Codee Training Series

April 26-27, 2022

NERSC

codee

Shift Left Performance

Automated Code inspection for Performance

First: Introduction to Codee - Shift Left Performance

#1 Introduction to Codee tools: Shift Left Performance

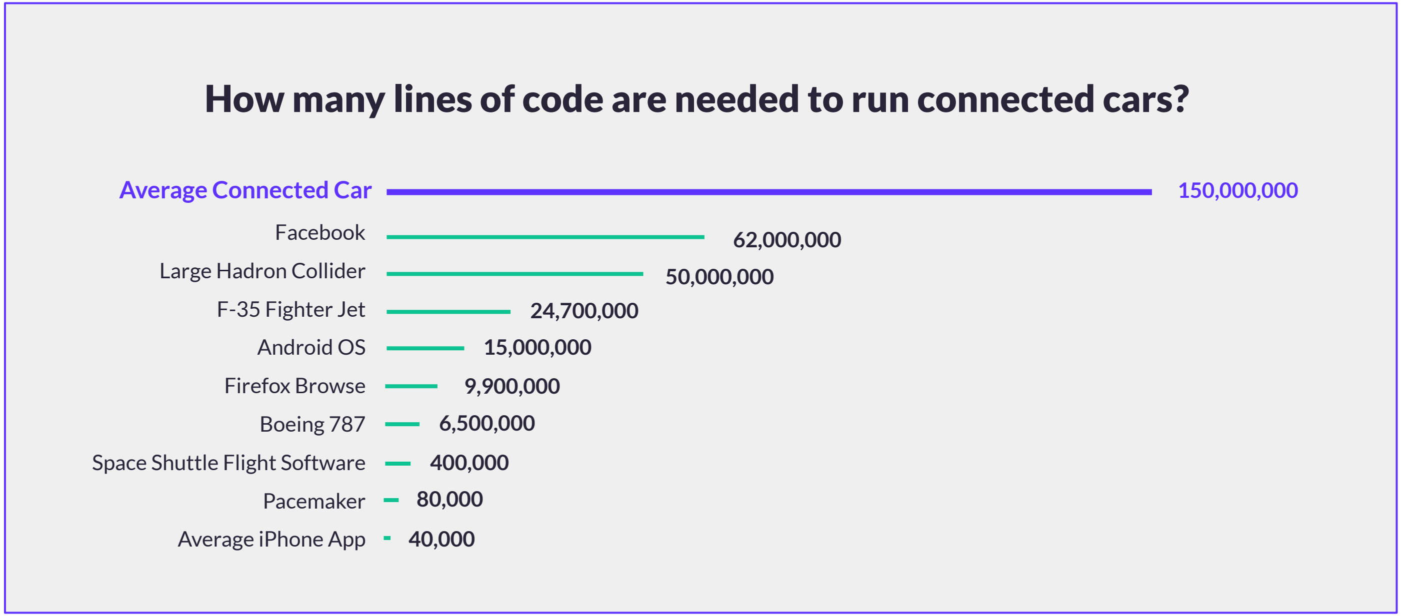
- Introduction to Codee and the **shift left** approach
- **Open catalog of coding rules for performance** optimization
- **Automated code inspection with Codee**: Discover and Adopt
- **Quick start to Codee**: Canny image processing
- Hands-on: **Optimizing PI** on Perlmutter

Format:

- Remote lectures (~30’), demos. and hands-on sessions

Software Size and Complexity continues to grow Faster than Ever

The ability of the software to make time-sensitive decisions is key



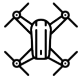
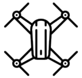
Source: https://www.uswitch.com/guides/car-insurance/data-security-in-connected-cars/

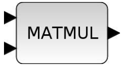
Need for New Developer Tools to Shift Left Performance

The ability of software developers to write code that runs fast on modern hardware is key

Anyone Developing in C/C++ is a Candidate to Adopt Codee

Opportunities are extensive across many market verticals

Market	Audio encoding	Compression	Simulation	Image processing	Video encoding	Bio informatics	Astro physics	5G networks	Maths kernels	Simulation CFD
	MP3	UNZIP	SPEC CPU 2017	CANNY	FFMPEG	GROMACS	HACCmk	Linux Kernel	MATMUL	NASA NPB
Automotive										
Telecommunications										
MilAero drone										
Consumer electronics										
High Performance Computing (HPC)										



Shift Left Performance

Automated Code Inspection For Performance

Benefits

Deliver faster applications
for modern low-power hardware

Save costs in software development

Integrate a repeatable, scalable and robust solution
into the development workflow

Performance Optimization Roadmap using Codee

- Steps you need to take to get the maximum performance out of your C/C++ application.
- Optimization areas are ultimately the same on any type of processor, namely:
  - Memory traffic control
  - Vectorization
  - Multi-threading

- However, tuning your C/C++ code for a given type of processor may require focus on specific areas.
- Codee assists the developer in writing hardware-friendly C/C++ code that runs efficiently on any type of processor.
- Codee supports the programming techniques required at any step of the performance optimization roadmap.

Step	Programming techniques specialized in performance
1	**Sequential scalar** optimization
2	**Sequential control flow** optimization
3	**Sequential memory** optimization
4	**Vectorization**
5	**Multi-threading**
6	**Offloading**

Open Catalog of Coding Rules for Performance

https://www.codee.com/knowledge/

Recommendations (40)
PWR001: Declare global variables as function parameters
PWR002: Declare scalar variables in the smallest possible scope
PWR003: Explicitly declare pure functions
PWR004: Declare OpenMP scoping for all variables

Opportunities (3)
OPP001: Multi-threading opportunity
OPP002: SIMD opportunity
OPP003: Offloading opportunity

Defects (11)
PWD002: Unprotected multithreading reduction operation
PWD003: Missing array range in data copy to the GPU
PWD004: Out-of-memory-bounds array access
PWD005: Array range copied to or from the GPU does not cover the used range

Remarks (14)
RMK001: Loop nesting that might benefit from hybrid parallelization using multithreading and SIMD
RMK002: Loop nesting that might benefit from hybrid parallelization using offloading and SIMD
RMK003: Potentially privatizable temporary variable

Glossary (22)
Locality of Reference
Loop fission
Loop interchange
Loop sectioning
Loop tiling
Loop unswitching
Loop-carried dependencies
Memory access pattern
Multithreading
Offloading

Navigating the Open Catalog by Stage of the Roadmap

https://www.codee.com/knowledge/

Sequential optimizations	SIMD/Vector execution	Multi-threaded execution	Offloading to accelerators
PWR001: Declare global variables as function parameters PWR002: Declare scalar variables in the smallest possible scope PWR003: Explicitly declare pure functions PWR004: Declare OpenMP scoping for all variables PWR007: Disable implicit declaration of variables PWR008: Declare the intent for each procedure parameter PWR010: Avoid column-major array access in C/C++ PWR012: Pass only required fields from derived data types as parameters RMK004: Avoid strided array access to improve performance RMK005: Avoid non-consecutive array access to improve performance RMK006: Avoid indirect array access to improve performance	PWR017: Transform while into for loop in order to allow vectorization PWR018: Call to recursive function within a loop may inhibit vectorization PWR019: Consider interchanging loops to favor vectorization by maximizing inner loop's trip count PWR020: Consider loop fission to enable vectorization PWR021: Temporary computation can be extracted to a vectorizable loop PWR022: Move invariant conditional out of the loop to facilitate vectorization PWR023: Add 'restrict' for pointer function parameters to hint the compiler that vectorization is safe	PWR006: Avoid privatization of read-only variables PWD001: Invalid OpenMP multithreading datascoping PWD002: Unprotected multithreading reduction operation PWD004: Out-of-memory-bounds array access PWD007: Unprotected multithreading recurrence PWD008: Unprotected multithreading recurrence due to out-of-dimension-bounds array access PWD009: Incorrect privatization in OpenMP parallel region PWD010: Incorrect sharing in OpenMP parallel region PWD011: Missing OpenMP last private clause RMK003: Potential temporary variable for the loop which might be privatizable, thus enabling the loop parallelization	PWR009: Use OpenMP teams to offload work to GPU PWR013: Avoid copying unused variables to the GPU PWR015: Avoid copying unnecessary array elements to or from the GPU PWR024: Loop can be rewritten in OpenMP canonical form PWR025: Consider annotating pure function with OpenMP 'declare simd' PWR026: Annotate function for OpenMP offload PWR027: Annotate function for OpenACC offload PWD003: Missing array range in data copy to the GPU PWD005: Array range copied to or from the GPU does not cover the used range PWD006: Missing deep copy of non-contiguous data to the GPU

Performance Optimization Platform

```
examples/matmul$ pwreport src/main.c:15 --level 2 -- -I src/include
Compiler flags: -I src/include

ACTIONS REPORT

  FUNCTION BEGIN at src/main.c:matmul:6:1
    6: void matmul(size_t m, size_t n, size_t p, double **A, double **B, double **C) {

    LOOP BEGIN at src/main.c:matmul:15:5
      15:     for (size_t i = 0; i < m; i++) {

      [PWR010] src/main.c:15:5 'B' multi-dimensional array not accessed in row-major order
      [RMK005] src/main.c:18:28 avoid non-consecutive array access for variable 'A' to improve performance
      [RMK005] src/main.c:18:38 avoid non-consecutive array access for variable 'B' to improve performance
      [RMK005] src/main.c:18:25 avoid non-consecutive array access for variable 'C' to improve performance
      [RMK005] src/main.c:18:25 avoid non-consecutive array access for variable 'C' to improve performance

      [OPP001] src/main.c:15:5 is a multi-threading opportunity
      [OPP003] src/main.c:15:5 is a offload opportunity
    LOOP END
  FUNCTION END

  FUNCTION BEGIN at src/main.c:main:24:1
    24: int main(int argc, char *argv[]) {

  FUNCTION END
```

Opportunities `(OPP)`
`Sequential, vectorization, multi-threading and GPU offloading`

Recommendations `(PWR)`
`Boost performance and ensure best practices`

Defects `(PWD)`
`Find and fix bugs in parallel code and correctness verification`

Remarks `(RMK)`
`Proficient usage of tools`

Scan source code without executing that code

Report human-readable actionable recommendations on where and how to fix performance issues

Compliance with performance optimization best practices (memory usage, vectorization, multi-threading, offload)

Optimize performance for **microprocessors** (x86, Arm, Power) and **accelerators** (GPU)

Automated fixes to actually implement code changes

Customization and **extension** of built-in rule set

Full workflow support: CI/CD, repository, IDE and issue trackers

First, produce the Codee Performance Optimization Report

```
$ pwreport --evaluation canny.c --include-tags all

Target  Lines of code Analyzed lines Analysis time # actions Effort Cost     Profiling
------- ------------- -------------- ------------- --------- ------ -------- ---------
canny.c 656           252            623 ms        114       579 h  18947€   n/a

ACTIONS PER OPTIMIZATION TYPE
Target  Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
------- ------------- -------------- ------------- ------------- -------------- ----------
canny.c 17            49             8             15            22             3

Target : analyzed directory or source code file
Lines of code : total lines of code found in the target (computed the same way as the sloccount tool)
Analyzed lines : relevant lines of code successfully analyzed
Analysis time : time required to analyze the target
# actions : total actionable items (opportunities, recommendations, defects and remarks) detected
Effort : estimated number of hours it would take to carry out all actions (serial scalar, serial control, serial memory, vectorization,
multithreading and offloading with 1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the actions, paying the average salary of 56,286€/year for a professional C/C++ developer
working 1720 hours per year
Profiling : estimation of overall execution time required by this target

SUGGESTIONS
  You can specify multiple inputs which will be displayed as multiple rows (ie. targets) in the table, eg:
        pwreport --evaluation some/other/dir canny.c --include-tags all

  Use --actions to find out details about the detected actions:
        pwreport --actions canny.c --include-tags all

  You can focus on a specific optimization type by filtering by its tag (serial-scalar, serial-control, serial-memory, vectorization,
multithreading, offloading), eg.:
        pwreport --actions --include-tags serial-scalar canny.c

1 file successfully analyzed and 0 failures in 123 ms
```

