

Naval Research Laboratory

Washington, DC 20375-5000



UNCLASSIFIED

NRL Report 9084

Allocation of Database Files Across Parallel Stores for Efficient Processing of Partial-Match Queries

THOR A. BESTUL AND SUSHIL JAJODIA

*Computer Science and Systems Branch
Information Technology Division*

October 27, 1987

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION / AVAILABILITY OF REPORT			
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE		Approved for public release; distribution unlimited.			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 9084			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laboratory	6b. OFFICE SYMBOL (If applicable) Code 5590	7a. NAME OF MONITORING ORGANIZATION			
6c. ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000			7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Office of Naval Research	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217			10. SOURCE OF FUNDING NUMBERS		
		PROGRAM ELEMENT NO. 63223G	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO. DN155-097
11. TITLE (Include Security Classification) Allocation of Database Files Across Parallel Stores for Efficient Processing of Partial-Match Queries					
12. PERSONAL AUTHOR(S) Bestul, Thor and Jajodia, Sushil					
13a. TYPE OF REPORT Formal	13b. TIME COVERED FROM 1/87 TO 5/87	14. DATE OF REPORT (Year, Month, Day) 1987 October 27	15. PAGE COUNT 13		
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>This report presents the results of research on the allocation of binary cartesian product database files across several stores (such as disks) which are accessible in parallel so that the average time required to perform the retrieval (by using several parallel accesses) of all buckets specified by a partial-match query is made small. The research reveals connections of theoretical interest between this allocation objective and the objective of constructing efficient packings of Hamming spheres in hypercubes. The viewpoint of binary cartesian product file allocation thus produced allowed in our research the creation of allocations for two particular sizes of the set of stores such that the average processing time for partial-match queries was smaller than that of the best allocation technique found in the literature for these same two cases. It also provides a framework in which good allocations for other particular sizes of the set of stores, and possibly for general sets of stores, might be found.</p>					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Thor Bestul			22b. TELEPHONE (Include Area Code) (202) 767-3770	22c. OFFICE SYMBOL Code 5594	

CONTENTS

INTRODUCTION	1
BINARY CARTESIAN PRODUCT FILES	1
PARTIAL-MATCH QUERIES	1
PACKINGS OF HAMMING SPHERES	2
PROBLEM STATEMENT AND SCOPE OF REPORT	2
RESULT 1: SPHERE-PACKING PARTITIONS	4
RESULT 2: SPHERE-PACKINGS AND SUBGROUPS	5
RESULT 3: SPECIAL CASES	5
RESULT 4: PARITY AND BCPF ALLOCATION	6
COMMENTS	6
APPLICATIONS: TWO EFFICIENT BCPF ALLOCATIONS	7
SUMMARY AND CONCLUSIONS	9
REFERENCES	9

ALLOCATION OF DATABASE FILES ACROSS PARALLEL STORES FOR EFFICIENT PROCESSING OF PARTIAL-MATCH QUERIES

INTRODUCTION

Database systems implemented on computers with multiple stores accessible in parallel can benefit from techniques for efficient allocation of binary cartesian product files. In particular, the time necessary to retrieve the set of all buckets satisfying a partial-match query, when averaged over all such queries, can be minimized by the judicious distribution of the buckets across the stores. When retrieval occurs by reading in parallel a bucket from each of the stores and repeating this parallel read until all of the buckets have been read, the limiting factor on retrieval time is the maximum over all the stores of the number of buckets per store. If the buckets are distributed so that buckets likely to be requested by the same partial-match query are evenly distributed among the parallel stores, then this maximum number of retrieved buckets per store is minimized.

This report presents the results of research into the allocation of binary cartesian product files across several parallel stores. The research produced four results that reveal an interesting connection between the efficient allocation of binary cartesian product files using k -bit keys and the efficient packing of Hamming spheres in k -dimensional hypercubes. The work also resulted in the creation of binary cartesian product file (BCPF) allocations for the cases of four and eight stores that empirically outperform the best such allocations found in the literature.

