Codee Training Series

April 26-27, 2022

NERSC

codee

Shift Left Performance

Automated Code inspection for Performance

Parallelizing MATrix MULtiplication on the GPU with OpenMP/OpenACC

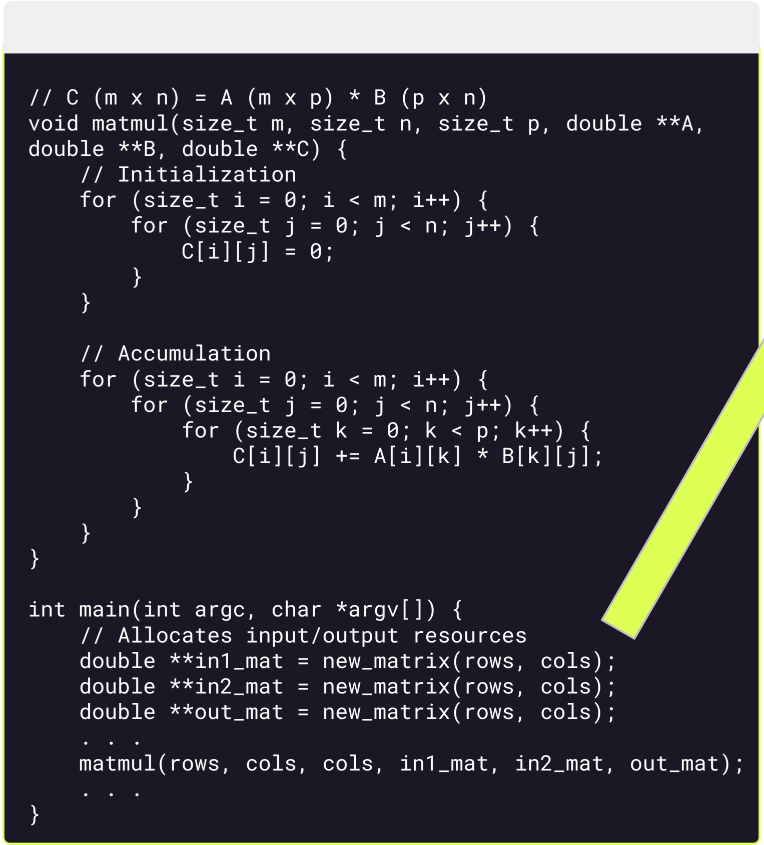
Goals:

- Produce OpenACC version for GPU
- Produce OpenMP version for GPU
- Build & run an OpenMP code on the GPU (for problem size N=1500)
- Build & run an OpenACC code on the GPU (for problem size N=1500)

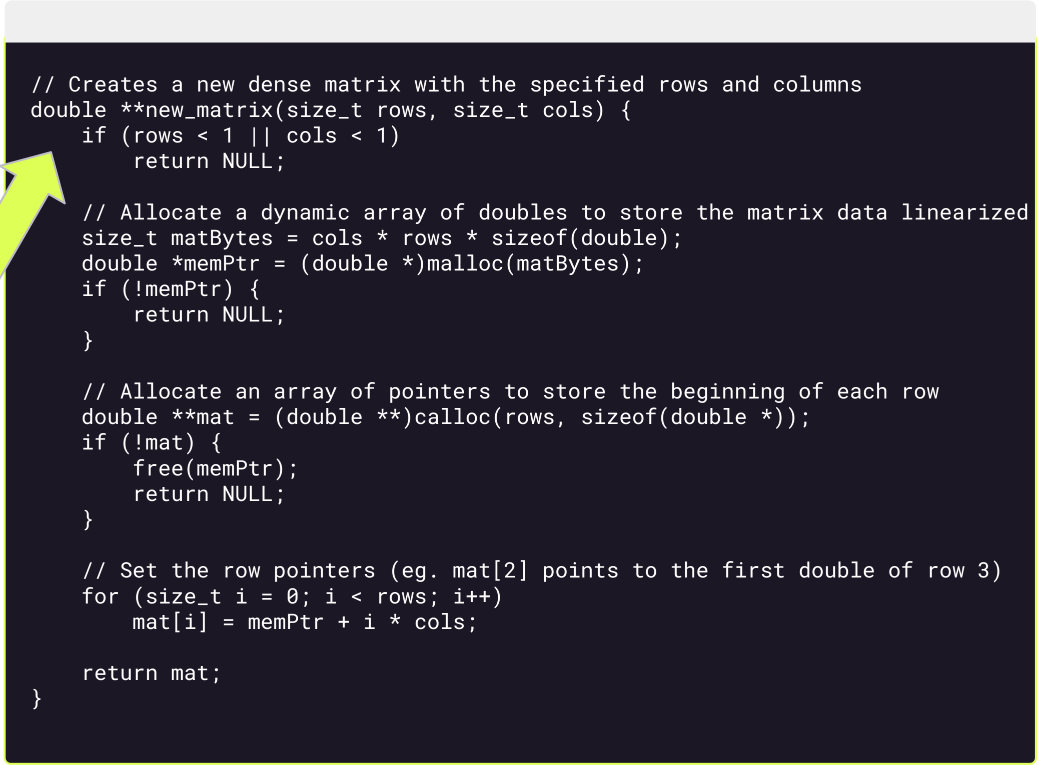
The GPU programming challenges: Example code MATMUL

		Challenges of GPU acceleration addressed in introductory course			Other GPU programming challenges to be addressed in next advanced course			
		Find opportunities for offloading	Optimize memory layout for data transfers	Identify defects in data transfers	Exploit massive parallelism through loop nest collapsing	Minimize data transfers across consecutive loop nests	Minimize data transfers through convergence loops	Identify auxiliary functions to be offloaded
Example codes used in this introductor y course	**PI**	**x**	**-**	**-**	**-**	**-**	**-**	**-**
	MATMUL	**x**	**x**	**x**	**x**	**x**	**-**	**-**
	LULESHmk	**x**	**x**	**x**	**x**	**x**	**x**	**x**
	HEAT	**x**	**-**	**-**	**-**	**x**	**x**	**-**
	Your code!	Probably all of these challenges apply, and even more!						

