FOSDEM 2025

Effect of kernel optimizations on HPC workloads performance

Alex Domingo

VUB VRIJE UNIVERSITEIT BRUSSEL

VUB HPC

VLAAMS SUPERCOMPUTER CENTRUM

Vlaanderen is supercomputing

WHO AM I?

Hi! I’m Alex (github: @lexming)

Not a kernel developer

Not even a software developer in C

HPC sysadmin and programmer

- HPC team of Vrije Universiteit Brussel (VUB) since 2019
- PhD in Computational Chemistry
- Maintainer of EasyBuild (easybuild.io): open source software build and installation framework for HPC

HPC IS ABOUT PERFORMANCE, RIGHT?

We want to squish as much performance as possible from the cluster, but do not treat all system layers equally:

User space

- Software compiled from source
- Modern compilers and toolchains
- Native instruction set of the CPU

Kernel space

- Generic Linux kernel images
- Packaged by Linux distributions

HPC IS ABOUT PERFORMANCE, RIGHT?

Performance without stability is useless

Kernel space

- Generic Linux kernel images
- Packaged by Linux distributions

HPC IS ABOUT PERFORMANCE, RIGHT?

Performance without stability is useless

Kernel space

- Generic Linux kernel images
- Packaged by Linux distributions

- Robust kernel images
- Regular security updates
- *Standard* feature set and configuration
- Extensive ecosystem of packages
- Supported by vendors

GOALS

Improve HPC cluster performance by optimizing the Linux kernel and ...

- do not break software already in use in userspace
- do not change kernel features and configuration
- do not break kernel modules (out-of-tree hardware drivers)
- do not introduce vulnerabilities
- do not break provisioning/administration/maintenance tooling

No patching

Keep kernel `.config`

Use `rpmbuild`

→ TLDR make a drop-in replacement for the kernel RPM shipped by Rocky Linux that performs better

BENCHMARKS

- HPCG
- OSU – bandwidth
- OSU – latency
- BLAS Test Level 3
- GROMACS – Single Node
- GROMACS – Multi Node
- CP2k – Single Node
- CP2k – Multi Node

EasyBuild

easybuild.io

ReFrame

reframe-hpc.readthedocs.org

- MiniBUDE – BM1
- MiniBUDE – BM2
- FFTW – Stock
- FFTW – Float+SSE
- MAFFT
- oneDNN
- numpy
- XNNPACK
- Llama.cpp – v3 8B

pts Phoronix Test Suite

phoronix-test-suite.com

Baseline

- 3 runs with kernel from distro
- 3 runs with unmodified self-compiled kernel

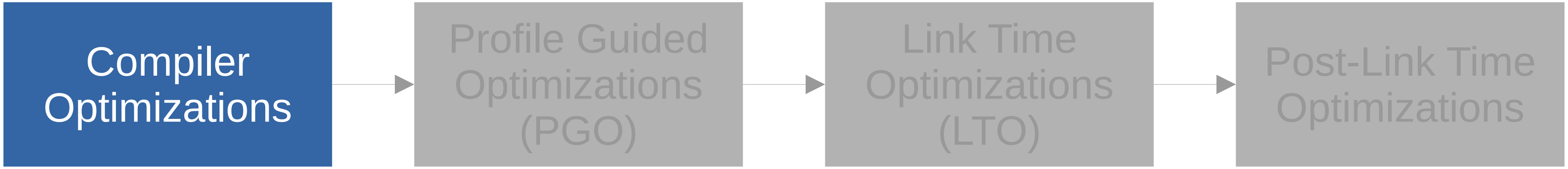
Benchmark

- 3 runs with modified kernel after fresh reboot

System

- Intel Skylake (Xeon Gold 6148 CPU @ 2.40GHz)
- Rocky Linux 8.10
- Kernel version `4.18.0-553.5.1.el8_10.x86_64`

