Partitioning of State-Space with the Sum-of-Digits and Hashing with Reduced Number of Address Collision

ELEAZAR JIMENEZ SERRANO
Department of Automotive Science
Kyushu University
744 Motooka, Nishi-ku, Fukuoka
JAPAN
eleazar.jimenez.serrano@kyudai.jp

Abstract: - We present a method for partitioning the state-space based on the sum-of-digits in order to conduct parallel state-space exploration and hashing. The method has a configuration for storing the universe of the state-space using multiple hash tables generating a reduced number of address collisions. This paper presents the partitioning result, the number of address collisions generated with the proposed hashing method, the efficiency in reducing the number of address collisions, an estimation of the number of address collisions, and the probability of the unfitness of our estimation for the size of our arrangement of multiple hash tables with respect to the real sizes.

Key-Words: - Partition, Hash, State-space, Petri nets

1 Introduction

The full exploration of a state-space is the storing in memory of all possible succession of states existing in a dynamical system represented with an explicit directed graph. In computing terms, the exploration demands a large amount of memory and time to accomplish. These are usually ignored because the entire state-space is sometimes just an implicit representation using a mathematical model of computation, like state machines and Petri nets.

Algorithms for parallel exploration of state-spaces through a network of distributed-memory multiprocessors based on the message passing paradigm are becoming important in order to speed-up the computing [1].

Graph partitioning algorithms [2-4] are used for distributing workloads of parallel computations, but usable only with explicit directed graphs. When the graph representing the state-space to explore is unknown, an approach is to suggest an a priori workload distribution based on a heuristically defined partition of the probable explicit graph.

For the first problem targeted in this paper, the objective of partitioning is to evenly distribute the number of vertices in the directed graph with edges linking to other vertices belonging to different partitions, balancing with this the workload.

In this paper, we propose using the function of sum-of-digits (SOD) of keys as our partitioning function. However, the domain of the partitioning function is not the unknown state-space but its deducible universe of keys.

Keys are the numeric descriptors of all the vertices of a directed graph, and a partitioning function determines membership of a key in a partition. The sizes of the partitions of the universe generated by the function are certainly not uniform. The sizes of their distribution appear like a normal probability distribution (NPD) [5-8]. Then by the principle of centrality, in the progression of the state-space exploration, the edges may be linked to vertices having keys with SOD following a NPD.

The previous contention may lead to a partitioning with a significant reduction of cross transitions [9]. For the parallel state-space exploration of any unknown explicit graph, this may be interpreted as an advantage for reducing network communication and hence reducing exploration time. Additionally, we show how to evenly assign workload to a number of parallel computing-memory nodes in a network.

For the second problem, the one of storing in the explored state-space in a reduced memory space, we present a set of hash functions where each function has as domain a partition of the universe of keys. The hash functions are based on the traditional modulo operation but applied in an unorthodox way, and an additional key conversion using the casting-out-nines operation. With this conversion we can ignore the constrain of mapping as evenly as possible the output range, i.e. to get uniformly

distributed hash addresses, and still produce a reduced number of address collisions.

We demonstrate its effectiveness in obtaining a reduced number of hash address collisions in 5 universes of keys, by means of exhaustive analysis and estimation.

Finally, we present the probability associated with the unfitness of our proposal of multiple hash tables with normal sizes.

Chapter 2 includes the necessary and sufficient background regarding state-space. Chapter 3 is about partitioning of the universe of keys. Chapter 4 covers storing with hashing and chapter 5 explains our scheme of multiple hash tables as a result of the partitioning. Chapter 6 contains the probability of unfitness and the increased number of address collisions. In the last section we provide a resume of the paper.

2 Background

In this section, first we describe the fundaments of the state-space and its implicit representation with Petri nets. For additional details about Petri nets the reader is addressed to [10-11].

2.1 Petri Nets

The implicit directed graph representing a dynamic system is denoted by a Petri net (PN). An ordinary PN is a tuple G_{PN}=(P, T, A, B, Q), where P is a finite non-empty set of numbered and ordered *i* places, T is a finite non-empty set of numbered and ordered *j* transitions, A is the set of directed arcs connecting places to transitions, B the set of directed arcs connecting transitions to places, and Q is a capacity function for the places mapping P → N={1, 2, ...}.

Tokens are black dots that exist only in the places, and by putting tokens in places we describe states of a PN system, where s_0 is the initial state. States are usually called markings. The function s is called a marking function, mapping P → N. A PN with initial marking is called a PN system. We say that a place *p* is marked when s(*p*) > 0. A marking *s* is represented with a vector $\vec{s} \in N^i$.

A PN is also mathematically represented with the pre and post-incident matrices $\vec{A}$ and $\vec{B}$ respectively having both *i* rows and *j* columns, with values of [A(p_i, t_j)] and [B(p_i, t_j)] respectively.