The source code of MATMUL using double**



```c
// C (m x n) = A (m x p) * B (p x n)
void matmul(size_t m, size_t n, size_t p, double **A,
double **B, double **C) {
    // Initialization
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            C[i][j] = 0;
        }
    }

    // Accumulation
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            for (size_t k = 0; k < p; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

int main(int argc, char *argv[]) {
    // Allocates input/output resources
    double **in1_mat = new_matrix(rows, cols);
    double **in2_mat = new_matrix(rows, cols);
    double **out_mat = new_matrix(rows, cols);
    . . .
    matmul(rows, cols, cols, in1_mat, in2_mat, out_mat);
    . . .
}
```

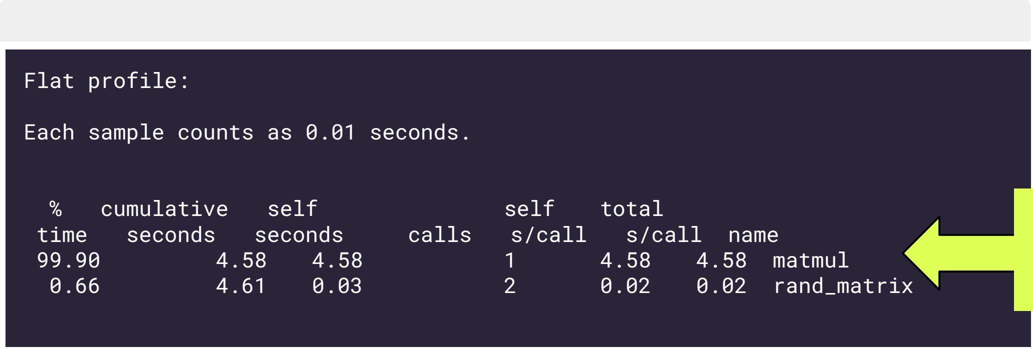


```c
// Creates a new dense matrix with the specified rows and columns
double **new_matrix(size_t rows, size_t cols) {
    if (rows < 1 || cols < 1)
        return NULL;

    // Allocate a dynamic array of doubles to store the matrix data linearized
    size_t matBytes = cols * rows * sizeof(double);
    double *memPtr = (double *)malloc(matBytes);
    if (!memPtr) {
        return NULL;
    }

    // Allocate an array of pointers to store the beginning of each row
    double **mat = (double **)calloc(rows, sizeof(double *));
    if (!mat) {
        free(memPtr);
        return NULL;
    }

    // Set the row pointers (eg. mat[2] points to the first double of row 3)
    for (size_t i = 0; i < rows; i++)
        mat[i] = memPtr + i * cols;

    return mat;
}
```

Profiling and validation of MATMUL

```
$ gcc -pg -I include matrix.c clock.c main.c -o matmul
$ ./matmul 1000
    - Input parameters
    n   = 1000
    - Executing test...
    time (s)= 4.589052
    size   = 1000
    chksum   = 20269164323
$ gprof ./matmul
```

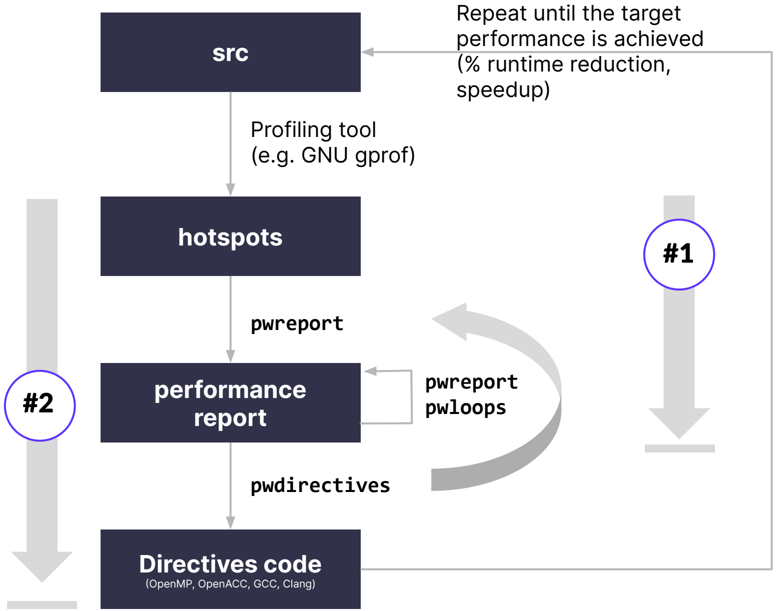
Note: we use GCC for a quicker profiling using the GPROF profiling tool, which reports the functions that consumes most of the runtime.



```
Flat profile:

Each sample counts as 0.01 seconds.


  %   cumulative   self              self    total
 time   seconds   seconds    calls   s/call   s/call  name
 99.90      4.58     4.58        1     4.58     4.58  matmul
  0.66      4.61     0.03        2     0.02     0.02  rand_matrix
```

