Deliverable D3.3

1st Version of MESHFREE

2022-09-06

Work package:	WP3 Development of 1st Version of Applications	
Author(s):	Almut Eisenträger	FRAUN
	Sebastian Fett	FRAUN
	Elisa Thiel	FRAUN
Reviewer #1	Manuel Arenaz	APP
Reviewer #2	Max Engelen	MAX
Dissemination Level	Public	
Nature	Report	

Confidentiality
This document may contain proprietary material of certain OPTIMA contractors. The commercial use of any information contained in this document may require a license from the proprietor of that information.

EuroHPC Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU) under grant agreement No 955739. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Greece, Germany, Italy, Netherlands,Spain,Switzerland.

Executive Summary

The software MESHFREE is used for simulations of computational fluid mechanics and more general continuum mechanics problems in industrial contexts. Its underlying numeric methods are based on describing the computation domain as a distributed point cloud, which moves with the same speed as the simulated material. For stability of the method, it is crucial to keep the point cloud quality consistent throughout the simulation. Points are regularly removed and added where necessary to guarantee point cloud quality and to allow for flexible refinement strategies. The point cloud organization tasks have to be run in every simulated time step and are potential hotspots in the code, depending on the precise setup being considered. Potentially millions of these points participate in the simulation and thus need to undergo organization tasks. It is also the part of the code most suited for running on FPGAs. In this deliverable, we provide details of the parts of the point cloud organization that were ported to FPGA systems for accelerated computation. In this first step, the routines are run separate from the main MESHFREE code. The full interface will be considered in WP5.

Table of Contents

List of Abbreviations

FPGA	Field Programmable Gate Array

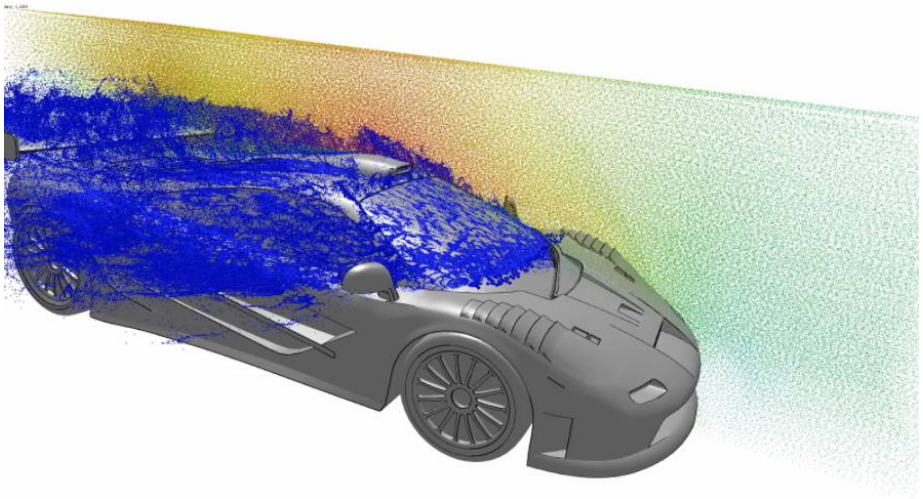
1. Introduction

The simulation software MESHFREE [1] has been developed with regard to industrial applications at the Fraunhofer Institute for Industrial Mathematics ITWM since 1999 and jointly with the Fraunhofer Institute for Algorithms and Scientific Computing SCAI since 2015. It is a meshfree approach to simulate flows and continuum mechanic processes. By not using a numeric mesh the method is versatile and efficient regarding simulations with moving boundary elements, free surfaces, interphases, and fluid-structure-interaction.

MESHFREE allows for flexible model development and thus enables a wide spectrum of applications, including large-scale applications in the context of safety-relevance and sustainable energy, such as:

- Virtual water management
- Simulation of vehicle rollover in a crash situation
- Hydropower

To model these applications in the industrial context, it is essential to efficiently handle large amounts of data, complex input data and detailed simulation results. Input geometries of more than a million boundary elements are standard and more than 5 million discretization points may be needed to achieve adequate precision for complex load cases. In addition, the simulation process is computationally expensive as MESHFREE employs implicit methods (solutions of linear systems of equations).



