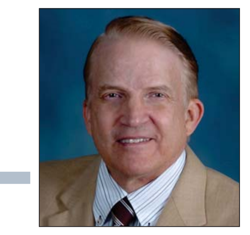
GPU K20 ACCELERATION OF TESTS FOR ASYMMETRIC FUNCTIONS



DANIEL P. ZULAICA and JON T. BUTLER

Dept. of Electrical and Computer Eng.
Naval Postgraduate School
Monterey, CA 93943-5121

Many systems that encrypt and decrypt plaintext messages use Boolean functions with specific properties. Bent Boolean functions, which are the most nonlinear of Boolean functions, have strong immunity to an attack (breaking of the code) that is based on linear functions. The only known approach to generating all bent functions is exhaustive search. Research into bent functions has inspired research into functions with similar properties. We have focused on asymmetric functions, especially those functions with large asymmetry. In the research reported here, we have extended our study of bent functions through exhaustive enumeration to maximally asymmetric functions. We show that a K20 GPU processor can achieve a computation speedup of 150 times that of a serial processor.

$x_1x_2x_3x_4$	$f = x_1x_2 \oplus x_2x_3 \oplus x_3x_4$	$g = x_1x_2 \vee x_1x_3 \vee x_1x_4 \vee x_2x_3 \vee x_2x_4 \vee x_3x_4$	$f \oplus g$
0000	0	0	0
0001	0	0	0
0010	0	0	0
0011	1	1	0
0100	0	0	0
0101	0	1	1
0110	1	1	0
0111	0	1	1
1000	0	0	0
1001	0	1	1
1010	0	1	1
1011	1	1	0
1100	1	1	0
1101	1	1	0
1110	0	1	1
1111	1	1	0
NS			5

Table 1. Example of Hamming Distances Between Functions.

Symmetric Function	*Distance to* $f = x_1x_2 \oplus x_2x_3 \oplus x_3x_4$	Symmetric Function	*Distance to* $f = x_1x_2 \oplus x_2x_3 \oplus x_3x_4$
0	7	-	6
01	11	1	10
02	7	2	6
03	7	3	6
04	6	4	5
012	11	12	10
013	11	13	10
014	10	14	9
023	7	23	6
024	6	24	5
034	6	34	5
0123	11	123	10
0124	10	124	9
0134	10	134	9
0234	6	234	5
01234	10	1234	9

Table 2. Example of Hamming Distances Among Functions.

The asymmetry of a Boolean function f is the fewest truth table entries that must be changed to convert f into a symmetric function. That is, the asymmetry of f is the minimum Hamming distance f is from a symmetric function. For example, the 3-variable function $f = x_1 x_2 x_3$ has asymmetry 0, since it is symmetric, while $f = x_1 x_2 x_3 + \overline{x}_1 \overline{x}_2 x_3$ has asymmetry 1, since it is not symmetric, and only one truth table entry must be changed to make it symmetric (that associated with $x_1 x_2 x_3 = 001$). As in the case of bent functions, one can analyze the asymmetry of a function by computing the Hamming distance between it and all symmetric functions. There are $2^{(n+1)}$ symmetric functions, and our program enumerates all for each function, determining the smallest Hamming distance.

Consider Table I. Here, an example 3-variable function is compared against one of the 16 3-variable symmetric functions. They differ in 5 entries.

Table II shows the distance this example function is from all 3-variable symmetric functions. The entry highlighted in green is the same symmetric function shown in Table I. Among all symmetric functions, the smallest distance is 5. Thus, this function has asymmetry 5.

The following is an outline of the problem implemented in sequential and parallel CUDA C code.

Create Function under test.

Number of bits (do separate programs for 2,3,4,5 – consolidate of time permits)

Function generator – ie. For n=4 there are 64000 functions to test.

For n=4 Create 2^(2^n) bit variables which is a 16-bit variable, 2-bytes or short unsigned int. Lab 8 hand out says use a simple counter, add bits, do the masking.

(n=2 is 4-bit
n=3 is 8-bit
n=4 is 16-bit
n=5 is 32-bit)

Create array of semetric functions.

Unsigned int n2_table(32) = 32 values for bits as integers.
Function generator.
For i=0 to 2^16 i++ – unsigned i

Exclusive or of symmetric function.
For j=0 to j<32;j++ - 32 4-bit symmetric functions

N2_table(j) = 0,1, …. To (01234 = 255)
So n2_table(j) xor I
Tally bits – right shift 0 to 15 bits and add (other methods possible)

Example of symmetric functions
Tables, for n=2, there are 8 symmetric functions: 0, 1, 6, 7, 8, 9, 14, 15

For n = 3, there are 16: 0, 1, 22, 23, 104, 105, 126, 127,128, 129, 150, 151, 232, 233, 254, 255

Errors or Problems Encountered During Programming

- The C and CUDA C programming proceeded relatively easily. Problems occurred testing for N = 5 functions, which used 32-bit integers.
  - o Using standard C 32-bit unsigned or 'long int' were needed to test half of the functions
    - * Doubling got the correct numbers.
  - o Using CUDA C host memory allocation limits per process errors occurred.
    - * This error occurred using half, or 2^31 functions in one call.
    - * Solving occurred by reducing the function calls until the error went away.
    - * Then loop to call the kernel until all functions were tested.
  - o This allowed testing of all 2^32 functions and confirmed that doubling the results for a test of the first half, 2^31, functions was correct.
    - * Using 'long int' was used in CUDA C.

