Viktor Krapivenskiy
Curriculum Vitae

About

Regarded by many as a computer programmer. Interested in techniques of writing clean and maintainable code, software design, compilers, parallel programming, systems programming, computational number theory. Author of a number of side projects and papers.

Skills

Software design · Algorithms and data structures · C · C++ · Go · Python · Lua · JavaScript · x86-64 assembly.

Experience

2021—present · Software architect (private company) · Developed market data providers for multiple exchanges, programs to perform algorithmic trading on multiple exchanges, programs for low-latency transmission of market data over the network, and other tools, in C · Implemented a fast JSON parser in C · Implemented efficient parallel calculation of a digital signature based on Pedersen hash, needed for dYdX cryptocurrency exchange, in x86-64 assembly and C · Implemented a fast emulator of EVM programs to calculate price slippage for a given amount for SushiSwap, Uniswap v2 and v3 pools, in x86-64 assembly and C.

2020 · Go developer (contract with Offscale) · Developed goffkv (goffkv-consul, goffkv-zk, goffkv-etcd) — a rewrite of liboffkv in Go.

2019 · Software developer (contract with Fantom foundation) · Developed tools for internal use.

2019 · Software architect (contract with Sikoba Research) · Implemented support for LLVM in the verifiable computation framework isekai (Crystal). See the following articles for more information:

- Isekai LLVM update #1;
- Isekai LLVM update #2: conditionals and loops;
- Isekai LLVM: final update.

2019 · C++ developer (contract with Offscale) · Developed liboffkv, a uniform interface for distributed key-value storages, in a team of four; implemented C bindings; made a contribution to ppconsul: transactions support (C++).

2018 · Software architect (private company) · Implemented bots and various utilites for analysis of order flow and trading on a number of cryptoexchanges (Python, MySQL).

2017 · Summer of Code Intern (Google) · Implemented Lua scripting for the strace project (C, Lua).

Awards

2016 Prizewinner of the All-Russian Olympiad in Informatics, Finals
2016 Gold winner of the Individual Olympiad of School Students in Informatics and Programming, Finals
2017 4th place in "LAToken hackathon": smart contract for tokenization of different kinds of assets
2018 1st place in "Global Changers" hackathon: client support bot system
2018 1st place in "IDACB & CryptoBazar hackathon": chat based on proxy re-encyption protocol
2018 1st place in "Phystech.Genesis" hackathon: mobile application for traveling
2018 3rd place in "CryptoBazar Serial Hacking: October": PoC software raytracer using Intel SGX
2018 1st place in "CryptoBazar Serial Hacking: November": LLVM IR interpreter with register-based VM
2018 Mentorship of two teams at "CryptoBazar Serial Hacking: December" that took 2nd—3rd places
2019 1st place in "CryptoBazar Serial Hacking: Grand Finale": network traffic record/replay tool
2020 2nd place in "VirusHack": automatic detection of deviations in a video stream

Projects

2016—present **luastatus**, a universal status bar content generator

luastatus is a universal status bar content generator. It allows the user to configure the way the data from event sources is processed and shown, with Lua. Its main feature is that the content can be updated immediately as some event occurs, be it a change of keyboard layout, active window title, volume or a song in your favorite music player (provided that there is a plugin for it) — a thing rather uncommon for tiling window managers. Its motto is:

> No more heavy-forking, second-lagging shell-script status bar generators!

It has a modular architecture, supporing plugins for providing data and barlibs for interacting with different status bars. It supports *i3wm*, *dwm*, *lemonbar*, *dzen*/*dzen2*, *xmobar*, *yabar*, *dvtm*, and others.

2017 **support for Lua scripting in strace**, Google Summer of Code—2017 project

I extended the *strace* project with tampering capability, allowing the user to inject fake syscall results, and read and write the memory of the process being traced.

2020 **libdeci**, an arbitrary-precision decimal arithmetic library for C

This is an arbitrary-precision decimal arithmetic library for C with add-on libraries **libdeci-kara** implementing Karatsuba multiplication, **libdeci-ntt** implementing multiplication via Number-Theoretic Transform (NTT), a variant of Fourier transform, and **libdeci-newt** implementing fast inversion and division using Newton's method. It is faster than the *mpdecimal* library.

