_\mu$-***wise independent*** if $\exists \bar{F} \subset F : |F - \bar{F}| \leq |F|/|R|^r$ and $\forall\, y_1, \ldots y_h \in R, \forall$ distinct $x_1, x_2, \ldots x_h \in D$:

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

We have established

Theorem 2.

$F_{M,G}$ is an explicit family of strongly r-practical $(\beta \log n)_1$-wise hash functions mapping $[0, p-1] \mapsto [0, p-1]$, and is defined in $(2 + r/\epsilon + k/\epsilon)^{2k(2+k/\epsilon)/\epsilon} n^\epsilon$ space, and can be evaluated in $(2 + r/\epsilon + k/\epsilon)^{2k(2+k/\epsilon)/\epsilon}$ time, where $p \approx n^k$. ▮

A lower bound

We model a probabilistic hash function as a family of $(h)_\mu$-wise independent hash functions $F_M = \{f_m(x)\}_{m \in M}$ where $f_m : S \mapsto S$ as a follows. Each f_m is defined by the same algorithm, which inputs x and then reads d locations in an array A containing z values belonging to S. Index m is the string of concatenated data contained in A. The algorithm can even be viewed as probabilistic since values found in A might be used with x in an adaptive search to determine which other array locations to access. These values and x are then used deterministically to compute a hash address in S.

Theorem 3.

Let F_M denote a family of $(h)_1$-wise independent hash functions on S, where $M \subset S^z$. Then the time complexity T to evaluate $f \in F_M$ satisfies either $T \geq h$ or $T\binom{z}{T} \geq |S|$.

Proof: We may suppose that each computation of f probes d entries in the array A. We show that d satisfies the constraint for T. For each $s \in S$, we partition M into $M(s) = \{M_1^s, M_2^s, \ldots, M_{|S|^d}^s\}$, where M_i^s is the set of strings in M that cause the computation for $f(s)$ to find, for the d probes in A, the i-th sequence in some enumeration of $|S|^d$. By construction, the computation for $f_m(s)$, for any $m \in M_i^s$, probes the same d-tuple $(i_1, i_2, \ldots, i_d)$ of locations within the array A. Let $M^+(s) = \{g \in M(s) : |g| > |M|/|S|^{d+1}\}$. Consider the set $H = \{M^+(s)\}_{s \in S}$. Suppose that the $d+1$ members of H, $M^+(s_1), M^+(s_2), \ldots, M^+(s_{d+1})$, have an element $g \subset M$ in common. Since $g \in M^+(s_i)$ for distinct $s_1, s_2, \ldots s_{d+1}$, it follows that for any $m, m' \in g$, $f_m(s_i) = f_{m'}(s_i)$, for $i = 1, 2, \ldots, d+1$. For $d < h$, this contradicts the assumption of $(h)_1$-wise independence, since $|g| > |M|/|S|^{d+1}$. It follows that any element g can be shared by at most d members of H.

There are $\binom{z}{d}$ size d subsets of array locations, and each subset defines an implicit partition of M into $|S|^d$ subsets, none of which can be present more than d times in the multiset $\cup_s M^+(s)$. Thus $\sum_{s \in S} \sum_{g \in M^+(s)} |g| \leq d|M|\binom{z}{d}$. On the other hand, $\sum_{s \in S} |\cup_{g \in M^+(s)} g| \geq |S|(|M| - |M|/|S|)$. Moreover, it is not possible that both inequalities hold simultaneously as equalities. Consequently, $|S| - 1 < d\binom{z}{d}$. ▌

It follows that k/ϵ work is needed per evaluation of an h-wise independent hash function that uses a database of $z = n^\epsilon$ random $k \log n$-bit words to map $[0, n^k - 1] \mapsto [0, n^k - 1]$, for $h \geq k/\epsilon$. The bound also shows that if $\lceil z/2 \rceil \binom{z}{\lceil z/2 \rceil} < |S|$, then $T \geq h$. If, for example, $|S| = n^2$, then $\log n$ work is necessary for $\log n$-wise independence even if $2 \log n$ random words are provided. Clearly, the extension to strongly r-practical $(h)_\mu$-wise independent hash functions is straightforward (with very mild dependence on μ and r).