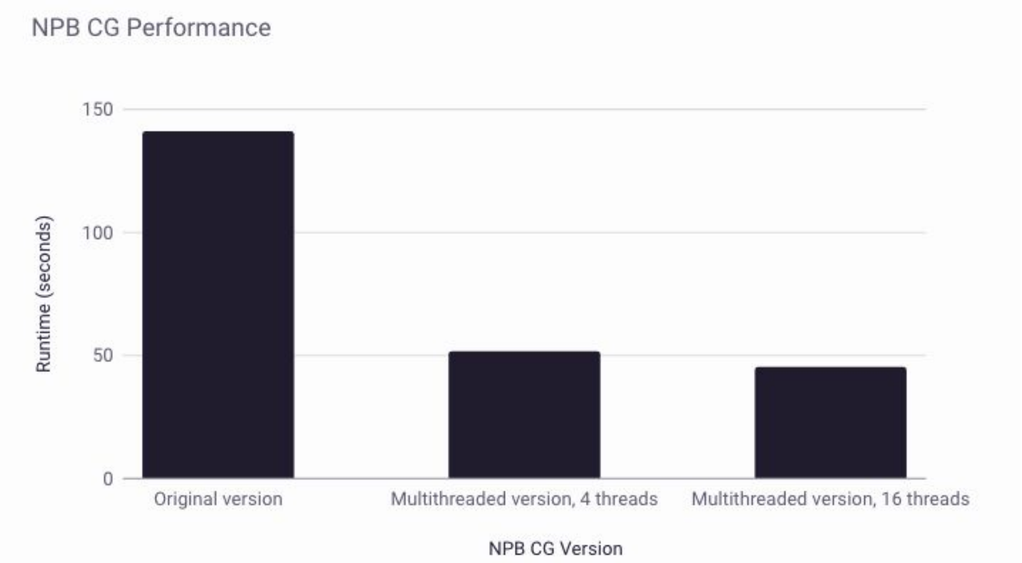
Second, produce the Codee Actions Report

```
$ pwreport --actions --level 2 canny.c:gaussian_smooth --include-tags all

ACTIONS REPORT
  ...
  [RMK010] canny.c:496:10 the vectorization cost model states the loop is not a SIMD opportunity due to strided memory
accesses in the loop body

  More information on: https://www.appentra.com/knowledge/rmk010
  ...

  [OPP001] canny.c:492:4 is a multi-threading opportunity
    Compute patterns:
      - 'forall' over the variable 'smoothedim'

    SUGGESTION: use pwloops to get more details or pwdirectives to generate directives to parallelize it:
      pwloops canny.c:gaussian_smooth:492:4
      pwdirectives --omp multi canny.c:gaussian_smooth:492:4 --in-place

    More information on: https://www.appentra.com/knowledge/opportunities
  ...


$ pwdirectives --omp multi canny.c:gaussian_smooth:492:4 --out-file canny_optimized.c
Successfully parallelized loop at 'canny.c:gaussian_smooth:492:4' [using multi-threading]:
...

$ cc canny.c -fopenmp -O3 -lm -o canny
$ ./canny testvecs/input/15360_8640.pgm 0.5 0.7 0.9
Total time: 14.594

$ cc canny_optimized.c -fopenmp -O3 -lm -o canny
$ ./canny testvecs/input/15360_8640.pgm 0.5 0.7 0.9
Total time: 8.488
```

And Measure the Performance Improvement enabled by Codee

- The **primary goal** is to **show performance gain** on the target application code

- Target **hardware platform** equipped with:
  - x86/Arm processor
  - Clang/GCC compiler
  - Linux OS