2020—present **calx**, a bc-like programming language

calx an attempt to make a modern replacement for bc, while preserving its best features, such as big-decimal numbers and explicit support for interactivity in the language. It is a full-fledged programming language with functions, local and global variables, lists, dicts, strings, etc.

2020 **"Speeding up decimal multiplication"**, a research project (**ArXiv.org URL**).

This research project achieves a 3x—5x speedup over the *mpdecimal* library. The paper describes the implementation and discusses further possible optimizations. It also present a simple cache-efficient algorithm for in-place $2^n \times n$ or $n \times 2^n$ matrix transposition, the need for which arises in the "six-step algorithm" variation of the matrix Fourier algorithm, and which does not seem to be widely known. Another finding is that use of two prime moduli instead of three makes sense even considering the worst case of increasing the size of the input, and makes for simpler answer recovery.

2022 **FiWiA**, a generator of x86-64 machine code for fixed-width multi-word arithmetics

The need for fixed-width multi-word arithmetics frequently arises in cryptography. In this setting, full unroll is usually desirable in two reasons: loop overhead and the need to pass the carry flag to the next iteration, which, without unrolling, would have to be done via saving the carry in a register and restoring it in the next iteration, which is suboptimal in performance. *fiwia* generates fully unrolled x86-64 assembly for fixed-width arithmetic operations, such as: addition/subtraction, masked addition/subtraction, negation, comparison, multiplication, bit shifts.

2022 **sloppy-json**, span-oriented C JSON parser with an external preprocessor to sweeten the process of parsing

Ergonomic parsing of JSON in C is quite a challenging task: virtually all existing parsers are either:

- DOM-like (e.g. *cJSON*): they convert JSON to "objects" with **dynamically allocated** arrays and dictionaries, which the user is supposed to fetch data from and then destroy the "objects"; or
- SAX-like (e.g. *YAJL*): they allow the user to iterate over JSON, which in practice means a lot of callbacks, which are tedious to write and lead to sufficient performance loss, because the callback functions are called by pointer and cannot be inlined.

In settings where we cannot afford dynamic memory allocation, e.g. when tail latency is important, DOM-like parsers cannot be used. Instead of focusing on JSON objects, *sloppy-json* instead operates on spans (a span is a pair of pointer and length) which, when parsed, represent JSON objects. It provides functions to classify a span, iterate over spans representing arrays and dictionaries (the iterator is a span or a pair of spans), convert spans into numbers, unescaping string spans, and everything else needed to "parse" JSON. It does not allocate any memory. It also comes with an external preprocessor written in python to "sweeten" the process of parsing. It also provides some utility functions to generate JSON (e.g. formatting numbers and escaping strings).

2025 **"glass: ordered set data structure for client-side order books"**, a paper

In this paper I present my implementation of ordered set, specifically for maintaining order books, based on a trie. It only supports integer keys and is optimized for market data, namely for what is called "sequential locality" in the paper. The following is the list of what I believe to be novelties:

- *cached path* to exploit *sequential locality* in market data, and fast truncation thereof on erase operation;
- a hash table (or, rather, a *cache table*) with hard $O(1)$ time guarantees on any operation to speed up key lookup (up to a pre-leaf node);
- hardware-accelerated "find next/previous set bit" operations with **BMI2** instruction set extension on x86-64;
- order book-specific features: the preemption principle and the tree restructure operation that prevent the tree from consuming too much memory.

The following speedups are achieved over C++'s standard std::map container: 6x—20x on modifying operations, 30x on lookup operations, 9x—15x on real market data, and a more modest 2x—3x speedup on iteration.

2025—present **jjhash**, a simple string hash function that has better speed and statistical qualities than FNV-2

jjhash is a simple string hash function. It is:

- faster than FNV-2 (6x faster on pointer-and-length strings, 3x faster on null-terminated strings);
- better than FNV-2 on statistical properties, especially for large power-of-two hash table sizes;
- streamable (i.e. supports fast calculation of the hash of concatenation $A + B$ by the contents of B and the hashing state of A);
- has both 32-bit and 64-bit variants.

References

shdownnine@gmail.com
https://github.com/shdown
https://www.linkedin.com/in/shdownnine