HW2

Antony Sunwoo

March 2020

1 Finding memory bugs

See val test01_solved.cpp, val test02_solved.cpp and comments within.

2 Optimizing matrix-matrix multiplication

The machine I used had:
- Processor: Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz 3.30GHz
- g++: gcc version 9.2.0 (MinGW.org GCC Build-20200227-1)

The given order of loops in the reference function is actually the ideal loop order (in order of j,p,i from outer to inner). This is because in 3 of the 4 memory access happening inside the loops, the index is unit incremented by i, and the remaining access' index is unit incremented by p. So by having i in the innermost loop, and p in the next outer loop it decreases the amount of cache misses dramatically.

After testing with different BLOCK_SIZE values, the optimal value I found was 64.

My timings after introducing blocking were:

No OpenMP

```
$ g++ -O3 --std=c++11 -march=native MMult1.cpp && ./a.exe
 Dimension       Time    Gflop/s       GB/s        Error
        64   0.213462   9.370093 149.921483 0.000000e+000
       128   0.202429   9.883381 158.134102 0.000000e+000
       192   0.195477  10.283154 164.530472 0.000000e+000
       256   0.183477  10.972852 175.565628 0.000000e+000
       320   0.186530  10.891631 174.266102 0.000000e+000
       384   0.184507  11.047991 176.767862 0.000000e+000
       448   0.213458  10.109574 161.753181 0.000000e+000
       512   0.209440  10.253455 164.055283 0.000000e+000
       576   0.230411   9.952805 159.244877 0.000000e+000
       640   0.210406   9.967168 159.474692 0.000000e+000
       704   0.210441   9.948071 159.169134 0.000000e+000
       768   0.275290   9.872894 157.966304 0.000000e+000
       832   0.235339   9.788949 156.623184 0.000000e+000
       896   0.289227   9.948216 159.171449 0.000000e+000
       960   0.358040   9.884214 158.147425 0.000000e+000
      1024   0.217395   9.878257 158.052110 0.000000e+000
      1088   0.259310   9.933388 158.934214 0.000000e+000
      1152   0.309172   9.889795 158.236716 0.000000e+000
      1216   0.357045  10.071816 161.149049 0.000000e+000
      1280   0.414876  10.109777 161.756438 0.000000e+000
      1344   0.000000        nan        nan 0.000000e+000
      1408   0.000000        nan        nan 0.000000e+000
      1472   0.000000        nan        nan 0.000000e+000
      1536   0.000000        nan        nan 0.000000e+000
      1600   0.000000       -inf       -inf 0.000000e+000
      1664   3.672207  10.037436 160.598980 0.000000e+000
      1728   2.106367   9.798445 156.775122 0.000000e+000
      1792   1.158893   9.931176 158.898814 0.000000e+000
      1856   1.282599   9.969492 159.511867 0.000000e+000
      1920   0.000000        nan        nan 0.000000e+000
      1984   0.000000        nan        nan 0.000000e+000
```

With OpenMP:

```
$ g++ -O3 -fopenmp --std=c++11 -march=native MMult1.cpp && ./a.exe
 Dimension       Time    Gflop/s       GB/s        Error
        64   0.433808   4.610700  73.771206 0.000000e+000
       128   0.152561  13.113987 209.823796 0.000000e+000
       192   0.085778  23.433983 374.943728 0.000000e+000
       256   0.068787  29.268116 468.289862 0.000000e+000
       320   0.079791  25.461719 407.387500 0.000000e+000
       384   0.065825  30.967440 495.479041 0.000000e+000
       448   0.063830  33.808075 540.929195 0.000000e+000
       512   0.066820  32.138337 514.213385 0.000000e+000
       576   0.079760  28.751702 460.027224 0.000000e+000
       640   0.068816  30.474773 487.596373 0.000000e+000
       704   0.061834  33.856486 541.703783 0.000000e+000
       768   0.077791  34.938605 559.017674 0.000000e+000
       832   0.075800  30.392104 486.273662 0.000000e+000
       896   0.083746  34.357373 549.717965 0.000000e+000
       960   0.100733  35.131923 562.110768 0.000000e+000
      1024   0.057810  37.147269 594.356312 0.000000e+000
      1088   0.079758  32.295531 516.728492 0.000000e+000
      1152   0.092723  32.976151 527.618410 0.000000e+000
      1216   0.097711  36.803342 588.853479 0.000000e+000
      1280   0.111673  37.558801 600.940818 0.000000e+000
      1344   0.000000        nan        nan 0.000000e+000
      1408   0.000000        nan        nan 0.000000e+000
```

```
1472   0.000000         nan        nan 0.000000e+000
1536   0.000000         nan        nan 0.000000e+000
1600   0.000000        -inf       -inf 0.000000e+000
1664   1.145908   32.166233 514.659726 0.000000e+000
1728   0.616319   33.487725 535.803606 0.000000e+000
1792   0.308175   37.346216 597.539459 0.000000e+000
1856   0.373002   34.280942 548.495076 0.000000e+000
1920   0.000000         nan        nan 0.000000e+000
1984   0.000000         nan        nan 0.000000e+000
```

Comparing the highest FLOP-rate from both runs, I achieved

$$\frac{37.559}{11.048} * 100 = 339.962\%$$

of my peak FLOP-rate by parallelizing the blocked code.

Note: I am not sure why for some matrix dimensions the timing is 0. I just assumed it was an unreasonable bug since it also happened even using the given MMult0 code as substitute for MMult1.

3 Finding OpenMP bugs

See omp_solved{2,. . . }.c and comments within.

4 OpenMP version of 2D Jacobi/Gauss-Seidel smoothing

The timings below are compiled with the same computer, for 1000 iterations.

The biggest N I used was 5000, since using 10000 gave segmentation fault probably due to memory limitations.

4.1 Jacobi

```
$ g++ -O3 -fopenmp --std=c++11 -march=native jacobi2D-omp.cpp && a.exe -n (N) -t (nu
```

N\T	2	4	8	16	32
100	0.94	0.97	1.05	1.13	1.39
1000	6.70	4.90	4.93	4.92	5.19
5000	141.58	98.79	105.18	95.23	95.07

4.2 Gauss-Seidel

```
$ g++ -O3 -fopenmp --std=c++11 -march=native gs2D-omp.cpp && a.exe -n (N) -t (number
```

N\\T	2	4	8	16	32
100	0.97	1.01	1.09	1.27	1.60
1000	7.04	5.77	5.73	5.82	5.98
5000	151.79	118.77	117.72	116.21	116.10