|  |  |
| --- | --- |
|  |  |

The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program   
of the Lobachevsky State University of Nizhni Novgorod   
among the world's research and education centers

Strategic initiative  
“Achieving leading positions in the field of supercomputer technology   
and high-performance computing”

**PROGRAMMING AND OPTIMIZATION   
FOR INTEL XEON PHI**

*Practice 10. Optimization of matrix multiplication.   
Improving memory efficiency*

Nizhni Novgorod

2014

# Objectives

The objective of this practice is to discuss improving memory efficiency on Xeon Phi. Our main focus is on cache efficiency. We use matrix multiplication of square matrices as example and suppose matrices fit RAM. This objective includes the following activities:

1. Using Intel MKL for matrix multiplication on CPU and Xeon Phi.

2. Sequential implementation of matrix multiplication by definition.

3. Performance analysis, vectorization, improving memory access pattern.

4. Sequential and parallel implementation of block matrix multiplication algorithm.

# Abstract

The main concept of this practice is studying basic principles of performance optimization using a classical example of dense matrix multiplication. Unlike previous practices that investigated various optimization techniques, here we focus on improving memory efficiency. Intel MKL is used as a reference implementation for correctness and performance. We start with a naive implementation of matrix multiplication and discuss how order of loops influences performance on Xeon Phi. We investigate how alignment and compiler directives affect vectorization. Several versions of block matrix multiplication are implemented.

# BRIEF OVERVIEW

We start with general problem statement and briefly overview competitive algorithms and their complexity, including Strassen and Coppersmith-Winograd. Although asymptotically superior, these algorithms are not used for matrices of reasonable size (that fit RAM) because of huge constants hidden in asymptotic complexity estimations. Thus, we use matrix multiplication by definition to illustrate optimization on Xeon Phi and single-precision floating-point arithmetic.

Matrix multiplication using Intel MKL is considered. We discuss parameters of the corresponding MKL routine and demonstrate test applications that call it on host and coprocessor in sequential and parallel mode. Our experiment confirms huge advantage of single CPU core over single Xeon Phi core. Surprisingly, in parallel Xeon Phi does not outperform 16 CPU cores. Our investigation shows that this is caused by lack of warm-up that is significant for MKL performance, especially on Xeon Phi. However, warm-up does not significantly affect performance on large matrices, thus later we provide results without specification of warming-up. With a proper warm-up Xeon Phi outperforms 16 CPU cores. All further optimization is performed on Xeon Phi.

Here is a standard implementation of matrix multiplication:

void mult(ELEMENT\_TYPE \* A, ELEMENT\_TYPE \* B,

ELEMENT\_TYPE \* C, int n)

{

int i, j, k;

for(j = 0; j < n; j++ )

for(i = 0; i < n; i++ )

C[j \* n + i] = 0;

for(i = 0 i < n; i++ )

for(j = 0; j < n; j++ )

for(k = 0; k < n; k++ )

C[j \* n + i] += A[j \* n + k] \* B[k \* n + i];

}

Performance of this implementation is several orders of magnitude worse compared to MKL. This should be no surprise: MKL matrix multiplication routine developed by experts is de facto etalon for high performance matrix multiplication. An interesting effect is that our implementation is much faster for N = 1025 compared to N = 1024. It is caused by a well-known effect: for N = 1024 we access memory with such a stride that all data is loaded into the same cache lane. This is very inefficient because each operation will load new data into the same cache lane and extrude previously loaded data, while most other cache lanes are not utilized. The described situation does not occur for N = 1025.

We investigate how ordering of loops influence computational time. Experimentally we find the best order and explain why it is superior from memory efficiency point of view.

The next step is vectorization of the innermost loop. Vectorization report shows that it was vectorized, but compiler generated several versions of the loop. These include at least vector and scalar versions as compiler does not know matrix size. Additionally, matrices are passed as pointers that theoretically could overlap. We use #pragma simd with alignment parameters to prevent generation of unnecessary versions of the loop and gain some performance benefit.

We discuss block matrix multiplication that aim to improve memory access locality, particularly by reducing cache miss ratio. We compare square and rectangular blocks and demonstrate how to give compiler a hint about block size to eliminate some inefficient prefetch.

OpenMP parallelization is performed by a single pragma for 240 threads. The resulting version is an order of magnitude slower compared to MKL. We show that 120 threads is the optimal configuration for our implementation.

We do not aim to outperform MKL. Papers [3, 4] demonstrate more efficient approaches to improving memory efficiency.

# FOR STUDENTS

An overview of theoretical advancements in efficient matrix multiplication algorithms is presented in [1, 2]. Approaches to high performance implementation of matrix multiplication are described in [3, 4], these include techniques used in MKL for CPU and Xeon Phi. Paper [5] describes matrix multiplication implementation of GOTO BLAS, a long time competitor to MKL in high performance matrix multiplication.

# References

1. Vassilevska Williams V. Efficient Algorithms for Path Problems in Weighted Graphs. 2008. URL: [http://theory.stanford.edu/~virgi/thesis.pdf]
2. Vassilevska Williams V. Multiplying matrices in O(n2.373) time. 2014. URL: [http://theory.stanford.edu/~virgi/matrixmult-f.pdf]
3. Alexander Heinecke, Karthikeyan Vaidyanathan, Mikhail Smelyanskiy, Alexander Kobotov, Roman Dubtsov, Greg Henry, George Chrysos, Pradeep Dubey Design and Implementation of the Linpack Benchmark for Single and Multi-Node Systems Based on Intel® Xeon Phi™ co-processor. //In Proceedings of the 2013 IEEE International Parallel and Distributed Processing Symposium, May 2013.
4. Tyler M. Smith, Robert van de Geijn, Mikhail Smelyanskiy , Jeff R. Hammond, and Field G. Van Zee. Opportunities for Parallelism in Matrix Multiplication // FLAME Working Note #71 . The University of Texas at Austin, Department of Computer Science. Technical Report TR-13-20. 2013.
5. Kazushige Goto, Robert van de Geijn. Anatomy of high-performance matrix multiplication //ACM Trans. Math. Soft., 34(3), May 2008.

# Individual work

1. Create a memory access map for the naive version for 8-way associative cache, cache lane is 64 bytes, cache size is 32 KB. Explain why the program is slow for N = 1024.
2. Find optimal block sizes for square and rectangular blocks for block matrix multiplication.
3. Find optimal number of OpenMP threads for block matrix multiplication depending of matrix and block sizes.
4. Investigate how warm-up affects computational time of all implemented versions.
5. Study papers [3, 4] and try these approaches on practice.