Blond GSOC 2016

BLonD code optimization strategy for parallel and concurrent architectures

By Oleg Iakushkin

26.08.2016

CMake to build external libraries

Problem:

- Windows expects Debug and Release configurations
- Linux does not

Performance is a concern so one must be able to forward compilation arguments to libraries

Sub objective: Remove git “submodules” that are realy complex external projects

Main objective: Fluent development for contributors from other platforms

CMake to build external libraries

- Solution: always build release for linux, debug+release for MSVC
- Call Cmake from Cmake

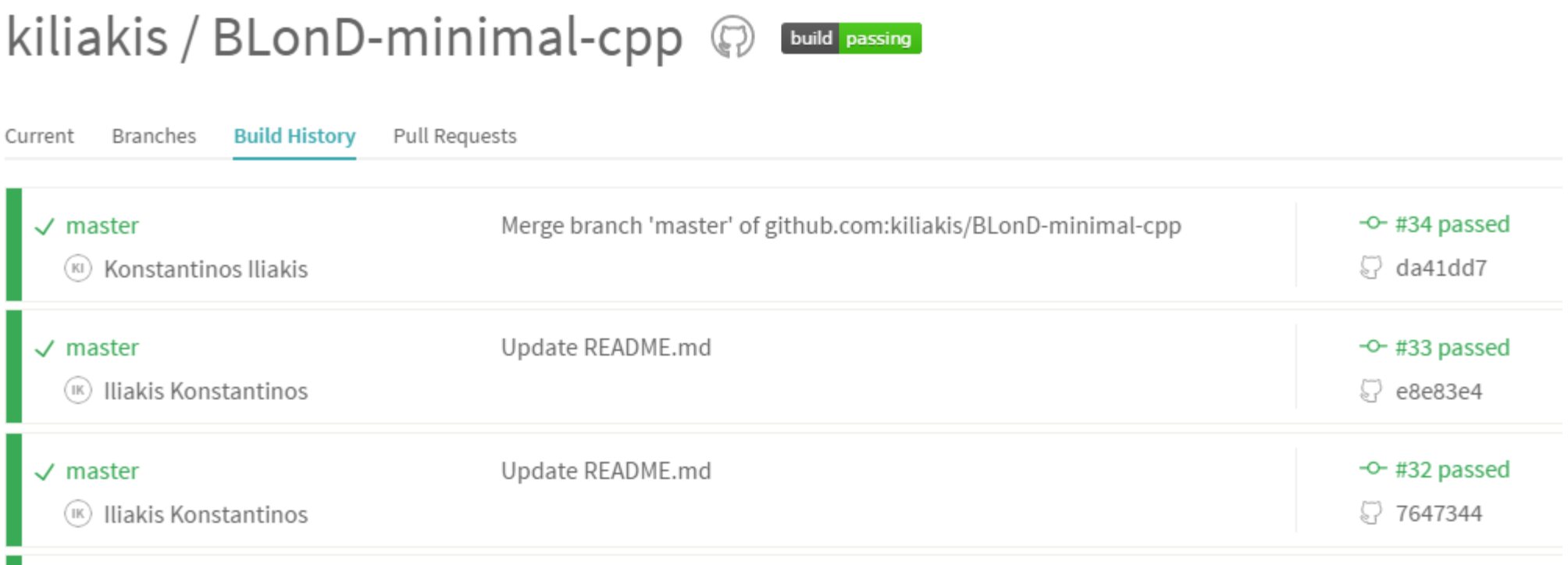
To build depends only once on configuration

```
execute_process(
        COMMAND cmake --build .  --config Release
        WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}/external/build
)
if(WIN32 AND NOT MINGW)
    execute_process(
        COMMAND cmake --build .  --config Debug
        WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}/external/build
    )
endif()
```

```
execute_process(
        COMMAND ${CMAKE_COMMAND}
        #options
        -DWITH_FFTW=${WITH_FFTW}
        -DWITH_BENCHMARK=${WITH_BENCHMARK}
        -DWITH_GOOGLETEST=${WITH_GOOGLETEST}
        #destinations
        -DINSTALL_DIR=${EXTERNAL_INSTALL_DIR}
        -DCOMPILE_FLAGS=${COMPILE_FLAGS}
        -DGOOGLETEST_ROOT=${GOOGLETEST_ROOT}
        -DBENCHMARK_ROOT=${BENCHMARK_ROOT}
        -DGOOGLETEST_ROOT=${GOOGLETEST_ROOT}
        #forward all collected compilation arguments
        -DCMAKE_BUILD_TYPE=${CMAKE_BUILD_TYPE} # relevant only for Make-like build files
        -DCMAKE_CXX_FLAGS=${CMAKE_CXX_FLAGS}
        -DCMAKE_CXX_FLAGS_DEBUG=${CMAKE_CXX_FLAGS_DEBUG}
        -DCMAKE_CXX_FLAGS_RELEASE=${CMAKE_CXX_FLAGS_RELEASE}
        -DCMAKE_EXE_LINKER_FLAGS_RELEASE=${CMAKE_EXE_LINKER_FLAGS_RELEASE}
        -DCMAKE_EXE_LINKER_FLAGS_DEBUG=${CMAKE_EXE_LINKER_FLAGS_DEBUG}
        -v
        -G "${CMAKE_GENERATOR}"
        ${PROJECT_SOURCE_DIR}/external/
        WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}/external/build
)
```

