codee

# Automated Inspection of C/C++/Fortran Code Using Codee for Performance Optimization on HPE/Cray

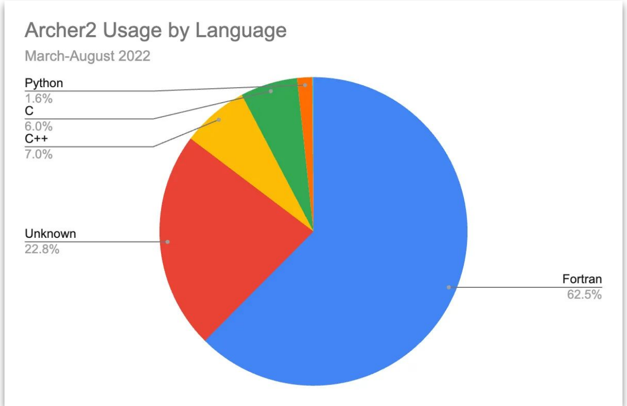
CUG2024 tutorial

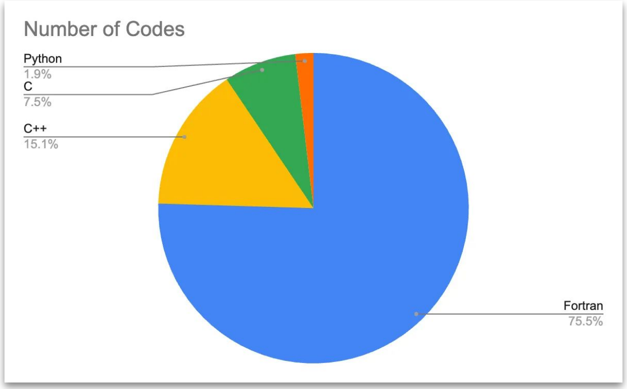
May 6, 2024

Revision May 6th, 2024

# Fortran: A Long History, Still Alive!

- Created in the 1950s by IBM
- 1st compiler created with Fortran
- Widely used in
  - Climate & Weather
  - Automotive
  - Oil and Gas
  - Aerospace
  - Defense
  - Energy & Utilities
  - Manufacturing
  - High Performance Computing
  - Scientific Research
- Number 14th in the TIOBE Index [https://www.tiobe.com/tiobe-index/](https://www.tiobe.com/tiobe-index/)





Source: [https://cpufun.substack.com/p/is-fortran-a-dead-language](https://cpufun.substack.com/p/is-fortran-a-dead-language)

# Your Main Drivers for Fortran Modernization?

- Enforcing the **modernization of Fortran** code bases is valuable by itself.
- Using modern Fortran increases the **quality** of the code and **facilitates maintenance**.
- The modernization process helps **find bugs** and **avoid introducing hidden bugs** in code.
- As a result, the modernization process helps ensure **correctness** of the Fortran code.
- Overall, **enforce Fortran modernization before addressing performance optimization**.

# What Fortran community is saying about this…

“Always use IMPLICIT NONE everywhere. **It is amazing how many bugs this can find and avoid** compared to the default typing rules.”

“All subprograms should be CONTAINed. Generally in modules, but also in the main program unit. If the subprograms are in individual files, use INCLUDEs in a module to compile them together. **Again, amazing how many interface bugs show up when this is enforced**.”

“**Many many more could be suggested. Here are a few in no specific order that help compilers find more bugs at compile time, and help programs scale better**:

- Always specify intent attributes for dummy arguments.
- Always use assumed shape for array dummy arguments. Perhaps with the CONTIGUOUS attribute.

…”

“Always use Standard conforming code. **Turn on all warnings** (e.g., -std=f2018 -Wall with gfortran) and fix any issues by using Standard conforming code. There are really very few compiler extensions from the Olden Days that do not have modern, Standard conforming, replacements.”

Source: https://fortran-lang.discourse.group/t/our-initiative-to-publish-the-fortran-lang-top-10-recommendation-for-fortran-modernization-is-it-really-new-or-even-feasible/7774/18

# Top 10 Recommendations for Fortran Modernization

fortran-lang.discourse.group

### 1. Strict compliance with modern Fortran standards

Remove deleted legacy features not be supported by recent compilers, and avoid compiler-specific extensions, ensuring that Fortran code remains compatible across various compilers and development environments.

### 2. Declare procedures in modules

Declare related procedures within a module to enhance code modularity and readability, while also helping avoid runtime errors linked to implicit interfaces. Separate the definition of procedures into modules and their implementation into submodules, leveraging incremental compilation to reduce times.

### 3. Restrict data visibility with modules

Move globally accessible data, such as common blocks, into modules to encapsulate data and provide controlled access interfaces through specific procedures, improving code readability and minimizing side from global data storage.

### 4. Improve dummy arguments semantics

Enhance the definitions of dummy arguments to improve the predictability of procedures, helping avoid issues that arise from incorrect assumptions about data type, flow, or structure.

### 5. Improve data type consistency and management

Ensure consistency in data types by avoiding implicit typing and using a fixed real type, improving code readability and portability across different development environments. Use derived data types to represent complex multi-field structures. Leverage allocatable for safe memory handling.

### 6. Avoid legacy control-flow constructs

Replace outdated and error-prone control-flow constructs with more robust and maintainable language features from recent standards (e.g., Fortran 2008, 2018, 2023), improving code maintainability and reducing the likelihood of bugs.

### 7. Enhance source code semantics

Leverage keywords from recent Fortran standards to improve the clarity and intent of applicable code statements.

### 8. Adherence to code conventions

Establish and adhere to a consistent coding standard, such as variable naming of free-form format, to promote readability and ease collaboration among developers

### 9. Adopt modern development practices

Integrate modern development practices, such as automated testing, version control, or dependency managers, to enhance quality, maintainability, collaboration, and distribution of Fortran software.

### 10. Proper C/C++ interoperability

Ensure seamless interoperability between Fortran and C/C++ to allow Fortran programs to effectively interact with a wide range of systems and libraries written in other languages (e.g., high-performance environments).

# Top 20 Checkers for Fortran Modernization

**GitHub Catalog Link**
github.com/codee-com/open-catalog

- [M01] Tune compiler flags to mark non-standard and removed features in modern Fortran standards.
- [M01] Consider using more standard-compliant compilers like gfortran to flag non-standard and removed features.
- [M02] Encapsulate an external procedure into an importable module to avoid calls to an implicit interface that can lead to undefined behavior.
- [M03] Transform common block into a module for better data encapsulation.
- [M03] Use the keyword only to explicitly state what to import from a module.
- [M04] PWR008: Declare the intent for each procedure argument.
- [M04] Declare array dummy arguments as assumed-shape arrays.
- [M05] PWR007: Always use implicit none to disable implicit declarations.
- [M05] Prefer real(kind=kind_value) for declaring consistent floating types.
- [M06] PWR063: Avoid using legacy and old-style Fortran constructs.
- [M07] PWR003: Explicitly declare pure functions.
- [M07] Add an explicit parameter attribute to constant variables.
- [M07] Add an explicit save attribute when initializing variables in their declaration.

