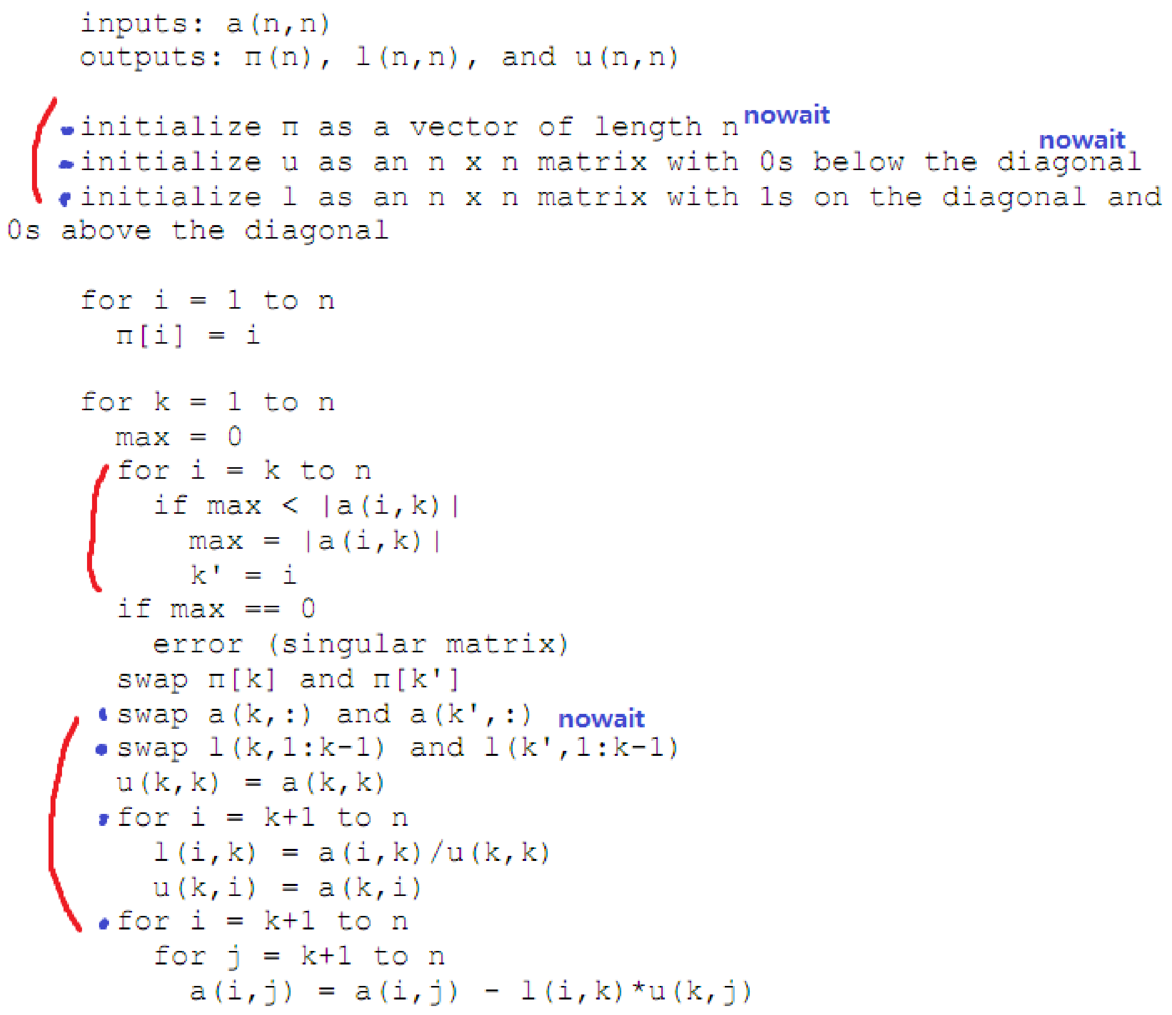
Report for assignment#1

20141440 Seungwon Lee

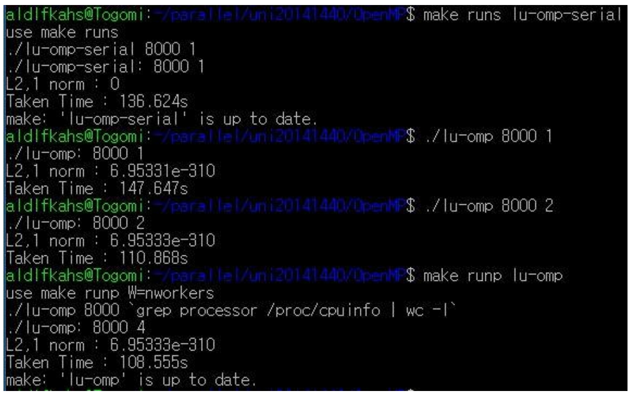
Below figure is my pseudo-code. I implemented the code same as written in handout pdf file. Red parts show where I make parallel. Blue string 'nowait' means that I use 'nowait' option for that parts



```
      inputs: a(n,n)
      outputs: π(n), l(n,n), and u(n,n)

     •initialize π as a vector of length n nowait
                                                              nowait
     •initialize u as an n x n matrix with 0s below the diagonal
     •initialize l as an n x n matrix with 1s on the diagonal and
  0s above the diagonal

      for i = 1 to n
        π[i] = i

      for k = 1 to n
        max = 0
        for i = k to n
          if max < |a(i,k)|
            max = |a(i,k)|
            k' = i
        if max == 0
          error (singular matrix)
        swap π[k] and π[k']
       •swap a(k,:) and a(k',:)  nowait
       •swap l(k,1:k-1) and l(k',1:k-1)
        u(k,k) = a(k,k)
       •for i = k+1 to n
          l(i,k) = a(i,k)/u(k,k)
          u(k,i) = a(k,i)
       •for i = k+1 to n
          for j = k+1 to n
            a(i,j) = a(i,j) - l(i,k)*u(k,j)
```

