**Meeting.0: Introduction, organization**

Introduction, regulations, platform enrollment, configuration of working environment.

**Meeting 1: Basics of C++ threads**

1. Using a global function, function object, class method, and lambda function create 4 threads that display on the screen information about the thread and an additional text (being an argument passed when creating a thread).

2. Create a global function that prints 50 times (each time on a separate line) the text passed as a function’s argument. Create 20 threads using this function. Using the mutex, ensure that messages output by threads will be placed in separate lines.

3. Create a function that increments 10000000 times a certain global integer variable. Measure the time (std::chrono) of its one-time execution. Create a vector of 10 threads with this function. Check the time and the effect of these threads on the incremented global variable. Use the <atomic> and <mutex> to ensure the correct result of the threads. Compare the operation times of the <atomic>, <mutex>, and unsynchronized variants (for the executable compiled in the debug and release configuration).

**Meeting 2,3: The producer and consumers scheme**

1. Create a Queue class representing a FIFO (first in first out) queue of length limited to the value passed as a constructor parameter, whose elements are arrays of ARRSIZE (=100000) integers. As an element of the Queue, You may use std::array<> or allocate arrays dynamically and use a unique\_ptr<> or regular pointer.

2. Create the Producer class, which produces tables of ARRSIZE random integers and inserts them into the queue described in point 1 (queue and the number of arrays to be produced passed as the argument of the constructor).

3. Create a Consumer class that takes tables from the queue (the queue passed as constructor argument) and sorts them (sorts a single table, outputs its checksum, discards it, and then takes the next one).

Producer and Consumer should yield() if the Queue is full or empty, respectively. Ensure that the cooperating threads properly finish their jobs and destruct (no produced table may be discarded prior to being sorted, Queue may be destructed only if it is empty and all the consumers and producers attached to it are destructed). At the end, each consumer should report how many tables it sorted.

4. Create: 1 Queue thread, 1 Producer thread, and several Consumer threads. The capacity of the queue must be many times bigger than the maximum number of Consumer threads planned to be created during experiments; the number of tables to be produced by the Producer thread must be many times bigger than the queue capacity. Check what is the dependence of the speed of retrieving tables from the queue by consumers depending on number of created Consumers threads. Check if it is beneficial to create a number of threads greater or significantly greater than thread::hardware\_concurrency() or greater than the number of physical cores in the CPU?

In the report include the information on: parameters of the computer system used for experimenting with threads (RAM size, CPU type, CPU speed, number of physical and logical cores), compiler parameters (version, mode: Release/Debug, architecture: x64/x86) and experiment parameters (number of sorted arrays, array size, que size). Comparison of the speed of retrieving arrays from the queue obtained for various numbers of Consumers should be presented in a tabular and graphical form. What is the best number of Consumers in relation to the number of physical and logical cores? As an appendix include in the report the source code of your implementation.

**Meeting 4: Basics of MPI**

1. Installing and using MPI in VisualStudio

Download from: https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi

msmpisetup.exe - MS MPI for Windows

msmpisdk.msi - MPI SDK for Visual Studio

Install both, then check if the following environment variables are set:

MSMPI\_BENCHMARKS=C:\Program Files\Microsoft MPI\Benchmarks\

MSMPI\_BIN=C:\Program Files\Microsoft MPI\Bin\

MSMPI\_INC=C:\Program Files (x86)\Microsoft SDKs\MPI\Include\

MSMPI\_LIB32=C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x86\

MSMPI\_LIB64=C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x64\

Run Visual Studio, create Visual C++ Win32 Console Application

In project properties (assuming we compile in x64 mode), set (or add) to Project->*Your\_project* Properties->Configuration properties:

->C/C++->General->Additional Include Directories-> $(MSMPI\_INC);$(MSMPI\_INC)\x64

->Linker->General->Additional Library Directories-> $(MSMPI\_LIB64)

->Linker->All Options->Additional Dependencies-> msmpi.lib

Now input the example MPI application code, compile it.

#include "mpi.h"

#include "stdafx.h"

#include <iostream>

using namespace std;

int main(int argc, char \*argv[])

{

int rank, size;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

cout << "Hello from process " << rank << " out of " << size << endl;

MPI\_Finalize();

return 0;

}

Running MPI executable app (number\_of\_processes defaults to the number of CPU cores; a firewall may require setting permissions at a first run of MPI executable):

mpiexec [–n number\_of\_processes] executable\_file.exe

You may add a command to VS (Tools->External Tools->Add):

Title: RunMPI

Command: c:\Program Files\Microsoft MPI\Bin\mpiexec.exe

Arguments: $(TargetName).exe

Initial Dir.: $(BinDir)

Close on exit: *unchecked*

And then add it to the toolbar.

2. Write a program, that tests:

- point-to-point routines MPI\_Send() and MPI\_Recv() (but use also MPI\_ANY\_SOURCE for the latter)

- non-blocking routines MPI\_Isend() and MPI\_Irecv() (use also MPI\_Wait() and MPI\_Test())

- MPI\_Barrier()

3. Write a program, that finds the number of prime numbers smaller or equal then a given number. For instance, the program should find 78498 primes up to 1000000, 664579 up to 10^7, and 5761455 up to 10^8. Use MPI\_Reduce() to sum numbers of primes found by MPI processes. Checking if a (positive) number is prime may be done by testing if the reminder from its division by successive numbers is 0:

int is\_prime(int nr)

{

if (nr < 2)

return 0;

for (int i = 2; i <= sqrt(int(nr)); i++)

if (nr%i == 0)

return 0;

return 1;

}

**Meeting 5,6: The producer and consumers scheme using MPI**

1. Port the producer-consumers program created using threads to the MPI environment. You may use only the point-to-point routines (with MPI\_Recv() receiving MPI\_ANY\_SOURCE) or other routines as well. However, first use only blocking routines, if the scheme works correctly try to increase parallelism by employing non-blocking routines.

2. Check what is the dependence of the speed of retrieving tables from the queue by consumers depending on number of created Consumer processes and compare these results to the results obtained for C++ threads.

In the report include the information on: parameters of the computer system used for experimenting with MPI and threads (RAM size, CPU type, CPU speed, number of physical and logical cores), compiler parameters (version, mode: Release/Debug, architecture: x64/x86) and experiment parameters (number of sorted arrays, array size, que size). Comparison of the speed of retrieving arrays from the queue in the case of MPI and C++ threads obtained for various numbers of Consumers should be presented in a tabular and graphical form. If you tested both the blocking and non-blocking routines then show the results of both versions. What is the best number of Consumers in relation to the number of physical and logical cores? As an appendix include in the report the source code of your implementation.

**Links on MPI**

https://www.codingame.com/playgrounds/349/introduction-to-mpi/introduction-to-distributed-computing

http://mpitutorial.com/tutorials/

http://miclab.pl/core/index.php/MICLAB:MPI [PL]

http://jedrzej.ulasiewicz.staff.iiar.pwr.wroc.pl/Komputery-i-Syst-Rownolegle/wyklad/MPI22.pdf [PL]