Quickstart Guide of Codee
Nuclear Structure and Nuclear Reactions (NUCCOR)
Step-by-Step in Perlmutter @NERSC

Step 1. Setup the environment

Load Codee last version and the GNU programming environment [1] through the modules management system:

```
$ module load codee
$ module load PrgEnv-gnu
```

Run Codee configuration wizard to set up the compiler used to generate executables.In this case, select the option "*No target compiler (Skip)*" [2], meaning that Codee will not attempt to interpret the user messages emitted by the compiler.

```
$ pwreport --configuration-wizard
```

Step 2. Connect to a CPU node

Pelmutter has CPU and GPU nodes. To run an interactive job in a CPU node [3]:

```
$ salloc --nodes 1 --qos interactive --time 01:00:00 --constraint cpu --reservation=codee_day2_cpu -A ntrain8
```

[1] Note we will use the Nvidia compiler *nvc* to generate executables for GPUs.
[2] Note Codee 2023.1 can be used to generate code for any compiler. The tool provides integrations with the GNU, LLVM, Intel and Microsoft compilers to analyze the compiler optimizations (e.g. compiler's vectorization report).
[3] Note that the *--reservation* "*codee_day2_cpu*" may need to be renamed to any other reservation available to the user.

Step 3: Understanding the implementation of NUCCOR

NUCCOR is a nuclear physics application for deciphering the structure and reactions of atomic nuclei (https://www.olcf.ornl.gov/caar/summit-caar/nuccor/) developed at the Oak Ridge National Laboratory (ORNL). The implementation of the code [4] is written in the Fortran programming language and consists of the following files: *Makefile*, *mtc.F90*, *mtc_main.F90*, *mtc_openmp.F90* and *mtc_patch.f90*.

Later in this step-by-step guide we will use the following Fortran source code snippet to illustrate the usage of Code capabilities:

```
 1
 2       subroutine contract_simple(dst, src, op)
 3           use, intrinsic :: iso_fortran_env, only : int64, real64
 4
 5           real(real64), dimension(:,:,:,:), intent(inout) :: dst
 6           real(real64), dimension(:,:,:,:,:,:), intent(in) :: src
 7           real(real64), dimension(:,:,:,:), intent(in) :: op
 8
 9           integer :: i, j, m, a, b, e, f
10           real(real64) :: temp
11
12           do j = 1, size(dst, 4)
13               do i = 1, size(dst, 3)
14                   do b = 1, size(dst, 2)
15                       do a = 1, size(dst, 1)
16                           temp = 0.0d0
17                           do m = 1, size(op, 1)
18                               do f = 1, size(op, 3)
19                                   do e = 1, size(op, 4)
20                                       temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
21                                   end do
22                               end do
23                           end do
24                           dst(a, b, i, j) = 0.5d0*temp
25                       end do
26                   end do
27               end do
28           end do
29       end subroutine contract_simple
```

Step 4: Producing the Codee Screening Report

The Codee Screening Report is intended to help assess the speed-up potential of the code. Invoke the *pwreport --screening* command, enabling multithreading and GPU offload (*--include-tags all*). The Codee best practice recommendation is to start with the analysis of the open-source project as a whole. For this purpose, let's take advantage of Codee integration with build systems through the compile commands JSON file. The commands below show how to produce it from *Makefile*

[4] Note the sources of the source codes used in the NERSC course are distributed as a ZIP file, which creates the folder *nersc_examples_2023/* once uncompressed.