KERNEL BINARY OPTIMIZATIONS



Compiler Optimizations

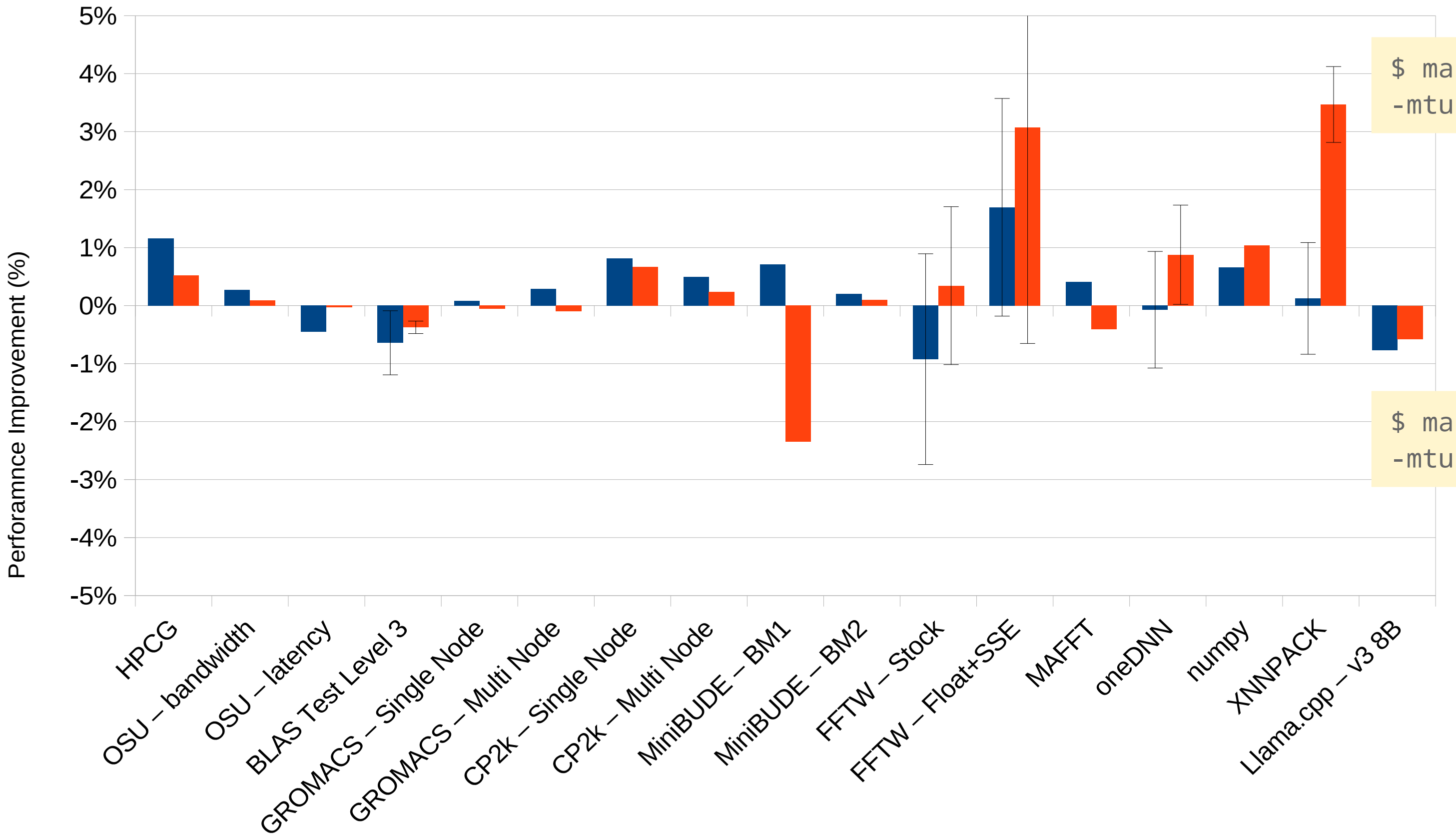
Profile Guided Optimizations (PGO)

Link Time Optimizations (LTO)

Post-Link Time Optimizations

- GCC vs LLVM
- -O2, -O3
  - Function inlining
  - Vectorization
  - Loop optimization
- -march
  - SSE, AVX

