HPC Assignment 2 - 1D Heat Diffusion

David Kroukamp 536705
and
Wade Pimenta 544147

May 21, 2015

Contents

1 1D Heat Diffusion **3**
1.1 Description 3
1.2 The Update Function 3

2 Serial Code **4**
2.1 Description of Functions 4
2.1.1 InitialiseToZero 4
2.1.2 DiffuseHeat 4
2.1.3 PrintPoints 4
2.1.4 ProcessOutput 4
2.1.5 main 5
2.2 How to run the code 5
2.3 Results 5
2.3.1 Input 5
2.3.2 Output 6

3 CUDA Code **7**
3.1 Description of Functions 7
3.1.1 InitialiseToZero 7
3.1.2 PrintPointsGPU 7
3.1.3 PrintPointsCPU 7
3.1.4 ProcessOutput 7
3.1.5 DiffuseHeat 8
3.1.6 main 8
3.1.7 DiffuseHeatCPU 9
3.2 How to run the code 9
3.3 Results 9
3.3.1 Input 9
3.3.2 Output 9

4 Adapted Code **11**

References **12**

1 1D Heat Diffusion

1.1 Description

1D heat diffusion is the diffusion of heat along an infinitely narrow pipe. Initially, the whole pipe is at a stable and fixed temperature; for clarity purposes, we set our pipes initial temperature to be zero. At the start (time 0), we set both ends to a specified temperature, which remains fixed through the computation. We then calculate how the temperatures change in the rest of the pipe over time. Mathematically, the problem is to solve a 1D differential equation representing heat diffusion:

$$\frac{\delta u}{\delta t} = \frac{\delta^2 u}{\delta x^2}$$

Our approach is to discretize the problem space by representing U by a one-dimensional array and computing values for a sequence of discrete time steps. Assume we have the values for U at time step k in an array Uk, then for the next time step $k+1$ update the second array $Ukp1$ as

$$Ukp1[i] = Uk[i] + \frac{dt}{dx \times dx}(Uk[i+1] - 2Uk[i] + Uk[i-1])$$

, where dt and dx represents the intervals between discrete time steps and between discrete points respectively.
We have solved this problem by implementing the update function above in a serial program, an MPI program, and a CUDA program.

1.2 The Update Function

The update function for 1D heat diffusion is

$$Ukp1[i] = Uk[i] + \frac{dt}{dx \times dx}(Uk[i+1] - 2Uk[i] + Uk[i-1])$$

. The issue we face when using this function was if $dx \times dx \geq dt$ then the values on our pipe would tend toward negative infinity. Upon further investigation, we found that the value of $\frac{dt}{dx \times dx} \leq \frac{1}{4}$ is optimal. [Hea] also mentions that dx is not the interval between discrete points, but rather the difference in temperature between $Uk[0]$ and $Uk[1]$. Since we still has issue getting the equation to work correctly for certain values of dt and dx, we decided to rather set $\frac{dt}{dx \times dx} = \frac{1}{4}$.

2 Serial Code

2.1 Description of Functions

2.1.1 InitialiseToZero

Input
: An array of floats.

Purpose
: Initialises all values in the array to zero.

2.1.2 DiffuseHeat

Input
: Initial array of temperatures along the pipe, an empty array to be used for temporarily storing the next set of temperatures at each time step, the heat being applied, the time steps, and the time which we want to stop at.

Purpose
: Computes the resultant heats for the pipe after the given time has elapsed by evaluating the points at the given time steps.

2.1.3 PrintPoints

Input
: The array to be printed and the current time.

Purpose
: Prints out the elements of the given array. The current time is used as an input to give a title to the output.

2.1.4 ProcessOutput

Input
: The array needing to be saved, the test case number, and the runtime.

Purpose
: Saves the results for the given test case. This consists of the runtime for the given test case, and (if a reasonable amount of points) the resultant array.

2.1.5 main

Input

None.

Purpose

The heart of the serial program.
While the end of the input text file is not encountered, the following takes place:
First the necessary values are read from the input text file. These are: the number of points, the end time, the time steps, and the temperatures of the endpoints.
The necessary arrays are then initialised and the endpoints of the current temperatures are set to the specified temperatures.
The change in temperature is calculated.
The clock is set up, and the DiffuseHeat method is run.
The final runtime is calculated and the output is then processed for the test case.