files through the *bear* tool, and how to pass the JSON file to the *pwreport* tool [5] [6]. Note the screening shows the progress through all files successfully analyzed (flag *--show-progress*).

```
$ pwreport --screening --include-tags all mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90 --show-progress
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
[1/4] mtc_patch.f90
[2/4] mtc.F90
[3/4] mtc_openmp.F90
[4/4] mtc_main.F90

SCREENING REPORT

Target         Lines of code Optimizable lines Analysis time # checks Effort Cost    Profiling
-------------- ------------- ----------------- ------------- -------- ------ ------- ---------
mtc.F90        98            0                 31 ms         5        20 h   654€    n/a
mtc_openmp.F90 116           0                 23 ms         5        20 h   654€    n/a
mtc_patch.f90  54            0                 153 ms        0        0 h    0€      n/a
mtc_main.F90   209           0                 73 ms         0        0 h    0€      n/a
-------------- ------------- ----------------- ------------- -------- ------ ------- ---------
Total          477           0                 280 ms        10       40 h   1308€   n/a

CHECKS PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Target         Scalar Control Memory Vector Multi Offload
-------------- ------ ------- ------ ------ ----- -------
mtc.F90        0      0       5      0      0     0
mtc_openmp.F90 0      0       5      0      0     0
mtc_patch.f90  0      0       0      0      0     0
mtc_main.F90   0      0       0      0      0     0
-------------- ------ ------- ------ ------ ----- -------
Total          0      0       10     0      0     0

TOTAL NUMBER OF LOOPS FOR CODEE AUTO AND GUIDED MODES
                    ---------------- # checks ----------------
Codee mode # Loops Scalar Control Memory Vector Multi Offload
---------- ------- ------ ------- ------ ------ ----- -------
Auto       0       0      0       0      0      0     0
Guided     8       0      0       8      0      0     0

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)
Optimizable lines : relevant lines of code that Codee detects as optimizable
Analysis time : time required to analyze the target
# checks : total actionable items (opportunities, recommendations, defects and remarks) detected
```

[5] Codee best practice recommendation is to use the integration with build systems for large code bases. In particular, this is implemented through Codee support for JSON compile commands databases. Build systems like cmake, ninja and waf offer built-in support. For Makefiles the bear tool is available in Linux and can be used as follows (not in Perlmutter):

```
$ cd nersc_examples_2023/NUCCOR/
$ bear -- make
$ pwreport --screening --config compile_commands.json --include-tags all --show-progress
```

[6] Note that analysis of Fortran source codes with Codee follows similar behaviour as compilers. It requires that the source code of a Fortran module is analyzed before any usage of the module. This implies specifying the Fortran source codes in an order that guarantees that Codee first analyzes a Fortran module source file, before analyzing the Fortran source code that uses that module. It is work-in-progress to improve these usability issues. Below an example Codee invocation of gfortran and pwreport with a wrong ordering of the source codes in the command line:

```
$ gfortran -o out mtc.F90  mtc_main.F90  mtc_openmp.F90  mtc_patch.f90
   3 |    use :: mtc_patch_module, only : mtc_patch
     |        1
Fatal Error: Cannot open module file 'mtc_patch_module.mod' for reading at (1): No such file or directory

$ pwreport --screening mtc.F90 mtc_main.F90 mtc_openmp.F90 mtc_patch.f90 --show-failures
pwreport: error: failed to analyze 'mtc.F90'

error: Semantic errors in mtc.F90
mtc.F90:3:12: error: Cannot read module file for module 'mtc_patch_module': Source file 'mtc_patch_module.pw.mod' was not found
              use :: mtc_patch_module, only : mtc_patch
              ^^^^^^^^^^^^^^^^

1 file successfully analyzed and 3 failures in 63 ms
```

```
Effort : estimated number of hours it would take to carry out all checks (scalar, control, memory, vector, multi and
offload with 1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the checks, 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

Codee mode : Available Codee mode for the loop:
  - Auto: Codee optimizes the code automatically
  - Guided: Codee identifies the performance issue, and the programmer must apply the changes to the code

SUGGESTIONS
  Use --verbose to get more details, e.g:
        pwreport --verbose --screening --include-tags all mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90
--show-progress

  Use --checks to find out details about the detected checks:
        pwreport --checks --include-tags all mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90 --show-progress

  You can focus on a specific optimization type by filtering by its tag (scalar, control, memory, vector, multi,
offload), eg.:
        pwreport --checks --include-tags scalar mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90 --show-progress

4 files successfully analyzed and 0 failures in 288 ms
```

Step 5: Producing the Codee Checks Report

The Codee output suggests subsequent Codee commands useful to dig deeper into the performance issues discovered by the tool. The suggested next step is to produce the Codee Checks Report to list all the checks found in the code, by invoking the *pwreport --checks* command. The output format is similar to other static code analyzers in order to facilitate the user uptake and integration in the development workflow. The details about the checks reported in the screening phase are shown, including the position where the checks were detected in the source code and the title of each check.

```
$ pwreport --checks --include-tags all mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
CHECKS REPORT

mtc.F90:62:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc.F90:62:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc.F90:62:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc.F90:62:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc.F90:62:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc_openmp.F90:73:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc_openmp.F90:73:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc_openmp.F90:73:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc_openmp.F90:73:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'
mtc_openmp.F90:73:5 [PWR001]: function 'transpose_1_4_3_2_real64' uses global variable 'int64'

SUGGESTIONS

  Use --verbose to get more details, e.g:
        pwreport --verbose --checks --include-tags all mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90

  43 loops could not be analyzed, get more information with pwloops:
        pwloops --non-analyzable --include-tags all mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90

  More details on the defects, recommendations and more in the Knowledge Base:
        https://www.codee.com/knowledge/

4 files successfully analyzed and 0 failures in 291 ms
```

