Epidemics Simulation

With MPI parallelism

Blejdea Emil Alexandru

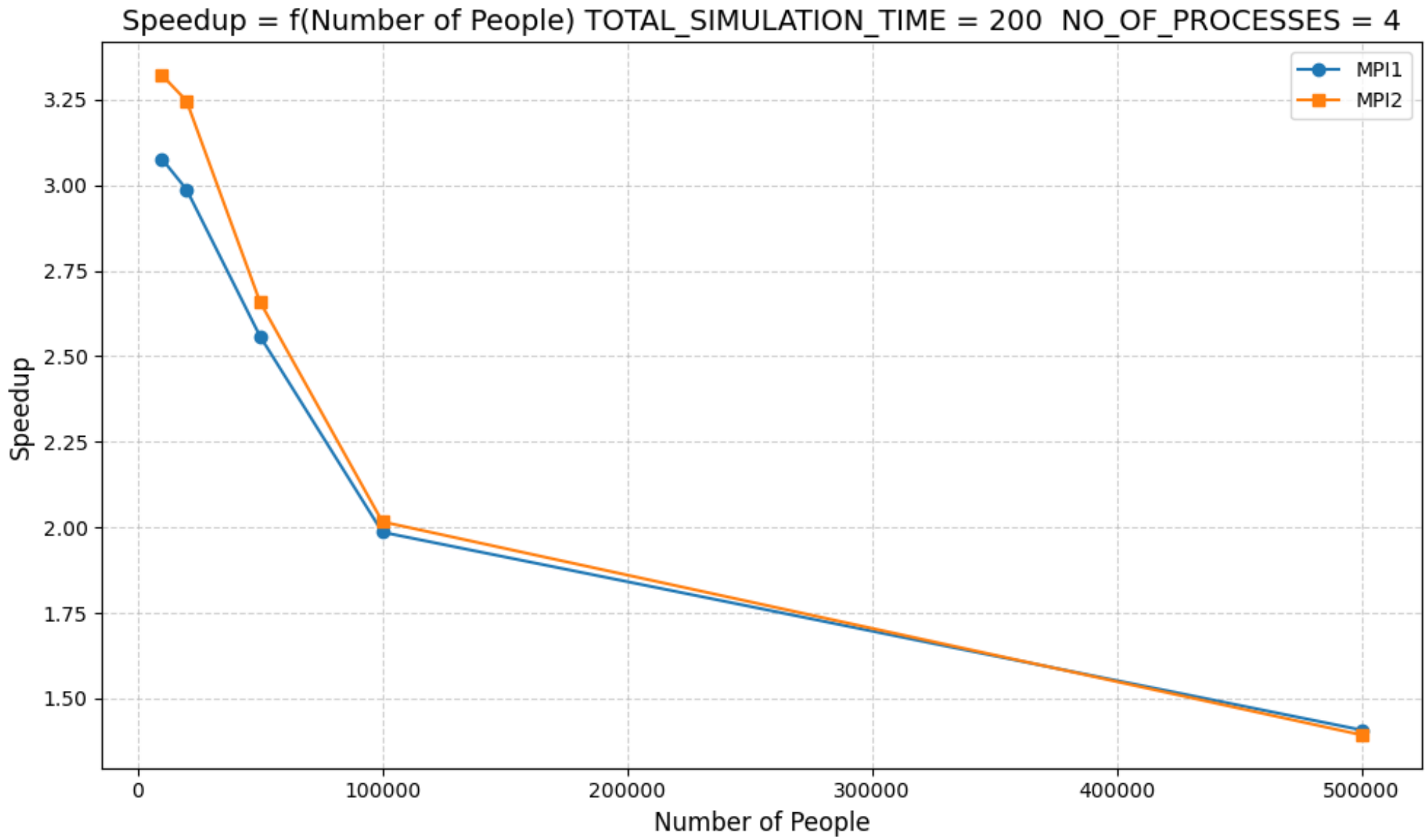
Functionality

- Both the serial and parallel algorithms use the same logic to solve the problem:
  - store the input information of the people in a custom array;
  - create an integer matrix that represents our movement grid;
  - at the beginning of each iteration we parse the array and set the points on the grid where we have located at least one infected person;
  - we then parse the array a second time to determine which of the people got infected during this iteration (by checking the infection array) and in the same loop we also update the state of each person, preparing them for the next iteration;
  - return the runtime of the functional part.
- Parallel details:
  - We use MPI_Scatterv to distribute work across processes (only the root performs the file read operation), this also targets the case where the number of people is not divisible by the number of processes;
  - In each iteration, after each process sets the points on the infection grid we perform an All to All Reduce on it with Logic OR in order to transfer the across all processes;
  - We use an equivalent MPI_Gatherv to retrieve the final state of the people array, only the root process performs the file write operation (Note that for all functions you need to define the flag SAVE_RESULTS during compilation if you wish for the operation to be performed);
  - There are two different iteration functions, one using only processes and the other one using threads to take advantage of the full capacity of the CPU.