2.2 How to run the code

1. Open the terminal in the same location as heat.c
2. Type: gcc -std=c99 heat.c -o heat
3. Type: qsub serial_pbs.pbs
4. To see if the program ran successfully, type: qstat
5. If the program ran successfully, to view the results, type: nano serial.log

2.3 Results

2.3.1 Input

The contents of the input file were as follows: 12 5 0.1 500
120 5 0.1 500
1200 5 0.1 500
12000 5 0.1 500
120000 5 0.1 500
1200000 5 0.1 500
Each line represents a test case of the format [number of points] [end time] [time step] [applied heat]

2.3.2 Output

The results after running the code on the input were as follows:

```
Runtime for test case 1 with 12 points:
0
Resultant temperatures:
500.00 437.25 379.59 331.69 297.43 279.58 279.58 297.43 331.69 379.59 437.25
500.00

Runtime for test case 2 with 120 points:
0

Runtime for test case 3 with 1200 points:
0

Runtime for test case 4 with 12000 points:
7

Runtime for test case 5 with 120000 points:
85

Runtime for test case 6 with 1200000 points:
846
```

Note: the runtimes are calculated in milliseconds.

3 CUDA Code

3.1 Description of Functions

3.1.1 InitialiseToZero

Input

An array of floats.

Purpose

Initialises all values in the array to zero.

3.1.2 PrintPointsGPU

Input

An array, the size of the array, and the current time.

Purpose

Prints out the elements of the given array. The current time is used as an input to give a title to the output. (This method was used to easily test the output while the CUDA portion of the code was being executed.)

3.1.3 PrintPointsCPU

Input

An array and the current time.

Purpose

Prints out the elements of the given array. The current time is used as an input to give a title to the output.

3.1.4 ProcessOutput

Input

The array needing to be saved, the test case number, and the runtime.

Purpose

Saves the results for the given test case. This consists of the runtime for the given test case, and (if a reasonable amount of points) the resultant array.

3.1.5 DiffuseHeat

Input

Initial array of temperatures along the pipe, an empty array to be used for temporarily storing the next set of temperatures at each time step, the number of points, the heat being applied, the time steps, and the time which we want to stop at.

Purpose

Uses CUDA to compute the values for the temperatures along the pipe at each time step. This is done by having each thread calculate the temperature value at its respective index. The current temperatures are then updated to be the calculated temperatures and the current time is incremented by the time step value.

3.1.6 main

Input

None.

Purpose

The heart of the program.
The program first deletes the output file (if it exists). This is done because we don't want to append to the output file; we only want output that corresponds to the given input.
While the end of the input text file is not encountered, the following takes place:
First the necessary values are read from the input text file. These are: the number of points, the end time, the time steps, and the temperatures of the endpoints.
The necessary arrays are then initialised for both the host and device arrays and the endpoints of the current temperatures are set to the specified temperatures.
The needed arrays are then copied to the device and the block size and grid size are calculated. (NOTE: NUMPOINTS-2 is used because we don't need to calculate the temperatures of index 0 and the final index.) The change in temperature is calculated.
The events needed to time the method are then declared, and the DiffuseHeat method is run.
The final runtime is calculated, the resultant array is then copied to the host result and the output is then processed for the test case.

3.1.7 DiffuseHeatCPU

Same as described in the serial version (in the previous section). It is in the parallel version to compare results and calculate speedup.

3.2 How to run the code

1. Open the terminal in the same location as heat.cu
2. Type: nvcc heat.cu -o heat_cuda
3. Type: qsub cuda_pbs.pbs
4. To see if the program ran successfully, type: qstat
5. If the program ran successfully, to view the results, type: nano cuda.log

3.3 Results

3.3.1 Input

The contents of the input file were as follows: 12 5 0.1 500
120 5 0.1 500
1200 5 0.1 500
12000 5 0.1 500
120000 5 0.1 500
1200000 5 0.1 500
Each line represents a test case of the format [number of points] [end time] [time step] [applied heat]

3.3.2 Output

The results after running the code on the input were as follows:
Runtime for test case 1 with 12 points:
0.065536
Resultant temperatures:
500.00 437.25 379.59 331.69 297.43 279.58 279.58 297.43 331.69 379.59 437.25 500.00

Runtime for test case 2 with 120 points:
0.049120

```
Runtime for test case 3 with 1200 points:
0.060416

Runtime for test case 4 with 12000 points:
0.224256

Runtime for test case 5 with 120000 points:
1.877984

Runtime for test case 6 with 1200000 points:
18.647009
```
Note: the runtimes are calculated in milliseconds.

4 Adapted Code

We used the following resources for help with certain aspects of our code:
[Arr] - An example on summing up an array using MPI in C.
[Rea] - An example on reading from a text file.
[Str] - An example on converting a string to a double.

References

[Arr] *MPI Example - Array Assignment - C Version.* `https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_array.c`. Accessed: 2014-05-18.

[Hea] *The 1D diffusion equation.* `http://hplgit.github.io/num-methods-for-PDEs/doc/pub/diffu/sphinx/._main_diffu001.html`. Accessed: 2014-05-15.

[Rea] *How to read a text file by c program.* `http://www.cquestions.com/2010/06/how-to-read-text-file-by-c-program.html`. Accessed: 2014-05-18.

[Str] *Convert string to double.* `http://www.cplusplus.com/reference/cstdlib/atof/`. Accessed: 2014-05-18.