CSS434 Parallel and Distributed Computing

Spring 2014

Program 2: Heat2D

Christopher Tran

Student ID: 1368253

# Design and Implementation

## Design

Based on Heat2D and the requirement to use a single one-dimensional array, I adapted the Heat2D class to use a single one-dimensional array. This was then used as the base of conversion to a MPI supported version. Using MatrixMult as a template for calculating and allocating sections to each node, starting, ending and offsets were all stored in arrays by rank index so they would not have to be calculated for each cycle.

From then, it was a process of mimicking Heat2D using MatrixMulti as a design template. The only section that needed to be distributed was Eurlers algorithm. The others were so minimal in costs that attempts to optimize them resulted in no performance gains.

By preparing for parallel before Eulers algorithm, we can achieve the most effective and simple distribution of workload. Ranks will pass border information before executing the algorithm as expected in the specifications. To keep the program simple, each rank would keep in memory the full size of the array but one process the section allocated to them based on the starting and ending rows.

## Optimizations Implemented

### Interval Printing

Executing time was getting significant and was narrowed down to the master accumulating data from the ranks. As data increased, each node would take longer each cycle to get information and print. This caused increase in nodes to have little improvement and beyond 4 nodes, the times were getting worse. It occurred to me that there was no need to gather data until an interval was reached. The code to gather information from the other ranks was implemented into the interval print code. This increased performance significantly.

### Staggered Communication

Following guidance from MPIJava documentation, its poor performance when all ranks are sending simultaneously. By staggering each rank to send and receive, you can achieve improved performance as half the ranks will be in either state instead of all in the same state. After implementing the staggered send\receive, the performance difference was minor. This will likely be more evident when node count is significant.

## Optimizations Not Implemented

### Master Heats

The master rank will always be assigned the bottom row, or in the case of the program, the top row viewed. Therefore, it’s a waste of a small number of cycles for each rank to heat the first row. Attempts to implement this restriction resulted in the row never getting the values needed.

### State Change

It occurred to me that the ranks above master do not have anything to do until the border from the rank below sends an array with values different than 0.0. There could be a design that would restrict Eulers method from processing until the border information is different than all 0.0 values. I did not have time to implement but I believe this would be a significant performance difference.

# Performance Evaluation

## Size 1000

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Run1** | **Run2** | **Run3** | **Run4** | **Average** | **Change** | **Change from Sequential** |
| Sequential | 7172 | 7576 | 7625 | 7329 | 7426 |  |  |
| 1 Node | 9162 | 9319 | 9420 | 9564 | 9366 | -1941 | -1941 |
| 2 Node | 7049 | 7841 | 7130 | 7197 | 7304 | 2062 | 121 |
| 3 Node | 5288 | 5510 | 5524 | 5518 | 5460 | 1844 | 1966 |
| 4 Node | 4484 | 4659 | 4800 | 5017 | 4740 | 720 | 2686 |
| 5 Node | 4211 | 4310 | 4168 | 4161 | 4213 | 528 | 3213 |
| 6 Node | 3808 | 3854 | 3922 | 3835 | 3855 | 358 | 3571 |
| 7 Node | 3876 | 3831 | 3800 | 3813 | 3830 | 25 | 3596 |
| 8 Node | 3651 | 3653 | 3655 | 3837 | 3699 | 131 | 3727 |

Performance gains were seen from 2 nodes and above

## Size 100

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Type | Run1 | Run2 | Run3 | Run4 | Average | Change | Change from Sequential |
| Sequential | 99 | 101 | 99 | 101 | 100 |  |  |
| 1 Node | 157 | 143 | 143 | 148 | 148 | -48 | -48 |
| 2 Node | 855 | 855 | 841 | 862 | 853 | -706 | -753 |
| 3 Node | 874 | 863 | 866 | 861 | 866 | -13 | -766 |
| 4 Node | 832 | 877 | 890 | 883 | 871 | -5 | -771 |

No performance gain

## Size 5000

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Run1** | **Run2** | **Run3** | **Run4** | **Average** | **Change** | **Change from Sequential** |
| Sequential | 166898 | 165763 | 163914 | 165615 | 165548 |  |  |
| 1 Node | 212005 | 213152 | 211646 | 212548 | 212338 | -46790 | -46790 |
| 2 Node | 111140 | 11182 | 111730 | 11403 | 61364 | 150974 | 104184 |
| 3 Node | 76394 | 76195 | 75589 | 75619 | 75949 | -14586 | 89598 |
| 4 Node | 59983 | 59951 | 60055 | 59916 | 59976 | 15973 | 105571 |
| 5 Node | 49696 | 50366 | 50030 | 54967 | 51265 | 8712 | 114283 |
| 6 Node | 44879 | 44050 | 46821 | 45004 | 45189 | 6076 | 120359 |
| 7 Node | 41723 | 41674 | 39536 | 39373 | 40577 | 4612 | 124971 |
| 8 Node | 36020 | 39251 | 36863 | 40196 | 38083 | 2494 | 127465 |

Performance gains significant at each step

## Conclusion

Single node performance was worse than Sequential. This is likely due to the additional overhead of executing MPI. This repeated no matter how large the array was. The single node performance was always larger than the sequential version.

As expected, performance gains were noticed when the size of the array was large and the larger the array, the more significant improvements. When the array size is small, the performance gains are either minimal or performance was worse. At 5000, the differences are significant. At 1000, the improvements are small but noticeable. At 100, there was no increase and the performance was worse.

After a certain point, the diminishing returns takes into effect. No matter how many nodes we add into the mix, the performance difference is minimal. However, there comes a breaking point there increasing the number of nodes makes performance worse.