HPC Assignment 2 Zixiao Yang, zy1032

All the programs are run on

```
11th Gen Intel(R) Core(TM) i7-1195G7 @ 2.90GHz ~ 2.92 GHz  RAM 32 GB
```

1. Problem 1 Finding Memory Bugs

   (a) We change

   ```
   for ( i = 2; i <= n; i++ )
   ```

   to

   ```
   for ( i = 2; i < n; i++ )
   ```

   because we only alocate n times the size of *int*.

   We also change

   ```
   delete[] x;
   ```

   to

   ```
   free(x)
   ```

   because x is allocated using *malloc()*.

   (b) We change

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

   to

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

   to define all the undefined entries.

2. Optimizing matrix-matrix multiplication
   We repeat the computations for several matrix sizes and we report the runtime as follows.
   We set $Block_size = 32$ as a constant

$MatrixSize$	$TimeforBlocked(-O3)$	$TimeforOpenMP(-O3)$	$TimeforOpenMP(-O2)$
32	0.1796	0.0270	0.0972
224	0.1870	0.0382	0.1614
416	0.1878	0.0550	0.1359
608	0.2090	0.0578	0.1452
800	0.1931	0.0518	0.1249
992	0.3730	0.0945	0.2506
1184	0.3424	0.0861	0.2030

Some errors occur when matrix sizes are greater than 1400. There are *nan* values for larger matrix size. From the table, we can see that the OpenMP is far faster than the Blocked version, because the computation are proceed parallelly with OpenMP. We rerun the whole process with the $-O2$ flag, but we got slower results.

3. Finding OpenMP bugs

   (a) omp2

   We change

   `#pragma omp for schedule(dynamic,10)`

   to

   `#pragma omp for schedule(dynamic,10) reduction(+: total)`

   We include reduction to avoid potential race of multiple threads updating the shared variable total at the same time.

   (b) omp3

   We remove the barrier

   `#pragma omp barrier`

   because two threads will not reach this barrier to avoid the infinite wait.

   (c) omp4

   We shrink N from 1024 to 256 to fulfill the small stack size limit.

   (d) omp5

   We switch the lock/unlock order of *locka* and *lockb* so the two sections share the same lock/unlock order. This is to prevent portential deadlocks.

   (e) omp6

   We make sum a global variable so that sum will be updated if the sum in dotprod is updated. We also include *return sum* to the function *dotprod* to fix a small error.

4. OpenMP version of 2D Jacobi / Gauss-Seidel smoothing
   We write the OpenMP implementations of both methods. Here are the results.

Methods	Matrix size	Residual after 4500 iters	Time elapsed for 5000 iters
Jacobi	100	0.183577	0.494653
Jacobi	200	0.868066	0.796452
Jacobi	300	0.993961	1.395920
GS	100	0.041612 / 2	0.665437
GS	200	1.070824 / 2	0.842405
GS	300	1.805903 / 2	1.500414

   The residuals are devided by 2 in GS, because the initial error was 2 instead of 1 for some reason.
   We can see from the table, that the GS enjoys a higher precision at a cost of slightly more time. One thing to be noticed is that both methods are slow in terms of convergent rate for large matrices.