The hotspot function matmul() takes almost 100% of the runtime

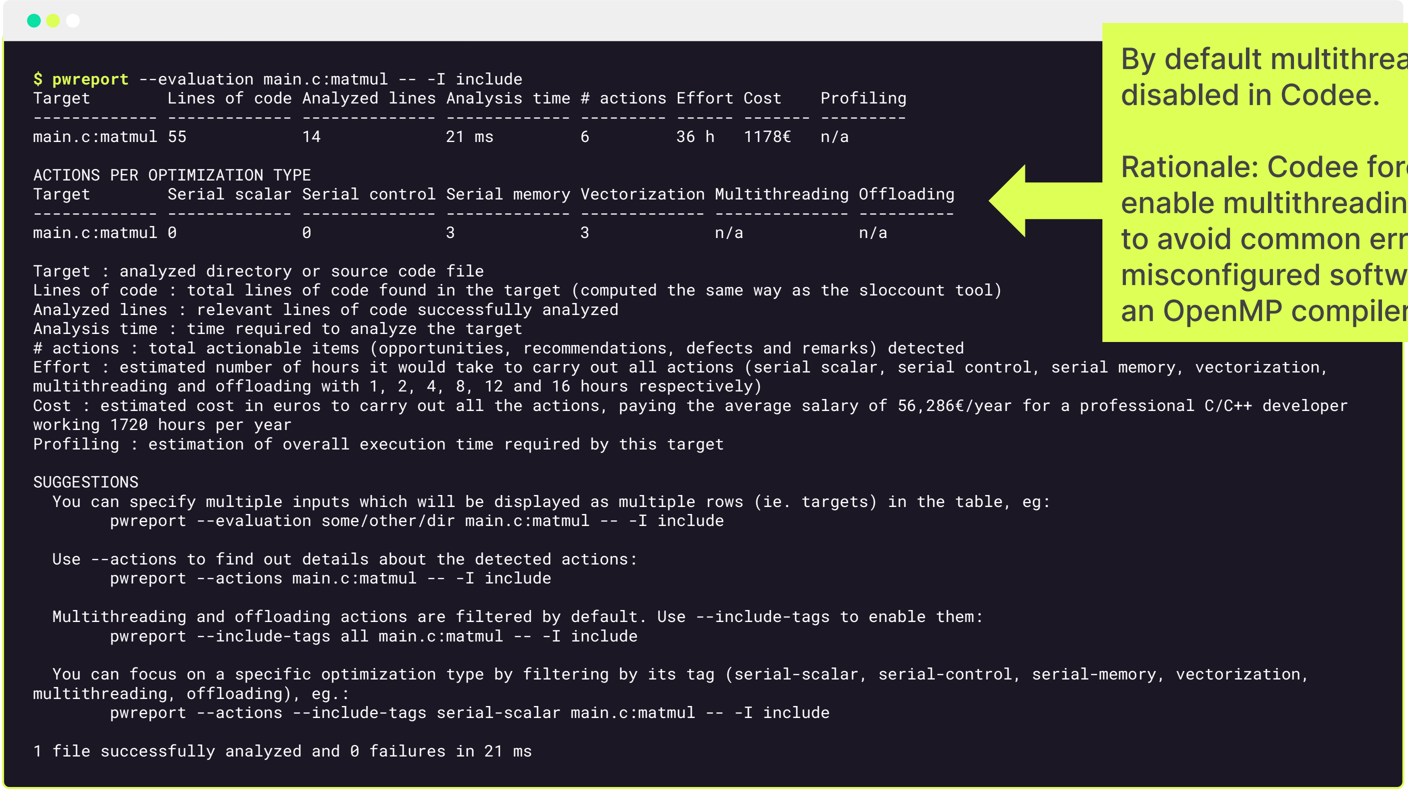
Inspecting the code and optimizing its performance with Codee


src
Repeat until the target performance is achieved (% runtime reduction, speedup)
Profiling tool (e.g. GNU gprof)
hotspots
#1
pwreport
performance report
pwreport
pwloops
#2
pwdirectives
Directives code
(OpenMP, OpenACC, GCC, Clang)

#1 **Get the performance optimization report for the whole code base**

#2 **Create performance-optimized code for the hotspot automatically**

1: Produce the entry-level report for default #actions (`pwreport --evaluation`)

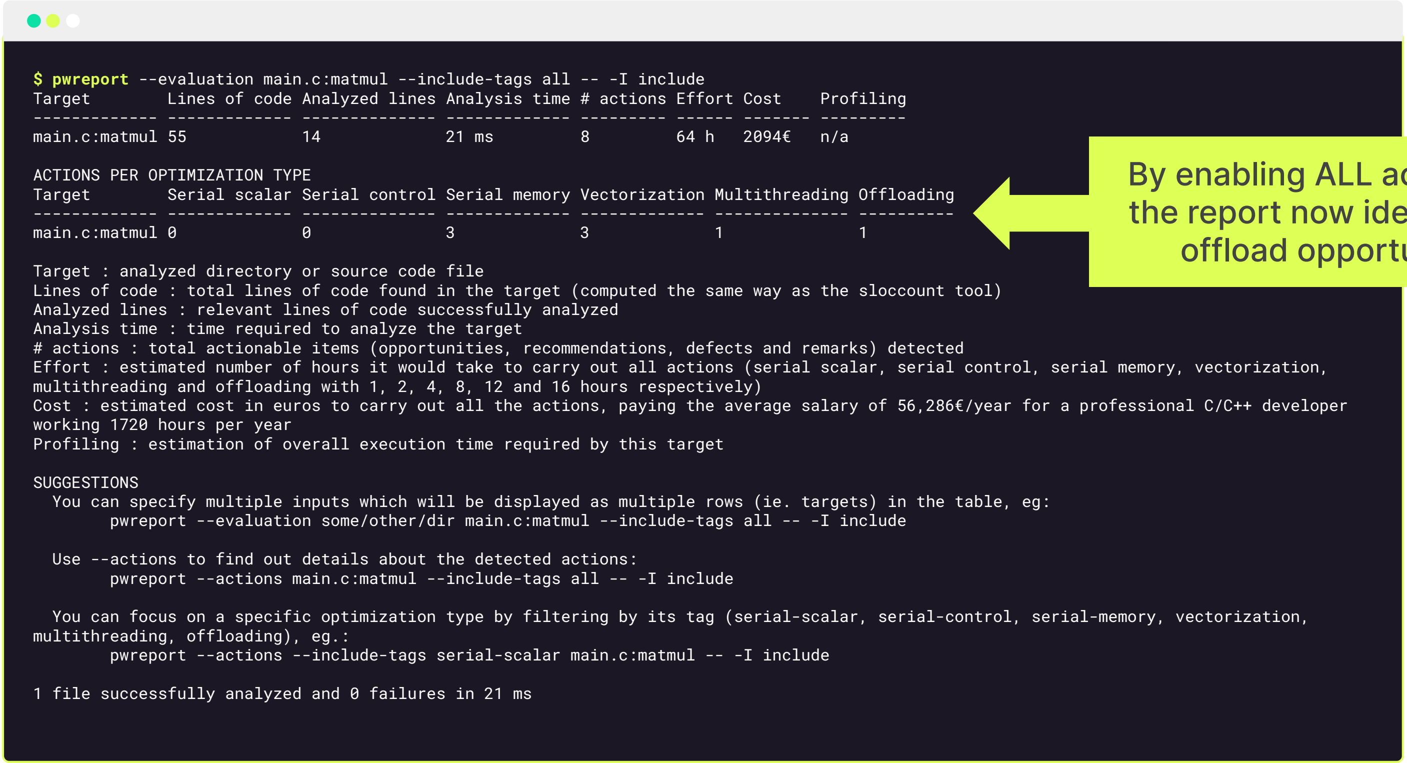


```
$ pwreport --evaluation main.c:matmul -- -I include
Target        Lines of code Analyzed lines Analysis time # actions Effort Cost    Profiling
------------- ------------- -------------- ------------- --------- ------ ------- ---------
main.c:matmul 55            14             21 ms         6         36 h   1178€   n/a

ACTIONS PER OPTIMIZATION TYPE
Target        Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
------------- ------------- -------------- ------------- ------------- -------------- ----------
main.c:matmul 0             0              3             3             n/a            n/a

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 main.c:matmul -- -I include

  Use --actions to find out details about the detected actions:
        pwreport --actions main.c:matmul -- -I include

  Multithreading and offloading actions are filtered by default. Use --include-tags to enable them:
        pwreport --include-tags all main.c:matmul -- -I include

  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 main.c:matmul -- -I include

1 file successfully analyzed and 0 failures in 21 ms
```

By default multithreading and offloading are disabled in Codee.

