CSE 5441 (Fall 2019, Dr. Jones)
Serial AMR (Lab 1)

Caleb Lehman
lehman.346@osu.edu

October 21, 2019

Overview

For this lab, I prepared a (serial) `C` program to perform a Adaptive Mesh Refinement (AMR)[1]. The input to the problem is a tiling of a rectuangular grid composed of disjoint boxes along with their initial temperatures[2]. The program then iteratively updates the values in each box based on the current values of neighboring boxes. In addition to the inputs described above, two additional inputs are given: α (affect-rate), which determines the magnitude of the effect of neighboring boxes, and ε (epsilon), which determines when convergence is reached and the program terminates.

Algorithm

Given α, ε, a description of grid-aligned boxes, and initial Domain Specific Values (DSV), the rough pseudocode for my implementation is as follows:

```
1:  procedure AMR(α, ε, N, Boxes, InitialDSV)
2:      DSV ← InitialDSV
3:      DSV' ← InitialDSV
4:      while (max DSV − min DSV) / max DSV > ε do
5:          for i < N do
6:              DSV'[i] ← (1 − α) · DSV[i] + α · Σ_{j∈nhbr} DSV[j] · Overlap(i, j, Boxes)
7:          end for
8:          DSV ← DSV'
9:      end while
10: end procedure
```

Note the use of the ε (epsilon) parameter on line 4 to check for convergence and the use of the α (affect-rate) parameter on line 6 to weight the effect of neighboring boxes[3]. Updated DSV are commited on line 8. The particular implementation details can be found in the `common.c` and `amr.c` files, which contain the code for pre-processing the input and running the stencil code.

Summary

- Runtimes increased when either parameter was decreased, as expected.
- Selected parameters $\alpha = 0.1$, $\varepsilon = 0.2$, which yielded a runtime of roughly 4 minutes to run with the `testgrid_400_12206` file. Full timing results with these parameters are presented in the results section.

[1] In this case, the *mesh* is static, so the program may be more accurately described as a serial *stencil code/computation.*

[2] While the values are specifically assumed to be temperatures, the setup is general enough to encompass any type of values. Throughout the report, I simply refer to them as "Domain Specific Values" (DSV).

[3] Some boxes touch the edge of the grid, in which case they are assigned themselves as an additional neighbor. This is equivalent to assuming that the area outside the grid has the same DSV as the nearest box.

Tests

Environment

The program was developed and tested on the Pitzer cluster at the Ohio Supercomputer Center.

For development, I loaded the `intel/18.0.3` module, which allowed the program to be compiled with version 18.0.3 of the `icc` C-compiler.

For testing, I loaded the `python/3.6-conda5.2` module, which loads a python environment with the `NumPy`, `SciPy`, and `Matplotlib` packages, amoung others. `Python` is only necessary for collecting and plotting the data from testing, not for the actual exectuion of the program.

Timing

The program was timed using several methods[4]:

- The `time` function declared in the `time.h` header
- The `clock` function declared in the `time.h` header
- The `clock_gettime` function declared in the `time.h` header
- The `UNIX` utility `time`

Test Files

Dr. Jones provided several files for testing purposes:

- `testgrid_50_78`: 50x50 grid with 78 boxes
- `testgrid_50_201`: 50x50 grid with 201 boxes
- `testgrid_200_1166`: 200x200 grid with 1166 boxes
- `testgrid_400_1636`: 400x400 grid with 1636 boxes
- `testgrid_400_12206`: 400x400 grid with 12206 boxes

[4]Each method reports in different units/structues, all of which were converted seconds.

Results

The first requirement of this project was to determine values for α (affect-rate) and ε (epsilon) such that the program converged in between 3 to 6 minutes when run on the `testgrid_400_12206` test file. I ran a sweep over both parameters, capturing the runtimes reported by `clock_gettime` (see Figure 1). As expected, the runtime increases when either of the parameters decrease.

I selected parameters $\alpha = 0.1$, $\varepsilon = 0.2$ and ran the program on each of the test files. The results are tabulated in Table 1. In particular, the program took roughly 4 minutes, 21 seconds to run on the `testgrid_400_12206` file.

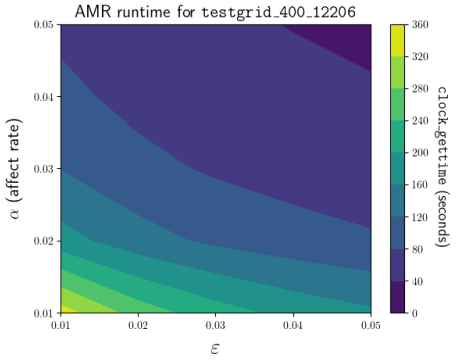

Figure 1: The runtime of the `amr` program on the `testgrid_400_12206` test file for various values of α, ε.

Test file	`#include <time.h>`			UNIX
	`time`	`clock`	`gettime`	`time`
`50,78`	0.00	0.01	0.02	0.07
`50,201`	0.00	0.07	0.08	0.12
`200,1166`	3.00	2.95	2.96	3.01
`400,1636`	7.00	7.12	7.13	7.18
`400,12206`	261.00	260.47	261.09	261.18

Table 1: Runtimes (seconds) for each of the test files using parameters $\alpha = 0.1$, $\varepsilon = 0.2$. Test file names are abbrviated from `testgrid_n_m` to `n,m`.

Project Usage

This section details basic commands needed to build and run the project and is only applicable for the code submission corresponding to this report. For full details, see the `README` included in the code submission.

Building

To build the `amr` executable, navigate to the top level of the submitted directory and build as follows:

```
# Ensure that you have icc compiler

$ make
...
$ ./amr
Usage: amr [affect-rate] [epsilon]

affect-rate: float value controlling the effect of neighboring boxes
epislon    : float value determining the cutoff for convergence
```

To compile this report from source:

```
# Ensure that you have pdflatex compiler
# and that the results/ directory has
# the necessary plots for the report

$ make report
```

Running

The syntax to run the program is:

```
$ ./amr [affect-rate] [epsilon] <[test-file]
```