BINARY CARTESIAN PRODUCT FILES

A binary cartesian product file is a collection of "buckets" of data elements; each bucket is referenced by some k -bit binary key. Since the keys are k bits long, the set of keys can be regarded as a subset of the cartesian product of k binary attributes, hence the name binary cartesian product file. This is standard terminology from the literature [1-3]. More general problems of this same type involve the allocation of buckets referenced by keys that are regarded as elements of the cartesian product of several attribute domains, each of which may contain several elements.

PARTIAL-MATCH QUERIES

A k -bit partial-match query is a specification of a subset of the collection of all possible k -bit keys. Specifically, it is a specification that indicates the required values of certain bits, while leaving others undetermined [4-6]. A key is considered to satisfy a partial-match query if the bit values of the key match the bit values of the query in those positions where they are specified; the values of the other bits in the key are not important. We write a partial-match query as a k -bit string of zeros, ones, and asterisks, with the asterisks indicating those bits whose values are undetermined. For example, a partial-match query might be given by 10^*11^* ; this query would be satisfied by the key values 1011100 and 1001101 but not by 1101101.

PACKINGS OF HAMMING SPHERES

Each of the 2^k possible k -bit keys can be thought of as corresponding to one of the vertices of a k -dimensional unit hypercube. Each bit in the key gives the coordinate along one of the k axes of the vertex corresponding to the key. Thus the key consisting of k 0s gives the coordinate vector of a vertex at the origin, and the key consisting of k 1s corresponds to the vertex furthest from the origin.

We define the *Hamming distance* [7] between any two vertices of the k -dimensional unit hypercube as the minimum number of hypercube edges in a path connecting the two vertices. This distance turns out to be equal to the number of 1s in the bit-wise exclusive-or of the coordinate vectors of the two vertices. Thus in a 6-dimensional unit hypercube, the Hamming distance between the vertices with coordinates 100101 and 110010 is 4. We can define about any vertex in the k -dimensional hypercube a *Hamming sphere* [8] of radius r by using the Hamming distance as metric. That is, a Hamming sphere of radius r about a vertex p in the hypercube is the set of all vertices that are within a Hamming distance r of the vertex p .

A *packing* of a set of Hamming spheres in the unit hypercube with *packing radius* r is a collection of pairwise disjoint Hamming spheres of radius r in the hypercube. We commonly refer to the set of *centers* of the pairwise disjoint sphere of radius r as the packing, as long as no confusion arises.

PROBLEM STATEMENT AND SCOPE OF REPORT

This work assumes that the buckets of the binary cartesian product file being allocated do not vary significantly in size from one another and that each bucket is larger than the minimum size block of data that is retrievable in one access from a store. This work furthermore deals only with BCPF allocations in which the number of stores is a power of two. The reason for this stems from theoretical considerations. Specifically, that our constructions of BCPF allocations are based on cosets of subgroups of the natural group representing all k -bit binary keys. Since the order of any such subgroup must divide that of the natural group and since the latter is a power of two, we see that the number of cosets of any such subgroup will also be a power of two. In other words, our particular theoretical approach restricts the numbers of stores about which we can assert results.

Given some k , n , and the set of all k -bit binary keys distributed among 2^n stores, we define the *access time* for a given partial-match query relative to this distribution as the maximum over all the stores of the number of keys in a store that match the query.

For this report, we define an *order l BCPF allocation* as one in which the set of 2^m keys that match an arbitrary partial-match query with m undefined bits will contain at most 2^{m-n+l} elements in each store. For example, Table 1 shows an order 0 BCPF allocation of all 2-bit keys among $2^1 = 2$ stores:

Table 1 — An Order 0
BCPF Allocation

A_0	A_1
00	01
11	10

Note that any partial-match query leaving one bit undefined, such as 0^* , matches only $2^{1-1+0} = 1$ key in each store. The access time for any partial-match query leaving one bit undefined using this

allocation would be 1. Observe that the same is not true of the allocation in Table 2 for the same set of keys over the same number of stores:

