Homework 2

Cong Yu

March 14, 2020

github:

```
https://github.com/heliumyc/hpc_homework
```

1 Finding Memory bugs

For val_test01, there are two errors:

1. delete is used which mismatches malloc and it should be free
2. it allocates int array of size n but tris to read and write to position n+1 (index n).

For val_test02, the program tries to use uninitialized data in a newed array.

2 Optimizing matrix-matrix multiplication

Machine information:

```
Architecture:  x86_64
Processor Name: 6-Core Intel Core i7
Processor Speed: 2.2 GHz
Number of processors: 1
Total number of cores: 6
L1 cache: 64 KB
L2 cache (per core): 256 KB
L3 cache: 9 MB
Hyper-Threading Technology: Enabled
Memory: 16 GB
```

- In MMult1, the best order is supposed to be j-p-i. Because the CPU fetches data via cache line, the memory access time is reduced if we read array data sequentially.

- The best BLOCK SIZE should be 48 or 64 and both reach the roughly same optimal time

(experiment ranges from 4 to 72)

- FLOP-percentage is at peak :

For omp multi-thread blocking mmult: $41.548/(2.2 * 6 * 4) = 78.69\%$

Time for blocking

Dimension	Time	Gflop/s	GB/s	Error
64	2.798738	0.714665	11.434634	0.000000e+00
128	3.079668	0.649642	10.394280	0.000000e+00
192	2.949185	0.681585	10.905358	0.000000e+00
256	2.973606	0.677045	10.832723	0.000000e+00
320	2.950382	0.688594	11.017506	0.000000e+00
384	3.166501	0.643749	10.299982	0.000000e+00
448	3.071128	0.702664	11.242616	0.000000e+00
512	3.168179	0.677829	10.845265	0.000000e+00
576	3.264838	0.702404	11.238467	0.000000e+00
640	3.256561	0.643977	10.303640	0.000000e+00
704	2.966741	0.705651	11.290408	0.000000e+00
768	4.082646	0.665722	10.651559	0.000000e+00
832	3.571540	0.645022	10.320349	0.000000e+00
896	4.580854	0.628113	10.049804	0.000000e+00
960	5.058059	0.699664	11.194631	0.000000e+00
1024	3.236734	0.663472	10.615557	0.000000e+00
1088	3.696069	0.696910	11.150558	0.000000e+00
1152	4.421231	0.691583	11.065326	0.000000e+00
1216	5.223907	0.688391	11.014259	0.000000e+00

1280	6.153863	0.681573	10.905161	0.000000e+00
1344	7.446900	0.652007	10.432112	0.000000e+00
1408	8.611343	0.648287	10.372586	0.000000e+00
1472	9.252391	0.689445	11.031116	0.000000e+00
1536	10.814957	0.670161	10.722569	0.000000e+00
1600	12.027636	0.681098	10.897570	0.000000e+00
1664	13.908687	0.662527	10.600438	0.000000e+00
1728	15.072107	0.684679	10.954870	0.000000e+00
1792	17.318526	0.664558	10.632933	0.000000e+00
1856	18.964697	0.674245	10.787926	0.000000e+00
1920	20.801535	0.680516	10.888255	0.000000e+00

Time for OMP

Dimension	Time	Gflop/s	GB/s	Error
64	0.197831	10.110429	161.766858	0.000000e+00
128	0.135221	14.795627	236.730035	0.000000e+00
192	0.081309	24.721884	395.550143	0.000000e+00
256	0.065687	30.649435	490.390954	0.000000e+00
320	0.051685	39.307974	628.927581	0.000000e+00
384	0.047805	42.640173	682.242776	0.000000e+00
448	0.076880	28.069359	449.109746	0.000000e+00
512	0.087533	24.533544	392.536701	0.000000e+00
576	0.066770	34.345394	549.526305	0.000000e+00
640	0.059702	35.126916	562.030650	0.000000e+00
704	0.051548	40.612501	649.800013	0.000000e+00
768	0.065416	41.548239	664.771820	0.000000e+00
832	0.064715	35.597805	569.564884	0.000000e+00
896	0.085093	33.813370	541.013918	0.000000e+00
960	0.094435	37.475088	599.601400	0.000000e+00
1024	0.074737	28.733930	459.742887	0.000000e+00
1088	0.068885	37.392930	598.286881	0.000000e+00
1152	0.077152	39.631471	634.103530	0.000000e+00
1216	0.101429	35.454152	567.266434	0.000000e+00
1280	0.126682	33.108864	529.741816	0.000000e+00
1344	0.126867	38.271791	612.348660	0.000000e+00
1408	0.157027	35.552049	568.832784	0.000000e+00
1472	0.149335	42.716242	683.459876	0.000000e+00
1536	0.212875	34.046995	544.751919	0.000000e+00
1600	0.212644	38.524561	616.392977	0.000000e+00
1664	0.249281	36.965858	591.453727	0.000000e+00
1728	0.259865	39.711203	635.379243	0.000000e+00
1792	0.309971	37.129803	594.076855	0.000000e+00
1856	0.335884	38.069221	609.107541	0.000000e+00
1920	0.348047	40.672060	650.752955	0.000000e+00
1984	0.610232	25.595270	409.524321	0.000000e+00

3 OMP bug

2. add critical region and to prevent race condition

3. two solutions: limit the thread to 2 or remove the barrier in the print result function

4. use 'ulimit -s unlimited' before run, or else stac overflow

5. dead lock, adjust the lock&unlock statement

6. the variable sum in the function overwrites the shared variable sum. making sum global variable and deleting the local sum solves this.

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

Machine information:

```
Architecture:  x86_64
Processor Name: 6-Core Intel Core i7
Processor Speed: 2.2 GHz
Number of processors: 1
Total number of cores: 6
L1 cache: 64 KB
L2 cache (per core): 256 KB
L3 cache: 9 MB
Hyper-Threading Technology: Enabled
Memory: 16 GB
```

Iterations needed to converge

Iterations	Jacobi	Gauss-Seidel
N=10	332	170
N=50	7177	3680
N=100	28141	14429
N=150	62890	32246
N=200	111424	57131
N=250	173743	89084

Jacobi timing

N \ Threads	1	2	4	6	8
10	0.000222	0.000497	0.000548	0.000697	0.001276
50	0.059832	0.052627	0.052681	0.052620	0.064647
100	0.738714	0.610693	0.560673	0.562656	0.696234
150	3.654725	3.126153	2.664340	2.931137	3.437034
200	11.489494	9.593579	8.195153	9.401875	11.897939
250	27.514397	23.024206	20.766778	23.430747	31.071356

Gauss-Seidel timing

N \ Threads	1	2	4	6	8
10	0.000152	0.000397	0.000571	0.000619	0.000720
50	0.036064	0.029779	0.026489	0.024384	0.029910
100	0.404068	0.342580	0.308409	0.320770	0.379683
150	2.035859	1.645190	1.493869	1.594564	1.927190
200	6.336907	5.179915	4.765368	5.308991	6.491525
250	15.211912	12.523597	11.787166	12.534422	15.608009