Rationale: Codee forces the user to explicitly enable multithreading and offloading capabilities to avoid common errors resulting from a misconfigured software environment (eg. lack of an OpenMP compiler with offload)

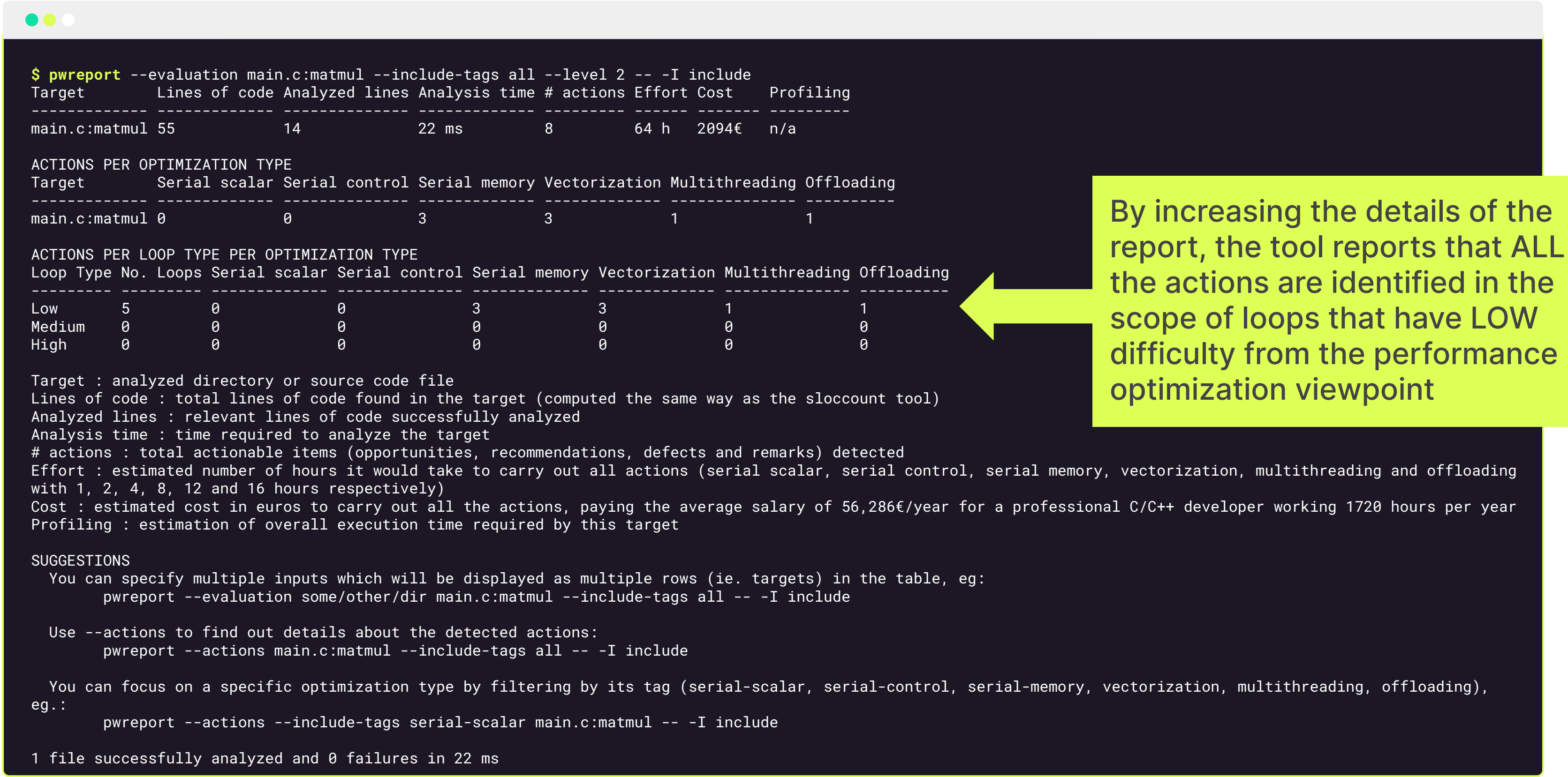
2: Produce the entry-level report for ALL #actions (`pwreport --evaluation --include-tags all`)



```
$ pwreport --evaluation main.c:matmul --include-tags all -- -I include
Target        Lines of code Analyzed lines Analysis time # actions Effort Cost    Profiling
------------- ------------- -------------- ------------- --------- ------ ------- ---------
main.c:matmul 55            14             21 ms         8         64 h   2094€   n/a

ACTIONS PER OPTIMIZATION TYPE
Target        Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
------------- ------------- -------------- ------------- ------------- -------------- ----------
main.c:matmul 0             0              3             3             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)
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 main.c:matmul --include-tags all -- -I include

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

  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 main.c:matmul -- -I include

1 file successfully analyzed and 0 failures in 21 ms
```