To determine the succession of states, or markings, in a PN system with initial marking s_0, is throughout complete exploration of every succession, or firing of enabled transition. Each marking can be generated with the state transition function $\vec{s}_n = \vec{s}_c+(\vec{B}-\vec{A})\times\vec{\sigma}$. In the formula $\vec{s}_c$ is the current marking, $\vec{s}_n$ is the next marking and $\vec{\sigma}$ is a firing count vector representing the number of times every transition has fired.

Occurrence of every single transition is carried out for complete exploration of the state-space of a PN system. Occurrence of solely concurrent transitions could lead to incomplete exploration of the state-space.

The set of markings of a PN system is called the reachability space and usually denoted with R(s_0) and its cardinality is expressed with |R(s_0)|.

2.1.1 Reachability graph

All possible markings and all possible firings of a PN system are explicitly drawn in a way they represent a directed graph of R(s_0).

Given a PN system (G_{PN}, s_0), G_R stands for the reachability graph of R(s_0), defined as the succession of markings generated by the PN system, according to every single transition occurrence. The graph is the tuple G_R=(V, E) where V is the set of vertices (markings) and E the set of edges (transition firing) connecting pair-wise vertices. Self-loops are not permitted.

Now, vertices can be numerically represented by state descriptors called keys. It is common practice and more convenient to work with the set of keys, denoted with K, than directly with the markings of R(s_0).

Different methods exist to define keys. In the simplest injective method, a key is generated from the concatenation of markings in all places of a PN, where a marking in a place uses a number of digits according to its tokens capacity, with explicit left-side zeros. In this paper, we present our solution for these sets of keys.

3 Partitioning

Here we present the first problem formulation towards achieving a parallel exploration of the state-space. It is the partitioning of the state-space across multiple computing-memory nodes.

A *x*-way partition of a reachability graph G_R of R(s_0) is the mapping of vertices into *x* subsets, i.e. V → [*1*, *2*, ..., *x*], where

$$\bigcup x = V \quad with\, x_y \cap x_z = \varepsilon \quad and\, y \neq z\,.$$

Given a reachability graph G_R of R(s_0), the graph partitioning problem seeks to find a *x*-way partition in which each subset contains roughly the same

number of vertices and the number of cut edges is minimized.

A cut edge is a subset of E with endpoint vertices belonging to different partitions. These are the subsets of edges whose removal breaks the reachability graph into disjoint subsets of vertices.

Each subset represents data to be assigned to a single computing-memory node (the workload) and cut edges represent the network communication among nodes required by the distribution. Thus, the solution to the problem attempts to find a distribution that balances the computation done by each node while minimizing the total inter-node communication.

Now, the way how the number of edges per vertex might fluctuate in the reachability graph during the course of an exploration is unknown. Moreover, it is also unknown how any a priory partitioning of the graph might end up distributed. Therefore, since the size and structure of the state-space is unknown, probably the best alternative may be to work with the deducible finite universe of states.

3.1 Keys in Partitions

We extend the problem of finding a partitioning for K as the problem of finding a partitioning for the universe of keys Ω, where $K \subseteq \Omega$.

We define for the universe of keys Ω, the minimum key k_{min} is 0 and the maximum key k_{max} is an integer with n 9's, where n is decided by the analyst. E.g., for a state-space with a key k_{max} of 4 digits, its keys are in a universe of keys from 0 to 9999. The cardinality of |K| might be ignored, but the cardinality of $|\Omega|$ is $k_{max}+1$.

All keys in a partition having the same SOD can be exhaustively constructed with the theory of partitions and compositions [12].

Let us enumerate the digits of a key as α_1, α_2, ...; e.g. for a key k=47293, α_1=4, α_2=7, α_3=2, α_4=9 and α_5=3. The length of a key (number of digits) is the function d(k).

A partition of an integer key is an n-length SOD $\alpha_1+\alpha_2+...+\alpha_n$ such that the sum of α's is SOD(k), and all α's are ordered from largest to smallest; e.g. the partitions of 3 are 3, 21 and 111.

A composition is similar to a partition except for the order of α's does not matter. E.g., the compositions of 3 are 3, 12, 21 and 111.

A weak composition is a composition allowing α's in the sequence to be zero. Adding zeroes at the left side of the leftmost non-zero digit of a sequence is not considered to define a different composition. If the length of the composition is not bounded by a number n then it can be infinite. E.g., the weak compositions of 3 with n=4 are 3, 12, 21, 30, 102, 111, 120, 201, 210, 300, 1002, 1011, 1020, 1101, 1110, 1200, 2001, 2010, 2100 and 3000.

Keys existing in a partition having the same SOD are obtained with the function

$$k = 9c + s. \quad (1)$$

In (1), s is the SOD and c is a set of strictly monotonically increasing non-negative and non-linearly separated integers. E.g., for the partition of s=2, the set of integers c = {0, 1, 2, 11, 12, 22} produces the ordered keys k = {2, 11, 20, 101, 110, 200}.

