Assignment in the Finite Element Method 2020

Division of Solid Mechanics

Isabelle Frodé & Sara Enander
Date: May 26 2020

Contents

1 Introduction **2**

2 Procedure **4**
2.1 Stationary Temperature Distribution 4
2.2 Transient Temperature Distribution 5
2.3 Thermal Expansion 6

3 Results **8**
3.1 Stationary Temperature Distribution 8
3.2 Transient Temperature Distribution 8
3.3 Thermal Expansion 10

4 Discussion **12**

5 Bibliography **14**

1 Introduction

This project investigates the temperature distribution inside an integrated circuit and to what extend the generated heat causes a thermal expansion to the circuit. The problem was solved and analyzed after creating a finite element program in Matlab using the CALFEM toolbox.

As the COVID-19 outbreak have spread, the data usage has increased, forcing Netflix to reduce their streaming quality with 25 %. The generated heat, Q, in the die is assumed to be proportional to the data consumption. Streaming full quality, it is assumed $Q = 5 \cdot 10^7$ W/m^3.[3]

The first part of the project deals with the stationary temperature distribution, by creating a solver in Matlab after dividing the model in finite elements. The second part of the project investigates how the temperature distribution changes in time the first twenty minutes after starting streaming. Further on, the displacement field and the von Mises stress field, following from thermal expansion in the circuit, is determined and analyzed, giving an idea of how the temperature cause the circuit to expand.

A simplified model of the circuit, presented in Figure 1, was used to analyze the problem. The model consists of three different components, a package made from silver filled epoxy, a die made from silicon and a frame made from copper. The material data could be gathered from the *Assignment in The Finite Element Method 2020* problem discription.[3]

	ag-epoxy	silicon	copper
E (Young's modulus) [Gpa]	7	165	128
ν (Poisson's ratio) [-]	0.3	0.22	0.36
α (expansion coefficient) [1/K]	$4 \cdot 10^{-5}$	$2.6 \cdot 10^{-6}$	$17.6 \cdot 10^{-6}$
ρ (density) [kg/m^3]	2500	2530	8930
C$_p$ (specific heat) [J/(kg K)]	1000	703	386
k, (thermal conductivity) [W/(m K)]	5	149	385

Table 1: Material constants used in calculations [3]

The circuit is mounted on a PCB-board which makes it reasonable to suggest that the bottom boundaries of the copper frame are fixed in the x- and y- direction. Furthermore, plane strain conditions are assumed to hold and the thickness of the circuit is assumed 10 mm.[3]