2. Definition of tasks for hardware acceleration

In MESHFREE, the computational domain is defined by the volume that is filled by the material to be simulated. The domain is covered with a point cloud made up of mass-less numerical points. The points are distributed evenly according to a given smoothing length definition, which sets the interaction radius between points and thus describes the discretization accuracy. This smoothing length can be defined to change temporally and spatially and may even depend on the simulation results.
In the Lagrangian approach, which is the default for MESHFREE, points are moved in each time step according to the velocity just calculated for the material at this location. That way, the changing shape of the domain, especially free surfaces or phase boundaries, is tracked directly. Due to this movement and the changing discretization requirements, the quality of the point cloud may locally deteriorate. To counteract this, the point cloud needs to be updated regularly. Points that get too close to each other are merged and holes in the point cloud need to be filled with new points. The tasks associated with this are summarized under pointcloud organization.
The MESHFREE code basis is completely written in Fortran as this is very suited for efficient large scale numeric computation. One drawback of using Fortran is that the interface to the FPGA cannot be defined directly - an interfacing over multiple different high-level languages has to be designed and tested. This is the prerequisite for any future implementation and portation work.
As described above, there are multiple tasks within the pointcloud organization. As a prototype for porting we choose the routine for seeking holes - subroutine `SeekHoles`. We expect that porting efforts for the other tasks can profit from the prototype as the data structures and logic are similar from the complexity.
The subroutine `SeekHoles` searches for holes in the point cloud and defines the locations for new points to be added. The algorithm is called for each existing point and works in a local coordinate system scaled by a certain hole radius dependent on the smoothing length. The central point defines several candidate locations evenly distributed on the surface of a sphere — or half-sphere in case of boundary points — centered at this point. For each of these candidate locations, the distance to all the nearest neighbors of the central point is computed. If the distance is too small, the candidate location is discarded. If the candidate location is sufficiently far from all the neighbor points, and by design from the central point, then these coordinates are returned as one member of a list of new point candidates for further processing.
So far, we have ported the innermost loop for the distance computation between the candidate locations and the existing neighbors to FPGAs, but we plan to extend this to larger parts of the `SeekHoles` subroutine during WP5. We hope to be able to restructure the subroutine so that it can even be ported as a whole. This would significantly reduce the communication overhead and offer an overall speedup, given that the `SeekHoles` routine is called for each of the potentially millions of points in each simulation time step.

3. Implementation details

The MESHFREE code is written largely in Fortran. We did not succeed in interfacing directly from Fortran to FPGA. Hence, we re-implemented our SeekHoles algorithm as a C routine called from Fortran via interfacing. The innermost loop, which is perfectly parallelizable, is offloaded onto the FPGA. Data is streamed between the C code and the FPGA. The offloaded innermost loop reads 3 arrays representing the neighbors of the current point, calculates the distance to the new point candidate and returns this to the C routine.

The original Fortran innermost loop is implemented as follows, where `r_xT`, `r_yT` and `r_zT` contain the x-, y- and z-coordinates, respectively, of the neighbors of the central point, while `Dnborn` is the number of neighbors considered.

```
subroutine SeekHoles( . . . )
     ...
       do j = 1, number_checks
       ...
          do k = 1,Dnborn
              dist(k) = &
                   ( r_xT(k)-p_check(1) )**2 + &
                   ( r_yT(k)-p_check(2) )**2 + &
                   ( r_zT(k)-p_check(3) )**2
              if ( dist(k) .lt. DRR ) then
                 flag = 1
                 n_cycle = k
                 goto 70
              end if
            end do
            n_cycle = Dnborn
 70         continue
       ...
       end do
     ...
end subroutine
```

The following class `distCalcKernel` is implemented for the maxj FPGA. It takes the input streams `r_xT`, `r_yT` and `r_zT`, the input scalars `p_check_0`, `p_check_1` and `p_check_2` and computes the loop described above in a streaming manner.

```
class distCalcKernel extends Kernel {

    distCalcKernel(KernelParameters parameters) {
       super(parameters);

       DFEType scalarType = dfeFloat(11, 53);

       DFEVar r_xT = io.input("r_xT", scalarType);
       DFEVar r_yT = io.input("r_yT", scalarType);
       DFEVar r_zT = io.input("r_zT", scalarType);

       DFEVar p_check_0 = io.scalarInput("p_check_0", dfeFloat(11, 53));
       DFEVar p_check_1 = io.scalarInput("p_check_1", dfeFloat(11, 53));
       DFEVar p_check_2 = io.scalarInput("p_check_2", dfeFloat(11, 53));

       DFEVar dist =
          ( r_xT-p_check_0 )*( r_xT-p_check_0)
   +      ( r_yT-p_check_1 )*( r_yT-p_check_1)
   +      ( r_zT-p_check_2 )*( r_zT-p_check_2);

       io.output("dist", dist, dfeFloat(11, 53));
    }
}
```

This needs to be linked from C to the FPGA via the class `passthroughManager` below.

```
public class passthroughManager extends  MAX5CManager{

    public passthroughManager(EngineParameters params) {

           super(params);
           Kernel kernel =
           new distCalcKernel(makeKernelParameters("distCalcKernel"));
           KernelBlock kernelBlock = addKernel(kernel);

           kernelBlock.getInput("r_xT") <== addStreamFromCPU("r_xT");
           kernelBlock.getInput("r_yT") <== addStreamFromCPU("r_yT");
           kernelBlock.getInput("r_zT") <== addStreamFromCPU("r_zT");

           addStreamToCPU("dist") <== kernelBlock.getOutput("dist");

    }

    public static void main(String[] args) {
           passthroughManager manager =
           new passthroughManager(new EngineParameters(args));
           manager.build();
    }
}
```

4. Concluding remarks

Due to interfacing over multiple different high level languages, the design of the interfacing and the initial steps of the implementation were more complex and thus took longer than anticipated. This, unfortunately, led to delays in the work package and resulted in the first implementation being much smaller than intended. As this is highly unfavorable for FPGA due to the large communication overhead for a currently small calculation, performance evaluation is not meaningful at this time and will be delayed.
In WP5 we anticipate porting a larger part of the seek-holes-algorithm to the FPGA. Currently, the calculation is done separately for every single discretization point. In order to avoid the overhead, meaningful chunks of points should be processed together whenever possible. Only when this is achieved, other point cloud organization tasks can be ported.

5. References

[1] MESHFREE website https://www.meshfree.eu/. Accessed August 30, 2022.