CSCI551

German Razo

Report

**Data Storage**

.

To save the matrix data I used an array of pointers and each pointer points to an array of type double pointer of size of the matrix plus one. So, if matrix size 8, then every pointer in the matrix would point to an array of doubles of size plus one, so 9. I used size plus one because I stored the augmented matrix A | b including b in one array.

For the calculation of L2 norm I had to store the original augmented matrix in an array of double pointer array of type double and I saved the result of L2norm in a one pointer array of type double.

One of the most important reasons of why I used a double pointer array is because swapping the rows would be more efficient than swapping every element row by row. This gave a better performance to the times.

**Data partition and work and exploit parallelism**

To efficiently take advantage of parallelism I used it in two parts of my code, in forward substitution and back substitution.

Forward substitution: In this step, the two for loops run in parallel, so every row is divided among threads. Every thread performs the multiplication and division to obtain the resultant matrix after forward substitution.

Back substitution: I made back substation column oriented, so one of the benefits is that there is no loop dependence and the inner loop can be parallelized efficiently because every column is divided among the threads. Column oriented worked really well for me because I did row oriented before and I encountered race conditions in my program, however, column oriented prevents race conditions and are handled better and gives better performance. I parallelized the inner for loop because it is the loop that iterates the most and every thread is assign a column, so it exploits parallelism well.

**How the parallel work is synchronized**

Synchronizations happens at the end of the parallel block where all the threads join after the directives # pragma omp parallel. All threads are join to the master thread.

**Pseudocode of key elements of the parallelization strategy**

**what compiler/linker flags you chose to use and why**

**Justification for implementation choices**

I implemented the augmented matrix **A | b** with a two-dimensional double array because it is more efficient to swap only the addresses of every row instead of every element in the row. This implementation improved my timing and results. As mentioned, with column oriented in back substitution showed a better performance because every column was assigned to every thread.

**how successful your approach was, and what you would do differently or try next if you had more time!**

Table of minimum times from all runs and L2norm with minimum times highlighted with n = 8000

|  |  |  |  |
| --- | --- | --- | --- |
| # core | 1st run | 2nd run | 3rd run |
| 1 |  |  |  |
| 2 |  |  |  |
| 5 |  |  |  |
| 10 |  |  |  |
| 20 |  |  |  |
| 30 |  |  |  |
| 40 |  |  |  |
| 50 |  |  |  |
| 60 |  |  |  |

L2-norms for each run

|  |  |  |  |
| --- | --- | --- | --- |
| # core | 1st run | 2nd run | 3rd run |
| 1 |  |  |  |
| 2 |  |  |  |
| 5 |  |  |  |
| 10 |  |  |  |
| 20 |  |  |  |
| 30 |  |  |  |
| 40 |  |  |  |
| 50 |  |  |  |
| 60 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # cores | Minimum time | Speedup | Efficiency | L2 norm |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |