ASSIGNMENT 3 REPORT

Initial output (no -fopenmp flags):
Pcon = 0.0004 Av. Num. Clusters. = 764.0000 Av. Largest Cluster = 4051.0000
Time elapsed: 0.1637 seconds

PARALLELISM OVER VERTICES:
Data Race:
The lines:

```
Ncon[i] = Ncon[i] + 1;
Ncon[j] = Ncon[j] + 1;
Lcon[Maxcon*i+Ncon[i]-1] = j;
Lcon[Maxcon*j+Ncon[j]-1] = i;          /* i is connected to j */
```

in the j indexing loop are the sources of a data race.

Ncon and Lcon are set to use shared memory by default, which can cause elements to be overwritten. The first scenario is if multiple threads have the same value of j for their given i value and try to update the value of Ncon[j] and/or Lcon[Maxcon*j+Ncon[j]-1] at the same time, one will be overwritten. The second scenario is if j in one thread is equal to i in another thread and they both need to update Ncon, then the j thread will be updating the value Ncon[i] in the i thread and vice versa. Eliminate the data race by adding #pragma omp atomic before Ncon lines and #omp critical before Lcon lines.

Results from different number of threads with pragmas in place:
1 thread: Pcon = 0.0004 Av. Num. Clusters. = 764.0000 Av. Largest Cluster = 4051.0000
Time elapsed: 0.1580 seconds (without pragmas 0.1721 seconds)
2 threads: Pcon = 0.0004 Av. Num. Clusters. = 790.0000 Av. Largest Cluster = 4013.0000
Time elapsed: 0.1227 seconds (without pragmas 0.1293 seconds)
3 threads: Pcon = 0.0004 Av. Num. Clusters. = 760.0000 Av. Largest Cluster = 4049.0000
Time elapsed: 0.1088 seconds (without pragmas 0.1235 seconds)
4 threads: Pcon = 0.0004 Av. Num. Clusters. = 775.0000 Av. Largest Cluster = 4017.0000
Time elapsed: 0.1010 seconds (without pragmas 0.0985 seconds)

Performance only seems to barely be slowed by the pragmas on the 1 thread and even less so on the 4 threads, as it serves to partially reserialise the code at certain points, however results appear to remain the same and reproducible with number of threads used.

Optimising scheduling of iterations to threads:
Set number of threads to 4:
Chunk Size: Nvert/1 Runtime: 0.1225 seconds
Chunk Size: Nvert/2 Runtime: 0.1222 seconds
Chunk Size: Nvert/4 Runtime: 0.1152 seconds
Chunk Size: 4 Runtime: 0.0950 seconds
Chunk Size: 2 Runtime: 0.0997 seconds
Chunk Size: 1 Runtime: 0.0948 seconds

Chunk size of Nvert/4 will not be the optimal chunk size as later values of i have fewer values of j to loop through, so the first chunk has significantly more work than the last chunk. In theory, the optimal chunk size will be 1, to minimise the difference in workload between threads.

Set Np=8 and Ngraphs=20 and run on CoW:
Initial (4 threads): Time elapsed: 19.9146 seconds

Cameron Fantham

PARALLELISM OVER P:

OpenMP pragmas:

```
#pragma omp parallel for default(none) shared(Np,Ngraphs)
  private(ip,igraph,i,j,xi,Pcon,avlclus,avnclus,Ncon,Lcon,nclus,lclus) schedule(static,1) ordered

#pragma omp ordered
printf("Pcon = %12.4f Av. Num. Clusters. = %12.4f Av. Largest Cluster = %12.4f\n",
    Pcon,avnclus/(double)Ngraphs,avlclus/(double)Ngraphs);
```

1 thread: Time elapsed: 24.6769 seconds (without ordering 24.6769 seconds)
2 threads: Time elapsed: 14.9461 seconds (without ordering 14.2417 seconds)
3 threads: Time elapsed: 10.4177 seconds (without ordering 10.3133 seconds)
4 threads: Time elapsed: 8.0728 seconds (without ordering 7.5893 seconds)

Overall, parallelism over P is more effective than parallelisation over vertices (8.0728 vs 19.9146 seconds). Including ordering adds between 0.1 and 1 seconds to total run time.

PARALLELISM OVER GRAPHS:

OpenMP pragmas:

```
#pragma omp parallel for default(none) shared(Ngraphs,Pcon) private(igraph,i,j,xi,Ncon,Lcon,nclus,lclus)
  reduction(+:avlclus,avnclus) schedule(dynamic,1)
```

Dynamic vs Static: Using dynamic here is slightly faster than static (1 to 3 seconds faster) and the results obtained are the same for both regardless of the number of threads used.

1 thread: Time elapsed: 25.0515 seconds (28.4448 seconds using static scheduling)
2 threads: Time elapsed: 12.9292 seconds (14.1070 seconds using static scheduling)
3 threads: Time elapsed: 9.5400 seconds (10.2085 seconds using static scheduling)
4 threads: Time elapsed: 7.0728 seconds (7.9268 seconds using static scheduling)

Parallelism over graphs is more effective than parallelisation over P (7.0728 vs 8.0728 seconds), as the overhead for parallelism over graphs relative to the distribution of work is less than that for parallelism over P. However, on 1 thread parallelism over P seems to perform slightly better(24.6769 vs 25.0515 seconds).

NESTED PARALLELISM (P AND GRAPHS):

Environment variables:

```
export OMP_NESTED=TRUE
export OMP_NUM_THREADS=2,2
```

OpenMP pragmas:

```
#pragma omp parallel for default(none) shared(Ngraphs,Np) private(ip,Pcon,avlclus,avnclus) schedule(static,1)
  num_threads(2)

#pragma omp parallel for default(none) shared(Ngraphs,Pcon) private(igraph,i,j,xi,Ncon,Lcon,nclus,lclus)
  reduction(+:avlclus,avnclus) schedule(static,1) num_threads(2)
```

Set Np=2, Ngraphs=20:

2*2 threads: Time elapsed: 1.4760 seconds

The time 1.4760 seconds relatively quick for the work being done here, overall quite similar in performance to parallelism over graphs.