Step 6: Diagnose Fortran unsupported features in Codee

The Fortran support in Codee is evolving fast and it is relevant to identify unsupported Fortran constructs. The *pwreport* command with the *--non-analyzable* flag serves for this purpose. Note it reports the usage of modern Fortran polymorphic types, which are not supported in the current version of the LLVM Flang project that provides Codee with the Fortran front-end.

```
$ pwreport --non-analyzable mtc_patch.f90 mtc.F90 mtc_openmp.F90 mtc_main.F90
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
Count Type
----- ----

Count : Number of occurrences of the unsupported feature
Type : Type of unsupported feature

4 files successfully analyzed and 0 failures in 69 ms
```

The Codee support for Fortran leverages the LLVM Flang front-end. Thus, in order for Codee to provide more precise checks (e.g. GPU offload opportunities), the target Fortran functions must be available in source files completely parseable by LLVM Flang front-end. For illustrative purposes, three target Fortran functions were outlined in separate files. Find below the output of the Codee Screening Report for those source files.

```
$ pwreport --screening --include-tags all split/*
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
SCREENING REPORT

Target                                Lines of code Optimizable lines Analysis time # checks Effort Cost    Profiling
------------------------------------- ------------- ----------------- ------------- -------- ------ ------- ---------
split/mtc_transpose_1_4_3_2_real64.F90 15            0                 19 ms         6        24 h   785€    n/a
split/mtc_contract.F90                 29            0                 65 ms         1        4 h    130€    n/a
split/mtc_contract_simple.F90          25            0                 44 ms         1        4 h    130€    n/a
------------------------------------- ------------- ----------------- ------------- -------- ------ ------- ---------
Total                                 69            0                 128 ms        8        32 h   1047€   n/a

CHECKS PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Target                                Scalar Control Memory Vector Multi Offload
------------------------------------- ------ ------- ------ ------ ----- -------
split/mtc_transpose_1_4_3_2_real64.F90 0      0       6      0      0     0
split/mtc_contract.F90                 0      0       1      0      0     0
split/mtc_contract_simple.F90          0      0       1      0      0     0
------------------------------------- ------ ------- ------ ------ ----- -------
Total                                 0      0       8      0      0     0

TOTAL NUMBER OF LOOPS FOR CODEE AUTO AND GUIDED MODES
                    ---------------- # checks ----------------
Codee mode # Loops Scalar Control Memory Vector Multi Offload
---------- ------- ------ ------- ------ ------ ----- -------
Auto       0       0      0       0      0      0     0
Guided     4       0      0       4      0      0     0

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)
Optimizable lines : relevant lines of code that Codee detects as optimizable
Analysis time : time required to analyze the target
# checks : total actionable items (opportunities, recommendations, defects and remarks) detected
Effort : estimated number of hours it would take to carry out all checks (scalar, control, memory, vector, multi and
offload with 1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the checks, 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

Codee mode : Available Codee mode for the loop:
  - Auto: Codee optimizes the code automatically
  - Guided: Codee identifies the performance issue, and the programmer must apply the changes to the code

SUGGESTIONS
  Use --verbose to get more details, e.g:
       pwreport --verbose --screening --include-tags all split/mtc_contract.F90 split/mtc_contract_simple.F90
split/mtc_transpose_1_4_3_2_real64.F90

  Use --checks to find out details about the detected checks:
       pwreport --checks --include-tags all split/mtc_contract.F90 split/mtc_contract_simple.F90
split/mtc_transpose_1_4_3_2_real64.F90

  You can focus on a specific optimization type by filtering by its tag (scalar, control, memory, vector, multi,
offload), eg.:
       pwreport --checks --include-tags scalar split/mtc_contract.F90 split/mtc_contract_simple.F90
split/mtc_transpose_1_4_3_2_real64.F90

3 files successfully analyzed and 0 failures in 132 ms
```

Next, let's use Codee to get insights about the Fortran unsupported constructs present in the target files. In this case, the *pwreport --non-analyzable* commands identified function parameters declared as "deferred shape arrays". This is work in progress in Codee at the moment and support will be available in upcoming releases.

```
$ pwreport --non-analyzable split/
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
Count Type
----- -----------------------------------------
12    Unsupported usaged of deferred shape array

Count : Number of occurrences of the unsupported feature
Type : Type of unsupported feature

3 files successfully analyzed and 0 failures in 88 ms
```

Finally, in order to illustrate the results that future releases of Codee will provide for this code, let's modify the source code to avoid the usage of deferred shape arrays. Note that in this new version the construct "dimension" is used to provide the compiler and Codee with additional information about the shape of the arrays (e.g. size of each array dimension).