Table 2 — An Order 1
BCPF Allocation

A_0	A_1
00	11
01	10

In particular, we see that the partial-match query 1^* matches two keys in store A_1 and none in store A_0 . While reading in parallel one bucket from each store, we see that two reads are required to retrieve both of the buckets whose keys match 1^* . The first read retrieves those buckets with keys 00 and 11, and discards the bucket with key 00. The second read retrieves those buckets with keys 01 and 10, retaining only that bucket with key 10. Naturally, both buckets 11 and 10 cannot be retrieved simultaneously because access from any single store is taken to be serial, given that we assume the buckets to be larger than the store's block size. The second allocation given above is considered an order 1 allocation because any partial-match query leaving m bits undefined will match at most $2^{m-n+1} = 2^{m-1+1} = 2^m$ keys in any given store.

Table 3 shows an order 1 binary cartesian product file allocation of the 4-bit numbers among $2^2 = 4$ stores:

Table 3 — A Larger Order 1
BCPF Allocation

A_0	A_1	A_2	A_3
0000	0001	0010	0011
1100	1101	1110	1111
1010	1011	1000	1001
0110	0111	0100	0101

Note here, for example, that any partial-match query leaving two bits undefined, such as $0^{**}1$, will match at most $2^{2-2+1} = 2$ keys in any store. It is not hard to show that *no* order 0 BCPF allocation can exist for the set of 4-bit binary keys over 4 stores, although this proof is not given here. However, among the order 1 allocations that do exist, some have lower *average* access times than others. This average is computed over all possible partial-match queries, under the assumption that all such queries are equally likely to occur.

An order 0 binary cartesian product file allocation is defined to be *strictly optimal* [1], in the sense that in any distribution of 2^m keys over 2^n stores, some store will contain at least 2^{m-n} keys and we see that an order 0 binary cartesian product file allocation allows retrieving the corresponding 2^m buckets in 2^{m-n} sets of parallel accesses, the minimum possible.

In this report, we first present a proof that the problem of finding an order l BCPF allocation as defined above is equivalent to that of finding a collection of 2^n packings of Hamming spheres in the k -dimensional unit hypercube where the spheres involved are of radius $(n - l + 1)/2$, and given that a sphere-packing is defined by the set of centers of spheres, the 2^n sphere-packings form a partition of the vertices of the hypercube.

Second, we present a proof of how a BCPF allocation can actually be found from a *single* sphere-packing which is also a subgroup of the natural additive group defined on the vertices of the unit hypercube, by showing how a mutually exclusive, collectively exhaustive set of packings can be created from a single packing which is such a subgroup.

For the special cases of two and four stores, we present a proof that the problem of finding an order 0 BCPF allocation is in fact equivalent to that of finding *any* single sphere-packing of 2^{k-1} and 2^{k-2} spheres of radius 1 and $3/2$, respectively, in the k -dimensional unit hypercube.

Having established the previous results, we present a short demonstration of the strict optimality of a simple BCPF allocation technique for the case of two stores.

Finally, we present the two new BCPF allocations for the cases of four and eight stores and the results of our experiments comparing the average access times of the new allocations to those found in the literature for the same cases.

RESULT 1: SPHERE-PACKING PARTITIONS

Given an order l BCPF allocation of the set of all k -bit keys among 2^n stores, we show that each store represents a packing of spheres of radius $(n - l + 1)/2$ in the k -dimensional hypercube. Because we have an order l BCPF allocation, any partial-match query with $m = n - l$ bits undefined can match no more than $2^{m-n+1} = 2^0 = 1$ key in any single store.

