GitHub:

https://github.com/syalexandra/high-performance-computing.git

Question 1

val_test01.cpp:

Line 80: change i<=n to i<n
Line 86 delete [] x doesn’t work for me. Change it to free(x)

val_test02.cpp:

Add

```
 for(i=5;i<10;i++)
  {
     x[i]=0;
  }
```

To the code to initialize the x.

Question 2

The processor : 2.8 GHz Quad-Core Intel Core i7

g++ version: gcc version 9.2.0 (Homebrew GCC 9.2.0_3)

Blocking algorithm:

```
flops = 2*m*n*k*NREPEATS/time/1e9;
bandwidth =2*m*n*(1+k/BLOCK_SIZE)*NREPEATS/time/1e9;
```

Blocking vs OMP:

The second column is the time using blocking algorithm. Gflop/s and GB/s is the flops and bandwidth using blocking algorithm.
reference_time is the time using original naive algorithm.
Openmp_time is the time using openmp on original naive algorithm.

```
 Dimension       Blocking_Time    Blocking_Gflop/s       Blocking_GB/s
Error   Error2      Reference_Time      Openmp_Time
        16   0.119162  16.783900   2.097987 0.000000e+00 0.000000e+00
0.096399   6.644519
        64   0.093256  21.448043   1.675628 0.000000e+00 0.000000e+00
0.070331   0.131960
       112   0.094832  21.096439   1.506888 0.000000e+00 0.000000e+00
0.080903   0.051491
       160   0.087072  23.050349   1.584712 0.000000e+00 0.000000e+00
0.098240   0.042802
       208   0.095319  21.147476   1.423388 0.000000e+00 0.000000e+00
0.106753   0.045052
       256   0.094741  21.250208   1.411147 0.000000e+00 0.000000e+00
0.129880   0.041757
       304   0.095050  21.281446   1.400095 0.000000e+00 0.000000e+00
0.115388   0.041965
       352   0.091684  21.882265   1.429807 0.000000e+00 0.000000e+00
0.125566   0.040262
       400   0.088549  23.128437   1.503348 0.000000e+00 0.000000e+00
0.123788   0.040232
       448   0.097304  22.177602   1.435604 0.000000e+00 0.000000e+00
0.129817   0.042246
       496   0.105791  20.761982   1.339483 0.000000e+00 0.000000e+00
0.121360   0.042475
       544   0.105044  21.456233   1.380456 0.000000e+00 0.000000e+00
0.128279   0.044533
       592   0.097974  21.176505   1.359303 0.000000e+00 0.000000e+00
0.128603   0.041180
       640   0.103933  20.177922   1.292648 0.000000e+00 0.000000e+00
0.115123   0.041059
       688   0.131220  19.854331   1.269754 0.000000e+00 0.000000e+00
0.146387   0.055903
       736   0.115662  20.682070   1.320730 0.000000e+00 0.000000e+00
0.138118   0.050317
       784   0.144548  20.002642   1.275679 0.000000e+00 0.000000e+00
0.194485   0.058117
       832   0.137459  16.759335   1.067602 0.000000e+00 0.000000e+00
0.169156   0.049732
       880   0.188179  14.485612   0.921812 0.000000e+00 0.000000e+00
0.204262   0.062751
```

```
       928   0.268634  11.899890   0.756566 0.000000e+00 0.000000e+00
0.325855   0.079543
       976   0.343296  10.832799   0.688149 0.000000e+00 0.000000e+00
0.407745   0.096295
      1024   0.222418   9.655170   0.612877 0.000000e+00 0.000000e+00
0.232312   0.055896
      1072   0.228524  10.781583   0.683906 0.000000e+00 0.000000e+00
0.278557   0.068299
      1120   0.276610  10.158187   0.643956 0.000000e+00 0.000000e+00
0.368251   0.081330
      1168   0.299219  10.650484   0.674774 0.000000e+00 0.000000e+00
0.409181   0.094238
      1216   0.333779  10.773870   0.682227 0.000000e+00 0.000000e+00
0.485877   0.107742
      1264   0.367711  10.984103   0.695196 0.000000e+00 0.000000e+00
0.525793   0.122288
      1312   0.439550  10.275979   0.650081 0.000000e+00 0.000000e+00
0.614567   0.138051
      1360   0.500737  10.047015   0.635326 0.000000e+00 0.000000e+00
0.685749   0.161820
      1408   0.514981  10.840436   0.685226 0.000000e+00 0.000000e+00
0.789181   0.179695
      1456   0.589908  10.464774   0.661236 0.000000e+00 0.000000e+00
0.841151   0.195005
      1504   0.669686  10.160201   0.641768 0.000000e+00 0.000000e+00
0.911583   0.216724
      1552   0.772322   9.680700   0.611281 0.000000e+00 0.000000e+00
1.001045   0.240776
      1600   0.834857   9.812459   0.619411 0.000000e+00 0.000000e+00
1.199590   0.260447
      1648   0.924994   9.677489   0.610715 0.000000e+00 0.000000e+00
1.375403   0.293995
      1696   1.022926   9.538132   0.601757 0.000000e+00 0.000000e+00
1.544920   0.320631
      1744   1.061784   9.991559   0.630202 0.000000e+00 0.000000e+00
1.688538   0.350548
      1792   1.127471  10.207952   0.643693 0.000000e+00 0.000000e+00
1.794334   0.372553
      1840   1.284328   9.700799   0.611572 0.000000e+00 0.000000e+00
1.993491   0.415280
      1888   1.417682   9.494173   0.598415 0.000000e+00 0.000000e+00
2.177910   0.494974
      1936   1.564172   9.278153   0.584677 0.000000e+00 0.000000e+00
2.362942   0.482470
      1984   1.603886   9.738263   0.613550 0.000000e+00 0.000000e+00
2.431355   0.519642
```

