MATRIX MULTIPLICATION REPORT

January 17, 2020

Cheng Chen (605428367)

0.1 Abstract

This report summarizes the results of parallel matrix multiplication and the blocked version of it. The results are compared against the sequential version.

0.2 Results Summary

0.2.1 Problem size 1024^3

Performance metric	Sequential	Parallel	Parallel-blocked
GFlops	0.649662	81.449	60.1233
Speedup	1	125.37	92.5455

0.2.2 Problem size 2048^3

Performance metric	Sequential	Parallel	Parallel-blocked
GFlops	0.292448	87.4917	77.5286
Speedup	1	299.17	265.102

0.2.3 Problem size 4096^3

Performance metric	Sequential	Parallel	Parallel-blocked
GFlops	0.19768	44.7062	117.761
Speedup	1	226.154	595.715

0.2.4 Summary

It's clear that in all of 3 problems sizes, paralleled versions are much better than sequential one. However, one may notice that only when problem size is 4096^3, parallel-blocked one is better than not-blocked version. This is because when problem size is not that large, a whole line of data can be cached all together, so the blocked version cannot increase the spatial locality.

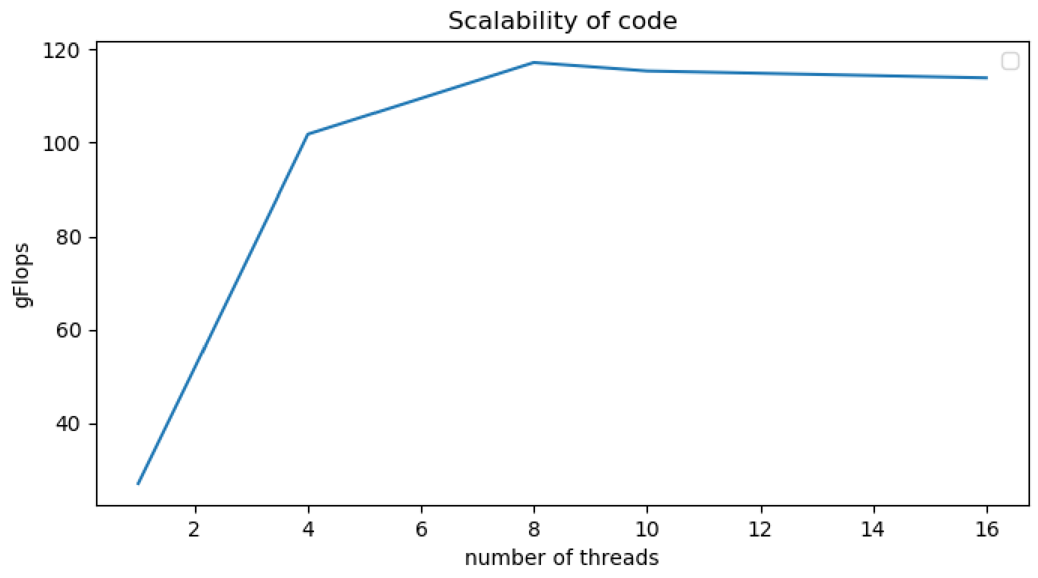
0.3 Impact of each optimization

Optimization	GFlops	Speedup
sequential	0.19768	1
parallel (not optimized)	0.918494	4.646
omit c initialization	0.932321	4.716
loop permutation	37.104	187.697
static scheduling	40.2511	203.617
number of threads	40.3017	203.873
block size optimized	117.761	595.715

It can be seen that the most influential optimizations made are loop permutation and blocking.

0.4 Code scalability

The results are shown below as a plot:



This result is reasonable as m5.2xlarge supports 8 threads, so the performance is maximized when number of threads is set to be 8. Note that the performance does not drop much when number of threads is greater than 8. As only 8 threads can essentially proceed in parallel, more threads will need to wait for some other threads to finish. This will create some overhead, but not that much.

0.5 Conclusion

Several optimizations are made to speedup the code:

- omission of C array initialization: in C++, the array of numbers have value of 0 by default, so there is no need to initialize the C array
- loop permutation: make k-loop at the lowest level to utilize spatial locality
- static scheduling: as the workload more or less remains the same, static scheduling is better to reduce the overhead
- number of threads: performance is maximized when number of threads aligns with number of CPUs
- blocking: blocked version improves the performance to a large extent as spatial locality is increased

It is clear that the optimized parallelization speeds up the code to a great extent, and the blocked version only has advantage if the problem size is large enough.