Gaussian Elimination

March 12, 2014

Joe Houng

Abstract

OpenMP makes parallelization a one line implementation. Here we find what is possible and what is not in parallelizing a for loop.

Approach

Problem 5.4:

a) Determine whether the outer loop of the row-oriented algorithm can be parallelized

I hypothesis that the outer loop of the row-oriented algorithm cannot be parallelized. This is because the inner loop depends on x[col] where col=row+1. If the outer loop was parallelized for two threads where n = 4, then thread1 would assign x[3] and x[2] and thread2 would would assign x[1] and x[0]. The problem with this is that x[1] is dependent on x[2] (x[col] where col = row+1) and x[2] would not have been assigned yet if thread2 went before thread1.

b. Determine whether the inner loop of the row-oriented algorithm can be parallelized.

The inner loop of the row-oriented algorithm can be parallelized. This is because x[row] - x[n] will have been assigned by the outer loop so that x[col] (where col = row+1) isn't dependent on any of the other threads.

c. Determine whether the (second) outer loop of the column-oriented algorithm can be parallelized

The second outer loop of the column-oriented algorithm can be parallelized. Since x[0..n] have been set in the first outer loop, the only line that has a dependent variable is x[row] -= A[row][col]*x[col] in the inner loop. However since x[col] is always set by the same thread no threads should need to depend on each other.

d. Determine whether the inner loop of the column-oriented algorithm can be parallelized.

The inner loop of the column-oriented algorithm can also be parallelized. This is for the same reason why the second outer loop of the column-oriented algorithm can be parallelized. x[col] is the only thing that the inner loop depends on, and since x[0-n] are already assigned in the first outer loop the inner loop has to be parallelizable.

e. Refer to 5p4_e.c

f. Refer to 5p4_f.c

In this program I timed how long it took for each omp program to calculate the x's 50 times for each type of scheduling; dynamic, guided, and static. They all used a chunk of 2 and used 20 threads. However the program could not handel 10,000 variables so i tested against 1000 variables.

Guided Time:
- Row - 2.632 s
- Column - 2.882 s

Dynamic Time:
- Row - 2.289 s
- Column - 2.328 s

Static TIme:
- Row - 3.304 s
- Column - 3.515 s

Materials

- CF405 lab machine with 16 physical cores

Results

The results indicated that all my hypothesis was incorrect. I could make no sense of the results that I had. In the column oriented solution the only time when the program accurately calculated the x array were when only the inner loop was parallelized. However no matter how i parallelized the loops for the row oriented solution occasionally there would be incorrect x's implying the parallelization failed. Despite these pitfalls I still tested for problem f, dynamic scheduling seemed like the best option for that specific data set.

Run Instructions

Use the makefile by running "make 5p4_e" for the solution for problem e, or "make 5p4_f" for the solution for problem e. When running 5p4_e.c, do "./5p4_e <threadnum> <size>". Before running 5p4_f, first set the environment variable OMP_SCHEDULE by typing "export OMP_SCHEDULE='<type>, <chunk>'". Then run "./5p4_f threadnum".