By enabling ALL actions in the report now identifies 1 offload opportunity

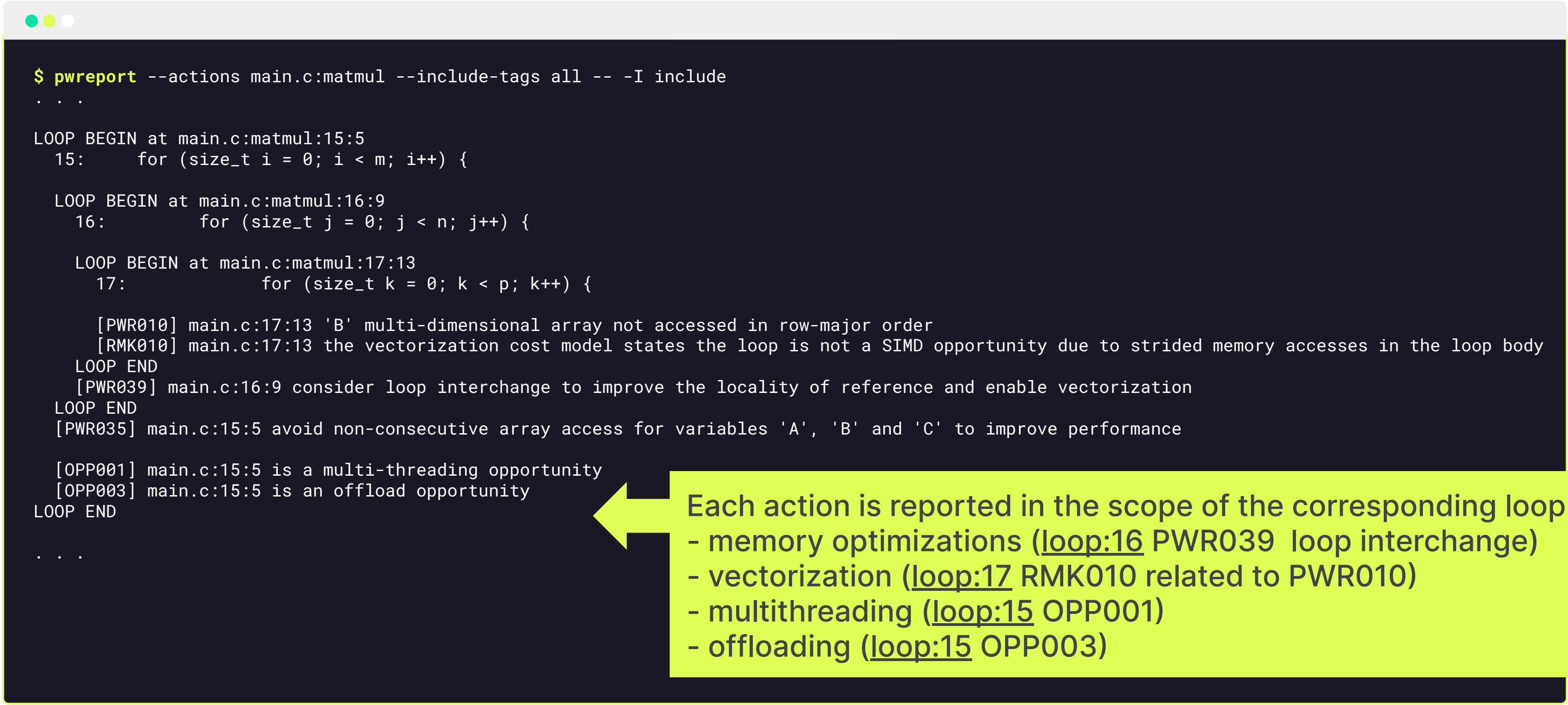
3: Produce the report of ALL #actions per type of loops (pwreport --evaluation --include-tags all --level 2)



```
$ pwreport --evaluation main.c:matmul --include-tags all --level 2 -- -I include
Target        Lines of code Analyzed lines Analysis time # actions Effort Cost    Profiling
------------- ------------- -------------- ------------- --------- ------ ------- ---------
main.c:matmul 55            14             22 ms         8         64 h   2094€   n/a

ACTIONS PER OPTIMIZATION TYPE
Target        Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
------------- ------------- -------------- ------------- ------------- -------------- ----------
main.c:matmul 0             0              3             3             1              1

ACTIONS PER LOOP TYPE PER OPTIMIZATION TYPE
Loop Type No. Loops Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
--------- --------- ------------- -------------- ------------- ------------- -------------- ----------
Low       5         0             0              3             3             1              1
Medium    0         0             0              0             0             0              0
High      0         0             0              0             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)
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 main.c:matmul --include-tags all -- -I include

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

  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 main.c:matmul -- -I include

1 file successfully analyzed and 0 failures in 22 ms
```

By increasing the details of the report, the tool reports that ALL the actions are identified in the scope of loops that have LOW difficulty from the performance optimization viewpoint

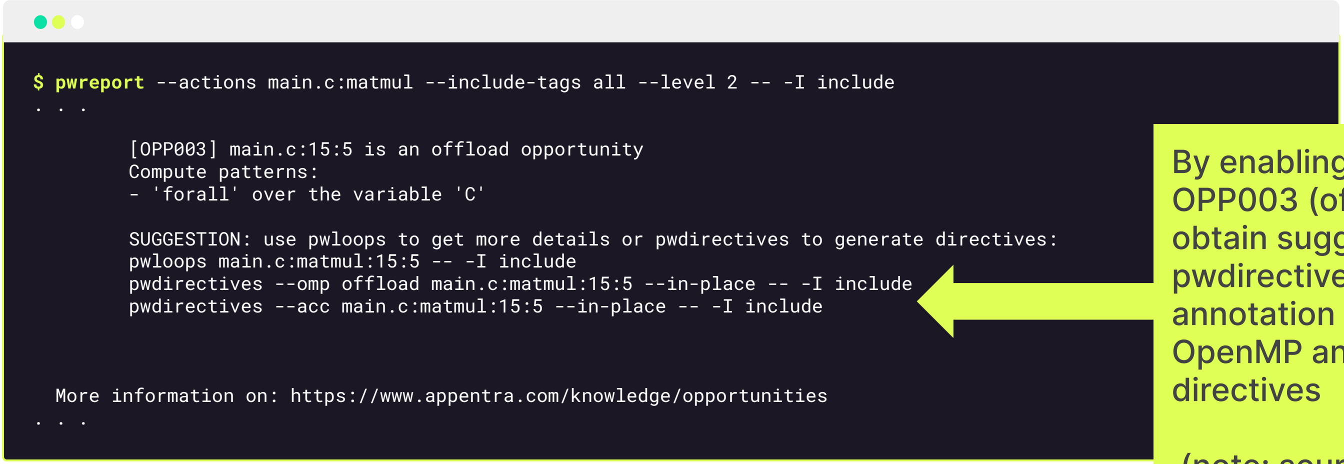
4: Produce the Codee Actions Report for the target function (`pwreport --actions`)



```
$ pwreport --actions main.c:matmul --include-tags all -- -I include
. . .

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

  LOOP BEGIN at main.c:matmul:16:9
    16:         for (size_t j = 0; j < n; j++) {

    LOOP BEGIN at main.c:matmul:17:13
      17:             for (size_t k = 0; k < p; k++) {

      [PWR010] main.c:17:13 'B' multi-dimensional array not accessed in row-major order
      [RMK010] main.c:17:13 the vectorization cost model states the loop is not a SIMD opportunity due to strided memory accesses in the loop body
    LOOP END
    [PWR039] main.c:16:9 consider loop interchange to improve the locality of reference and enable vectorization
  LOOP END
  [PWR035] main.c:15:5 avoid non-consecutive array access for variables 'A', 'B' and 'C' to improve performance

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

. . .
```

