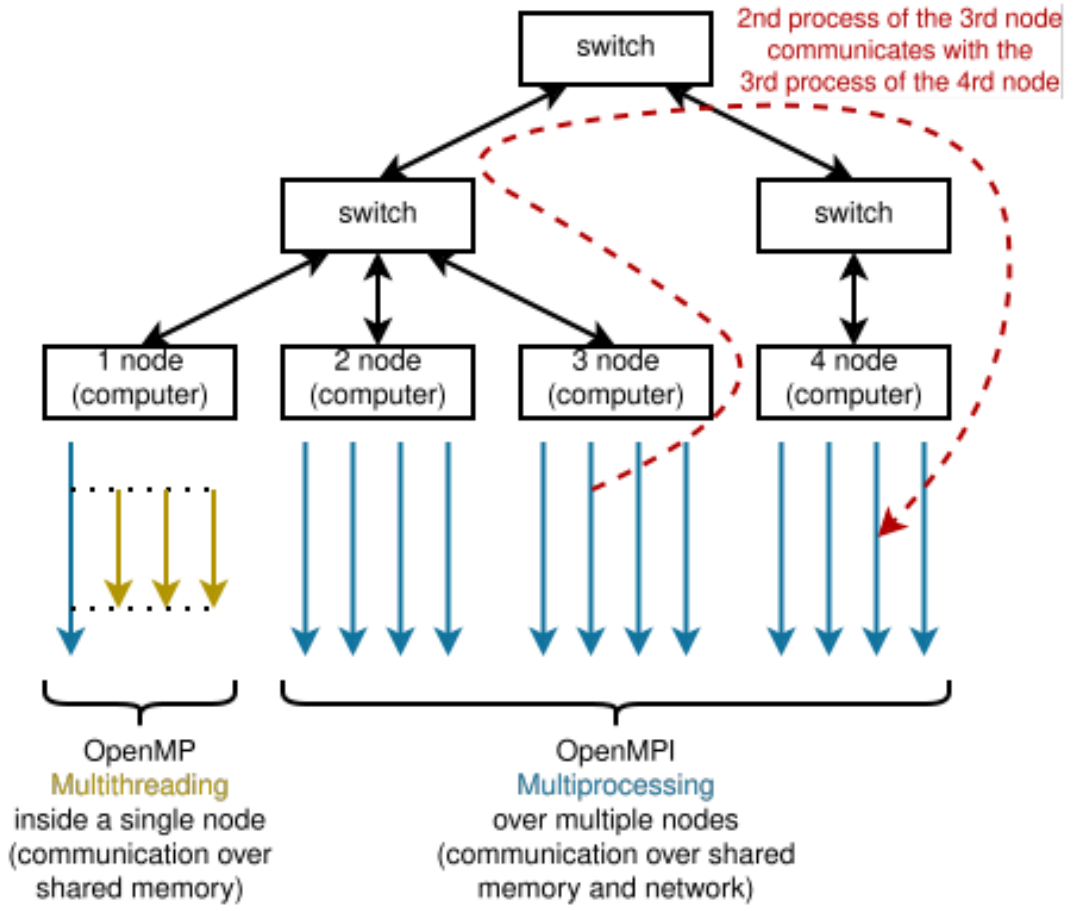
Where and how to learn MPI

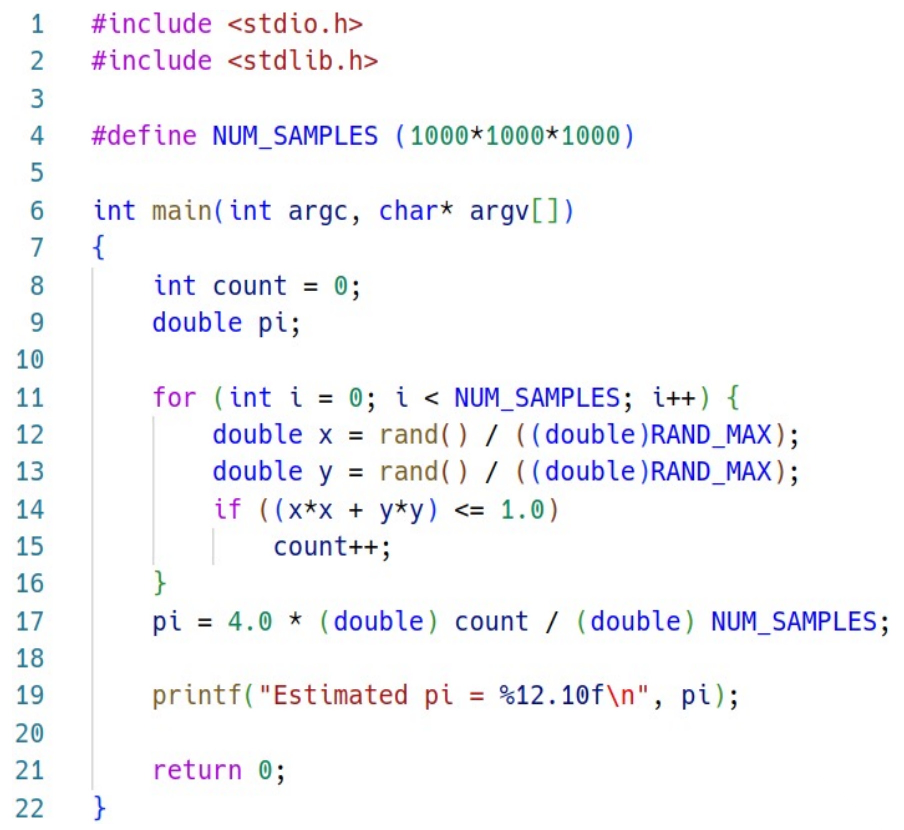
Parallelization with OpenMP and OpenMPI

- A cluster consists of multiple nodes (computers) connected by a fast network (e.g., fibre optic)
- A node has many cores (in case of DACHS' AMD 9254 / 9454: 48 / 96cores)

- Inside a node, we can run a program in parallel with Multithreading: OpenMP
- Using multiple nodes, we can use Multiprocessing combined with network communication: OpenMPI



Simple example program



```c
#include <stdio.h>
#include <stdlib.h>

#define NUM_SAMPLES (1000*1000*1000)

int main(int argc, char* argv[])
{
    int count = 0;
    double pi;

    for (int i = 0; i < NUM_SAMPLES; i++) {
        double x = rand() / ((double)RAND_MAX);
        double y = rand() / ((double)RAND_MAX);
        if ((x*x + y*y) <= 1.0)
            count++;
    }
    pi = 4.0 * (double) count / (double) NUM_SAMPLES;

    printf("Estimated pi = %12.10f\n", pi);

    return 0;
}
```

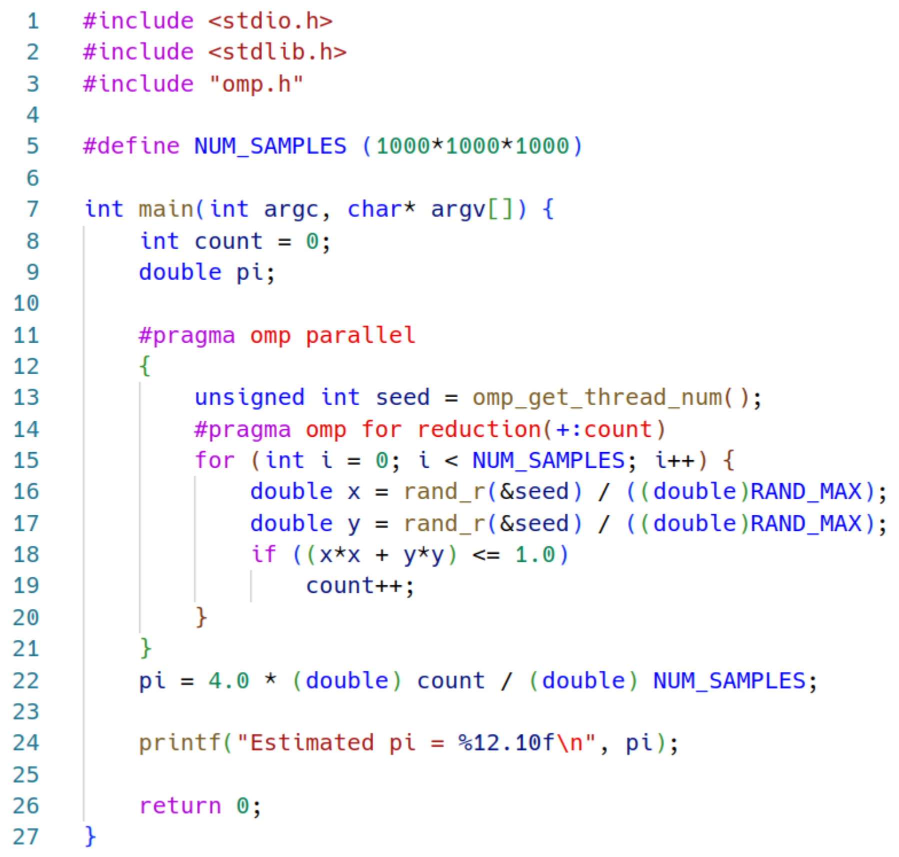
- Estimating Pi with the Monte Carlo Method
- Define how many points are generated
- Generate one random point in a square with side length 2r

