

Quickstart guide of Codee

1. Download the binary distribution package of Codee for your system.
2. Uncompress the Codee package in the desired location. This "*Codee installation folder*" contains the subfolder `bin/`, `lib/` and `examples/`).
3. Add the location of the Codee binary available in the subfolder '`bin/`' of the Codee installation folder to the environment PATH. Setup in the system terminal window:

```
Linux:          export PATH="$PATH:<Codee installation folder>/bin"
Windows-terminal: set PATH="%PATH%;<Codee installation folder>\bin"
Windows-powershell: $env:path += "<Codee installation folder>\bin"
MacOS:          export PATH=$PATH:<Codee installation folder>/bin
```

Alternatively, setup the path to the Codee binaries in the computer boot process:

```
Linux-bash:    Add the export PATH command to .bashrc and re-open the terminal
Windows:       Add the export PATH command in System properties and re-open the terminal
```

4. Copy the Codee license file to the *Codee installation folder* with the name "`codee.lic`". Alternatively, specify its location through the `CODEE_LICENSE_PATH` environment variable.

5. Run the Codee command-line tools to show the Codee version installed in the system:
`codee --version`

6. Using the performance-demos repository as base, go to the MATMUL/serial folder:

```
Linux:          git clone https://github.com/codee-com/performance-demos-fortran.git
                  cd performance-demos-fortran/MATMUL/serial
Windows:        git clone https://github.com/codee-com/performance-demos-fortran.git
                  cd performance-demos-fortran\MATMUL\serial
```

7. Produce the Screening Report of Codee (`screening` command):

```
codee screening matmul.f90 --target-arch cpu -- -O3
```

```
Date: 2024-04-08 Codee version: 2024.2
Compiler flags: -O3
```

```
[Fortran] target compiler: <none> (Compiler Agnostic Mode)
```

```
[1/1] matmul.f90 ... Done
```

SCREENING REPORT

```
---Number of files---
```

Total	C	C++	Fortran
1	0	0	1

```
Lines of code Analysis time # checks Profiling
```

Lines of code	Analysis time	# checks	Profiling
21	50 ms	7	n/a

CHECKS PER CATEGORY AND PRIORITY LEVELS

Checks per category							Priority		
Scalar	Control	Memory	Vector	Multi	Offload	Quality	L1	L2	L3
0	0	2	2	1	n/a	2	2	1	4

```
Lines of code : total lines of code found in the target (computed the same way as the sloccount tool)
```

```
Analysis time : time required to analyze the target
```

```
# checks : total actionable items (opportunities, recommendations, defects and remarks) detected
```

```
Profiling : estimation of overall execution time required by this target
```

```

RANKING OF CHECKERS

Checker Level Priority # Title
-----
-
PWR003 L1 P18 1 Explicitly declare pure functions
PWR063 L1 P12 1 Avoid using legacy Fortran constructs
PWR050 L2 P6 1 Consider applying multithreading parallelism to forall loop
PWR035 L3 P2 2 Avoid non-consecutive array access to improve performance
RMK010 L3 P0 2 The vectorization cost model states the loop is not a SIMD opportunity due to strided memory
accesses in the loop body

SUGGESTIONS

Use 'checks' to find out details about the detected checks:
codee checks matmul.f90 --target-arch cpu -- -O3

1 file, 1 function, 5 loops successfully analyzed and 0 non-analyzed files in
21 ms

```

8. Follow the suggestions in order to produce the Checkers Report (option `--checks`) that lists all the checks applicable to your code.

```
codee checks matmul.f90 --target-arch cpu -- -O3
```

```

Compiler flags: -I include -O3

[Fortran] target compiler: <none> (Compiler Agnostic Mode)

[1/1] matmul.f90 ... Done

CHECKS REPORT

matmul.f90 [PWR063] (level: L1): Avoid using legacy Fortran constructs
matmul.f90:17:7 [PWR050] (level: L2): Consider applying multithreading parallelism to forall loop
matmul.f90:11:7 [PWR035] (level: L3): Avoid non-consecutive array access to improve performance
matmul.f90:17:7 [PWR035] (level: L3): Avoid non-consecutive array access to improve performance
matmul.f90:12:10 [RMK010] (level: L3): The vectorization cost model states the loop is not a SIMD opportunity due to
strided memory accesses in the loop body
matmul.f90:19:13 [RMK010] (level: L3): The vectorization cost model states the loop is not a SIMD opportunity due to
strided memory accesses in the loop body

SUGGESTIONS

Use --verbose to get more details, e.g:
codee checks --verbose matmul.f90 --target-arch cpu -- -O3

Use --level to filter checks with a specific level of priority, e.g:
codee checks --level L1 matmul.f90 --target-arch cpu -- -O3

More details on the defects, recommendations and more in the Open Catalog of Best Practices for Performance:
https://github.com/codee-com/open-catalog/

1 file, 1 function, 5 loops successfully analyzed and 0 non-analyzed files in 20 ms

```

9. Show the detailed Checkers Report (option `--verbose`). As an example, focus on the checker *PWR050*, related to applying multithreading parallelization for multicore CPUs. The detailed Codee output, which includes links to the open catalog available in the website, precise location in the source code, etc..., is as follows:

```
codee checks --verbose matmul.f90 --target-arch cpu -- -O3
```

```

matmul.f90:17:7 [PWR050] (level: L2): Consider applying multithreading parallelism to forall loop
Suggestion: Use 'rewrite' to automatically optimize the code
Documentation: https://github.com/codee-com/open-catalog/tree/main/Checks/PWR050
AutoFix (choose one option):
* Using OpenMP 'for' (recommended):
codee rewrite --multi omp-for --in-place matmul.f90:17:7 -- -O3
* Using OpenMP 'taskwait':
codee rewrite --multi omp-taskwait --in-place matmul.f90:17:7 -- -O3
* Using OpenMP 'taskloop':
codee rewrite --multi omp-taskloop --in-place matmul.f90:17:7 -- -O3

```

10. Note that the check PWR050 contains an *AutoFix* section that remarks the existence of source code rewriting capabilities that facilitate the usage of Codee as a coding assistant. Focus on PWR050 and implement multithreading with OpenMP “for” option, by executing the following command:

```
codee rewrite --multi omp-for -o matmul_codee.f90 matmul.f90:17:7 -- -O3
```

11. Compile the original source code of MATMUL (`matmul.f90`) and the optimized source code of MATMUL (`matmul_codee.f90`). For instance, using the GFortran compiler:

```
gfortran matmul.f90 time.f90 main.f90 -o matmul -O3
```

```
gfortran matmul_codee.f90 time.f90 main.f90 -o matmul_codee -O3 -fopenmp
```

12. Run the original MATMUL (`matmul`) and the optimized MATMUL (`matmul_codee`):

```
Linux:      ./matmul 1500
           ./matmul_codee 1500
```

```
Windows:   .\matmul 1500
           .\matmul_codee 1500
```