Given some store A , let us define each key p in A to be the coordinate of the center of a sphere of radius $(n - l + 1)/2$ in the hypercube (that is, the k bits of the key give the k binary coordinates of the point in the unit hypercube). Now suppose some two of these spheres intersect, and let the centers of these two spheres be the points p and q . Then it must be the case that $|p - q| < n - l + 1$, where $|p - q|$ gives the Hamming distance from p to q . So $|p - q| \leq n - l$, which means that the keys p and q are within $n - l$ bit changes of each other. But then there is some partial-match query with $n - l$ bits undefined such that both p and q match the query. This contradicts our observation above that at most one key from any single store can match such a query. Therefore no two spheres can intersect, and we must have a sphere-packing. So each store gives a sphere-packing and since the stores form a partition of the 2^k keys, the corresponding sphere-packings form a partition of the vertices of the hypercube.

Given a set of 2^n sphere-packings involving spheres of radius $(n - l + 1)/2$ that form a partition of the vertices of the k -dimensional hypercube, we show that each sphere-packing corresponds to a store in such a fashion that the set of stores thus defined creates an order l BCPF allocation.

Let the coordinates of the vertices at the centers of the sphere in each packing give the keys in some store. Let p and q be distinct keys in the same store. Since these two keys correspond to the vertices at the centers of two distinct spheres in a packing of spheres of radius $(n - l + 1)/2$, we know that the Hamming distance between these two vertices must be greater than or equal to $n - l + 1$. Now suppose that p and q both match some partial-match query having $n - l$ bits undefined. Then the keys differ in at most $n - l$ bits, which means that the Hamming distance between the corresponding vertices is less than or equal to $n - l$, which is a contradiction. So we see that any partial-match query in which $n - l$ bits are undefined can match at most 1 key from any given store. We show by induction on i that a partial-match query in which $n - l + i$ bits are undefined can match at most 2^i keys in any given store. Suppose this holds for i_0 , and let $i = i_0 + 1$. The set of all keys that match a given partial-match query with $n - l + i$ bits undefined can be divided into two disjoint subsets: those matching the query with its first undefined bit

set to 0, and those matching the query with its first undefined bit set to 1. Each of these modified queries is one which has $n - l + i - 1 = n - l + i_0$ bits undefined, however, and so each of the corresponding subsets contains at most 2^{i_0} keys. Thus the set itself contains at most $2 \cdot 2^{i_0} = 2^i$ keys.

Thus each sphere-packing, when regarded as a store, has the property that any partial-match query with $m = n - l + i$ bits undefined matches at most $2^i = 2^{m-n+l}$ keys in each store. This, together with the fact that the sphere-packings form a partition of the hypercube vertices, shows us that the corresponding stores together yield an order l BCPF allocation.

RESULT 2: SPHERE-PACKINGS AND SUBGROUPS

The second result involves the natural group of the k -bit binary coordinates of the vertices of the unit hypercube under the operation of bit-wise modulo 2 addition. Let us denote this group by $\langle H, \bullet \rangle$, where H is the set of vertices of the cube, and \bullet denotes the bit-wise modular addition. Suppose that $\langle A, \bullet \rangle$ is a subgroup of $\langle H, \bullet \rangle$ with $|A| = 2^{k-n}$, such that elements of A define a single packing of spheres (of some radius) within the hypercube. Now, let σ be any symmetry operation on the hypercube. It is clear that if A defines a sphere-packing in H , then $\sigma(A)$ does also, because symmetries are distance-preserving permutations.

Now, let h be any element in H , and consider the mapping $\sigma_h(p) = p \bullet h$. We show that σ_h is a symmetry of the hypercube. Let p and q be any two elements of A , and let $|p - q| = d$, where $|p - q|$ again gives the Hamming distance between p and q . Then the number of 1s in $p \bullet q$ is d . Now let us compute $|\sigma_h(p) - \sigma_h(q)| = |p \bullet h - q \bullet h|$. This will be the number of 1s in $(p \bullet h) \bullet (q \bullet h)$. But this latter expression can be rewritten as $(p \bullet q) \bullet (h \bullet h)$, which simplifies to $p \bullet q$. The number of 1s in $p \bullet q$ is d , so $|\sigma_h(p) - \sigma_h(q)|$ is also d . So σ_h is a symmetry, and thus we see that $\sigma_h(A)$ defines a sphere-packing for any h in H .