We also remark that the counting argument for Theorem 3 gives an average case time bound that exceeds $T(1 - o(1))$. More precisely, let $T < h$ be the bound from Theorem 3. Then the time, averaged over all items in S, is at least $T - \frac{(T-1)\binom{z}{T-1} + \ldots + 1\binom{z}{1}}{|S|} > T - \sqrt{T-1}$. It also follows that the average time is $T - O(1)$ if $T < cz$ for fixed $c < 1/2$. An amusing example is the case $|S| = n$, $1 < h < n$, and $z = n - 1$. The worst case time is 2, and the average, evidently, is at least $1 + 1/n$. A variety of readily found constructions (exhibiting quite different degrees of fairness) achieve the average $1 + (h-1)/n$, which is easily seen to be optimal.

Applications

Our first corollary is immediate.

Corollary 1.

For load factor $\alpha << .75$, $O(\log n)$-wise independent hash functions can be used for double hashing with constant expected probing for unsuccessful search. ▌

It should be noted that the [SS-89] result only needs $O(\log n)$-wise independent hash functions that map, say, $[0, n^4] \mapsto [0, n-1]$.

Randomized routing schemes and PRAM emulation have had a substantial and fruitful recent literature [VB-81], [Up-82], [Al-82], [Pi-84], [Up-84], [UW-84], [KU-86], [Ra-87]. In particular, [KU-86] and [Ra-87] show formally (and perhaps plausibly) how $n \log n$-processor Omega-like networks can, with very high probability, emulate an $n \log n$-processor PRAM with an optimal performance penalty that is "only" a multiplicative factor of $\log n$.

As is well known, three measures determine the efficiency of a PRAM emulation scheme: The first is *latency*, the number of steps on the network needed to completely emulate one step of the PRAM. Second is *pipeline depth*, the number of independent processes per processor that can be used to mask network latency. The third factor is the *switch complexity*, which concerns queue length, the complexity of queue management, and the availability of such embellishments as combining or multiprefix operations. The notion of exploiting large scale parallel slackness to mask network latency can be traced to Smith [S-78], and has been a subject of theoretical study in [MV-84] and [Va-89].

Both Karlin and Upfal [KU-86] and Ranade [Ra-87] presented schemes for an n-processor emulation of n-processor PRAM algorithms. For these parameters, no pipelining is possible. Moreover, since their feasibility results were based on hash functions that were $\log n$ degee congruential polynomials, each PRAM memory reference takes $\log n$ work no matter what. Given the resources required by address calculations, Karlin and Upfal did not even bother 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 [Ra-87] mentions that a scheme using $\log n$ reads per fetch readily solves the problem: his solution is to specify the location of items by their row number (which is in $[0, n-1]$) and the cell address of their module, but modulo the $\log n$ modules in a row of an $n \times \log n$ Omega network.

It is a simple matter to adapt the [KU-86] and [Ra-87] constructions to the case of emulating an $n \log n$ PRAM machine on a machine having one column of n processors interconnected by an $n \times \log n$ Omega network. A PRAM step of $n \log n$ parallel instructions is emulated by pipelining $\log n$ instructions per processor.

The machine would also have n memory modules, say, one per processor. Ranade's Common

PRAM emulation scheme applies with a few simple modifications: First, the hash function would still be used to map the PRAM address $x \in [0, n^k - 1]$ into $[0, \log n - 1] \times [0, n - 1] \times [0, \gamma n^{[k-1]}]$. The data packets are always kept locally lexicographically sorted (with the value x used to break and disambiguate ties). The first field, which, in Ranade's scheme, designated the column number of the destination module, is still used for the sorting of packets, but has no meaning in this case since only one column is active. The local process number (in $[1, \log n]$) for each packet might be explicitly listed in a separate field. The first phase of the algorithm requires each processor to provide its data in sorted order to the next stages as appropriate. This preprocessing can be done simply by following Ranade's approach: each processor uses its row of switches as a systolic bubble sorter for its packets. The "column" numbers cannot be arbitrarily set to the local process number for emulation of the Common PRAM, since combining would not be adequate to guarantee that only $O(\log n)$ messages would arrive at each memory module, per PRAM step. Such a scheme would, however, be adequate for an EREW emulator, and in this case the bubblesort can be skipped.

Now we can attain optimal speedup in emulation.

Corollary 2.

A T-time $n \log n$ processor PRAM algorithm with n^k words of shared memory can, with high probability, be emulated on an n processor $n \times \log n$ Omega network in time $O(T \log n)$.

Proof: (Sketch) The only issue to address concerns memory contention at the cell and module levels. We use open hashing with chaining and verify that w.h.p. data is well distributed; in each phase, only $O(\log n)$ references queue at any module, and they reference $O(\log n)$ buckets containing chains altogether comprising $O(\log n)$ items. ▌

Double hashing (c.f [SS-89]) provides a formally simpler hashing method with essentially the same performance.