● PUBLISHED ● IMMINENT ● PLANNED

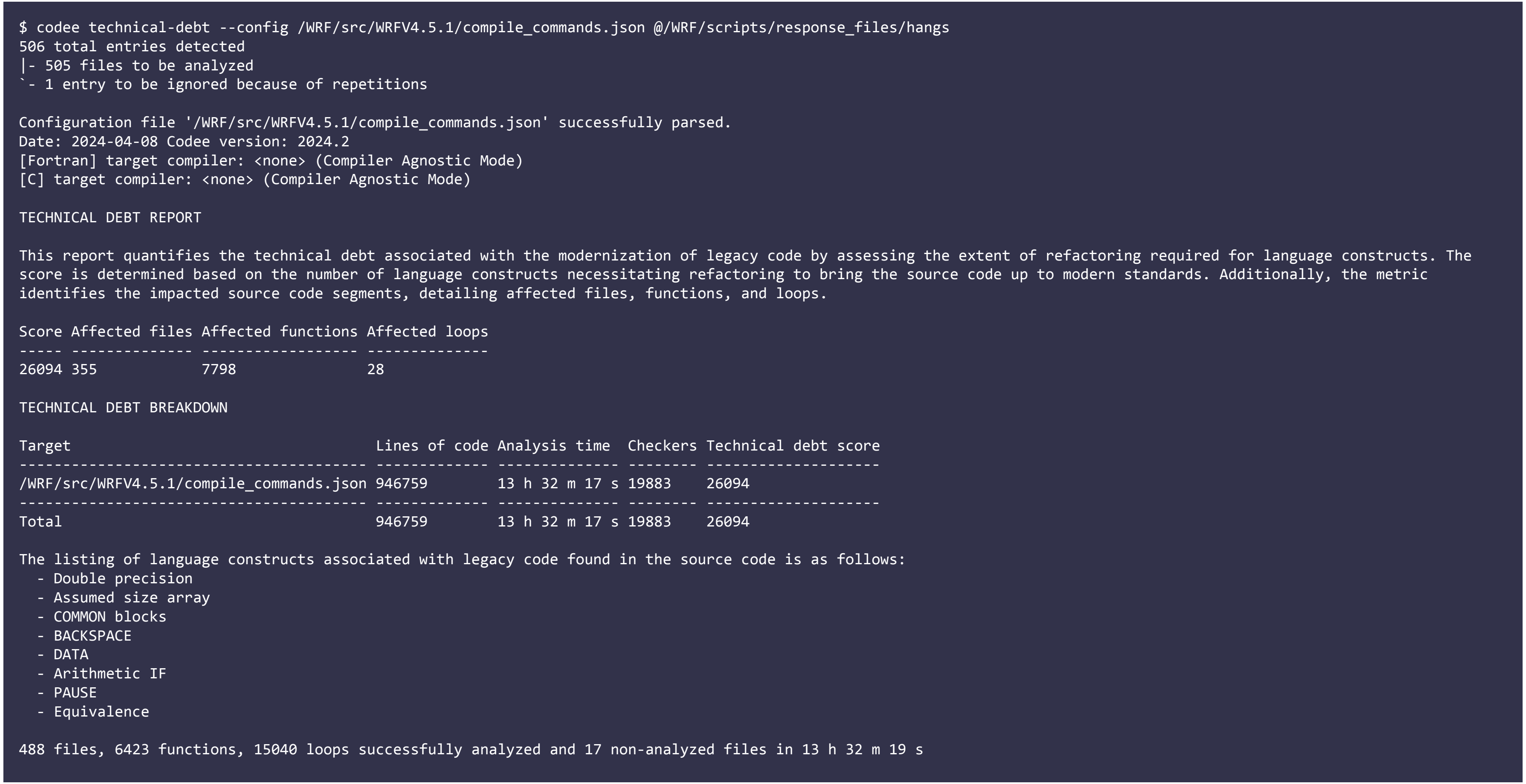
MAIN FEATURES

# Codee for Fortran Modernization (and Optimization)

- **Static Analysis**: Analyze every code line to find and fix code modernization opportunities and run sanitizers on your code.
- **Code Coverage**: Measure code coverage metrics and discover lines with missing tests on every pull request.
- **Autofix**: Automatically generate fixes for code modernization issues, always under the control of the programmer and preserving 100% code correctness.
- **Reports**: Get a deeper understanding of your organization's code health with powerful insights, modernization reports, and optimization reports.
- **Self-hosting**: Deploy on-prem on your private system within minutes, and retain full control of your source code and privacy.
- **CI/CD automation**: Enable automated testing on all CI systems, test every code change and pull-request to find code issues before merges and public releases.
- **Technical Debt**: Quantify the extent of code refactoring required to modernize your Fortran code.
- **ROI**: Quantify savings in development effort to modernize your code, and tailor the ROI estimation to your organization.

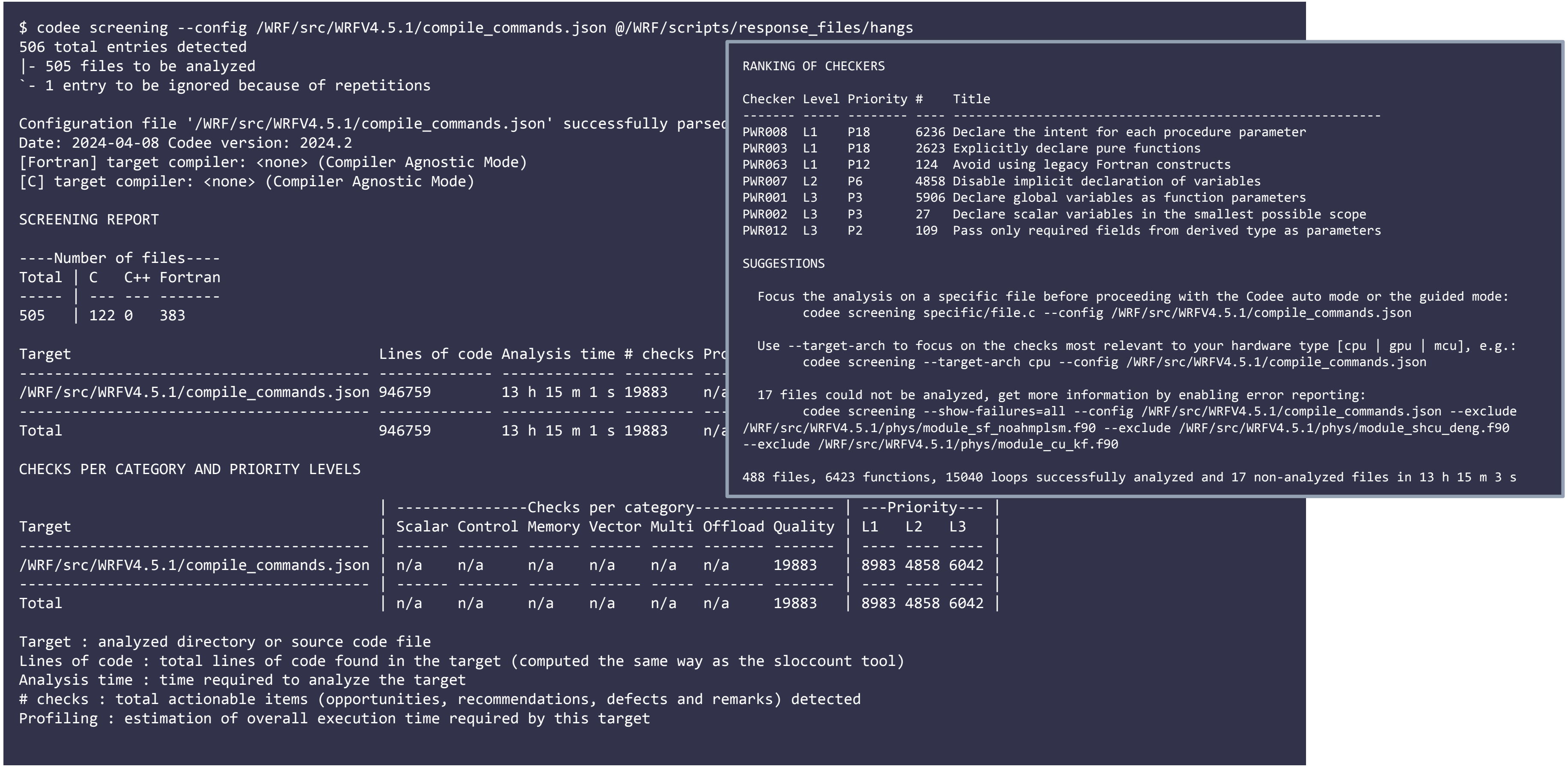
**Codee provides a systematic, predictable workflow**
**that is a complement to the HPE/Cray software development tools**

# WRF | Technical Debt



```
$ codee technical-debt --config /WRF/src/WRFV4.5.1/compile_commands.json @/WRF/scripts/response_files/hangs
506 total entries detected
|- 505 files to be analyzed
`- 1 entry to be ignored because of repetitions

Configuration file '/WRF/src/WRFV4.5.1/compile_commands.json' successfully parsed.
Date: 2024-04-08 Codee version: 2024.2
[Fortran] target compiler: <none> (Compiler Agnostic Mode)
[C] target compiler: <none> (Compiler Agnostic Mode)

TECHNICAL DEBT REPORT

This report quantifies the technical debt associated with the modernization of legacy code by assessing the extent of refactoring required for language constructs. The
score is determined based on the number of language constructs necessitating refactoring to bring the source code up to modern standards. Additionally, the metric
identifies the impacted source code segments, detailing affected files, functions, and loops.

Score Affected files Affected functions Affected loops
----- -------------- ------------------ --------------
26094 355            7798               28

TECHNICAL DEBT BREAKDOWN

Target                                   Lines of code Analysis time  Checkers Technical debt score
---------------------------------------- ------------- -------------- -------- --------------------
/WRF/src/WRFV4.5.1/compile_commands.json 946759        13 h 32 m 17 s 19883    26094
---------------------------------------- ------------- -------------- -------- --------------------
Total                                    946759        13 h 32 m 17 s 19883    26094

The listing of language constructs associated with legacy code found in the source code is as follows:
  - Double precision
  - Assumed size array
  - COMMON blocks
  - BACKSPACE
  - DATA
  - Arithmetic IF
  - PAUSE
  - Equivalence

488 files, 6423 functions, 15040 loops successfully analyzed and 17 non-analyzed files in 13 h 32 m 19 s
```