KERNEL INSTRUCTION SET



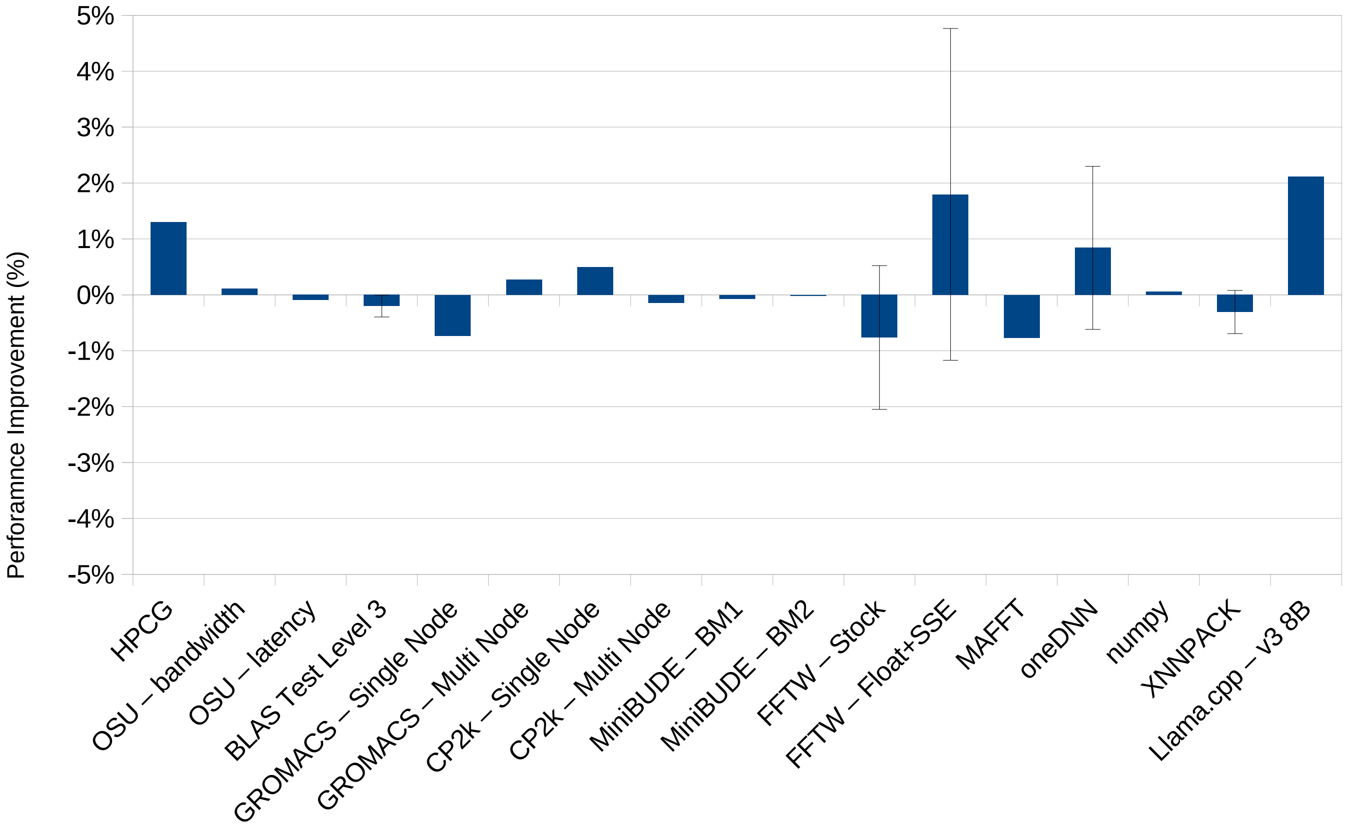
```
$ make KCFLAGS=”-march=haswell
-mtune=generic”
```

Generic → AVX2
0.24% ± 0.27

```
$ make KCFLAGS=”-march=native
-mtune=native”
```