Figure 1 shows the distribution of 4-variable functions according to asymmetry. For example, 32 have 0 asymmetry. These are the 32 symmetric functions. Also, Figure 1 shows that there are 2,880 functions that have a maximum asymmetry of 7.

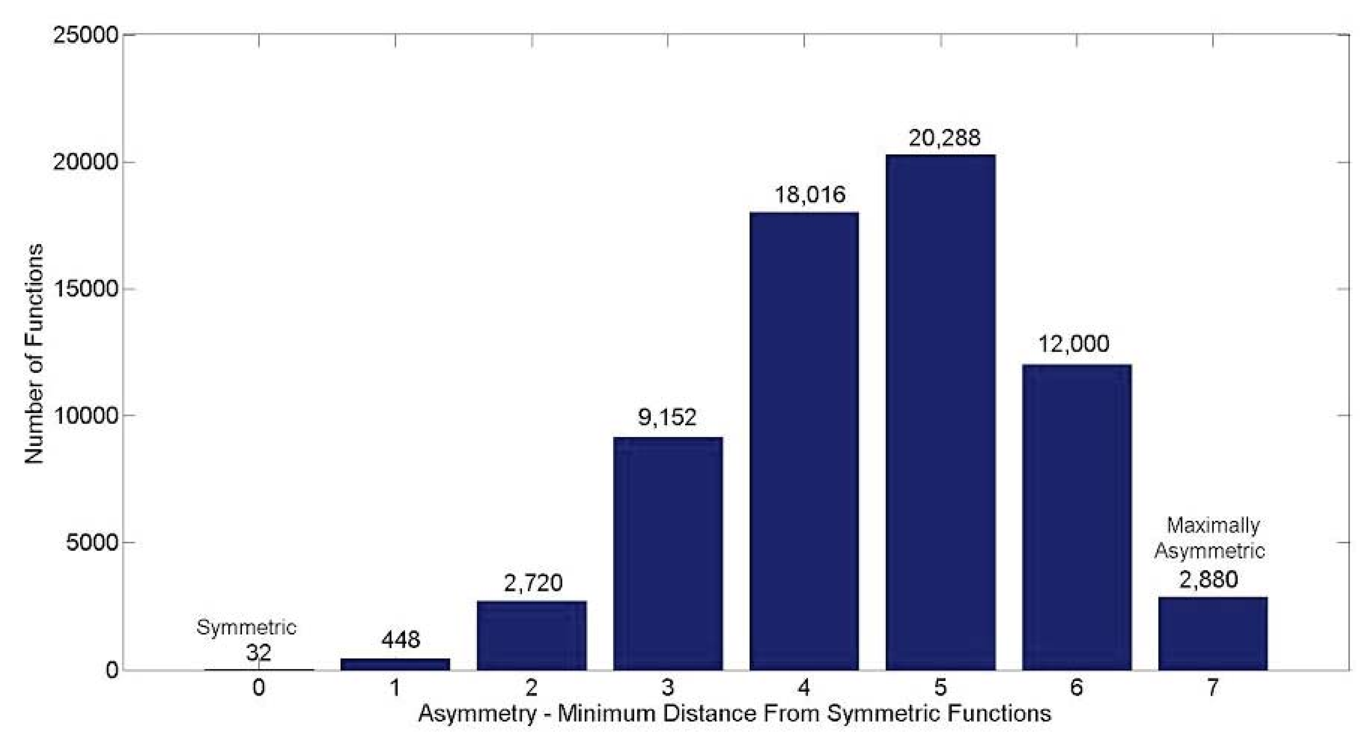


Figure 1. Distribution of 4-Variable Functions to Non-Symmetry

Figures 2 and 3 show a block diagram of asymmetry computation and an architecture of the asymmetry computation respectively.

