COMP 429/529 Parallel Programming: Project 1

Due: 11.00 pm on Monday, November 1, 2021

Notes: You may discuss the problems with your peers but the submitted work must be your own work. Late assignment will be accepted with a penalty of -20 points per day. Please submit your source code through blackboard. This assignment is worth 20% of your total grade. **This is an individual assignment - no partners.**

Description

In this assignment you will implement a parallel version of the Marching Cubes algorithm in OpenMP. Additionally, the algorithm transforms (twists) the initial shape and saves the output at every step. You don't have to fully understand all the details about the algorithm and the data structures it uses, but some level of understanding is essential.

Marching Cubes

Marching Cubes is a simple algorithm for creating a triangle mesh from a uniform grid of cubes superimposed over a region of the function. Each vertex in the mesh defines whether it's the inside or the outside of a shape that is being represented. If all 8 vertices of the cube are positive, or all 8 vertices are negative, the cube is entirely above or entirely below the surface and no triangles are emitted. Otherwise, the cube straddles the function and some triangles and vertices are generated. Since each vertex can either be positive or negative, there are technically 2^8 possible configurations, but many of these are equivalent to one another. There are only 15 unique cases (other cases are rotated version of these), shown here:

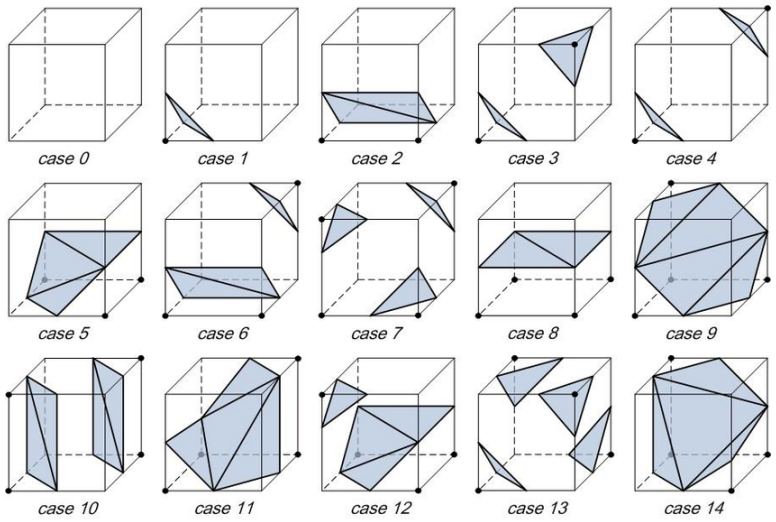

Figure 1: 15 unique triangle meshing configurations.

You can see an example of meshing a sphere on a figure below. (a) shows a binary (voxel) grid. It can serve as an input to the Marching Cubes algorithm to generate smoother surfaces at arbitrary angles, to generate a shape show on Figure 2 (b).

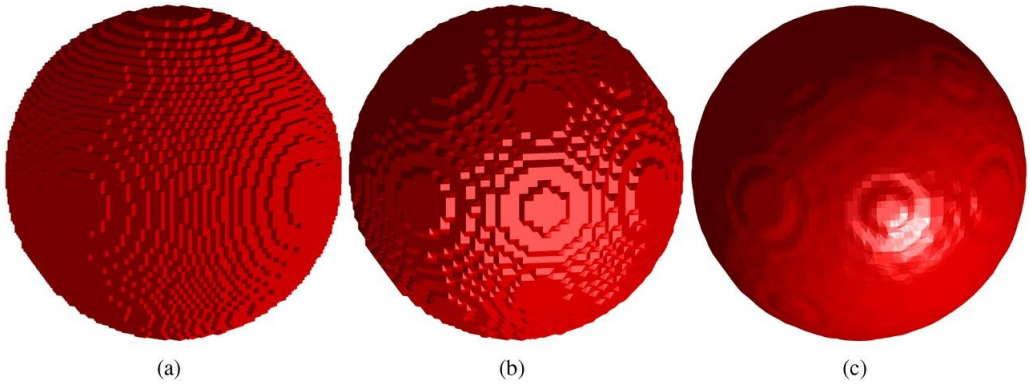

Figure 2: (a) voxel grid. (b) marching cubes. (c) smoothing + marching cubes

However, instead of working on a voxel representation of a shape, the code provided to you uses Signed Distance Function (SDF) representation of the shapes. When passed the coordinates of a point in space, SDF returns the shortest distance between that point and some surface. The transformation of the shape is also done by using the SDF representation. You won't need to modify the shape given to you.
You can find additional details in the sources given below (clickable)

