Chapter 24

The Effects of Polynomial Degrees
On The Hierarchical Segmentation Method

Dong-U Lee and Wayne Luk[1], John D. Villasenor[2],
Peter Y.K. Cheung[3],

[1] *Department of Computing*
Imperial College, London, United Kingdom
{dong.lee, w.luk}@ic.ac.uk

[2] *Electrical Engineering Department*
University of California, Los Angeles, USA
villa@icsl.ucla.edu

[3] *Department of EEE Imperial College, London, United Kingdom*
p.cheung@ic.ac.uk

Abstract This chapter presents the effects of polynomial degrees on the hierarchical segmentation method (HSM) for approximating functions. HSM uses a novel hierarchy of uniform segments and segments with size varying by powers of two. This scheme enables us to approximate non-linear regions of a function particularly well. The degrees of the polynomials play an important role when approximating functions with HSM: the higher the degree, the fewer segments are needed to meet the same error requirement. However, higher degree polynomials require more multipliers and adders, leading to higher circuit complexity and more delay. Hence, there is a tradeoff between table size, circuit complexity and delay. We explore these tradeoffs with four functions: $\sqrt{-\log(x)}$, $x\log(x)$, a high order rational function and $\cos(\pi x/2)$. We present results for polynomials up to the fifth order for various operand sizes between 8 and 24 bits.

Introduction

In this chapter, we explore the effects of polynomial degrees when approximating functions with the hierarchical segmentation method (HSM) presented

P. Lysaght and W. Rosenstiel (eds.),
New Algorithms, Architectures and Applications for Reconfigurable Computing, 301–313.

in [8]. HSM is based on piecewise polynomial approximations, and uses a novel hierarchy of uniform segments and segments with size varying by powers of two. This scheme enables us to use segmentations that take the non-linear regions of a function into account, resulting in significantly fewer segments than the traditional uniform segmentation.

The degrees of the polynomials play an important role when approximating functions with HSM: the higher the degree, the fewer segments are needed to meet the same error requirement. However, higher degree polynomials require more multipliers and adders, leading to higher circuit complexity and more delay. Hence, there is a tradeoff between table size, circuit complexity and delay. We explore these tradeoffs with the following four non-linear compound and elementary functions:

$$f_1 = \sqrt{-\log(x)} \tag{24.1}$$

$$f_2 = x \log(x) \tag{24.2}$$

$$f_3 = \frac{0.0004x + 0.0002}{x^4 - 1.96x^3 + 1.348x^2 - 0.378x + 0.0373} \tag{24.3}$$

$$f_4 = \cos(\pi x/2) \tag{24.4}$$

where the input x is an n-bit number over $[0, 1)$ of the form $0.x_{n-1}..x_0$. Note that the functions f_1 and f_2 cannot be computed for $x = 0$, therefore we approximate these functions over $(0, 1)$ and generate an exception when $x = 0$. In this work, we implement an n-bit in, n-bit out system. However, the position of the decimal (or binary) point in the input and output formats can be different in order to maximize the precision that can be described. We present results for polynomials up to fifth order for various operand sizes between 8 and 24 bits. We have opted for faithful rounding [3], meaning that the results are rounded to the nearest or next nearest, i.e. they are accurate within 1 ulp (unit in the last place).

The principal contribution of this chapter is a review of HSM and a quantitative analysis of the effects of the polynomial degrees. The novelties of our work include:

- a scheme for piecewise polynomial approximations with a hierarchy of segments;
- quantitative analysis of the effects of using different order polynomials;
- evaluation with four compound functions;
- hardware implementation of the proposed method.

The rest of this chapter is organized as follows: Section 24.1 covers background material and previous work. Section 24.2 reviews HSM. Section 24.3 explores how the polynomial degrees affect HSM at various operand sizes. Section 24.4 discusses evaluation and results, and Section 24.5 offers conclusions and future work.

24.1 Background

Many applications including digital signal processing, computer graphics and scientific computing require the evaluation of mathematical functions. Applications that do not require high precision, often employ direct table look-up methods. However, this can be impractical for precisions higher than a few bits, because the size of the table is exponential in the input size.

Recently, table look-up and addition methods have attracted significant attention. Table look-up and addition methods use two or more parallel table look-ups followed by multi-operand addition. Such methods include the symmetric bipartite table method (SBTM) [14] and the symmetric table addition method (STAM) [15]. These methods exploit the symmetry of the Taylor approximations and leading zeros in the table coefficients to reduce the look-up table size. Although these methods yield significant improvements in table size over direct look-up techniques, they can be inefficient for functions that are highly non-linear.