3.2 Partitioning Size

Let us define the variable X of partitions with a and b the minimal and maximal partition values in X respectively. In this paper, the scheme how we want to put keys in partitions follows a probability density function $f(x)$, i.e. for the variable X of partitions

$$p[a \le X \le b] = \int_a^b f(x)dx = 1. \quad (2)$$

In (2), $f(x)$ is the probability density function and (a, b) represents the range of X.

We implemented a partitioning scheme for the universe of keys Ω following the NPD generated from the SOD of the keys. We selected the SOD function because our intuition is that the way how events in a state-space modify the components in the state vector may obey the principle of centrality, i.e.

$$X \sim N(\hat{\mu}, \hat{\sigma}). \quad (3)$$

In (3), if the variable X is the SOD(k) of all keys in Ω, its distribution appears like a NPD with estimated mean and sigma of $k_{max}/2$ and $k_{max}/6$ respectively.

3.3 Partitioning Function

A partitioning function Pt determines membership in a partition. The partition function Pt is the mapping of keys in K to partitions in X, producing the distribution of keys to partitions according to $f(x)$, i.e.

$$Pt : K \to X. \quad (4)$$

In (4), the keys k_{min} and k_{max} are mapped to the values of a and b respectively.

Thus, we defined our partitioning function as the sum-of-digits of a key, i.e.

$$Pt(k) = SOD(k). \quad (5)$$

3.4 Workload Distribution

Traditionally, workload distribution assigns one processor-memory node to one partition for the exploration and hashing of only that partition. In our case it is different.

The maximal number of partitions in the universe of keys may limit the maximal number of nodes to work with. But also, the maximal number of nodes in the network may limit the distribution of partitions among them.

We assign a number of partitions to a processor-memory node in a manner to distribute the same number of keys in the universe among all nodes. In this way, processor-memory nodes may explore and hash different number of partitions, but the partitions have a number of keys from the universe of keys as evenly as possible to the other nodes [9].

E.g., for an undefined state-space, for its universe of keys Ω where k_{max} is 9999, the partition corresponds to 37 groups of SOD, and the sizes of the 37 hash tables may follow a NPD.

For a target computing architecture of 4 nodes, the figure 1 presents a proposed scheme for parallel computing.

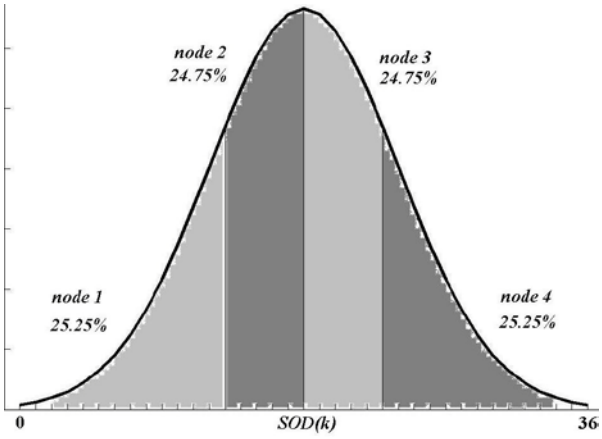

Fig. 1. Distribution of partitions to nodes.

The density each node holds is oriented to be as balanced as possible. In the example, it is around 25%.

Furthermore, the hash tables in the nodes, numerically identified along the number of partitions, may be assigned in contiguous order to the nodes to take advantage of the principle of centrality and reduce network communication as much as possible.

4 Hashing

One method to store the information of R(s_0) is to use its set K to generate an index for a static array. This arrangement is called a hash table [13].

Usually, when the size of the array is smaller than the key k_{max} in K, or |K| is ignored, a non-injective hash function h is used to accommodate the information in the hash table. The function is a map of the set K to the number of slots in the tables, i.e.

$$h : K \to \{0,1,2,...,m-1\}. \quad (6)$$

In (6), m is the number of slots, also called the size of the space of hash addresses in the hash table.

4.1 Hash Functions

Hash functions are required to have the following properties [14]:

- They must be easy to compute.
- They must distribute the hash addresses uniformly.

The second property is to distribute the addresses as evenly as possible and avoid a large number of hash address collisions, meaning the mapping of 2 or more keys to the same address.

The simplest hash function to generate uniformly distributed addresses is by arithmetic operation of division, where m is the number of slots in the table, k divides m and the residue is the respective address h(k). The address is also the result of the modulo operation in (7) when using a computer programming language.

$$h(k) = \text{mod}(m,k). \quad (7)$$

It is relevant to mention that mod(m,0) is 0 and the case mod(0,k) is not considered due to there are no tables with size of 0.

5 Multiple Hash Tables

If we partition the universe of keys Ω according to Pt, then we can generate a number of hash tables according to X with sizes that might follow $f(x)$. This scheme of multiple hash tables is suitable for a computing scenario of parallel exploration of the state-space.