Note that when the source code of a function is refactored to avoid deferred shape arrays, required changes in the function prototype may require cascaded changes in the call site.

```
 1
 2       subroutine contract_simple(dst, src, op, O, P, Q, R, S, T)
 3           use, intrinsic :: iso_fortran_env, only : int64, real64
 4
 5           integer, intent(in) :: O, P, Q, R, S, T
 6
 7           real(real64), dimension(1:O,1:P,1:Q,1:R), intent(inout) :: dst
 8           real(real64), dimension(1:T,1:T,1:P,1:Q,1:R,1:S), intent(in) :: src
 9           real(real64), dimension(1:S,1:O,1:T,1:T), intent(in) :: op
10
11           integer :: i, j, m, a, b, e, f
12           real(real64) :: temp
13
14           do j = 1, size(dst, 4)
15               do i = 1, size(dst, 3)
16                   do b = 1, size(dst, 2)
17                       do a = 1, size(dst, 1)
18                           temp = 0.0d0
19                           do m = 1, size(op, 1)
20                               do f = 1, size(op, 3)
21                                   do e = 1, size(op, 4)
22                                       temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
23                                   end do
24                               end do
25                           end do
26                           dst(a, b, i, j) = 0.5d0*temp
27                       end do
28                   end do
29               end do
30           end do
31       end subroutine contract_simple
```

At this moment we use Codee to produce the Codee Screening Report for the new Fortran versions (without deferred shape arrays). Note that Codee reports more checks, including opportunities for multithreading and GPU offloading.

```
$ pwreport --screening --include-tags all adapted/mtc_contract_simple.F90
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
SCREENING REPORT

Target                       Lines of code Optimizable lines Analysis time # checks Effort Cost         Profiling
---------------------------- ------------- ----------------- ------------- -------- ------ ------- ---------
adapted/mtc_contract_simple.F90 26             26                85 ms              9         60 h   1963€   n/a
---------------------------- ------------- ----------------- ------------- -------- ------ ------- ---------
Total                        26                26                85 ms              9         60 h   1963€   n/a

CHECKS PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Target                          Scalar Control Memory Vector Multi Offload
------------------------------ ------ ------- ------ ------ ----- -------
adapted/mtc_contract_simple.F90 0     0        6         1         1         1
------------------------------ ------ ------- ------ ------ ----- -------
Total                          0        0        6         1         1         1

TOTAL NUMBER OF LOOPS FOR CODEE AUTO AND GUIDED MODES
                  ---------------- # checks ----------------
Codee mode # Loops Scalar Control Memory Vector Multi Offload
---------- ------- ------ ------- ------ ------ ----- -------
Auto       0       0      0       0      0      0     0
Guided     6       0      0       5      1      1     1

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)
Optimizable lines : relevant lines of code that Codee detects as optimizable
Analysis time : time required to analyze the target
# checks : total actionable items (opportunities, recommendations, defects and remarks) detected
Effort : estimated number of hours it would take to carry out all checks (scalar, control, memory, vector, multi and
offload with 1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the checks, 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

Codee mode : Available Codee mode for the loop:
  - Auto: Codee optimizes the code automatically
  - Guided: Codee identifies the performance issue, and the programmer must apply the changes to the code

SUGGESTIONS
  Use --verbose to get more details, e.g:
         pwreport --verbose --screening --include-tags all adapted/mtc_contract_simple.F90

  You can specify multiple inputs which will be displayed as multiple rows (ie. targets) in the table, eg:
         pwreport --screening some/other/dir --include-tags all adapted/mtc_contract_simple.F90

  Use --checks to find out details about the detected checks:
         pwreport --checks --include-tags all adapted/mtc_contract_simple.F90

  You can focus on a specific optimization type by filtering by its tag (scalar, control, memory, vector, multi,
offload), eg.:
         pwreport --checks --include-tags scalar adapted/mtc_contract_simple.F90

1 file successfully analyzed and 0 failures in 86 ms
```

