EFFICIENT FUNCTION EVALUATIONS WITH LOOKUP TABLES FOR STRUCTURED MATRIX OPERATIONS[†]

K. Sobti, L. Deng, C. Chakrabarti
Dept. of Electrical Engineering
Arizona State University
email: {ksobti, ldeng2, chaitali}@asu.edu

N. Pitsianis, X. Sun
Dept. of Computer Science
Duke University
email: {nikos, xiaobai}@cs.duke.edu

J. Kim, P. Mangalagiri, K. Irick, M. Kandemir, V. Narayanan
Dept. of Computer Science and Engineering
Pennsylvania State University
email: {jskim, mangalag, irick, kandemir, vijay}@cse.psu.edu

ABSTRACT

A hardware efficient approach is introduced for elementary function evaluations in certain structured matrix computations. It is a comprehensive approach that utilizes lookup tables for compactness, employs interpolations with adders and multipliers for their adaptivity to non-tabulated values and, more distinctively, exploits the function properties and the matrix structures to claim better control over numerical dynamic ranges. We demonstrate the effectiveness of the approach with simulation and synthesis results on evaluating, in particular, the cosine function, the exponential function and the zero-order Bessel function of the first kind.

Keywords: digital arithmetic, elementary function evaluation, table lookup, structured matrices, geometric tiling

1. INTRODUCTION

Matrix operations are fundamental to many signal processing applications and problems pertaining to scientific and engineering studies. In many cases, matrix element evaluations comprise the bulk of the computations. This is because the numerical evaluation of elementary functions or transcendental functions that characterize these matrix elements is usually expensive. Moreover the number of these computations could be very large - sometimes as large as 10^6 for the N-body interaction problems such as those in particle dynamics in astrophysics, molecular modeling and quantum chemistry. Hence efficient techniques are required for evaluation of these functions.

Conventionally, elementary function evaluation has been done in software. However, the overhead in latency associated with such implementations is often too large to meet the high speed and high throughput requirements of many applications. The current trend is to implement these functions in hardware using lookup table (LUT) based schemes. Such schemes are usually area-efficient since the memory is much denser than logic modules such as adders and multipliers [1].

In many scientific applications, stringent accuracy requirements have to be met while still using fast lookup schemes and small lookup tables. Higher-accuracy evaluation typically requires a finer sampling for table entries and hence a larger table. In fact, the table size grows exponentially with the increase in the accuracy requirements. For instance, a direct lookup implementation of bessel J_0 function for the input dynamic range of $[0, 100]$, requires about 4 MB of table memory, which is larger than typical memory capacities in present day FPGAs. Although recent advances in VLSI technology have made memory cheaper, the memory space utilization still remains one of the most pressing challenges in system design. This is because the increase in data space requirements of applications far exceeds the increase in the memory capacities. Thus innovative techniques need to be developed for satisfying the conflicting requirements of large accuracy and small table size.

This paper introduces a new approach for designing efficient lookup table based schemes for elementary function evaluation. In particular, we have used the techniques for efficiently evaluating trigonometric functions, square-root extraction, logarithmic exponential function, and more complex functions such as the Bessel functions. The approach enables us to satisfy the accuracy constraints as well as the architectural constraints of area and memory size. The proposed approach first designs a LUT scheme that satisfies the given accuracy requirement by choosing linear interpolation and suitable sampling density. It then modifies this implementation to meet the area and memory constraints by using a combination of the following two techniques. The first technique trades-off memory size with logic resources by varying the

[†] This work is supported by a grant from DARPA W911NF-05-1-0248

interpolation degree. The second technique, known as geometric tiling, first clusters the input data into tiles according to their dynamic range and then operates on each tile sequentially. This technique results in smaller sized lookup-tables at the expense of a pre-processing cost. The final scheme is guaranteed to meet all the three constraints of area, memory size and accuracy.