Generic → AVX512
0.38% ± 0.53

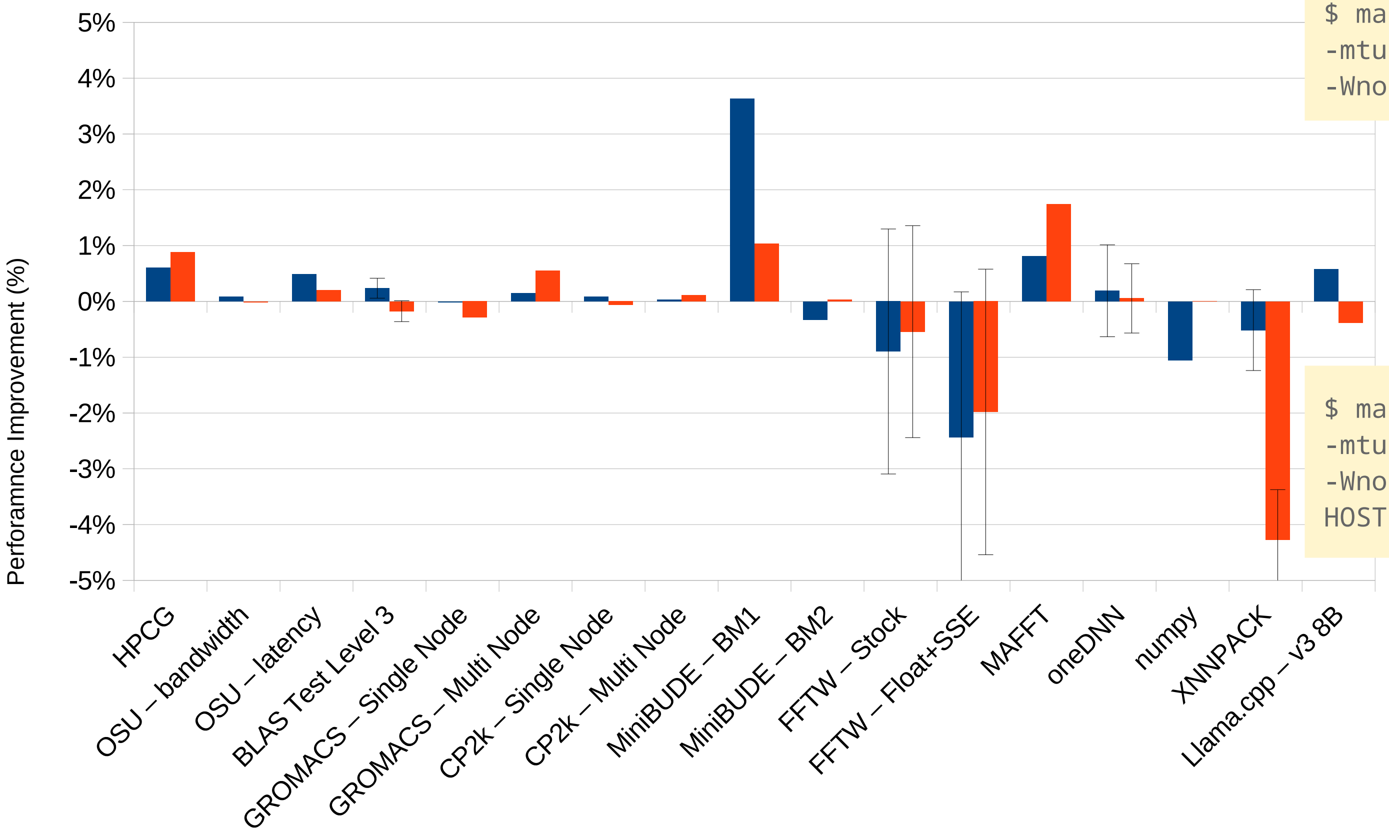
KERNEL COMPILER



```
$ make LLVM=1 HOSTCC=clang
HOSTLD=ld.lld
```

GCC → LLVM
0.23% ± 0.34

KERNEL OPTIMIZATION LEVEL



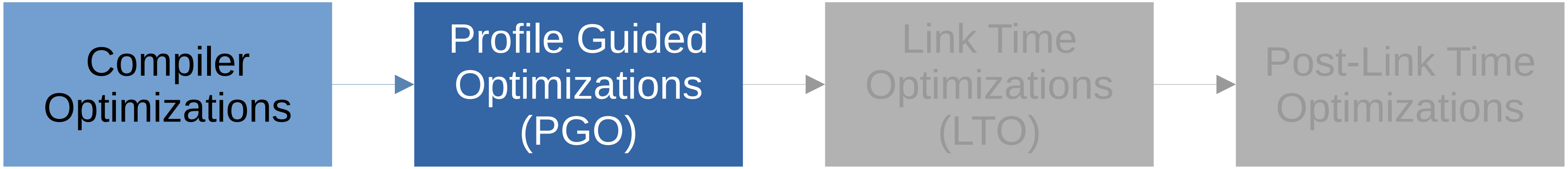
```
$ make KCFLAGS="-O3 -march=native
-mtune=native -Wno-stringop-overflow
-Wno-array-bounds"
```