It should be noted that a formulation comparable to Corollary 2, conditioned on the existence of suitable hashing algorithms/hardware, was recently given in [Va-89]. Versions of the basic counting estimates, with the exception of the bound on the aggregate number of collision items encountered by a batch of references queued at one memory module, can be found in [MV-84] along with some early analysis of pipelining and various hashing schemes.

Conclusions

Real machines have significant amounts of memory. We have shown how to exploit the capacity to store a sublinear sized database of random words in local memory to define highly random hash functions that can be evaluated in constant time. For the development of probabilistic algorithms and the use of large scale parallel machines, this capability has, at least, theoretical importance. We have also shown that such functions have an intrinsic tradeoff between their evaluation time and the storage reserved for precomputed data (or their amortized evaluation time and the space reserved for active storage).

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 [VB-81], then the same destinations could be used for many consecutive routings.

What these fast hash functions really provide is nearly uniform 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. As is well known, these hash functions also provide a way for common memory references to be fully combined en route to their destination within Ranade's simple queue management scheme, and this is important if combining is required to avoid hot spot contention.

The most significant open question is how to find good weak concentrators 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. Also of interest is the question of how to reduce the d random numbers stored per edge.

Acknowledgements

The author thanks F.T. Leighton and J.P. Schmidt for stimulating discussion.

References

[**AGM-87**] N. Alon, Z. Galil, and V.D. Milman. "Better Expanders and Superconcentrators," Journal of Algorithms, **8**, 1987, pp. 337–347.

[**Al-82**] R. Aleliunas. "Randomized parallel communication," 13*th* PODC, Aug., 1982, pp. 60-72.

[**CW-79**] J. L. Carter and M. N. Wegman "Universal Classes of Hash Functions," Journal of Computer and System Sciences 18, pp. 143–154 (1979).

[**FKS-84**] 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.

[**KU-86**] A. Karlin and E. Upfal. "Parallel Hashing - an Efficient Implementation of Shared Memory," 18*th* Annual Symposium on Theory of Computing, May, 1986, pp. 160–168.

[**LM-88**] G. Lueker and M. Molodowitch. "More Analysis of Double Hashing," 20*th* Annual Symposium on Theory of Computing, May, 1988, pp. 354–359.

[**LPS-88**] A. Lubotzky, R. Phillips, and P. Sarnak. "Ramanujan Graphs," Combinatorica, **8**(3), 1988 pp. 261–277.

[**Me-82**] K. Mehlhorn. "On the Program size of Perfect and Universal Hash functions," Proc. 23*rd* Ann. Symp. on Foundations of Computer Science, 1982, pp. 170–175.

[**Me-84**] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, Berlin Heidelberg, 1984.

[**MV-84**] 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.

[**Pi-84**] N. Pippenger. "Parallel communication with limited buffers," 25*th* Annual Symposium on Theory of Computing, May, 1984, pp. 127–136.

[**Ra-87**] A.G. Ranade. "How To Emulate Shared Memory," 28*th* Annual Symposium on Foundations of Computer Science, October 1987, pp. 185–194.

[**Sm-78**] B. Smith. "A pipelined, shared resource MIMD computer," Proceedings 1978 International Conference on Parallel Processing, 1978, pp. 6–8.

[**SS-89**] J.P. Schmidt and A. Siegel. "On aspects of universality and performance for closed hashing," 21*st* Annual Symposium on Theory of Computing, May, 1989.

[**Up-82**] E. Upfal. "Efficient schemes for parallel computation," 13*th* PODC, Aug., 1982, pp. 55-59.

[**Up-84**] 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.

[**UW-84**] E. Upfal and A. Wigderson. "How to share memory in a distributed system," 25*th* Annual Symposium on Foundations of Computer Science, October 1984, pp. 1701–180.

[**Va-89**] L.G. Valiant. "General Purpose Parallel Parallel Architectures," TR-07-89, Center for Research in Computing Technology, Harvard University, Cambridge, MA, 1989. (To appear in *Handbook of Theoretical Computer Science.*)

[**VB-81**] 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.

[**WC-79**] M. N. Wegman and J. L. Carter. "New Classes and Applications of Hash Functions," 20*th* Annual Symposium on Foundations of Computer Science, October 1979, pp. 175–182.

[**Ya-81**] A.C. Yao. "Should Tables Be Sorted?," Journal of the Association for Computing Machinery, Vol 28, No. 3, July 1981, pp. 615–628.