Piecewise polynomial approximation [11] which is the method we use in this chapter, involves approximating a continuous function f with one or more polynomials p of degree d on a closed interval $[a, b]$. The polynomials are of the form

$$p(x) = c_d x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0 \tag{24.5}$$

and with Horner's rule, this becomes

$$p(x) = ((c_d x + c_{d-1})x + \cdots)x + c_0 \tag{24.6}$$

where x is the input. The aim is to minimize the distance $\|p - f\|$. Our work is based on minimax polynomial approximations, which involve minimizing the maximum absolute error. The distance for minimax approximations is:

$$\|p - f\|_\infty = \max_{a \le x \le b} |f(x) - p(x)|. \tag{24.7}$$

Piñeiro et al. [13] divide the input interval into several uniform segments. For each segment, they store the second degree minimax polynomial approximation coefficients, and accumulate the partial terms in a fused accumulation tree. Such approximations using uniform segments [12], [1] are suitable for functions with linear regions, but are inefficient for non-linear functions, especially when the function varies exponentially. It is desirable to choose the boundaries of the segments to cater for the non-linearities of the function. Highly non-linear regions may need smaller segments than linear regions. This approach minimizes the amount of storage required to approximate the function, leading to more compact and efficient designs. HSM uses a hierarchy of uniform segments (US) and powers of two segments (P2S), that is segments with the size varying by increasing or decreasing powers of two.

Similar approaches to HSM have been proposed for the approximation of the non-linear functions in logarithmic number systems. Henkel [4] divides the interval into four arbitrary placed segments based on the non-linearity of the function. The address for a given input is approximated by another function that approximates the segment number for an input. This method only works if the number of segments is small and the desired accuracy is low. Also, the function for approximating the segment addresses is non-linear, so in effect the problem has been moved into a different domain. Coleman et al. [2] divide the input interval into seven P2S that decrease by powers of two, and employ constant numbers of US nested inside each P2S, which we call P2S(US). Lewis [9] divides the interval into US that vary by multiples of three, and each US has variable numbers of uniform segments nested inside, which we call US(US). However, in both cases the choice of inner and outer segment numbers is done manually, and a more efficient segmentation could be achieved using our segmentation scheme.

24.2 The Hierarchical Segmentation Method

Let f be a continuous function on $[a, b]$, and let an integer $m \geq 2$ specify the number of contiguous segments into which $[a, b]$ has been partitioned: $a = u_0 \leq u_1 \leq \ldots \leq u_k = b$. Let d be a non-negative integer and let P_i denote the set of functions p_i whose polynomials are of degree less or equal to d. For $i = 1, \ldots, m$, define

$$h_i(u_{i-1}, u_i) = \min_{p_i \in P_i} \max_{u_{i-1} \leq x \leq u_i} |f(x) - p_i(x)|. \tag{24.8}$$

Let $e_{max} = e_{max}(u) = \max_{1 \leq i \leq m} h_i(u_{i-1}, u_i)$. The segmented minimax approximation problem is that of minimizing e_{max} over all partitions u of $[a, b]$. If the error norm is a non-decreasing function of the length of the interval of approximation, the function to be approximated is continuous and the goal is to minimize the maximum error norm on each interval, then a balanced error solution is optimal. The term "balanced error" means that the error norms on each interval are equal [5].

We presented an algorithm to find these optimum boundaries in [8], based on binary search. This algorithm is implemented in MATLAB, where the function to be approximated f, the interval of approximation $[a, b]$, the degree d of the polynomial approximations and the number of segments m are given as inputs. The program outputs the segment boundaries $u_{1..m-1}$ and the maximum absolute error e_{max}. Figure 24.1 shows the optimum segment boundaries for 16 and 24-bit second order approximations to f_1. We observe that f_1 needs more segments in the regions near zero and one, i.e. smaller segments are needed when the curve has rapidly changing non-linear regions.