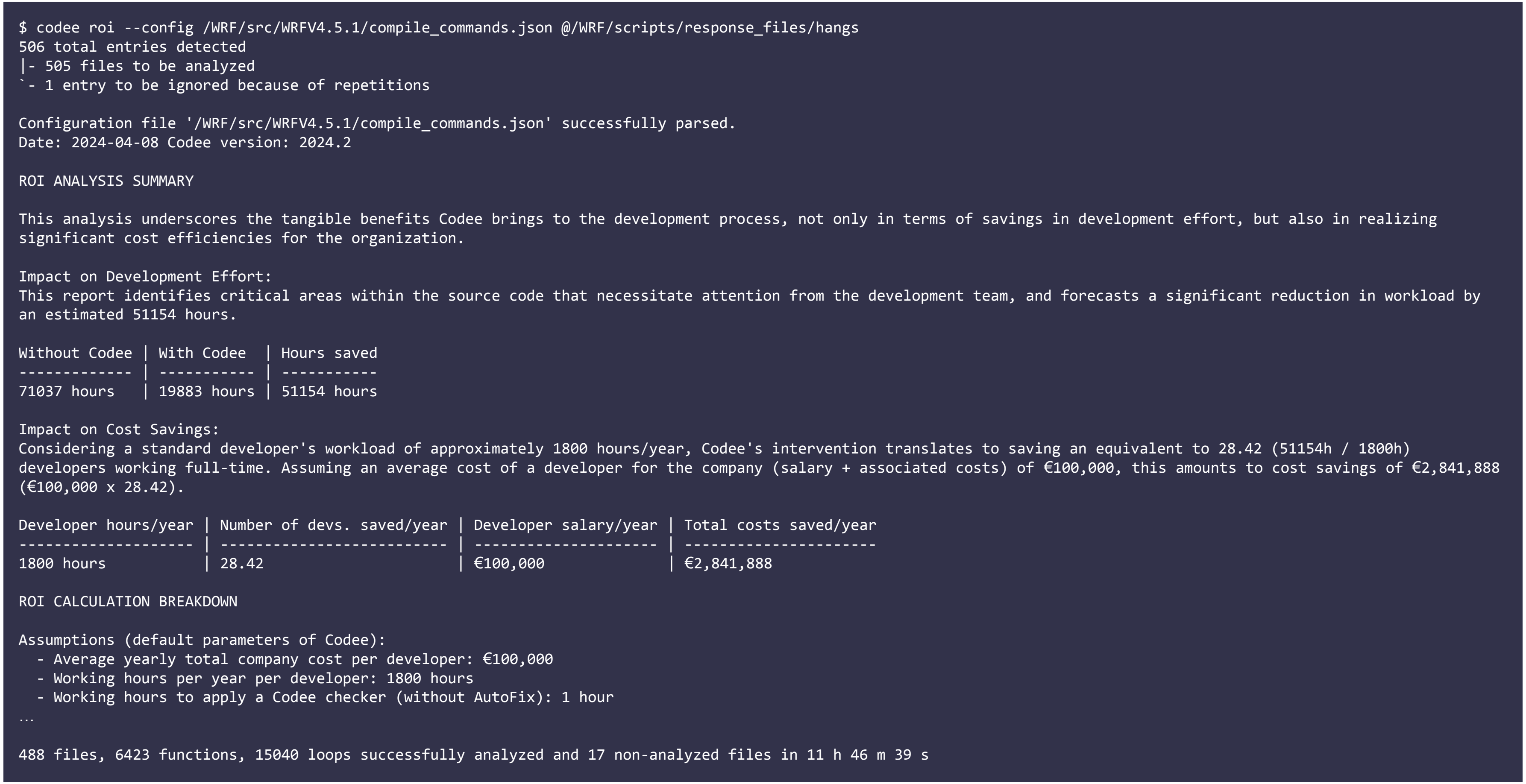
# WRF | Screening with Ranking



```
$ codee screening --config /WRF/src/WRFV4.5.1/compile_commands.json @/WRF/scripts/response_files/hangs
506 total entries detected
|- 505 files to be analyzed
`- 1 entry to be ignored because of repetitions

Configuration file '/WRF/src/WRFV4.5.1/compile_commands.json' successfully parsed
Date: 2024-04-08 Codee version: 2024.2
[Fortran] target compiler: <none> (Compiler Agnostic Mode)
[C] target compiler: <none> (Compiler Agnostic Mode)

SCREENING REPORT

----Number of files----
Total | C   C++ Fortran
----- | --- --- -------
505   | 122 0   383

Target                                   Lines of code Analysis time # checks Pro
---------------------------------------- ------------- ------------- -------- ---
/WRF/src/WRFV4.5.1/compile_commands.json 946759        13 h 15 m 1 s 19883    n/a
---------------------------------------- ------------- ------------- -------- ---
Total                                    946759        13 h 15 m 1 s 19883    n/a

CHECKS PER CATEGORY AND PRIORITY LEVELS

                                         | ---------------Checks per category--------------- | ---Priority--- |
Target                                   | Scalar Control Memory Vector Multi Offload Quality | L1   L2   L3   |
---------------------------------------- | ------ ------- ------ ------ ----- ------- ------- | ---- ---- ---- |
/WRF/src/WRFV4.5.1/compile_commands.json | n/a    n/a     n/a    n/a    n/a   n/a     19883   | 8983 4858 6042 |
---------------------------------------- | ------ ------- ------ ------ ----- ------- ------- | ---- ---- ---- |
Total                                    | n/a    n/a     n/a    n/a    n/a   n/a     19883   | 8983 4858 6042 |

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)
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
------- ----- -------- ---- ---------------------------------------------------------
PWR008  L1    P18      6236 Declare the intent for each procedure parameter
PWR003  L1    P18      2623 Explicitly declare pure functions
PWR063  L1    P12      124  Avoid using legacy Fortran constructs
PWR007  L2    P6       4858 Disable implicit declaration of variables
PWR001  L3    P3       5906 Declare global variables as function parameters
PWR002  L3    P3       27   Declare scalar variables in the smallest possible scope
PWR012  L3    P2       109  Pass only required fields from derived type as parameters

SUGGESTIONS

  Focus the analysis on a specific file before proceeding with the Codee auto mode or the guided mode:
        codee screening specific/file.c --config /WRF/src/WRFV4.5.1/compile_commands.json

  Use --target-arch to focus on the checks most relevant to your hardware type [cpu | gpu | mcu], e.g.:
        codee screening --target-arch cpu --config /WRF/src/WRFV4.5.1/compile_commands.json

  17 files could not be analyzed, get more information by enabling error reporting:
        codee screening --show-failures=all --config /WRF/src/WRFV4.5.1/compile_commands.json --exclude
/WRF/src/WRFV4.5.1/phys/module_sf_noahmplsm.f90 --exclude /WRF/src/WRFV4.5.1/phys/module_shcu_deng.f90
--exclude /WRF/src/WRFV4.5.1/phys/module_cu_kf.f90

488 files, 6423 functions, 15040 loops successfully analyzed and 17 non-analyzed files in 13 h 15 m 3 s
```

# WRF | ROI



```
$ codee roi --config /WRF/src/WRFV4.5.1/compile_commands.json @/WRF/scripts/response_files/hangs
506 total entries detected
|- 505 files to be analyzed
`- 1 entry to be ignored because of repetitions

Configuration file '/WRF/src/WRFV4.5.1/compile_commands.json' successfully parsed.
Date: 2024-04-08 Codee version: 2024.2

ROI ANALYSIS SUMMARY

This analysis underscores the tangible benefits Codee brings to the development process, not only in terms of savings in development effort, but also in realizing
significant cost efficiencies for the organization.

Impact on Development Effort:
This report identifies critical areas within the source code that necessitate attention from the development team, and forecasts a significant reduction in workload by
an estimated 51154 hours.

Without Codee | With Codee  | Hours saved
------------- | ----------- | -----------
71037 hours   | 19883 hours | 51154 hours

Impact on Cost Savings:
Considering a standard developer's workload of approximately 1800 hours/year, Codee's intervention translates to saving an equivalent to 28.42 (51154h / 1800h)
developers working full-time. Assuming an average cost of a developer for the company (salary + associated costs) of €100,000, this amounts to cost savings of €2,841,888
(€100,000 x 28.42).

Developer hours/year | Number of devs. saved/year | Developer salary/year | Total costs saved/year
-------------------- | -------------------------- | --------------------- | ----------------------
1800 hours           | 28.42                      | €100,000              | €2,841,888

ROI CALCULATION BREAKDOWN

Assumptions (default parameters of Codee):
  - Average yearly total company cost per developer: €100,000
  - Working hours per year per developer: 1800 hours
  - Working hours to apply a Codee checker (without AutoFix): 1 hour
…

488 files, 6423 functions, 15040 loops successfully analyzed and 17 non-analyzed files in 11 h 46 m 39 s
```

# Usage of Codee: Command-Line Interface Tool

**List of Codee reports to get started:**

- Technical debt report: `codee technical-debt <input>`
- Screening report: `codee screening <input>`
- ROI report: `codee roi <input>`

