High-Performance RegEx Matching with Parabix

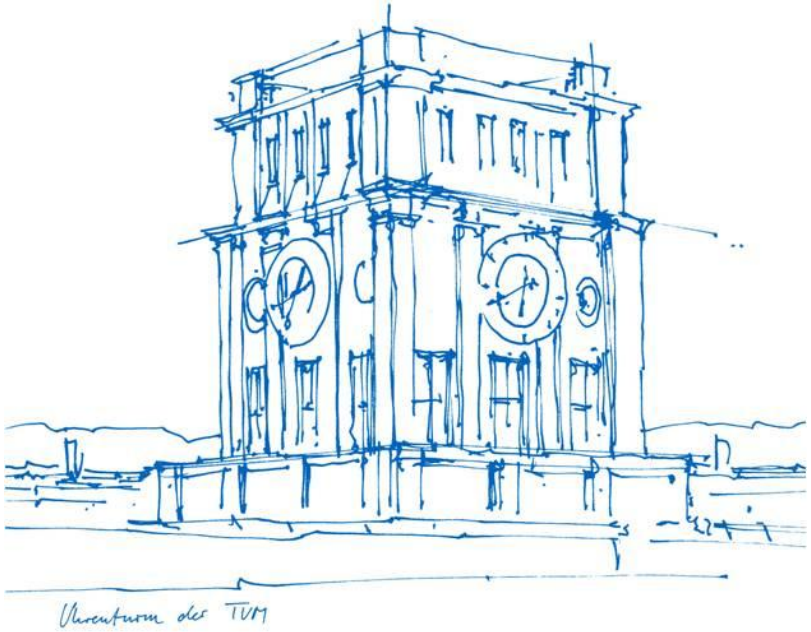
Elmi Ahmadov

Technische Universität München

TUM Informatics

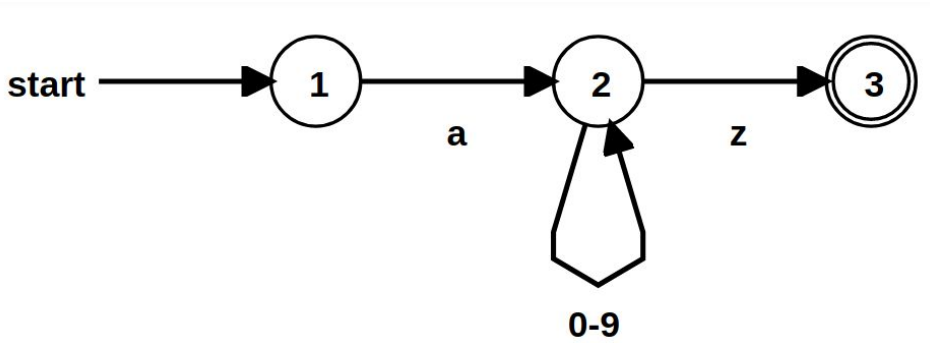
Presentation for Database Implementation

22.02.2022



Why do we need another approach for RegEx?

- Traditional approaches of Finite State Automata for RegEx process a single byte/character at a time.
- They are also hard to parallelize.



- What is Parabix (Parallel Bit Streams)?
  - Parabix instead processes the input data with bitwise operations.
  - Can process a chunk of the input data at the same time.

