Daniel Cheng
CS 378

December 13, 2013

Parallel Gaussian Elimination

Gaussian Elimination

Gaussian elimination is an algorithm for solving systems of linear equations that uses elementary row operations to transform a matrix into an upper triangular matrix (row echelon form), then back substitute until a solution is found (transform the matrix into reduced row echelon form). These elementary row operations are: 1) Swapping two rows, 2) Multiplying a row by a non-zero number, 3) Adding a multiple of one row to another row. These two phases are referred to as Forward Elimination and Back Substitution. I will focus on Forward Elimination, as this dominates the runtime.

Partial pivoting can be done to improve the numerical stability of the algorithm. By selecting the next pivot with the largest absolute value, one can avoid dividing by very small numbers and thus reduce possible errors that could result from the limitations of floating point representations.

The algorithm to perform forward elimination with partial pivoting on a matrix `A` is as follows. The parallelizable portion is shown in red.

```
for k = 1 ... m: Find pivot for column k:
  i_max  := argmax (i = k ... m, abs(A[i, k]))
  if A[i_max, k] = 0
    error "Matrix is singular!"
  swap rows(k, i_max)
  Do for all rows below pivot:
   for i = k + 1 ... m:
    Do for all remaining elements in current row:
     for j = k ... n:
      A[i, j]  := A[i, j] - A[k, j] * (A[i, k] / A[k, k])
    Fill lower triangular matrix with zeros:
    A[i, k]  := 0
```

This can be easily parallelized with an OpenMP `#pragma omp parallel for`. Each row below the pivot can be operated on in parallel because writes are restricted to its own row, and reads for each thread are restricted to its own row and the pivot row, to which no thread will write.

