Spring 2022: HIGH PERFORMANCE COMPUTING
Assignment 2 (due Mar. 23, 2021 *before* class)
Yuan Chen yc5588

1. **Finding Memory bugs.** The homework repository contains two simple programs that contain bugs. Use valgrind to find these bugs and fix them. Add a short comment to the code describing what was wrong and how you fixed the problem. Add the solutions to your repository using the naming convention `val_test01_solved.cpp`, `val_test02_solved.cpp`, and use the Makefile to compile the example problems.

   (a) `val_test01`: two bugs. First, inside the for loop, index in x can go up to n-1, not n. Second, the change the `delete x` to `free x` because the initialization for x is `malloc`.

   (b) `val_test02`: one error. We need to initialize the data, for example, initialize all entries to 0 to avoid the memory leak. When computing with or printing uninitialized data, there would be memory leak.

2. **Optimizing matrix-matrix multiplication.** In this homework you will optimize the matrix-matrix multiplication code from the last homework using blocking. This increases the computational intensity (i.e., the ratio of flops per access to the slow memory) and thus speed up the implementation substantially. The code you can start with, along with further instructions are in the source file `MMult1.cpp`. Specifying what machine you run on, hand in timings for various matrix sizes obtained with the blocked version and the OpenMP version of the code.

   (a) Rearrange loops to maximize performance: $C = A \times B$, where A has size $m \times k$ and B has size $k \times n$. After examining six different order of loops, the best order of loop is `n k m`, as in `MMult0`. This may because in the index calculation, `a[i+p*m];b[p+j*k];c[i+j*m]`, i only involves in addition operation; p involves in one multiplication operation; j involves in 2 multiplication operation, so it's better to iterate i in the most inner loop, then p, then j in the outer loop.

   (b) What is the optimal value for BLOCK_SIZE? After testing with O2, the optimal value for BLOCK_SIZE is 16.

   (c) in figure below, the left is serial version; middle is the block version with size 16; right is the block with openmp version.
   CPU: AMD EPYC Processor (with IBPB)
   number of CPU: 4
   CPU MHz: 2894.562

   (d) performance in GFlops = (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node). The best flop rate is around 14 Gflops/s. The percentage peak FLOP-rate is calculated by $14/(2.94 \times 4 \times 2.5 \times 4) = 11.2\%$

Serial version

Dimension	Time	Gflop/s	GB/s
16	1.018790	1.963116	31.409864
64	0.815170	2.453672	39.258750
112	0.661349	3.025055	48.400884
160	0.647201	3.101110	49.617761
208	0.646270	3.119064	49.905029
256	0.641355	3.139083	50.225334
304	0.641958	3.150985	50.415767
352	0.637373	3.147692	50.363070
400	0.646834	3.166192	50.659066
448	0.678163	3.182080	50.913283
496	0.685512	3.204075	51.265203
544	0.709198	3.178026	50.848419
592	0.639143	3.246141	51.938259
640	0.648384	3.234429	51.750862
688	0.805724	3.233471	51.735537
736	0.754293	3.171354	50.741666
784	0.908253	3.183411	50.934575
832	0.720274	3.198398	51.174371
880	0.856982	3.180798	50.892774
928	1.006045	3.177507	50.840104
976	1.190292	3.124322	49.989153
1024	0.698590	3.074024	49.184388
1072	0.833117	2.957389	47.318232
1120	0.993017	2.829614	45.273821
1168	1.178800	2.703451	43.255210
1216	1.383537	2.599201	41.587218
1264	1.682009	2.401281	38.420494
1312	1.942391	2.325385	37.206160
1360	2.217216	2.269022	36.304352
1408	2.252451	2.478464	39.655429
1456	3.555639	1.736187	27.778992
1504	3.347146	2.032820	32.525115
1552	3.516267	2.126294	34.020703
1600	3.546185	2.310088	36.961408
1648	5.593869	1.600256	25.604089
1696	5.196778	1.877471	30.039543
1744	5.891884	1.800592	28.809467
1792	8.420374	1.366824	21.869185
1840	6.932520	1.797183	28.754930
1888	7.424532	1.812871	29.005935
1936	8.046973	1.803489	28.855827
1984	9.070931	1.721881	27.550095