- Video animation of how a mesh is produced
- Reading with some code descriptions
- Presentation with multiple example images
- Information on SDF (you don't need the ray marching part)
- Sample SDF functions and transformations

Transformation

The transformation that you are going to use takes the SDF of a shape and twists it by a specific amount of degrees defined by a variable *twist*. The program generates a number of frames specified by a variable *frameNum*. In each subsequent frame *twist* grows until it reaches *maxTwist*, resulting in *frameNum* objects generated. An example with 5 such frames is shown on Figure 3. Note that increasing the number of frames would generate more objects that represent the intermediate steps, instead of further twisting it.

Figure 3: Transforming (or twisting) a shape shown on 5 frames

Serial Code

We are providing you with a working serial program that implements the Marching Cubes algorithm. The provided code creates a smooth triangle mesh of a shape provided, given the desired resolution.

```
The program may be run from the command prompt as follows:
./mcubes [-n <int>] [-f <int>] [-t <int>] [-c] [-o]

With the arguments interpreted as follows:
-n <int> Number of mesh cubes in one dimension (resolution)
-f <int> Number of frames
-t <int> Thread count
-c enables testing (comparing against the serial implementation)
-o enables saving output .obj files

NOTE:
-c option should be DISABLED in ALL the reported experiments
-o option should be enabled in SOME experiments (when mentioned)
```

The workflow of the serial algorithm given to you is the following:

- Starting in the *main* function (from "main.cpp" file), read the command line arguments and initialize the global variables.
- Initialize the SDF of the shape and the transformation operation.
- For every frame in range $[0, frameNum)$ run *MarchCubes* (from "MeshReconstruction.cpp" file)

MarchCubes:

- For every cube in the mesh, map the vertices to corresponding edges that will form triangles for meshing.
- Save the vertices, normals (needed for a smooth appearance) and triangles of the mesh.

Saving:

- If -o option is enabled, save the mesh in .obj format.

Requirements

To run the program, you will only need a C++14 compiler and OpenMP library installed. On KUACC machines, C++ compilers (LLVM and GCC) are available as loadable modules.

```
To compile using gcc, type
      cmake . #once is enough, unless you change the compiler
      make
To compile using clang, type
      cmake -D CMAKE_C_COMPILER=clang -D CMAKE_CXX_COMPILER=clang++ .
      make
To clean the objects and executables
      rm CMakeCache.txt #if needed to change the compiler before the cmake command
      make clean
Example run:
       ./demo/mcubes -n 100 -f 20 -t 8 -c -o
Example command series to recompile and rerun:
       make clean; make; ./demo/mcubes -n 100 -f 20 -t 8 -c -o
```

Output files

The easiest way to view the output (.obj) files is to use online tools, like 3dviewer.net or creators3d.com. In case you want to see the files from the KUACC cluster machine, there are 2 ways to do so: 1) to *scp* the files to your machine first, then use any viewer you want. 2) ssh to the cluster with -X option (it will enable opening a window with a graphical interface) and use tools like obj-viewer.

Assignment

Part 1: Single-Frame Parallelization

In this part of the assignment you will have to implement an OpenMP version of the program, which will create a single (running with -f 1) object frame. In this part you will have to report your experiment (scaling) results with -c option *disabled* and answer the following questions (in the report):

- What is (are) the type(s) of parallelism achievable and why?
- What is the minimum resolution that consistently gives you at least 2x performance improvement against the serial code?
- Were you able to achieve perfect scaling or not? Why do you think you are getting the result you observe?

Part 2. Multi-Frame Parallelization

In this part of the assignment, you will have to implement parallel version of the program with OpenMP, but this time you are not limited by the number of frames. In this part you

will have to report your performance scaling results with -c option *disabled* and answer the following questions in the report:

- Does the approach from the previous part work in this case as well? Is it possible to achieve better scaling than in Part 1 and how?
- What is (are) the type(s) of parallelism achievable and why?
- What is the minimum resolution that consistently gives you at least 2x performance improvement against the serial code if 50 frames are being generated?
- Were you able to achieve perfect scaling or not? Why do you think you are getting the result you observe?