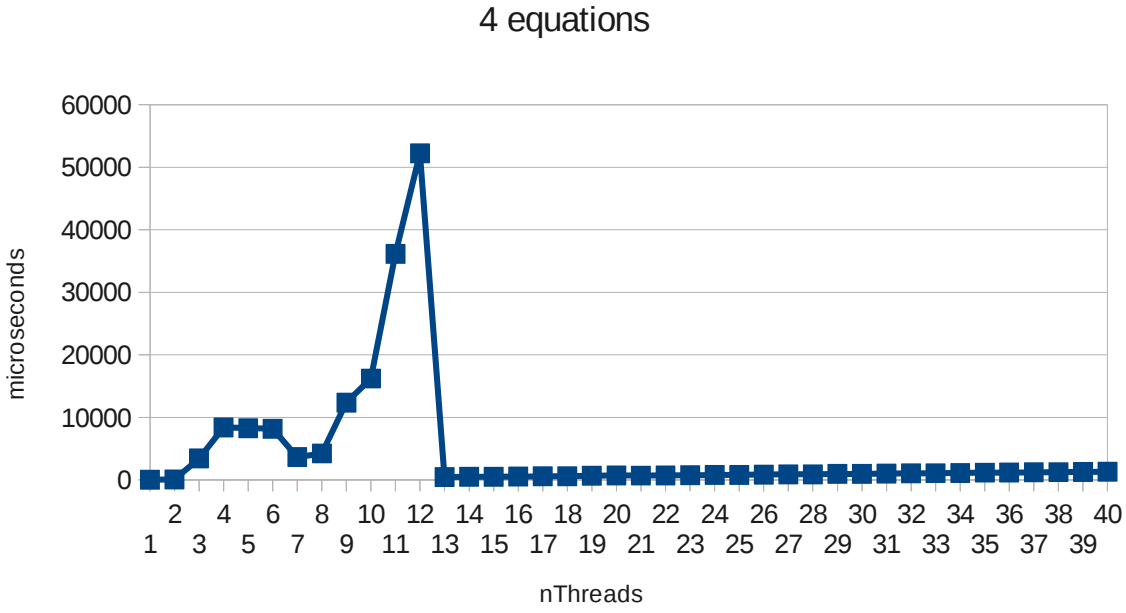
Results

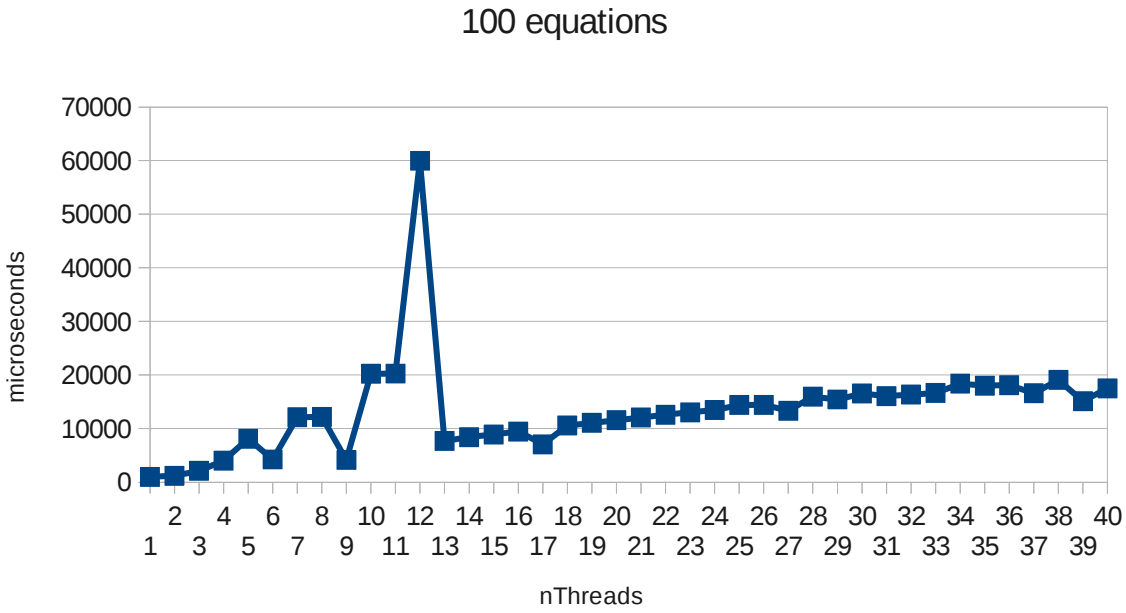
I parallelized the algorithm using OpenMP and ran it with nThreads from 1 up to 40 on Lonestar. I timed it with matrix sizes 4x5 (representing 4 equations with 4 variables), 100x101, 1000x1001, and 2000x2001.

Correctness was verified by using back-substitution to ensure that a correct solution was found for the

system of equations (if a solution existed).

In the smaller matrices, the time to create and schedule threads dominated the overall runtime, as shown by the graphs (they peak around 12, i.e. the number of cores). In these cases, there was no gain from running with more threads.