Block version

Dimension	Time	Gflop/s	GB/s	Error
16	0.535403	3.735510	59.768160	0.000000e+00
64	0.537634	3.720300	59.524807	0.000000e+00
112	0.554151	3.610238	57.763809	0.000000e+00
160	0.546533	3.672314	58.757028	0.000000e+00
208	0.552857	3.646072	58.337150	0.000000e+00
256	0.559090	3.600968	57.615488	0.000000e+00
304	0.552698	3.659865	58.557845	0.000000e+00
352	0.545871	3.675323	58.805164	0.000000e+00
400	0.558328	3.668094	58.689508	0.000000e+00
448	0.593262	3.637466	58.199450	0.000000e+00
496	0.593643	3.699921	59.198728	0.000000e+00
544	0.612031	3.682571	58.921129	0.000000e+00
592	0.565746	3.667277	58.676437	0.000000e+00
640	0.578368	3.625983	58.015722	0.000000e+00
688	0.714695	3.645308	58.324932	0.000000e+00
736	0.660722	3.620477	57.927637	0.000000e+00
784	0.794089	3.641079	58.257266	0.000000e+00
832	0.636817	3.617555	57.880877	0.000000e+00
880	0.789620	3.452150	55.234403	0.000000e+00
928	0.895276	3.570648	57.130367	0.000000e+00
976	1.048107	3.548166	56.770655	0.000000e+00
1024	0.597578	3.593648	57.498364	0.000000e+00
1072	0.707056	3.484659	55.754541	0.000000e+00
1120	0.802888	3.499686	55.994973	0.000000e+00
1168	0.924330	3.447717	55.163474	0.000000e+00
1216	1.031087	3.487671	55.802730	0.000000e+00
1264	1.203102	3.357134	53.714141	0.000000e+00
1312	1.286944	3.509714	56.155425	0.000000e+00
1360	1.431202	3.515166	56.242650	0.000000e+00
1408	1.596629	3.496504	55.944057	0.000000e+00
1456	1.778272	3.471490	55.543834	0.000000e+00
1504	1.956118	3.478391	55.654256	0.000000e+00
1552	2.232667	3.348738	53.579808	0.000000e+00
1600	2.507785	3.266627	52.266038	0.000000e+00
1648	2.587023	3.460201	55.363209	0.000000e+00
1696	2.822014	3.457390	55.318238	0.000000e+00
1744	3.196102	3.319317	53.109075	0.000000e+00
1792	3.284595	3.503985	56.063754	0.000000e+00
1840	3.706336	3.361543	53.784696	0.000000e+00
1888	3.926327	3.428069	54.849098	0.000000e+00
1936	4.164185	3.485106	55.761696	0.000000e+00
1984	4.481575	3.485173	55.762773	0.000000e+00

Openmp version

Dimension	Time	Gflop/s	GB/s	Error
16	0.967893	2.066348	33.061572	0.000000e+00
64	0.147299	13.578913	217.262608	0.000000e+00
112	0.232874	8.590979	137.455672	6.135658e+01
160	0.141388	14.195259	227.124147	3.092282e-11
208	0.152257	13.239210	211.827368	1.455192e-11
256	0.259148	7.768789	124.300621	0.000000e+00
304	0.156573	12.919216	206.707457	7.730705e-12
352	0.213958	9.376858	150.029720	1.091394e-11
400	0.145821	14.044606	224.713689	4.774847e-12
448	0.158011	13.657064	218.513025	0.000000e+00
496	0.319535	6.873835	109.981356	5.229595e-12
544	0.158085	14.257205	228.115279	2.955858e-12
592	0.146450	14.166946	226.671135	3.183231e-12
640	0.207553	10.104187	161.666997	0.000000e+00
688	0.198461	13.127421	210.038728	2.728484e-12
736	0.222777	10.737765	171.804235	3.296918e-12
784	0.261077	11.074658	177.194530	3.751666e-12
832	0.200668	11.480273	183.684372	0.000000e+00
880	0.245926	11.084184	177.346947	1.705303e-12
928	0.225207	14.194536	227.112569	1.989520e-12
976	0.270321	13.757208	220.115331	1.364242e-12
1024	0.198765	10.804155	172.866486	0.000000e+00
1072	0.202502	12.167018	194.672283	1.648459e-12
1120	0.216885	12.955491	207.287854	1.762146e-12
1168	0.299824	10.628997	170.063953	1.762146e-12
1216	0.433095	8.303231	132.851704	0.000000e+00
1264	0.299770	13.473577	215.577230	1.989520e-12
1312	0.327455	13.793673	220.698767	2.103206e-12
1360	0.525171	9.579575	153.273201	2.160050e-12
1408	0.463804	12.036579	192.585267	0.000000e+00
1456	0.604095	10.219018	163.504292	2.501110e-12
1504	0.510360	13.332051	213.312822	3.012701e-12
1552	0.745958	10.022836	160.365384	2.785328e-12
1600	0.900271	9.099485	145.591755	0.000000e+00
1648	0.683545	13.095868	209.533892	3.353762e-12
1696	0.755130	12.920696	206.731131	2.899014e-12
1744	1.211847	8.754306	140.068893	2.842171e-12
1792	0.932389	12.343749	197.499982	0.000000e+00
1840	1.164900	10.695348	171.125561	3.069545e-12
1888	1.105940	12.170389	194.726229	3.467449e-12
1936	1.248239	11.626482	186.023715	3.183231e-12
1984	1.312199	11.902973	190.447569	0.000000e+00

