Merge Sort (OpenACC)

Introduction

The purpose of this assignment is to implement MergeSort in C/C++ both as a serial algorithm and a parallel algorithm using OpenACC, as well as test the performance of these methods.

MergeSort algorithm

You need to write two programs, ***mergesort*** and ***mergesort_acc***, that will take as input the number of elements that should be sorted, N. The programs should first create an array of N doubles and fill it with random values. Then, they should sort the list and report the execution time for the sorting step. The programs should accept an optional parameter "-v" that would check that the values in the array are in sorted order, using the pseudocode below, and reports "Sorted" or "Not sorted". Finally, the programs should free memory and exit.

```
bool check_sorted(const double * const array, const size_t N){
      for(size_t i=1; i < N; ++i){
            if(array[i] > array[i-1]){
                  return false;
            }
      }
      return true;
}
```

Parallelization strategy

Feel free to try any parallelization strategy you wish, including strategies we discussed in class. The only requirement is that the sort must be entirely done on the GPU.

Testing

Test both programs for $N \in \{10K, 100K, 1M, 10M, 100M, 1B\}$. For the OpenACC method, the sort time should include data transfers between the host and the device.

What you need to turn in

1. The source code of your program.
2. A short report that includes the following:
   a. A short description of how you went about parallelizing the mergesort algorithm for the GPU.
   b. Timing results for your serial and parallel execution, both as a table and a figure showing parallel speedup for increasing sizes of the array. For each experiment, timing should be obtained as the average of 5 executions of the experiment for the given array length.
   c. A brief analysis of your results. Some things to consider might be:
      i. How does the size of the array affect the runtime? Why do you think that is?
      ii. Can you estimate the minimum array size that will lead to speedup (s > 1)?

Submission specifications

- A makefile must be provided to compile and generate the executable file.
- The executable files should be named 'mergesort' and 'mergesort_acc'.
- Your programs should take as arguments the size of the array, N, and an optional flag $-v$. For example, the program would be invoked as follows,

```
./mergesort 100000 -v
[or]
./mergesort_acc 100000 -v
```

- All files (code + report) MUST be in a single directory and the directory's name MUST be your university student ID. Your submission directory MUST include at least the following files (other auxiliary files may also be included):

```
<Student ID>/mergesort.c
<Student ID>/mergesort_acc.c
<Student ID>/Makefile
<Student ID>/report.pdf
```

- Submission MUST be a tar.gz archive.
- The following sequence of commands should work on your submission file:

```
tar xzvf <Student ID>.tar.gz
cd <Student ID>
make
ls -ld mergesort
ls -ld mergesort_acc
```

This ensures that your submission is packaged correctly, your directory is named correctly, your makefile works correctly, and your output executable files are named correctly. If any of these does not work, modify it so that you do not lose points. I can answer questions about correctly formatting your submission BEFORE the assignment is due. Do not expect questions to be answered the night it is due.

Evaluation criteria

The goal for this assignment is for you to become more familiar with the OpenACC API and develop efficient parallel programs. As such, the following things will be evaluated:

1. follows the assignment directions,
2. solve the problem correctly,
3. do so in parallel,
4. achieve speedup:
    - 5 / 10 points will be reserved for this criterion.
    - points will be awarded by comparing the relative performance of your parallel solution to the solution of the serial program.
    - The speedups obtained will probably depend on the size of the array size. It is not expected that you get good speedups for small N values.