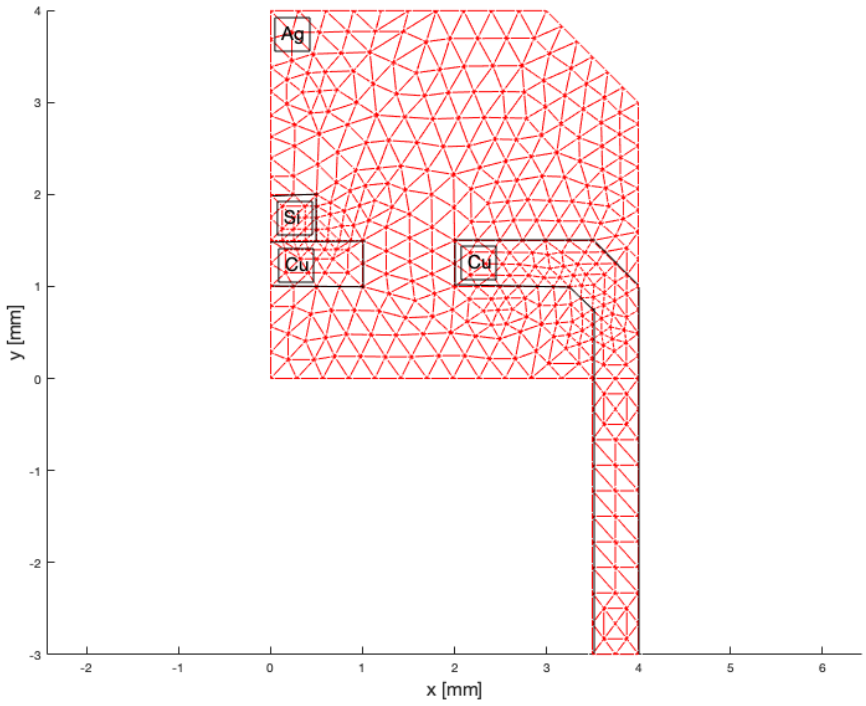


Figure 1: Model of half the circuit roughly divided in finite elements. Symmetry in the x=0 axes will give other half of the circuit. The silver filled epoxy is represented in the figure by domain Ag, the copper region is represented by the subdomain Cu and the silicon region is represented by the subdomain Si

The boundaries are thermally insulated, except the top boundary which could be assumed to obey Newton convection, stated in equation 1, whereas $\alpha_c = 40$ W/(m^{2}K).[3]

$$q_n = \alpha_c(T - T_\infty) \tag{1}$$

The environment surrounding the circuit could be assumed $T_\infty = 18°$ and the initial temperature in the circuit $T_0 = 30°$.

2 Procedure

The circuit was drawn in the Matlab-program PDEtool and a grid of triangular elements was created. From this program three different matrices were exported into Matlabs workspace, named p, t and e. The matrix p included the coordinates for all the nodes. The t matrix included all the elements associated with the node numbers, and which subdomain the element belonged to. The e matrix contained all the node pairs of the boundary elements, and which boundary segment it belonged to. The program CALFEM was used for most calculations, and the Matlab code is included in Appendix. All material data used in this assignment are presented in Table 1.

2.1 Stationary Temperature Distribution

To solve the stationary heat distribution in the circuit the balance equation for a two-dimensional body was used [1]:

$$div(t\mathbf{q}) = tQ \tag{2}$$

where $\mathbf{q}$ is the heat flow, t is the thickness and Q the heat supplied. Multiplying with a weight function v and integrating over the area and boundaries gives the general weak formulation of the two-dimensional heat flow:

$$\int_A (\nabla v)^T t\mathbf{D}\nabla T dA = -\int_{\mathcal{L}_h} vhtd\mathcal{L} - \int_{\mathcal{L}_g} vq_n td\mathcal{L} + \int_A vQtdA \tag{3}$$

where T=g on $\mathcal{L}_g$, T is the temperature, D is the constitutive matrix and h is known heat flow. Insertion of the approximation T=$\mathbf{Na}$ and Galerkin weight function v=$\mathbf{Nc}$ leads to the FE-formulation

$$\mathbf{Ka} = \mathbf{f}_b + \mathbf{f}_l = \mathbf{f} \tag{4}$$

where

$$\begin{aligned} \mathbf{K} &= \int_A \mathbf{B}^T\mathbf{D}\mathbf{B}tdA \\ \mathbf{f}_b &= -\int_{\mathcal{L}_h} \mathbf{N}^T htd\mathcal{L} - \int_{\mathcal{L}_g} \mathbf{N}^T q_n td\mathcal{L} \\ \mathbf{f}_l &= \int_A \mathbf{N}^T QtdA \end{aligned} \tag{5}$$

The stiffnes matrix ($\mathbf{K}$) and load vector ($\mathbf{f}_l$) was solved by first creating an `edof-` matrix, connecting the element number with its nodes. The stiffnes matrix and load vector for each element was then calculated using the CALFEM function `flw2e`. The global stiffness matrix and the load vector were then assembled using the `indx`-notation shown in Appendix.

