University of Basel

Lecturer:
Prof. Dr. **Florina M. Ciorba**
Prof. Dr. Heiko Schuldt
Prof. Dr. Christian Tschudin
Prof. Dr. Isabel Wagner

Seminarraum 05.002,
Spiegelgasse 5,
CH-4051 Basel

Assistants:
Ali Ajorian, M.Sc.
Jonas Korndörfer, M.Sc.
Shiva Parsarad, M.Sc.
Yiming Wu, M.Sc.

https://vorlesungsverzeichnis.unibas.ch/en/home?id=285227

Foundations of Distributed Systems 2024 (45402-01, 45402-02) **Fall Semester 2024**

Assignment 3 - OpenMP and MPI Basics

(20 Points)

Starting Date: October 22, 2024
Deadline: November 21, 2024 - 23:55

Assistant contact:
jonas.korndorfer@unibas.ch

Useful information

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 registration and sheet will be used for setting the time slots for the assignments interview & 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`, where `UserName` is the short name of your UniBas account and `password` is your UniBas password.

To login to miniHPC you must be connected to the UniBas VPN (instructions available online[1]).

The solutions of this assignment must be delivered in a zip or tar file containing:

- All source codes containing the solutions.
- All job scripts used to execute and collect performance measurements of the exercises.
- A single PDF file containing the requested plots and or written answers.

You can find rich information, definitions, and syntax for OpenMP and MPI calls in the lecture slides. You can additionally consult the following URLs:

- OpenMP https://www.openmp.org
- OpenMP 5.2 Complete Specifications https://www.openmp.org/wp-content/uploads/OpenMPRefGuide-5.2-Web-2024.pdf
- OpenMPI https://www.open-mpi.org/

[1] https://its.unibas.ch/de/anleitungen/netzwerkzugang/anleitung-vpn/

To execute and test your solution for this exercise, create a job script similar to the following example. In this example, lines prepended by #SBATCH inform the batch scheduler about the desired job configuration. The second # on each line denotes a comment, to help understand what each variable in the job script represents. This example is also included in a separate file (JobScriptExample.job) in the exercise handout. To execute a job script on miniHPC, use `sbatch your-job-script-file-name.job`.

```
#!/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. With nodes=2 and ntasks-per-node=2 you will be running on 4 MPI ranks.
#SBATCH --cpus-per-task=2 # Number of OpenMP threads.
#SBATCH --partition=xeon # Partition from which you request nodes. More information about the partitions can be found in the attached miniHPC_Usage.pdf file.
#SBATCH --output=OUTPUT.txt # File where the output of your program will be directed to.
#SBATCH --hint=nomultithread # Disable the usage of hardware multithreading.

ml intel # Load the Intel compiler.
srun yourCode.out 10000 1024 0 0 0.75 # Run code for this exercise with the needed parameters.
```

Recall 1. 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`** to ensure no more than 2 MPI ranks on the same node.

Recall 2. To compile an **OpenMP** program you should used the following command:

```
module load intel
icc -O3 -fopenmp mycode.c -o mycode
```

Recall 3. To compile an **MPI** program you should used the following command:

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

Please ensure and verify that any optimization you introduce does not affect the correctness of the results.

1 Task - OpenMP Basics: (8 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) *(2 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 *(3 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 *(3 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: (12 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 *(2 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 *(3 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 *(3 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 *(4 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.