Module 21 Lab
OpenACC CUDA Vector Add

GPU Teaching Kit – Accelerated Computing

OBJECTIVE

Implement a vector addition using OpenACC directives.

LOCAL SETUP INSTRUCTIONS

The most recent version of source code for this lab along with the build-scripts can be found on the Bitbucket repository. A description on how to use the CMake tool in along with how to build the labs for local development found in the README document in the root of the repository.

The executable generated as a result of compiling the lab can be run using the following command:

```
./OpenAccVectorAdd_Template -e <expected.raw> \
  -i <input0.raw>,<input1.raw> -o <output.raw> -t vector
```

where `<expected.raw>` is the expected output, `<input0.raw>,<input1.raw>` is the input dataset, and `<output.raw>` is an optional path to store the results. The datasets can be generated using the dataset generator built as part of the compilation process.

LOCAL DEVELOPMENT & OBTAINING A PGI COMPILER LICENSE

The usage of OpenACC directives requires access to the PGI OpenACC compiler. Please follow the instructions on Bitbucket repository to download the tools, generate the license file and install the license.

QUESTIONS

(1) Compare the implementation difficulty of this kernel compared to the CUDA implementation. What difficulties did you have with this implementation?

ANSWER: **CUDA programming requires a lot of boilerplate code and runtime calls that are totally removed when using OpenACC.**

(2) What features are available to you in CUDA or OpenCL that made programming in either simpler?

ANSWER: **Answers may vary, but for vector add there is no debate that OpenACC is simpler than CUDA or OpenCL.**

CODE TEMPLATE

The following code is suggested as a starting point for students. The code handles the import and export as well as the checking of the solution. Students are expected to insert their code is the sections demarcated with `//@@`. Students expected the other code unchanged. The tutorial page describes the functionality of the `wb*` methods.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <wb.h>

int main(int argc, char *argv[]) {
  wbArg_t args;
  float *__restrict__ input1;
  float *__restrict__ input2;
  float *__restrict__ output;
  int inputLength;

  args = wbArg_read(argc, argv);

  wbTime_start(Generic, "Importing data and creating memory on host");
  input1 = (float *)wbImport(wbArg_getInputFile(args, 0), &inputLength);
  input2 = (float *)wbImport(wbArg_getInputFile(args, 1), &inputLength);
  output = (float *)malloc(inputLength * sizeof(float));
  wbTime_stop(Generic, "Importing data and creating memory on host");

  //@@ Insert vector addition code here.
  wbTime_start(GPU, "Copy to GPU, compute, and copy back to host.");
  wbTime_stop(GPU, "Copy to GPU, compute, and copy back to host.");

  wbSolution(args, output, inputLength);

  // Release memory
  free(input1);
  free(input2);
```

```c
  free(output);

  return 0;
}
```

CODE SOLUTION

The following is a possible implementation of the lab. This solution is intended for use only by the teaching staff and should not be distributed to students.

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <wb.h>

int main(int argc, char *argv[]) {
  wbArg_t args;
  float *__restrict__ input1;
  float *__restrict__ input2;
  float *__restrict__ output;
  int inputLength;

  args = wbArg_read(argc, argv);

  wbTime_start(Generic, "Importing data and creating memory on host");
  input1 = (float *)wbImport(wbArg_getInputFile(args, 0), &inputLength);
  input2 = (float *)wbImport(wbArg_getInputFile(args, 1), &inputLength);
  output = (float *)malloc(inputLength * sizeof(float));
  wbTime_stop(Generic, "Importing data and creating memory on host");

  //@@ Insert vector addition code here.
  wbTime_start(GPU, "Copy to GPU, compute, and copy back to host.");
  int i;
// sum component wise and save result into vector c
#pragma acc kernels copyin(input1[0 : inputLength],                          \
                                  input2[0 : inputLength]),                  \
                                  copyout(output[0 : inputLength])
  for (i = 0; i < inputLength; ++i) {
    output[i] = input1[i] + input2[i];
  }
  wbTime_stop(GPU, "Copy to GPU, compute, and copy back to host.");

  wbSolution(args, output, inputLength);

  // Release memory
  free(input1);
  free(input2);
  free(output);

  return 0;
}
```