**Codee reports linking with the Github Open Catalog:**

- Checks report: `codee checks [--verbose] <input>`

**GitHub Catalog Link**
github.com/codee-com/open-catalog

**Additional Codee features for performance optimization:**

- Annotate OpenMP: `codee rewrite <input>`

**List of checkers related to Fortran modernization and optimization:**

| | **Modernization** | **Performance** |
|---|---|---|
| **Checks** | PWR001, PWR002, PWR003, PWR007, PWR008, PWR012, PWR063. | PWR051, PWR054, PWR039, PWR055 and many more... |

# Codee Help: Usage of Codee command-line

```
Usage:
  codee <command> --config <compile_commands> [OPTIONS] <filter>...
  codee <command> [OPTIONS] <input>... [-- <compiler flags>]

Arguments:
  <filter>
          Determine which parts of the inputs will be analyzed. It is composed
          of a filepath, followed by an optional list of function names or
          specific positions in the file. For specifying positions, use the
          format "line number:column number". Use commas to separate items in
          the list. For instance:
              path/to/file.ext:foo,bar
              test.c:3:2,2

  <input>
          Determine the files to analyze. Follows the same syntax as <filter>

  <compiler flags>
          A gcc-compatible list of compiler options to apply to the <input>
          files
```

# Codee Help: Main commands and basic options

```
Commands:
  checks
          Report opportunities, recommendations and other actionable items found
          in the input(s)

  rewrite
          Apply an AutoFix

  roi
          Estimate the financial impact that Codee will eventually have on the
          codebase

  screening
          Print a screening report of the given input(s)

  technical-debt
          Generate a technical debt report on the modernization of legacy code


Common options:

      --config <config file>
          Load the analysis options from the specified configuration file

      --show-progress, --show-progress=<none|files|functions>
          Show how the analysis progresses by printing a message for each input
          file or function (defaults to `files`)
```

# Codee Help: Options to select subsets of checkers

```
Common options:

      --check-id <id>[,<id>]*
          Enable the checks that match the specified ID(s) only

      --target-arch <arch>
          Filter the checks by target architecture

      --include-categories <category>[,<category>]*
          Enable the checks that match the specified categories, in addition to
          those enabled by default

      --only-categories <category>[,<category>]*
          Enable the checks that match the specified categories only

      --level <L1|1|high|L2|2|medium|L3|3|low>
          Filter the checks by priority level

      --list-available-checkers
          List all available defects, recommendations and remarks
```

# Codee Help: Options to filter input files/directories

```
--lang <language>
    Filter the input files by language (C, C++, Fortran)

--exclude <file|directory>
    Skip the specified file or directory. `--exclude` may be set several
    times

--no-warnings
    Disable warning messages

--brief
    Minimize the verbosity of the output by omitting table legends,
    suggestions and others
```

# Codee Technical Debt Report

```
$ codee technical-debt himeno.f90
. . .
TECHNICAL DEBT REPORT

This report quantifies the technical debt associated with the modernization of legacy code by assessing the
extent of refactoring required for language constructs. The score is determined based on the number of
language constructs necessitating refactoring to bring the source code up to modern standards. Additionally,
the metric identifies the impacted source code segments, detailing affected files, functions, and loops.

Score Affected files Affected functions Affected loops
----- -------------- ------------------ --------------
10    1              6                  3

TECHNICAL DEBT BREAKDOWN

Lines of code Analysis time Checkers Technical debt score
------------- ------------- -------- --------------------
214           224 ms        10       10

The listing of language constructs associated with legacy code found in the source code is as follows:
  - PAUSE

1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 225 ms
```

Score and affected source code

# Codee Screening with Ranking Report

```
$ codee screening himeno.f90
. . .
SCREENING REPORT

Lines of code Analysis time # checks Profiling
------------- ------------- -------- ---------
214           194 ms        10       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      n/a   n/a     6       | 3  0  7  |

RANKING OF CHECKERS

Checker Level Priority # Title
------- ----- -------- - ------------------------------------------------------------------
RMK015  L1    P27      1 Tune compiler optimization flags to increase the speed of the code
PWR054  L1    P12      1 Consider applying vectorization to scalar reduction loop
PWR063  L1    P12      1 Avoid using legacy Fortran constructs
PWR001  L3    P3       5 Declare global variables as function parameters
PWR035  L3    P2       2 Avoid non-consecutive array access to improve performance

1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 195 ms
```

Total number of checkers triggered

Checkers per category/priority

List of checkers reported, ordered by priority

# Codee ROI Report

```
$ codee roi himeno.f90
. . .
ROI ANALYSIS SUMMARY

This analysis underscores the tangible benefits Codee brings to the development process, not only in terms of savings
in development effort, but also in realizing significant cost efficiencies for the organization.

Impact on Development Effort:
This report identifies critical areas within the source code that necessitate attention from the development team, and
forecasts a significant reduction in workload by an estimated 292 hours.

Without Codee | With Codee | Hours saved
------------- | ---------- | -----------
302 hours     | 10 hours   | 292 hours

Impact on Cost Savings:
Considering a standard developer's workload of approximately 1800 hours/year, Codee's intervention translates to saving
an equivalent to 0.16 (292h / 1800h) developers working full-time. Assuming an average cost of a developer for the
company (salary + associated costs) of €100,000, this amounts to cost savings of €16,222 (€100,000 x 0.16).

Developer hours/year | Number of devs. saved/year | Developer salary/year | Total costs saved/year
-------------------- | -------------------------- | --------------------- | ----------------------
1800 hours           | 0.16                       | €100,000              | €16,222
. . .
```

Saved hours

Saved costs

# Codee Checks Report

```
$ codee checks himeno.f90
. . .
CHECKS REPORT

himeno.f90 [PWR063] (level: L1): Avoid using legacy Fortran constructs
himeno.f90:136:1 [PWR001] (level: L3): Declare global variables as function parameters
himeno.f90:164:1 [PWR001] (level: L3): Declare global variables as function parameters
himeno.f90:223:1 [PWR001] (level: L3): Declare global variables as function parameters
himeno.f90:255:1 [PWR001] (level: L3): Declare global variables as function parameters
himeno.f90:275:1 [PWR001] (level: L3): Declare global variables as function parameters

1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 188 ms
```

```
$ codee checks --verbose --check-id pwr063 himeno.f90
. . .
himeno.f90 [PWR063] (level: L1): Avoid using legacy Fortran constructs
  PAUSE:
    131:   pause
  Suggestion: Remove the legacy fortran constructs and refactor the code to comply with modern Fortran standards.
  Documentation: https://github.com/codee-com/open-catalog/tree/main/Checks/PWR063
. . .
1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 145 ms
```

# Codee Rewrite (I)

First, run Codee to produce the Checks Report in verbose mode. For those checks that have AutoFix capabilities, the tool will suggest invocations of the *codee rewrite* command.

```
$ codee checks --verbose --check-id pwr051 himeno.f90
. . .
himeno.f90:293:6 [PWR051] (level: L2): Consider applying multithreading parallelism to scalar reduction loop
  Suggestion: Use 'rewrite' to automatically optimize the code
  Documentation: https://github.com/codee-com/open-catalog/tree/main/Checks/PWR051
  AutoFix (choose one option):
      * Using OpenMP 'for' with built-in reduction (recommended):
        codee rewrite --multi omp-for --in-place himeno.f90:293:6
      * Using OpenMP 'for' with explicit privatization:
        codee rewrite --multi omp-for --in-place --explicit-privatization gosa himeno.f90:293:6
      * Using OpenMP 'taskwait':
        codee rewrite --multi omp-taskwait --in-place himeno.f90:293:6
      * Using OpenMP 'taskloop':
        codee rewrite --multi omp-taskloop --in-place himeno.f90:293:6

. . .

1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 145 ms
```

# Codee Rewrite (II)

Second, run Codee to annotate the source code with OpenMP multithreading directives. The tool will provide details about the actual changes implemented in the source code.

```
$ codee rewrite --multi omp-for --in-place himeno.f90:293:6
. . .
Results for file 'himeno.f90':
  Successfully applied AutoFix to the loop at 'himeno.f90:jacobi:293:6' [using multi-threading]:
      [INFO] himeno.f90:293:6 Parallel scalar reduction pattern identified for variable 'gosa' with associative,
commutative operator '+'
      [INFO] himeno.f90:293:6 Parallel forall: variable 'wrk2'
      [INFO] himeno.f90:293:6 Available parallelization strategies for variable 'gosa'
      [INFO] himeno.f90:293:6   #1 OpenMP scalar reduction (* implemented)
      [INFO] himeno.f90:293:6   #2 OpenMP atomic access
      [INFO] himeno.f90:293:6   #3 OpenMP explicit privatization
      [INFO] himeno.f90:293:6 Loop parallelized with multithreading using OpenMP directive 'for'
      [INFO] himeno.f90:293:6 Parallel region defined by OpenMP directive 'parallel'

Successfully updated himeno.f90

Minimum software stack requirements: OpenMP version 3.0 with multithreading capabilities
```