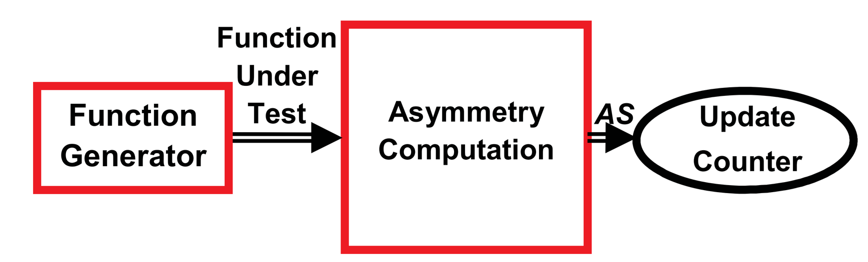


Figure 2. Block Diagram of the Asymmetry Computation

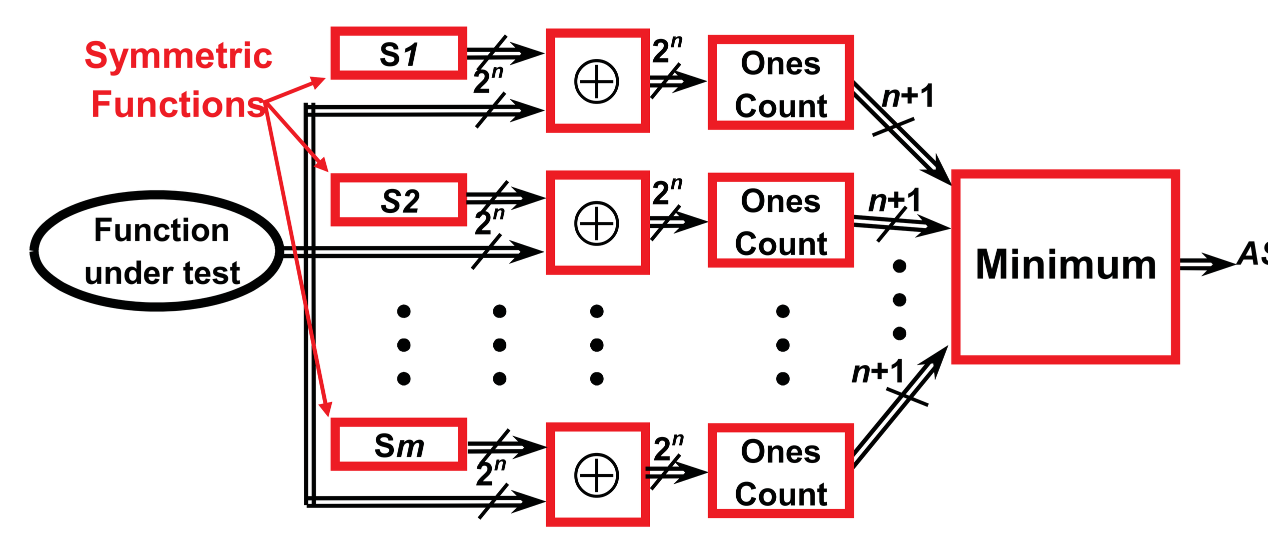


Figure 3. Architecture of the Asymmetry Computation

Run Time Summary – Total Wall Clock Time (Seconds)					
	N				
	2	3	4	5 – full	5 – half
Serial	0.02	0.02	0.07	---	6937.74
Parallel	3.43	3.43	3.33	88.01	46.03

Run Time Summary – User Program Time (Minus wait and system times) (Seconds)					
	N				
	2	3	4	5 – full	5 – half
Serial	< 0.02	< 0.02	0.05	---	6931.97
Parallel	0.01	0.01	0.02	75.66	37.82

Run Time Summary – Average Time Per Function Under Test Tested (User Time, Wall Clock Time)					
	N				
	2 (msec)	3 (msec)	4 (usec)	5 – full (nsec)	5 – half (nsec)
Serial	< 1.25, 1.25	< 0.08, 0.08	0.763, 1.07	---	3228.0, 3230.6
Parallel	0.625, 0214.4	0.039, 13.4	0.305, 50.812	17.6, 20.5	17.6, 21.4

The final three tables show the timing summaries using the Linux 'time' function found in the directory '/usr/bin'. This might be what an end user (customer) might see. Using this for all programs established a common timing environment for, at the very least, the total wall clock time. This allows for some interesting average times processing each function under test.

Daniel P. Zulaica was a research associate in the Department of Electrical and Computer Engineering at the Naval Postgraduate School in Monterey, CA. He holds a B.S. degree in Physics from the University of Texas – Arlington. His research interests include digital systems, computer vision and image processing, radar characterization processing, VLSI mixed signal design, power systems distance learning web interfaces, cyber warfare, and embedded processing. He is now a research associate at Gantz-Mountain Intelligence Automation Systems, Inc.

Jon T. Butler is a Distinguished Professor Emeritus in the Department of Electrical and Computer Engineering at the Naval Postgraduate School in Monterey, CA. From 1974 until 1987, he was a faculty member in the Department of Electrical Engineering and Computer Science at Northwestern University in Evanston, IL. He holds a Ph.D. (EE) degree from Ohio State University and an M.Engr.(EE) and B.E.E. degree from Rensselaer Polytechnic Institute, Troy, NY. His research interests include reconfigurable computing, combinatorial scientific computing, and logic design.