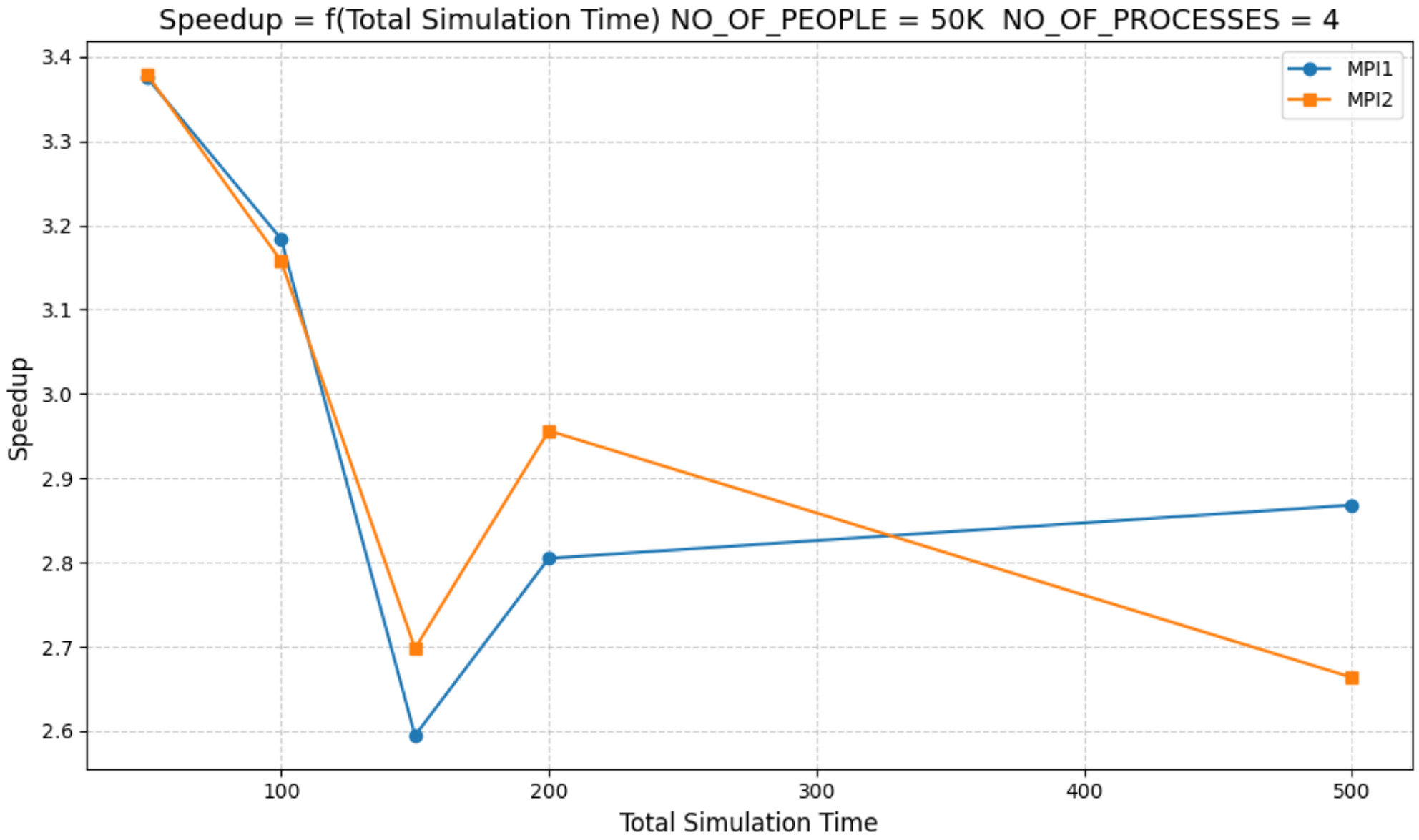
Project Structure

- The functions and data structures used in the epidemics simulation are defined inside of a custom C library "mpi_epidemics.h"
- The simulation program is found in the "simulation.c" file and it allows you to simulate one instance of the epidemics scenarios:
  - it takes the command line arguments <total_simulation_time> <input_file>;
  - prints the time values for the given scenario at the standard output.
- The benchmark program found in the "benchmark.c" file was used to calculate the speedup values used on the first two graphs, it should not be used with SAVE_RESULTS defined.
- Compilation & Execution:
  - mpicc -g -Wall -std=c99 -o exec <main C source> mpi_epidemics.c -fopenmp [-DSAVE_RESULTS]
  - mpiexec -n <no of processes> exec <total simulation time> <input file>