The rest of the paper is organized as below. In section 2 we start with a basic description of structured matrices followed by existing approaches for reduction in memory size. In section 3 we describe our approach to exploit the geometric structures of the kernel functions in matrix evaluations. We then describe the proposed approach to design efficient LUT scheme. We demonstrate in section 4 the effectiveness of our approach. Finally, in section 5, we provide conclusions.

2. PRELIMINARIES

2.1. Structured matrices: geometric structures

Matrices used in scientific applications have a certain structure. These matrices are typically a linear or linearized convolutional transform on two discrete, finitely bounded data sets. Let $\kappa(t,s)$ be the kernel function of a convolutional transform. Let $\mathbf{T}$ and $\mathbf{S}$ denote, respectively, the target set and source set of sample points at which the kernel function is sampled and evaluated, in $\mathbf{T},\mathbf{S} \subset \mathbf{R}^3$. The discrete transform becomes

$$g(\mathbf{t}_i) = \sum_{\mathbf{s}_j \in \mathbf{S}} \kappa(\mathbf{t}_i - \mathbf{s}_j) \cdot f(\mathbf{s}_j), \qquad \mathbf{t}_i \in \mathbf{T}. \tag{1}$$

We give a few examples for transform kernel functions. For the gravitational interaction, the kernel function is essentially $1/\|\mathbf{t}-\mathbf{s}\|$, i.e., the interaction between a particle located at $\mathbf{t}$ and a particle at $\mathbf{s}$ is reciprocally proportional to their Euclidean distance. The square-root extraction is therefore needed. This kernel function is used to describe Newtonian particle interactions from molecular dynamics to galaxy dynamics as well as electro-static interactions [2]. Kernel functions of the form $e^{ik\|\mathbf{t}-\mathbf{s}\|}/\|\mathbf{t}-\mathbf{s}\|$ appears in 3D image processing based on computational electromagnetics. This requires the evaluation of cosine and sine functions, in addition to the square-root extraction. This function may be transformed into the zero-order Bessel functions of the first kind for certain 2D image processing problems [3]. These functions are smooth except that $e^{ik\|\mathbf{t}-\mathbf{s}\|}/\|\mathbf{t}-\mathbf{s}\|$ has a singular point at $\mathbf{t}=\mathbf{s}$, is translation invariant, and vanishes at infinity.

The equations in (1) can be represented in a matrix form

$$\mathbf{g}(\mathbf{T}) = \mathbf{K}(\mathbf{T},\mathbf{S}) \cdot \mathbf{f}(\mathbf{S}),$$

where, the vector $f(\mathbf{S})$ is a given function defined on the source set $\mathbf{S}$, the matrix $\mathbf{K}(\mathbf{T},\mathbf{S})$ is the aggregation of point-point interaction over the target set and the source set and represents the cluster-cluster interaction. The vector $\mathbf{g}(\mathbf{T})$ is the transformed function over the target set $\mathbf{T}$. Notice that the matrix rows and columns are *indexed* with corresponding target and source locations $\mathbf{t}_i$ and $\mathbf{s}_j$ respectively. Section 3 describes the techniques that exploit the geometric properties of structured matrices for efficient computation of interaction function $\kappa(\mathbf{t_i},\mathbf{s_j})$.

2.2. Reduction in lookup table size

We introduce first certain pre-processing, interpolation schemes and post-processing steps which help in reduction of the size of the lookup tables.

In preprocessing, the dynamic range of the input can be reduced by *folding* or/and *scaling* [4] [5]. In folding, when applicable, the entire dynamic range of the input is mapped onto a much smaller range. For trigonometric functions, for example, the folding can be based on the trigonometric identities such as

$$\begin{aligned} &\sin(2n\pi+u) = \sin(u), \quad \cos(2n\pi+u) = \cos(u); \\ &\sin(2u) = 2\sin(u)\cos(u), \quad \cos(2u) = 2\cos^2(u) - 1. \end{aligned} \tag{2}$$