Step 7: Producing the Codee Checks Report

The process to produce the Codee Checks Report is similar, just follow the suggested commands.

```
$ pwreport --checks --verbose --include-tags all adapted/mtc_contract_simple.F90
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

pwreport: warning: Fortran support is experimental
CHECKS REPORT

adapted/mtc_contract_simple.F90:2:5 [PWR007]: procedure 'contract_simple' does not have or inherit IMPLICIT NONE policy
  Suggestion: add IMPLICIT NONE in the specification part of the procedure
  Documentation: https://www.codee.com/knowledge/pwr007

adapted/mtc_contract_simple.F90:14:9 [PWR035]: avoid non-consecutive array access for variable 'op' to improve
performance
  Non-consecutive array access:
    22:                                   temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to
avoid non-sequential access to variable 'op'.
  Documentation: https://www.codee.com/knowledge/pwr035

adapted/mtc_contract_simple.F90:14:9 [PWR050]: consider applying multithreading parallelism to forall loop
  Suggestion: use pwdirectives to automatically optimize the code
  Documentation: https://www.codee.com/knowledge/pwr050
  AutoFix (choose one option):
      * Using OpenMP 'for' (recommended):
        pwdirectives --multi omp-for --in-place adapted/mtc_contract_simple.F90:14:9
      * Using OpenMP 'taskwait':
        pwdirectives --multi omp-taskwait --in-place adapted/mtc_contract_simple.F90:14:9
      * Using OpenMP 'taskloop':
        pwdirectives --multi omp-taskloop --in-place adapted/mtc_contract_simple.F90:14:9

adapted/mtc_contract_simple.F90:14:9 [PWR055]: consider applying offloading parallelism to forall loop
  Suggestion: use pwdirectives to automatically optimize the code
  Documentation: https://www.codee.com/knowledge/pwr055
  AutoFix:
    * Using OpenMP (recommended):
        pwdirectives --offload omp-teams --in-place adapted/mtc_contract_simple.F90:14:9
    * Using OpenAcc:
        pwdirectives --offload acc --in-place adapted/mtc_contract_simple.F90:14:9

adapted/mtc_contract_simple.F90:15:13 [PWR035]: avoid non-consecutive array access for variable 'op' to improve
performance
  Non-consecutive array access:
    22:                                   temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to
avoid non-sequential access to variable 'op'.
  Documentation: https://www.codee.com/knowledge/pwr035

adapted/mtc_contract_simple.F90:16:17 [PWR035]: avoid non-consecutive array access for variable 'op' to improve
performance
  Non-consecutive array access:
    22:                                   temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to
avoid non-sequential access to variable 'op'.
  Documentation: https://www.codee.com/knowledge/pwr035

adapted/mtc_contract_simple.F90:17:21 [PWR035]: avoid non-consecutive array access for variable 'op' to improve
performance
  Non-consecutive array access:
    22:                                   temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to
avoid non-sequential access to variable 'op'.
  Documentation: https://www.codee.com/knowledge/pwr035

adapted/mtc_contract_simple.F90:19:25 [PWR035]: avoid non-consecutive array access for variable 'op' to improve
performance
  Non-consecutive array access:
    22:                                   temp = temp + op(m, a, e, f)*src(e, f, b, i, j, m)
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to
avoid non-sequential access to variable 'op'.
  Documentation: https://www.codee.com/knowledge/pwr035
```

```
adapted/mtc_contract_simple.F90:21:33 [RMK010]: the vectorization cost model states the loop is not a SIMD opportunity
due to strided memory accesses in the loop body
  Documentation: https://www.codee.com/knowledge/rmk010

SUGGESTIONS

  More details on the defects, recommendations and more in the Knowledge Base:
        https://www.codee.com/knowledge/

1 file successfully analyzed and 0 failures in 92 ms
```

Step 8: Generating code optimized for GPU (offloading)

The process to produce GPU-enabled Fortran source code with OpenMP and OpenACC directives is similar, just follow the suggested *pwdirectives* commands (see step-by-step guide MATMUL C/Fortran for the details).

Additional remarks about Codee Fortran support

Active development in Codee is focused on rapidly improving the Fortran supports, including better support for modern Fortran language constructs as well as integration with Fortran build systems to facilitate the screening of large Fortran codes. In addition, sampling real-world Fortran applications will be used to drive the roadmap of the Codee Fortran support so that it serves the Fortran developers community better.