Rashed Almasri **NE533** **03/29/2024**

Project Part 2

Introduction

In this part of the project, new parameters are introduced. Instead of working on the problem as an essential 1D problem, we start to see it as a more realistic 2D problem, taking into account the axial profile of the temperature, and the temperature change due to the coolant flow rate on the cladding outer surface.

These changes, as mentioned above, add a new dimension, and multiple new challenges as well. Mainly when it comes to calculating the BCs for the cladding, as we need to change some of the definitions and new improved functions.

This is a small introduction, and before we dive into our input file and the analysis of the output files, we will go into a small recap of part 1.

Part 1 Recap

In part 1, the code was missing half of the objective, a time-dependent solution to the problem. That wasn't as easy as expected, as we needed to introduce new material properties like density and specific heat, as well as enter the modules for the transient problems between the kernels, functions, and executioners.

So for the heat generation function, we replaced the ConstantFunction with the ParsedFunction module, as we wrote the time-dependent heat generation function. Additionally, we modify the HeatSource kernel to include the aforementioned function, as well as introduce a new kernel; HeatConductionTimeDerivative to operate with the temperature as a transient property.

In the materials module, we change the module into GenericConstantMaterial and add specific heat and density properties to all materials included. Finally, we modified the executioner to Transient, and we added convergence criteria as it proved to be challenging. I also added a TimeStepper block to modify the time step depending on the output which increased the speed of the code overall.

Additionally, one of the comments in the previous part was on the convergence of the mesh. While the number of elements was written as a square in the initial code, I realized that was unnecessary. Yes, we need many elements on the x-axis to simulate the blocks and the temperature change. But on the y-axis, it's a 1D problem here, and even one element would work, but I adjusted it to 10 elements here.

The output file can be seen in Figure 1.

The modified code can be found in Appendix 1 of this report.

