SHA message digest computation on GPU

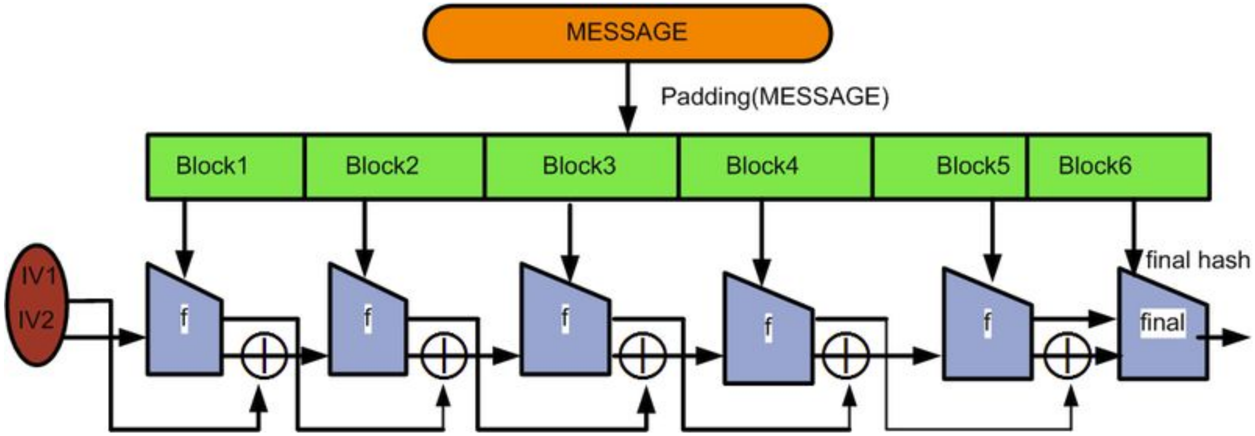
Nachiket Trivedi 201401047
Maulik Trapasiya 201401051

Objectives

- ➔ Implement SHA-1 hash function on GPU
- ➔ Explain the SHA hash function
- ➔ Performance analysis
- ➔ Future Improvements

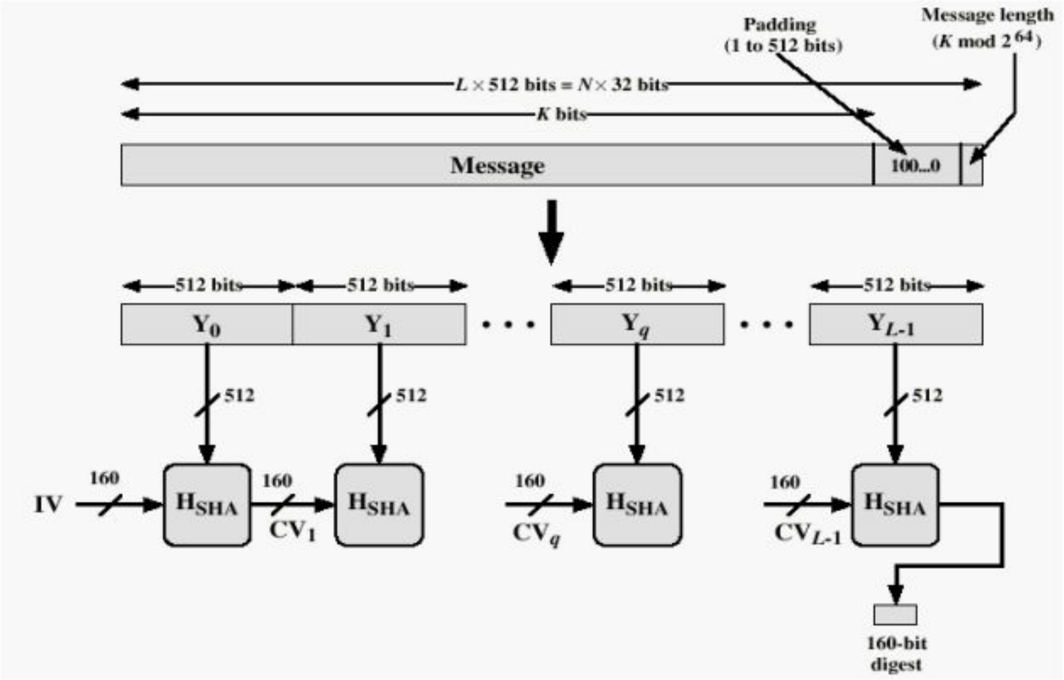
Hash Functions

- ➔ Mostly used for providing authentication, integrity and nonrepudiation.
- ➔ Most hash functions including SHA based on Merkle–Damgård construction method
- ➔ The hash function takes the message as input and gives a digest of a fixed length as the output
- ➔ Message converted into an input which is multiple of a pre-defined by block size.
- ➔ The output of one step acts as initialization input of the next one.



