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

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 3. Building and running applications on Intel Xeon Phi*

Nizhni Novgorod

2014

# Objectives

The objective of this practice is to study ways of building and running applications on Intel Xeon Phi.We pay special attention to programming models of Xeon Phi and building for one or several coprocessors for each programming model under consideration.

# Abstract

We present several examples that demonstrate porting applications to Intel Xeon Phi. We consider programming models and ways of building and running applications. There is a brief overview of programming models followed by several examples. In this practice we consider Offload mode, Coprocessor-only mode and Symmetric mode.

# BRIEF OVERVIEW

First we briefly overview programming models introduced in the previous lecture and point out advantages and disadvantages of each model.

We discuss building and running applications in Offload mode. We use a simple example of an application that prints the maximum number of threads on processor and coprocessor using OpenMP. We write a function that prints the number of threads using **offload\_attribute** directive so that it can work on the coprocessor. Parameters of the directive are discussed. Then we develop function **main()**. The number of Xeon Phi coprocessors is obtained using the standard function **\_Offload\_number\_of\_devices()**. Directive **offload** is used to notify the compiler that the following code block can be executed on the coprocessor. Parameter **target(mic:<device\_id>)** specifies which coprocessor (indexed from 0) the code will be executed on. Offload code is executed synchronously, that is, the host program execution is stalled while offload code is running on the coprocessor. Asynchronous computations can be organized using several CPU threads with each thread using either a separate coprocessor or do some computations on the processor. We point out differences between **offload** and **offload\_attribute**. The program is build using Intel C/C++ Compiler (version 13 or later) and launched on the coprocessor. We demonstrate command lines to build and run the application.

Another problem we consider is dot product which was previously suggested for individual work. We create a function to compute dot product of two vectors. The computation is performed in parallel using all available cores. The code does not contain any specific directives for Xeon Phi. Again we use **offload\_attribute** directive to mark the code that can run on the coprocessor. Function **main** is responsible for testing the solution by comparing results computed on CPU and Xeon Phi.

Then we focus on native mode. In this model only Xeon Phi is used. Application development is essentially the same as for multicore CPUs (naturally, performance optimization might be significantly different), the only difference is building and running. We consider the following examples:

1. Dot product in coprocessor-only mode.

We discuss the process of building the usual CPU-oriented code for Xeon Phi coprocessors by using **–mmic** compiler flag. Different ways of launching applications on the coprocessor are demonstrated, including command line, MPI Hydra, SLURM. Thus students master launching applications on various environments.

1. MPI version of dot product for computing N independent dot products.

This problem is a generalization of the previous one and can use the previously developed function of computing dot product for a pair of vectors. Here we have an additional degree of parallelism due to the presence of N independent subproblems. We demonstrate simultaneous utilization of several coprocessors using MPI.

1. Dot product in symmetric mode.

In symmetric mode one can use both processors and coprocessors, each running a separate MPI process. Data exchange is done via MPI message passing routines. We demonstrate building and launching programs in symmetric mode. While solving the previous problem, each coprocessor executed an MPI process and CPUs were not utilized. In symmetric mode we can fully utilize all processors and coprocessors. This requires a modification of building, so that the code is compiled for both CPUs and Xeon Phi, as well as application launching for both devices that we illustrate.

# FOR STUDENTS

We recommend [2] for further details of using OpenMP and [3] for **offload** directive. Some simple examples of programs for Xeon Phi are presented in [4], chapter 2.

# References

1. Intel Xeon Phi Coprocessor System Software Developers Guide, revision 2.03, 2012
2. Best Known Methods for Using OpenMP on Intel Many Integrated Core (Intel MIC) Architecture, Volume 1a, January 29, 2013
3. Green R.W. Effective Use of the Intel Compiler's Offload Features: [<http://software.intel.com/en-us/articles/effective-use-of-the-intel-compilers-offload-features>]
4. Jeffers J., Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. Morgan Kaufmann, 2013.

# individual work

1. Implement matrix-vector multiplication in Offload mode.
2. Implement matrix-vector multiplication in coprocessor-only mode. Suppose there is only one coprocessor.
3. Implement matrix-vector multiplication in symmetric mode. Provide two levels of parallelism: simultaneous computation of row-vector products and parallel computation of each row-vector product.

For each problem build and run your application on Xeon Phi, analyze the correctness. Compare execution time on host and coprocessor.