Now, each partition contains a subset of keys from Ω. In this paper, we already analysed the sets of keys based on the proposed partition scheme. Now we present our set of hash functions for each partition.

5.1 Size of Multiple Hash Tables

Given the maximal number of slots we can have, i.e. the size of the universe of keys or k_{max}, it is straightforward the allocation of slots to the tables by multiplying every integer value in the range (a, b) of X by the number of slots, and rounding the number up (or down) to the next higher integer, i.e.

$$\lceil m \times N(x;\hat{\mu},\hat{\sigma}) \rceil \quad \forall x \; in \; (a,b). \qquad (8)$$

Focusing only in the direct space assigned for hashing, in a exploration scenario using shared-memory, in theory our approach may be more memory efficient and flexible than a single hash table if tables are only created in real-time as they are required. In a distributed-memory setting, this is interpreted as a reduced number of nodes in the network engaged in the exploration and hashing of the state-space. Moreover, reducing the number of slots may result in a reduced number of hash addresses for certain scenarios described later in this paper.

5.2 Set of Hash Functions

We designed our hash functions based on the modulo operation, which is the simplest function. Here, we provide an explanation and justification for the functions presented next.

Given that keys are in partitions according to their SOD, the evidence shows that within a partition with size $m' < m$, there are 2 subsets of keys; the largest one usually has the keys with a numerical value greater than m', while the smallest one has all the other keys. Thus, we decided to use the modulo operation in an unorthodox way. By inverting the values in the modulo operation we may avoid repeating many times the same result for the hash function, and produce less address collisions, i.e.

$$h(k) = \mathrm{mod}(k, m'). \qquad (9)$$

For the universes of keys, those with keys of 3 to 7 digits, for all the partitions and their respective hash tables with normally distributed sizes, we present evidence indicating that the number of address collisions generated with (9) may be in general less than with (7) for the same number of slots. Table 1 shows the total number of address collisions and figures 2 to 6 show the distribution of all address collisions according to the SOD.

Table 1. Number of Address Collisions

Digits	(7)	(9)
3	931	588
4	9585	5678
5	96541	56298
6	968534	712918
7	9702283	6571856

Furthermore, we also observed that an additional conversion of keys before hashing may reduce even more the number of address collisions. Given that keys are in partitions following the formula in (1), a casting-out-nines key conversion with the formula in (10) reduces the range of the sets of keys, making the sets more compact.

$$k' = \frac{k - SOD(k)}{9}. \qquad (10)$$

Generating the hash address with the converted key produces additional improvements in the reduction of address collisions. The table 2 shows the number of address collisions, just for sets of keys with 3 to 7 digits.

Table 2. Number of Address Collisions

Digits	(7)&(10)	(9)&(10)
3	635	212
4	7096	2526
5	73640	28406
6	754966	268938
7	7703678	2903740

5.3 Lower Address Collisions

In the previous section we analyzed the number of address collisions in multiple hash tables with a total number of slots equal to the size of the universe of keys.

Now, for multiple hash tables still with normalized sizes, but a number of slots less than the size of the set, by common sense the lower the number of slots the higher the number of hash address collisions might be.

However, we observed that tables with a size which is multiple of 3 is in most of the cases the best size for a hash table to obtain the lowest number of address collisions [7]. The table 3 shows the new number of address collisions, where the reduction exist only for the inverted modulo operation and converted keys.

Table 3. Number of Address Collisions

Digits	(7)&(10)	(9)&(10)
3	637	204
4	7100	2424
5	73642	26976
6	754972	271102
7	7703683	2900222

5.4 Derivation to the Approximation of the Number of Collisions

We have not provided complete evidence to generalize the benefit of our proposal; rather, we presented limited results indicating a trend towards reducing in general the number of address collisions.