In the ideal case, one would use these optimum boundaries to approximate the functions. However, from a hardware implementation point of view, this can

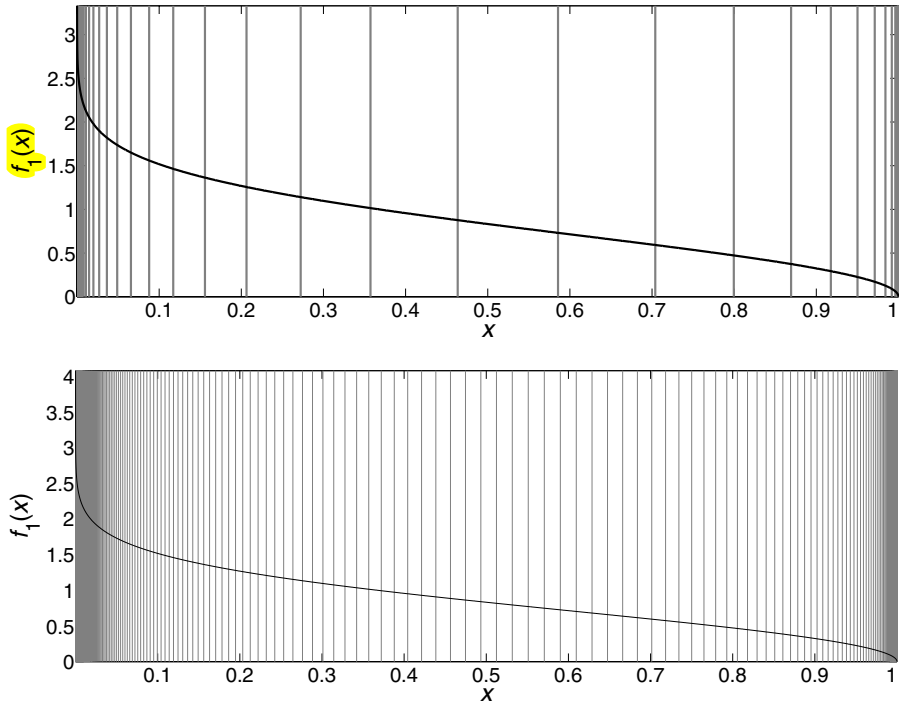

Figure 24.1. Optimum locations of the segments for 16 and 24-bit operands and second order approximations to f_1, resulting in 4, operands and second order approximations to f_1, resulting in 44 and 358 segments respectively.

be impractical. The circuit to find the right segment for a given input could be complex, hence large and slow. A more hardware-friendly systematic segmentation scheme is needed which leads to the development of HSM. Nevertheless, the optimum segments give us an indication of how well a given segmentation scheme matches the optimum segmentation in terms of numbers of segments. Moreover, they provide information on the non-linearities of a function.

HSM, which is based on piecewise polynomial approximations and uses an efficient hierarchy of US and P2S is also presented in [8]. This scheme enables us to use segmentations that take the non-linear regions of a function into account, resulting in significantly fewer segments than the traditional uniform segmentation. The hierarchy schemes H we have chosen are P2S(US), P2SL(US), P2SR(US) and US(US). These four schemes cover most of the non-linear functions of interest. P2S means segments that increase by powers of two up to the mid-point then decrease by powers of two. P2SL are segments that increase by powers of two, and P2SR are segments that decrease by powers of two. US are the conventional uniform segments. P2SL(US) would mean that the outer segmentation is P2SL and there are inner segments US nested inside the outer segments.

We have implemented HSM in MATLAB, which deals with the four schemes. The program called HFS (hierarchical function segmenter) takes the following inputs: the function f to be approximated, the polynomial degree d, input range, the requested output error e_{max}, ulp of the input, operand size n, hierarchy scheme H, number of bits for the outer segment v_0, and precisions of the polynomial coefficients and the data paths. HFS divides the input interval into outer segments whose boundaries are determined by H and v_0. HFS finds the optimum number of bits v_1 for the inner segments for each outer segment, which meets the requested output error constraint. For each outer segment, HFS starts with $v_1 = 0$ and computes the error e of the approximation. If $e > e_{max}$ then v_1 is incremented and the error e for each inner segment is computed, i.e. the number of inner segments is doubled in every iteration. If it detects that $e > e_{max}$ it increments v_1 again. This process is repeated until $e \leq e_{max}$ for all inner segments of the current outer segment. This is the point at which HFS obtains the optimum number of bits for the current outer segment. HFS performs this process for all outer segments.

Figure 24.2 shows the segmented functions obtained from HFS for 16-bit second order approximations to the four functions. It can be seen that for f_3, the segments produced by HFS closely resemble the optimum segments in Figure 24.1. Double precision is used for the data paths to get these results. Note that for relatively linear functions such as f_4, the advantages of using HSM over uniform segmentation is small. The advantages of HSM become more apparent as the functions become more non-linear (f_1 being an extreme example).