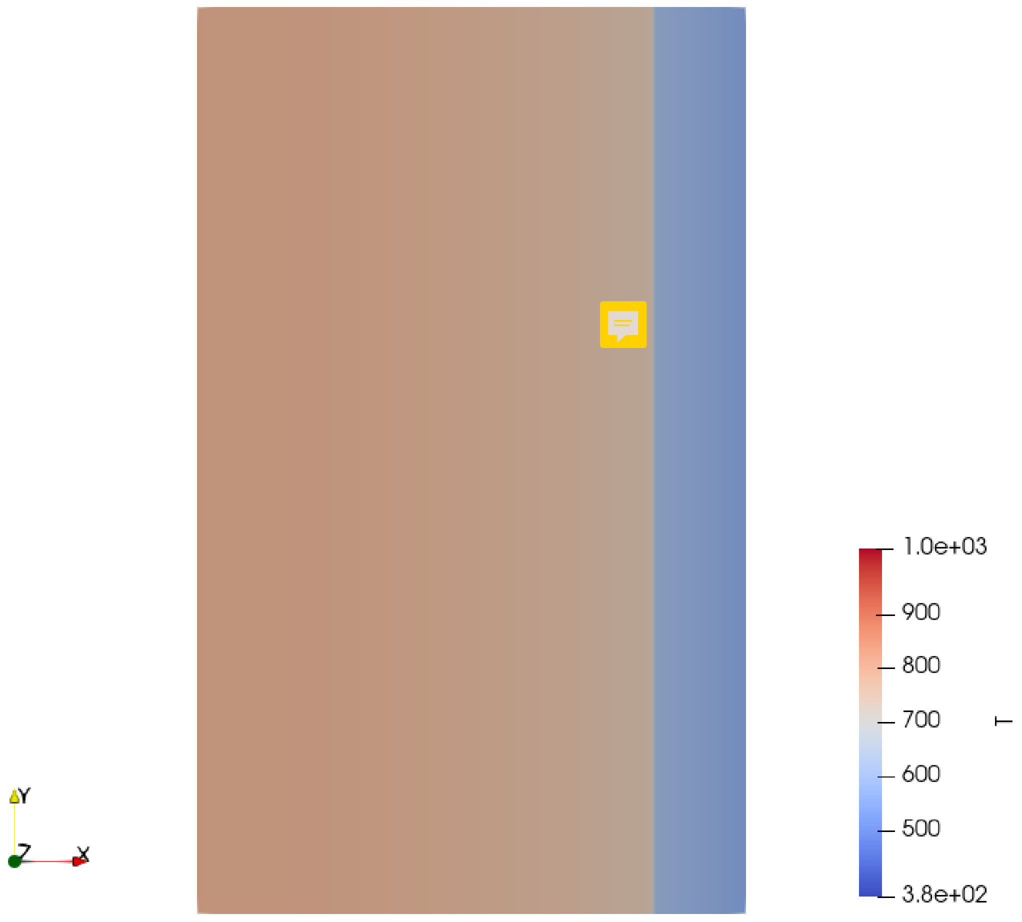

Figure 1: Part 1 Time-dependent output

The Geometry & Variable

A similar approach from Part 1 is implemented here. The total length of the mesh is 100 as the mesh should be 1 m long. Elements for the x-axis are 500 to give more accurate dimensions for the blocks, but on the y-axis, ten elements should be sufficient.

For each block, SubdomainBoundaingBoxGenerator is used to define the blocks for the cladding, gap, and fuel. The coordinate type is RZ with the RZ axis being Y.

Functions, Variables & Kernels

This time, we need to define two different functions. Both will be ParsedFunction, as both are changing with z (or in our mesh, y). The first one is the heat generation function. Using the equation:

```
LHR0*cos(A*(y/Z0-1))/(pi/(2^2))
```

where LHR0 is 350, A is 1.2083 and Z0 is 0.5 m (50 cm). Note here that A is the calculated value of 2(1.1)/pi.

The other function will be the temperature profile, which describes the change in the temperature of the coolant due to the flow rate alongside the cladding surface. The equation is as follows:

```
Tin+((1/1.2)*((Z0*LHR0)/(mdot*cpw))*(sin(1.2)+sin(1.2*((y/Z0)-1))))
```

Where Tin is the inlet coolant temperature at 500 K, mdot is the flow rate of 0.25 kg/s-rod and cpw is the specific heat of the coolant at 4200 J/kg-K.

For the variables, we are only concerned with one variable which is the Temperature T.

The kernels here are very simple. Just two kernels, one for the heat source using the function for the heat source and only active in the fuel block. And one for the heat conduction throughout the entire generated mesh.

Boundary Conditions and Materials

Boundary conditions here are different. We need to prepare two BCs, one for the centerline temperature which is a maximum (derivative = 0). This one is NeumannBC, with variable T and a value of 0, the boundary is on the left. The second one is a constant value of temperature on the right side, but it should be changing alongside the y-axis as well, therefore, we need to implement a FunctionDirichletBC and use the function we wrote above for the temperature profile.

As for the materials, nothing changes from the previous problem.

Problem, Executioner, VectorPostprocessors and Outputs

The problem will be FEProblem, and a new Preconditioning of type SMP is written to counter some earlier issues that were occurring due to a divergence of the code, as there were doubts about the Jacobian of the overall input file.

The executioner is Steady, but now to better analyze the data, a VectorPostprocessor has been added with the LineValueSampler type to measure the temperature profiles of interest and give an output of CSV files.

Finally, for the output, the exodus for paraview is set at true, and CSV is set to execute with the final results.

Output File and Comparison

Now let us see the output file for the temperature, as seen in Figure 2. As we can see the temperature is at maximum somewhere around the center. We can see how the temperature also slightly increases for the cladding, but not as much.