Each action is reported in the scope of the corresponding loop:
- memory optimizations (loop:16 PWR039 loop interchange)
- vectorization (loop:17 RMK010 related to PWR010)
- multithreading (loop:15 OPP001)
- offloading (loop:15 OPP003)

5: Produce the detailed actions for the target function (`pwreport --actions --level 2`)

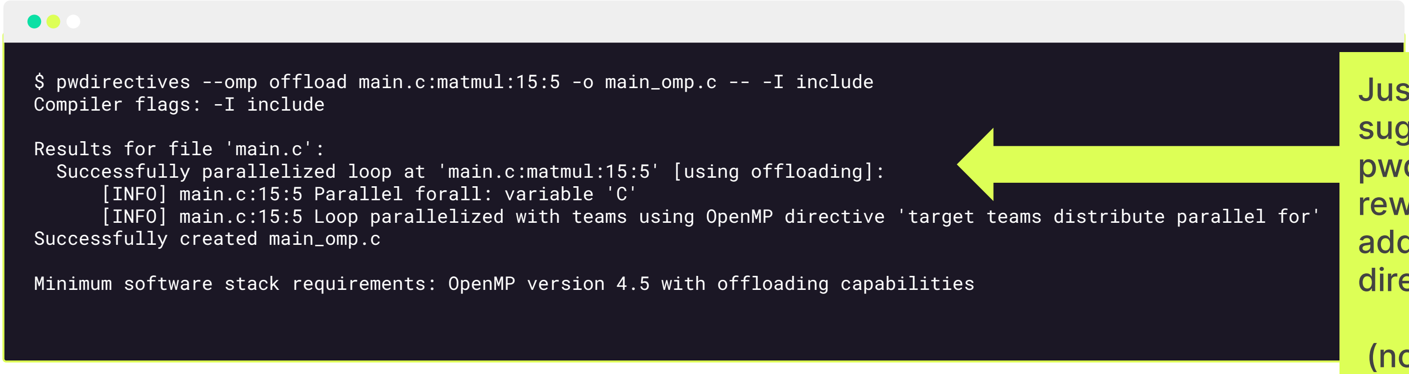


```
$ pwreport --actions main.c:matmul --include-tags all --level 2 -- -I include
. . .

        [OPP003] main.c:15:5 is an offload opportunity
        Compute patterns:
        - 'forall' over the variable 'C'

        SUGGESTION: use pwloops to get more details or pwdirectives to generate directives:
        pwloops main.c:matmul:15:5 -- -I include
        pwdirectives --omp offload main.c:matmul:15:5 --in-place -- -I include
        pwdirectives --acc main.c:matmul:15:5 --in-place -- -I include



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

By enabling the detailed report for OPP003 (offload opportunity) you obtain suggestions to invoke pwdirectives for automatic annotation of the source code with OpenMP and OpenACC offload directives

(note: source code edited "in-place" by default")

6a: Annotate the code for GPU + OpenMP (`pwdirectives --omp offload`)

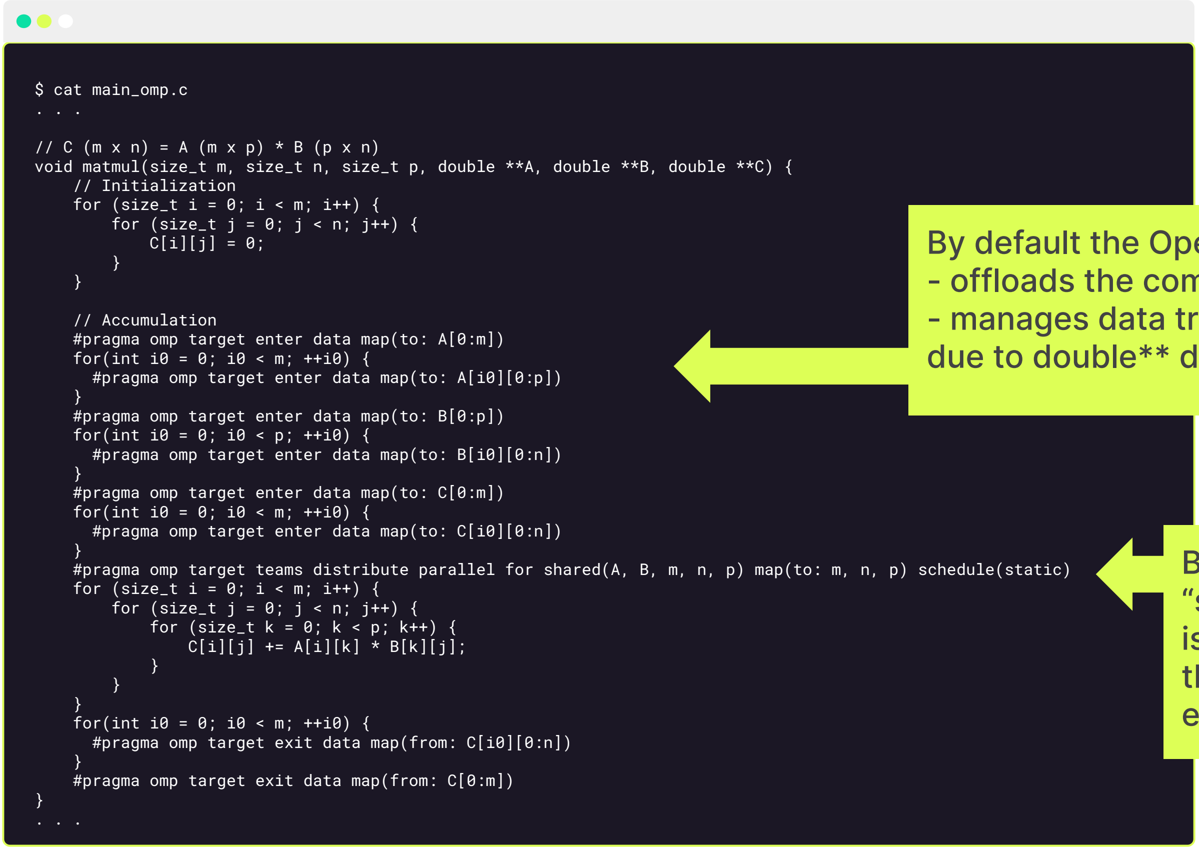


```
$ pwdirectives --omp offload main.c:matmul:15:5 -o main_omp.c -- -I include
Compiler flags: -I include