For the exponential function, we may use $\alpha^x = \alpha^{x/2} \cdot \alpha^{x/2}$ to reduce the initial range by a factor of 2, at the expense of a shift in the input and a multiplication at the output. For some other functions such as power functions, one may use multiplicative change in the input variable, i.e., variable scaling to reduce the dynamic range,

$$x^\beta = c^\beta (x/c)^\beta. \tag{3}$$

The power functions include in particular the division ($\beta = -1$) and the square root ($\beta = 1/2$). The folding and scaling can be performed in either hardware or software. Equation (3) already includes the scaling recovery in the post-processing, where c^β is pre-selected and pre-evaluated. Both folding, scaling, their reverse operations, and the associated overhead shall be taken into consideration in the design and development of table construction and lookup,

Between the preprocessing and postprocessing, the table lookups may be tightly coupled with interpolations to adaptively meet accuracy requirements without increasing the table size. A simple example is the approximation of a function f at a non-entry value x by a linear interpolation of two tabulated neighbor values $f(x_1)$ and $f(x_2)$ such that $x_1 < x < x_2$,

$$f(x) \approx f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) \tag{4}$$

If higher accuracy is required, one may use a higher-degree interpolation scheme, such as the Taylor interpolation scheme. According to Taylor's theorem [6] [7], a function f at x can be approximated by a Taylor polynomial of degree n,

$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k, \tag{5}$$

where x_0 indicates a sampling point, the function is assumed smooth enough at x_0 to warrant the existence of the derivatives $f^{(k)}(x)$ at x_0. The approximation error is described as follows,

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}, \tag{6}$$

where $\xi(x)$ is between x_0 and x. One determines from the error estimates the minimal approximation degree n, taking into consideration the sample space between table entries and the behavior of the higher order derivatives. The formation of the Taylor polynomial in equation 5 requires not only tabulating $f(x_0)$ but also the derivatives at x_0 up to the n order.

3. EFFICIENT LUT SCHEME

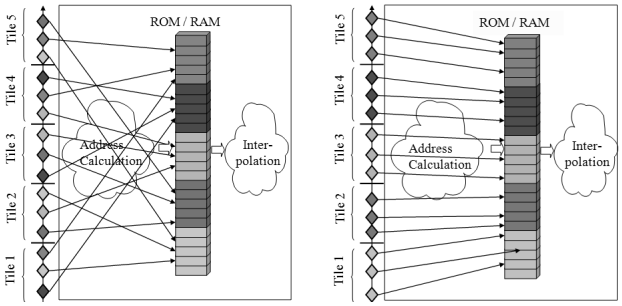
We introduce in this section our approach for designing efficient lookup table based schemes for evaluating structured matrices. We first present a new matrix partition or tiling scheme, called geometric tiling, followed by its use in designing efficient look up schemes.

3.1. Geometric tiling

Conventional matrix partitioning technique, or plain tiling, as it is often called, partitions the matrix into smaller blocks. It is typically used to enhance cache performance while evaluating large matrices. We introduce a new tiling scheme called geometric tiling, which partitions the matrices based on their geometric and numeric structures. It also enhances cache performance but has other advantages which make it suitable for LUT based schemes.

First we start with a very simplistic, one dimensional view of geometric tiling. Assume that the input data lies between the interval [a,b]. If the input is processed in the order in which it is received, it results in random accesses to the lookup table as shown in Figure 1(a). Next consider a clustering scheme where the input dynamic range interval is subdivided into n sub-intervals or tiles $[(a, t_1), (t_1, t_2), ..., (t_j, t_{j+1}), ..., (t_{n-1}, b)]$ and the input data is clustered into these tiles. If the tiles are processed one by one, the lookup table access is localized, see Figure 1(b). This fundamental property of geometric tiling to limit the distributed range of numerical values per tile and the dynamic range of the distance between tiles, helps in reducing computational resource consumption as described later in this section.

