OpenMP GPU offloading

PRACE Summer of HPC 2022 - Day 4

Leon Bogdanović

University of Ljubljana, FME, LECAD lab

OpenMP GPU offloading

- OpenMP is a **directive-based method** that can be used to offload computation-intensive tasks to GPUs
- from OpenMP 4.0 on, **new device constructs support GPU offloading**
- the **execution model relies on the host**: on it the OpenMP program begins execution and then offloads tasks or data to a target device, e.g., GPU

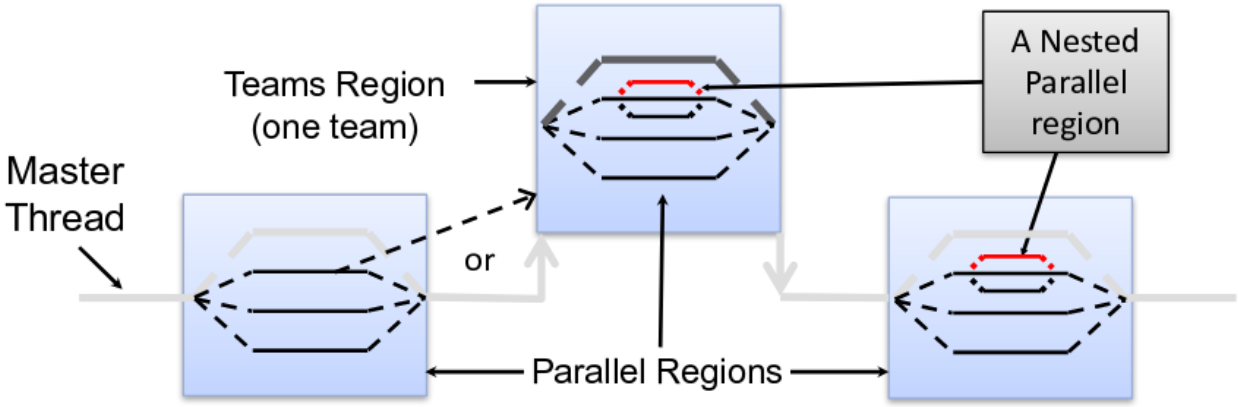
OpenMP device constructs and accelerator model

The main OpenMP device constructs are:

- `target`
- `teams`

The **target** construct:

- generates a target task
- an initial thread is running sequentially on a target device



Source of image: OpenMP Accelerator Model, IWOMP 2016

The **teams** construct:

- generates a league of thread teams
- the master thread of each team executes the region sequentially

Important OpenMP 4.x device constructs

Device construct	Description
#pragma omp target	Map variables to a device data environment and execute the construct on the device.
#pragma omp target data	Creates a data environment for the extent of the region.
#pragma omp declare target	A declarative directive that specifies that variables and functions are mapped to a device.
#pragma omp teams	Creates a league of thread teams where the master thread of each team executes the region.
#pragma omp distribute	Specifies loops which are executed by the thread teams.
#pragma omp ... simd	Specifies code that is executed concurrently using SIMD instructions.
#pragma omp distribute parallel for	Specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams.

Exercise 1: Riemann sum with OpenMP

Code:

`riemann_openmp_cpu_exercise.c`

Notebook:

`13_Riemann_sum_OpenMP_GPU.ipynb`

Complete the following tasks:

- use OpenMP to parallelize the for loop in the Riemann sum C code
- execute the code for the maximum threads available on the CPU (hint: use `!lscpu` to get information on the CPU)

Notebook **on Google Colaboratory**:

https://colab.research.google.com/drive/1f40HGhyKUiVobYIf_1IcGj5_e1BlQEJu

From OpenMP on CPU to OpenMP on GPU

Code:

`riemann_openmp_gpu.c`

Notebook:

`13_Riemann_sum_OpenMP_GPU.ipynb`

Offloading example - the Riemann sum computation on the GPU using OpenMP:

- to the OpenMP directive**:**

```
#pragma omp parallel for reduction(+:sum)
```

- appropriate **device constructs** are added to enable offloading to a GPU:

```
#pragma omp target teams distribute parallel for simd map(tofrom:
sum) map(to: n) reduction(+:sum)
```

- `map` is used to copy data from the host to the device and vice versa

Notebook **on Google Colaboratory**:

https://colab.research.google.com/drive/1f40HGhyKUiVobYIf_1IcGj5_e1BlQEJu

OpenMP GPU offloading with different compilers

Code:

`riemann_openmp_gpu.c`

Notebook:

`13_Riemann_sum_OpenMP_GPU.ipynb`

NVIDIA HPC (pgcc):

- `pgcc -fast -Minfo=all -ta=tesla -mp=gpu riemann_openmp_gpu.c -o riemann_openmp_gpu`
- if only `-mp` is used, then the OpenMP code will be executed on the CPU

GNU (gcc):

- `gcc-8 -O3 -Wall riemann_openmp_gpu.c -o riemann_openmp_gpu -fopenmp -foffload=-lm -fno-stack-protector -lm`
- a special offloading compiler to NVPTX, e.g., `gcc-8-offload-nvptx` and a plugin for offloading to NVPTX `libgomp-plugin-nvptx1` must be installed

Notebook **on Google Colaboratory**:

https://colab.research.google.com/drive/1Zbum3NETtjJvwzFfOIk3puaf_yMvNWjU

Exercise 2: Riemann sum with OpenMP vs. OpenACC

Code:

`riemann_openmp_gpu.c`

`riemann_openacc_gpu_solution.c`

Notebook:

`13_Riemann_sum_OpenMP_GPU.ipynb`

`16_Riemann_sum_OpenACC_GPU.ipynb`

Complete the following tasks:

- for the Riemann sum codes compare the performance of OpenMP vs. OpenACC offloading to GPU: which code is faster?
- also compare directive-based GPU offloading performance to CUDA and OpenMP CPU: which approach would you choose based on performance and effort?

Best practices for OpenMP on GPUs (source: NVIDIA)

- always use the `teams` and `distribute` directive to expose all available parallelism
- aggressively `collapse` loops to increase available parallelism
- use the `target data` directive and `map` clauses to reduce data movement between CPU and GPU
- use accelerated libraries whenever possible
- use OpenMP tasks to go asynchronous and better utilize the whole system
- use host fallback (`if` clause) to generate host and device code
- give `loop` a try

THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu