Lab Class High Performance Computing
Prof. Tobias Kreilos

Instruction sheet for lab class 12

Exercise 1: Sieve of Eratosthenes

The sieve of Eratosthenes is an algorithm for finding all prime numbers up to a given limit N.

In pseudocode, the serial algorithm is:

```
1. Create a list of consecutive integers from 2 through N
2. Let k equal 2
3. Mark all multiples of k (the number k should not be marked)
4. Find the smallest number in the list greater than k that is not marked.
    a. If there is no such number, stop.
    b. Otherwise, let k equal this number and repeat from 3.
5. The remaining numbers that are not marked are all primes up to N.
```

Your task is to implement a parallel version of the algorithm with MPI. Use (1d) domain decomposition, i.e. split the list of integers among the processes.
Both steps 3 and 4 need to be carefully analyzed in the parallelization.

Thoroughly test your implementation.

Upload your solution to moodle.

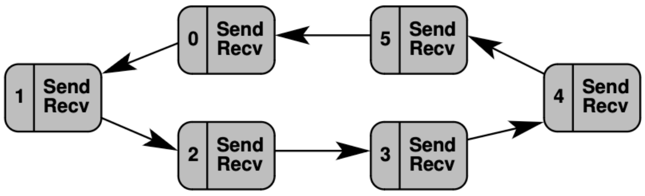
Perform a strong scaling study of your algorithm with N=1.000.000 fixed. Show plots of performance, speedup and efficiency.

Exercise 2: MPI functions not covered

There are two important MPI functions which have not been covered in the lecture. These are MPI_Sendrecv and MPI_Comm_split. Read about the two functions. What do they do and where could you use them?

Exercise 3: Hybrid ring communication

Revisit the ring communication pattern, where each process sends its rank to the right neighbor.



Implement a hybrid version, where each process runs the same number of threads. Each thread sends two numbers to the thread with the same ID on the process with the next rank. The two numbers are rank and thread ID.

Example values with 3 ranks and 2 threads per rank:

Rank	**ID**	**Sent**	**Received**
0	0	(0,0)	(2,0)
0	1	(0,1)	(2,1)
1	0	(1,0)	(0,0)
1	1	(1,1)	(0,1)
2	0	(2,0)	(1,0)
2	1	(2,1)	(1,1)

Remember to set MPI_THREAD_MULTIPLE.

Upload your solution to moodle.

Exercise 4: Show off your skills

In this semester, you have learned a lot about writing high performance programs. Among other things, you now are proficient in

- Parallelizing computations with OpenMP
- Parallelizing computations with MPI
- Hybrid parallelization with MPI and OpenMP
- Profiling code to identify bottlenecks and find optimization potential
- Optimizing cache access to speed up execution times
- Minimizing impacts from memory access in ccNUMA systems
- Optimizing serial programs

One of our main case studies that we used to learn all these things was the Jacobi algorithm to find steady states of the heat equation. Now it is time to combine everything you have learned and write the fastest possible implementation of that algorithm!

The benchmark setup is found on moodle, together with a sample implementation in pure MPI.

- Use the same benchmark parameters as are used in the example.
  - Matrix size 3072x3072, epsilon 3e-6
- Do not use a different output format in main.cpp or a different initial condition.
- Make sure that your program can be compiled with “make release” – feel free to modify the compiler command in the makefile, however.
- Provide an sbatch script “run.sh” that can be used to submit your program to the cluster – you may adjust the run parameters as required to give the optimal speed!

Hints:

- You can freely choose a parallelization strategy (MPI/OpenMP/Hybrid/2D distribution,…).
- Choose the number and distribution of processes that gives the fastest results.
- Also optimize the Jacobi algorithm itself, e.g. by overlapping computation and communication, reducing the amount of work, … In the lecture we discussed a lot of optimizations but never put all of them together.

Upload your solution to moodle. We will compare all implementations and create a ranking of the fastest teams!
Note that I will not trust your results but compile the code anew and run it on the cluster – so make sure, that you your code can be compiled and executed with **“make release && sbatch run.sh”**