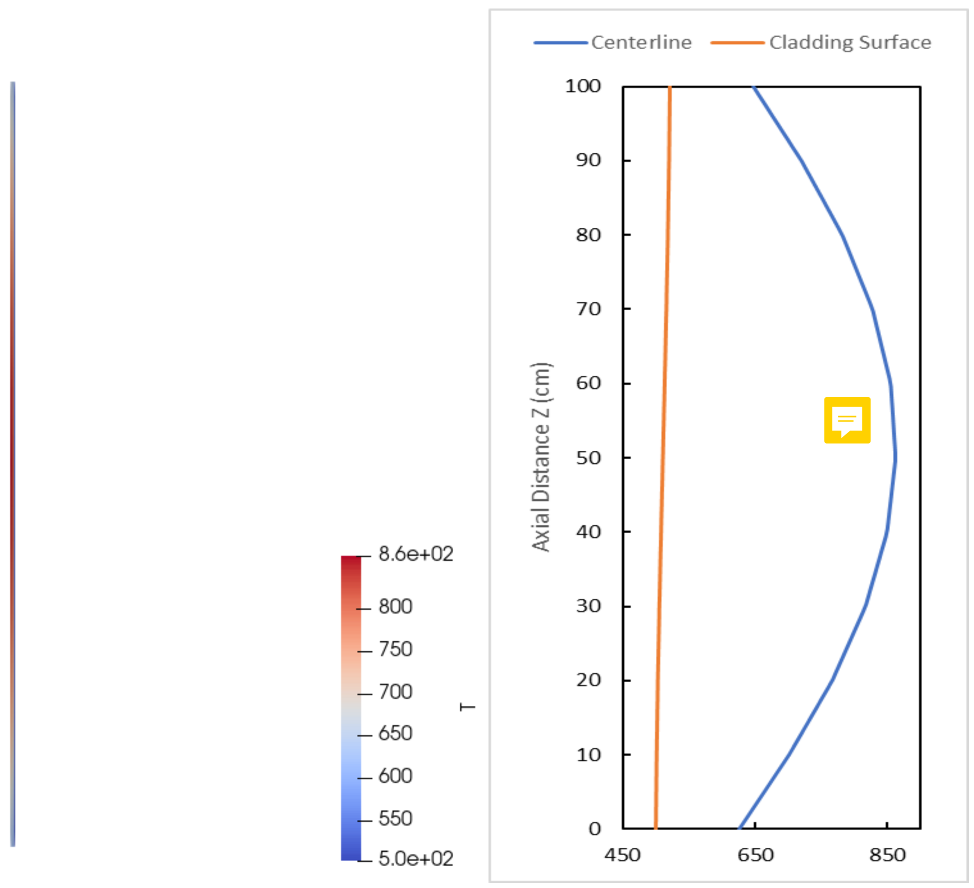

Figure 2: Temperature axial profile for both the cladding and the centerline.

Note from the CSV output file which is also included with the submission, the maximum temperature is 862.0355 K at exactly the middle of the rod or Z0 = 0.5 m.

Now for the temperature profiles at different points on the z-axis (y-axis), in Figure 3, we can see the temperature profiles at z = 0.25, 0.5, and 1 m. We can see that the maximum of the centerline is one at 0.5 m, and the temperature profile near the cladding is highest at the upper edge of the rod. Which makes sense and agrees with the expected outcome we had.