O2 → O3 (GCC)
0.10% ± 0.48

```
$ make KCFLAGS="-O3 -march=native
-mtune=native -Wno-array-bounds
-Wno-stringop-overflow" LLVM=1
HOSTCC=clang HOSTLD=ld.lld
```

O2 → O3 (LLVM)
-0.18% ± 0.52

KERNEL BINARY OPTIMIZATIONS


Compiler Optimizations
Profile Guided Optimizations (PGO)
Link Time Optimizations (LTO)
Post-Link Time Optimizations

- GCC vs LLVM
- `-O2, -O3`
  - Function inlining
  - Vectorization
  - Loop optimization
- `-march`
  - SSE, AVX

- GCOV
  - Branch probabilities
  - Expression values
- AutoFDO
  - Feedback directed
  - Hardware sampling

KERNEL PROFILE GUIDED OPTIMIZATIONS

Procedure with GCC - GCOV

- Compile kernel image with coverage tooling (GCOV)

```
$ rpmbuild --define "buildid .opt01" --with gcov -bb kernel.spec
```

- Reboot into kernel with GCOV
- Run target application/workflow, kernel will generate profile in *debugfs*
- Copy profile data in *debugfs* into a known/stable location

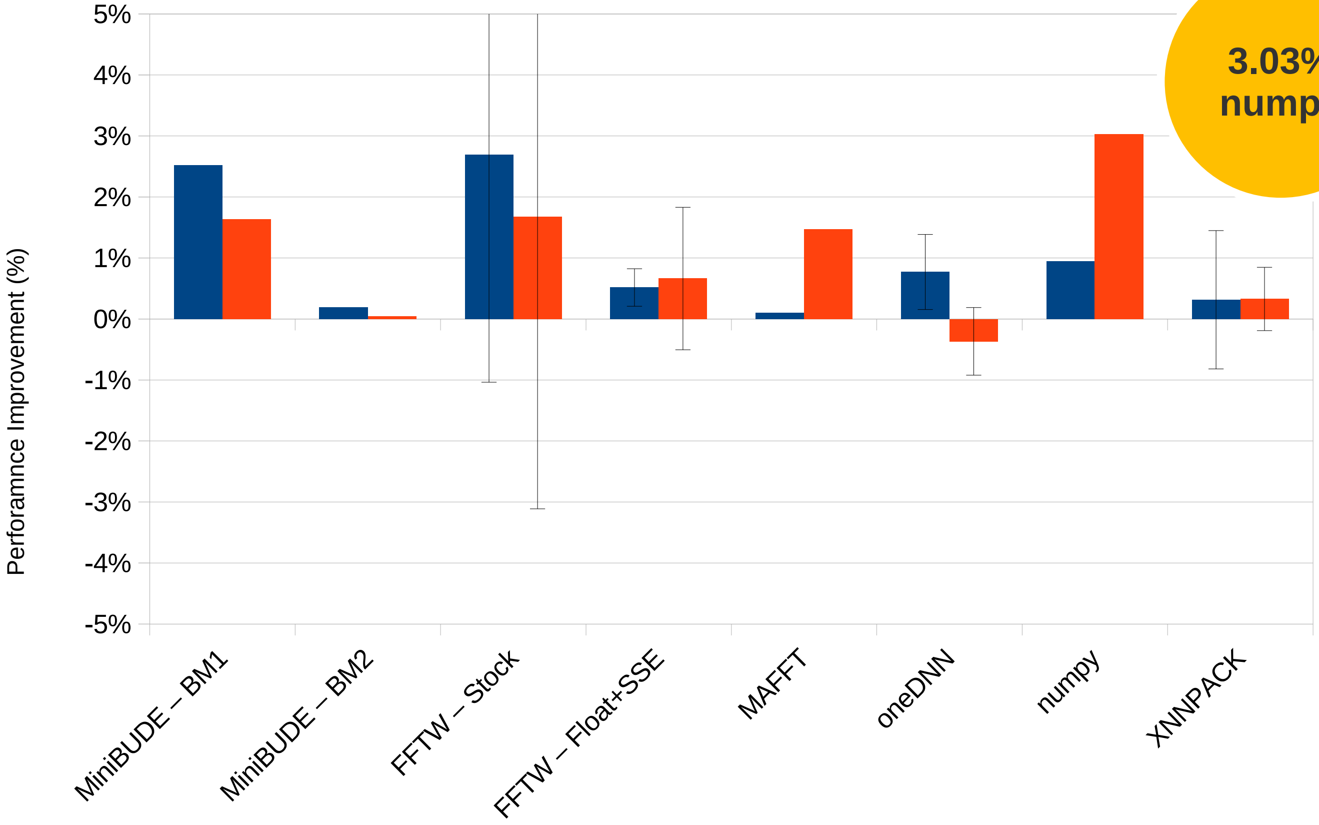
```
$ sudo cp -r /sys/kernel/debug/gcov/* /path/to/profile
```