To calculate the boundary vector ($\mathbf{f}_b$) the e-vector was used to find the nodes on the boundary which had convection. This was done by first reducing the e-vector to only containing the first, second and fifth rows. These contain the node numbers that are related to the elements on the boundary and the boundary segment they are related to. The nodes on the boundaries with convection were then found using `ismember`. The coordinates of these were then found using the t-vector, and the length of each element was calculated. The length is necessary for the calculation of the following equation:

$$\mathbf{f}_b = -\int_{\mathcal{L}_g} \mathbf{N}^T \alpha (T - T_\infty) t d\mathcal{L} = \int_{\mathcal{L}_g} \mathbf{N}^T \alpha (T_\infty) t d\mathcal{L} - \int_{\mathcal{L}_g} \mathbf{N}^T \alpha \mathbf{N} \mathbf{a} t d\mathcal{L} \quad (6)$$

Where $\mathcal{L}_g$ is the top boundary with convection. Since the right integral of Equation 6 includes $\mathbf{a}$, it had to be included in the stiffness matrix, to satisfy the FE-formulation. This was done by writing a function that assembles the matrix for each element and this new $\mathbf{K}$-matrix was then added to the existing, creating a total $\mathbf{K}$-matrix. The left integral was also calculated and assembled into a global $\mathbf{f}_b$-matrix. The $\mathbf{f}_b$ and $\mathbf{f}_l$ matrices were then added together, creating a global $\mathbf{f}$-matrix. The FE-equation was then solved using `solveq`. The solution - the stationary temperature distribution - was plotted for full quality and for reduced quality.

2.2 Transient Temperature Distribution

The second task was to solve the transient temperature problem in the circuit for the first 20 minutes from startup. The balance equation has another term taking into consideration the accumulation of heat [2]:

$$\rho C_p \dot{T} + div(\mathbf{q}) - Q = 0 \quad (7)$$

where ρ is the material density and C_p the specific heat capacity. Multiplying the balance equation with a weight function v and integrating gives the weak formulation:

$$\int_A v \mathbf{q}^T \mathbf{n} dA - \int_V (\nabla v)^T \mathbf{q} dV - \int_V v Q dV + \int_V v \rho C_p \dot{T} dV = 0 \quad (8)$$

As a result of the time dependence, the FE-formulation also gets an extra term, called $\mathbf{C}$:

$$\mathbf{C}\dot{\mathbf{a}} + \mathbf{K}\mathbf{a} = \mathbf{f}_b + \mathbf{f}_l = \mathbf{f} \quad (9)$$

where

$$\mathbf{C} = \int_V \mathbf{N}^T \rho C \mathbf{N} dV \quad (10)$$

This means that the equation is no longer linear and an approximation is needed to calculate the solution. The approximation used in this project was a generalised trapezoidal method:

$$\dot{\mathbf{a}} = \frac{\mathbf{a}_{n+1} - \mathbf{a}_n}{\Delta t} \implies (\mathbf{C} + \Delta t \mathbf{K})\mathbf{a}_{n+1} = \mathbf{C}\mathbf{a}_n + \Delta t \mathbf{f}_{n+1} \tag{11}$$

In Matlab the **C**-matrix for each element was found using the `plantml` function, and then assembled using `indx`-notation. Then the values of the **a**-vector were calculated as shown in Equation 11, with time steps of 1 second, and the values of the first 20 minutes after startup were stored in a matrix.

2.3 Thermal Expansion

For the third task the stresses and strains in the material due to thermal expansion were investigated. For this problem, the differential equations of equilibrium are [1]:

$$\widetilde{\nabla}^T \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \tag{12}$$

where $\boldsymbol{\sigma}$ is the stress in all directions, $\mathbf{b}$ the body forces and

$$\widetilde{\nabla} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

The weak formulation is then given by:

$$\int_A \mathbf{v}^T \mathbf{t} dA - \int_V (\widetilde{\nabla}\mathbf{v})^T \boldsymbol{\sigma} dV + \int_V \mathbf{v}^T \mathbf{b} dV = 0 \tag{13}$$

where $\mathbf{b}$ is the internal body forces and $\mathbf{t}$ is the traction vector. With galerkin weight functions the FE-formulation becomes:

$$\mathbf{K}\mathbf{a} = \mathbf{f}_b + \mathbf{f}_l + \mathbf{f}_{\Delta T} = \mathbf{f} \tag{14}$$

where

$$\begin{aligned} \mathbf{K} &= \int_A \mathbf{B}^T \mathbf{D} \mathbf{B} t dA \\ \mathbf{f}_b &= \int_A \mathbf{N}^T \mathbf{t} dA \\ \mathbf{f}_l &= \int_V \mathbf{N}^T \mathbf{b} dV \\ \mathbf{f}_{\Delta T} &= \int_V \mathbf{B}^T \mathbf{D} \boldsymbol{\epsilon}^{\mathbf{\Delta T}} dV \end{aligned} \tag{15}$$

The new **K**-matrix was calculated using the function `plante`. The global

matrix was then assembled using the `indx`-notation shown in Appendix. The constitutive matrix $\mathbf{D}$ is given by:

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix} \tag{16}$$

The initial strain $\boldsymbol{\epsilon_0}$ is given by:

$$\boldsymbol{\epsilon_0} = (1+\nu)\alpha\Delta T \cdot [1 \;\; 1 \;\; 0]^T \tag{17}$$

The initial stress could then be calculated using Equation 18:

$$\boldsymbol{\sigma_0} = \mathbf{D}\boldsymbol{\epsilon_0} \tag{18}$$

The initial stress was stored in a matrix. Since there was no external load or internal forces both $\mathbf{f}_b$ and $\mathbf{f}_l$ could be set to zero. $\mathbf{f}_{\Delta T}$ was then calculated for each element using the CALFEM function `plantf` and then assembled using `insert`. The boundary conditions along the bottom boundaries of the conductive frame, $u_x = u_y = 0$, were inserted. Along the x=0 axis, the boundary conditions were fixed in the x-direction. The displacement field was acquired using `solveq`. The displacement was then plotted for full quality and 25% reduced quality.

The effective von Mises stess was then calculated throughout the body, using that the stress vector $\boldsymbol{\sigma}$ is given by [1]:

$$\boldsymbol{\sigma} = \mathbf{D}(\boldsymbol{\epsilon} - \boldsymbol{\epsilon_0}) \tag{19}$$

The stress vector was obtained from the CALFEM function `plants` by inserting the displacement field vector. The σ_{zz} component was calculated using [1]:

$$\sigma_{zz} = \nu(\sigma_{xx} + \sigma_{yy}) - \alpha E \Delta T \tag{20}$$

The effective von Mises stress was then given by [3]:

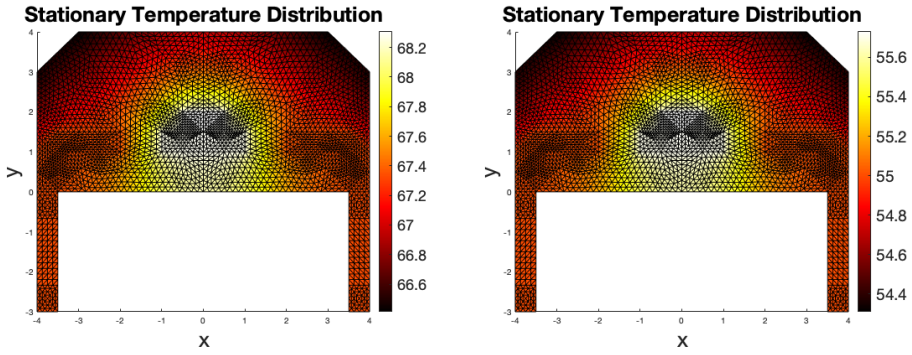
$$\sigma_{eff} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{xx}\sigma_{zz} - \sigma_{yy}\sigma_{zz} + 3\tau_{xy}^2 + 3\tau_{xz}^2 + 3\tau_{yz}^2} \tag{21}$$

The effective von Mises stress field was then plotted for full and reduced quality, and the peak stresses and the corresponding node coordinates were calculated.