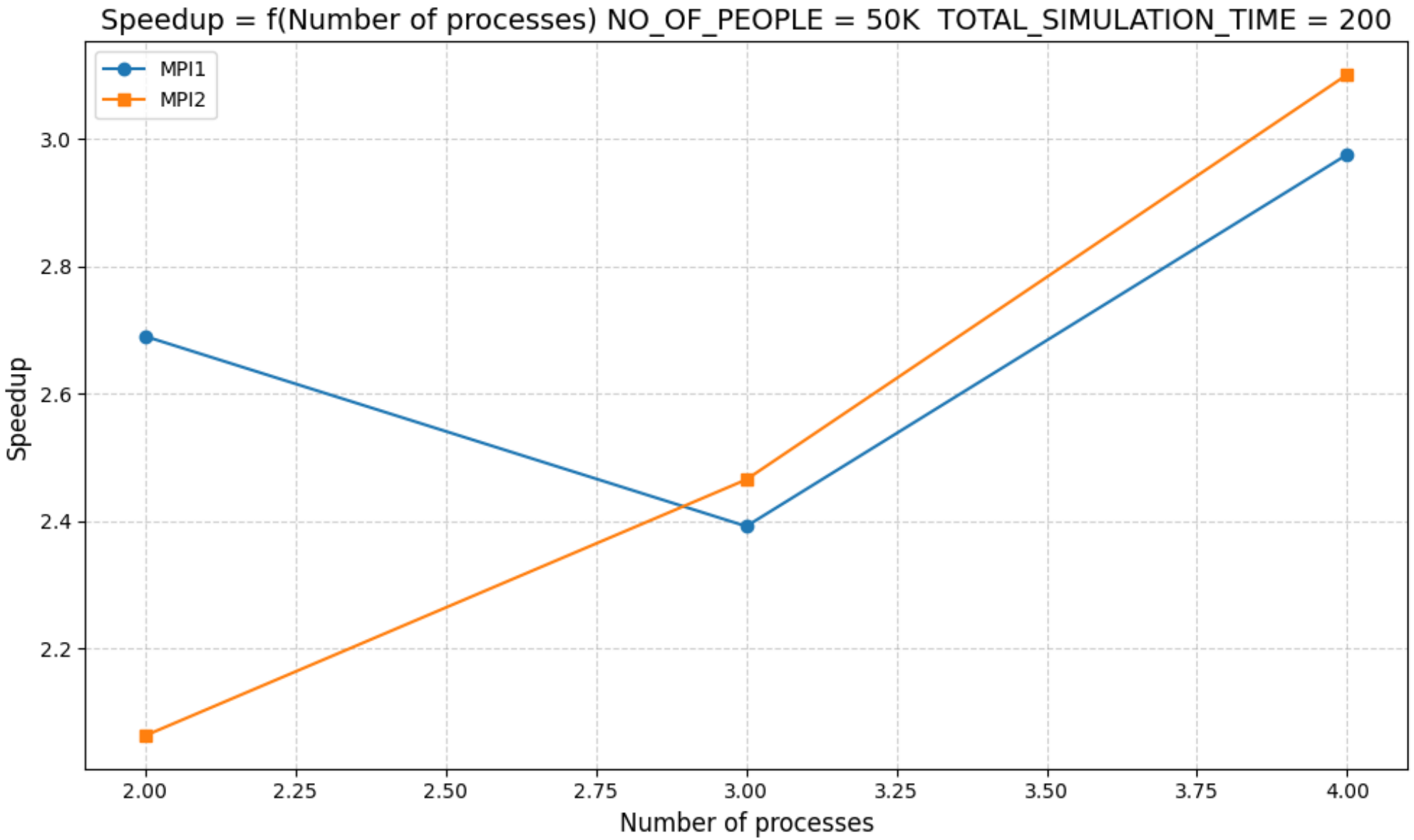
Test results 1



Test results 2



Test results 3



Conclusions

- Performance for the parallel version drops significantly with increasing population size due to increased communication between processes, but the speedup remains >1 even at 500k people.
- Performance for the parallel version drops when the simulation time exceeds 100 but remains rather stable afterwards.
- Performance of the parallel version with threads grows almost linearly with the number of processes while the regular version, despite performing better at first, take a dip in performance when the number of processes does not divide the number of people, its performance afterwards is slightly worse that the other version.
- Limitations:
  - Despite all the speedups being calculated on average values, we could use more computing power for more accurate results;
  - We could extend the number of tested scenarios so the graphs better represent the actual functions;
  - Testing should be done on a higher performing computer with more cores;

That's it, bye :0