Results for file 'main.c':
  Successfully parallelized loop at 'main.c:matmul:15:5' [using offloading]:
      [INFO] main.c:15:5 Parallel forall: variable 'C'
      [INFO] main.c:15:5 Loop parallelized with teams using OpenMP directive 'target teams distribute parallel for'
Successfully created main_omp.c

Minimum software stack requirements: OpenMP version 4.5 with offloading capabilities
```

Just copy & paste the suggested invocation of pwdirectives, which will rewrite the code for you adding OpenMP directives

(note: source code edited "in-place" by default" and in this example we are using “-o” to write a separate source code file)

Code rewritten by pwdirectives for GPU + OpenMP

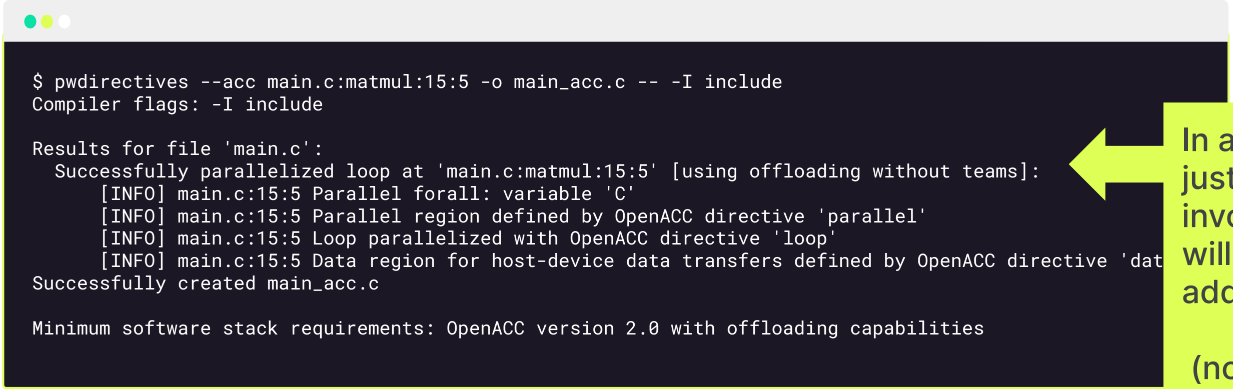


```
$ cat main_omp.c
. . .

// C (m x n) = A (m x p) * B (p x n)
void matmul(size_t m, size_t n, size_t p, double **A, double **B, double **C) {
    // Initialization
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            C[i][j] = 0;
        }
    }

    // Accumulation
    #pragma omp target enter data map(to: A[0:m])
    for(int i0 = 0; i0 < m; ++i0) {
      #pragma omp target enter data map(to: A[i0][0:p])
    }
    #pragma omp target enter data map(to: B[0:p])
    for(int i0 = 0; i0 < p; ++i0) {
      #pragma omp target enter data map(to: B[i0][0:n])
    }
    #pragma omp target enter data map(to: C[0:m])
    for(int i0 = 0; i0 < m; ++i0) {
      #pragma omp target enter data map(to: C[i0][0:n])
    }
    #pragma omp target teams distribute parallel for shared(A, B, m, n, p) map(to: m, n, p) schedule(static)
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            for (size_t k = 0; k < p; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    for(int i0 = 0; i0 < m; ++i0) {
      #pragma omp target exit data map(from: C[i0][0:n])
    }
    #pragma omp target exit data map(from: C[0:m])
}
. . .
```

By default the OpenMP generated code:
- offloads the computation with "target teams"
- manages data transfers with enter/exit data due to double** data types

By default the OpenMP “schedule(static)” is used as it is the schedule supported by the Nvidia programming environment

6b: Annotate the code for GPU + OpenACC (pwdirectives --acc)

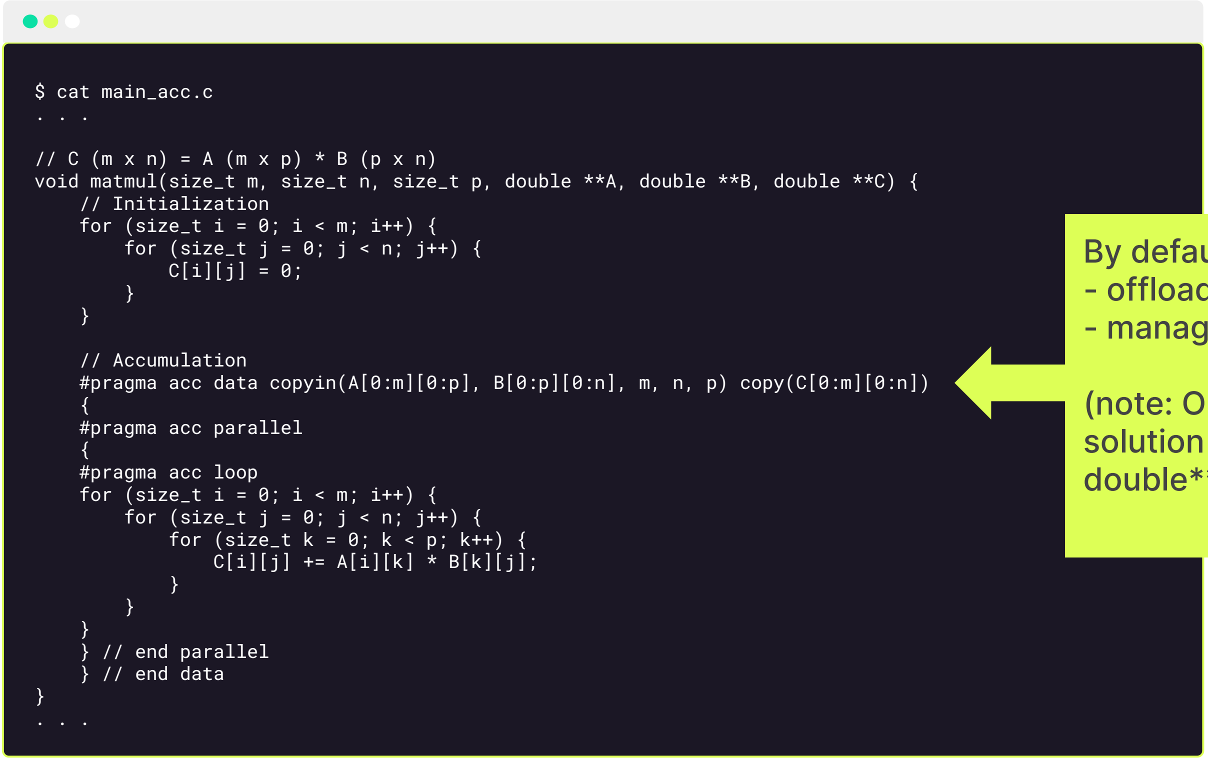