Conclusion:the second column is the time using block_size as 16, the seventh column is the reference time, the eighth column is using openmp to parallel the for loop on the original algorithm. We can see that using blocking and openmp can make the speed faster on large matrix, for small matrix, the original algorithm is faster. And in general using openmp is faster that the blocking.

Question 3:

omp_bug2:
Line 20: add private (tid,total)
Line 37: add reduction(+:total)

omp_bug3:
The code is not running in parallel.
Line 38: add parallel before sections.

omp_bug4:
Use memory rather than stack to store a.
Line 16: change double a[N][N] to double *a.
a=(**double***)malloc(N*N***sizeof**(**double**));
Line 37 a[i*N+j] = tid + i + j;
Line 40:printf("Thread %d done. Last element= %f\n”,tid,a[N*N-1]);

omp_bug5:
Change the location of the locks to avoid the deadlock. For details, please look at the code.

omp_bug6:
Make the variable sum a global variable.
Move #pragma amp parallel into the dotprod function.
Move the tid=omp_get_thread_num() into the for loop.

Question 4:

Jacobi algorithm **with openmp:**

```
g++ -fopenmp -O3 jacobi2D-omp.cpp -o jacobi2D-omp

./jacobi2D-omp -n 100
Jacobian Algorithm: 0.287738

./jacobi2D-omp -n 10000
Jacobian Algorithm: 5.67298
```

Jacobi algorithm **without openmp:**

```
g++ -O3 jacobi2D-omp.cpp -o jacobi2D-omp


./jacobi2D-omp -n 100
Jacobian Algorithm: 0.092099

./jacobi2D-omp -n 1000
Jacobian Algorithm: 14.007
```

Gauss seidel algorithm **with openmp:**

```
g++ -fopenmp -O3 gs2D-omp.cpp -o gs2D-omp

./gs2D-omp -n 100
Jacobian Algorithm: 0.419656

./gs2D-omp -n 1000
Jacobian Algorithm: 15.79
```

Gauss seidel algorithm **without openmp:**

```
./gs2D-omp -n 100
Jacobian Algorithm: 0.105112

 ./gs2D-omp -n 1000
Jacobian Algorithm: 43.0664
```