Compare Parabix with ripgrep

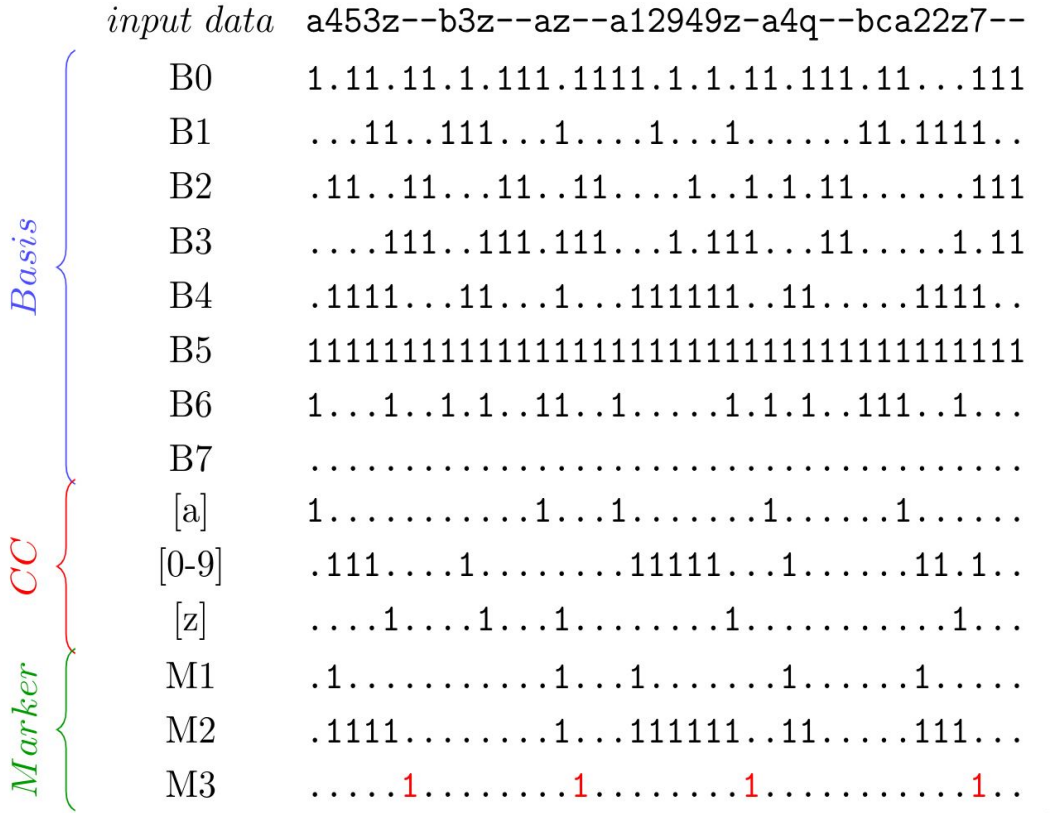
- ripgrep is one of my favorite Linux tool.
- It's not a detailed comparison but should give an idea how fast Parabix (with LLVM) is!

```
$ ./compare-with-ripgrep.sh
rg --count-matches "a[0-9]*z" ../input/1gb.txt
63111049
8.42s
./parabix_llvm "a[0-9]*z" ../input/1gb.txt
63111049
1.77s
```

Parabix - Parallel bit streams

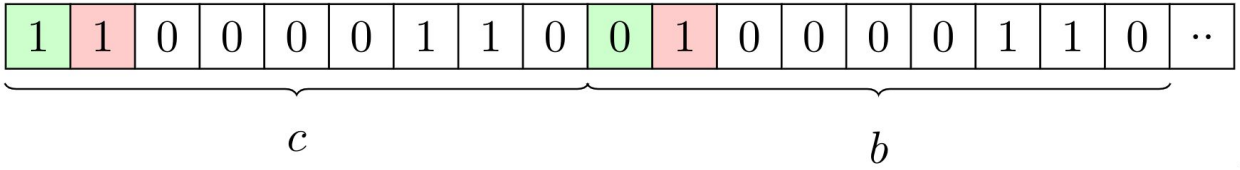
- There are three main bit streams:
  - ***Basis Bit streams***
  - ***Character Class Bit streams***
  - ***Marker Bit streams***