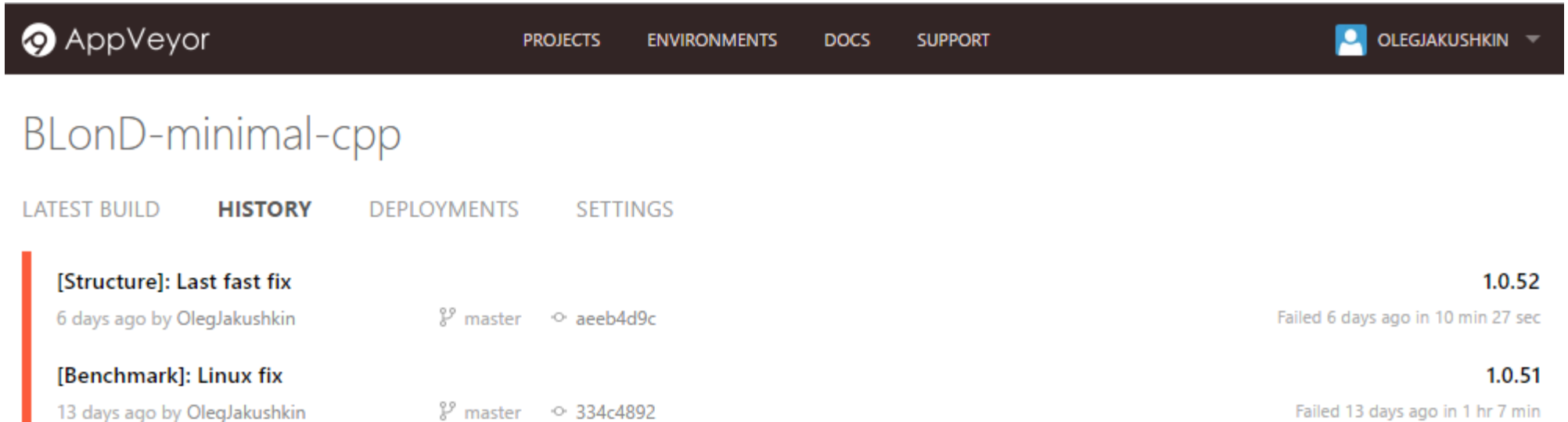
Travis (public CI SaaS)

- Script that rebuilds, retests, code on each commit
- Similar to GitLab CI (for future reference)
- Build history and GitHub badges



AppVeyor (public Windows CI SaaS)