3 Results

3.1 Stationary Temperature Distribution

The stationary temperature distribution in the circuit, streaming with full quality and streaming with quality reduced by 25% is presented in Figure 2.



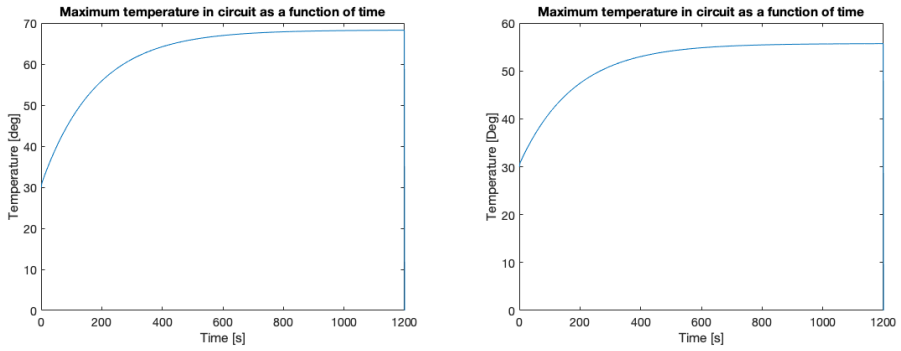
(a) Full quality **(b)** Reduced quality

Figure 2: Stationary temperature distribution in the circuit [°C]

The peak temperature streaming in full quality was calculated 68.31°C and the corresponding temperature streaming in reduced quality calculated 55.73°C.

3.2 Transient Temperature Distribution

Figure 3 shows the maximum temperature in the circuit as a function of time, the first 20 minutes after startup, for both full quality and streaming with a 25% reduction of quality.

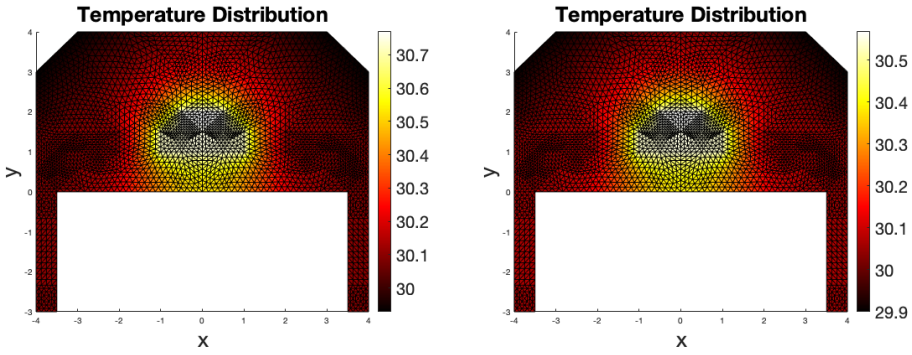


(a) Full quality **(b)** Reduced quality

Figure 3: Maximum Temperature [°C] in the circuit as a function of time [s], first 1200 seconds from startup

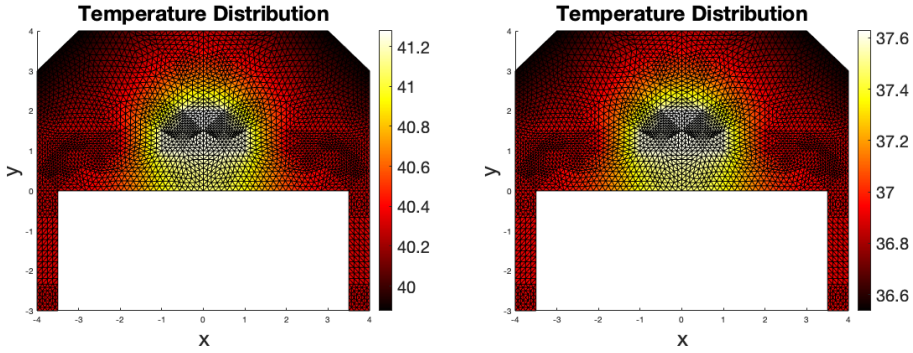
Peak temperature during the first 20 minutes streaming in full quality was calculated 68.26°C and the corresponding temperature streaming in reduced quality 55.70°C. The peak temperature in the interval $I \in [0, 1200]$ seconds is reached after 1200 seconds.