Part 3. Inter-Frame + Intra-Frame Parallelization

In this part of the assignment, you will have to combine the approaches from Part 1 and Part 2. You should've been able to observe the difference between the approaches by now and see in what cases one works better than the other. In this part you will need to generalize your approach, making your parallel code efficient in any of such cases. Again, you need to disable correctness tests and answer the questions below:

- What are the cases when you would like to use the combined approach?
- What feature of OpenMP are you using to implement it?
- Were you able to achieve perfect scaling or not? Why do you think you are getting the result you observe?

Experiments

You are going to conduct an experimental performance study on KUACC. The aim of this experimental study is to get performance data across different numbers of threads with different mesh sizes (resolution) and number of frames. In these experiments, **you will disable the testing feature** and report separately **with -o option both disabled and enabled**.

Part 1

- The first experiment will be a strong scaling study such that you will run your parallelized code multiple times under different thread counts. Please use the command line arguments below for this study.

```
bash$ ./mcubes -n 320 -f 1 -t <num-of-threads>
```

  <num-of-threads>will be 1, 2, 4, 8, 16, and 32. Plot the speedup and execution time figures as a function of thread count and include them in your report.

- In the second experiment, you will evaluate your code's performance under different input sizes, but with fixed thread count. The command that you will run is:

```
bash$ ./mcubes -n <input-sizes> -f 1 -t 16
```

<input-sizes>that you will test are 20, 40, 80, 160 and 320. Plot the execution time as a function of input size.

Part 2 and Part 3

Note that you will have to report the same experiments both for Part 2 and Part 3.

- The first experiment will be a strong scaling study such that you will run your parallelized code multiple times under different thread counts. Please use the command line arguments below for this study.

```
bash$ ./mcubes -n 160 -f 32 -t <num-of-threads>
bash$ ./mcubes -n 160 -f 32 -t <num-of-threads> -o
bash$ ./mcubes -n 320 -f 6 -t <num-of-threads>
bash$ ./mcubes -n 320 -f 6 -t <num-of-threads> -o
```

<num-of-threads>will be 1, 2, 4, 8, 16, and 32. Plot the speedup and execution time figures as a function of thread count and include them in your report.

- In the second experiment, you will evaluate your code's performance under different input sizes and frame number, but with fixed thread count. The command that you will run is as follows:

```
bash$ ./mcubes -n <input-sizes> -f 10 -t 16
bash$ ./mcubes -n <input-sizes> -f 10 -t 16 -o
bash$ ./mcubes -n 80 -f <frame-num> -t 16
bash$ ./mcubes -n 80 -f <frame-num> -t 16 -o
```

<input-sizes>that you will test are 20, 40, 80, 160 and 320. <frame-num>that you will test are 1, 4, 6, 8, 32 and 100. Plot the execution time as a function of input size.

Compiler experiments

Do the performance experiments from the previous section (the one for Part 2 and Part 3) with both gcc and llvm (clang) compilers. Do you see any difference in terms of running time and scaling?

Important Remarks

In this assignment, you will only have to work on two files: *demo/main.cpp* and *lib/Meshreconstruction.cpp*. You don't need to do any changes to other files, although you are not restricted. Additionally, you can do any changes to data structures and their types if you like.

Important! To ensure that all of your performance data is taken from the same machine on the cluster, All performance data for experiments can be submitted as a single job. To collect data for both gcc and clang compilers in one job you can first compile them separately and provide separate path to the executables.

More details on how to run your code in KUACC cluster can be found in Section Environment below.

Submission

- Document your work in a well-written report which discusses your findings.
- Your report should present a clear evaluation of the design of your code, including bottlenecks of the implementation, and describe any special coding or tuned parameters.
- We have provided a timer to allow you to measure the running time of your code.
- Submit both the report and source code electronically through blackboard.
- Please create a parent folder named after your username(s). Your parent parent folder should include a report in pdf and a subdirectory for source code. Include all the necessary files to compile your code. Be sure to delete all object and executable files before creating a zip file.
- GOOD LUCK.