- Script that rebuilds, retests, code on each commit
- Similar to Travis CI (for future reference)
- Build history and GitHub badges
- Tests do fail. Log 60k lines (browsable only in firefox)
  4k depends build log; 6k Blond build (with tests and demos)



Coveralls (public Test cover analysis SaaS)

- Runs from gcc test and profiler files (same files used for PGO)
- Shows line hits

COVERAGE	JOB	FILES COVERED	RAN	
61.88	34.1 (BUILD_TYPE=Debug)	30	about 3 hours ago	TRAVIS JOB 34.1
61.88	34.2 (BUILD_TYPE=Release)	30	about 3 hours ago	TRAVIS JOB 34.2

FILES

SEARCH:

ALL 30 | CHANGED 12 | SOURCE CHANGED 2 | COVERAGE CHANGED 12

SHOW 100 ENTRIES

COVERAGE	FILE	LINES	RELEVANT	COVERED	MISSED	HITS/LINE
43.96	beams/Slices.cpp	564	182	80	102	8695918.0 + 375039
92.59	beams/Distributions.h	104	27	25	2	7200381.0 + 800001
70.59	include/utilities.h	288	17	12	5	5209332.0 + 134
94.44	include/math_functions.h	375	108	102	6	4075064.0 + 236
71.43	input_parameters/GeneralParameters.cpp	196	112	80	32	3903741.0 + 1.0

Documentation

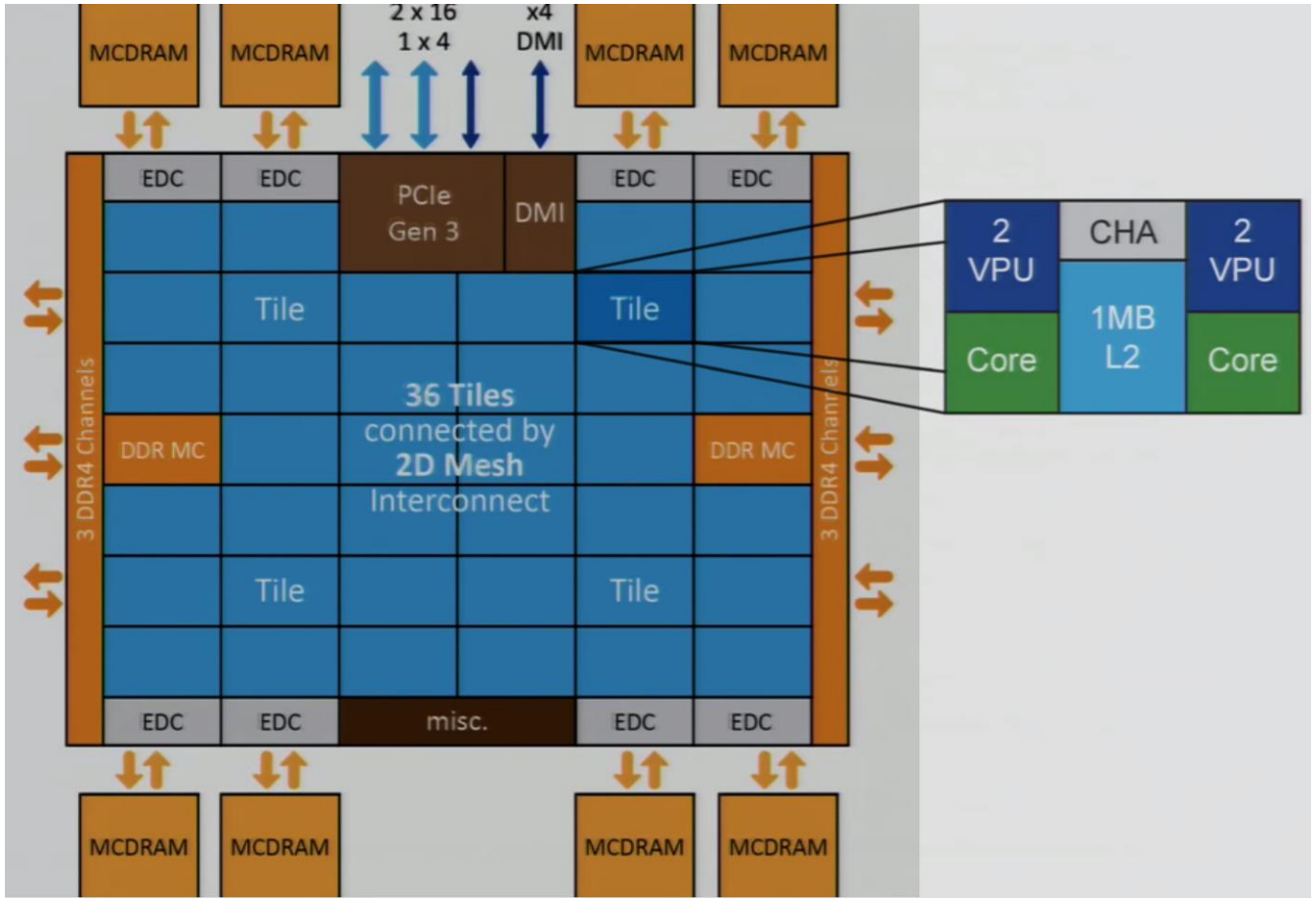
- **FAQ**
- How to add a source file?
- How to Run Benchmarks?
- How to create new Benchmarks?
- How to create documentation?
- How to fork with CI and gh-pages?
- How CI scripts work?
- How to auto-format code on build?
- How to profile code?
- How to add a library?
- **TLDR**
- Project structure
- Cmake Options
- Advanced compilation options

Kernel vs Pipeline vs Mix

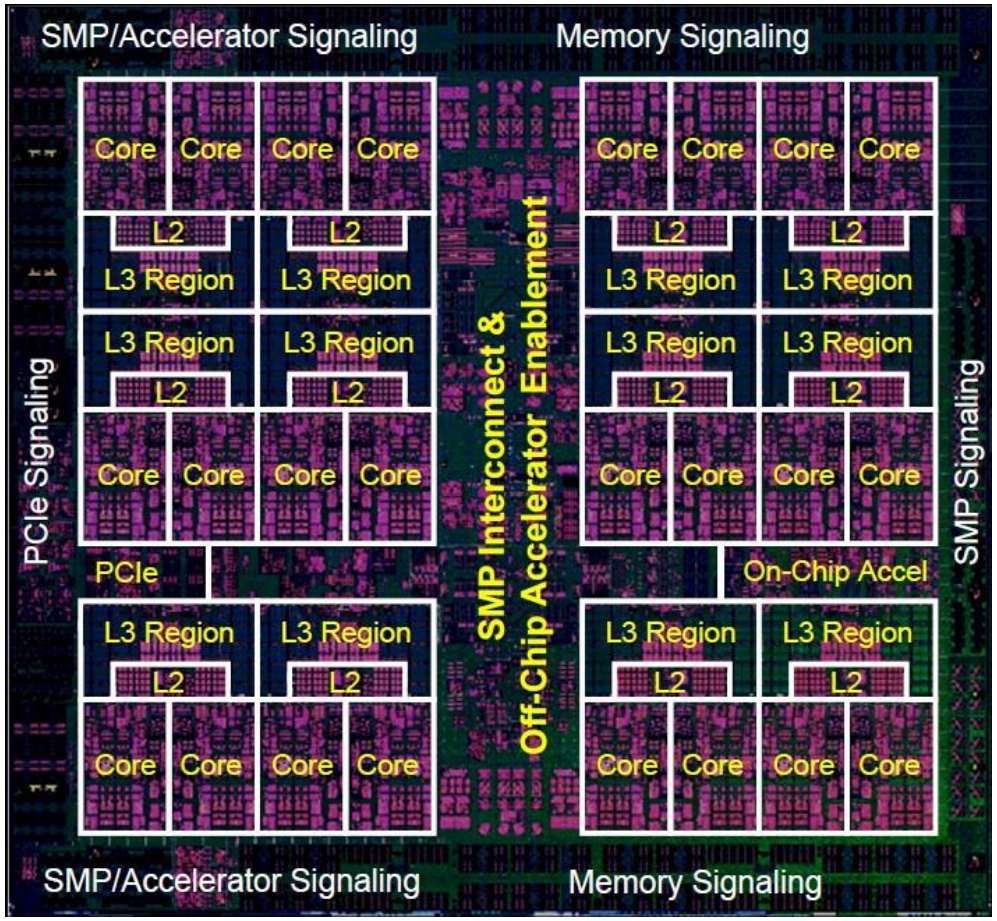
- Compilers act differently on various platform\architecture combos
- CPUs and Coprocessors make fine-tuning of a library hard – a library can be used in different settings and clusters
- There is an architectural diversity in TOP500