- Compile kernel using profiled data (use same *buildid* and without GCOV)

```
$ rpmbuild --define "buildid .opt01" --define "kcflags -fbranch-probabilities
-fprofile-correction -fprofile-dir=/path/to/profile" --without gcov -bb kernel.spec
```

- Reboot into new optimized kernel

KERNEL PROFILE GUIDED OPTIMIZATIONS

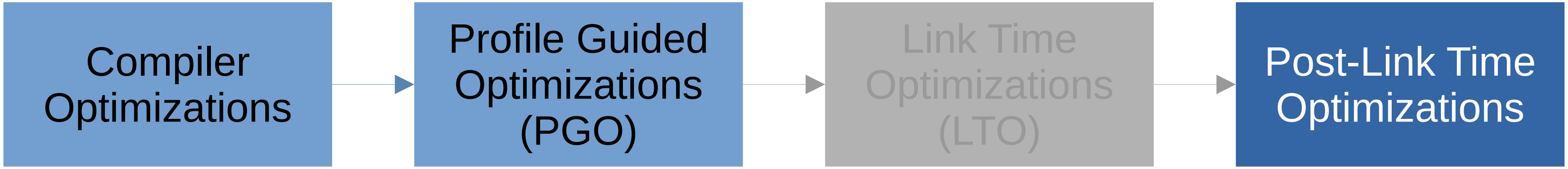


3.03% numpy

PGO (all tests)
1.01% ± 0.60

PGO (numpy)
1.06% ± 0.64

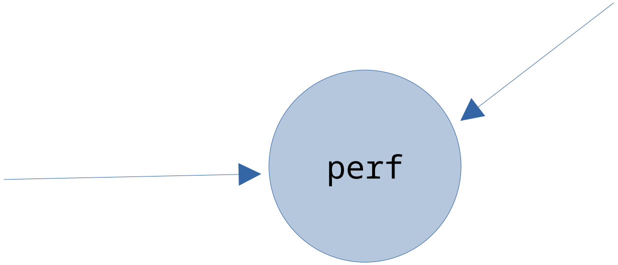
KERNEL BINARY OPTIMIZATIONS


Compiler Optimizations
Profile Guided Optimizations (PGO)
Link Time Optimizations (LTO)
Post-Link Time Optimizations

- GCC vs LLVM
- `-02, -03`
  - Function inlining
  - Vectorization
  - Loop optimization
- `-march`
  - SSE, AVX

- GCOV
  - Branch probabilities
  - Expression values
- AutoFDO
  - Feedback directed
  - Hardware sampling

Kernel 6.13



- BOLT
  - Improve binary layout
  - Hardware sampling
  - Part of LLVM project, but independent of compiler

CONCLUSIONS

Compiler optimizations of the kernel have a **negligible** effect on application performance

- Less than 0.50% on average
- Can impact negatively certain applications

PGO of the kernel can have a **small but significant** performance improvement

- 3% for a single target application with just branch probabilities
- There is **room for improvement**: PGO with value estimates, AutoFDO, BOLT
- Non-target applications got no negative impact, 1% improvement overall
- **Difficult to apply** on a diverse HPC ecosystem, kernels tuned per research domain?

ACKNOWLEDGEMENTS

VUB HPC

VLAAMS
SUPERCOMPUTER
CENTRUM

Vlaanderen
is supercomputing

Thank you for your attention!