Parallel computing
Manuel Göster, Ilker Canpolat

Assignment 02: MPI - Message Passing Interface

Task 1: Getting started with MPI

used CPU: i7 4500U

a) Parallel image rasterization

Show how the number of processes affects the number of images produced within the same time frame:
command: mpirun -np $processes cpurender/cpurender -i input/cubes.obj -o output/cubes.png -w 1920 -h 1080 -d 2

processes / images produced	execution time	execution time / image	images / sec
1	real 0m0,390s user 0m0,233s sys 0m0,024s	0.39s	2.56
2	real 0m0,406s user 0m0,496s sys 0m0,072s	0.20s	4.93
3	real 0m0,577s user 0m1,085s sys 0m0,080s	0.19s	5.19
4	real 0m0,635s user 0m1,800s sys 0m0,101s	0.16s	6.29
8	real 0m1,183s user 0m3,584s sys 0m0,284s	0.15s	6.76
16	real 0m2,183s user 0m7,358s sys 0m0,503s	0.14s	7.33
32	real 0m4,256s user 0m14,506s	**0.13s**	**7.52**

	sys 0m1,155s		
64	real 0m8,943s user 0m30,019s sys 0m3,010s	0.14s	7.16
128	real 0m25,929s user 1m13,863s sys 0m20,619s	0.20s	4.94

What happens when you run your program with (far) more processes than you have cores in your CPU?
The performance shrinks because the OS has to save the process context every time a process ousts another process from a cpu core. These context changes are expensive. With 128 processes running in parallel the sys time rises to 20s!

How good does the MPI execution scale?
It scales well because the different processes do not have to communicate with each other. Thus it scales well up to 32 processes.

Which speedup do you achieve compared to rendering the same number of images consecutively in a single process?
compared a single process running 32 times in a row to running 32 processes in parallel:
single: 0.39s * 32 = 12.48s
parallel: 4.256s
speedup: 12.48s / 4.256s = **2.93x**

Show the changes you made in the source code for this task in your report (for instance using screenshots).
in main.cpp I added between line 28 and 29:

```
// custom code
int rank, size;
MPI_Init ( &argc, &argv );
MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
MPI_Comm_size ( MPI_COMM_WORLD, &size );
printf ( "Hello from process %d out of %d\n", rank, size );
// alter output name
output.append(std::to_string(rank));
// -----------
```

in rasterizer.cpp renderMeshFractal method:

```
void renderMeshFractal(
        std::vector<Mesh> &meshes,
        std::vector<Mesh> &transformedMeshes,
        unsigned int width,
        unsigned int height,
        std::vector<unsigned char> &frameBuffer,
        std::vector<float> &depthBuffer,
        float largestBoundingBoxSide,
        int depthLimit,
        float scale = 1.0,
        float3 distanceOffset = {0, 0, 0}) {
  // custom code
  int rank;
  MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
  float rotationAngle = (rank * 7) % 360;

  // Start by rendering the mesh at this depth
  for (unsigned int i = 0; i < meshes.size(); i++) {
    Mesh &mesh = meshes.at(i);
    Mesh &transformedMesh = transformedMeshes.at(i);
    //runVertexShader(mesh, transformedMesh, distanceOffset, scale, width, height);
    runVertexShader(mesh, transformedMesh, distanceOffset, scale, width, height, rotationAngle); // <------------ add rotationAngle
    rasteriseTriangles(transformedMesh, frameBuffer, depthBuffer, width, height);
  }
```

b) master rank

Show in your report how you implemented the MPI communication "protocol" (you can for example show screenshots of your code).
Show in your report how you implemented the MPI communication "protocol" (you can for example show screenshots of your code).
Show in your report how you implemented the MPI communication "protocol" (you can for example show screenshots of your code).
Show in your report how you implemented the MPI communication "protocol" (you can for example show screenshots of your code).