- Let's build bit streams for the given input and RegEx `(a[0-9]*z)`

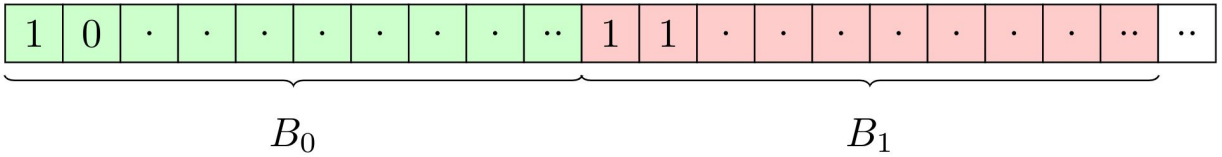


Parabix - Basis bit streams

- Processing single byte of input data at a time is inefficient. The solution is transposing a bit matrix.
- Memory representation of input data



- Memory representation after the transpose operation



Parabix - Character class bit streams

- Matching a single character class is easy, e.g. **CC = [a]**
  - CC[a] = 01100001 = $(b_6 \ \& \ \sim b_7) \ \& \ (\sim b_4 \ \& \ b_5) \ \& \ (\sim b_2 \ \& \ \sim b_3) \ \& \ (b_0 \ \& \ \sim b_1)$
- How to match a range like **[0-9]**? General idea is splitting the given range into 3 parts:
  - common part: never changes = $!(b_7 \mid b_6) \ \& \ (b_5 \ \& \ b_4)$
  - low part: low bits changes rarely = b_3
  - high part: high bits changes frequently = $(b_2 \mid b_1)$
  - final result = $!(b_7 \mid b_6) \ \& \ (b_5 \ \& \ b_4) \ \& \ !(b_3 \ \& \ (b_2 \mid b_1))$

0	00110000
1	00110001
2	00110010
3	00110011
4	00110100
5	00110101
6	00110110
7	00110111
8	00111000
9	00111001

Parabix - Marker bit streams

- Marker bit stream shows the bit past the last match
- We have two main operations: **`Advance`** and **`MatchStar`**

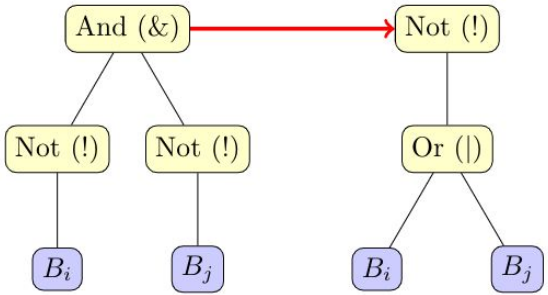
```
def RegExMatching(Input, RegEx (a[0-9]*z))
    BuildBasisBitStreams(Input)
    BuildCCBitStreams(RegEx)
    M1 = Advance(M0, CC([a]))
    M2 = MatchStar(M1, CC([0-9]))
    M3 = Advance(M2, CC([z]))

    return M3
```

