

Quickstart guide of Codee using SARIF

1. This is a continuation of the Quickstart Guide to Codee, which helps to start using Codee integration with SARIF, the Static Analysis Results Interchange Format ([link](#)). First, take advantage of Codee integration with CMake (version 3.5 or later) and produce the Codee Checks Report of the whole project:

```
$ codee checks --config build/compile_commands.json
3 total entries detected

Configuration file 'build/compile_commands.json' successfully parsed.
Date: 2024-04-08 Codee version: 2024.2
[C] target compiler: <none> (Compiler Agnostic Mode)

[1/3] matrix.c ... Done
[2/3] clock.c ... Done
[3/3] main.c ... Done

CHECKS REPORT

main.c:16:9 [PWR039] (level: L1): Consider loop interchange to improve the locality of reference and
enable vectorization
matrix.c:49:1 [PWR003] (level: L1): Explicitly declare pure functions
main.c:9:9 [PWR053] (level: L1): Consider applying vectorization to forall loop
matrix.c:55:9 [PWR054] (level: L1): Consider applying vectorization to scalar reduction loop
main.c:17:13 [PWR010] (level: L3): Avoid column-major array access in C/C++
main.c:18:17 [PWR048] (level: L3): Replace multiplication/addition combo with an explicit call to
fused multiply-add
main.c:15:5 [PWR035] (level: L3): Avoid non-consecutive array access to improve performance
main.c:17:13 [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 --config build/compile_commands.json

Use --level to filter checks with a specific level of priority, e.g:
    codee checks --level L1 --config build/compile_commands.json

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

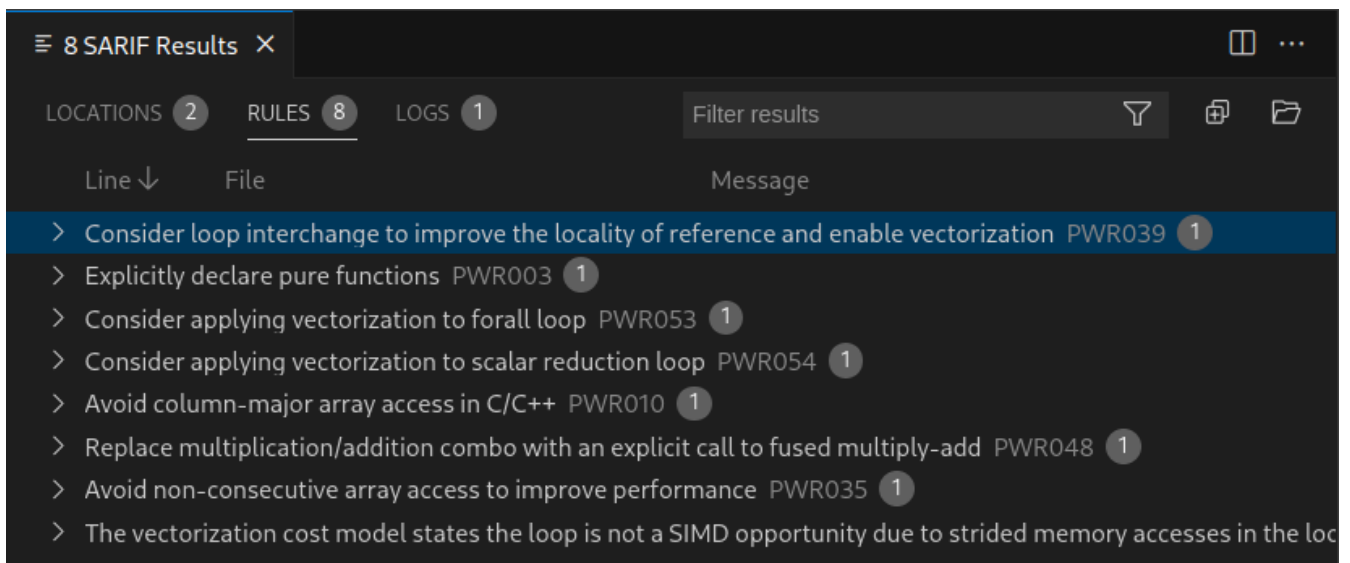
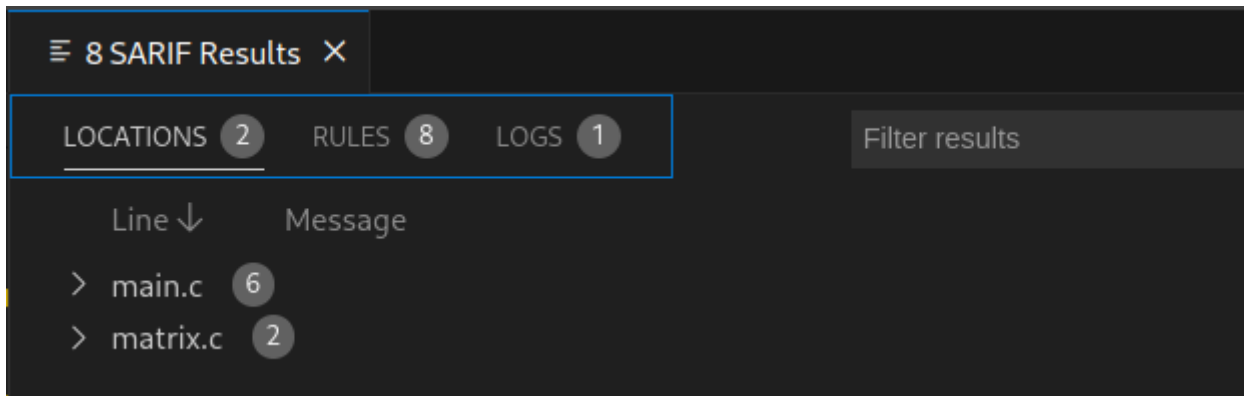
Consider using Codee with a target compiler in order to filter out optimizations that are already
applied by your compiler. For example, for GCC:
    codee checks --target-compiler-cc gcc --config build/compile_commands.json

3 files, 7 functions, 11 loops successfully analyzed and 0 non-analyzed files in 30 ms
```