in main.cpp this has been added:

```
// custom code
int rank, size;
float rotationAngle = 0;
int tag = 1;
MPI_Status status;

MPI_Init ( &argc, &argv );
MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
MPI_Comm_size ( MPI_COMM_WORLD, &size );
printf ( "Hello from process %d out of %d\n", rank, size );
// alter output name
output.append(std::to_string(rank));

// I am the master
if (rank == 0) {
  // send individual rotationAngle to all processes
  for (int dest = 1; dest < size; dest++) {
    float individualRotationAngle = (dest) % 360;
    MPI_Send ( &individualRotationAngle, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD );
  }
// I am a slave
} else {
  // wait for receiving individual rotation angle
  MPI_Recv ( &rotationAngle, 1, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, &status );
  printf ( "process %d received rotationAngle %f\n", rank, rotationAngle );
}
// ----------
```

and the line that calls the rasterise function was changed to:
std::vector<unsigned char> frameBuffer = rasterise(meshs, width, height, depth, **rotationAngle**);
in rasteriser.cpp and rasteriser.hpp:
some function declarations had to be changed in order to pass the rotationAngle through the functions.

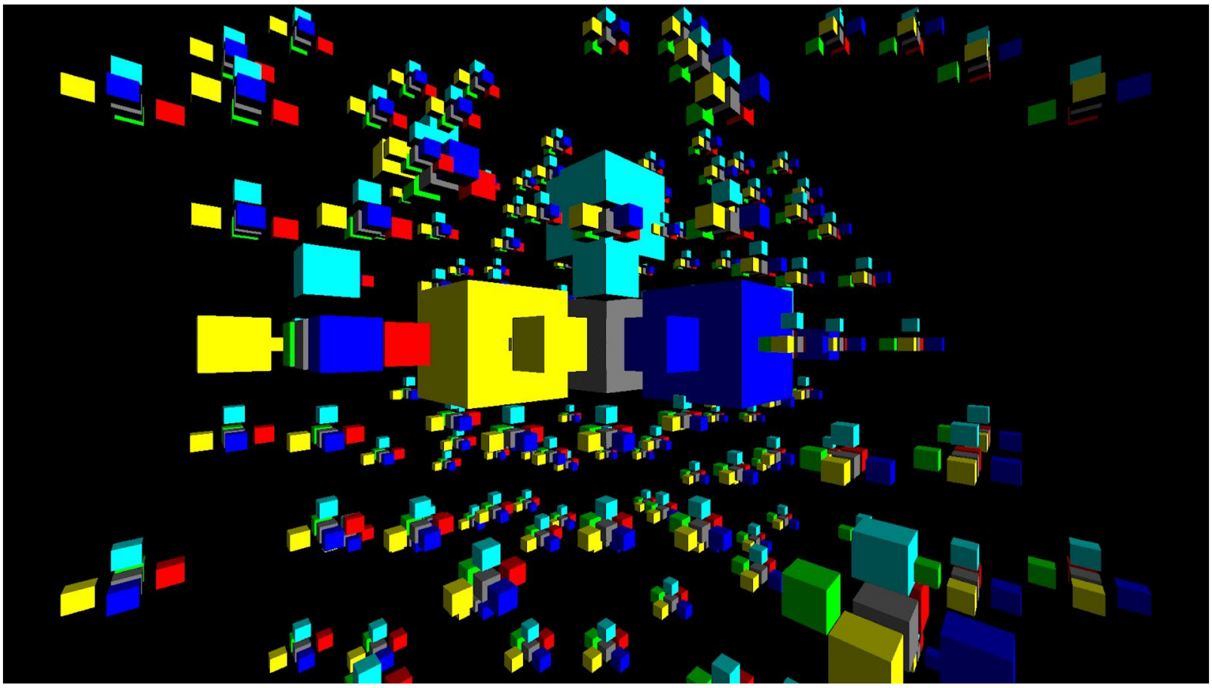
c) broadcast

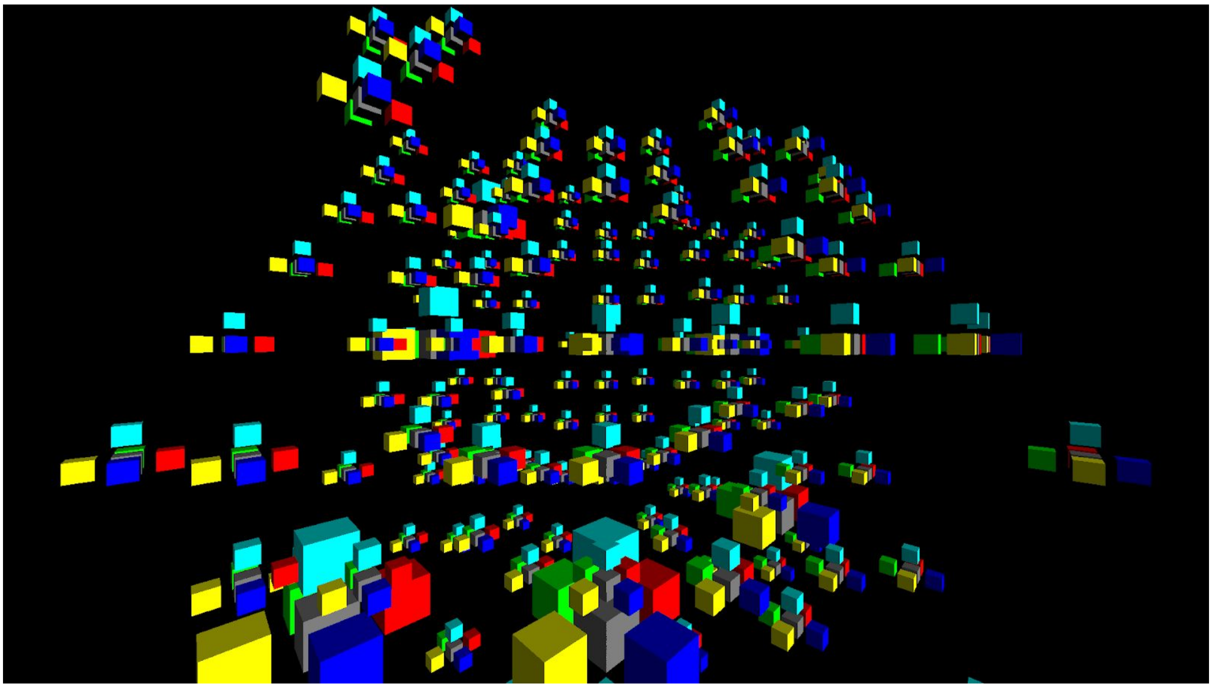
```
std::cout << "Loading '" << input << "' file... " << std::endl;

std::vector<Mesh> meshs = loadWavefront(input, false);

// custom code
// create float3 mpi datatype
MPI_Datatype MPI_FLOAT3;
{
  int count = 3;
  int array_of_blocklengths[] = {1,1,1};
  MPI_Aint array_of_displacements[] = {offsetof(float3, x), offsetof(float3, y), offsetof(float3, z)};
  MPI_Datatype array_of_types[] = {MPI_FLOAT, MPI_FLOAT, MPI_FLOAT};
  MPI_Type_create_struct( count, array_of_blocklengths, array_of_displacements,
                          array_of_types, &MPI_FLOAT3 );
  MPI_Type_commit(&MPI_FLOAT3);
}

// create float4 mpi datatype
MPI_Datatype MPI_FLOAT4;
{
  int count = 4;
  int array_of_blocklengths[] = {1,1,1,1};
  MPI_Aint array_of_displacements[] = {offsetof(float4, x), offsetof(float4, y), offsetof(float4, z), offsetof(float4, w)};
  MPI_Datatype array_of_types[] = {MPI_FLOAT, MPI_FLOAT, MPI_FLOAT, MPI_FLOAT};
  MPI_Type_create_struct( count, array_of_blocklengths, array_of_displacements,
                        array_of_types, &MPI_FLOAT4 );
  MPI_Type_commit(&MPI_FLOAT4);
}
```