Notice also, that since $\langle A, \bullet \rangle$ is a subgroup of the abelian group $\langle H, \bullet \rangle$, it is a normal subgroup thereof. Thus $\sigma_h(A)$ for any h in H , being $A \bullet h$, is a coset of A . But the cosets of A form a partition of the hypercube. So we see that the cosets of the normal subgroup A define a collection of sphere-packings that form a partition of the hypercube. Also, since each coset contains 2^{k-n} elements, we see that there are 2^n of them.

Thus we can conclude that any single packing of 2^{k-n} spheres in the k -dimensional unit hypercube which is also a subgroup of the natural group on the hypercube gives rise to a partition consisting of 2^n packings, and hence to a BCPF allocation, as from Result 1 above.

RESULT 3: SPECIAL CASES

For the special case of 2 stores, we show that the problem of finding an order 0 BCPF allocation is equivalent to that of finding *any* single sphere-packing of 2^{k-1} unit Hamming spheres in a k -dimensional hypercube. To do this, it is sufficient to show that given an arbitrary packing of 2^{k-1} unit spheres in a k -dimensional hypercube, we can find a pair of packings of unit spheres that form a partition of the hypercube, as well as the converse.

We prove the converse first. Let A and B be an appropriate pair of packings. Assume without loss of generality that $|A| \geq |B|$. Then $|A| \geq 2^{k-1}$. Thus we can delete spheres from A so that only 2^{k-1} are left, and the resulting set will be a packing containing the required number of unit spheres.

Now suppose that we have a single packing A , consisting of 2^{k-1} unit spheres. Consider the k -dimensional hypercube H as being a pair of $(k-1)$ -dimensional hypercubes H_1 and H_2 , with corresponding vertices joined by unit edges to form H . Consider the symmetry operation σ given by “flipping” H through a plane separating H_1 and H_2 . We apply σ to A to produce the sphere-packing A' . We now show that A and A' form a partition of H . Let p be a point in A , and suppose that p is also in A' . Since p is in A , $\sigma(p)$ is in A' . Thus we have the distinct points p and $\sigma(p)$ both in the sphere-packing A' . This means that these two points, which are distance 1 apart, are the centers of nonintersecting spheres of unit radius, which is a contradiction. Thus no point in A is also in A' . Thus A and A' are disjoint, and since each contains 2^{k-1} points, they must together form a partition of the 2^k points of the k -dimensional hypercube.

The argument for the case of four stores is similar, involving splitting the hypercube into four pieces arranged in a square, and taking advantage of the square’s rotational symmetry of order 4.

RESULT 4: PARITY AND BCPF ALLOCATION

In this section we show how we can use the parities of keys as the basis of an order 0 BCPF allocation in the case of two stores.

By using result 3, we need only find a single packing of 2^{k-1} Hamming spheres of radius 1 in the hypercube to find an order 0 BCPF allocation. It turns out that such a packing can always be found in a k -dimensional unit hypercube. One need only take as centers all those vertices whose binary coordinates have even parity, that is, that contain an even number of bits. There are 2^{k-1} of these, and clearly no two are within distance 1 of each other, because two distinct vertices within distance 1 would have coordinates with different parities, one even and one odd.

Thus we see here that one store should contain all the numbers with even parity, and the other all numbers with odd parity. This result is seen to be a special case of two different BCPF allocation methods which have appeared in the literature [1,2].

COMMENTS

We note that results from research in Communications Theory concerning the construction of error-correcting codes (and hence concerning the packing of Hamming spheres in hypercubes) [7-9] can in principle be used to produce BCPF allocations. For example, consider the (7,4) Hamming code [8]. This represents a packing of 16 Hamming spheres of radius 3/2 in the 7-dimensional hypercube and is also a subgroup of the natural group of the hypercube. This means that the (7,4) Hamming code could be used to define an allocation of the set of 7-bit numbers among $2^{7-4} = 8$ stores such that any partial-match query with m bits undefined will match at most 2^{m-2} numbers in any store, and thus the retrieval for such a query could be using no more than 2^{m-2} sets of parallel accesses.

