Codee Training Series

April 25-26, 2023

Shift Left Performance

Automated Code inspection for Performance

April 24th, 2023

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)

First consideration 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/

The source code of MATMUL using double**

```
6    void matmul(size_t m, size_t n, size_t p, double **A, double **B,
double **C) {
7        // Initialization
8        for (size_t i = 0; i < m; i++) {
9        for (size_t j = 0; j < n; j++) {
10               C[i][j] = 0;
11               }
12       }
13
14       // Accumulation
15       for (size_t i = 0; i < m; i++) {
16               for (size_t j = 0; j < n; j++) {
17               for (size_t k = 0; k < p; k++) {
18               C[i][j] += A[i][k] * B[k][j];
19               }
20               }
21       }
22   }
23
24   int main(int argc, char *argv[]) {
. . .
40       // Allocates input/output resources
41       double **in1_mat = new_matrix(rows, cols);
42       double **in2_mat = new_matrix(rows, cols);
43       double **out_mat = new_matrix(rows, cols);
. . .
59       matmul(rows, cols, cols, in1_mat, in2_mat, out_mat);
```

```
3    // Creates a new dense matrix with the specified rows and columns
4    double **new_matrix(size_t rows, size_t cols) {
5        if (rows < 1 || cols < 1)
6        return NULL;
7
8        // Allocate a dynamic array of doubles to store the matrix data linearized
9        size_t matBytes = cols * rows * sizeof(double);
10       double *memPtr = (double *)malloc(matBytes);
11       if (!memPtr) {
12               return NULL;
13       }
14
15       // Allocate an array of pointers to store the beginning of each row
16       double **mat = (double **)calloc(rows, sizeof(double *));
17       if (!mat) {
18               free(memPtr);
19               return NULL;
20       }
21
22       // Set the row pointers (eg. mat[2] points to the first double of row 3)
23       for (size_t i = 0; i < rows; i++)
24               mat[i] = memPtr + i * cols;
25
26       return mat;
27   }
```

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

```
5    // C (m x n) = A (m x p) * B (p x n)
6    void matmul(size_t m, size_t n, size_t p, double **A, double **B, double **C) {
7        // Initialization
8        for (size_t i = 0; i < m; i++) {
9                for (size_t j = 0; j < n; j++) {
10               C[i][j] = 0;
11               }
12       }
13
14       // Accumulation
15       #pragma omp target teams distribute parallel for map(to: A, B, C, m, n, p) map(from: C) schedule(static)
16       for (size_t i = 0; i < m; i++) {
17           for (size_t j = 0; j < n; j++) {
18               for (size_t k = 0; k < p; k++) {
19                       C[i][j] += A[i][k] * B[k][j];
20               }
21           }
22       }
23   }
```

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

Produce Codee Checks Report and follow suggestions!

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

```
$ pwreport --checks --verbose main_pwd006.c:matmul --include-tags gpu -- -lm -fast -I include/
Compiler flags: -lm -fast -I include/

[C] target compiler: <none> (Compiler Agnostic Mode)

CHECKS REPORT

main_pwd006.c:15:5 [PWD003]: missing 'A', 'B', 'C' and 'C' array ranges in data copy to accelerator device
  Suggestion: specify the 'A', 'B', 'C' and 'C' array ranges to be copied to device memory
  Documentation: https://www.codee.com/knowledge/pwd003

main_pwd006.c:15:5 [PWD006]: missing deep copies of non-contiguous arrays 'A', 'B' and 'C' in data transfer to accelerator device
  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
  Documentation: https://www.codee.com/knowledge/pwd006

main_pwd006.c:16:5 [PWR035]: avoid non-consecutive array access for variables 'A', 'B' and 'C' to improve performance
  Non-consecutive array access:
        19:                C[i][j] += A[i][k] * B[k][j];
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to avoid non-sequential access to variables 'A', 'B' and 'C'.
  Documentation: https://www.codee.com/knowledge/pwr035

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 23 ms
```

Benchmarking on Perlmutter @NERSC using Nvidia toolchain

```
$ nvc -fast -mp -target=gpu -Minfo=mp -I include matrix.c clock.c main_pwd006.c -o matmul_pwd006
matrix.c:
clock.c:
main_pwd006.c:
matmul:
         12, #omp target teams distribute parallel for
         12, Generating "nvkernel_matmul_F1L12_2" GPU kernel
         16, Loop parallelized across teams and threads(128), schedule(static)

$ ./matmul_pwd006 3000
- 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

Documentation at the Open Catalog of Best Practices

https://www.codee.com/knowledge/pwd006/

PWD006: Missing deep copy of non-contiguous data to the GPU

Issue

The copy of a non-scalar variable to an accelerator device has been requested but none or only a part of its data will be transferred because it is laid out non-contiguously in memory.

Relevance

The data of non-scalar variables might be spread across memory, laid out in non-contiguous regions. One classical example is a dynamically-allocated two-dimensional array in C/C++, which consists of a contiguous array of pointers pointing to separate contiguous arrays that contain the actual data. Note that the elements of each individual array are contiguous in memory but the different arrays are scattered in the memory. This also holds for dynamically-allocated multi-dimensional arrays as well as for structs containing pointers in C/C++.

In order to offload such non-scalar variables to an accelerator device using OpenMP or OpenACC, it is not enough to add it to a data movement clause. This is known as deep copy and currently is not automatically supported by either OpenMP or OpenACC. To overcome this limitation, all the non-contiguous memory segments must be explicitly transferred by the programmer. In OpenMP 4.5, this can be achieved through the *enter/exit data* execution statements. Alternatively, the code could be refactored so that it uses variables with contiguous data layouts (eg. flatten an array of arrays).

Actions

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.

Code example

The following OpenMP code declares that the bi-dimensional array *A* should be copied to the accelerator device (see the clause *map(tofrom:A)* and the data type *int*** of the array *A*). However, this is incorrect and will not copy the data to be accessed in the loop body because OpenMP treats the pointer *A* as a zero-length array. Thus, the actual data of the variable will not be copied. As a result, dereferencing A in the GPU will cause invalid memory accesses, since its data has not been copied.

```
void foo(int **A) {
   #pragma omp target teams distribute parallel for map(tofrom:A)
   for (size_t i = 0; i < 10; i++) {
    A[i][i] += i;
   }
}
```

Adding the array ranges (see *map(tofrom:A[0:10][0:10])*) could be seen as a solution:

```
void foo(int **A) {
   #pragma omp target teams distribute parallel for map(tofrom:A[0:10][0:10])
   for (size_t i = 0; i < 10; i++) {
    A[i][i] += i;
   }
}
```

However, this OpenMP code does not handle non-contiguous memory properly because deep copy is not automatically supported. Therefore, each contiguous memory segment must be individually mapped to the accelerator device. This can be done through OpenMP 4.5 *enter/exit data* execution statements as follows:

```
void foo(int **A) {
   #pragma omp target enter data map(to:A[0:10])
   for (size_t i = 0; i < 10; i++) {
    #pragma omp target enter data map(to:A[i][0:10])
   }

   #pragma omp target teams distribute parallel for
   for (int i = 0; i < 10; i++) {
    A[i][i] += i;
   }

   for (size_t i = 0; i < 10; i++) {
    #pragma omp target exit data map(from:A[i][0:10])
   }
   #pragma omp target exit data map(from:A[0:10])
}
```

codee

www.codee.com

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/