24.3 The Effects of Polynomial Degrees

The degrees of the polynomials play an important role when approximating functions with HSM: the higher the degree, the fewer segments are needed to meet the same error requirement. However, higher degree polynomials require more multipliers and adders, leading to higher circuit complexity and more delay. Hence, there is a tradeoff between table size, circuit complexity and delay.

Table size. As the polynomial degree d increases, the width of the coefficient look-up table increases linearly. One needs to store $d + 1$ coefficients per segment.

Circuit Complexity. As d increases, one needs more adders and multipliers to perform the actual polynomial evaluation. These increase in a linear manner: since we are using Horner's rule, d adders and d multipliers are required.

Delay. Note that the polynomial coefficients in the look-up table can be accessed in parallel. Hence the delay differences between the different polynomial degrees occur when performing the actual polynomial evaluation. Due to the serial nature of Horner's rule, the increase in delay of the polynomial evaluation is again linear (d adders and d multipliers).

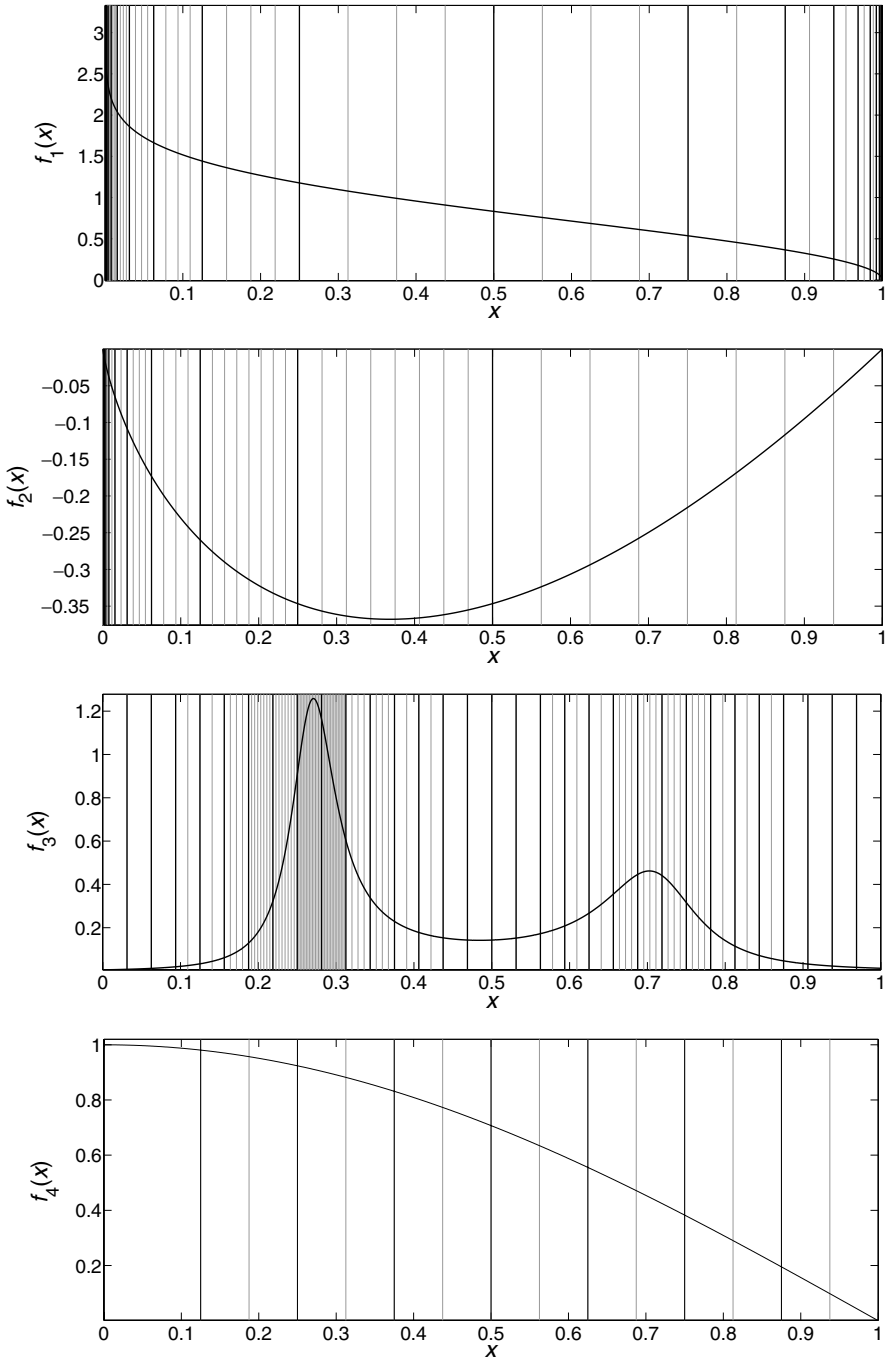

Figure 24.2. The segmented functions generated by HFS for 16-bit second order approximations. f_1, f_2, f_3 and f_4 employ P2S(US), P2SL(US), US(US) and US(US) respectively. The black and grey vertical lines are the boundaries for the outer and inner segments respectively.

We vary the polynomial degree for a given approximation, the circuit complexity and delay are predictable. However, for the table size, although the "number of coefficients per segment" (the width of the table) is predictable, the size of the look-up table depends on the total of number of segments (the depth of