Codee Training Series

April 26-27, 2022

NERSC

codee

Shift Left Performance

Automated Code inspection for Performance

Identifying defects in MATrix MULtiplication on the GPU with OpenMP/OpenACC

Goals:

- Produce an OpenMP version for GPU using the “map” clause (do not use “enter/exit data”)
- Identify the defect PWD006 in the OpenMP version for GPU using “map”
- Build & run an OpenMP code on the GPU (for problem size N=1500)

The GPU programming challenges: Example code PI

		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;
}
```

The source code of MATMUL with OpenMP (defect PWD006 - Deep Copy -)

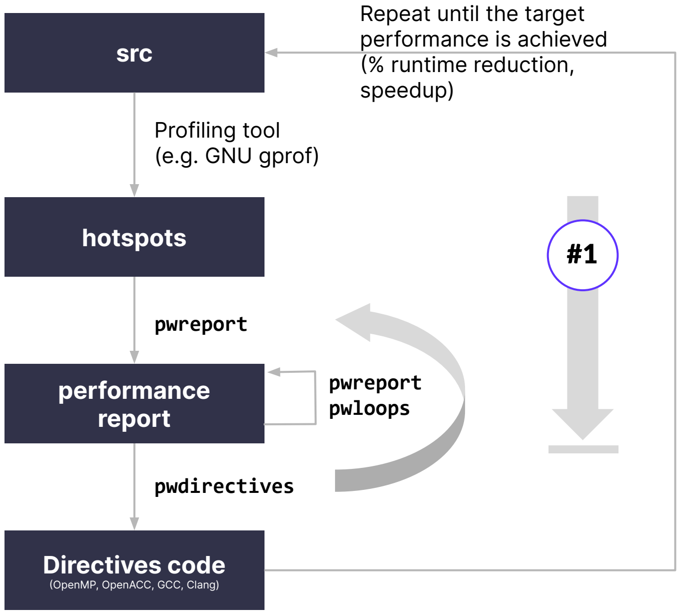
```
// C (m x n) = A (m x p) * B (p x n)
void matmul(size_t m, size_t n, size_t p, double **A, double **B, double **C) {
    // Initialization
    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            C[i][j] = 0;
        }
    }

    // Accumulation
    #pragma omp target teams distribute parallel for map(to: A, B, C, m, n, p) map(from: C) 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];
            }
        }
    }
}
```

Important note: This is the only line of the source code that was modified, by adding an OpenMP offload pragma.

Note there are hidden errors in this OpenMP offload pragma, more specifically in the “map” clause

Inspecting the code and optimizing its performance with Codee



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

Opportunities `(OPP)`
`Sequential, vectorization, multi-threading and GPU offloading`

Recommendations `(PWR)`
`Boost performance and ensure best practices`

Defects `(PWD)`
`Find and fix bugs in parallel code and correctness verification`

Remarks `(RMK)`
`Proficient usage of tools`

From all the actions in the performance optimization report, let's focus on the so-called "Defects"

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

```
$ pwreport --evaluation --level 2 main_pwd006.c:matmul --include-tags all -- -I include
Compiler flags: -I include

Target               Lines of code Analyzed lines Analysis time # actions Effort Cost    Profiling
-------------------- ------------- -------------- ------------- --------- ------ ------- ---------
main_pwd006.c:matmul 56            15             22 ms         8         68 h   2225€   n/a

ACTIONS PER OPTIMIZATION TYPE
Target               Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
-------------------- ------------- -------------- ------------- ------------- -------------- ----------
main_pwd006.c:matmul 0             0              3             3             0              2

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             0              2
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_pwd006.c:matmul --include-tags all -- -I include

  Use --actions to find out details about the detected actions:
        pwreport --actions main_pwd006.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_pwd006.c:matmul -- -I include

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

The report contains 2 actions related to offloading

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

```
$ pwreport --actions --level 2 main_pwd006.c:matmul --include-tags all -- -I include
. . .
LOOP BEGIN at main_pwd006.c:matmul:30:5
          #pragma omp target teams distribute parallel for map(to: A, B, C, m, n, p) map(from: C) schedule(static)
  30:     for (size_t i = 0; i < m; i++) {
  31:         for (size_t j = 0; j < n; j++) {
  32:             for (size_t k = 0; k < p; k++) {
  33:                 C[i][j] += A[i][k] * B[k][j];
  34:             }
  35:         }
  36:     }

. . .

  [PWD006] main_pwd006.c:29:5 missing deep copies of non-contiguous arrays 'A', 'B' and 'C' in data transfer to accelerator device
    29:     #pragma omp target teams distribute parallel for map(to: A, B, C, m, n, p) map(from: C) schedule(static)

    SUGGESTION: use OpenMP 4.5 enter/exit data execution statements to ensure that all the memory segments are copied to the memory
of the accelerator device

    More information on: https://www.appentra.com/knowledge/pwd006

. . .
```

One of the actions related to offload is the defect PWD006, triggered due to the improper usage of the “map” clause for the double** data type

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

```
$ nvc -mp=gpu -fast -gpu=cc80 -I include matrix.c clock.c main_pwd006.c -o matmul_pwd006

$ ./matmul_pwd006 1500
- Input parameters
n        = 1500
- Executing test...
Fatal error: expression 'HX_CU_CALL_CHECK(p_cuStreamSynchronize(stream[dev]))' (value 1) is not equal to expression 'HX_SUCCESS' (value 0)
Aborted
```

And the execution of the OpenMP-enabled code reported to suffer from defect PWD006 actually fails codee

www.codee.com

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/