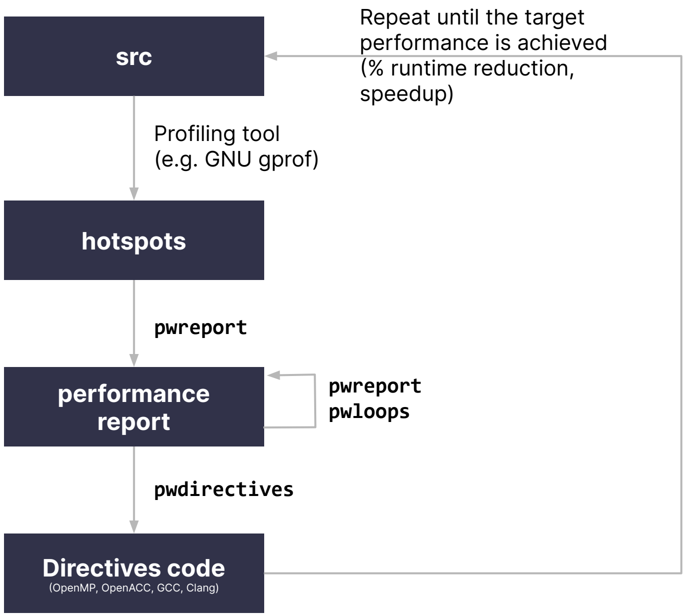
The loop cg.c:conj_grad:458:5 runs 3 times faster that the original version on an AMD Ryzen 7 4800H laptop with 8 cores and 16 hardware threads, 16 GB of memory, Linux Ubuntu 20.04 operating system and CLANG 10 compiler.

Blog post “A touch of parallelism: example of NPB CG Benchmark”:
https://www.codee.com/touch-of-parallelism-example-of-npb-cg-benchmark/

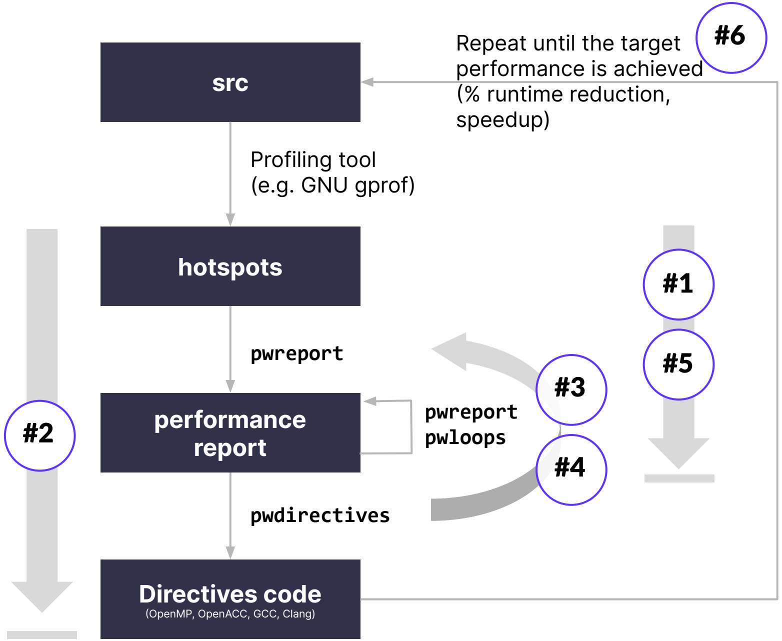
Improvement in Performance enabled by Codee

Codee's programming techniques specialized in performance	Micro processors	Micro controllers	Other devices	Performance gains following Codee's best practices
Sequential Scalar optimization	X	X	X	HACCmk runs **70% faster** (from 37s down to 11s)
Sequential Control flow optimization	X	X	X	HotSpot3D runs **26% faster** (from 2.7s down to 2s)
Sequential Memory optimizations	X	X	X	Canny runs **63% faster** (from 9s down to 4.5s)
Vectorization SIMD	X		X	Hotspot runs **17% faster** (from 4s down to 3.3s)
Multithreading multicore CPU	X		X	HACCmk runs **92% faster** (from 92s down to 6.5s) NPB CG runs **63% faster** (from 141ms down to 52ms)
Offloading GPU			X	MATMUL runs **96% faster** (from 57s down to 2.4s)

Typical Use Cases for C/C++ Developers: Profile guided!



Typical Use Cases for C/C++ Developers: Profile guided!



- **#1** **Get the performance optimization report for the whole code base**
- **#2** **Create performance-optimized code for the hotspot automatically**
- **#3** **Unlock new performance optimization opportunities in the code**
- **#4** **Integration with compilers**
- **#5** **Integration with build systems**
- **#6** **Benchmark Codee performance impact on your hardware platform**

codee

www.codee.com

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/