# Codee Rewrite (and III)

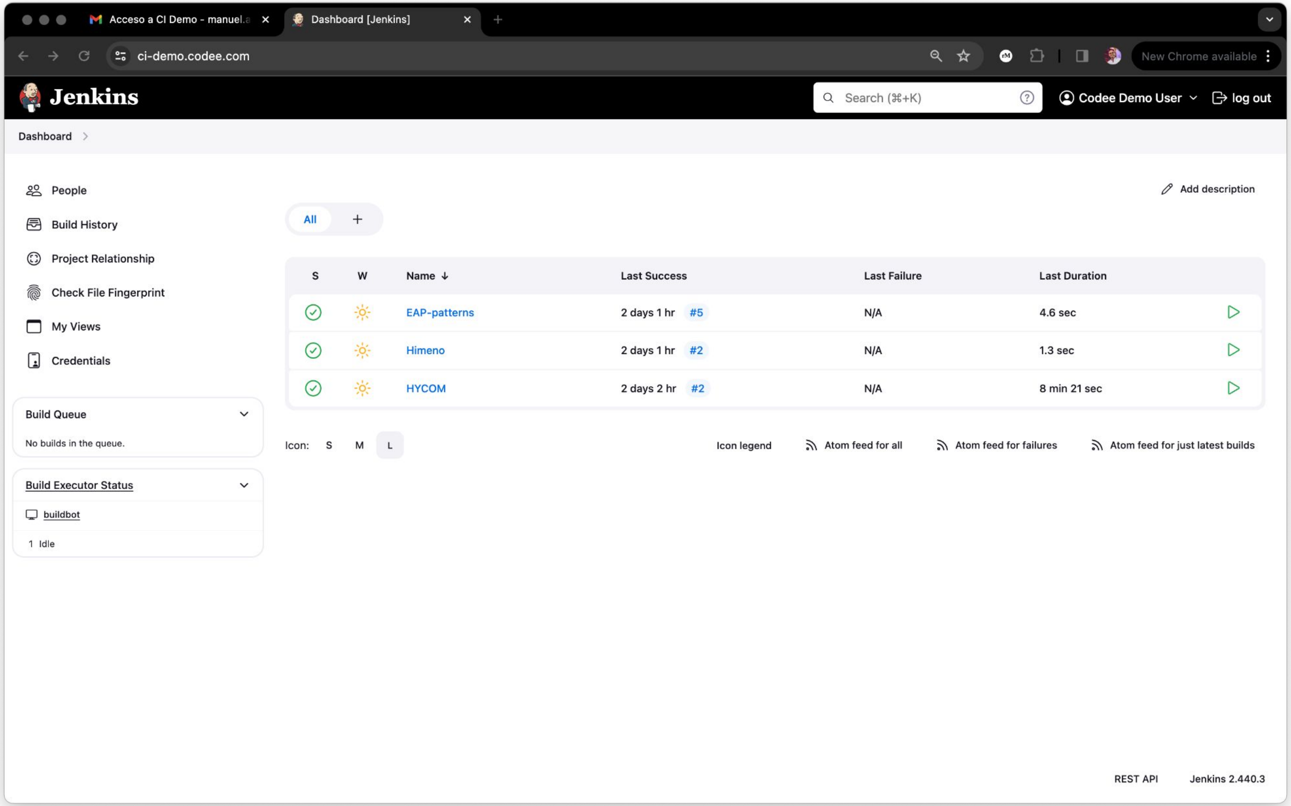
Finally, review the source code comparing the original code and the optimized code. The tool just adds annotations of OpenMP directives. As a coding assistant tool does not replace proper testing and benchmarking of the optimized code on your target hardware.

```
! Codee: Loop modified by Codee (2024-04-29 11:40:52)
! Codee: Technique applied: multithreading with 'omp-for' pragmas
!$omp parallel default(none) shared(a, b, bnd, c, gosa, imax, jmax, kmax, p, wrk1, wrk2) private(i, j, k, s0, ss)
!$omp do private(i, j, s0, ss) reduction(+: gosa) schedule(auto)
do k=2,kmax-1
   do j=2,jmax-1
      do i=2,imax-1
         s0=a(I,J,K,1)*p(I+1,J,K) &
              +a(I,J,K,2)*p(I,J+1,K) &
              +a(I,J,K,3)*p(I,J,K+1) &
              +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K) &
                          -p(I-1,J+1,K)+p(I-1,J-1,K)) &
              +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1) &
                          -p(I,J+1,K-1)+p(I,J-1,K-1)) &
              +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1) &
                          -p(I+1,J,K-1)+p(I-1,J,K-1)) &
              +c(I,J,K,1)*p(I-1,J,K) &
              +c(I,J,K,2)*p(I,J-1,K) &
              +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
         ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
         GOSA=GOSA+SS*SS
        wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
      enddo
    enddo
enddo
!$omp end parallel
```

# Codee Help: Options for CI/CD pipelines

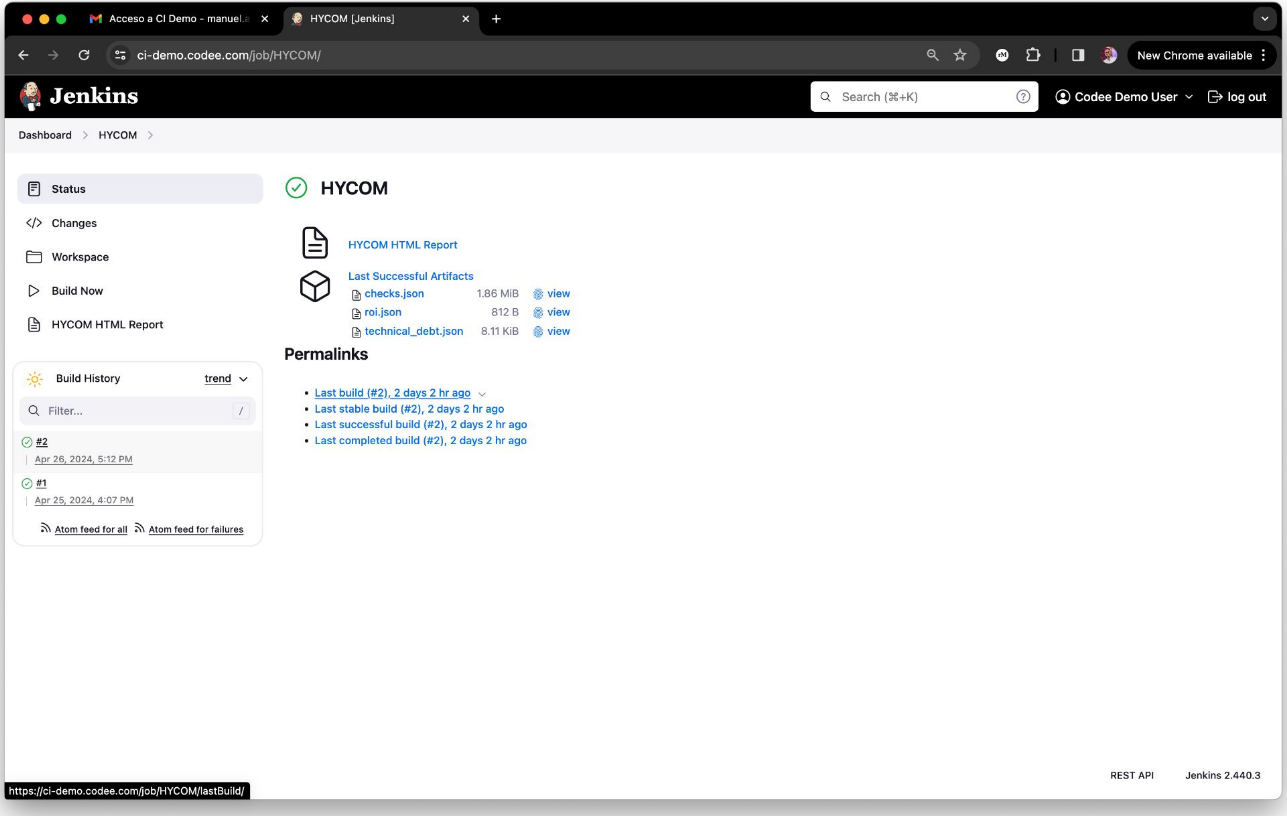
```
--json
    Output results in JSON format

--csv
    Output results in CSV format

--accept-eula
    Confirm the acceptance of the EULA
```

# Codee Invocation in CI/CD Pipelines



| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| ✓ | ☀ | EAP-patterns | 2 days 1 hr #5 | N/A | 4.6 sec |
| ✓ | ☀ | Himeno | 2 days 1 hr #2 | N/A | 1.3 sec |
| ✓ | ☀ | HYCOM | 2 days 2 hr #2 | N/A | 8 min 21 sec |

# Codee Invocation in CI/CD Pipelines



Dashboard > HYCOM >

- Status
- Changes
- Workspace
- Build Now
- HYCOM HTML Report

Build History trend

Filter...

