Kanban

Progress report

Todor Milev, Ph.D.

Senior C++ programmer
FA Enterprise System

May 18

Outline

Outline

Outline

Current architecture

- OS: Ubuntu at the moment. Aspirations to support everything.

Current architecture

- OS: Ubuntu at the moment. Aspirations to support everything.
- Computational engine in openCL/C/C++.

Current architecture

- OS: Ubuntu at the moment. Aspirations to support everything.
- Computational engine in openCL/C/C++.
    - Crypto functions in openCL/C++.

Current architecture

- OS: Ubuntu at the moment. Aspirations to support everything.
- Computational engine in openCL/C/C++.
    - Crypto functions in openCL/C++.
    - Data management? Expected solution: stl library (no database).

Current architecture

- OS: Ubuntu at the moment. Aspirations to support everything.
- Computational engine in openCL/C/C++.
  - Crypto functions in openCL/C++.
  - Data management? Expected solution: stl library (no database).
- Networking, management, testing, other non-computational tasks: nodejs.

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.
  - openCL major memory types: `__local`, `__global`, `__constant`

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.
  - openCL major memory types: `__local`, `__global`, `__constant`
  - Mixed arguments: 1 version per combination:

```
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __global const uint32_t * b)
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __local const uint32_t * b)

secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __constant const uint32_t * b)
```

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.
  - openCL major memory types: `__local`, `__global`, `__constant`
  - Mixed arguments: 1 version per combination:

```
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __global const uint32_t * b)
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __local const uint32_t * b)

secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __constant const uint32_t * b)
```

  - No C++ templates in openCL: emulate with macros.

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.
  - openCL major memory types: `__local`, `__global`, `__constant`
  - Mixed arguments: 1 version per combination:

```
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __global const uint32_t * b)
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __local const uint32_t * b)

secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __constant const uint32_t * b)
```

  - No C++ templates in openCL: emulate with macros.
  - No `free`, `malloc`: custom memory pools: 0 runtime memory allocation, minimal openCL-statically allocated RAM memory use

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.
  - openCL major memory types: `__local`, `__global`, `__constant`
  - Mixed arguments: 1 version per combination:

```
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __global const uint32_t * b)
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __local const uint32_t * b)

secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __constant const uint32_t * b)
```

  - No C++ templates in openCL: emulate with macros.
  - No `free`, `malloc`: custom memory pools: 0 runtime memory allocation, minimal openCL-statically allocated RAM memory use
- 4 basic modes of running. TO DO: combine (arbitrarily?).
  1. openCL on GPU.
  2. openCL on CPU.
  3. C/C++ on CPU using openCL entry points.
  4. C/C++ direct function call.

Computational engine: cryptography

Core crypto functions: code compiled as both openCL/C++.

- Forked Pieter Wuille's secp256k1 library (used in fabcoin & bitcoin).
- Heavy but formal modifications.
  - openCL major memory types: `__local`, `__global`, `__constant`
  - Mixed arguments: 1 version per combination:

```
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __global const uint32_t * b)
secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __local const uint32_t * b)

secp256k1_fe_mul_inner(uint32_t *r, const uint32_t *a, __constant const uint32_t * b)
```

  - No C++ templates in openCL: emulate with macros.
  - No `free`, `malloc`: custom memory pools: 0 runtime memory allocation, minimal openCL-statically allocated RAM memory use
- 4 basic modes of running. TO DO: combine (arbitrarily?).
  1. openCL on GPU.
  2. openCL on CPU.
  3. C/C++ on CPU using openCL entry points.
  4. C/C++ direct function call.
- TO DO: computation load balancing.

Core crypto: errors.

- No debugger.
- Error hunting: hardcore: not even `printf` available!
- At least 4 types of errors seen so far:
  - Easy: compiles as C/C++, doesn't as openCL or other way round.
  - Easy: runtime errors show in C/C++.
    - Fix: use `printf`, temporarily breaking openCL, debug as usual.
  - Harder: C/C++ runs correctly, openCL crashes.
    - Manually return from offending function.
    - Print memory pool & compare with C/C++ output.
  - Hard core: 1 machine/setup works correctly, another doesn't.
    - Fix?

Core crypto: testing.

- Compare 4 modes of running against one another.
- Generate signatures & verify.
- Tamper signature and invalidate.
- Compare outputs with same input between runs.

Computational engine: C++ driver

- 3 Binary pipes:
  - Command pipe
    - nodejs $\longrightarrow$ C++.
    - nodejs end: non-blocking; C++ end: blocking.
  - Input data pipe
    - nodejs $\longrightarrow$ C++.
    - nodejs end: non-blocking; C++ end: blocking.
  - Data pipe nodejs $\leftarrow$ C++.
    - nodejs $\longleftarrow$ C++.
    - nodejs end: non-blocking; C++ end: blocking.
- TCP protocol: chosen over named/unnamed pipe, std::cin/std::cout (portability).
- Nodejs $\leftrightarrow$ C++ communication local: no networking, no authentication, no encoding.

Nodejs: networking and management

- TO DO: Networking.
    - p2p discovery (FAB RPC call, Nader's smart engine, ...)
    - Kanban id: which peers are running Kanban? Quick options:
        1. Use FAB rpc: `getPeerInfo`, get IP addresses; connect on separate port to ask whether they run Kanban.
        2. Alternatively, add Kanban discovery information to `getPeerInfo`.
        3. Alternatively, make new RPC function, e.g., `getKanbanPeerInfo`.
        4. Alternatively, proposals? Do Kanban p2p in a brand new way or as part of FAB RPC calls?
- Data.

To do list for next week(s)

Immediate to do list.

1. Figure out crypto bug(s) causing inconsistent runs across machines (work machines works correctly, laptop doesn't).
2. At the moment, tests are parallelized, node runtime is not. Fix.
3. Add timing tests for
   - Signature on the CPU.
   - Signature verification on the CPU and GPU.
   - Public key generation on the CPU and GPU.
4. Bootstrap a powerful machine for all tests.

To do list for the near future.

1. Start signing and verifying actual transactions.
2. Implement in-RAM data search.
3. p2p network discovery.
4. Write heavy test suites.
5. Prepare detailed benchmarks.