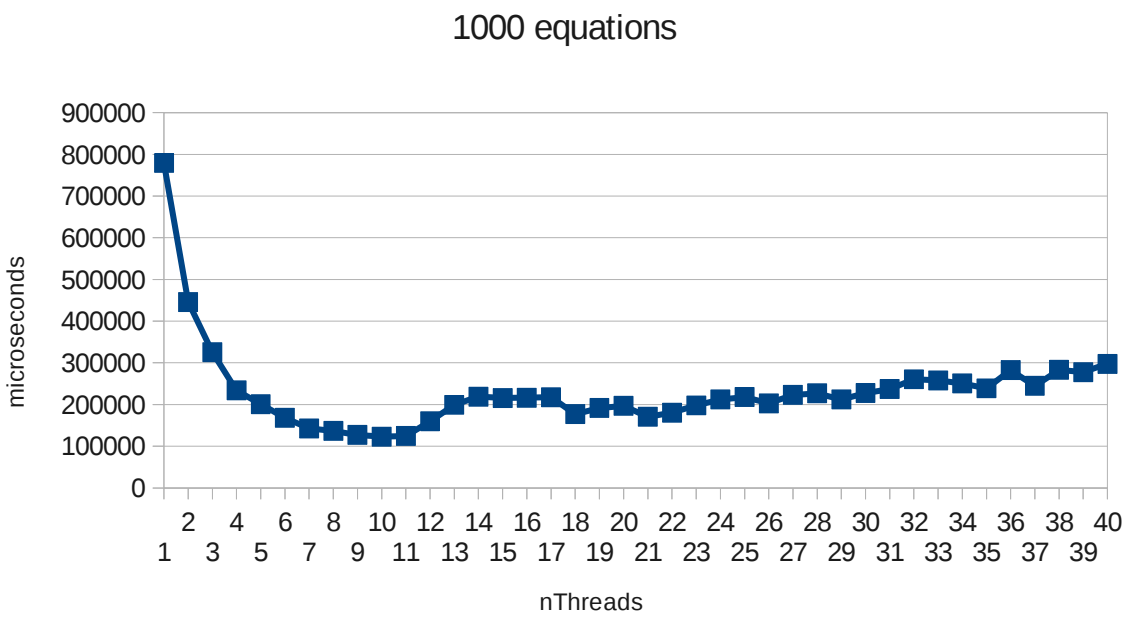


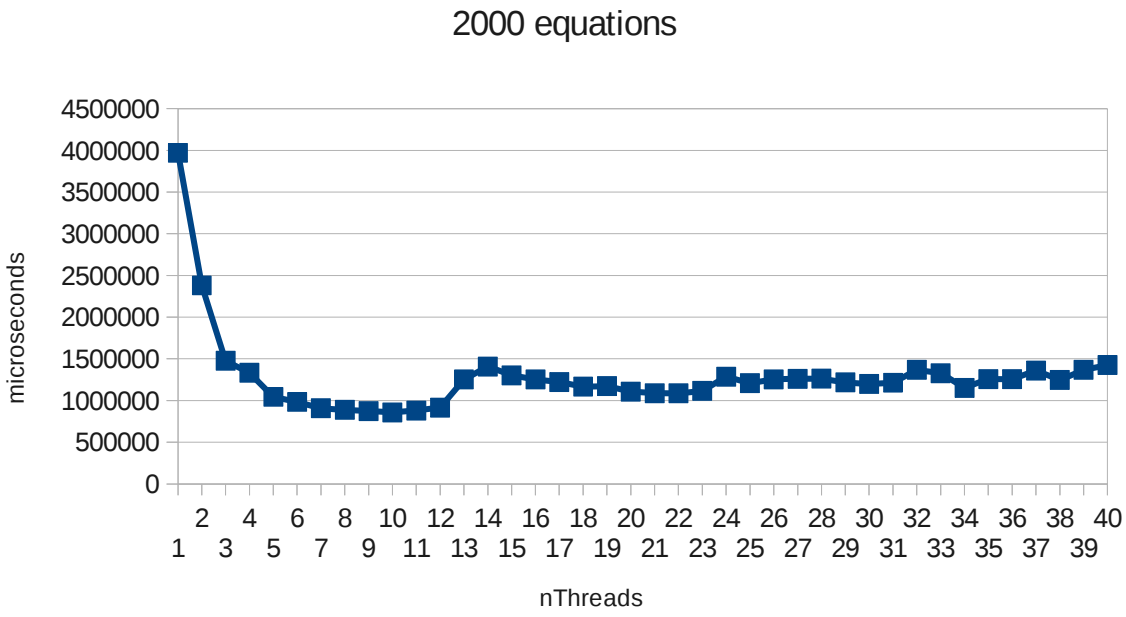
I found it strange that the peak exists even with 4 equations, because I assumed that OpenMP would simply ignore the extra unneeded threads.

