University of Basel

Lecturers:
Prof. Dr. Christian Tschudin
Prof. Dr. Heiko Schuldt
Prof. Dr. Florina M. Ciorba

Assistants:
Fabrizio Parrillo
Ralph Gasser
Gabrielle Poerwawinata
Thomas Jakobsche

https://dmi.unibas.ch/de/studium/computer-science-informatik/lehrangebot-hs21/lecture-foundations-of-distributed-systems/

Join Zoom Meeting https://unibas.zoom.us/j/64769764960?pwd=bGp6cTBXb1NuNG9tTjhiT2szSmVIZz09

Foundations of Distributed Systems HS 2021 (45402-01) **Fall Semester 2021**

Assignment 5 - OpenMP and MPI Basics

(50 Points)

Starting Date: November 23, 2021
Deadline: December 12, 2021 - 23:55

Assistants contact:
gabrielle.poerwawinata@unibas.ch
thomas.jakobsche@unibas.ch

The first thing you need to do is **register your group in the Google sheet** provided by the following link:
Google sheet to register your group
This sheet will also be used for setting the time slots for the assignments evaluation.

In this assignment you will work on the miniHPC cluster (see attached description **miniHPC_Usage.pdf**).
To connect to the cluster use:
ssh -X UserName@cl-login.dmi.unibas.ch
The **UserName** is the same as your short name for the university of Basel and the **password** is the same of your UniBas email.

To run your exercises you should create a **`job script`** using the form at the following URL:
Form for generating your job script file
To submit the generated **`job script`** to the batch scheduler use the command: **`sbatch jobscript.job`**.
More information about how to execute your exercises on miniHPC also on the attached file **miniHPC_Usage.pdf**.

The delivered solutions should be in a single **tar file.**
To create the **tar file** use the following command:
`tar -zcvf archive-name.tar.gz directory-with-your-solutions`

You can find a lot of information, definitions and syntax for OpenMP and MPI commands in the course slides. Furthermore, you can also consult the following URLs:

- OpenMP
- OpenMP 5.0 Complete Specifications
- An open source MPI implementation

The job script created with the link provided in the introduction of the exercise will be similar to the following example. This is a commented (a line prepended by #) example to help you understand what each variable in the job script represents. The example below is also attached in a separate file (JobScriptExample.job) to the exercise.

```
#!/bin/bash

#SBATCH -J TEST # Job name.
#SBATCH --time=00:05:00 # Maximum estimated running time for this job.
#SBATCH --exclusive # You will run experiments exclusively on the allocated nodes.
#SBATCH --nodes=2 # Number of nodes you request from the batch scheduler.
#SBATCH --ntasks-per-node=2 # Number of MPI ranks per node. If you use nodes=2 and ntasks-per-node=2 you will be running 4 ranks.
#SBATCH --cpus-per-task=2 # Number of OpenMP threads.
#SBATCH --partition=xeon # Which node partition you request nodes from. You can find more information about the partitions on the attached miniHPC_Usage.pdf file.
#SBATCH --output=OUTPUT.txt # The files where the output of your program will be directed to.
#SBATCH --hint=nomultithread # Disable the usage of hyperthreads.

ml intel # Load the intel compiler.
srun yourCode.o 10000 1024 0 0 0.75 # Run your code for this exercise with the needed parameters. Recall that the Xeon nodes on miniHPC have only 20 cores each and that you will use 8 OpenMP threads per rank in certain experiments. Therefore, you can not place more then 2 MPI ranks per node. For the parallel experiments use a maximum of ntasks-per-node=2 this way you ensure that you will not have more than 2 ranks on the same node.
```

Please make sure that any optimization you introduce does not affect the correctness of the results.

1 Task - OpenMP Basics: (20 points)

Recall that to compile an **OpenMP** program, you should write the following command in your job script:

module load intel
icc -O3 -fopenmp hello.c -o hello

Sines (calculates the sine function of an expression) *(6 Points)*
Paralellize the following code.
Skeleton in **codeSkeletons/sines.c**.

```
double* a = (double*) malloc (N * sizeof (double));
for (i = 0; i < N; i++)
  a[i] = sin (2.0 * M_PI / 1000.0 * i);
```

- Hand in the source code.

What the heck *(7 Points)*
Correct the following code.
Skeleton in **codeSkeletons/whatTheHeck.c**.

```
static const int a[] = {1,2,3,4,5,6,7,8,9,10};

int sum(const int* arr, int size) {
  int s=0, i;
  #pragma omp parallel for
  for(i=0; i < size; i++) {
    s += arr[i];
  }
  return s;
}

int main() {
  printf("sum: %d\n", sum(a, 10));  //expected 55
  return 0;
}
```

Hint: Compile and run the whatTheHeck.c code template to see what happens and you will understand the exercise tittle.

- Hand in the source code.
- Describe what is happening in the whatTheHeck.c code and why it is happening.

Fibonacci *(7 Points)*
Write a program to calculate the following Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

- Hand in a sequential version of the source code.
- Hand in an OpenMP parallel version of the source code.

2 Task - MPI Basics: (30 points)

Recall that in order to compile an **MPI** program you should write the following command in your job script:

module load intel
mpiicc -O3 mycode.c –o mycode

Hello World *(7 Points)*

Write a program that sends a char array from one process to the next until the last one prints it to the console.

- Use the blocking version of MPI_Send, MPI_Receive.
- Use the non blocking version of MPI_Send, MPI_Receive.
- Use MPI_Bcast.
- Hand in the source code.

Tree-Structured Global Sum *(8 Points)*

For this task you should write a tree-structured sum function that uses MPI. In the tree_template.c skeleton provided, all the processes participate in beginning, and processes drop out as the algorithm proceeds. Write the missing code and test it.
Skeleton in **codeSkeletons/tree_template.c**.

- Complete global_sum using the primitives MPI_Send, MPI_Receive.
- Hand in the source code.

Ring-Pass Global Sum *(8 Points)*

For this task you should write a ring-pass sum function that uses MPI.
Skeleton in **codeSkeletons/ring_template.c**.

- Complete global_sum using MPI_Send, MPI_Receive.
- Hand in the source code.

Reduce *(7 Points)*

Write a program which calculates the cumulative sum of all numbers from 1 to 2000.

- Use MPI_Reduce to implement the program.
- Hand in the source code.