the table) as well. To explore the behavior of the table size, we have calculated table sizes at various operand sizes between 8 to 16 bits and polynomial degrees from one to five. Double precision is used for the data paths to get these results. The bit widths of the polynomial coefficients are assumed to be the same as the operand size. Mesh plots of these parameters for the four functions are shown in Figure 24.3.

Interestingly, all four functions share the same behavior. We observe that the plots have an exponential behavior in table size in both the operand width and the polynomial degree. Although first order approximations have the least circuit complexity and delay (one adder and one multiplier), we can see that they perform poorly (in terms of table size) for operand widths of more than 16 bits. Second order approximations have reasonable circuit complexity and delay (two adders and two multipliers) and for the bit widths used in these experiments (up to 24 bits), they yield reasonable table sizes for all four functions.

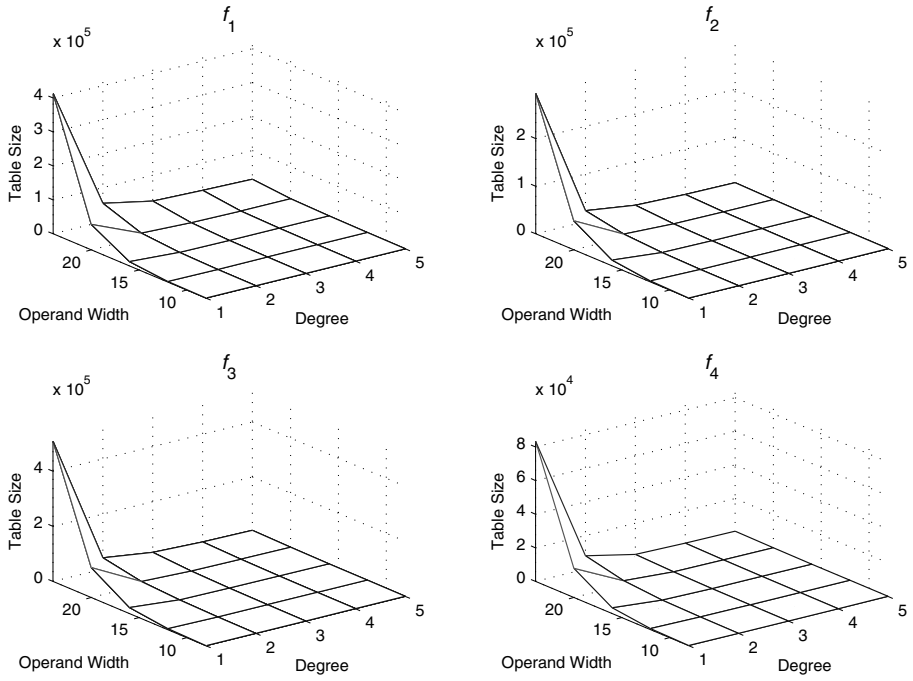

Figure 24.3. Variations of the table sizes to the four functions with varying polynomial degrees and operand bit widths.

Another observation is that the improvements in table size of using third or higher order polynomials are very small. For instance, looking at the 24-bit results to f_2, the table sizes of first, second and third order approximations are 294768, 22680 and 8256 bits respectively. The difference between first and second order is a factor of 11.3, whereas the difference between second and third order is a factor of 2.7. Therefore, the overhead of having an extra adder and multiplier stage for third order approximations maybe not be worth while for a table size reduction of a factor of just 2.7.