Daniel Cheng
CS 378

December 13, 2013

With more equations, the benefit of running with more threads becomes more evident.



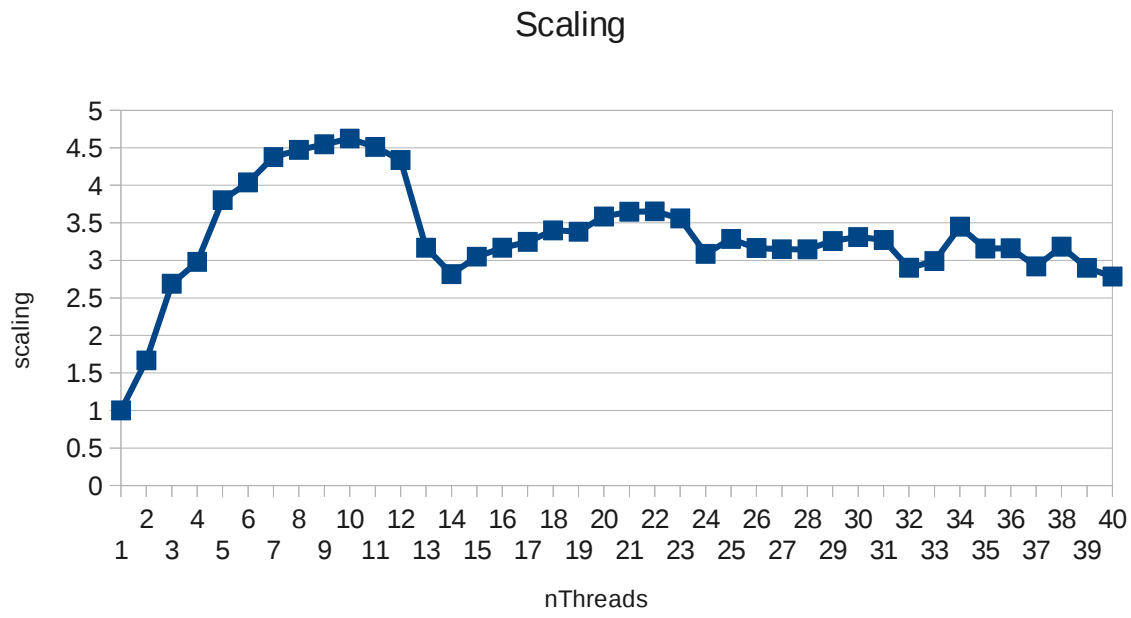


As expected, runtime is best with ~12 threads. With more, scheduling gets in the way and runtime increases.