However, in practice, allocation methods found by other methods seem to result in better average access times than those found by transferring Communications Theory results. This is because in the context of Communications Theory, sphere-packings are sought in which the minimum intercenter distance is as large as possible, while a sphere-packing for use in the construction of a binary cartesian product file allocation needs to go further, and in fact minimize the *number* of intercenter distances that are at the minimum value.

Nevertheless, the hypercube view of BCPF allocation seems to be useful. Heuristics for creating subgroups of the hypercube vertices whose elements are widely scattered have, by using result 2

above, produced BCPF allocation formulas for special cases that have empirically lower average access times than the best we have found in the literature. These formulas are discussed below.

APPLICATIONS: TWO EFFICIENT BCPF ALLOCATIONS

This section gives two new BCPF allocations formulas, one for four stores, and one for eight stores. For their particular cases, both appear to have average access times that are lower than the best technique we have found in the literature [2]. Both of these formulas involve performing a heuristic "scrambling" of the vertices of the k -dimensional hypercube by using transformations such as flips and Gray codes and dividing the resulting cube into a set of equally sized subcubes, one for each store. For 2^n stores, this division is accomplished by projecting the vertices of the k -dimensional cube into those of an n -dimensional cube; all vertices with the same image under the projection are defined to be in the same store. The scrambling operations are such that all key values that ultimately project onto the zero vertex, i.e., those keys which form the kernel of the combined scrambling and projecting operations, form a subgroup of the hypercube vertices; thus the kernel and all of its cosets define a BCPF allocation according to result 2 above.

To visualize the allocation mapping for the case of four stores, imagine that every vertex of the hypercube is labeled with a value and that this value is initially equal to the hypercube address of the vertex. Choose two axes in the hypercube (since $2^2 = 4$ stores); these will be the axes of the "collapsed" hypercube after the projection operation is performed. Also select a third axis orthogonal to both of these for use in a flipping operation. Consider a plane parallel to this third axis and bisecting the cube. The labels of half of the vertices in the hypercube—those vertices with coordinate 1 along the third axis are "flipped," i.e., exchanged with those of the vertices at the plane-symmetric positions across the bisecting plane. Now the (binary reflected) Gray code values of the all the labels in the hypercube are computed, and each label is replaced with its corresponding Gray code value. At this point the scrambling is complete; the labels of all the vertices projecting to the same location after the hypercube is collapsed are defined to be key values in the same store.

To use this scrambling as part of a mapping from key values to store numbers, start with a key value and compute the *inverse* of the scrambling operation defined above. This gives the hypercube address of the vertex at which the label equaling the key value would be found, if the forward scrambling operation were performed. This hypercube address determines which subcube, i.e., which store, the key should be assigned to. So after computing the inverse of the scrambling operation for a key, one need only project the resulting value by selecting from it bits in the appropriate positions, i.e., those corresponding to the first two axes selected. The concatenation of these bits gives the store number (in binary) for the key.

The scrambling operation for eight stores is similar to the one described above and consists of a single flip and an application of the Gray code.

The formulas for the inverse scrambling and projection operations for four and eight stores are given below. To enable the description of the formulas, we introduce some notations. The function $igr(x)$ gives the inverse binary reflected Gray code value for the number x . The function $proj(x, i_1, i_2, \dots, i_p)$ (project) returns a binary number obtained by selecting and concatenating together (in the order specified) the bits from the positions i_1, i_2, \dots, i_p in the number x , where position 0 is the least significant bit. (For example, $proj(101110, 4, 1, 0) = 010$.) We define $tog(x, i_1, i_2, \dots, i_q)$ (toggle) which returns the binary number x but with the bits in positions i_1, i_2, \dots, i_p toggled. (For example, $tog(101110, 4, 1, 0) = 111101$.) Finally, we define x_{msb} as the most significant bit of x .