#2 Apr 26, 2024, 5:12 PM

#1 Apr 25, 2024, 4:07 PM

Atom feed for all Atom feed for failures

## HYCOM

HYCOM HTML Report

Last Successful Artifacts

| | | |
|---|---|---|
| checks.json | 1.86 MiB | view |
| roi.json | 812 B | view |
| technical_debt.json | 8.11 KiB | view |

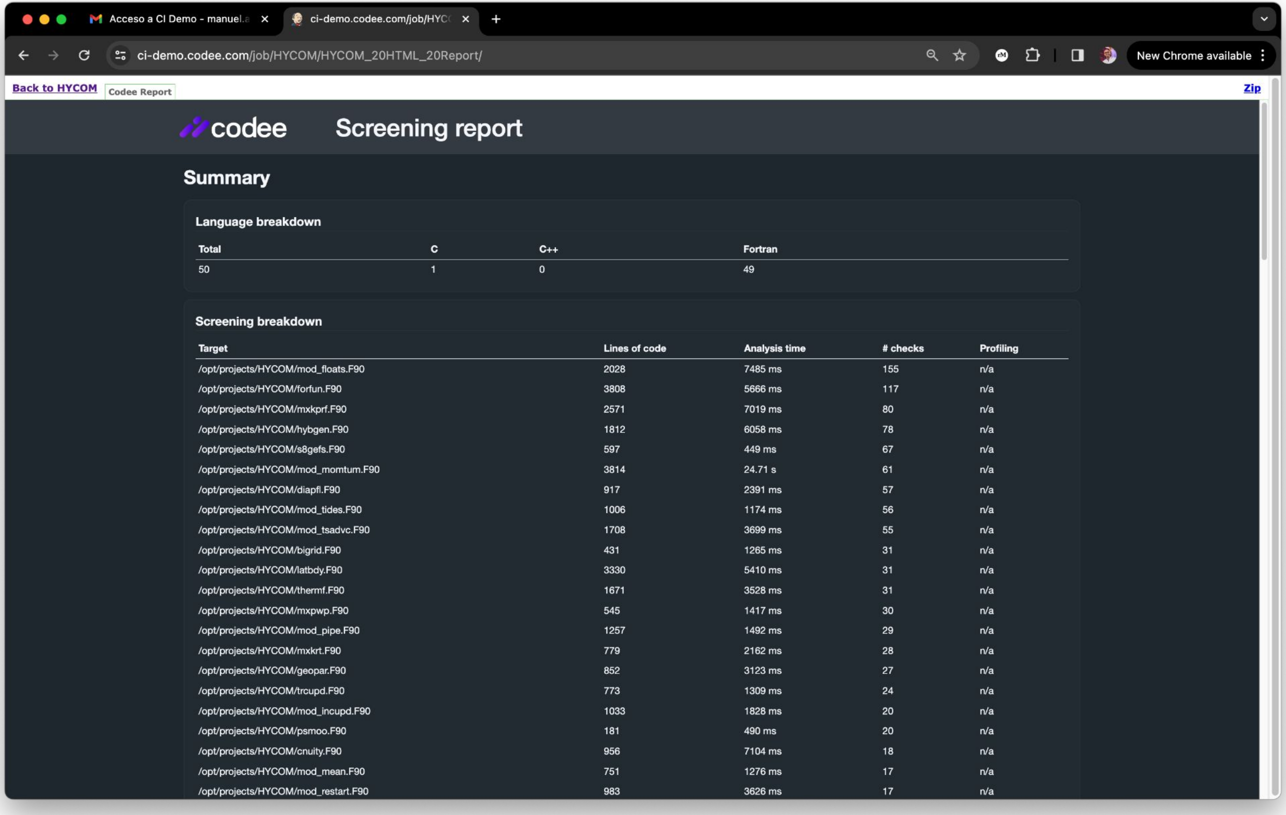
### Permalinks

- Last build (#2), 2 days 2 hr ago
- Last stable build (#2), 2 days 2 hr ago
- Last successful build (#2), 2 days 2 hr ago
- Last completed build (#2), 2 days 2 hr ago

REST API Jenkins 2.440.3

https://ci-demo.codee.com/job/HYCOM/lastBuild/

# Codee Invocation in CI/CD Pipelines


Acceso a CI Demo - manuel.a
ci-demo.codee.com/job/HYC
ci-demo.codee.com/job/HYCOM/HYCOM_20HTML_20Report/
New Chrome available
Back to HYCOM
Codee Report
Zip
codee