			Cores	Rmax (TFlop/s)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2

Architectures (CPU)



Knights Landing [1,2, II.] (Xeon Phi)



Power9 [I.]

- L3 – one vs many
- VPU vs RISC
- One GRID vs Pairs

Architectures (GPU)

Many cores, kernel focus
PCIe data transfer from main memory

NVidia (Tesla K20)

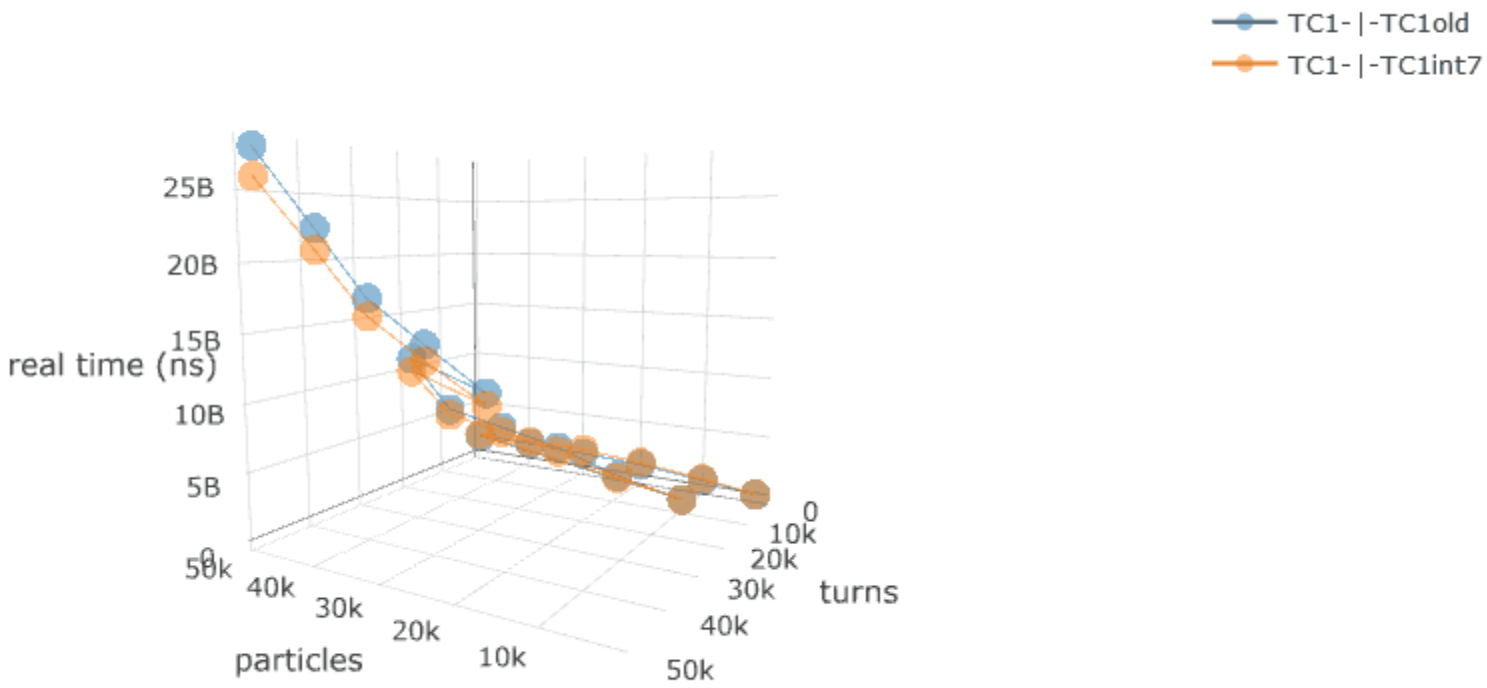
Benchmarking before architectural decisions