Hence, we conclude that for operand sizes of 16 bits or fewer, first order approximations yield good results. For operand sizes between 16 and 24 bits, second order approximations are perhaps more appropriate. We predict that for operand sizes larger than 24 bits, the table size improvements of using third or higher order polynomials will appear more dramatic.

In [8], we have compared the number of segments of HSM to the optimum segmentation. For first and second order approximations, the ratio of segments obtained by HSM and the optimum is around a factor of two. To explore how this ratio behaves at varying operand widths and polynomial degrees, we obtain the results shown in Figure 24.4. We can see that the ratios are around a factor of two at various parameters, with HSM showing no obvious signs of degradation.

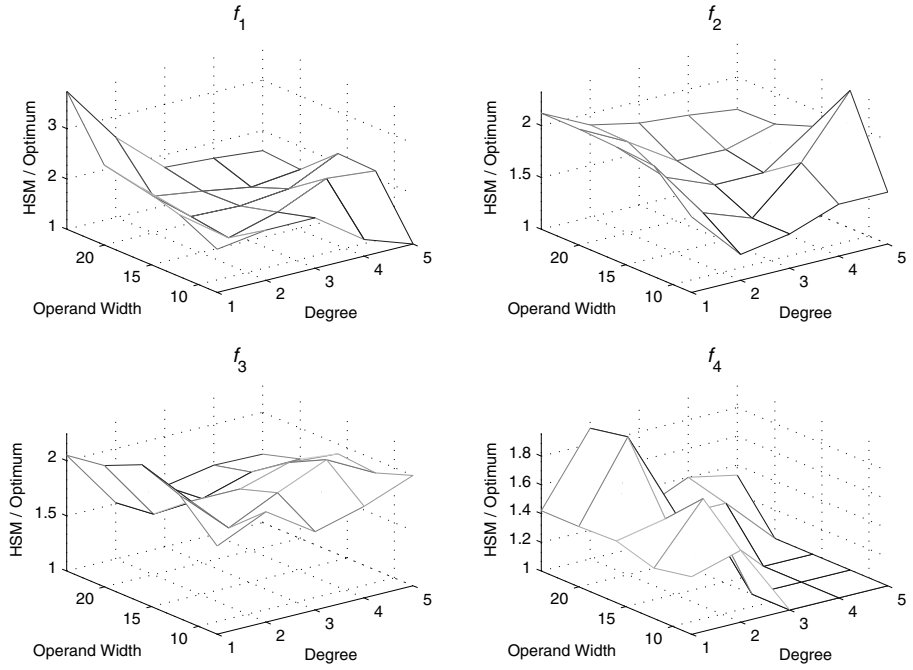

Figure 24.4. Variations of the HSM / Optimum segment ratio with polynomial degrees and operand bit widths.

24.4 Evaluation and Results

Table 24.1 compares HSM with direct table look-up, SBTM and STAM for 16 and 24-bit approximations to f_2. We observe that table sizes for direct look-up approach are not feasible when the accuracy requirement is high. SBTM/STAM significantly reduce the table sizes compared to the direct table look-up approach, at the expense of some adders and control circuitry. In the 16-bit results, HSM_4 has the smallest table size, being 546 and 8.5 times smaller than direct look-up and STAM. The table size improvement of HSM is of course at the expense of more multipliers and adders, hence higher latency. Generally, the higher the polynomial degree, the smaller the table size. However, in the 16-bit case, HSM_5 actually has larger table size than HSM_4. This is because of the extra overhead of having to store one more polynomial coefficient per segment exceeds the reduction in number of segments compared to HSM_4. We observe for the 24-bit results, the differences in table sizes between HSM and other methods are even larger. Moreover, the reductions in table size by using higher order polynomials get greater as the operand widths increase (i.e. as the accuracy requirement increases). For applications that require relatively low accuracies and latencies, SBTM/STAM may be preferred. For high accuracy applications that can tolerate longer latencies, HSM would be more appropriate.

