Gaussian Elimination with OpenMP

Lab #3

Jason Boutte
May 14, 2015

1. Data Storage

I chose to store the A matrix and b vector in an augmented matrix. To create the augmented matrix I created an array of pointers to pointers of size n, this created the array of rows. Each pointer in the array was then used to create array of doubles making up the columns of the matrix. The size of these arrays were n+1, where 1 to n represented A matrix and n+1 held the values of the b vector.

2. Partitioning

For the forward elimination portion of the code I chose to partition this by the rows that need updating during annihilation of the value in the current column.
Each thread is performing an annihilation of a value in its row.

```
FOR each master row of the matrix except the last row
  Find and swap the pivot row
  Parallelize for block
  FOR each row below the master row
    Calculate the factor for the master and current row
    FOR each column in the current row
      Calculate the intermediate value using column of master row and factor
      Subtract intermediate value from the current column of the current row
```

As for the back substitution portion I chose to partition by columns of the current row where each thread calculates a value for the column and subtracts it from the right hand side.

```
FOR each row of the matrix in reverse
  Assign the rows b value to a variable
  Parallelize for block
  FOR each column of the row in reverse not including the last value column and ending
before the diagonal
    Multiply value in the matrix of row and column by its corresponding result value
    Subtract this value from the variable
  Divide the variable by diagonal of the current row
```

3. Synchronization

Unlike OpenMPI where a barrier was needed to synchronize the processes before beginning the timing, OpenMP doesn’t require this because the program runs in serial until it encounters a parallel construct. At this point the threads are created, the code is duplicated and all threads execute the code. At the end of a parallel construct there is an implied barrier where all threads join the master and once again there isn’t a need for a barrier as there would be with OpenMPI. The way I chose to parallelize my code required no synchronization within the parallel regions.

4. Explanation

I chose to store the matrix and vector as an augmented matrix using pointers to pointers because it made swapping the pivot very fast as there was no need to perform copies on the arrays or swapping elements in the vector holding the b values. For parallelization I chose to partition the forward elimination by the first inner loop because the rows performing annihilation all depended on the same master row. I could not parallelize by the outer loop because each set of annihilations depended on the previous set updating their master row thus creating a non-parallelizable dependency. I chose to partition the back substitution by columns rather than rows because each row depended on the row after it to calculate its x value.

5. Timings

a. 1 Core

Run	Timing (Seconds)	L^2-norm
1	3086.64	0.00082
2	3101.90	0.00106
3	3084.77	0.00064
4	3086.14	0.00281
5	3088.81	0.00052

b. 2 Cores

Run	Timing (Seconds)	L^2-norm
1	1629.95	0.01649
2	1629.86	0.00109
3	1643.56	0.00197
4	1618.68	0.10568
5	1626.34	0.00259

c. 5 Cores

Run	Timing (Seconds)	L^2-norm

1	665.43	0.006694
2	666.04	0.001471
3	660.48	0.009977
4	659.12	0.000540
5	665.75	0.001626

d. **10 Cores**

Run	Timing (Seconds)	L^2-norm
1	377.57	0.00054
2	487.73	0.00186
3	504.59	0.01254
4	378.75	0.00245
5	379.46	0.00082

e. **20 Cores**

Run	Timing (Seconds)	L^2-norm
1	663.42	0.00211
2	350.82	0.00138
3	459.17	0.00037
4	347.92	0.01442
5	474.94	0.00060

f. **30 Cores**

Run	Timing (Seconds)	L^2-norm
1	462.82	0.00059
2	464.62	0.00032
3	343.84	0.00109
4	435.12	0.00033
5	464.57	0.00105

