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

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**

*Lecture 9. Optimization of applications for Intel Xeon Phi:   
Intel MKL, Intel VTune Amplifier XE*

Nizhni Novgorod

2014

# Objectives

The objective of this lecture is to study tools of performance analysis and optimization for Xeon Phi. We consider a widely used profiler Intel VTune Amplifier XE, that is part of Intel Parallel Studio XE. High performance Intel Math Kernel Library is briefly discussed. The materials are given in a context of Xeon Phi programming.

# Abstract

Tools for performance analysis and optimization help to detect performance-related problems and investigate reasons of these problems, optimize software using hand-written code or implementations of standard algorithms from mathematical libraries. In this lecture we study two widely used tools from Intel Parallel Studio XE: profiler Intel VTune Amplifier XE (Amplifier in the following text) и mathematical library Intel Math Kernel Library (MKL). We suppose a reader is familiar with these tools and focus on applying them for Xeon Phi.

The first segment is devoted to MKL on Xeon Phi: offloading computations in Automatic offload and Compiler assisted offload. The second segment considers Amplifier: general overview, running on Xeon Phi, important performance evaluation metrics.

# BRIEF OVERVIEW

The first segment is devoted to MKL with the main focus on using MKL on Xeon Phi. We overview ways of calling MKL routines and general recommendations to increase performance.

Intel Math Kernel Library [7] is one of the most high performance mathematical libraries on Intel hardware. It includes lots of basic functions used in high-performance applications. There include linear algebra (BLAS, LAPACK, sparse linear algebra), FFT, vector versions of mathematical routines (trigonometric, hyperbolic, exponential, logarithmic, power, round-off), vector PRNG, statistical functions, interpolation, and others. MKL supports both shared memory systems and cluster systems. If supports Xeon Phi from version 11.0. MKL is capable of simultaneously using CPU and Xeon Phi.

There are 3 modes of using MKL:

* Automatic Offload (AO) is a transparent mode of heterogeneous computing
* Compiler Assisted Offload (CAO) provides control of offload
* Native Execution using Xeon Phi as separate processors.

Automatic Offload is the simplest way to use MKL on systems with one or several coprocessors. It requires minimal code change for porting to Xeon Phi. Coprocessor is used automatically for calls to MKL routines, including automatic data transfers between host and coprocessor memory. The library provides load balancing between CPU and Xeon Phi to achieve maximum performance. Thus, Xeon Phi will not be used in case all available coprocessors are busy or a particular function is more efficient on CPU. By default, work distribution between CPU and Xeon Phi is done automatically; we describe how one can force a particular distribution.

In Compiler Assisted Offload mode software developer controls offload via compiler directives. It is essentially offload mode of Xeon Phi programming with support for all MKL functions, unlike Automatic Offload mode. However, speedup on Xeon Phi over CPU is not guaranteed. All offload capabilities can be used to optimize performance, including explicit data transfers between host and coprocessor memory. This allows to reuse data in coprocessor memory reducing data transfer overhead. Additionally, it is possible to overlap computations and data transfers or perform simultaneous computations on CPU and Xeon Phi. Unlike Automatic Offload mode, there is no automatic load balancing. However, it is possible to use both models in the same program. Some calls might be done in Automatic Offload mode while others in Compiler Assisted Offload mode. The only limitation is that one has to specify workload distribution between CPU and Xeon Phi for code in Automatic Offload, otherwise only CPU is used.

Another segment of the lecture focuses on performance optimization for Xeon Phi using Amplifier. We overview main features of Amplifier and demonstrate ways of launching it on a coprocessor in GUI mode and from command line. We describe main performance efficiency metrics obtained from profiling.

We strongly recommend to first optimize an application for CPU using Amplifier to detect hot spots. Hotspot analysis allows to find functions and pieces of code that take most computational time. This information is often sufficient for optimization; other types of analysis provide more detailed information on program execution. We also demonstrate how to find candidates for optimization using compiler reports. In particular, one can retrieve information about most time consuming functions and loops, as well as minimum, maximum and average number of iterations of those loops.

It is important to keep checking correctness while optimizing the code, especially for parallel implementations. Intel Inspector XE is a tool that detects multithreading errors. Amplifier allows to investigate behavior of threads during execution of an application. However, the latter analysis and Inspector are only supported for CPU. Thus, we give the following recommendation:

* Use Inspector with disabled offload to detect data races and deadlocks. After these issues are resolved, proceed to debugging on Xeon Phi with enabled offload.
* Use Amplifier for multithreaded analysis with disabled offload. After all issues are resolved, proceed to optimization on Xeon Phi with enabled offload. Pay special attention to thread synchronization and load balancing, as number of threads on Xeon Phi is significantly larger compared to CPU.

We briefly describe Amplifier features and types of analysis and demonstrate profiling on Xeon Phi and interpretation of results. We enlist main metrics of performance evaluation: cycles per instruction, compute to data access ratio, memory access latency, TLB cache efficiency, vectorization intensity, memory bandwidth. Methodology of using information retrieved from a profiler is discussed.

# FOR STUDENTS

Detailed information on using MKL on Xeon Phi is presented in [1]. Performance metrics and optimization techniques are discussed in [2, 4, 5]. Using Intel VTune Amplifier XE on Xeon Phi is described in [2, 3].

# References

1. Intel Corporation. Advanced Intel Xeon Phi Coprocessor Workshop, Intel Math Kernel Library 11.0. Support for Intel Xeon Phi Coprocessor, September 2012.
2. Intel Corporation. Advanced Intel Xeon Phi Coprocessor Workshop, Performance Tuning for Intel Xeon Phi Coprocessors, September 2012.
3. Belinda L. Intel VTune Amplifier XE video tutorial 5: Using the command line, 2013 [<http://software.intel.com/ru-ru/videos/intel-vtune-amplifier-xe-video-tutorial-5-using-the-command-line>]
4. Mackay D. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 1: Optimization Essentials, 2012 [<http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-1-optimization>]
5. Cepeda S. Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events, 2012 [<http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding>]
6. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. -Morgan Kaufmann, 2013.
7. Intel Math Kernel Library Documentation [<http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation>]
8. Intel Developer Zone [<http://software.intel.com/en-us/mic-developer>]
9. Intel Compiler Documentation. Thread Affinity Interface [<http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011Update/compiler_c/optaps/common/optaps_openmp_thread_affinity.htm>]
10. Intel Xeon Phi Coprocessor. Performance Monitoring Units Documentation [<http://software.intel.com/sites/default/files/forum/278102/intelr-xeon-phitm-pmu-rev1.01.pdf>]

# Individual Work

1. Describe, in which cases MKL can be used and why?
2. What are features of MKL on coprocessor?
3. Describe ways of using MKL on coprocessor, advantages and disadvantages of each way.
4. What kind of information useful for optimization can one retrieve using Amplifier?
5. What are analysis types of Amplifier? Describe the difference between them.
6. Which role does a profiler play in the process of software optimization?
7. What are features of code profiling on coprocessor?