**CSCE 659 Fall 2017**

**HW 3: Dense Matrix Computations with OpenMP**

**Haiping Xue**

**UIN:122006286**

Problem 1

a.

For this homework, I added #pragma omp parallel to code that can be parallelized in the required section. For each subroutine in the code, I parallelized the for loops inside them. However, some of the for loops can not be parallelized since the inner loop will modify the matrix concurrently if they are parallelized. Therefore, I parallelized only inner loops for such kind of for loop.

b.

|  |  |  |  |
| --- | --- | --- | --- |
| LU |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 2.44E+02 | 1.00E+00 | 1.00E+00 |
| 2 | 1.48E+02 | 1.64E+00 | 8.20E-01 |
| 4 | 8.56E+01 | 2.85E+00 | 7.11E-01 |
| 10 | 8.24E+01 | 2.96E+00 | 2.96E-01 |
| 20 | 8.07E+01 | 3.02E+00 | 1.51E-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| solve\_L |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 6.93E-01 | 1.00E+00 | 1.00E+00 |
| 2 | 4.07E-01 | 1.70E+00 | 8.52E-01 |
| 4 | 2.68E-01 | 2.59E+00 | 6.46E-01 |
| 10 | 2.73E-01 | 2.54E+00 | 2.54E-01 |
| 20 | 2.72E-01 | 2.55E+00 | 1.27E-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| solve\_U |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 6.81E-01 | 1.00E+00 | 1.00E+00 |
| 2 | 4.71E-01 | 1.45E+00 | 7.23E-01 |
| 4 | 2.70E-01 | 2.52E+00 | 6.31E-01 |
| 10 | 2.78E-01 | 2.45E+00 | 2.45E-01 |
| 20 | 2.71E-01 | 2.51E+00 | 1.26E-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| matvec |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 8.43E-02 | 1.00E+00 | 1.00E+00 |
| 2 | 4.34E-02 | 1.94E+00 | 9.71E-01 |
| 4 | 2.22E-02 | 3.80E+00 | 9.49E-01 |
| 10 | 1.59E-02 | 5.31E+00 | 5.31E-01 |
| 20 | 1.58E-02 | 5.32E+00 | 2.66E-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| saxpy |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 1.98E-05 | 1.00E+00 | 1.00E+00 |
| 2 | 1.41E-05 | 1.41E+00 | 7.03E-01 |
| 4 | 1.19E-05 | 1.66E+00 | 4.15E-01 |
| 10 | 9.06E-06 | 2.18E+00 | 2.18E-01 |
| 20 | 1.10E-05 | 1.80E+00 | 9.02E-02 |

|  |  |  |  |
| --- | --- | --- | --- |
| norm |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 6.32E-05 | 1.00E+00 | 1.00E+00 |
| 2 | 3.49E-04 | 1.81E-01 | 9.05E-02 |
| 4 | 1.17E-04 | 5.40E-01 | 1.35E-01 |
| 10 | 3.39E-04 | 1.86E-01 | 1.86E-02 |
| 20 | 1.16E-03 | 5.46E-02 | 2.73E-03 |

c. compilation and execution

compile instruction

icc -qopenmp -o matrix\_just.exe matrix\_just.c

job file

#BSUB -J test #Set the job name to "ExampleJob2"

#BSUB -L /bin/bash #Uses the bash login shell to initialize the job's execution environment.

#BSUB -W 0:10 #Set the wall clock limit to 6hr and 30min

#BSUB -n 20 #Request n cores

#BSUB -R "span[ptile=20]" #Request n cores per node.

#BSUB -R "rusage[mem=2560]" #Request 2560MB per process (CPU) for the job

#BSUB -M 2560 #Set the per process enforceable memory limit to 2560MB.

#BSUB -o testOut.%J #Send stdout and stderr to "Example2Out.[jobID]"

module load intel/2017A

export OMP\_STACKSIZE=16M

./matrix\_just.exe 8192

export OMP\_NUM\_THREADS=20

./matrix\_just.exe 8192

export OMP\_NUM\_THREADS=10

./matrix\_just.exe 8192

export OMP\_NUM\_THREADS=4

./matrix\_just.exe 8192

export OMP\_NUM\_THREADS=2

./matrix\_just.exe 8192

export OMP\_NUM\_THREADS=1

./matrix\_just.exe 8192

2.

a. My strategy is using OpenMP Affinity directives for LU routine since it consumes most computation time in this program. First of all, I set the environment OMP\_PLACES=cores to define a series of places to which the threads are assigned. Set OMP\_PROC\_BIND to true. Then In the parallel section of the LU routine, I used proc\_bind(close) to make threads’ assignment goes successively through the available places. This will bond all thread in a fixed position. Inside the first big for loop, I parallelized the first inner loops with schedule(dynamic, 512). I also tried to parallelize the second for loop, however, it performs worse than the code without binding. Therefore, I chose only to parallel the first inner loop.

I also tried some other strategies: export OMP\_PLACES=sockets, threads, set the schedule to static, guided, spread and with OMP\_PLACES = “{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15}”. But none of them shows significant improvement.

b.

|  |  |  |  |
| --- | --- | --- | --- |
| LU |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 2.44E+02 | 1.00E+00 | 1.00E+00 |
| 2 | 1.06E+02 | 2.31E+00 | 1.15E+00 |
| 4 | 7.94E+01 | 3.07E+00 | 7.67E-01 |
| 10 | 7.78E+01 | 3.13E+00 | 3.13E-01 |
| 20 | 7.68E+01 | 3.17E+00 | 1.59E-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| LU No Strategy |  |  |  |
| cores | Time | Speedup | Efficiency |
| 1 | 2.44E+02 | 1.00E+00 | 1.00E+00 |
| 2 | 1.48E+02 | 1.64E+00 | 8.20E-01 |
| 4 | 8.56E+01 | 2.85E+00 | 7.11E-01 |
| 10 | 8.24E+01 | 2.96E+00 | 2.96E-01 |
| 20 | 8.07E+01 | 3.02E+00 | 1.51E-01 |

Compared to the original code, the strategy is 5 secs faster. It’s 5% faster than the original one for 20 cores.

c. compilation and execution instructions

compile instruction

icc -qopenmp -o matrix.exe matrix.c

job file

#BSUB -J test #Set the job name to "ExampleJob2"

#BSUB -L /bin/bash #Uses the bash login shell to initialize the job's execution environment.

#BSUB -W 0:10 #Set the wall clock limit to 6hr and 30min

#BSUB -n 20 #Request n cores

#BSUB -R "span[ptile=20]" #Request n cores per node.

#BSUB -R "rusage[mem=2560]" #Request 2560MB per process (CPU) for the job

#BSUB -M 2560 #Set the per process enforceable memory limit to 2560MB.

#BSUB -o testOut.%J #Send stdout and stderr to "Example2Out.[jobID]"

module load intel/2017A

export OMP\_PLACES=cores

export OMP\_PROC\_BIND=TRUE

export OMP\_STACKSIZE=16M

export OMP\_NUM\_THREADS=20

./matrix.exe 8192

export OMP\_NUM\_THREADS=10

./matrix.exe 8192

export OMP\_NUM\_THREADS=4

./matrix.exe 8192

export OMP\_NUM\_THREADS=2

./matrix.exe 8192

export OMP\_NUM\_THREADS=1

./matrix.exe 8192