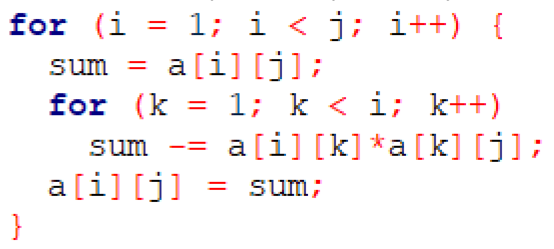
Parallel Progrmg: Sci & Engrg

Machine Problem 5

yhyuan2

A. Q: How does parallelizing this program with OpenMP compare to using pthreads?
A: Using OpenMP can make compiler do parallelization for us. But it will lose precision for improving performance. When using pthreads, we need to split interval manually.

B. Q: Were there any loops in the program that you could not parallelize (e.g. due to size, dependencies, etc.)? If so, pick one and explain why not.
A: There is a loop with dependency in ludcmp.c.


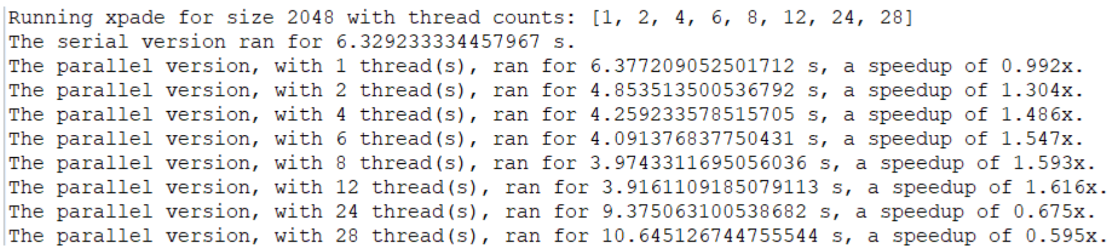
```
for (i = 1; i < j; i++) {
  sum = a[i][j];
  for (k = 1; k < i; k++)
    sum -= a[i][k]*a[k][j];
  a[i][j] = sum;
}
```

There are dependencies between a[i][j] and a[k][j], so I could not parallelize.

C. Q: Were you able to achieve the desired speedup? If not, why not? If so, how could you achieve a higher speedup?
A: I could not achieve the desired speedup. There are two main hotspots in ludcmp.c from vtune result. I could parallelize the most top one. But second one, which is the screenshot in last question, has dependencies. Thus, I could not parallelize second hotspot to achieve the desired speedup.

D. Q: Briefly comment on the parallel scaling of your implementation for data-size 2048, including a copy of the output of the benchmarking program in your report.

A: In my implementation, I set OMP_NUM_THREADS to 4. When there is only one thread for running program, it has an overhead of creating OpenMP threads and switching between four threads. When there are 24 or 28 threads, speedup is lower might because OpenMP threads are switched between different cores.



```
Running xpade for size 2048 with thread counts: [1, 2, 4, 6, 8, 12, 24, 28]
The serial version ran for 6.329233334457967 s.
The parallel version, with 1 thread(s), ran for 6.377209052501712 s, a speedup of 0.992x.
The parallel version, with 2 thread(s), ran for 4.853513500536792 s, a speedup of 1.304x.
The parallel version, with 4 thread(s), ran for 4.259233578515705 s, a speedup of 1.486x.
The parallel version, with 6 thread(s), ran for 4.091376837750431 s, a speedup of 1.547x.
The parallel version, with 8 thread(s), ran for 3.9743311695056036 s, a speedup of 1.593x.
The parallel version, with 12 thread(s), ran for 3.9161109185079113 s, a speedup of 1.616x.
The parallel version, with 24 thread(s), ran for 9.375063100538682 s, a speedup of 0.675x.
The parallel version, with 28 thread(s), ran for 10.645126744755544 s, a speedup of 0.595x.
```