## Screening report

### Summary

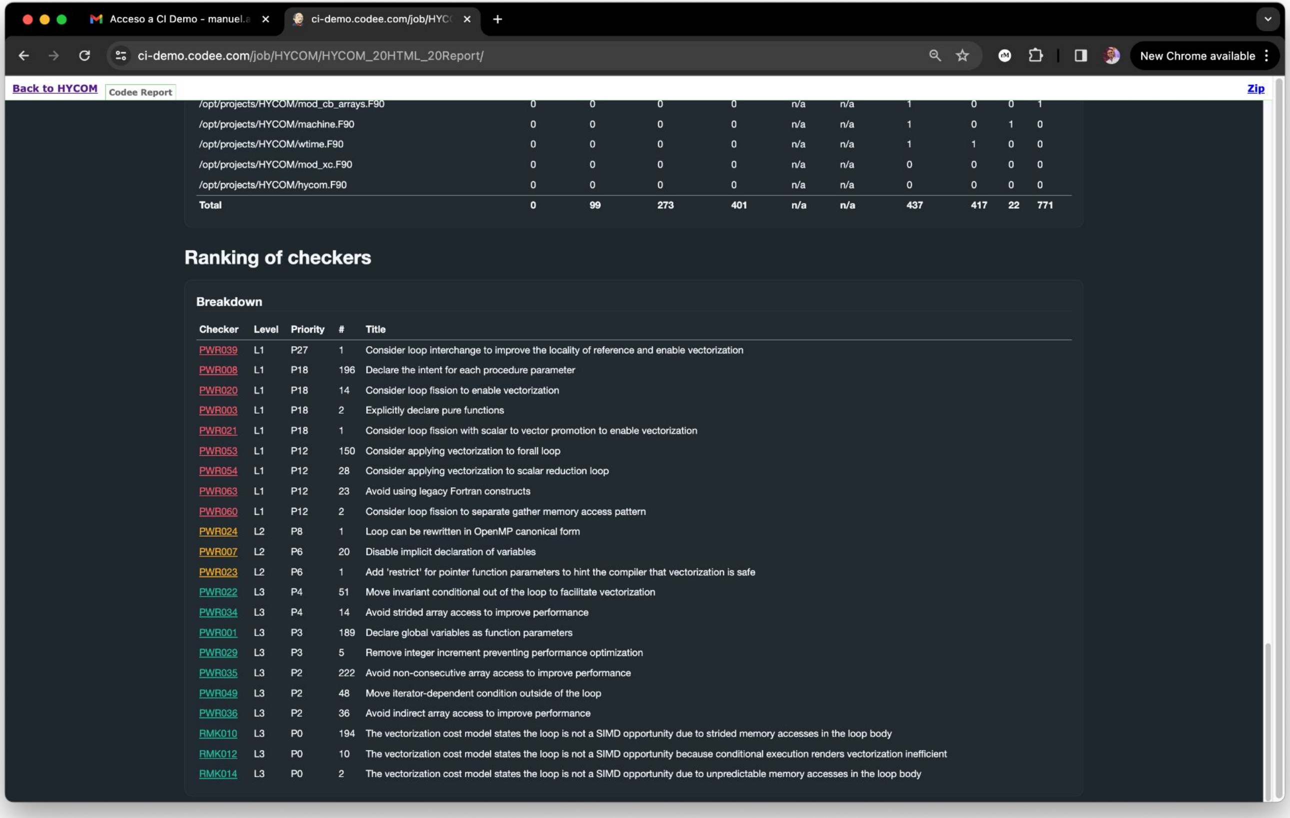
**Language breakdown**

| Total | C | C++ | Fortran |
|---|---|---|---|
| 50 | 1 | 0 | 49 |

**Screening breakdown**

| Target | Lines of code | Analysis time | # checks | Profiling |
|---|---|---|---|---|
| /opt/projects/HYCOM/mod_floats.F90 | 2028 | 7485 ms | 155 | n/a |
| /opt/projects/HYCOM/forfun.F90 | 3808 | 5666 ms | 117 | n/a |
| /opt/projects/HYCOM/mxkprf.F90 | 2571 | 7019 ms | 80 | n/a |
| /opt/projects/HYCOM/hybgen.F90 | 1812 | 6058 ms | 78 | n/a |
| /opt/projects/HYCOM/s8gefs.F90 | 597 | 449 ms | 67 | n/a |
| /opt/projects/HYCOM/mod_momtum.F90 | 3814 | 24.71 s | 61 | n/a |
| /opt/projects/HYCOM/diapfl.F90 | 917 | 2391 ms | 57 | n/a |
| /opt/projects/HYCOM/mod_tides.F90 | 1006 | 1174 ms | 56 | n/a |
| /opt/projects/HYCOM/mod_tsadvc.F90 | 1708 | 3699 ms | 55 | n/a |
| /opt/projects/HYCOM/bigrid.F90 | 431 | 1265 ms | 31 | n/a |
| /opt/projects/HYCOM/latbdy.F90 | 3330 | 5410 ms | 31 | n/a |
| /opt/projects/HYCOM/thermf.F90 | 1671 | 3528 ms | 31 | n/a |
| /opt/projects/HYCOM/mxpwp.F90 | 545 | 1417 ms | 30 | n/a |
| /opt/projects/HYCOM/mod_pipe.F90 | 1257 | 1492 ms | 29 | n/a |
| /opt/projects/HYCOM/mxkrt.F90 | 779 | 2162 ms | 28 | n/a |
| /opt/projects/HYCOM/geopar.F90 | 852 | 3123 ms | 27 | n/a |
| /opt/projects/HYCOM/trcupd.F90 | 773 | 1309 ms | 24 | n/a |
| /opt/projects/HYCOM/mod_incupd.F90 | 1033 | 1828 ms | 20 | n/a |
| /opt/projects/HYCOM/psmoo.F90 | 181 | 490 ms | 20 | n/a |
| /opt/projects/HYCOM/cnuity.F90 | 956 | 7104 ms | 18 | n/a |
| /opt/projects/HYCOM/mod_mean.F90 | 751 | 1276 ms | 17 | n/a |
| /opt/projects/HYCOM/mod_restart.F90 | 983 | 3626 ms | 17 | n/a |

# Codee Invocation in CI/CD Pipelines



Acceso a CI Demo - manuel.e
ci-demo.codee.com/job/HYC
ci-demo.codee.com/job/HYCOM/HYCOM_20HTML_20Report/
New Chrome available
Back to HYCOM
Codee Report
Zip

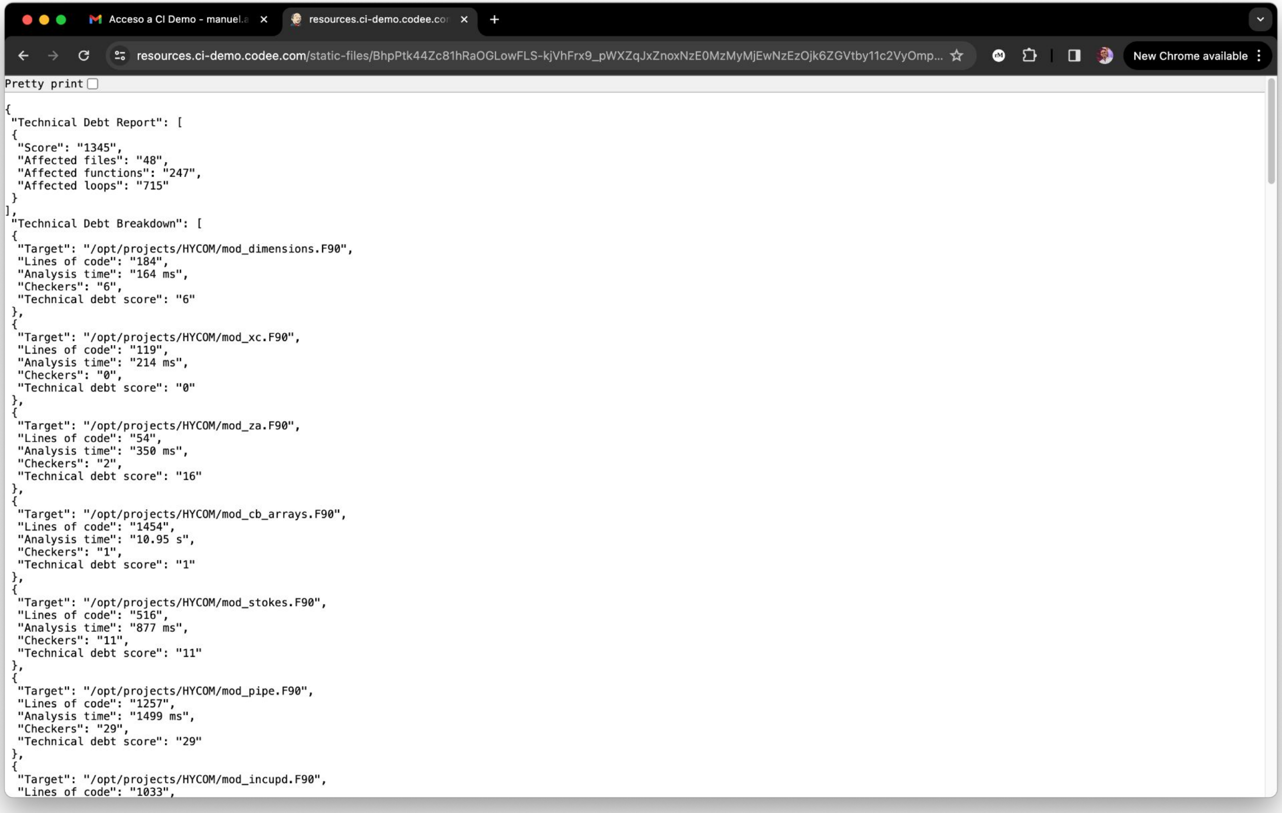
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| /opt/projects/HYCOM/mod_cb_arrays.F90 | 0 | 0 | 0 | 0 | n/a | n/a | 1 | 0 | 0 | 1 |
| /opt/projects/HYCOM/machine.F90 | 0 | 0 | 0 | 0 | n/a | n/a | 1 | 0 | 1 | 0 |
| /opt/projects/HYCOM/wtime.F90 | 0 | 0 | 0 | 0 | n/a | n/a | 1 | 1 | 0 | 0 |
| /opt/projects/HYCOM/mod_xc.F90 | 0 | 0 | 0 | 0 | n/a | n/a | 0 | 0 | 0 | 0 |
| /opt/projects/HYCOM/hycom.F90 | 0 | 0 | 0 | 0 | n/a | n/a | 0 | 0 | 0 | 0 |
| **Total** | **0** | **99** | **273** | **401** | **n/a** | **n/a** | **437** | **417** | **22** | **771** |

## Ranking of checkers

### Breakdown

| Checker | Level | Priority | # | Title |
|---|---|---|---|---|
| PWR039 | L1 | P27 | 1 | Consider loop interchange to improve the locality of reference and enable vectorization |
| PWR008 | L1 | P18 | 196 | Declare the intent for each procedure parameter |
| PWR020 | L1 | P18 | 14 | Consider loop fission to enable vectorization |
| PWR003 | L1 | P18 | 2 | Explicitly declare pure functions |
| PWR021 | L1 | P18 | 1 | Consider loop fission with scalar to vector promotion to enable vectorization |
| PWR053 | L1 | P12 | 150 | Consider applying vectorization to forall loop |
| PWR054 | L1 | P12 | 28 | Consider applying vectorization to scalar reduction loop |
| PWR063 | L1 | P12 | 23 | Avoid using legacy Fortran constructs |
| PWR060 | L1 | P12 | 2 | Consider loop fission to separate gather memory access pattern |
| PWR024 | L2 | P8 | 1 | Loop can be rewritten in OpenMP canonical form |
| PWR007 | L2 | P6 | 20 | Disable implicit declaration of variables |
| PWR023 | L2 | P6 | 1 | Add 'restrict' for pointer function parameters to hint the compiler that vectorization is safe |
| PWR022 | L3 | P4 | 51 | Move invariant conditional out of the loop to facilitate vectorization |
| PWR034 | L3 | P4 | 14 | Avoid strided array access to improve performance |
| PWR001 | L3 | P3 | 189 | Declare global variables as function parameters |
| PWR029 | L3 | P3 | 5 | Remove integer increment preventing performance optimization |
| PWR035 | L3 | P2 | 222 | Avoid non-consecutive array access to improve performance |
| PWR049 | L3 | P2 | 48 | Move iterator-dependent condition outside of the loop |
| PWR036 | L3 | P2 | 36 | Avoid indirect array access to improve performance |
| RMK010 | L3 | P0 | 194 | The vectorization cost model states the loop is not a SIMD opportunity due to strided memory accesses in the loop body |
| RMK012 | L3 | P0 | 10 | The vectorization cost model states the loop is not a SIMD opportunity because conditional execution renders vectorization inefficient |
| RMK014 | L3 | P0 | 2 | The vectorization cost model states the loop is not a SIMD opportunity due to unpredictable memory accesses in the loop body |