- Increase count by one if the point lies in a circle with radius r inside the square
- Calculate Pi
- Print Pi with ten decimal places

Simple example program with OpenMP 1/3

- OpenMP specification has been defined by a consortium of industry and research since 1997
- OpenMP 5.0 was released in 2018
- Implemented in the compiler via compiler directives
- In C/C++: `#pragma omp`
- In Fortran: `!$OMP`
- Enables Multithreading

Simple example program with OpenMP 2/3



```c
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"

#define NUM_SAMPLES (1000*1000*1000)

int main(int argc, char* argv[]) {
    int count = 0;
    double pi;

    #pragma omp parallel
    {
        unsigned int seed = omp_get_thread_num();
        #pragma omp for reduction(+:count)
        for (int i = 0; i < NUM_SAMPLES; i++) {
            double x = rand_r(&seed) / ((double)RAND_MAX);
            double y = rand_r(&seed) / ((double)RAND_MAX);
            if ((x*x + y*y) <= 1.0)
                count++;
        }
    }
    pi = 4.0 * (double) count / (double) NUM_SAMPLES;

    printf("Estimated pi = %12.10f\n", pi);

    return 0;
}
```

- omp header
- Variables accessible by each thread
- Start of parallel region: creates threads (variables defined in the region are thread local)
- Get the number of the current thread
- Split loop iterations across threads using a reduction to collect the values of each thread into a variable visible to all threads
- Use only thread-safe functions in a parallel region (use a different seed in each thread for different random values)
- End of parallel region

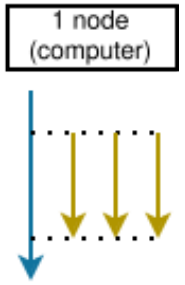
Simple example program with OpenMP 3/3

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --partition=gpu1
#SBATCH --time=00:10:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=48

module load mpi/openmpi/5.0.8-gnu-14.3
gcc -O2 -fopenmp monte_carlo_pi_openmp.c -o monte_carlo_pi_openmp

OMP_NUM_THREADS=1 /usr/bin/time ./monte_carlo_pi_openmp
OMP_NUM_THREADS=2 /usr/bin/time ./monte_carlo_pi_openmp
OMP_NUM_THREADS=4 /usr/bin/time ./monte_carlo_pi_openmp
OMP_NUM_THREADS=8 /usr/bin/time ./monte_carlo_pi_openmp
OMP_NUM_THREADS=16 /usr/bin/time ./monte_carlo_pi_openmp
OMP_NUM_THREADS=32 /usr/bin/time ./monte_carlo_pi_openmp
```

- One node
- One task (process)
- Up to 48 threads per process
- Compile needs -fopenmp
- Environment variable OMP_NUM_THREADS defines number of threads
- With 4 threads



threads	1	2	4	8	16	32
time in s	9.25	4.63	2.33	1.17	0.59	0.30

Simple example program with OpenMPI 1/4

- The MPI specification has been defined by universities & companies since 1994
- MPI 4.1 was released in 2023
- Implemented as libraries, e.g., as OpenMPI
- Offers C and Fortran API
- Enables Multiprocessing and network communication

Simple example program with OpenMPI 2/4

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define NUM_SAMPLES (1000l*1000*1000*10)

int main(int argc, char* argv[])
{
    int rank, size;
    long count = 0;
    double pi;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    srand(rank);
    for (long i = 0; i < NUM_SAMPLES / size; i++) {
        double x = rand() / ((double)RAND_MAX);
        double y = rand() / ((double)RAND_MAX);
        if ((x*x + y*y) <= 1.0)
            count++;
    }

    int root = 0;
    long global_count;
    MPI_Reduce(&count, &global_count, 1, MPI_LONG, MPI_SUM,
        root, MPI_COMM_WORLD);
    if (root == rank) {
        pi = 4.0 * (double) global_count /
            (double) NUM_SAMPLES;

        printf("Estimated pi = %12.10f\n", pi);
    }

    MPI_Finalize ();

    return 0;
}
```