In practice, the concept of geometric data clustering can easily be extended to matrices as follows. Imagine that all the particles, the sources and targets, are bounded in a box B (see Figure 2). Specifically, the box B is subdivided into eight non-overlapping smaller boxes of equal size, B_0, B_1, B_2, up to B_7. Each and every particle falls into one of the smaller boxes. Denote by $\mathbf{T}_i$ and $\mathbf{S}_i$ the respective clusters of



(a) Plain Tiling (b) Geometric Tiling

Fig. 1. Random and localized lookup table accesses in case of (a) plain tiling and (b) geometric tiling.

the target particles and source particles in box B_i. This particle clustering scheme induces a virtual partition of the interaction matrix into 8×8 tiles (sub-matrices). The (i, j) tile is $\mathbf{K}(\mathbf{T}_i, \mathbf{S}_j)$, the interaction matrix between clusters $\mathbf{T}_i$ and $\mathbf{S}_j$. This tile is empty if one of the clusters is empty. For the case illustrated in Figure 2, the 8×8 virtual partition renders a 2×2 partition of the interaction matrix (with the zero partitions omitted) where the particle-particle interactions with similar values are grouped into one partition.

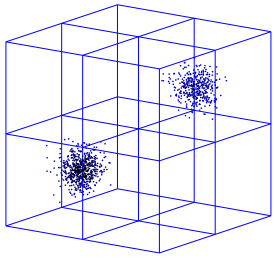


Fig. 2. Two sample clusters located in two sub-divided regions

As a result of geometric tiling, structured matrices exhibit the following properties -

1. Reduction of dynamic range at the input of function evaluation. Computing pairwise difference $(\mathbf{t} - \mathbf{s})$ and distance $\|\mathbf{t} - \mathbf{s}\|$ is common to the class of convolutional kernel functions we are interested in. If we partition the matrix into two tiles on each dimension as shown in Figure 2, the difference $\mathbf{t} - \mathbf{s}$ between any source-target pair $(\mathbf{t}, \mathbf{s})$ in the same sub-box is less than, or at most equal to, half of the maximum particle-to-particle difference in the big box, in each and every component in the magnitude.

2. Reduction of dynamic range at the output of function evaluation. Consider Figure 3, where target and source tiles are shown in $\hat{\imath}$ and $\hat{\jmath}$ dimensions respectively and the height of the tile (in $\hat{k}$ dimension) is used to represent the interaction strength of the source-target pairs. We observe that for each tile, the interaction strength is at the same level. This is be-

cause the interaction functions considered here, such as those in section 2.1, are smooth and the input to these functions has limited numerical range in each tile. Conventional tiling scheme would not show similar pattern because particles with near and far distances are scattered in each tile.

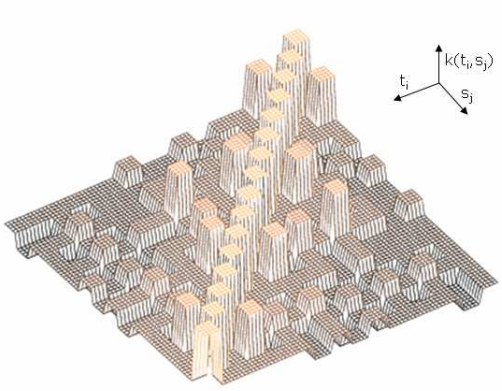


Fig. 3. The numerical landscape of a large matrix that has been partitioned using geometric tiling.

3.2. Impact of geometric tiling on lookup schemes

The impact of geometric tiling on development of efficient LUT schemes is summarized as follows.

1. The reduced dynamic range at the input of kernel evaluation helps to achieve the same accuracy with smaller number of lookup table entries though the sampling density is the same. Further, geometric tiling scheme can also be utilized to reduce the hardware cost. Consider the situation where the memory for a LUT is very limited. In this case, the numerical range of the input parameters in successive cycles should be kept small. Geometric tiling enables the entire range of the input to be segmented into small tiles that helps to meet the spatial constraint as well as the numerical accuracy requirement.

2. The restricted dynamic range of the inputs inside each tile and across tiles, increases the data locality at the output as well. If geometric tiling is utilized, when we apply lookup table for the function evaluation, only one segment of the table needs to be loaded for processing a tile. In fact, this segment may be loaded once and used by many tiles with the same range. Specifically, we organize the tiles according to their numerical ranges. We then traverse the tiles and accumulate the submatrix-vector products in increasing order of their levels. For instance in Figure 3, the tiles on the main diagonal are at the same level and are traversed in successive cycles. Such a traversal can reduce the variation in numerical range of the output across tiles. At the hardware level, this reduces the bit-width requirements of registers that are used for storing outputs and the intermediate values. It also reduces the bit-width requirements for accumulation logic required for performing matrix-vector operations. These reductions have a direct impact on the area and power requirements of the design.

3. Certain preprocessing and postprocessing techniques, such as input scaling and input folding, can be applied efficiently after using geometric tiling. The input folding parameter n and input scaling parameter β (see equations 2 & 3) need not be calculated for every particle-particle interaction, as in the case of plain tiling. Instead they are computed only once for each tile and reused for all particle-particle interactions within a tile.

Overhead: Geometric tiling requires pre-processing of input data to bin them into tiles based on their numerical value. Fortunately, this overhead is linearly proportional to the total number of the particles, which is smaller than the total number of the interaction entries by an order of magnitude. Additionally, if the reloading of LUT segment and function evaluation do not overlap, then it may have an adverse impact on the latency of the design. However, the use of dual buffers for LUT memory can easily avoid this at the expense of additional memory and associated control circuitry.

3.3. Proposed LUT scheme

We now present our approach for designing an efficient LUT based kernel evaluation scheme, given the area and memory constraints of the target architecture.

Algorithm 1 Efficient LUT scheme

Input: Kernel function, user specifications such as input dynamic range, accuracy, area and memory size constraints.

Output: LUT scheme that satisfies the area, memory and accuracy specifications.

Procedure:

1: Plain tiling based kernel implementation using linear interpolation such that accuracy constraints are satisfied.
2: **if** Memory size is not sufficient **then**
3: Increase the degree of interpolation and decrease the sampling density *ensuring* that accuracy constraints are not violated.
4: **end if**
5: **if** Area constraints are not met **then**
6: Decrease the degree of interpolation and increase the sampling density *ensuring* that accuracy constraints are not violated.
7: Apply Geometric tiling by choosing proper number of tiles.
8: **end if**

The above algorithm first implements the kernel function using linear interpolation such that the accuracy constraints are met. Next it checks whether the current implementation meets the memory constraints or not. If not, it modifies the current implementation by iteratively increasing the degree of interpolation and decreasing the sampling density such that

memory constraints are just met. The modified implementation is then checked against the area constraints. If it fails to meet the area constraints, the degree of interpolation is iteratively reduced and the sampling density increased such that area constraints are just met. The resulting implementation may now violate memory constraints. This is now corrected by applying geometric tiling scheme. Suitable number of tiles are chosen to segment the table memory such that each segment meets the required memory constraints. The implementation so obtained meets all the area, memory and accuracy constraints required by the target application.

4. EXPERIMENTAL RESULTS

We present in this section, the primary simulation and synthesis results to demonstrate the effect of the proposed approach. We show the experimental results for three functions $\cos(x)$, e^{-x} and $\mathrm{J}_0(x)$.

The setup for the experiments is as follows. The functions are evaluated using the method described in section 3 and mapped to Xilinx FPGA. The synthesis environment consists of Synplify Pro 6.2 and Xilinx ISE 8.2i. The target platform is a Xilinx Virtex2Pro-100 device. Each XC2VP100 device consists of 444 blocks of Block SelectRAM, 18Kb each in size, so the total available BRAM is 7,992Kb or 999KB. After synthesis and physical mapping, the power is estimated using XPower. In the tables that follow,