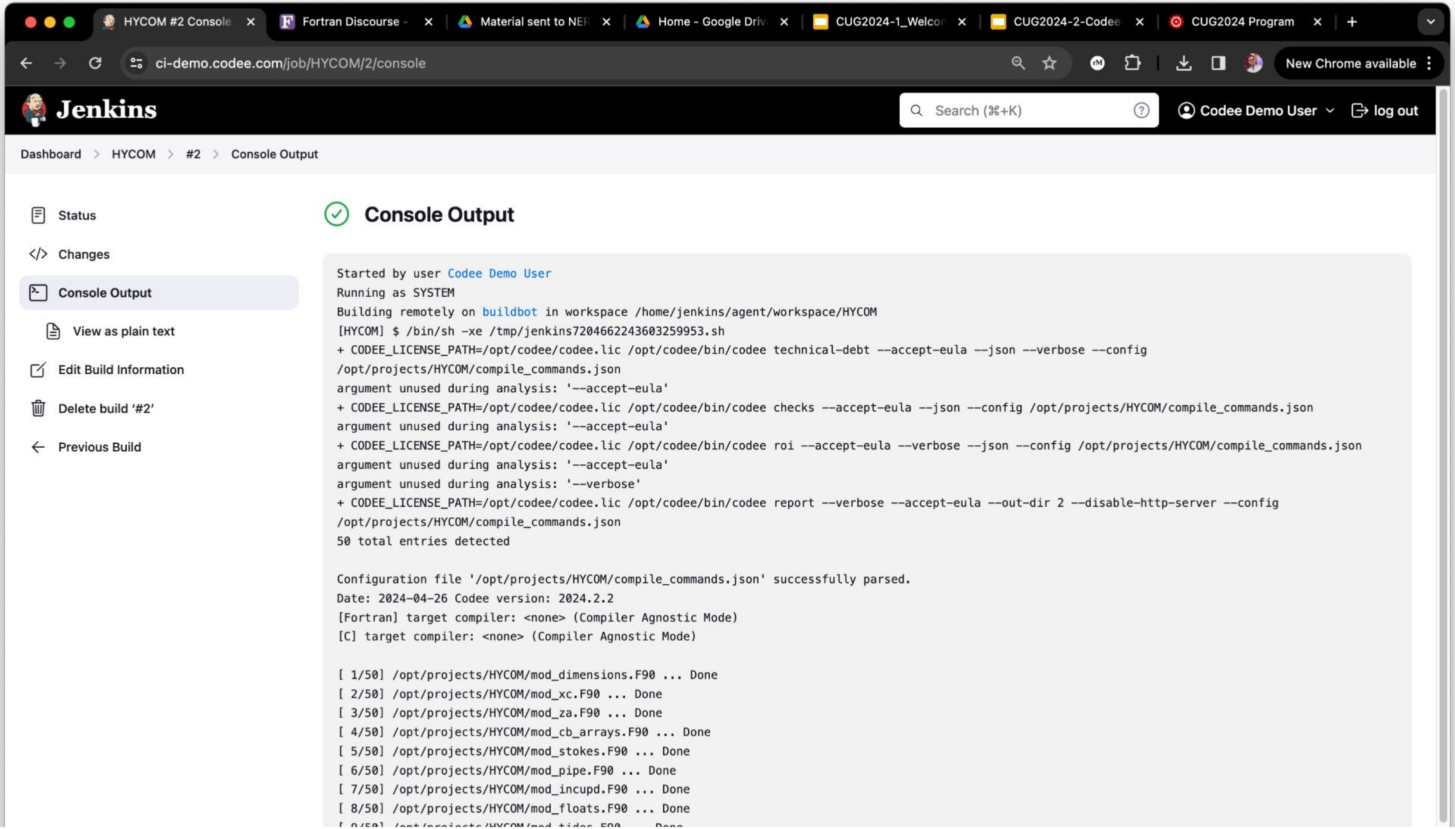
# Codee Invocation in CI/CD Pipelines


Acceso a CI Demo - manuel.a
resources.ci-demo.codee.cor
resources.ci-demo.codee.com/static-files/BhpPtk44Zc81hRaOGLowFLS-kjVhFrx9_pWXZqJxZnoxNzE0MzMyMjEwNzEzOjk6ZGVtby11c2VyOmp...
New Chrome available

Pretty print □

```
{
 "Technical Debt Report": [
 {
  "Score": "1345",
  "Affected files": "48",
  "Affected functions": "247",
  "Affected loops": "715"
 }
],
 "Technical Debt Breakdown": [
 {
  "Target": "/opt/projects/HYCOM/mod_dimensions.F90",
  "Lines of code": "184",
  "Analysis time": "164 ms",
  "Checkers": "6",
  "Technical debt score": "6"
 },
 {
  "Target": "/opt/projects/HYCOM/mod_xc.F90",
  "Lines of code": "119",
  "Analysis time": "214 ms",
  "Checkers": "0",
  "Technical debt score": "0"
 },
 {
  "Target": "/opt/projects/HYCOM/mod_za.F90",
  "Lines of code": "54",
  "Analysis time": "350 ms",
  "Checkers": "2",
  "Technical debt score": "16"
 },
 {
  "Target": "/opt/projects/HYCOM/mod_cb_arrays.F90",
  "Lines of code": "1454",
  "Analysis time": "10.95 s",
  "Checkers": "1",
  "Technical debt score": "1"
 },
 {
  "Target": "/opt/projects/HYCOM/mod_stokes.F90",
  "Lines of code": "516",
  "Analysis time": "877 ms",
  "Checkers": "11",
  "Technical debt score": "11"
 },
 {
  "Target": "/opt/projects/HYCOM/mod_pipe.F90",
  "Lines of code": "1257",
  "Analysis time": "1499 ms",
  "Checkers": "29",
  "Technical debt score": "29"
 },
 {
  "Target": "/opt/projects/HYCOM/mod_incupd.F90",
  "Lines of code": "1033",
```

# Codee Invocation in CI/CD Pipelines



## Console Output

```
Started by user Codee Demo User
Running as SYSTEM
Building remotely on buildbot in workspace /home/jenkins/agent/workspace/HYCOM
[HYCOM] $ /bin/sh -xe /tmp/jenkins7204662243603259953.sh
+ CODEE_LICENSE_PATH=/opt/codee/codee.lic /opt/codee/bin/codee technical-debt --accept-eula --json --verbose --config
/opt/projects/HYCOM/compile_commands.json
argument unused during analysis: '--accept-eula'
+ CODEE_LICENSE_PATH=/opt/codee/codee.lic /opt/codee/bin/codee checks --accept-eula --json --config /opt/projects/HYCOM/compile_commands.json
argument unused during analysis: '--accept-eula'
+ CODEE_LICENSE_PATH=/opt/codee/codee.lic /opt/codee/bin/codee roi --accept-eula --verbose --json --config /opt/projects/HYCOM/compile_commands.json
argument unused during analysis: '--accept-eula'
argument unused during analysis: '--verbose'
+ CODEE_LICENSE_PATH=/opt/codee/codee.lic /opt/codee/bin/codee report --verbose --accept-eula --out-dir 2 --disable-http-server --config
/opt/projects/HYCOM/compile_commands.json
50 total entries detected

Configuration file '/opt/projects/HYCOM/compile_commands.json' successfully parsed.
Date: 2024-04-26 Codee version: 2024.2.2
[Fortran] target compiler: <none> (Compiler Agnostic Mode)
[C] target compiler: <none> (Compiler Agnostic Mode)

[ 1/50] /opt/projects/HYCOM/mod_dimensions.F90 ... Done
[ 2/50] /opt/projects/HYCOM/mod_xc.F90 ... Done
[ 3/50] /opt/projects/HYCOM/mod_za.F90 ... Done
[ 4/50] /opt/projects/HYCOM/mod_cb_arrays.F90 ... Done
[ 5/50] /opt/projects/HYCOM/mod_stokes.F90 ... Done
[ 6/50] /opt/projects/HYCOM/mod_pipe.F90 ... Done
[ 7/50] /opt/projects/HYCOM/mod_incupd.F90 ... Done
[ 8/50] /opt/projects/HYCOM/mod_floats.F90 ... Done
[ 9/50] /opt/projects/HYCOM/mod_tides.F90 ... Done
```

# Labs

**Main quickstart guides for the course:**

- Quickstart - Fortran modernization - Himeno
- Quickstart - Fortran modernization - Himeno (with compile_commands.json)
- Quickstart - Fortran modernization - HYCOM
- Quickstart - Fortran performance - Himeno

**Optional quickstart guides:**

- Quickstart - Fortran performance - MATMUL
- Quickstart - C performance - MATMUL

**Extra resources:**

- Quickstart - VSCode SARIF
- performance-demos Github repository
- performance-demos-fortran Github repository

codee

Automated Code Inspection for Performance

**www.codee.com**

info@codee.com

Subscribe: codee.com/newsletter/

Spain

codee_com

in /codee-com/