Figure 4 shows the temperature distribution 2 seconds after startup and figure 5 shows the temperature distribution 1 minute after startup when initial temperature was assumed $T_0 = 30°$.



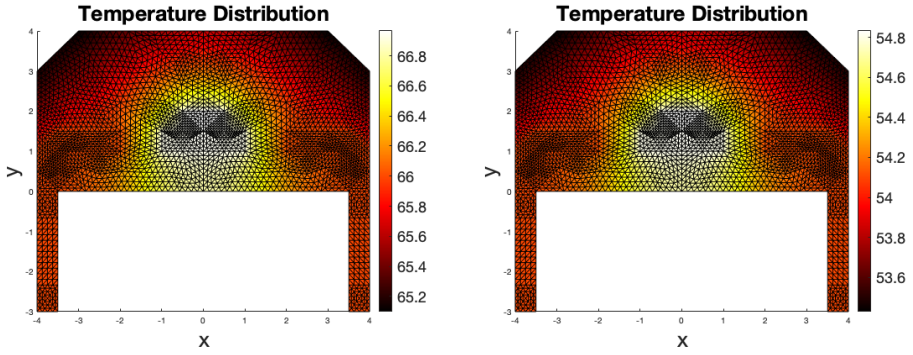
(a) Full quality **(b)** Reduced quality

Figure 4: Temperature Distribution in the circuit after 2 seconds [°C]



(a) Full quality **(b)** Reduced quality

Figure 5: Temperature Distribution in the circuit after 1 minute [°C]

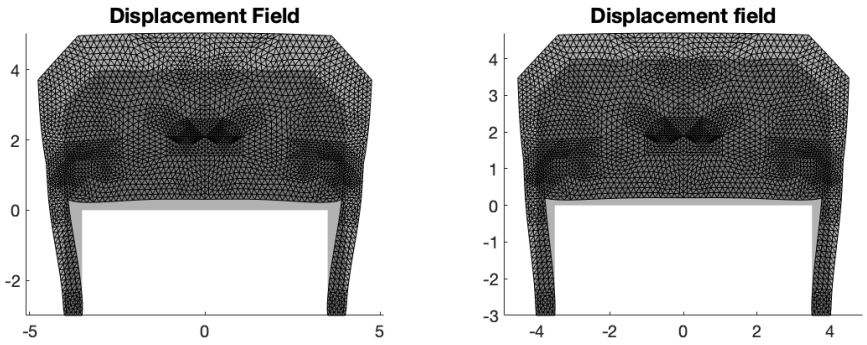


(a) Full quality **(b)** Reduced quality

Figure 6: Temperature distribution in the circuit after 10 minutes [°C]

3.3 Thermal Expansion

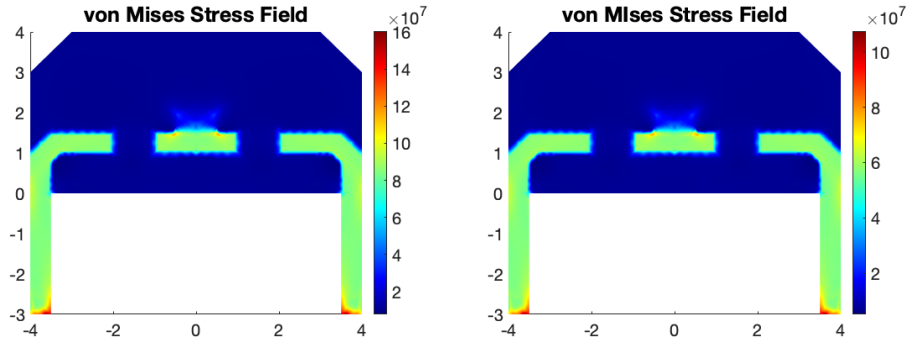
The stationary temperature distribution causes a displacement field due to thermal expansion, which is shown in Figure 7. The displacement field for both the full quality and the case when streaming quality is reduced by 25%, is magnified by a factor 100, due to the fact that thermal expansion causes a small displacement field.



