CSCI 59000 HPC Spring 2017

Report for Assignment 2

-techniques to improve sequential matrix multiplication algorithm

Feng Li

02/15/17

1. Abstract

Matrix multiplication is one of the most basic but important linear algebra problems. In the first lab, different orders of loops were tried to explore the effects of spatial locality. In this lab, two typical optimization techniques, blocking and loop unrolling, are tried to further improve the performance of sequential dgemm(double precision general matrix multiplication). After applying both techniques, best performance of multiplication of size 1000 matrix has increased from 1.28 gflops to 2.31 gflops(80% increase). However, compared with theoretical peak performance of the HPC resources(10 gflops) and the best performance achieved by auto-tuning tools(7.96 gflops), current algorithm still leaves much room for improvement

2. Introduction

Blocked(Tied) Matrix Multiplication

Block algorithms can achieve better data reuse in memory. By moving small blocks into fast local memory, there elements can then be repeatedly reused.

Loop unrolling

The loop unrolling method tries to increase program’s speed by reducing loop-controlling instructions, thus branch penalties are also decreased.

ATLAS

ATLAS(Automatically Tuned Linear Algebra Software) provides highly optimized linear algebra kernels for arbitrary cache-based architectures. Once the auto-tuning is completed, ,one can have about the performance of underlying computer systems.

3. Implementation and Experimental Results

Implementation

Matrix layout

Matrix are stored in 1-D arrays in C style(row major).

Verification

Before evaluating the performance of all the six methods, we need to make sure all the algorithms are correctly implemented. One approach is to compare the result of each method with standard scientific calculation library, eg. blas. In Big Red II there is an C implementation of blas, which provide efficient solution for DGEMM problems.

However, since the underlying algorithms of different methods are different, it's a little tricky when comparing the resulting matrix, due to precision and rounding issues. To compare each element of the resulting matrix, one approach is to make sure the difference is under certain threshold.

Analysis

Both methods involve memory operations. Whenever there is a read or write, system will check whether data is in cache first, if not, memory access will take place, which will use significantly more CPU cycles. So examining cache and TLB access patterns seems a good way to analyze the difference performance of the six algorithms.

Six level events are used in low level PAPI functions:

PAPI_L1_DCA: L1 data cache access

PAPI_L1_DCM: L1 data cache misses

PAPI_L1_ICA: L1 instruction cache accesses

PAPI_L1_ICM: L1 instruction cache misses

PAPI_TLB_DM: Data translation lookaside buffer misses

PAPI_TLB_IM: Instruction translation lookaside buffer misses

Among all the 6 events, PAPI_TLB_IM is not compatible with PAPI_L1_ICA. To solve this problem, I repeated the experiment to obtain all the information of all the six types of events.

For timing using PAPI, the high level function 'PAPI_get_real_usec()' can be used. Two timestamps are recorded before and after one execution of multiplication, respectively.

Experiment results

To find the effects of block size in block algorithm, block size ranging from 32 to 1024 are used. For each configuration, there will be 3 runs and average values of corresponding requested events are recorded.

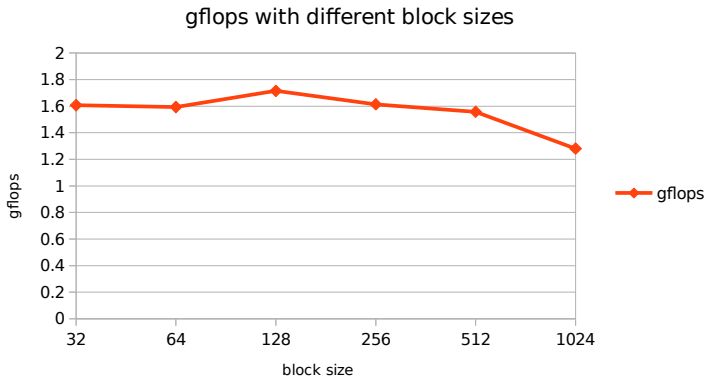

Figure 1 gflops of different block sizes

Figure 1 shows the performance of different block sizes, among which block size 128 has the best gflops of 1.71. The figure suggests that with increasing block size, the performance of blocked matrix multiplication is nearly stable among block size 32, 64, 128, and starts to drop with block size larger than 128.

Since the block size 128 achieved the best performance in experiment 1, different unrolling sizes are then applied to this case. The result is shown in figure 2, where the algorithm gives best gflops of 2.31 when unrolling size equals 8.

A more detailed discussion of the results will be in next section.

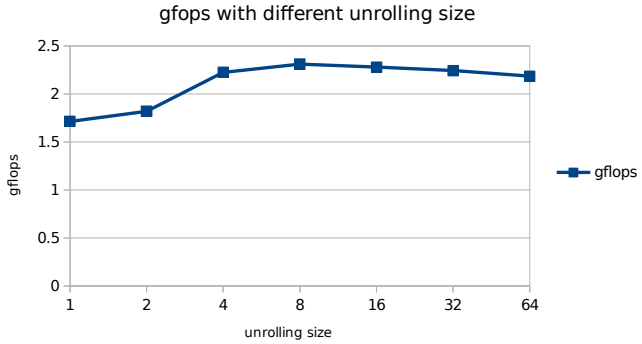

Figure 2 gflops of different unrolling sizes

4. Discussion

Block matrix multiplication

In order to understand the behavior of different different block sizes, both L1 Data Cache Miss Ratio(Figure 3, left) and TLB Data misses(Figure 3, right) are examined.