- the number of operations represent computations, expressed in terms of additions and multiplications for address calculation and interpolation.
- area is reported in terms of number of occupied slices and 18×18bit multipliers
- memory size is expressed both as number of table entries to store coefficients in the terms in equation 5 and as actual size of the memory
- power consumption is measured assuming 125 MHz clock and 12.5% activity factor for all the inputs

The accuracy result is computed in Matlab. The error values, reported in the tables below, represent the largest deviation between the fixed point implementation and its double precision counterpart. The distributed range of the particles in the target set $\mathbf{T}$ and the source set $\mathbf{S}$, for $\cos(x)$ and $\mathrm{J}_0(x)$ is $(0, 100)$, and for e^{-x} is $(0, 10)$. Each set contains 5K particles. The sample points for generating the lookup table entries are uniformly distributed in their specific ranges.

4.1. Memory size reduction by interpolation

First we present results to demonstrate how we can trade-off memory size by utilizing additional logic resources. This is done by varying the degree of Taylor interpolation polynomial.

We use Bessel function as a test example. The results are shown in Table 1. Each individual configuration is labeled TI_n where n stands for the interpolation degree.

Table 1. Results of varying degrees of LUT based interpolation implementation of $\mathrm{J}_0(x)$

Configuration		TI_0	TI_1	TI_2	TI_8
Degree n		0	1	2	8
# operation	Add +	3	6	8	27
	Mult *	1	3	6	31
LUT	# entries	1.6e+6	12802	2403	864
	size	4.3MB	34.4kB	6.45kB	2.32kB
Occupied slices		N.A.	382	716	4416
18×18bit multipliers		N.A.	12	24	132
Power(mW)@125MHz		N.A.	804	866	1664
Max.abs.err		1.83e-5	1.65e-5	1.77e-5	2.86e-5

The direct lookup scheme TI_0 uses a table whose size is so large that it cannot simply fit into a single XC2VP100 device. The large table size was due to a very high sampling density that was required in order to achieve the high accuracy. The Taylor polynomial interpolation scheme with degree 1 (TI_1) significantly reduces the table size, though it requires additional multipliers and adders. If we further increase the degree of Taylor polynomial, for example configurations TI_2 and TI_8, the table size can be further reduced. However, this requires additional computational resources, thereby increasing both area and power. In fact, very high degree of interpolation (as in TI_8) is not beneficial since the reduction in memory is limited but the cost in other resources such as area and power consumption is significantly high.

4.2. Memory size reduction due to proposed approach

Next we present the simulation and synthesis results that show the reduction in memory size and other logic resources, due to the combined use of geometric tiling and LUT based interpolation schemes. We use GT and PT to denote geometric tiling and plain tiling respectively.

Table 2. LUT implementations of $\mathrm{J}_0(x)$ with geometric tiling

Function $\mathrm{J}_0(x)$		Plain		Geometric
Configuration		PT_1	PT_2	GT_3
Taylor degree n		8	2	2
# operation	Add +	27	8	8
	Mult *	31	6	6
Lookup table	# entries	864	2403	282
	size	2.32kB	6.45kB	0.76kB
Occupied slices		4416	716	766
18×18bit multipliers		132	24	24
Power(mW) @125MHz		1664	866	894
Max.abs.err		2.86e-5	1.77e-5	3.24e-6

In Table 2 we present the implementation results for the

$J_0(x)$ function. The accuracy requirement of the application is of the order of 10^{-5}. Assume that the target architecture imposes the constraints of maximum available size of 3KB for the table size and 1K for the number of occupied slices. Configuration TI_1, corresponding to the degree 1 of Taylor polynomial interpolation scheme (see Table 1), achieves the accuracy constraints but violates the memory constraint. Step 2 of the Algorithm 1 requires that the interpolation degree be increased which yields the configuration PT_1 in Table 2. However such high degree of interpolation results in an implementation which violates the area constraint. The degree of interpolation is lowered but the accuracy requirement demands a large table size that again violates the memory constraint, see PT_2. So geometric tiling based scheme GT_3 is applied. It uses lower degree of interpolation (degree 3 in this table) and thus fewer slices and multipliers, fewer number of lookup table entries and so smaller memory. In GT_3, the numerical range of each particle set has been partitioned into 50 tiles of equal size. Note that the choice of the number of tiles is a function of the available memory, the required accuracy and the acceptable overhead for loading segments of the lookup table into local memory.