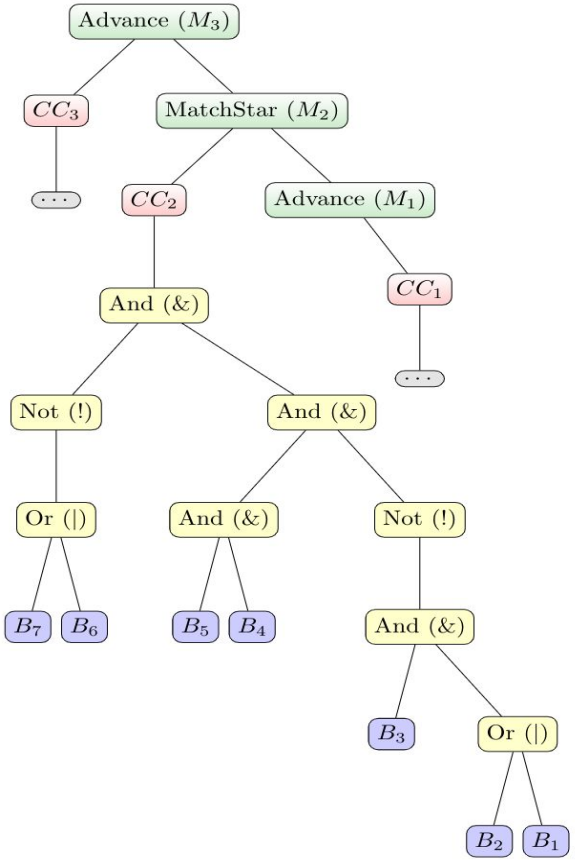
Parabix - SIMD

- Transpose bit matrix with SIMD instructions.
- Each basis bit stream stores either 64 or 128 bits (depends on available SIMD instructions on CPU).
- This allows use to perform 64 bytes of input data at a time using SIMD instructions for
  - and
  - add
  - or
  - xor
  - not

Parabix - Code gen (LLVM)

- We build AST from basis bit streams and character class bit streams and marker bit streams.
- All CC and Marker nodes internally use `And, Or, Xor, Not` and `B` operations.
- Apply basic optimizations to reduce the number of bitwise operations.



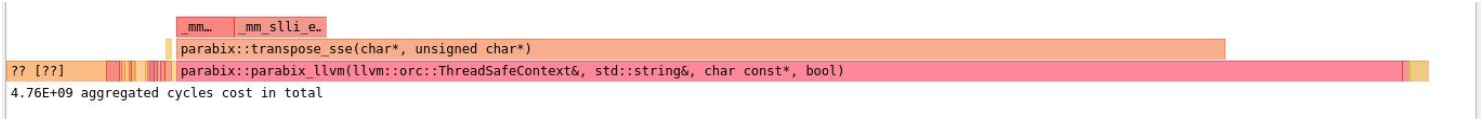


Parabix - Evaluation

- Parabix with LLVM code gen shows a significant improvement (~15-40x) over.
  - C++ standard RegEx library (std::regex)
  - DFA (Custom implementation).
- LLVM compilation time is **~10ms**.