Then, for four stores, we define $all_{4,k}(x)$, the number from 0 to 3 of the store in which to place the bucket with k -bit key x , as

$$all_{4,k}(x) = \begin{cases} proj(igr(x), k/2, 0) & \text{if } x_{msb} = 0 \\ tog(proj(igr(x), k/2, 0), 0) & \text{otherwise.} \end{cases}$$

In these definitions, integer division is assumed.

For eight stores, we define $all_{8,k}(x)$, the number from 0 to 7 of the store in which to place the bucket with k -bit key x , as

$$all_{8,k}(x) = \begin{cases} proj(igr(x), k/2, (k/2 + 1)/2, 0) & \text{if } x_{msb} = 0 \\ tog(proj(igr(x), k/2, (k/2 + 1)/2, 0), 1) & \text{otherwise.} \end{cases}$$

Tables 4 and 5 show the average access times (assuming equal likelihood among all queries) for various values of k for four and eight stores, both for the method of Du [2] and for the formulas given above.

Table 4 — Comparison of Allocation Techniques for Four Stores

Four Stores — Average Access Time		
k	Du's Method	$all_{4,k}$
3	1.111111	1.037037
4	1.234568	1.185185
5	1.481481	1.382716
6	1.766804	1.711934
7	2.231367	2.136260
8	2.794696	2.750800
9	3.631967	3.550678
10	4.685465	4.656201

Table 5 — Comparison of Allocation Techniques for Eight Stores

Eight Stores — Average Access Time		
k	Du's Method	$all_{8,k}$
4	1.111111	1.012346
5	1.234568	1.135802
6	1.371742	1.283951
7	1.627801	1.558299
8	1.963115	1.799726
9	2.338668	2.292333
10	2.922861	2.828837

SUMMARY AND CONCLUSIONS

An interesting theoretical relationship exists between the allocation of binary cartesian product files, using k -bit keys, across multiple stores accessible in parallel for the efficient processing of partial-match queries, and the efficient packing of Hamming spheres in k -dimensional hypercubes. This hypercube view of BCPF allocation provides a tool for the creation of good BCPF allocations, as has been demonstrated by the construction of two such allocations during this research.

Note that the new formulas given in this report cover four and eight stores, while results from the literature supply more general formulas. These new formulas were created through fairly informal methods involving visualizations of the hypercube, coupled with application of the results presented in this report. Efficient allocations for other numbers of stores, or useful general allocation techniques, might follow from this work.

REFERENCES

1. H.C. Du and J.S. Sobolewski, "Disk Allocation for Cartesian Product Files on Multiple Disk Systems," *ACM Trans. Database Systems* 7(1), 82-101 (1982).
2. H.C. Du, "Disk Allocation Methods for Binary Cartesian Product Files," *BIT* 26, 138-147 (1986).
3. C.C. Chang, R.C.T. Lee, and H.C. Du, "Some Properties of Cartesian Product Files," in *Proc. ACM-SIGMOD 1980*, pp. 157-168.
4. A.V. Aho and J.D. Ullman, "Optimal Partial-Match Retrieval When Fields Are Independently Specified," *ACM Trans. Database Systems* 4(2), 168-179 (1979).
5. W.A. Burkhard, "Hashing and Trie Algorithms for Partial-Match Retrieval," *ACM Trans. Database Systems*, 1(2), 175-187 (1976).
6. R.L. Rivest, "Partial-Match Retrieval Algorithms," *SIAM J. Comput.* 15(1), 19-50 (1976).
7. Vera Pless, *Introduction to the Theory of Error-Correcting Codes* (John Wiley and Sons, Inc., New York, 1982).
8. T.M. Thompson, *From Error-Correcting Codes Through Sphere Packings to Simple Groups* (Mathematical Association of America, Washington, DC, 1983).
9. W.W. Peterson and E.J. Weldon, Jr., *Error-Correcting Codes* (MIT Press, Cambridge, Mass., 1972).