Environment

Even if you develop and test your implementations on your local machine or on the computer labs on campus, you must collect performance data on the KUACC cluster.

- Accessing KUACC outside of campus requires VPN. You can install VPN through this link: https://my.ku.edu.tr/faydali-linkler/
- A detailed explanation is provided in http://login.kuacc.ku.edu.tr/ to run programs in the KUACC cluster. In this document, we briefly explain it for the Unix-based systems. For other platforms you can refer to the above link.
- In order to log in to the KUACC cluster, you can use ssh (Secure Shell) in a command line as follows: The user name and passwords are the same as your email account.

```
1    bash$ ssh $<$username$>$@login.kuacc.ku.edu.tr
2    bash$ ssh dunat@login.kuacc.ku.edu.tr //example
```

- The machine you logged into is called login node or front-end node. **You are not supposed to run jobs in the login node** but only compile them at the login node. The jobs run on the compute nodes by submitting job scripts.

- To run jobs in the cluster, you have to change your directory to your scratch folder and work from there. The path to your scratch folder is

```
bash$ cd  /scratch/users/username/
```

- To submit a job to the cluster, you can create and run a shell script with the following command:

```
bash$ sbatch <scriptname>.sh
```

- To check the status of your currently running job, you can run the following command:

```
bash$ squeue -u <your-user-name>
```

  A sample of the shell script is provided in Blackboard along with the assignment folder. In the website of the KUACC cluster, a lot more details are provided.

- To copy any file from your local machine to the cluster, you can use the scp (secure copy) command on your local machine as follows:

```
scp -r <filename> <username>@login.kuacc.ku.edu.tr:/kuacc/users/<username>/
scp -r src_folder dunat@login.kuacc.ku.edu.tr:/kuacc/users/dunat/  //example
```

  -r stands for recursive, so it will copy the src_folder with its subfolders.

- Likewise, in order to copy files from the cluster into the current directory in your local machine, you can use the following command on your local machine:

```
scp -r <username>@login.kuacc.ku.edu.tr:/kuacc/users/<username>/fileToBeCopied  ./
scp -r dunat@login.kuacc.ku.edu.tr:/kuacc/users/dunat/src_code ./  //example
```

- To compile the assignment on the cluster, you can use the GNU (gcc) or LLVM (clang) compiler. The compilation commands and flags for the compilers are provided in a Makefile in the assignment folder. **Important!** Note that when you want to use the LLVM compiler, GCC needs to be loaded as well. Before using the compilers, you firstly need to load their module if they are not already loaded as follows:

```
bash$ module avail //shows all available modules in KUACC
bash$ module list   //list currently loaded modules.
bash$ module load gcc/9.3.0  //loads GNU compiler
bash$ module load llvm/8.0.0-omp  //loads LLVM clang compiler
```

- If you have problems compiling or running jobs on KUACC, first check the website provided by the KU IT. If you cannot find the solution there, you can always post a question on Blackboard.

- Don't leave the experiments on KUACC to the last minutes of the deadline as the machine gets busy time to time. Note that there are many other people on campus using the cluster.

Grading

- 45% Correct Implementation: 15% for each part
- 30% Experiments: 10% for each part
- 25% Report: speedup and scaling figures, answering questions, compiler experiment results, explaining your findings, reporting performance, etc.

References

https://github.com/Magnus2/MeshReconstruction
https://youtu.be/_xk71YopsA
http://paulbourke.net/geometry/polygonise/
https://bit.ly/3Dynbse
http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/
https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm
https://www.creators3d.com/online-viewer
https://github.com/justint/obj-viewer
https://3dviewer.net/

Figures

Long, Zhongjie and Nagamune, Kouki. (2015). "A Marching Cubes Algorithm: Application for Three-dimensional Surface Reconstruction Based on Endoscope and Optical Fiber". Information (Japan). 18, 1425-1437.
Shijie Yan and Qianqian Fang. (2020). "Hybrid mesh and voxel based Monte Carlo algorithm for accurate and efficient photon transport modeling in complex bio-tissues". Biomed. Opt. Express 11, 6262-6270.