Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 2

Name: Shih-Ting(Emily) Huang
NetID: sth351

2. Optimizing matrix-matrix multiplication.

I run this program on CIMS crunchy6 server(hostname: crunchy6.cims.nyu.edu, CPU: Four AMD Opteron 6272 (2.1 GHz) (64 cores), Memory: 256 GB, OS: CentOS 7).

In order to accelerate the speed, I reduce the NREPEATS value from 1e9/(m*n*k)+1 to 10, so the following result is NREPEATS = 10, and I set block_size=32.

The following are table 1 showing the result of the blocked version and table 2 blocking+OpenMP version:

Table 1: the result of blocked version with block size = 32

Dimension	Time	Gflop/s	GB/s
32	0.000216	3.027556	48.440903
160	0.019265	4.252306	68.036897
480	0.555931	3.978621	63.657939
800	2.640992	3.877331	62.037292
1600	23.383862	3.503271	56.052332
1984	48.477844	3.221897	51.550358

Table 2: the result of OpenMP+ blocked version with block size = 32

Dimension	Time	Gflop/s	GB/s
32	0.001104	0.593773	9.500372
160	0.008081	10.137477	162.199635
480	0.148041	14.940745	239.051919
800	0.778134	13.159682	210.554905
1600	6.630963	12.354163	197.666602
1984	12.192199	12.810703	204.971251

According to the above results, we can find that openMP+blocking version is faster than blocking version except when dimension is small such as 32. Thus, we can know that when the problem is not big enough, it's not a good idea to parallelize it, because it can not overcome the overhead. I also experiment with different block size = 16, 28, 32, 36, 48 and we can find that from 16-32, the

speedup is more and more, but when block size = 48, the performance drops. We know that the L1 cache size is 32KB, and when block size is 48, the block matrix size A, B, C total is 48*48*8(double)*3 = 55296 that can't be fitted in L1 cache, so the performance is worse than block size = 32 or 36.

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

I run this program on CIMS crunchy6 server(hostname: crunchy6.cims.nyu.edu, CPU: Four AMD Opteron 6272 (2.1 GHz) (64 cores), Memory: 256 GB, OS: CentOS 7).

To make things easy, I set the maximum iteration = 5000 on both version, and in order to make the result more precise, I will repeat each experiment 5 times and record the median.

The following are table 3 showing the result of Jacobi OpenMP version and table 4 Gauss-Seidel OpenMP version:

Table 3: Jacobi OpenMP version (in seconds)

Dimension(N)	Thread_Num = 1	Thread_Num = 2	Thread_Num = 4	Thread_Num = 8
24	0.035169 s	0.044043 s	0.042806 s	0.045589 s
48	0.115563 s	0.135396 s	0.125206 s	0.103938 s
120	0.676759 s	0.502812 s	0.487143 s	0.546592 s
240	2.690035 s	2.455830 s	2.037243 s	2.368499 s
480	13.071230 s	8.208181 s	6.478027 s	6.024919 s

Table 4: Gauss Seidel OpenMP version (in seconds)

Dimension(N)	Thread_Num = 1	Thread_Num = 2	Thread_Num = 4	Thread_Num = 8
24	0.066843 s	0.063301 s	0.066902 s	0.077683 s
48	0.220008 s	0.143494 s	0.107263 s	0.107979 s
120	1.283211 s	0.678704 s	0.395383 s	0.257462 s
240	5.049566 s	2.574112 s	1.369727 s	0.769348 s
480	21.352834 s	10.660158 s	5.369157 s	2.814045 s

From the above result, we can find that when the dimension is small(N=24, 48), increasing the thread number will not improve the performance. When N>=120, in both methods, we can see the improvement of performance when increasing thread number. And we can also find that when the N is larger, the more speedup.