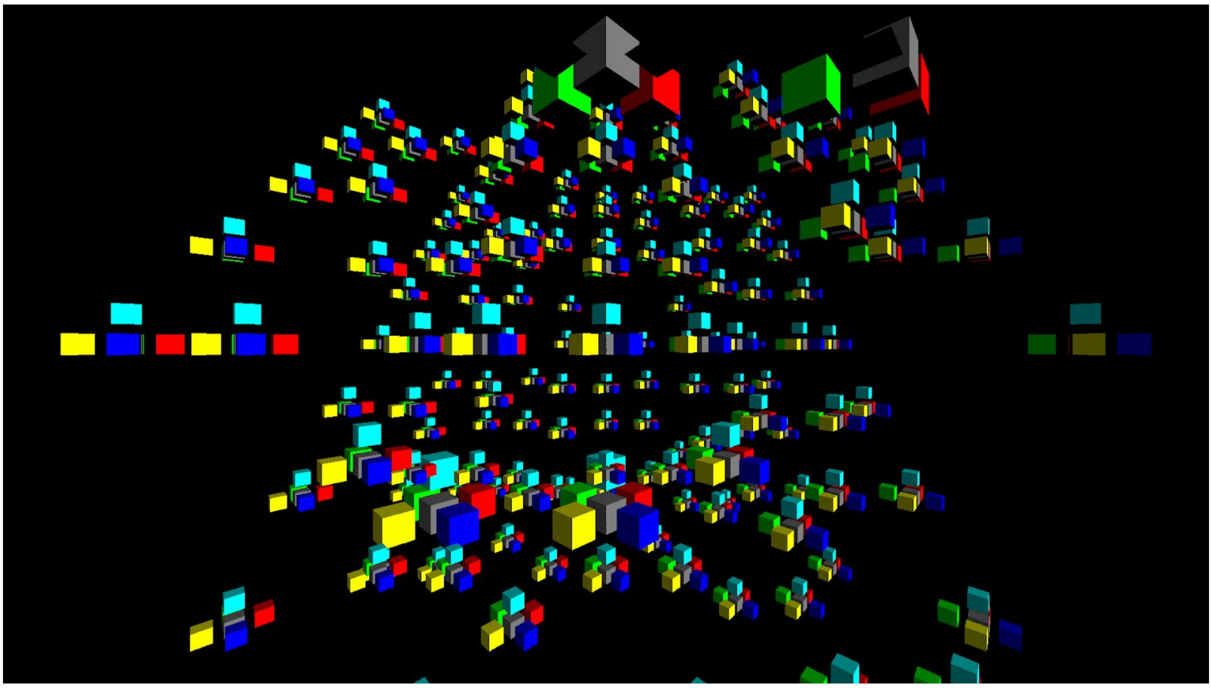
```
// as a slave
// throw out the contents of the vertices, normals and texture coordinate vectors
if (rank != 0) {
  for (auto &mesh : meshs) {
    for (auto &val : mesh.vertices) {
      val = float4{};
    }
    for (auto &val : mesh.textures) {
      val = float3{};
    }
    for (auto &val : mesh.normals) {
      val = float3{};
    }
  }
}

// broadcast memory of master to all slaves
for (int dest = 1; dest < size; dest++) {
  for (auto &mesh : meshs) {
    for (auto &val : mesh.vertices) {
      MPI_Bcast(&val, 1, MPI_FLOAT4, 0, MPI_COMM_WORLD);
    }
    for (auto &val : mesh.textures) {
      MPI_Bcast(&val, 1, MPI_FLOAT4, 0, MPI_COMM_WORLD);
    }
    for (auto &val : mesh.normals) {
      MPI_Bcast(&val, 1, MPI_FLOAT4, 0, MPI_COMM_WORLD);
    }
  }
}
// ----------
```

