Lab1 Report

Ke Xu 604761427

Your methods to parallelize and optimize the programs in mmul1.c and mmul2.c.

For mmul1.c: I just changed the order of the 3 loops and implemented the parallel computation for the for loops based on OpenMP. For the loop change, I changed the order of j and k loops so that when I do the multiplication of A[i][k] and B[k][j], every time I can load each A[i][k] in the cache to increase the speed.

For mmul2.c: I just changed the order of the 3 loops and implemented the parallel computation as did in mmul1.c. Besides, I also block the 2 outside loops which are I and k loops whose block size is 32, just the same as the number of threads. Tiling is a technique meant to keep my working set inside the caches while I work on it, in order to enjoy the memory latency.

The results comparing the sequential version with the optimized parallel version on three different input sizes (1024 _ 1024; 2048 _ 2048; 4096 _ 4096). If you get significantly different speedup number for the different sizes, please explain why.

For mmul1.c:
1024: 74.5x speed up compared to sequential one. GFlop/s=15.6
2048: 90.8x speed up compared to sequential one. GFlop/s=18.3
4096: 138x speed up compared to sequential one. GFlop/s=19.8

For mmul2.c:
1024: 76.5x speed up compared to sequential one. GFlop/s=20.9
2048: 101.5x speed up compared to sequential one. GFlop/s=29.1
4096: 161x speed up compared to sequential one. GFlop/s=30.3

mmul2 only : Performance results of your OpenMP implementation on cs133.seas.ucla.edu. Please express your performance in GFlop/s = (ni *nj * nk * 2)/exectime, and the speedup compared to the serial version. Please report the speedup and the performance number with each optimization techniques you have applied.

Loop interchange gives me 11x speed up and 2.5 GFlop/s on 2048x2048.
Parallelizing + loop interchange: 90.8x speed up and 23.3 GFlop/s for 2048x2048, 138x speed up and 23.8 GFlop/s on 4096x4096
Parallelizing + blocking + loop interchange: 101.5x speed up and 30.1 GFlop/s for 2048x2048, 161x speed up and 29.3 GFlop/s on 4096x4096

mmul2 only : For the input size 2048_2048, please quantify the impact of each optimization, including the block size selection and any other optimization you performed.

Loop interchange gives me 11x speed up and 2.5 GFlop/s on 2048x2048.
Parallelizing + loop interchange: 90.8x speed up and 23.3 GFlop/s for 2048x2048
Parallelizing + blocking + loop interchange: 101.5x speed up and 30.1 GFlop/s for 2048x2048

So the loop interchange and parallelizing can speed up 11x and 9x respectively and blocking can speed up about 1.2x. Since we choose the block size of i and k loops are 32, which is based on the size of cache line, so if we increase them or decrease them, the speed can be decreased. Moreover, since we interchange the loops and can already use the cache efficiently, the blocking cannot give us a quite efficient increase of speed.

mmul2 only : For the input size 2048 _ 2048, please report the scalability of your optimized code with different number of threads, including 2, 4, 8, 16, and 32.

To test the scalability, I need to change the number of threads in main.c.

The speed up and GFlop/s are as following:
16:144x speed up and 25.64 GFlop/s.
8: 102.2x speed up and 24.58 GFlop/s.
4: 74.9x speed up and 14.84 GFlop/s.
2: 36.4x speed up and 8.69 GFlop/s.

So as the number of threads decreases, the speed also decreases as the extent of parallelization decreases.

A discussion of the your results.

The influence of parallel computing and loop interchange is very obvious since the latter can utilize the cache efficiently. Since we interchange the loops and can already use the cache efficiently, the blocking cannot give us a quite efficient increase of speed again.