HPC Homework II

Zihan Zhang

March 2022

Processor Information: AMD Opteron(TM) Processor 6272 with Bulldozer micro-architecture (64 CPUs/cores). The cloud speed is 2.1GHz, and it has 8 flops per cycle. The total peak FLOP-rate is:

$$2.1 \times 10^9 \times 8 \times 64 \approx 1075 \text{ GFLOP/s}$$

Problem 2. Our goal is to make cache hits more and flop rate bigger to make it faster, so for the order of for loops, we should read the data in the sequential order as much as possible to lower the cost. For the matrix multiplication $A \times B = C$, we should loop the rows i of A first, then rows p of B, then columns j of C, so $i - p - j$ could be the sequence of for loop from inner to outer.

To prove the result experimentally, I use the brute-force method to go through all possible arrangements and generate the following results. The BLOCK_SIZE is 16, as the default setting:

- $i - p - j$

Dimension	Time	Gflop/s	GB/s	Error
256	0.227765	8.839241	70.990152	0.000000e+00
1024	0.278304	7.716335	61.790961	0.000000e+00
1504	1.424254	4.777337	38.244111	0.000000e+00

- $p - j - i$

Dimension	Time	Gflop/s	GB/s	Error
256	2.928394	0.687498	5.521471	0.000000e+00
1024	9.669836	0.222081	1.778380	0.000000e+00
1504	6.605722	1.030038	8.245783	0.000000e+00

- $j - i - p$

Dimension	Time	Gflop/s	GB/s	Error
256	16.962598	0.118689	0.953217	0.000000e+00
1024	19.509060	0.110076	0.881470	0.000000e+00
1504	83.128173	0.081851	0.655245	0.000000e+00

- $i-j-p$

Dimension	Time	Gflop/s	GB/s	Error
256	0.252106	7.985777	64.135773	0.000000e+00
1024	0.308489	6.961303	55.744810	0.000000e+00
1504	2.206201	3.084100	24.689202	0.000000e+00

- $p-j-i$

Dimension	Time	Gflop/s	GB/s	Error
256	2.992975	0.672664	5.402331	0.000000e+00
1024	8.327003	0.257894	2.065166	0.000000e+00
1504	5.575389	1.220389	9.769604	0.000000e+00

- $j-p-i$

Dimension	Time	Gflop/s	GB/s	Error
256	16.864483	0.119379	0.958763	0.000000e+00
1024	23.434846	0.091636	0.733807	0.000000e+00
1504	48.096059	0.141470	1.132512	0.000000e+00

Clearly, $i-p-j$ is the most time-efficient order, which is consistent with the prediction.

Block: I choose the dimension to be 2048 and obtain the following result for comparison, by setting up the BLOCK_SIZE as the divisor of 2048:

BLOCK_SIZE	Time	Gflop/s	GB/s	Error
4	4.763411	3.713330	29.721002	0.000000e+00
16	5.895390	2.846352	22.782019	0.000000e+00
64	3.347289	5.132472	41.079828	0.000000e+00
128	3.203082	5.363543	42.929295	0.000000e+00
256	2.320336	7.404043	59.261267	0.000000e+00
512	2.452787	7.004225	56.061158	0.000000e+00
1024	3.116361	5.512798	44.123922	0.000000e+00
2048	6.743058	2.547786	20.392240	0.000000e+00

The experimental result indicates the optimal block size is 256.

OpenMP: I implement OpenMP on the raw version of the code, set up 4 threads, use the optimal block size of 256, and get this result:

Dimension	Time	Gflop/s	GB/s	Error
256	0.088289	22.803098	183.137384	0.000000e+00
512	0.087452	24.556160	196.832967	0.000000e+00
768	0.116493	23.331170	186.892392	0.000000e+00
1024	0.132251	16.237980	130.030700	0.000000e+00
1280	0.216698	19.355523	154.965153	0.000000e+00
1536	0.583540	12.420322	99.427266	0.000000e+00
1792	1.322782	8.700734	69.644714	0.000000e+00
2048	2.271514	7.563180	60.534981	0.000000e+00

The flop-rate increases until it reaches $N = 512$, as the bottleneck of the memory. The flop-rate in the OpenMP version is far less than the theoretical optimal rate deduced from the processor's information, so 0% (achieves the peak flop-rate).
Similarly, we could combine the OpenMP strategy with the blocking to approach higher efficiency.

Problem 4. The results are following:

N	Thread	Jacobi	Gauss-Seidel
100	1	0.983146	0.570335
100	2	0.849995	0.338229
100	3	0.693320	0.231188
200	1	15.3308	8.9511
200	2	10.0522	5.99357
200	3	8.4823	3.66325