Next, we present two more functions to show that the geometric tiling based scheme always results in reduced hardware cost with better or at least comparable accuracy. We did not include the implementation results that incorporate folding and scaling due to space limitations.

Table 3. LUT implementations of cos(x) with geometric tiling

Function cos(x)		Geometric	Plain	
Configuration		GT_4	PT_5	PT_6
Taylor degree n		3	3	9
# operation	Add +	7	7	12
	Mult *	5	5	17
Lookup table	# entries	34	1602	68
	size	0.10kB	4.31kB	0.20kB
Occupied slices		650	593	2422
18×18bit multipliers		20	20	76
Power(mW) @125MHz		851	834	1311
Max.abs.err		4.06e-5	4.21e-5	5.21e-5

In Table 3 we present the results for the evaluation of the cos(x) function. While using geometric tiling, the numerical range is partitioned into 100 equal-sized segments. The comparison between GT_4 and PT_5 shows again the advantage of the LUT scheme with geometric tiling. GT_4 has the same accuracy, similar area and power, but significantly smaller memory size. PT_6 also has small memory, but has lower accuracy compared to GT_4 along with larger area and power consumption.

Finally, we present in Table 4 the results for evaluating the e^{-x} function. The exponential function is smooth at the specified input dynamic range, making it especially suitable for interpolation schemes. In this case, PT_8 does as well as

Table 4. LUT implementations of e^{-x} with geometric tiling

Function e^{-x}		Geometric	Plain	
Configuration		GT_7	PT_8	PT_9
Taylor degree n		3	3	9
# operation	Add +	7	7	13
	Mult *	5	5	17
Lookup table	# entries	9	81	9
	size	25B	223B	25B
Occupied slices		579	538	1862
18×18bit multipliers		20	20	68
Power(mW) @125MHz		843	820	1113
Max.abs.err		4.00e-5	4.06e-5	4.45e-5

GT_7 with respect to accuracy, area and power, at the cost of 9 times larger lookup table size. PT_9 has the same lookup table size and similar accuracy as GT_7 but costs 30% more power and 3 times larger area.

5. CONCLUSION

This paper introduces a new comprehensive approach for designing efficient lookup table based schemes for function evaluations in certain structured matrix computations. Specifically, the existing lookup table based schemes are combined with a new data clustering scheme called geometric tiling. This integrated approach has been demonstrated to successfully satisfy the accuracy, area and memory size requirements for the evaluation of trigonometric functions, exponential function and the Bessel function. It brings forth great opportunities for performance enhancement and optimization in the algorithm-architecture co-design space.

6. REFERENCES

[1] B. Parhami (editor), *Computer arithmetic algorithms and hardware designs*, Oxford University Press, 2000.

[2] B. Thidé, *Electromagnetic Field Theory*, Upsilon Books, 1997.

[3] W. C. Chew, *Waves and Field in Inhomogeneous Media*, Van Nostrand Reinhold, New York, 1990.

[4] J. Cao, B. W. Y. Wei, and J. Cheng, "High-performance architectures for elementary function generation," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, Jun 2001, pp. 136–144.

[5] P.T.P. Tang, "Table lookup algorithms for elementary functions and their error analysis," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, Jun 1991, pp. 232–236.

[6] J.W. Hauser and C.N. Purdy, "Approximating functions for embedded and asic applications," in *Proceedings of the 44th IEEE Midwest Symposium on Circuits and Systems*, Aug 2001, vol. 1, pp. 478–481.

[7] F. B. Hildebrandt, *Introduction to Numerical Analysis*, McGraw-Hill Book Company, 1956.