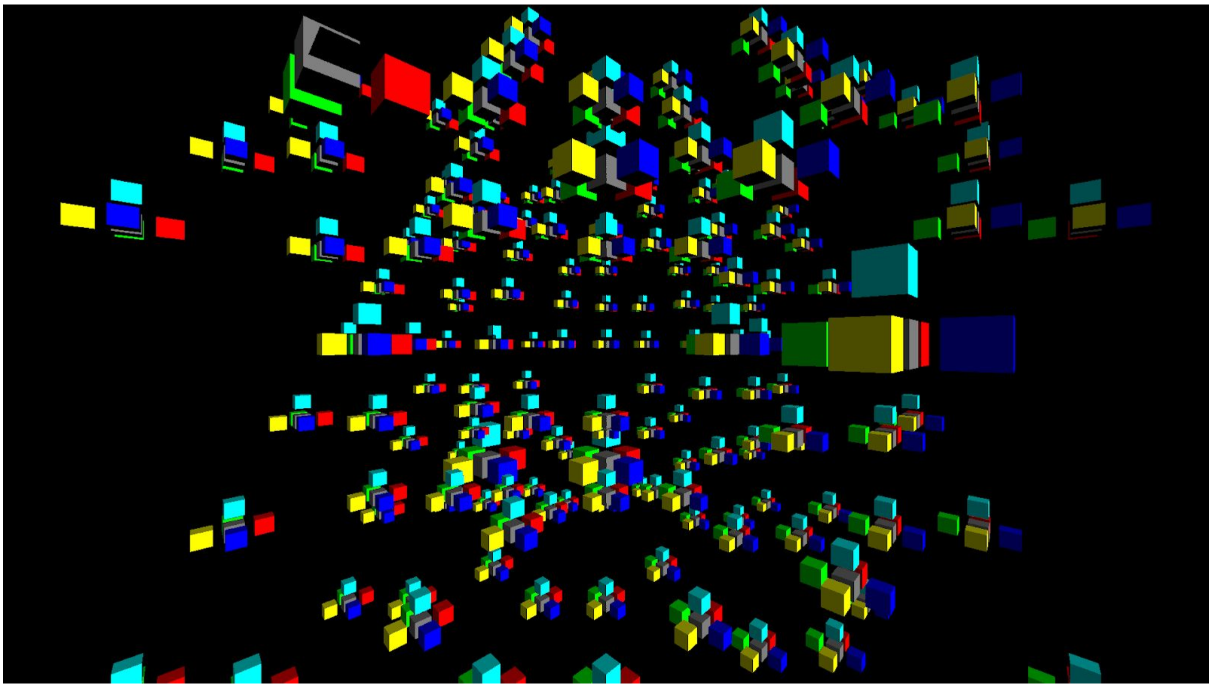
Task 2: Collective MPI computation

a) Collective Construction









c)
Measure the execution time of your current implementation. How does it scale with varying numbers of MPI ranks? What happens to the execution time when you launch more ranks than you have cores in your CPU?
ran on Intel® Core™ i7-7700K CPU @ 4.20GHz × 8
command: mpirun -np $processes cpurender/cpurender -i input/plant.obj -o output/plant.png -w 1920 -h 1080 -d 2

processes	execution time
1	real 0m9.106s user 0m8.741s sys 0m0.140s
2	real 0m5.031s user 0m9.207s sys 0m0.157s
3	real 0m3.736s user 0m9.811s sys 0m0.211s
4	real 0m3.091s user 0m10.251s sys 0m0.283s
8	**real 0m2.998s** **user 0m18.921s** **sys 0m0.823s**
16	real 0m4.006s user 0m24.589s sys 0m2.894s
32	real 0m5.342s user 0m34.529s sys 0m4.274s
64	real 0m8.555s user 0m55.022s sys 0m8.375s
128	real 0m15.352s user 1m39.023s sys 0m17.333s

It scales well until eight processes, because the CPU consists of eight subcores.
With more MPI ranks than cpu cores, the execution time rises again, because the os has to schedule the process-cpu times.

Parallel computing
Manuel Göster, Ilker Canpolat

Compare your measured runtime to the average execution time needed for computing a single image (which you measured in task 1a)). In both cases, use a number of MPI processors equal to the number of cores in your CPU.

ran on Intel® Core™ i7-7700K CPU @ 4.20GHz × 8

time mpirun -np 8 cpurender/cpurender -i input/plant.obj -o output/plant.png -w 1920 -h 1080 -d 3

1a):
real 0m15.348s
user 1m53.839s
sys 0m0.716s

2c):
real 0m2.998s
user 0m18.921s
sys 0m0.823s

What is the speedup of the cooperative construction of images over a single-threaded approach and briefly speculate on what could cause that approach to be faster than the other.

Speedup: 15.348 / 2.998 = 5.119x

Why this is faster: We separated the workload of the renderMeshFractal() function over multiple processes, running in parallel. This is way faster than using just one process. In addition, the computed reducing does not come with a big overhead.renderMeshFractalrenderMeshFractalrenderMeshFractal