Assignment 2

Xinyang Wang

1. omp_bug2: by default, variables 'tid' and 'total' are shared among all threads, so the final results are always four same sentences with 'tid' equal to the last thread number that reaches '♯pragma omp barrier', and 'total' equal to the true sum (maybe incorrect due to race condition). To avoid this, use '♯pragma omp parallel private(i,tid,total)' instead.

omp_bug3: '♯pragma omp barrier' can not be used omp sections block or closely nested inside a critical region. Delete it.
omp_bug4: the segmentation fault is caused by the large array size which exceeds the maximum thread stack size limit. One can choose a smaller N or enlarge the stack size limit before complie omp_bug4. To set a large stack size, we can use the command 'limit stacksize unlimited' or 'setenv KMP_STACKSIZE 20000000' for linux system. For other systems, see the './bashrc' files. Use the command 'source ./bashrc' before compiling.
omp_bug5: in eack omp section, the thread acquires 'locka' ('lockb') and then tries to get 'lockb' ('lock1') before releasing 'locka' ('lockb'), which cause deadlock. To solve this, we should release 'locka' before acquire 'lockb'. Execute 'omp_unset_lock(&locka);' before 'omp_set_lock(&lockb);'.

omp_bug6: we get an error message: "Error: reduction variable "sum" must be shared on entry to this OpenMP pragma. " '♯pragma omp for reduction(+:sum)' in the main function is not shared in dotprod. Define 'float sum' before dotprod function, delete 'float sum' in the dotprod and main function.

2. machine: Linux 2.6.32-642.13.1.el6.x86_64 (linax2.cims.nyu.edu) _x86_64(4 CPU)
N=100, max_iters=1000

	Jacobi2D-omp	Gauss-Seidel2D-omp
total iteration number	1000	1000
final residual	53.1112	48.4237
timings		
thread number =2	0.037482s	0.057203s
thread number =4	0.023662 s	0.045726s
thread number =8	0.095894 s	0.164615s

N=1000, max_iters=1000

	Jacobi2D-omp	Gauss-Seidel2D-omp
total iteration number	1000	1000
final residual	953.09	1315.16
timings		
thread number =2	3.650388s	3.109721s
thread number =4	2.363728s	1.639494s
thread number =8	2.910602 s	2.437863 s

(1)Gauss-Seidel method using red-block coloring may need more iterations than sequential Gauss-Seidel method. It may be slower than parallel Jacobi method since it has two for loops.
(2)Using 8 threads will make the program slower. I think this might because my machine only use 4 CPUs. If i want to use 8 threads, some threads on the same CPU have to wait.