From the left part of Figure 3, we can see that larger block size have better spacial locality(data loaded from memory can be better reused). However, when

block size is larger than 256, TLB data misses are greatly increased, thus the overall performance doesn't go up anymore.

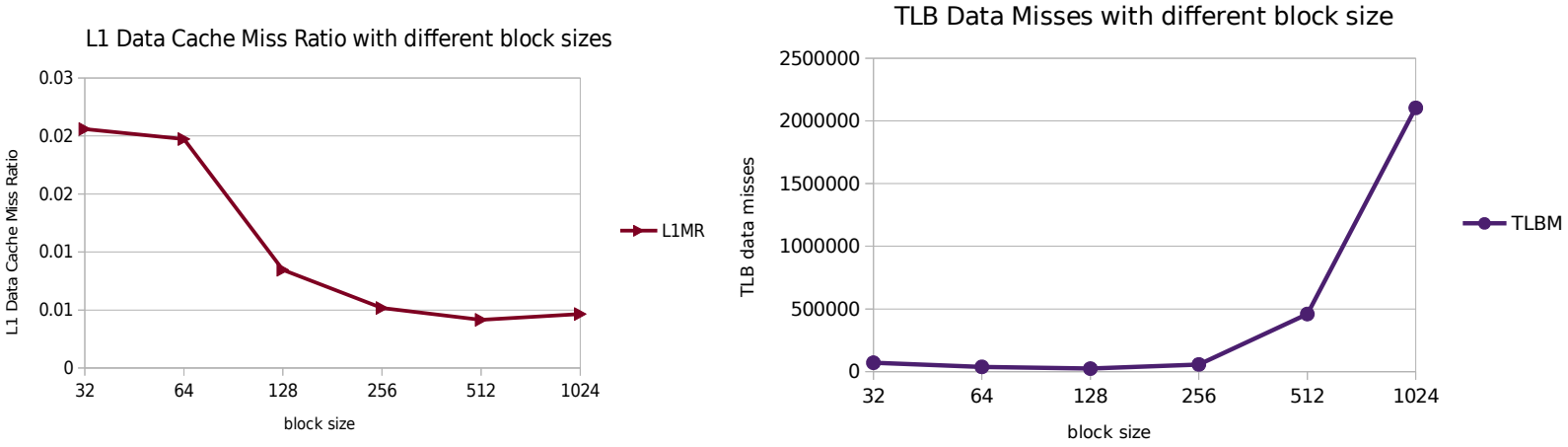

Figure 3 L1 cache miss ratio(left) and TLB data misses(right) for different block sizes

Loop unrolling

For loop unrolling, TLB misses and L1 Data Cache misses are also examined. As shown in Figure 4, the cases with larger unrolling size often have lower TLB misses, however L1 data cache miss ratio increases in the same time. As a result, the gflops of unrolling method doesn't keep increasing along with the unrolling size.

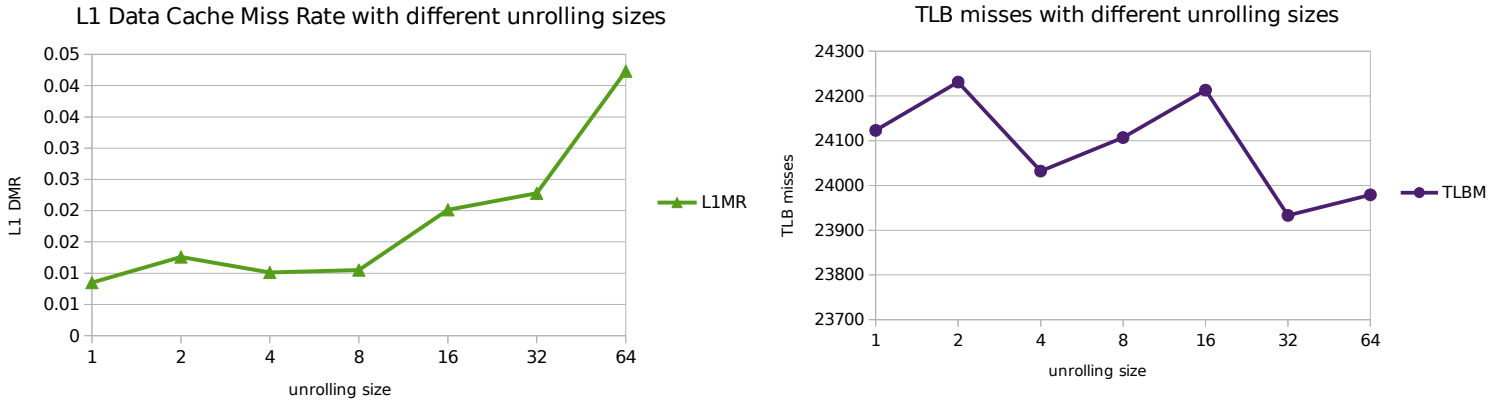

Figure 4 L1 cache miss ratio(left) and TLB data misses(right) for different block unrolling sizes

6. Conclusion

In conclusion, block method and loop unrolling are two typical ways to enhance the performance of matrix multiplication algorithm.

For block method, computational intensity can be greatly improved using larger block sizes; however if block size goes too large, there will be more TLB Data misses, which will decrease performance.

Loop unrolling is a way to optimize a program’s execution speed at the expense of its binary size. By unrolling the loops, instructions that control the loop is reduced, and delay in reading data from memory also decreases. However for very large unrolling size, the performance will drop due to higher data and instruction cache misses.