- Benchmarks are a way to expend testing of our libraries [IV. V.]
- They can be independent of main library codebase or be integrated into it
- Benchmark results can be compared thus project performance progress can be observed and architectural decisions judged over time
- Optimizations for specific new hardware can be achieved faster dew to easy to detect pressure points

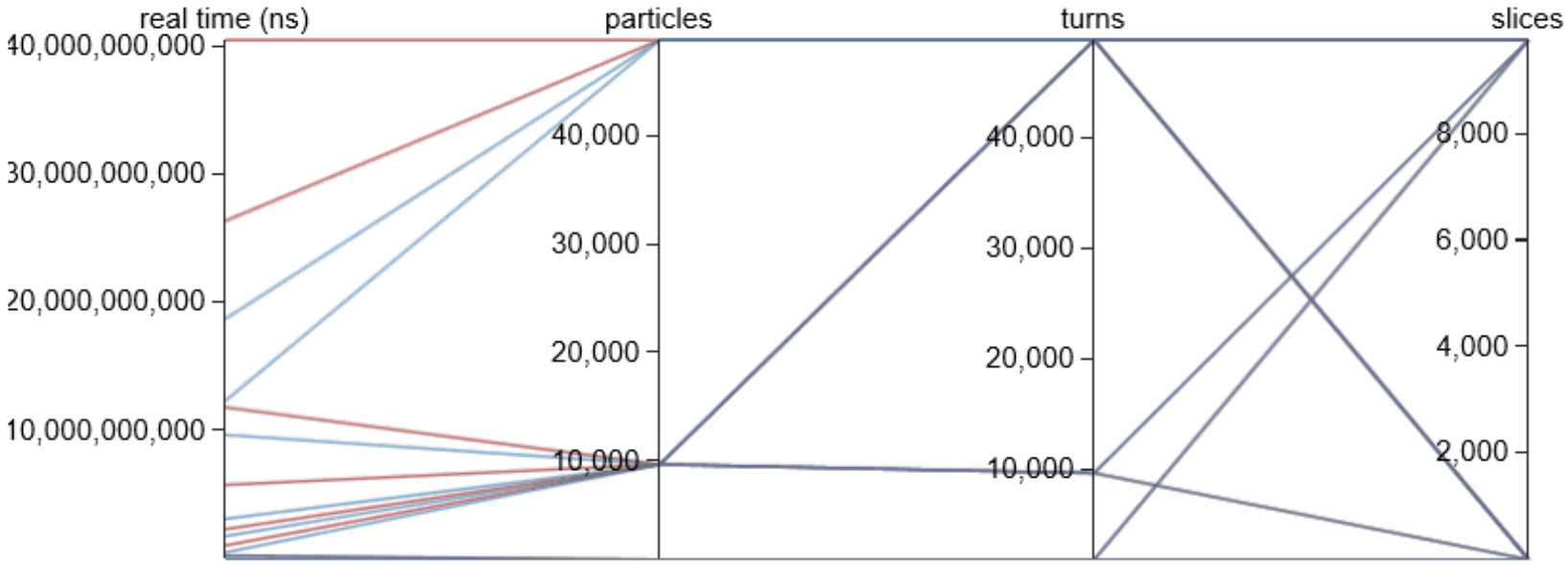
Benchmarks for BLoND

- Benchmarks run same function many times with different predefined data (e.g. array of pairs from {100, 500} up to {1000000, 1000000})
- Created a small benchmark visualization tool
  - 1D data + run-time = 2d lines plot
  - 2D data + run-time = 3d scatter plot
  - 3D+ = Parallel Coordinates
    - Run-time is first bar
    - Complex to grasp yet scalable
    - Each test – one line, tests group – one color
- Updated code structure to fit in benchmarks
  - They are not required for allmost everyone to build\run
  - They can require specific compiler/platform and thus are mostly optional

3d viewer


TC1-|-TC1old
TC1-|-TC1int7
real time (ns)
25B
20B
15B
10B
5B
0
particles
50k
40k
30k
20k
10k
turns
0
10k
20k
30k
40k
50k

Nd viewer



real time (ns)	particles	turns	slices	curve_name
40430219571	48828.1	48828.1	9765.63	3d-windows-i7-|-BMTC1Acceleration
26366605918	48828.1	48828.1	10	3d-windows-i7-|-BMTC1Acceleration
18751141548	48828.1	48828.1	9765.63	3d-hybrilit-|-BMTC1Acceleration
12388570627	48828.1	48828.1	10	3d-hybrilit-|-BMTC1Acceleration
11883913676	9765.63	48828.1	9765.63	3d-windows-i7-|-BMTC1Acceleration

How to add visualization?

1. **Run a benchmark** (save .json output) e.g.
   `./benchTC1_Acceleration` **`--benchmark_out=TC1_my.json`**
   `--benchmark_repetitions=3`
2. **Edit benchmarks_list.js** with your benchmark name, file, benchmark data axes names. One view can display multiple files (e.g. from different clusters)

```
benchmarks = [
    {name: "TC1-3d_data", ref: ["./3d-hybrilit.json", "./3d-windows-i7.json"], axes: ["particles", "turns", "slices"]},
    {name: "TC1-2d_data", ref: "./TC1.json", axes: ["particles", "turns"]},
];
```

3. **Run benchmark_browser.html** in Firefox (or from a web server)

General Benchmark problems

- Benchmarks contain code that is for testing of possible options that might not be merged into current main project source code
- MPI requires separate compiler mpicc
- Intel compiler is required for #pragma SIMD
- CUDA has nvcc and has a set of unique libraries
- Some benchmarks can require different libraries GSL or compilation options
- CI integration of specific benchmark code can be a giant pain.

Architectural solution

- Benchmarks
  - Prototypes
    - XviaOpenMP
    - XviaOpenCl
  - benchmarkX.cpp
  - benchmarkX.cmake

- BenchmarkX.cmake
  - cheks for libraries\includes\compilers
  - Sets defines for benchmarks
- BenchmarkX.cpp
  - Expands unit test and benchmarks default implementation performance
  - Additional unit test implementations are guarded by #ifdefs resolved at Cmake stage

Creating benchmarks that can be peer-reviewed

1. Take a unit test
2. Keep unit-test structure
3. Keep unit-test data access
4. Select a part that you want to speed up
5. Isolate data inputs and test results e.g:
   - In: RfP->omega_RF, RfP->voltage;
   - Out: Beam->dE;
6. Make a benchmark that will use test case in one of the runs to always keep the code work-proofed
7. Benchmark old code
8. Create a prototype that can be a class or function accepting isolated input data

This allows us to keep benchmarks and evaluate them on new\old platforms

Why this is needed?

- Clusters are different in:
  - OS,
  - nodes count and topology,
  - Accelerators (K20, K80, phi, none)
  - IBM, Intel, ARM, Sunway CPUs
- Platforms and compilers are different
  - MSVC vs GCC vs ICC
  - With and without optimizations

Different platforms show different results!