Figure 1) Pseudo-code

I made parallel the parts where for loops run many times. I merge them to reduce overheads by reusing threads. Also, I use 'nowait' option because initializing and swap itself are independent from other's computing. Thus, thread who complete its own job can progress to next job, which results in speedup. In the end, they are synchronized at the end of each red parts.

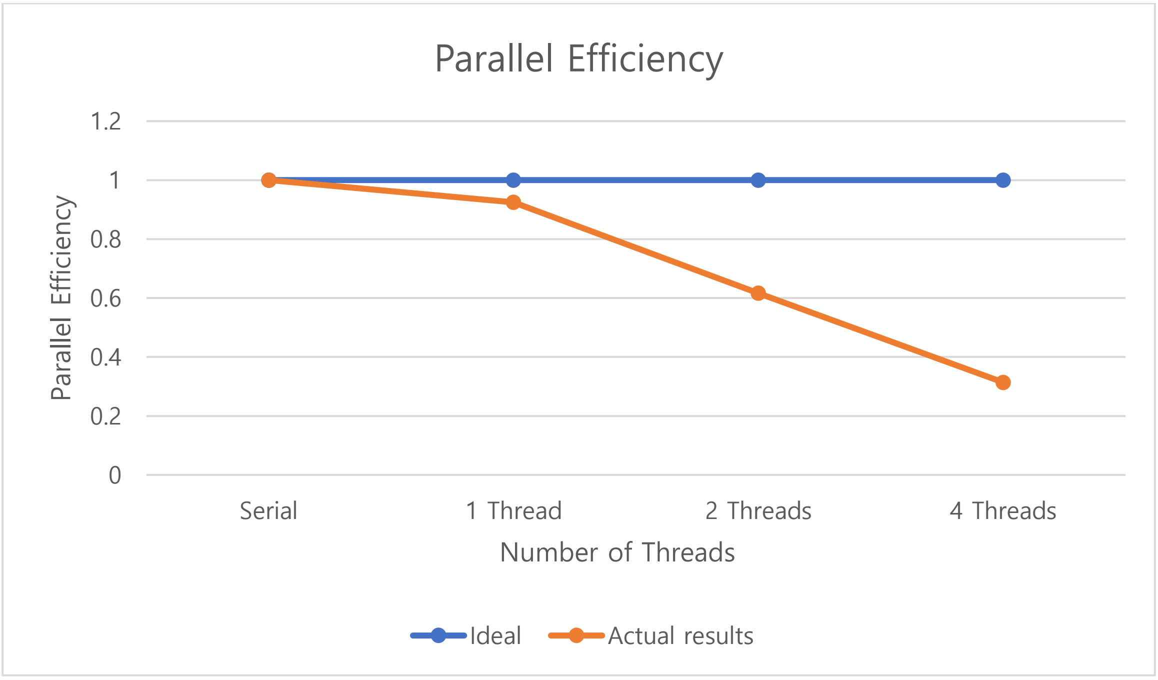


```
aldlfkahs@Togomi:~/parallel/uni20141440/OpenMP$ make runs lu-omp-serial
use make runs
./lu-omp-serial 8000 1
./lu-omp-serial: 8000 1
L2,1 norm : 0
Taken Time : 136.624s
make: 'lu-omp-serial' is up to date.
aldlfkahs@Togomi:~/parallel/uni20141440/OpenMP$ ./lu-omp 8000 1
./lu-omp: 8000 1
L2,1 norm : 6.95331e-310
Taken Time : 147.647s
aldlfkahs@Togomi:~/parallel/uni20141440/OpenMP$ ./lu-omp 8000 2
./lu-omp: 8000 2
L2,1 norm : 6.95333e-310
Taken Time : 110.868s
aldlfkahs@Togomi:~/parallel/uni20141440/OpenMP$ make runp lu-omp
use make runp W=nworkers
./lu-omp 8000 `grep processor /proc/cpuinfo | wc -l`
./lu-omp: 8000 4
L2,1 norm : 6.95333e-310
Taken Time : 108.555s
make: 'lu-omp' is up to date.
```

Figure 2) Measurements on my Ubuntu

	Taken Time(sec)	Parallel efficiency
Serial	136.624	1
1 Thread	147.647	0.925
2 Threads	110.868	0.616
4 Threads	108.555	0.314

Table) Time measurements and Parallel efficiency (n=8000)



Graph) Parallel efficiency

Results

We can figure out that even though more threads make speedup, the efficiency goes down. It means that we cannot anticipate huge performance improvements even if we use more threads. It's because there are some overheads in parallelizing such as initializing, merging results, and so on.