**Equation Solver Using CUDA**

1. **Kernel Using Global Memory**

* Here we implement the Jacobian Equation solver using Global Memory.
* We declare a shared memory array to store the differences calculated by each thread.
* We locate the element whose value has to be calculated, perform the calculations to get the new value of the element and then calculate its difference from the old value. We store this difference in the shared memory declared above.
* Before calculating the diffs and updating the values in the grid we sync all the threads.
* Once we calculate the diffs of all threads we accumalate them into the first element of the array by using a tree like algorithm.
* Finally we add the first element of diff from every block to the d\_diff variable in the device. We do this with the help of mutex.

1. **Kernel Using Texture Memory**

* Here we implement the Jacobian Equation Solver using Texture Memory.
* We declare 2 1D texture grids and bind them in the device.
* We also declare a shared memory array to store the differences calculated by each thread.
* We locate the element whose value has to be calculated, fetch the values using tex1Dfetch function and store the updated value in the second grid.
* Once all the threads reach the \_\_syncthreads() we calculate the difference in values as well as update the grid.
* Once we calculate the diffs, we use the same algorithm as that used in the global memory case to calculate the final d\_diff value.

**Speedups and Sensitivity Analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Grid Size** |  | **2048 x 2048** | | **4096 x 4096** | | **8192 x 8192** | |
|  |  | **Global** | **Texture** | **Global** | **Texture** | **Global** | **Texture** |
| **Block Size** | **8 x 8** | **1.58** | **1.69** | **1.56** | **1.62** | **1.23** | **1.28** |
| **16 x 16** | **6.17** | **6.7** | **6.15** | **6.42** | **4.85** | **5.201** |
| **32 x 32** | **10.30** | **11.2** | **9.82** | **10.7** | **7.85** | **8.54** |

We observe here that as the block size increases the speedup achieved is higher. However as the grid size is increased the speedup reduces. I believe it is due to the fact that more number of threads need to be synchronized and hence it takes more time and hence lesser speedup.