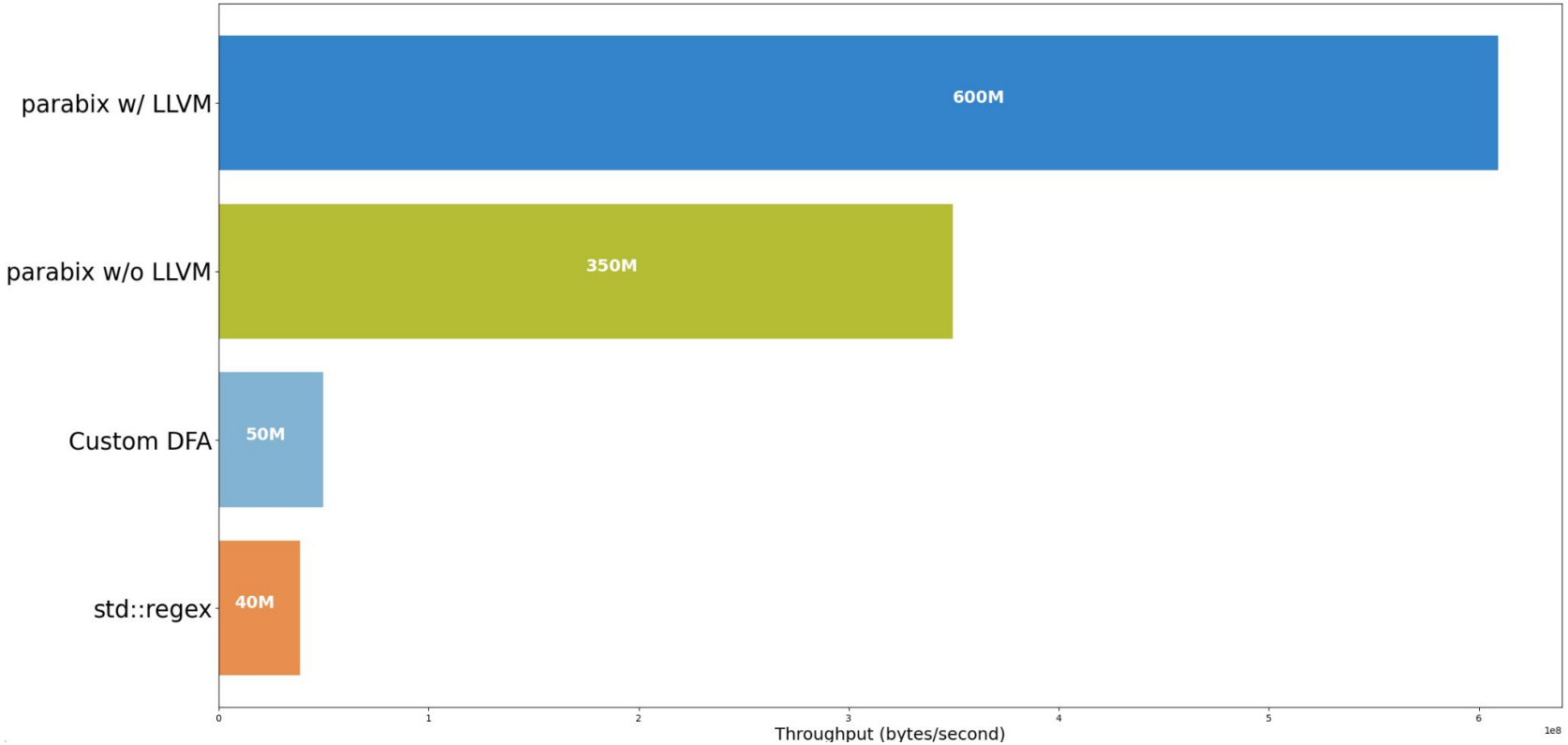
```
// Transpose to 2 nybble streams
lo_nybble0 = hsimd::packl(s0, s1);
lo_nybble1 = hsimd::packl(s2, s3);
lo_nybble3 = hsimd::packl(s4, s5);
lo_nybble4 = hsimd::packl(s6, s7);
hi_nybble0 = hsimd::packh(s0, s1);
hi_nybble1 = hsimd::packh(s2, s3);
hi_nybble3 = hsimd::packh(s4, s5);
hi_nybble4 = hsimd::packh(s6, s7);
// Transpose 2 nybble streams to 4 bit-pair streams.
bit01pair_0 = hsimd::packl(lo_nybble0, lo_nybble1);
bit01pair_1 = hsimd::packl(lo_nybble2, lo_nybble3);
bit23pair_0 = hsimd::packh(lo_nybble0, lo_nybble1);
bit23pair_1 = hsimd::packh(lo_nybble2, lo_nybble3);
bit45pair_0 = hsimd::packl(hi_nybble0, hi_nybble1);
bit45pair_1 = hsimd::packl(hi_nybble2, hi_nybble3);
bit67pair_0 = hsimd::packh(hi_nybble0, hi_nybble1);
bit67pair_1 = hsimd::packh(hi_nybble2, hi_nybble3);
// Transpose 4 bit-pairs streams to 8 bit streams.
bit0 = hsimd::packl(bit01pair_0, bit01pair_1);
bit1 = hsimd::packh(bit01pair_0, bit01pair_1);
bit2 = hsimd::packl(bit23pair_0, bit23pair_1);
bit3 = hsimd::packh(bit23pair_0, bit23pair_1);
bit4 = hsimd::packl(bit45pair_0, bit45pair_1);
bit5 = hsimd::packh(bit45pair_0, bit45pair_1);
bit6 = hsimd::packl(bit67pair_0, bit67pair_1);
bit7 = hsimd::packh(bit67pair_0, bit67pair_1);
```

cycles	instructions	L1-misses	LLC-misses	branch-misses	task-clock	scale	IPC	CPUs	GHz
11.58,	40.03,	0.02,	0.03,	0.00,	3.17,	1048576032,	3.46,	1.00,	3.65



Hotspot - Linux perf GUI tool

Parabix - Benchmark



RegEx is `(a[0-9]*z)`

Thanks! Questions?