- mpi header
- Workload should be greater than the overhead of OpenMPI: 10 times more samples
- Initialize MPI: MPI functions are callable after the MPI_Init call
- Get the number of the current process and the count of all processes
- Use a different seed in each process for different random values
- Split the loop manually for each process (for this simple split, the sample size must be divisible by the number of processes without remainder)

Simple example program with OpenMPI 3/4

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define NUM_SAMPLES (1000l*1000*1000*10)

int main(int argc, char* argv[])
{
    int rank, size;
    long count = 0;
    double pi;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    srand(rank);
    for (long i = 0; i < NUM_SAMPLES / size; i++) {
        double x = rand() / ((double)RAND_MAX);
        double y = rand() / ((double)RAND_MAX);
        if ((x*x + y*y) <= 1.0)
            count++;
    }

    int root = 0;
    long global_count;
    MPI_Reduce(&count, &global_count, 1, MPI_LONG, MPI_SUM,
        root, MPI_COMM_WORLD);
    if (root == rank) {
        pi = 4.0 * (double) global_count /
            (double) NUM_SAMPLES;

        printf("Estimated pi = %12.10f\n", pi);
    }

    MPI_Finalize ();

    return 0;
}
```

- No threading: non-thread-safe functions are usable, and all variables are only visible in one process
- We define the process with rank 0 as the root process that gathers all counts and prints the final result
- MPI_Reduce sums (using MPI_SUM) all counts and gathers them in global_count in the root process
- Only the root process makes the final calculation and prints the result
- MPI_Finalize ends all processes

Simple example program with OpenMPI 4/4

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --partition=gpu1
#SBATCH --time=00:10:00
#SBATCH --ntasks-per-node=48

module load mpi/openmpi/5.0.8-gnu-14.3
mpicc -O2 monte_carlo_pi_openmpi.c -o monte_carlo_pi_openmpi

/usr/bin/time mpirun -np 1 ./monte_carlo_pi_openmpi
/usr/bin/time mpirun -np 2 ./monte_carlo_pi_openmpi
/usr/bin/time mpirun -np 4 ./monte_carlo_pi_openmpi
/usr/bin/time mpirun -np 8 ./monte_carlo_pi_openmpi
/usr/bin/time mpirun -np 16 ./monte_carlo_pi_openmpi
/usr/bin/time mpirun -np 32 ./monte_carlo_pi_openmpi
/usr/bin/time mpirun -np 64 ./monte_carlo_pi_openmpi
```

- Two nodes
- Up to 48 tasks (processes) per node
- Compile needs wrapper that sets include paths and links libraries
- mpirun should be used to parallelize (-np defines the number of processes to use)

processes	1	2	4	8	16	32	64
time in s (1 B Samples)	15.81	8.95	5.48	3.82	3.19	3.56	3.76
time in s (10 B Samples)	140.44	71.43	37.17	19.44	11.13	7.78	5.87

Further Sources

- Best Practices: https://wiki.bwhpc.de/e/Development/Parallel_Programming
- OpenMP: https://www.openmp.org/specifications/
- OpenMPI: https://www.open-mpi.org/

Further Material:

- Courses: at HLRS the Parallel Programming Workshop: https://www.hlrs.de/training/hpc-training
- The bwHPC Training platform: https://training.bwhpc.de
- The bwHPC WIKI on Parallel Programming