Codee Training Series
April 26-27, 2022

NERSC

codee

Shift Left Performance
Automated Code inspection for Performance

©Appentra Solutions S.L.
April 2022
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

<table>
<thead>
<tr>
<th>Example codes used in this introductory course</th>
<th>Find opportunities for offloading</th>
<th>Optimize memory layout for data transfers</th>
<th>Identify defects in data transfers</th>
<th>Exploit massive parallelism through loop nest collapsing</th>
<th>Minimize data transfers across consecutive loop nests</th>
<th>Minimize data transfers through convergence loops</th>
<th>Identify auxiliary functions to be offloaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MATMUL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LULESHmk</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HEAT</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Your code!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Probably all of these challenges apply, and even more!
The source code of MATMUL using double**

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

```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 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

Repeat until the target performance is achieved (% runtime reduction, speedup)

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

<table>
<thead>
<tr>
<th>Target</th>
<th>Lines of code</th>
<th>Analyzed lines</th>
<th>Analysis time</th>
<th># actions</th>
<th>Effort Cost</th>
<th>Profiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>main_pwd006.c:matmul</td>
<td>56</td>
<td>15</td>
<td>22 ms</td>
<td>8</td>
<td>68 h</td>
<td>2225€</td>
</tr>
</tbody>
</table>

ACTIONS PER OPTIMIZATION TYPE

<table>
<thead>
<tr>
<th>Target</th>
<th>Serial scalar</th>
<th>Serial control</th>
<th>Serial memory</th>
<th>Vectorization</th>
<th>Multithreading</th>
<th>Offloading</th>
</tr>
</thead>
<tbody>
<tr>
<td>main_pwd006.c:matmul</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

ACTIONS PER LOOP TYPE PER OPTIMIZATION TYPE

<table>
<thead>
<tr>
<th>Loop Type No.</th>
<th>Loops</th>
<th>Serial scalar</th>
<th>Serial control</th>
<th>Serial memory</th>
<th>Vectorization</th>
<th>Multithreading</th>
<th>Offloading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Medium</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

\textbf{(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.