(a) Full quality **(b)** Reduced quality

Figure 7: Displacement field [mm] due to thermal expansion. Magnifying factor=100

The effective von Mises stress field caused by thermal expansion is shown in Figure 6. The figure shows that the copper frame is exposed to more stress then the silver filled epoxy and the silicon. The peak von Mises stress streaming with full quality is calculated $\sigma_{eff} = 1.60 \cdot 10^8$ N/m^2 and is found in the node placed in the coordinates $(x, y) = (3.63, -3)$. Due to symmetry, this peak will also appear in the node with the coordinates $(x, y) = (-3.63, -3)$. Looking at Figure 1, this coordinate is found in the copper, at the inside corner of the bottom frame. The peak stress streaming with reduced quality is calculated $\sigma_{eff} = 1.07 \cdot 10^8$ N/m^2 and found in the same node. In addition, the figure shows that there are more stress in the copper under the silicon, than in the rest of the copper frame.



(a) Full quality **(b)** Reduced quality

Figure 8: Effective von Mises stress field [N/m^2] due to thermal expansion. Magnifying factor = 100

4 Discussion

The calculated stationary heat distribution in Figure 2 is expected, with a greater temperature close to the heat generator (silicon-die) and colder further away. It is also significantly colder at the boundary where the circuit is cooled by a fan. The copper parts have the highest thermal conductivity which is also seen in the plots since the copper part close to the heat source is much warmer than the silver epoxy surrounding it. When the streaming quality was reduced by 25 % the maximum heat was reduced by 18 %. The distribution looks the same, but with a slightly less difference between the maximum and minimum temperature. This was also expected since the convection depends on the difference between the surrounding temperature and the specific temperature, and the heat flow is greater when there is a bigger temperature difference.

When the transient temperature distribution was plotted the solution went towards the stationary solution as time progressed. The fact that the solutions agree indicates that the time dependent solution is correct in terms of the stationary result. Figure 3 shows that the most of the change in temperature is reached after 5 minutes. After 20 minutes the heat distribution has almost reached stationary conditions. The change in temperature is decreasing the closer to the stationary solution it gets and the behavior of the transient temperature distribution is again expected due to convection.

The size of time steps was chosen to 1 second. This allowed to study the temperature during the first seconds with good precision. Since the first seconds are when the temperature changes the most, shown in Figure 3, the result would not have been as precise if bigger time steps would have been used. To obtain an even more exact solution the times steps could have been changed to smaller, for example if only the first seconds were to be analysed. For this task the time frame was 20 minutes and therefore it was enough to have 1 second time steps. Smaller time steps would not have contributed much to the final result, but increased the data capacity used.

The displacement field, showed in Figure 7, caused by thermal expansion could be interpreted as credible. The figure indicates that the bottom boundaries are fixed in x- and y- direction, which is expected since the circuit is mounted on a PCB-board. The symmetry of the circuit makes it logical for the module to expand symmetrical everywhere except for in the fixed nodes. Figure 7 shows that reduced quality causes a much similar displacement field, though noticeably smaller. This is an indication that the calculations are correct, since the material is expected to expand more for higher temperatures.

Figure 8 shows that the effective von Mises stress that affects the circuit is about 33 % less when streaming with a quality reduced by 25 % compared with full quality streaming. This means that reducing the quality significantly decreases the stress in the circuit, causing the circuit to expand a lot less, as discussed above. Further on, the von Mises stress was much higher in the copper than the other materials. The expansion coefficient is higher in copper than in silicon, which is one reason to the difference in stress between those two materials. Silver however has a greater expansion coefficient, so for that reason it could have had a greater stress field. But this is not the case as seen in figure 8. This is because the stress is also dependent on Young's Modulus, which is significantly lower in silver than copper (7 GPa compared to 128 GPa). This is a greater difference than for the expansion coefficient, and it explains why the stress in copper is much higher than in the silver filled epoxy.