Table 24.1. Comparison of direct look-up, SBTM, STAM and HSM for 16 and 24-bit approximations to f_2. The subscript for HSM denotes the polynomial degree, and the subscript for STAM denotes the number of multipartite tables used. Note that SBTM is equivalent to $STAM_2$.

operand width	method	table size (bits)	compression	multiplier	adder
16	direct	1,048,576	546.1	–	–
	SBTM	29,696	15.5	–	1
	$STAM_4$	16,384	8.5	–	3
	HSM_1	24,384	12.7	1	2
	HSM_2	4,620	2.4	2	3
	HSM_3	2,304	1.2	3	4
	HSM_4	1,920	1.0	4	5
	HSM_5	2,112	1.1	5	6
24	direct	402,653,184	77,672.3	–	–
	SBTM	2,293,760	442.5	–	1
	$STAM_6$	491,520	94.8	–	5
	HSM_1	393,024	75.8	1	2
	HSM_2	40,446	7.8	2	3
	HSM_3	11,008	2.1	3	4
	HSM_4	6,720	1.3	4	5
	HSM_5	5,184	1.0	5	6

Table 24.2. Hardware synthesis results on a Xilinx Virtex-II XC2V4000-6 FPGA for 16 and 24-bit, first and second order approximations to f_2 and f_3.

function	order	operand width	speed (MHz)	latency (cycles)	slices	block RAMs	block multipliers
f_2	1	16	202	11	332	2	2
		24	160	12	897	44	4
	2	16	153	13	483	1	4
		24	135	14	871	2	10
f_3	1	16	232	8	198	2	1
		24	161	10	418	37	2
	2	16	198	12	234	1	3
		24	157	13	409	3	4

The reference design templates have been implemented using Xilinx System Generator. These design templates are fully parameterizable, and changes to the desired function, input interval, operand width or finite precision parameters can result in producing a new design automatically.

A variant [7] of our approximation scheme to f_1 and f_4, with one level of P2S and US(P2S), has been implemented and successfully used for the generation of Gaussian noise samples [6]. Table 24.2 contains implementation results for 16 and 24-bit, first and second order approximations to f_2 and f_3, which are mapped and tested on a Xilinx Virtex-II XC2V4000-6 FPGA. The bit widths of the coefficients and the data paths have been optimized by hand to minimize the size of the multipliers and look-up tables. The design is fully pipelined generating a result every clock cycle. Designs with lower latency and clock speed can be obtained by reducing the number of pipeline stages. The designs have been tested exhaustively over all possible input values to verify that all outputs are indeed faithfully rounded.

We can see that for the 16-bit results, the first order approximations have faster speed, lower latency, and fewer slices and block multipliers. But they use slightly more block RAMs, which is due to the larger numbers of segments required. For the 24-bit cases, the first order results are again faster (in terms of both clock speed and latency) and use fewer block multipliers, but they use slightly more slices and many more block RAMs. This is due the inefficiency of the first order approximations for large operand bit widths as discussed in Section 24.3.

Our hardware implementations have been compared with software implementations (Table 24.3). The FPGA implementations compute the functions using HSM with 24-bit operands and second order polynomials. Software implementations are written in C generating single precision floating point numbers, and are compiled with the GCC 3.2.2 compiler. This is a fair comparison in terms of precision, since single precision floating point has 24-bit mantissa accuracy.

Table 24.3. Performance comparison: computation of f_2 and f_3 functions. The XC2V4000-6 FPGA belongs to the Xilinx Virtex-II family. The Athlon and the Pentium PCs are equipped with 512MB and 1GB DDR RAMs respectively.

function	platform	speed (MHz)	throughput (operations/second)	completion time (ns)
f_2	XC2V4000-6 FPGA	135	135 million	104
	AMD Athlon PC	1400	7.14 million	140
	Intel Pentium 4 PC	2600	0.48 million	2088
f_3	XC2V4000-6 FPGA	157	157 million	83
	AMD Athlon PC	1400	1.76 million	569
	Intel Pentium 4 PC	2600	1.43 million	692

For the f_2 function, the Virtex-based FPGA implementation is 20 times faster than the Athlon-based PC in terms of throughput, and 1.3 times faster in terms of completion time. We suspect that the inferior results of the Pentium 4 PC are due to inefficient implementation of the log function in the gcc math libraries for the Pentium 4 CPU. Looking at the f_3 function, the FPGA implementation is 90 times faster than the Athlon-based PC in terms of throughput, and 7 times faster in terms of completion time. This increase in performance gap is due to the f_3 function being more 'compound' than the f_2 function. Whereas a CPU computes each elementary operation of the function one by one, HSM looks at the entire function at once. Hence, the more compound a function is, the advantages of HSM get bigger.