```
$ pwdirectives --acc main.c:matmul:15:5 -o main_acc.c -- -I include
Compiler flags: -I include

Results for file 'main.c':
  Successfully parallelized loop at 'main.c:matmul:15:5' [using offloading without teams]:
      [INFO] main.c:15:5 Parallel forall: variable 'C'
      [INFO] main.c:15:5 Parallel region defined by OpenACC directive 'parallel'
      [INFO] main.c:15:5 Loop parallelized with OpenACC directive 'loop'
      [INFO] main.c:15:5 Data region for host-device data transfers defined by OpenACC directive 'dat
Successfully created main_acc.c

Minimum software stack requirements: OpenACC version 2.0 with offloading capabilities
```

In a similar manner, for OpenACC just copy & paste the suggested invocation of pwdirectives, which will rewrite the code for you adding OpenACC directives

(note: source code edited "in-place" by default" and in this example we are using “-o” to write a separate source code file)

Code rewritten by pwdirectives for GPU + OpenACC

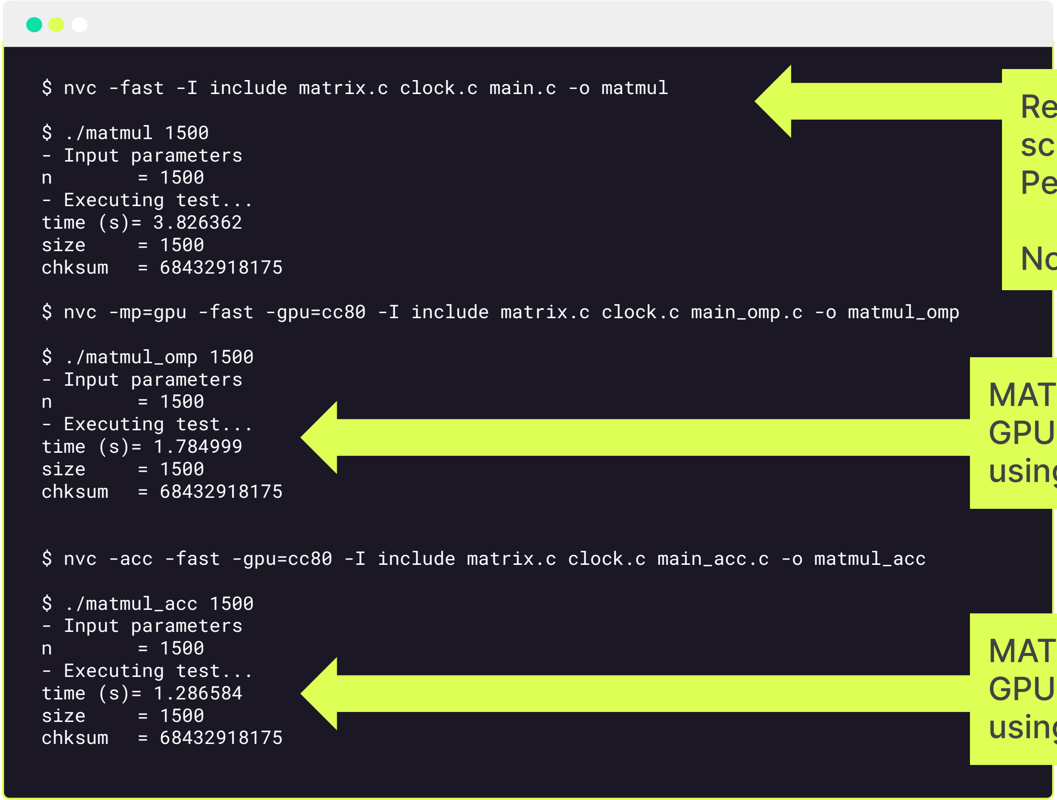


```
$ cat main_acc.c
. . .

// C (m x n) = A (m x p) * B (p x n)
void matmul(size_t m, size_t n, size_t p, double **A, double **B, double **C) {
    // Initialization
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            C[i][j] = 0;
        }
    }

    // Accumulation
    #pragma acc data copyin(A[0:m][0:p], B[0:p][0:n], m, n, p) copy(C[0:m][0:n])
    {
    #pragma acc parallel
    {
    #pragma acc loop
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            for (size_t k = 0; k < p; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    } // end parallel
    } // end data
}
. . .
```

By default the OpenACC generated code:
- offloads the computation with "parallel"
- manages data transfers with “data copy”

(note: OpenACC provides a more elegant solution to manage data transfers for double** data types)

7: Benchmarking on Perlmutter @NERSC (using Nvidia toolchain)



```
$ nvc -fast -I include matrix.c clock.c main.c -o matmul

$ ./matmul 1500
- Input parameters
n        = 1500
- Executing test...
time (s)= 3.826362
size     = 1500
chksum   = 68432918175

$ nvc -mp=gpu -fast -gpu=cc80 -I include matrix.c clock.c main_omp.c -o matmul_omp

$ ./matmul_omp 1500
- Input parameters
n        = 1500
- Executing test...
time (s)= 1.784999
size     = 1500
chksum   = 68432918175


$ nvc -acc -fast -gpu=cc80 -I include matrix.c clock.c main_acc.c -o matmul_acc

$ ./matmul_acc 1500
- Input parameters
n        = 1500
- Executing test...
time (s)= 1.286584
size     = 1500
chksum   = 68432918175
```

Remember using the launch, build and run scripts to conduct the experiments on Pelmutter @NERSC.

Note: See example scripts provided.

MATMUL code runs correctly on the GPU @perlmutter and 2.14x faster using OpenMP offload

MATMUL code runs correctly on the GPU @perlmutter and 2.97x faster using OpenACC offload

Final remarks about using Codee at NERSC

- First, remember to load the Codee module

```
$ module load codee
```

- The flag --help lists all the options available in the Codee command-line tools

```
$ pwreport --help
$ pwloops --help
$ pwdirectives --help
```

- You can run Codee command-line tools on the login nodes (no need to run them on the compute nodes)
- Build and run the example codes on the compute nodes using the batch scripts
  - Scripts tuned to use the appropriate reservations: *codee_day1*, *codee_day2*
- Remember to check the open catalog of rules for performance optimization:

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

codee

www.codee.com

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/