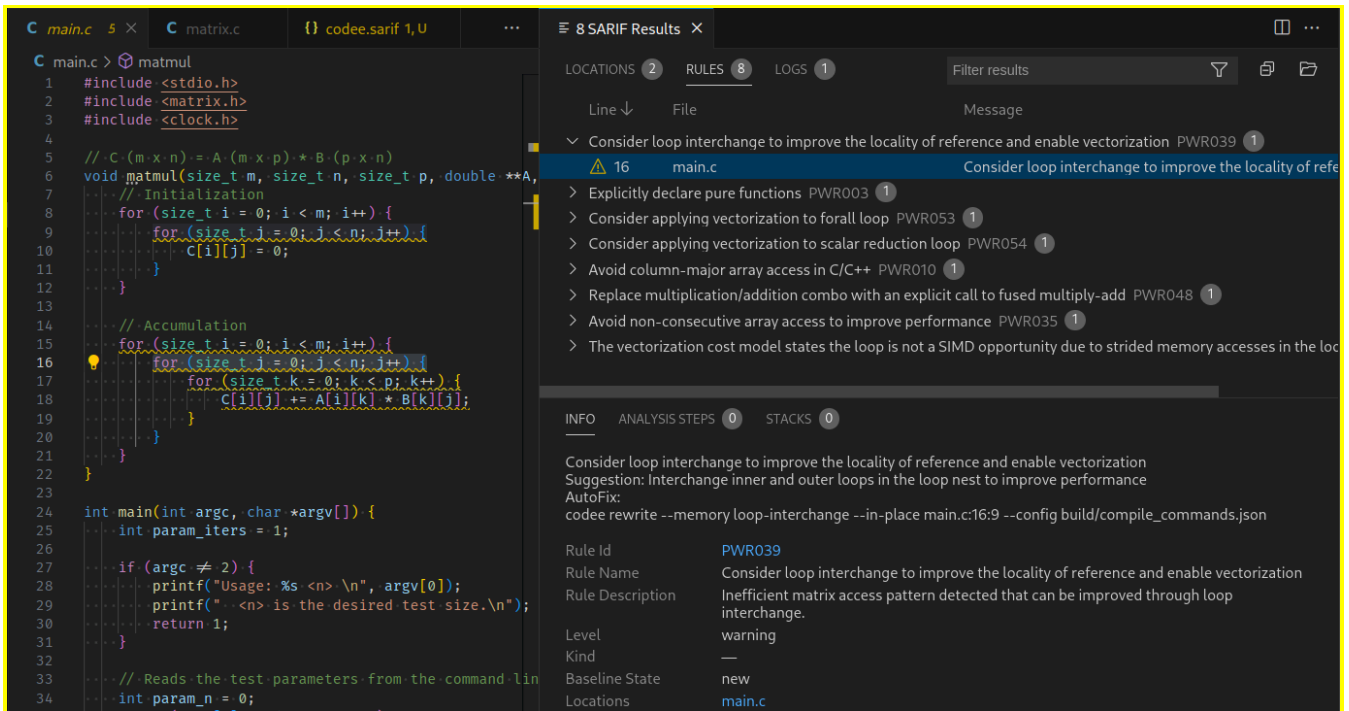
2. Next, generate the Codee Checks Report in SARIF format (--checks --sarif):

```
$ codee checks --config build/compile_commands.json --sarif > codee.sarif
```

3. Explore the contents of the SARIF file using the VSCode editor and its SARIF viewer plugin. Easily navigate the Codee Checks Report by source code file (called Locations) and by checker (called Rules).



4. Click on the corresponding source file (Location) or checker (Rule) in order to show detailed information, similar to the command line verbose mode (--verbose). As an example, focus on the checker *PWR039* related to enforcing memory efficiency on microprocessors through the *Loop Interchange* optimization.



- Finally, the rest of the optimization process continues as usual in the command-line, including the actions to modify the source code, compile and run the original MATMUL and the optimized MATMUL:

```
codee rewrite --memory loop-interchange -o main_codee.c main.c:16:9 -- -I include -O3
gcc main.c matrix.c clock.c -o matmul -I include -O3
gcc main_codee.c matrix.c clock.c -o matmul_codee -O3 -I include
```

Linux: ./matmul 1500
 ./matmul_codee 1500

Windows: .\matmul 1500
 .\matmul_codee 1500

Also measure the performance improvement obtained in your system (see [Codee leaflet for loop interchange](#)).

Note: The SARIF Viewer extension tested was the v3.3.7. The newer SARIF viewer versions (including the latest: v3.4.4, as of April 8, 2024) have several bugs that we properly identified and reported to Microsoft.