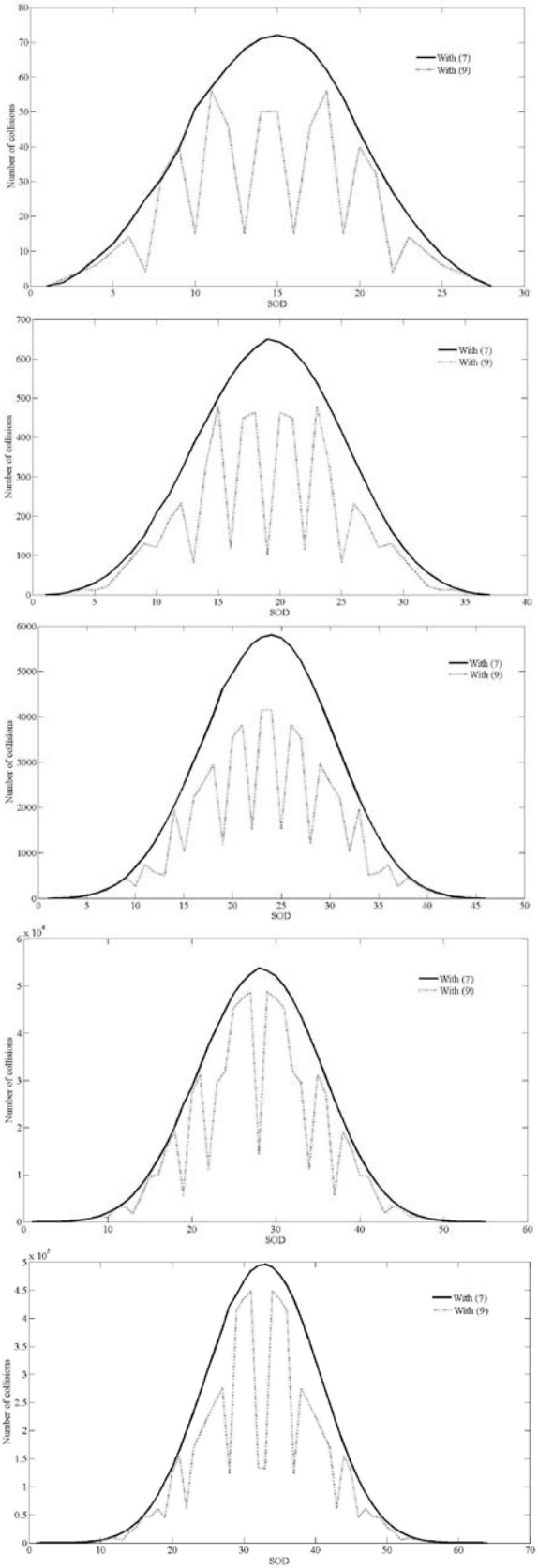

Fig. 2 to 6. Number of collisions for 3 to 7 digits keys.

In this section we offer additional evidence of the benefit of our proposal. Neither the previous results nor the following ones are conclusive, but we believe they reinforce our proofs at some degree.

In the following, it will be useful to have a general construction of the maximum and minimum keys within a partition of keys with same SOD and fixed number of digits *n*. We will denote such set by $x_{SOD,n}$. Notice that given a fixed *n*, in order for $x_{SOD,n}$ to be non-empty we must have that the condition $0 \leq SOD \leq 9n$ because $9n$ is the maximum SOD possible for *n* digits.

The minimum and maximum keys in a set $x_{SOD,n}$ are obtained by

$$k_{\min} = \sum_{j=0}^{k_c-1} 9\times 10^j + R\times 10^{k_c} \ . \qquad (11)$$

and

$$k_{\max} = R\times 10^{n-k_c} + \sum_{j=n-k_c+1}^{n} 9\times 10^j \ . \qquad (12)$$

respectively, where R=(SOD mod 9) and k_c=(SOD–R)/9.

Let us denote by *m'* the number of keys in a set $x_{SOD,n}$. We know that, by definition of the maximum and minimum keys, these *m'* keys are distributed within the range $[k_{min}, k_{max}]$. Then it is obvious to see that the proportion of integer numbers within such range is

$$\rho = \frac{m'}{k_{\max} - k_{\min}} . \qquad (13)$$

representing the density of keys in $x_{SOD,n}$ for a range $[k_{min}, k_{max}]$. Thus, given a random integer number within the range, the probability that it is a key of $x_{SOD,n}$ is ρ.

Traditionally, we know that the modulo operation *k* mod *m* repeats its values for any domain greater than *m*, and that the values will repeat γ times if the domain over which it is applied is γ times the size of *m*. Since the number of blocks of size *m* in the range $[k_{min}, k_{max}]$ is ρ^{-1}, each value can be repeated at most ρ^{-1}-1 times, as repetitions are only considered from a value above one.

So it is that the number of keys of $x_{SOD,n}$ in a block of size *m'* is $\rho m'$, and the maximum number of collisions of addresses for all keys in $x_{SOD,n}$ is

$$(\rho^{-1} - 1)\rho m' = (1-\rho)m'$$
$$= (1 - \frac{m'}{k_{\max} - k_{\min}})m'. \qquad (14)$$

This represents the maximum number of collisions of addresses in the hash table for the modulo operation as the hash function, denoted here by $C_{SOD,n}$.

For the case of keys with the proposed casting-out-nines conversion (CO9), the analysis is analogous. The range for these converted keys is defined with [k'_{min}, k'_{max}] and is order-preserving.

Following the treatment in (14), the maximum number of repetitions for all converted keys is

$$C'_{SOD,n} = (1 - \frac{9m'}{k_{\max} - k_{\min}})m' . \qquad (15)$$

We observe that $C_{SOD,n}$ and $C'_{SOD,n}$ are upper-bounds for the number of collisions to be expected in the respective hash tables, since we have neglected the effect of the non-homogeneous distribution of the keys (or converted keys) in the corresponding range.

