Laborator 3

Programare paralela si concurenta

Ciprian Paduraru

Aplication: Consider that we have 2 arrays (denoted A and B) on master machine with type elements struct Vec2D. Ex:

struct Vec2D { float x, float y; }
const int N = 16;
Vec2D A[N];
Vec2D B[N];

// Random initialize A and B values...

We want to compute (in parallel) the sum of distances between each point in A and its corresponding point in B: $\sum \sqrt{(A[i].x - B[i].x)^2 + (A[i].y - B[i].y)^2}$.

(Solutions are implemented in the folder of this lab for reference. No copy-paste please)

Static parallelism

1. Compute the sum in parallel (P processors) by using Send and Recv primitives to splitting the initial chunk of work in equal parts (for simplicity, N mod P = 0) among processes, compute locally the sum then use Gather function to get partial results on master.

2. Modify previous program to use Scatter instead of Send/Recv.

3. Use **MPI_Reduce** to compute the sum from partial results instead of adding serially on master. Explain complexity of Reduce and how it's implemented.

Dynamic parallelism

4. What would happen with previous strategy if one machine is slower than all others ? Some of them will stay idle waiting for the slow one to finish !
   To implement a better load balancing, consider that we split the initial chunk in smaller parts (again, consider a number divisible by N for simplicity) and send task to idle processes.

   Master should manage a queue of tasks. When a process finishes his allocated job and sends back the results to master, master should send him another job (if there is any still left on its queue). Also, implement termination.

   Use message tags to identify the type of a message.

Master to slave should probably use: NEW_TASK, TERMINATE while slave to master: FINISHED_TASK. Define these with enums.