Note that the FPGA implementations use only a fraction (less than 2%) of the device used, hence by instantiating multiple function evaluators on the same chip for parallel execution, we can expect even larger performance improvements.

24.5 Conclusion

In this chapter, we have presented the effects of polynomial degrees on the hierarchial segmentation method (HSM) for evaluating functions. The effects are explored using four functions: $\sqrt{-\log(x)}$, $x\log(x)$, a high order rational function and $\cos(\pi x/2)$. HSM uses a novel hierarchy of uniform segments and segments with size varying by powers of two. This scheme enables us to approximate non-linear regions of a function particularly well. Compared to other popular methods such as STAM, our approach has longer latencies and operators, but the size of the look-up tables are considerably smaller. The table size improvements get larger as the operand bit width increases.

Looking at the effects of the polynomial degrees, we have seen that first order approximations are appropriate for operand widths of fewer than 16 bits, and second order approximations are well suited for operand widths between

16 and 24 bits. We expect that the advantages of using higher order polynomials will become larger for operands widths of more than 24 bits. We have also seen that the performance of HSM over the optimum segmentation is around a factor of two throughout various operand widths and polynomial degrees.

References

[1] J. Cao, B.W.Y. We and J. Cheng, "High-performance architectures for elementary function generation", *Proc. 15th IEEE Symp. on Comput. Arith.*, 2001.

[2] J.N. Coleman, E. Chester, C.I. Softley and J. Kadlec, "Arithmetic on the European logarithmic microprocessor", *IEEE Trans. Comput. Special Edition on Comput. Arith.*, vol. 49, no. 7, pp. 702–715, 2000.

[3] D. Das Sarma and D.W. Matula, "Faithful bipartite rom reciprocal tables", *Proc. IEEE Symp. on Computer Arithmetic*, pp. 17–28, 1995.

[4] H. Henkel, "Improved Addition for the Logarithmic Number System", *IEEE Trans. on Acoustics, Speech, and Signal Process.*, vol. 37, no. 2, pp. 301–303, 1989.

[5] C.L. Lawson, "Characteristic properties of the segmented rational minimax approximation problem", *Numer. Math.*, vol. 6, pp. 293–301, 1964.

[6] D. Lee, W. Luk, J. Villasenor and P.Y.K. Cheung, "A hardware Gaussian noise generator for channel code evaluation", *Proc. IEEE Symp. on Field-Prog. Cust. Comput. Mach.*, pp. 69–78, 2003.

[7] D. Lee, W. Luk, J. Villasenor and P.Y.K. Cheung, "Hardware function evaluation using non-linear segments", *Proc. Field-Prog. Logic and Applications*, LNCS 2778, Springer-Verlag, pp. 796–807, 2003.

[8] D. Lee, W. Luk, J. Villasenor and P.Y.K. Cheung, "Hierarchical Segmentation Schemes for Function Evaluation", *Proc. IEEE Int. Conf. on Field-Prog. Tech.*, pp. 92–99, 2003.

[9] D.M. Lewis, "Interleaved memory function interpolators with application to an accurate LNS arithmetic unit", *IEEE Trans. on Comput.*, vol. 43, no. 8, pp. 974–982, 1994.

[10] O. Mencer, N. Boullis, W. Luk and H. Styles, "Parameterized function evaluation for FPGAs", *Proc. Field-Prog. Logic and Applications*, LNCS 2147, Springer-Verlag, pp. 544–554, 2001.

[11] J.M. Muller, *Elementary Functions: Algorithms and Implementation*, Birkhauser Verlag AG, 1997.

[12] A.S. Noetzel, "An interpolating memory unit for function evaluation: analysis and design", *IEEE Trans. Comput.*, vol. 38, pp. 377–384, 1989.

[13] J.A Piñeiro, J.D. Bruguera and J.M. Muller, "Faithful powering computation using table look-up and a fused accumulation tree", *Proc. 15th IEEE Symp. on Comput. Arith.*, 2001.

[14] M.J. Schulte and J.E. Stine, "Symmetric bipartite tables for accurate function approximation", *Proc. 13th IEEE Symp. on Comput. Arith.*, vol. 48, no. 9, pp. 175–183, 1997.

[15] J.E. Stine and M.J. Schulte, "The symmetric table addition method for accurate function approximation", *J. of VLSI Signal Processing*, vol. 21, no. 2, pp. 167–177, 1999.