MPI

AUTHORS: Garbarino Giacomo, Parmiggiani Manuel

INFN CLUSTER SPECIFICS

- CPU: Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz
- Number of cores: 256 (64 cores with 4 threads for each one in hyperthreading)

HOW TO EXECUTE A C PROGRAM USING MPI LIBRARY ON THE CLUSTER

1. mpiicc pi_parallelized_version.c -o pi_parallelized_version
2. mpiexec -hostfile machinefile.txt -perhost 32 -np 256 ./pi_parallelized_version

DISCUSSION

The program to parallelize implements the calculation of the pi through 100.000.000.000 iterations. The serial version of the algorithm takes too much time because the number of iterations is too big, so the solution lies in parallelization. To parallelize the problem with MPI, it's needed to add the instructions ***MPI_Init(&argc,&argv)***, ***MPI_Comm_size(MPI_COMM_WORLD,&nproc)*** and ***MPI_Comm_rank (MPI_COMM_WORLD,&rank)*** before the instruction ***sum=0.0***. Doing this, the MPI parallel section starts and an MPI environment is created, and all the following instructions are executed in parallel. Since the for loop is the hotspot, to parallelize it correctly, it's also necessary to change the iteration procedure with ***i=rank+1*** and ***i+=nproc*** to split up the iterated operations correctly among the 256 threads. Through this iteration, each thread calculates a partial part of the pi. In the end, with the instruction ***MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD)***, all the partial parts of the pi, contained in ***mypi***, are summed together and the final value is saved in the variable ***pi***. Once the value of ***pi*** is calculated, the thread with rank equal to zero prints it. The computed pi is equal to 3.14159265358979755689006 2, while the true value is equal to 3.14159265358979311599796 3. The two values are almost the same, so the parallelization is correct. The parallel computation takes about only 8.5 seconds. To measure the elapsed time with the instruction ***MPI_Wtime()***, the instruction ***MPI_Barrier(MPI_COMM_WORLD)*** at the beginning and at the end of the MPI section is necessary to ensure that all threads start together and to wait that each thread finishes. In the end, with the instruction ***MPI_Finalize()***, the MPI environment terminates.