Lab2 - Intel® Xeon Phi™ Programming

In this lab you will build and run a **1D linear convolution** algorithm.

1. Building and running on the Host

- Build the application, enabling OpenMP:

```
$ icpc –openmp native.cpp –xavx –o native
```

- Run the application on the host.

```
$ ./native input kernel /dev/null
# elements of input file: 100000000
# elements of kernel file: 1024
Number of OpenMP threads used: 32
Elapsed time = 0.453822 seconds
```

Note: The first two arguments are inputs to the program, the final argument is where the output is stored. Since we are not really interested in the values of the output, We send the output to ***`/dev/null`***.

Questions:

- How many threads were used at runtime?
- Can you explain, by looking at **`native.cpp`** how the number of threads being used is controlled?

2. Building and running on MIC

- Build the application:

```
$ icpc –openmp native.cpp –o native_mic –mmic
```

- SSH to the MIC card and run the application:

```
$ ssh mic0
$ ./native_mic input kernel /dev/null
# elements of input file: 100000000
# elements of kernel file: 256
Number of OpenMP threads used: 228
Elapsed time = 1.154362 seconds
```

Questions:

- Does the code run faster than the host version? If not, can you suggest why it might be slower than you expected?
- What's the purpose of the macro **__MIC__**. in **native.cpp**?

2. Building an Offload Enabled Version

There are two different offload models – explicit & implicit –in this lab we'll use the explicit method.

In the explicit offload model, code has been is added to the application to which code and data is offloaded to the MIC card.

Naïve Implementation

- Open the file **offload_explicit.cpp** and compare it with **native.cpp**. You should notice:
  - To meet the requirement to stream chunks of data to the coprocessor the input and output is partitioned. The chunk size can be modified via the variable **chunkSize** to find the sweet spot between overhead of transferring data and a sufficient workload size on the coprocessor.
  - The function **convolve(…)** has been declared with **__declspec(target(mic))** to make it available for the coprocessor. This function will be the only part that's compiled for the coprocessor. Note that it is also available on the host in case no coprocessor is found!
  - We offload the initialization of the OpenMP* runtime by using **#pragma offload target(mic)**. In return we get the number of available threads on the coprocessor.
  - While iterating over the different chunks of workload, each call to **convolve(…)** is offloaded by using **#pragma offload target(mic)**. The memory to transfer to the coprocessor (**input** & **kernel**) is specified, as well as the returned data (**output**). Note that both **input** and **output** have the same size (**chunkSize**). A requirement of the 1D linear convolution is to continue with the trailing **kernelSize – 1** elements. This is why those elements are copied in the beginning of **input** for the next chunk to compute.

- Now build and run the offload example:
- **$ icpc –openmp offload_explicit.cpp -o offload_explicit ↵**
  -offload-option,mic,compiler,"-O3"

```
$ ./offload_explicit input kernel /dev/null
# elements of input file: 100000000
# elements of kernel file: 256
Number of OpenMP threads used: 224
Elapsed time = 3.436618 seconds
```

Questions:

- Can you guess what the **`-offload-option`** option is doing?
- How do you think this code will behave when there is no coprocessor present?
  Hint - You can simulate this condition by recompiling with option ***`–no-offload`***

Advanced Implementation

In the previous example the data first offloaded, and then the computation took place: it's strictly **sequential**.

A much better approach would be to interleave transfer of data with computation

- Look at **`offload_explicit_v2.cpp`**. You should see that:
  - The buffer for input data has been doubled (**`input1`** & **`input2`**). This is required for the coprocessor to work with one buffer while the other is filled. Note that the host side will stay with using **`input1`**; the mapping to either **`input1`** or **`input2`** for the coprocessor takes place with the **`into`** clause.
  - Allocation and de-allocation of the buffers is done only once, in the beginning and end respectively. *Can you spot what controls the conditional allocation?*
  - The asynchronous transfer and population of buffers uses signals.
  - The primary loop already expects that the first data chunk has been transferred. So, a small bootstrapping is required in front of the loop to ensure data is transferred already.
  - There is additional code to handle the last chunk of data transferred .

- Now build and run the example:

```
$ icpc –openmp offload_explicit_v2.cpp -o offload_explicit_v2
$ ./offload_explicit_v2 input kernel output
# elements of input file: 100000000
# elements of kernel file: 256
Number of OpenMP threads used: 224
Elapsed time = 1.651928 seconds
```

Observing Transfers

You can also observe the communication (data transfer) between host and coprocessor by setting the environment variable **`$OFFLOAD_REPORT=x`** (level of reports **`x`**: **`1`**, **`2`** or **`3`**).

- Enable the reporting for the two different explicit offload versions:

```
$ OFFLOAD_REPORT=1 ./offload_explicit input kernel /dev/null
…
$ OFFLOAD_REPORT=1 ./offload_explicit_v2 input kernel /dev/null
…
```

Can you identify the different offloads?