

Benchmarking

Exercises

Getting Catch2 environment

```
your-laptop > ssh <your-cern-account>@lxplus9.cern.ch
# eulisse at lxplus903.cern.ch in ~ [13:45:14]

lxplus903.cern.ch > source ~eulisse/public/csc/exercises.sh
MODULEPATH=/cvmfs/alice.cern.ch/e19-x86_64/Modules/modulefiles:/cvmfs/alice.cern.ch/etc/toolchain/modulefiles/e19-x86_64 ; export MODULEPATH

lxplus903.cern.ch > module load Catch2/v3.7.0-3
Loading Catch2/v3.7.0-1
  Loading requirement: BASE/1.0 jq/v1.6-alice1-2 grid-base-packages/default Toolchain/GCC-v13.2.0 GCC-Toolchain/v13.2.0-alice1-1

lxplus903.cern.ch > pkg-config --cflags --libs catch2-with-main
-I/cvmfs/alice.cern.ch/e19-x86_64/Packages/Catch2/v3.7.0-3/include -L/cvmfs/alice.cern.ch/e19-x86_64/Packages/Catch2/v3.7.0-3/lib -lCatch2Main -lCatch2

lxplus903.cern.ch > type c++
c++ is /cvmfs/alice.cern.ch/e19-x86_64/Packages/GCC-Toolchain/v12.2.0-alice1-4/bin/c++
```

Preconfigured build / profiling environment in case you need one.

Getting the sources for the exercises

```
lplusplus.cern.ch > git clone https://github.com/ktf/csc-2024-benchmarking-profiling
lplusplus.cern.ch > find csc-2024-benchmarking-profiling -name "*.cc"
csc-2024-benchmarking-profiling/01-busy-loop/busy-loop-1.cc
csc-2024-benchmarking-profiling/01-busy-loop/busy-loop-2.cc
csc-2024-benchmarking-profiling/01-busy-loop/busy-loop-iteration-dep-correct.cc
csc-2024-benchmarking-profiling/01-busy-loop/busy-loop-iteration-dep.cc
csc-2024-benchmarking-profiling/01-busy-loop/busy-loop-return-value-dep.cc
csc-2024-benchmarking-profiling/01-busy-loop/busy-loop.cc
csc-2024-benchmarking-profiling/01-busy-loop/callstack-example.cc
csc-2024-benchmarking-profiling/02-benchmarking/01-vector-trivial-solution.cc
csc-2024-benchmarking-profiling/02-benchmarking/01-vector-trivial.cc
csc-2024-benchmarking-profiling/03-memory-allocation/01-trivial-leak.cc
csc-2024-benchmarking-profiling/03-memory-allocation/02-array-leak.cc
csc-2024-benchmarking-profiling/03-memory-allocation/03-padding.cc
csc-2024-benchmarking-profiling/03-memory-allocation/04-cost-std-map.cc
csc-2024-benchmarking-profiling/03-memory-allocation/05-cost-vector-vector.cc
```

First exercise is the 02-benchmarking/01-vector-trivial.cc

Getting the sources for the exercises

```
lxplus903.cern.ch > cat csc-2024-benchmarking-profiling/02-benchmarking/01-vector-trivial.cc
// Here is a simple example which measures lookups from a vector.
//
// - How do measurements change when changing the DATASET_SIZE?
// - How do the measuments change when you change the STRIDE?
// - How many samples do you need to have to have stable results?
// - How do results change when using different DataTypes?
TEST_CASE("Trivial Vector") {
    std::vector<DataType> v;

    BENCHMARK_ADVANCED("Vector Lookup")(Catch::Benchmark::Chronometer meter) {
        for (size_t i = 0; i < DATASET_SIZE; i++) {
            v.emplace_back(i * 2);
        }
        meter.measure([&] {
            for (size_t i = 0; i < ITERATIONS; i += STRIDE) {
                volatile DataType a = v[i % DATASET_SIZE];
            }
        });
    };
}
```

First exercise is the [02-benchmarking/01-vector-trivial.cc](#)

SPOILER

ALERT!

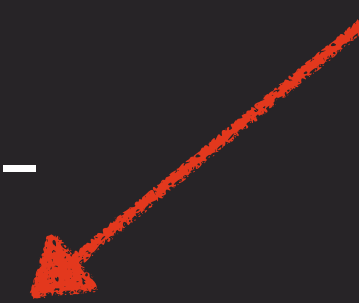
Ex 1: Trivial Vector benchmarking

Vector: 100M lookups

02-benchmarking/01-vector-trivial-solution.cc:37
.....

benchmark name	samples	iterations	mean
2**3,1	5	1	63.3093 ms
2**7,13*4096	5	1	63.3858 ms
2**10,13*4096	5	1	63.234 ms
2**22,13*4096	5	1	92.945 ms
2**30,13*4096	5	1	393.109 ms
2**7,13	5	1	64.5327 ms
2**10,13	5	1	63.3067 ms
2**22,13	5	1	63.3539 ms
2**30,13	5	1	63.4045 ms

Until we stay
inside L1 Cache, size
does not really



Ex 1: Trivial Vector benchmarking

Vector: 100M lookups

02-benchmarking/01-vector-trivial-solution.cc:37
.....

benchmark name	samples	iterations	mean
2**3,1	5	1	63.3093 ms
2**7,13*4096	5	1	63.3858 ms
2**10,13*4096	5	1	63.234 ms
2**22,13*4096	5	1	92.945 ms
2**30,13*4096	5	1	393.109 ms
2**7,13	5	1	64.5327 ms
2**10,13	5	1	63.3067 ms
2**22,13	5	1	63.3539 ms
2**30,13	5	1	63.4045 ms

We start to see
cache hierarchy
effects!

Ex 1: Trivial Vector benchmarking

Vector: 100M lookups

02-benchmarking/01-vector-trivial-solution.cc:37
.....

benchmark name	samples	iterations	mean
2**3,1	5	1	63.3093 ms
2**7,13*4096	5	1	63.3858 ms
2**10,13*4096	5	1	63.234 ms
2**22,13*4096	5	1	92.945 ms
2**30,13*4096	5	1	393.109 ms
2**7,13	5	1	64.5327 ms
2**10,13	5	1	63.3067 ms
2**22,13	5	1	63.3539 ms
2**30,13	5	1	63.4045 ms

With a 1GB buffer,
reading sparsely
becomes very slow....

Ex 1: Trivial Vector benchmarking

Vector: 100M lookups

02-benchmarking/01-vector-trivial-solution.cc:37
.....

benchmark name	samples	iterations	mean
2**3,1	5	1	63.3093 ms
2**7,13*4096	5	1	63.3858 ms
2**10,13*4096	5	1	63.234 ms
2**22,13*4096	5	1	92.945 ms
2**30,13*4096	5	1	393.109 ms
2**7,13	5	1	64.5327 ms
2**10,13	5	1	63.3067 ms
2**22,13	5	1	63.3539 ms
2**30,13	5	1	63.4045 ms

← Sequential reads are not affected, regardless of the size of the buffer

Ex 1: Trivial Vector benchmarking

Vector: 100 lookups

02-benchmarking/01-vector-trivial-solution.cc:67
.....

benchmark name	samples	iterations	mean	
2**3,1	5	264	63.5439	ns
2**7,13*4096	5	264	65.753	ns
2**10,13*4096	5	263	62.6768	ns
2**22,13*4096	5	187	78.1241	ns
2**30,13*4096	5	262	395.073	ns
2**7,13	5	264	62.5023	ns
2**10,13	5	264	68.5629	ns
2**22,13	5	264	64.6818	ns
2**30,13	5	264	63.8295	ns

Reading sparsely still has some overhead, even if the working set fits in memory.

Getting the sources for the exercises

```
lxplus903.cern.ch > git clone https://github.com/ktf/csc-2024-benchmarking-profiling
Cloning into 'csc-2024-benchmarking-profiling'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 27 (delta 5), reused 27 (delta 5), pack-reused 0 (from 0)
Receiving objects: 100% (27/27), 166.59 KiB | 4.50 MiB/s, done.
Resolving deltas: 100% (5/5), done.

lxplus903.cern.ch > source ~eulisse/public/csc/exercises.sh
lxplus903.cern.ch > module load Catch2/v3.7.0-1

lxplus903.cern.ch > cd csc-2024-benchmarking-profiling ; ls
01-busy-loop 02-benchmarking 03-catch2-bench 03-memory-allocation Makefile

lxplus903.cern.ch > make clean && make -j 10 all
```