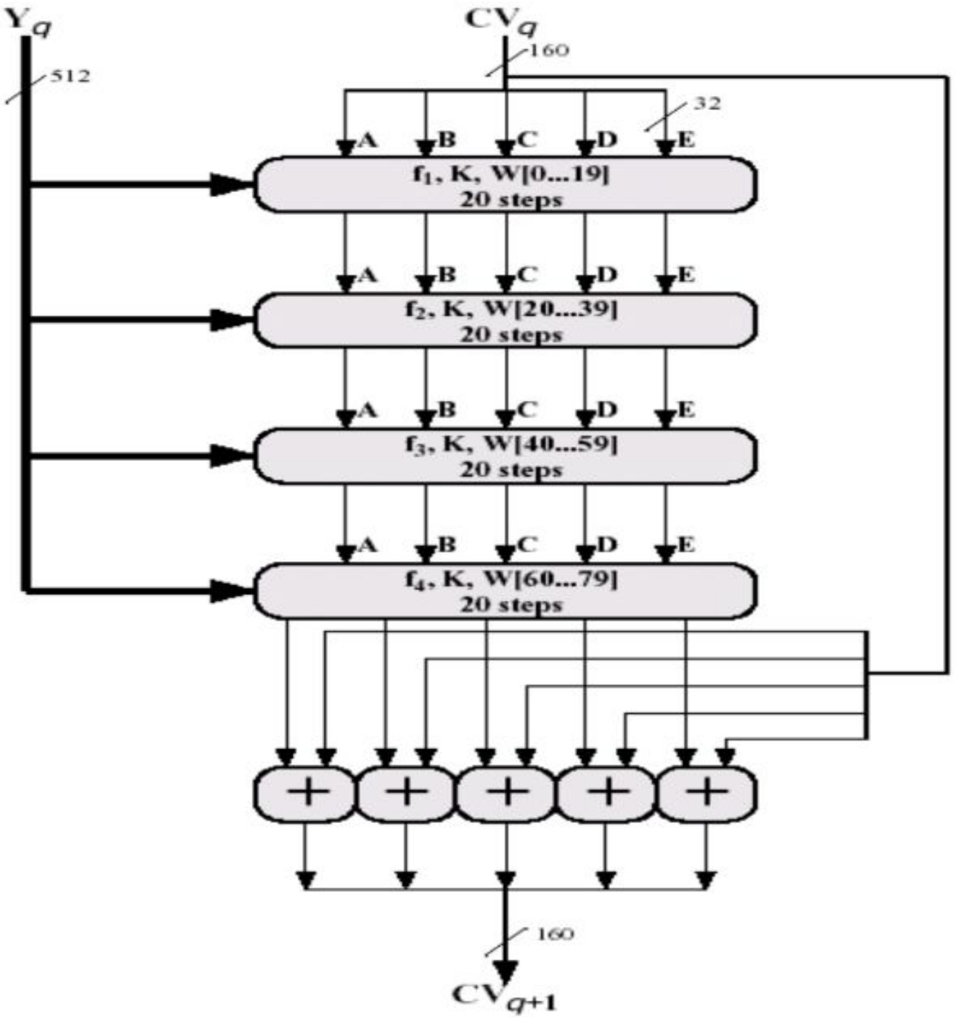
Secure Hash Algorithm-1

- ➔ The message is first converted into a multiple of 512 bits by padding with 100….0 and message length.



- ➔ Now, these blocks are sequentially given to a compression function which takes 5 buffer variables of 32 bits each as input shown in the figure as IV.

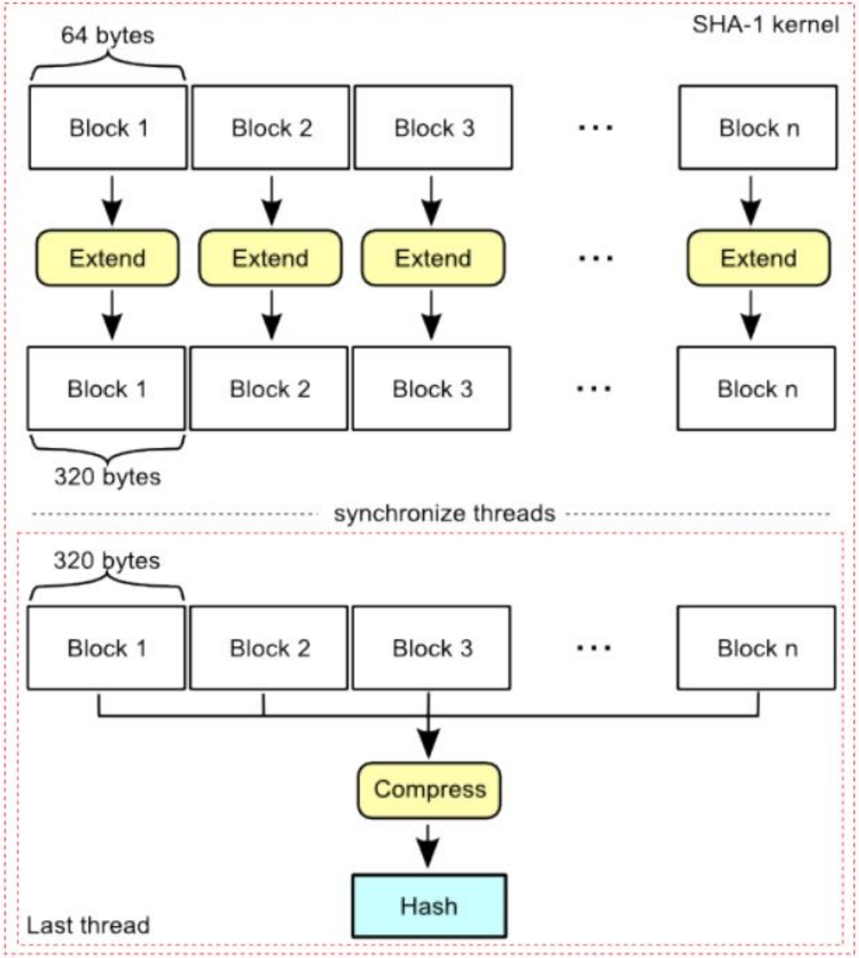
- ➔ Inside the function f, there are four steps with 20 rounds each thereby making a total of 80 rounds.
- ➔ The 512-bit block is the 16 32-bit sub-blocks are expanded to 80 32-bit sub-blocks, on sub-block for each round.
- ➔ This output replaces the buffer vectors which in turn acts as input for next step.
- ➔ output is added with the initial vectors to obtain the final 160-bit digest.