Daniel Cheng
CS 378

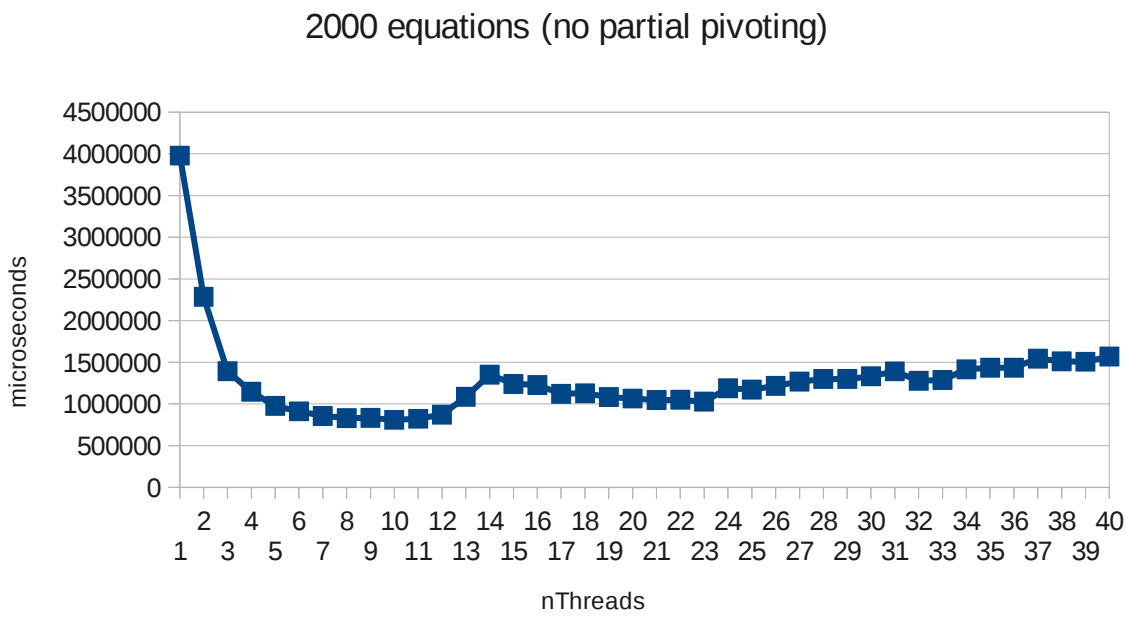
December 13, 2013

Scaling peaks at around ~12 threads at 4.5x performance gains.

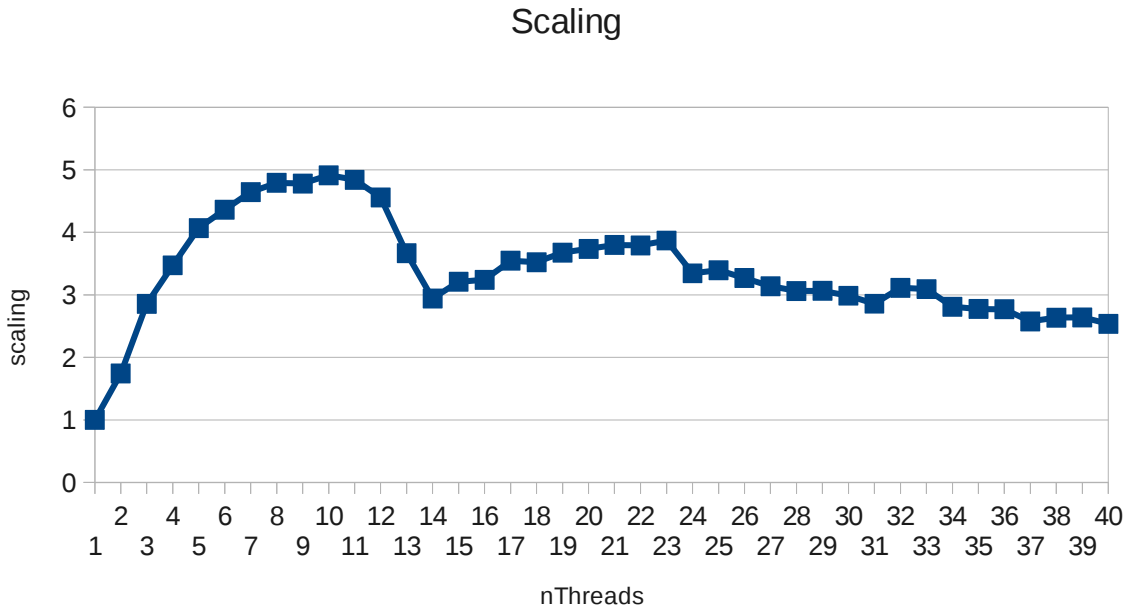


This does not peak at 12x because of the unparallelized portion that runs for each pivot (finding the maximum pivot and swapping the rows) and the number of loop iterations that are parallelized is not constant.

With no partial pivoting, running on 2000 equations produces the following runtimes:



and slightly better scaling:



which peaks closer to 5x instead of 4.5x.

Code for this project can be found on Github at https://github.com/qwerpi/gauss.