Now, with simple inequality rules we can show that $C_{SOD,n} > C'_{SOD,n}$, meaning that we can expect more address collisions without the CO9 key conversion.

The inequality may hold true if we consider that the lack of homogeneity in the distribution of the keys in a group affects both parameters in the same proportion.

The figures 7 to 11 show the estimations compared with the real number of collisions, including the adjustment from table 1 derived from the key conversion.

6 Address Collisions and Unfitness

We analyzed in the past section the number of address collisions under the conditions of using the modulo operation inversely, applying CO9 to the key, and a hash table with size equal to or less than the number of keys in the partition.

In this final section we present the probability of having a larger number of hash address collision in terms of the unfitness of the arrangement of multiple hash tables with sizes following the (estimated) NPD with respect to the (real) number of keys.

Considering a scenario of multiple hash tables with normally distributed sizes, for universes of keys, those with keys of 3 to 7 digits, and all the partitions according to their SOD, the figures 12 to 16 show the real and estimated distribution of the hash tables and their sizes.

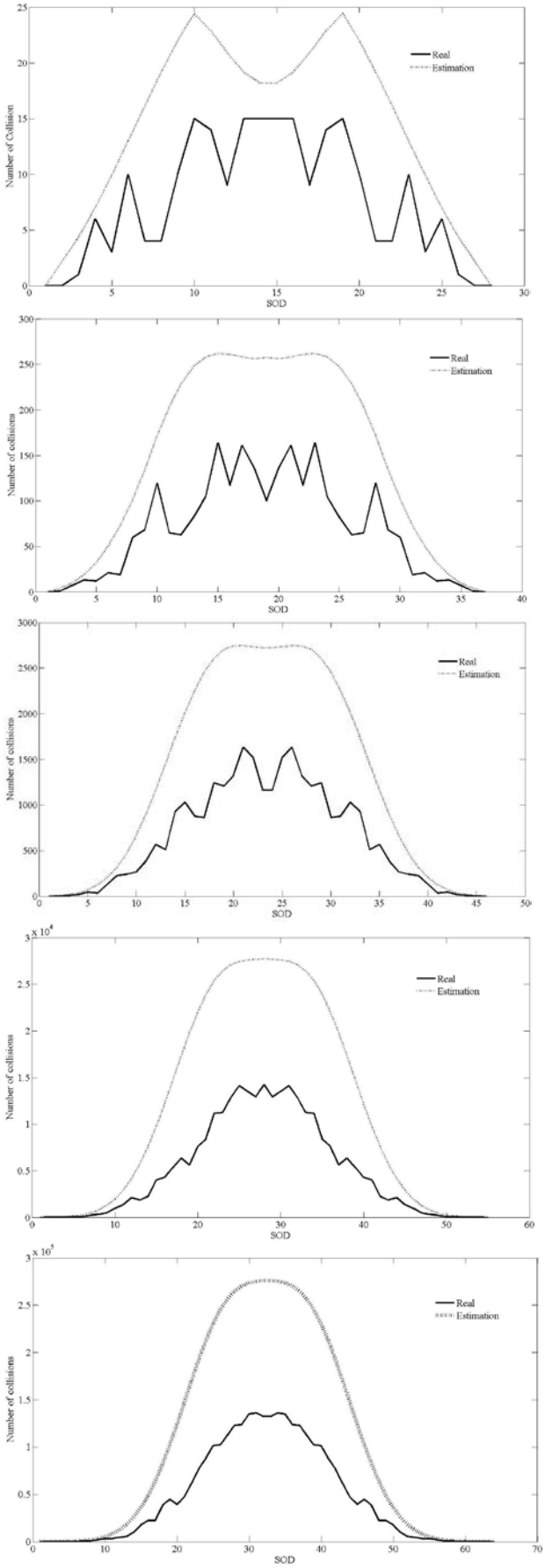

Fig. 7 to 11. Estimated number of collisions for 3 to 7 digits keys.

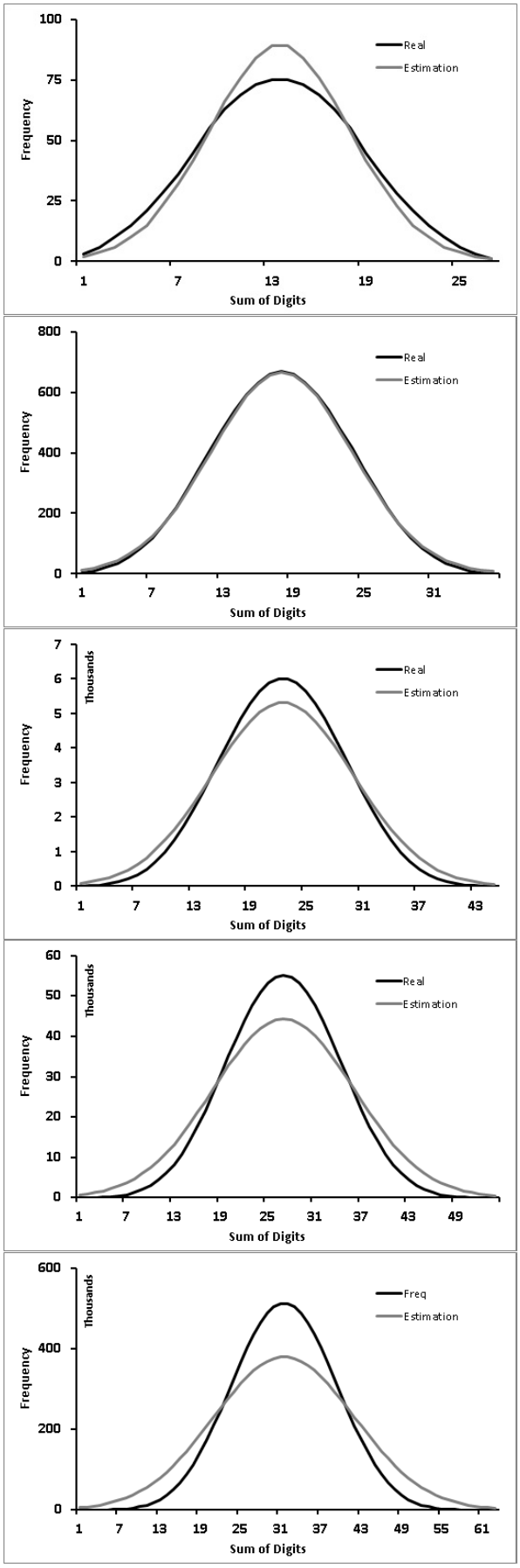

Fig. 12 to 16. Curve fitting of universe of keys for 3 to 7 digits keys.

6.1 Probability of More Collisions

Our procedure to calculate the unfitness was, first fitting the real number of keys in each partition to a NPD, then obtaining the mean and standard deviation of the fitting, and finally calculating the difference of the accumulated probabilities for the case when the estimation of the size of hash tables is less than the fitting. Fig. 17 and 18 graphically explains this difference.

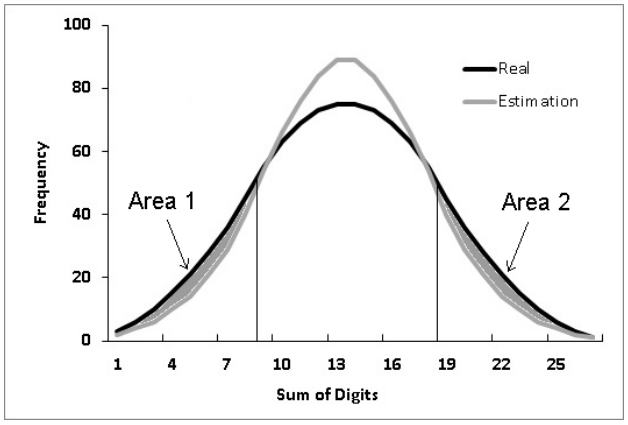

Fig. 17. Location of area 1 and 2 of accumulated probability

For the universe of keys of 3 digits and all the partitions according to their SOD, the figure 17 shows the area 1 and 2 of the difference of accumulated probability are located on the sides of the curve. However, for a universe of 4 digits or more, the accumulated probability is located at equidistant points from the middle of the curve. The figure 18 shows the area 3 of the difference of accumulated probability for the universe of keys of 6 digits.

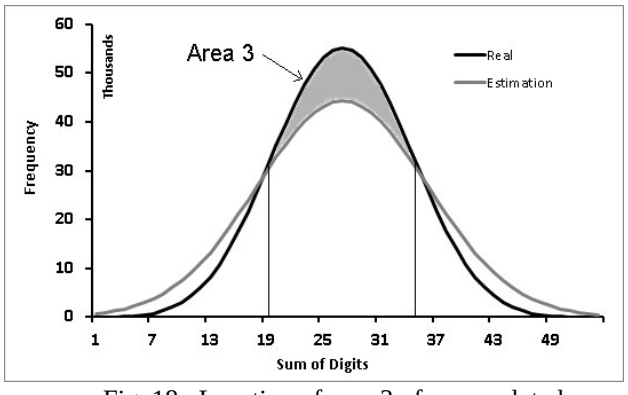

Fig. 18. Location of area 3 of accumulated probability

The values obtained for the mean and standard deviation from fitting the real observations to the NPD are shown in table 4.

Table 4. Parameters of Real Data

Digits	μ	Std. Err.	σ	Std. Err.
3	13.5135	0.1570	4.9615	0.1111
4	18.0018	0.0574	5.7423	0.0406
5	22.5002	0.0203	6.4223	0.0144
6	27.0000	0.0070	7.0356	0.0050
7	31.5000	0.0024	7.5993	0.0017

The values of the mean and standard deviation used for the estimation of the sizes of the hash tables are shown in table 5 for the case when corrections to the standard deviation as suggested in [8] are not considered.