6. Speed and Efficiency

a. Speedup

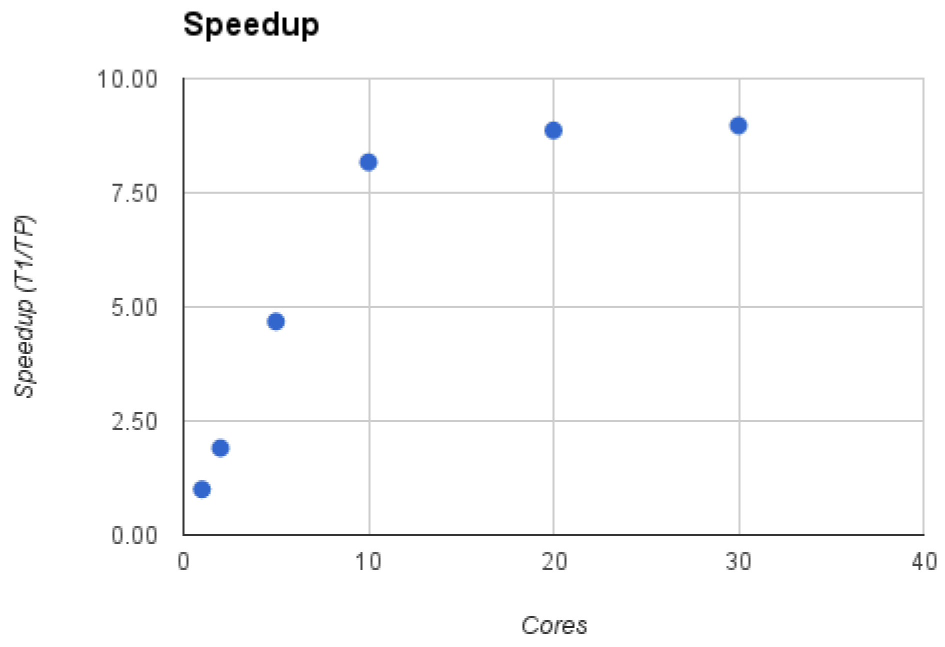
Cores	Speedup
1	1
2	1.91
5	4.68
10	8.17
20	8.87
30	8.97

b. Efficiency

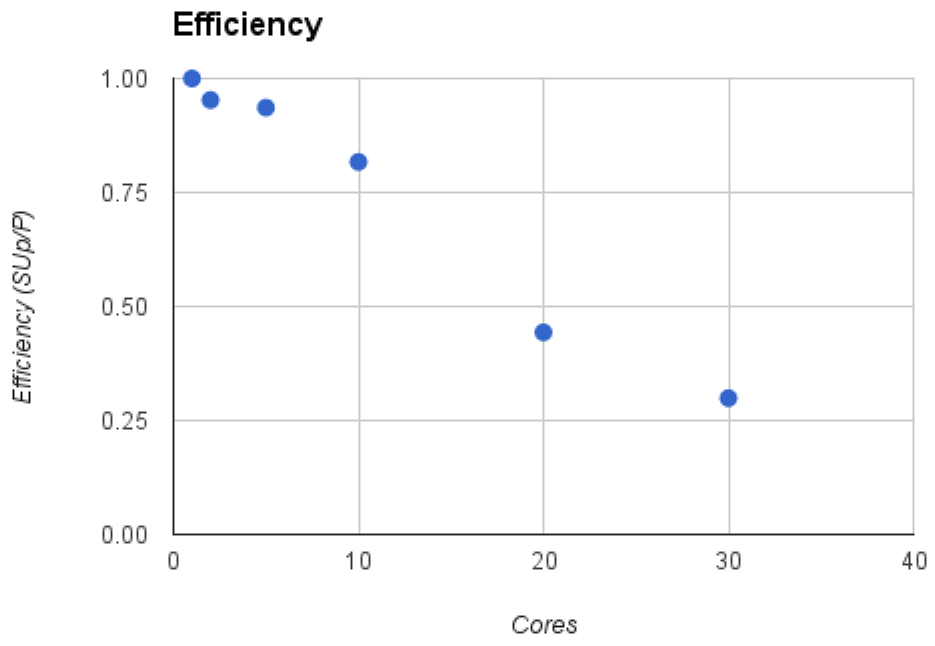
Cores	Efficiency
1	1
2	0.96
5	0.94
10	0.82
20	0.44
30	0.30

7. Graphs

a.


Speedup
Speedup (T1/TP)
10.00
7.50
5.00
2.50
0.00
0
10
20
30
40
Cores

b.


Efficiency
Efficiency (SUp/P)
1.00
0.75
0.50
0.25
0.00
0
10
20
30
40
Cores