Implementation of Streebog cryptographic hash function family on NVIDIA CUDA platform

Pavel A. Lebedev

Streebog [1] is a family of cryptographic hash functions intended to replace the current GOST R 34.11-94 national standard of Russian Federation. The family consists of Streebog-256 and Streebog-512 hash functions with hash value size 256 and 512 bits respectively that differ by the initial state and what part of final state constitutes the result.

The main difference of the proposed hash function from GOST R 34.11-94 is in compression function. Its base primitive operation denoted LPS is a chained application of three transforms:

1. S — nonlinear bijection. Treats 512 operand bits as an array of 64 bytes and replaces each of them according to the predefined substitution table;

2. P — byte reordering. Rearranges operand bytes as per the standard;

3. L — linear transformation. Operand is treated as 8 64-bit vectors and each of them is replaced by the result of multiplication over GF(2) with a predefined 64×64 matrix.

Only LPS and bitwise XOR of 512-bit blocks are used in compression function. Together with addition modulo 2^{512}, these are the only operations used in Streebog hash function.

A study of implementation details for these candidate hash functions using GPGPU computing is important to test how well-suited is the standard's developers choice of primitive operations and data units for modern massively parallel architectures. NVIDIA CUDA [2] was chosen as an implementation platform for this work for its ease of use, maturity and popularity.

Since every output of the compression function depends on its output for the previous block, it is impossible to distribute its calculation for blocks of the same message. This problem is not specific to Streebog and is shared by most hash functions, limiting the set of applications where parallel hash function calculation is beneficial. This leaves only one parallel distribution design viable: a small number of threads, preferably within the same warp to avoid synchronization, per message, processing all blocks sequentially. 512 bits of all state and block data is distributed between them in units of built-in integer types. While this design is significantly slower in comparison to a single CPU core, it is possible to process much more messages on a single GPU simultaneously, given its huge quantity of hardware threads that no multi-core/processor host system can match.

Transformation P, as specified in the standard, is essentially a transpose operation on a 8×8 byte matrix. Precalculations can be made for the matrix multiplication calculation to be done per byte. With these observations all three parts of LPS transform can be neatly merged into a loop, that the compiler effectively unrolls. Since it is impossible in this case to perform the transformation in place, bounce buffer is used to avoid extra copying. All predefined and precomputed values are stored in constant memory and dynamically allocated shared memory is used to store state that cannot be kept per-thread.

Addition modulo 2^{512} is implemented with carry-lookahead adder using intra-warp parallel scan. Putting a reasonable limit on message length, addition in calculation of total message bits processed can be simplified to a single-limb implementation. Bitwise XOR operations are trivially distributed between threads. These operations take very little time in comparison to the core LPS transform.

The final benchmarks compare the CPU-based implementation from [3] with NVIDIA GeForce GTX 580 board and take into account data copy time between CPU and GPU without memory pinning. Multiple configurations with a different number of messages and CUDA launch configurations were tested and the best one was chosen. With a message size of at least 16 KB, the GPU achieves the speed of 144 MB/sec, 3.7 times faster than the CPU.

This result shows that, Streebog hash function is sufficiently suited for GPGPU implementations. It uses few relatively straight-forward to implement primitive operations, although the most time-consuming one uses bytes as its data unit and cannot therefore be parallelized further.

References

[1] National standard of Russian Federation. Information Technology. Cryptographic data security. Hash function. — Project, 1st edition. http://www.infotecs.ru/laws/gost/proj/gost3411.pdf

[2] *NVIDIA Corporation* NVIDIA CUDA C Programming Guide Version 4.1 11/18/2011
`http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`

[3] Sergey Grebnev, Andrey Dmukh, Denis Dygin, Dmitry Matyukhin, Vladimir Rudskoy and Vasily Shishkin. Asymmetric Reply to SHA-3: Russian Hash Function Draft Standard. — Presented to CTCrypt 2012 Workshop.