Table 5. Parameters of Estimated Data

Digits	SOD(k_{max})	μ	σ
3	27	13.5	2.3
4	36	18.0	3.0
5	45	22.5	3.8
6	54	27.0	4.5
7	63	31.5	5.3

From our procedure to define the normalized size of the multiple hash tables, the probability of having a larger number of address collision in the 5 cases under analysis (universes of 3 to 7 digits keys) are shown in table 6.

Table 6. Probability of Address Collision

Digits	Prob. (%)
3	4.22
4	2.09
5	7.46
6	11.69
7	15.49

For the final evaluation of our method, although this probability is not the only one related to the probability of no hash address collision, it is necessary for identifying the error in its calculation.

7 Conclusion

In this paper we presented strong evidence showing that, in the partitioning and exploration of state-spaces, the hashing of the states using multiple tables with normalized size and special hash functions with inverted values in the modulo operation and CO9 key conversion, presents a reduced number of address collision, compared with the modulo operation alone.

First, we demonstrated that partitioning the universe of keys in terms of the SOD of the keys, there may be a way to assign partitions to networked nodes such that each node may not hold in their local memory the same number of hash tables, but the number of key the nodes can hash with their tables may be uniform.

Second, we exhibited the number of address collisions for universes of 3 to 7 digits keys, in multiple tables with normalized size, and total number of slots equal to or less than the size of the universe. The number of address collision for our proposal is less than, or in the worst case, equal to the number of collisions with the modulo operation alone.

Next, we showed upper-bound estimators of the number of address collisions for the modulo operation hashing keys with and without CO9 conversion. The number of address collisions, in universes of 3 to 7 digits keys, for our proposed CO9 key conversion is less than without it.

Finally, we presented the probability associated with the unfitness of our proposal for the normalized sizes of multiple hash tables with respect to the real number of keys in each partition of the universe of keys. This probability is important for the further definition of the probability of no address collisions. Also, due to our arrangement of tables allows a reduction in the used memory when exploring state-spaces of skewed sets of keys, understanding this unfitness may be also useful for a more exact evaluation of the required memory.

References:

[1] Ulrich Stern and David L. Dill, Parallelizing the murφ verifier, *Computer Aided Verification*, Springer-Heidelberg, LNCS vol. 1254, 1997, pp. 256-267.

[2] Horst D. Simon, Partitioning of unstructured problems for parallel processing, *Computing Systems in Engineering*, vol. 2, issue 2-3, 1991, pp. 135-148.

[3] V. Venkatakrishnan, Horst D. Simon and Timothy J. Barth, A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids, *The Journal of Supercomputing*, vol. 6, 1992, pp. 117-137.

[4] Bradford L. Chamberlain, *Graph partitioning algorithms for distributing workloads of parallel computations*, University of Washington, Computer Science & Engineering, 1998.

[5] Eleazar Jiménez Serrano, Multiple hash tables for skewed sets of integer keys, *Proc. of the*

12th WSEAS Int. Conf. on Applied Informatics and Communications, Istanbul, Turkey, 2012, pp. 241-246.

[6] V.s., On the design of special hash functions for multiple hash tables, *Proc. of the 9th IEEE Int. Conf. on Electrical Engineering, Computing Science and Automatic Control*, Mexico city, Mexico, 2012, pp. 252-256.

[7] Eleazar Jiménez Serrano and Fermin Franco Medrano, On the Collision Rate of Key Conversion prior Modulo for Multiple Hash Tables of Any Size, *Proc. of the 12th WSEAS Int. Conf. on Applications of Computer Engineering*, Cambridge, MA, USA, 2013, pp.61-68.

[8] Eleazar Jiménez Serrano, Partial exploration of state spaces and hypothesis test for unsuccessful search, *Electrical Engineering and Intelligent Systems*, Springer, LNEE 130, chapter 1, 2013, pp. 1-13.

[9] Eleazar Jiménez Serrano, Parallel exploration of state-space with reduced cross transitions partitioning, *Proc. of the 10th IEEE Int. Conf. on Electrical Engineering, Computing Science and Automatic Control*, Mexico city, Mexico, (submitted for publication).

[10] T. Murata, Petri nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, Vol.77, 1989, pp. 541-580.

[11] J. L. Peterson, *Petri Net Theory and the Modelling of Systems*, Prentice-Hall, 1981.

[12] Percy A. MacMahon, *Collected Papers Volume 1 – Combinatorics*, The MIT press, USA, 1978.

[13] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, 2nd ed., vol. 3. Addison-Wesley Longman, 1998.

[14] Matthias Kuntz and Kai Lampka, Probabilistic methods in state space analysis, *Validation of Stochastic Systems*, Springer-Verlag, Germany, LNCS vol. 2925, 2004, pp. 339-383.