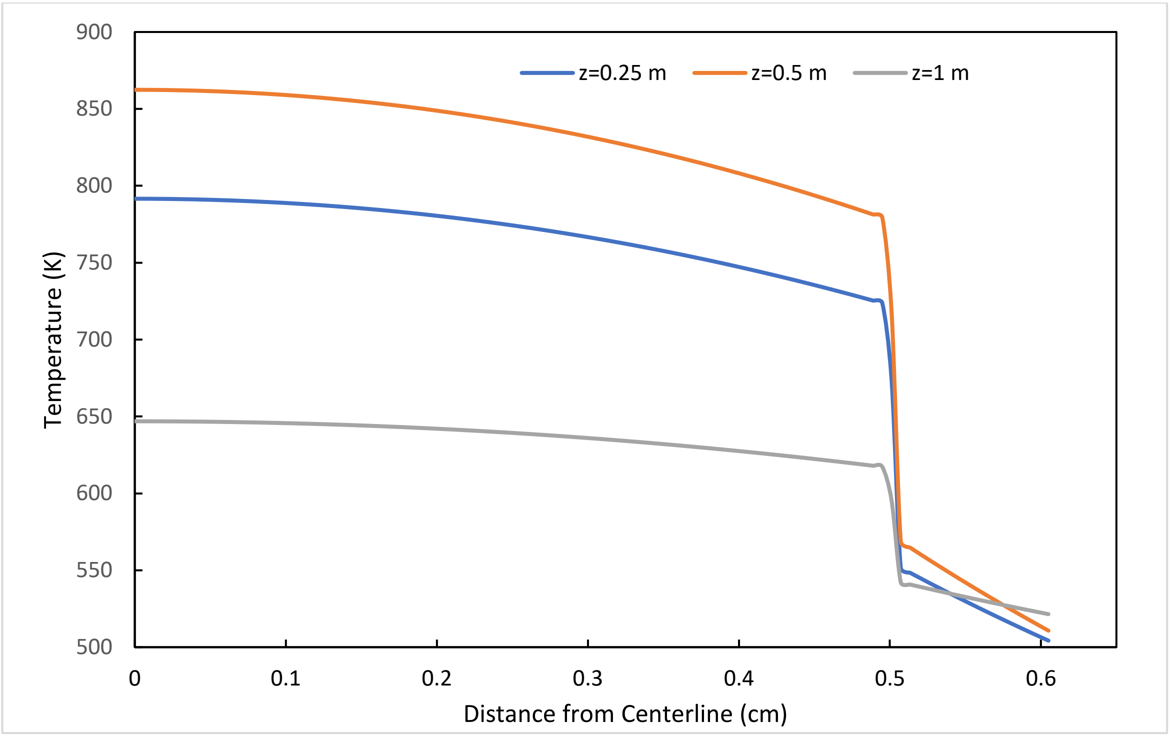

Figure 3: Temperature profiles for z = 0.25, 0.5, and 1 m.

Finally, we had a much more interesting and I would say realistic output for this part of the project. The temperature profiles behaved as expected in these cases, and we faced some interesting challenges we have faced, especially in the convergence of the code which was due to the function layout eventually. .

Appendix 1

```
[Mesh]
    [total]
        type = GeneratedMeshGenerator
        dim = 2
        nx = 250
        NY = 10
        xmax = 0.605
        ymax = 1
    []
    [gap]
        type = SubdomainBoundingBoxGenerator
        input = total
        bottom_left = '0.5 0 0'
        top_right = '0.505 1 0'
        block_id = '1'
    []
    [fuel]
        type = SubdomainBoundingBoxGenerator
        input = gap
        bottom_left = '0 0 0'
        top_right = '0.5 1 0'
        block_id = '2'
    []
[]

[Functions]
    # [generated_heat]
    #     type = ConstantFunction
    #     value = 445.6338
    # []
    [heat]
        type = ParsedFunction
        expression = 318.3098*exp(-((t-20)^2)/10)+150
    []
[]

[Variables]
    [T]
        initial_condition = 300
    []
[]
```

```
[Kernels]
    [time_conduct]
        type = HeatConductionTimeDerivative
        variable = T
    []
    [time_heat]
        type = HeatSource
        block = 2
        variable = T
        function = heat
    []
    # [heat]
    #     type = HeatSource
    #     variable = T
    #     function = generated_heat
    #     block = 2
    # []
    [conduction]
        type = HeatConduction
        variable = T
    []
[]

[BCs]
    [left]
        type = NeumannBC
        variable = T
        boundary = left
        value = 0
    []
    [right]
        type = DirichletBC
        variable = T
        boundary = right
        value = 550
    []
[]

[Materials]
    [fuel]
        type = GenericConstantMaterial
        prop_names = 'thermal_conductivity specific_heat density'
        prop_values = '0.33 0.33 10.98'
        block = 2
    []
    [gap]
        type = GenericConstantMaterial
        prop_names = 'thermal_conductivity specific_heat density'
        prop_values = '0.0026 5.19 0.0001786 '
        block = 1
    []
    [clad]
        type = GenericConstantMaterial
        prop_names = 'thermal_conductivity specific_heat density'
        prop_values = '0.17 0.285 6.56'
        block = 0
    []
[]

[Problem]
    type = FEProblem
[]

[Executioner]
    # type = Steady
    type = Transient
    solve_type = PJFNK
    end_time = 100
    nl_rel_tol = 1e-10
    nl_abs_tol = 1e-10
    automatic_scaling = true
    compute_scaling_once = false
    petsc_options_iname = '-pc_type -pc_hypre_type'
    petsc_options_value = 'hypre boomeramg'
    [TimeStepper]
        type = SolutionTimeAdaptiveDT
        dt = 1
    [][]

[Outputs]
    exodus = true
[]
```