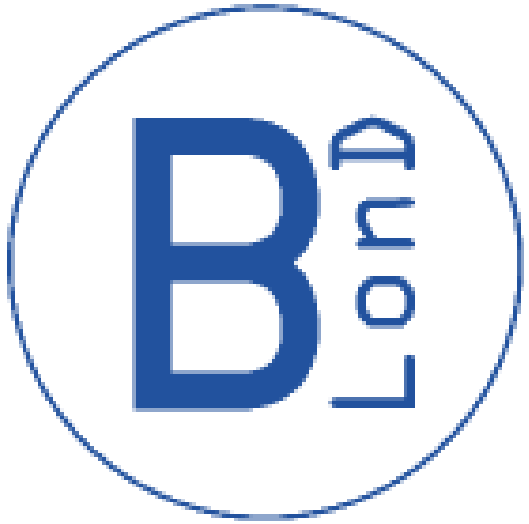
Google Summer of Code

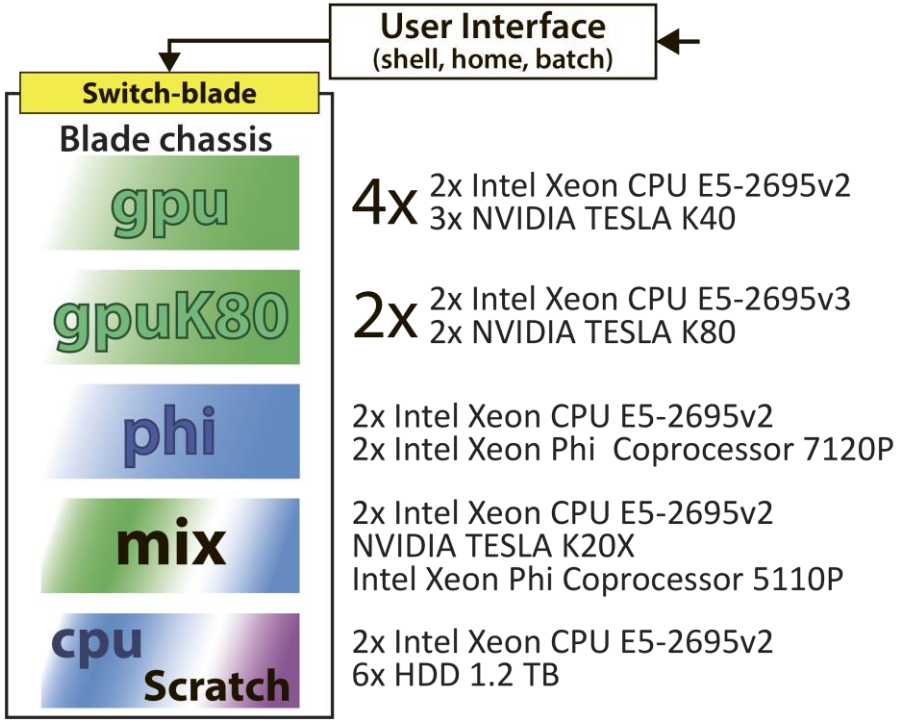
What I've done during this summer for C++ version of Blond

- CI testing
  - Linux
  - Windows
- External libraries auto-build
- Blond code Windows port
- Automated documentation generation
  - Coverage
  - Unit-tests
  - Benchmarks
- Started prototypes-benchmarks for TC1

Acknowledgements

- CERN SFT **BLonD** team developers are the best, I was glad to work with you at Google Summer of Code 2016!=)
- Microsoft Azure for research grant I used for initial Linux testing, and St. Petersburg University faculty cluster used for MPI testing
- Cluster used for Phi, GPU benchmarks: Heterogeneous cluster at LIT/JINR





References

YouTube:

I. RichReport

II. ANL Training

III. Charm++

IV. CppCon

V. BoostCon

Books:

[1] McCool, M.D., Robison, A.D. and Reinders, J., 2012. *Structured parallel programming: patterns for efficient computation*. Elsevier.

[2] Jeffers, J. and Reinders, J. eds., 2014. *High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches*.

[3] Jeffers, J. and Reinders, J., 2015. *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches*. Morgan Kaufmann.

[4] Arndt, J., 2010. *Matters Computational: ideas, algorithms, source code*. Springer Science & Business Media.

[5]Fabian R., 2016+ Data-Oriented Design book

Discussion

Thank you for your attention