OpenCL cryptographic computations for FAB coin

Todor Milev[*]
todor@fa.biz

May 30, 2018

1 Introduction

Public/private key cryptography is arguably the most important aspect of modern crypto-currency systems. The somewhat slow execution of private/public key cryptography algorithms appears to be one of the main bottlenecks of FAB's Kanban system.

In the present text we describe our port of the cryptographic functions that make up the computational core of fabcoin to the GPU programming language OpenCL. Whether this port is to be used in the final version of Kanban - as opposed to using directly the original fabcoin/bitcoin cryptopgrahic library from which the port was derived - remains to be decided.

1.1 Findings of our investigation

[Coming soon]

1.2 OpenCL vs CUDA

[Coming soon]

1.3 Benchmarks

Speed. All tabulated devices ran the same code. With the exception of a small header file (27 lines for OpenCL and 67 lines for C++), the code is both valid OpenCL and valid C/C++. The total code size is, at the time of writing, 7153 lines of code (= 5049 actual lines + 1294 comments + 810 blank) in a total of 22 files.

Device	run-time	SHA256	Pub. key	Sign	Verification
Nvidia Quadro K2000 (2 computational units)	OpenCL 1.1	19,007/sec	171/sec	121/sec	66/sec
Intel® Xeon® CPU E5-2630 @ 2.60GHz (12 cores)	OpenCL 1.2	96,028/sec	2,063/sec	1,044/sec	639/sec
	C++ std::thread	36,040/sec	13,937/sec	6,471/sec	5,505/sec
Intel® HD Graphics (SOC, see next) @ 1.1Ghz Intel® (23 computational units)	OpenCL 2.0	55,769/sec	N/A	N/A	N/A
Intel® Core™ CPU i5-8250U @ 1.6Ghz (8 cores)	OpenCL 2.0	130,844/sec	1,765/sec	1,018/sec	533/sec
	C++ std::thread	48,865/sec	13,643/sec	5,961/sec	4,136/sec
Radeon™ RX 480 Graphics @1266 Mhz	OpenCL 1.2	31,621/sec	N/A	N/A	N/A
Intel® Core™ CPU i5-7600 @ 3.50GHz (4 cores)	OpenCL 2.0	237,161/sec	1,497/sec	2,592/sec	856/sec
	C++ std::thread	84,259/sec	21,719/sec	6,930/sec	5,093/sec

Compilation times. Measures the difficulty of testing code changes.

[*]FA Enterprise System

Device	run-time	SHA256	init	Pub. key	Sign	Verification
Nvidia Quadro K2000	OpenCL 1.1	0.5 sec	23.3 sec	3.5 sec	6.1 sec	11.5 sec
Intel® Xeon® CPU E5-2630 @ 2.60GHz (12 cores)	OpenCL 1.2	0.1 sec	379.2 sec	147.7 sec	360.3 sec	362.6 sec
Intel® HD Graphics (SOC, see next) @ 1.1Ghz Intel® (23 computational units)	OpenCL 2.0	0.3 sec	69.4 sec	2.8 sec	19.3 sec	39.9 sec
Intel® Core™ CPU i5-8250U @ 1.6Ghz (8 cores)	OpenCL 2.0	0.1 sec	417.7 sec	154.5 sec	382.6 sec	379.8 sec
Radeon™ RX 480 Graphics @1266 Mhz	OpenCL 1.2	0.2 sec	45.1 sec	1.9 sec	11.3 sec	23.9 sec
Intel® Core™ CPU i5-7600 @ 3.50GHz (4 cores)	OpenCL 2.0	0.1 sec	234.7 sec	90.8 sec	235.0 sec	232.4 sec

1.4 The four main cryptographic functions

The four main cryptographic functions used to run FAB coin (and all other major crypto-currencies, powered by various algorithms) are as follows.

1. Secure hashing function.
2. Private/public key pair generation.
3. Cryptographic signature.
4. Cryptographic signature verification.

1) Among many secure hashing functions, FAB uses SHA256 (secure hash algorithm 2, 256 bit). The same algorithm is also used in bitcoin. The secure hash is used to digest messages into 256 bit = 32 byte lengths. SHA256 has the property that, given a message and its digest, it is practically impossible to forge another message that has the same digest. In this way, SHA256 can be reliably used to identify data.
2) Private keys are kept secret and are used to sign data - most importantly, transactions/transfers of funds. The public key is a function of the private key that is published openly alongside a message that may or may not be signed with the private key.

Private/public key pairs have the property that, given a message, a signature and a public key, everyone can verify that the signature was correctly generated with an unknown private key that matches the known public key, but no one can guess the private key from which the signature and the public key were generated. Furthermore, given a message, no one can generate a signature matching a given public key without knowing the secret private key that corresponds to it.

The main use case of private/public key pairs is as follows. A message - say, a statement of transfer of funds - is published in the open. To consent with the transfer, the original owner of the funds produces the cryptographic signature of the transfer message. Since the original owner's public key is published in the open and is therefore known, everyone can verify the correctness of this transfer, but only the original owner can generate it.
3) Cryptographic signatures are used to sign messages using private keys as described in 2). Among many cryptographic schemes for signing messages, FAB coin uses the industry standard ECDSA (Elliptic Curve Digital Signature Algorithm) over the elliptic curve **secp256k1**, see Section 1.5. This is the same algorithm stack as the one used in bitcoin.

The cryptographic signature is a function of the message being signed, the private key, and, in the particular case of (EC)DSA, a one-time use random number.
4) The signature verification algorithm is used to verify signatures generated as described in 2). The signature verification is a function of the signed message, the signature, and the public key of the signature owner.

1.5 ECDSA

Following Bitcoin, FAB coin uses the standard public/private key cryptography ECDSA over **secp256k1**. Here, ECDSA stands for Elliptic Curve Digital Signature Algorithm and **secp256k1** stands for the elliptic curve:

$$y^2 = x^3 + 7$$

(we do not specify the base point here), over the finite field:

$$\mathbb{Z}/p\mathbb{Z},$$

where

$$p = 2^{256} - 2^{32} - 977. \tag{1}$$

At present, fabcoin core uses bitcoin's implementation of **secp256k1**. We report here that we ported one of bitcoin's development forks - Pieter Wuille's C project libsecp256k1 [6] - to OpenCL. This development branch was chosen as our base as it is also the base of the top-google-search (partial) OpenCL port [5]. We did not choose to use [5] directly as the port appeared to not be self-contained and contained only an implementation of the signature verification function 4) from Section 1.4.

References

[1] Jason Evangelho. Intel is developing a desktop gaming gpu to fight Nvidia, AMD, `https://www.forbes.com/sites/jasonevangelho/2018/04/11/intel-is-developing-a-desktop-gaming-gpu-to-fight-nvidia-amd/#54a74d33578f`. 2018.

[2] Kamran Karimi, Neil G. Dickson, and Firas Hamze. A performance comparison of CUDA and opencl. *CoRR*, abs/1005.2581, 2010.

[3] Suejb Memeti, Lu Li, Sabri Pllana, Joanna Kolodziej, and Christoph W. Kessler. Benchmarking opencl, openacc, openmp, and CUDA: programming productivity, performance, and energy consumption. *CoRR*, abs/1704.05316, 2017.

[4] `www.nvidia.com` NVIDIA Corporation. NVIDIA openCL best practices guide, version 1.0, `https://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf`.

[5] Author: `https://github.com/hhanh00`. Project secp256k1-cl, `https://github.com/hhanh00/secp256k1-cl`. 2014.

[6] Pieter Wuille and contributors. libsecp256k1 `https://github.com/sipa/secp256k1`. 2015.