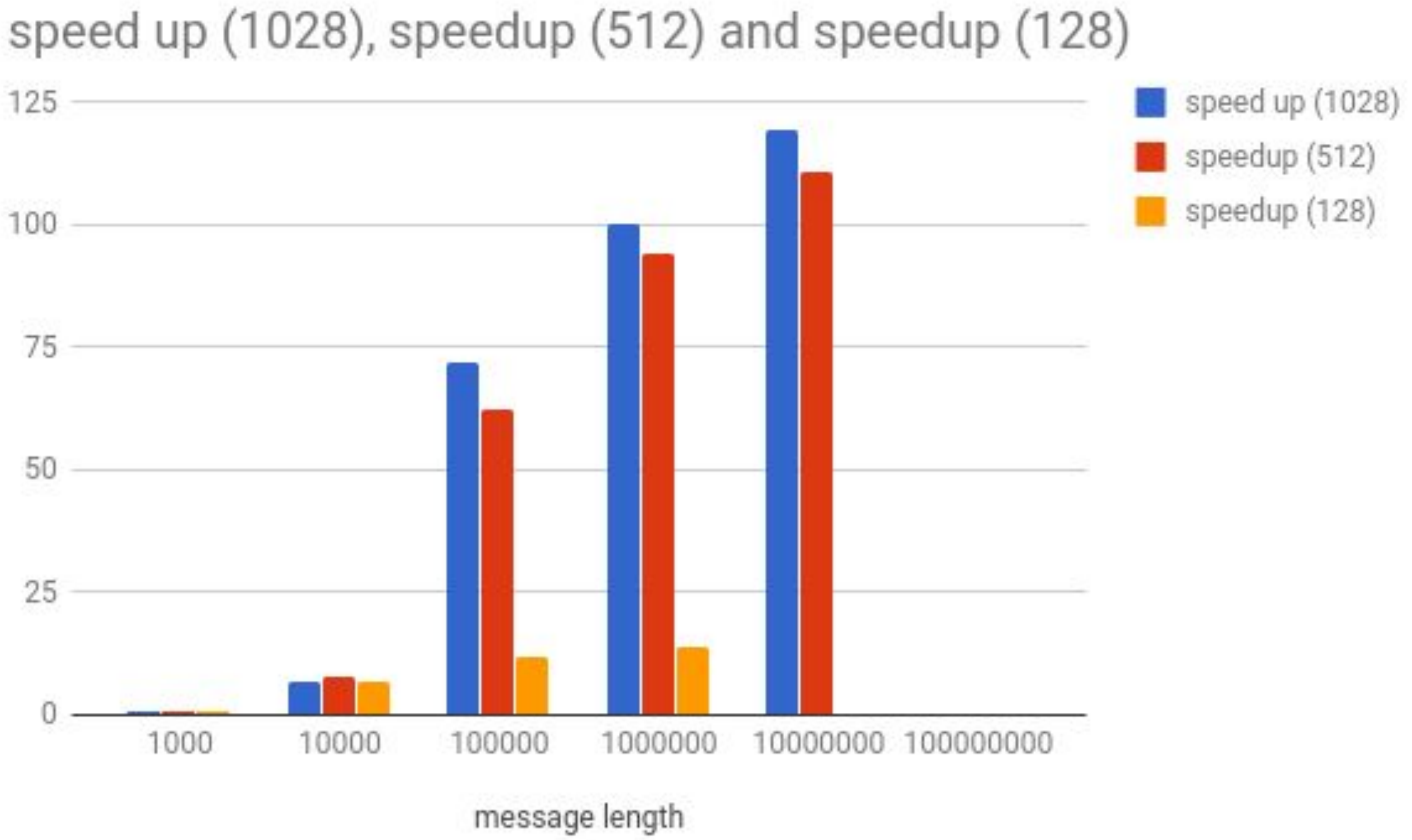
GPU Implementation

- ➔ SHA-1 is an avalanche algorithm and this poses as a great challenge while parallelizing.
- ➔ Expansion algorithm does not depend on any pre-computed value and only takes the 512-bit block as input.
- ➔ To do this, we invoked the kernel where each thread will compute the expansion function of each block.
- ➔ After the parallel computation of expansion function is completed, it is necessary to sync them for our next step.

- ➔ As the syncthreads() function only works within a block, we can’t have threads running parallely in two different blocks.
- ➔ If the input message length is greater than 1024*512=524288 bits, the kernel is re-launched and the same process is repeated again and again till all the sub-blocks are processed and synced.
- ➔ The next step is the compression function where we used a single thread which serially computes 1024 segments.
- ➔ The output of this thread will be stored in buffer values and will be given to next kernel launch as parameter.
- ➔ This whole process goes on in loop until all the segments are processed.



1024 thread_per_block	Size	KERNEL	cudaMemcpy	cudaMalloc	cudaFree
CPU	1000	0.012			
GPU	1000	0.014	0.67	0.264	0.193
CPU	10000	0.101			
GPU	10000	0.015	5.114	0.406	0.269
CPU	100000	1.006			
GPU	100000	0.014	1.119	0.382	0.267
CPU	1000000	10.011			
GPU	1000000	0.1	488.700012	0.39	0.28
CPU	10000000	100.711998			
GPU	10000000	0.842	4885.374023	0.449	0.25
CPU	100000000	813.338989			
GPU	100000000	16178.08496	32745.14648	0.58	0.361


speed up (1028), speedup (512) and speedup (128)
125
100
75
50
25
0
1000
10000
100000
1000000
10000000
100000000
message length
speed up (1028)
speedup (512)
speedup (128)

Conclusion

- ➔ As seen in the results, when the GPU is used at its fullest of 1024 threads per block, we see significantly better performance.
- ➔ As for higher inputs, the GPU implementation doesn’t work well as compared to CPU.
- ➔ Given the primary motive of this algorithm is to provide authentication and integrity, seldom arises a case of large inputs, as the messages for digital signatures of passwords are kept generally small.
- ➔ Right now in market, SHA-1 is more secure than its competitor MD5 but MD5 is faster than SHA.
- ➔ Thus, as our GPU implementation of SHA-1 can prove a better choice as compared to MD5.
- ➔ For future improvements, we can use a technique to sync threads across the blocks.
- ➔ Another improvement can be the use of SHA-2 for parallelization. The implementation of SHA-2 is different and more complex than SHA-1, and is subject to complete parallelization by using binary tree for processors.