Figure 1: Test for serial / block / omp version under flag O2

3. **Finding OpenMP bugs.** The homework repository contains five OpenMP problems that contain bugs. These files are in C, but they can be compiled with the C++ compiler. Try to find these bugs and fix them. Add a short comment to the code describing what was wrong and how you fixed the problem. Add the solutions to your repository using the naming convention `omp_solved`{2,...}`.c`, and provide a Makefile to compile the fixed example problems.

   Please see the code and comments in code.

4. **OpenMP version of 2D Jacobi/Gauss-Seidel smoothing.** I fix the iteration number to be 5000, and iterate from threads 1 to 10; N = 50 to 300. As seen in figure below, as N increases, timings increase. The optimal timing occurs at thread = 3 or 4.

   CPU: AMD EPYC Processor (with IBPB)
   number of CPU: 4
   CPU MHz: 2894.562

N	50		100		150		200		250		300	
thread	jacobi	gs	jacobi	gs	jacobi	gs	jacobi	gs	jacobi	gs	jacobi	gs
1	0.084725	0.068378	0.349482	0.349127	0.810961	0.785642	1.389841	1.405749	2.171193	2.205424	3.128233	3.374094
2	0.085739	0.088428	0.299159	0.336785	0.742602	0.619419	1.095202	1.232787	2.063873	1.875955	2.552198	2.567
3	0.062559	0.055769	0.27552	0.275873	0.932166	0.572242	1.032567	1.070695	1.94884	1.747235	2.576264	2.474203
4	0.224878	0.122284	0.433426	0.359188	0.672346	0.650108	1.185679	1.333181	1.690706	1.955268	2.589801	2.371411
5	0.498506	0.515009	0.693465	1.070897	0.992534	1.476711	1.612417	1.976394	2.161788	2.997712	3.299314	3.613674
6	0.453125	0.623714	0.743035	1.151335	1.316548	1.620289	1.640703	2.302111	2.212288	2.989664	3.170332	3.646692
7	0.525105	0.698965	0.875853	1.380369	1.328335	1.860047	1.606023	2.63148	2.801961	3.692752	3.529421	4.567578
8	0.513877	0.882859	0.843781	1.742843	1.29904	2.066668	1.787273	2.436708	2.859502	3.747574	3.238008	4.176138
9	0.674566	0.948136	1.027399	1.592374	1.287312	1.826041	1.687151	2.386017	2.330524	3.036826	3.169389	3.817183
10	0.647488	0.935064	0.927522	1.61362	1.29732	2.060228	1.739004	2.351671	2.419419	3.442687	3.026813	3.805112

Figure 2: omp test for different threads and N

Sample running code (in 4.sh): `make` // compile for all files

```
for i in {1..10..1} // iterate for diffrerent threads
    do
        export OMP_NUM_THREADS=$i
        echo "Number of threads $i"
        for n in {50..300..50} // iterate for diffrerent N
        do
            echo "N = $n"
            ./jacobi2D-omp $n
            ./gs2D-omp $n
        done
    done
```