The peak von Mises stress is shown in figure 8 to be in the inside corner of the bottom copper frame, which is expected since the bottom is attached to a board and therefore the stress will increase when the material wants to expand but is unable to. It is also a lot of stress in the copper below the silicon, where the heat is generated. This could be explained by the fact that the silicon has a smaller expansion coefficient than the copper. At the boundary where the silicon is attached to the copper, the copper is forced to expand less than it would naturally, causing a lot of stress in those nodes.

Overall, the results obtained by the created finite element program was expected. By reducing the streaming quality by 25 %, the heat distribution in the circuit is decreased by 18 % and the effective von Mises stress by 33 %. It takes about 20 minutes to reach peak temperature when the circuit is surrounded by room temperature and has an initial temperature of 30 degrees. Due to convection the temperature increases the most the first minutes after startup. The expansion and the stress that follows could have a negative effect on the performance and durability of the circuit. The script worked well for the number of elements used in calculations. A more thorough and accurate solution would be obtained if more elements were used, but the computational cost would also be higher. For the purpose of this assignment the number of elements was sufficient to see the temperature distribution clearly, and the effects in different materials.

5 Bibliography

References

[1] Niels Ottosen and Hans Petersson, *Introduction to the finite element method*, Prentice Hall Europe, 1992

[2] *Finite element formulation of transient heat transfer*, Lund University, Faculty of Engineering, 2020

[3] *Assignment in The Finite Element Method* Division of Solid Mechanics, Lund University, Faculty of Engineering, 2020

Appendix: Computer Code

Geometry and material parameters

```
%%%%%%%%%%%%%%%%%%%%%%%%% MATERIAL PARAMETERS

%%% Heat supply Q
eq = 5*10^7*10^(-9);

%%% Temperatures and alpha
TRum = 18;
T0 = 30;
alphac = 40*10^(-6);

%%% Conductivity
kAG = 5*10^(-3);
kSI = 149*10^(-3);
kCU = 385*10^(-3);

%%% Poisson's ratio
vAG = 0.3;
vSI = 0.22;
vCU = 0.36;

%%% Young's modulus
EAG = 7*10^9;
ESI = 165*10^9;
ECU = 128*10^9;

%%% Expansion coefficients
alphaAG=4*10^(-5);
alphaSI=2.6*10^(-6);
alphaCU=17.6*10^(-6);

%%%%%%%%%%%%%%%%%%%%%%%%% GEOMETRY

thick= 10;
p;
e;
t;

%%%%%%%%% Number of elements
nelm=length(t(1,:));
```

```
%%%%%%%%% edof
edof(:,1)=1:nelm;
edof(:,2:4)=t(1:3,:)';

%%%%%%%%% coordinates nodes
coord=p';

%%%%%%%%% degrees of freedom
ndof=max(max(t(1:3,:)));

%%%%%%%%% element coordinates
[Ex,Ey]=coordxtr(edof,coord,(1:ndof)',3);

%%%%%%%%%% nodes of elements
enod=t(1:3,:)';

%%%%%%%%%% number of nodes
nnod=size(coord,1);

%%%%%%%%%% dof number is node number
dof=(1:nnod)';

%%%%%%%%%% give each dof a number
dof_S=[(1:nnod)',(nnod+1:2*nnod)'];

for ie=1:nelm
    edof_S(ie,:)=[ie dof_S(enod(ie,1),:), dof_S(enod(ie,2),:),dof_S(enod(ie,3),:)];
    edof(ie,:)=[ie,enod(ie,:)];
end


%%%%%%%%%%%%%%%%%%%%%%%%% CREATING EMPTY MATRICES AND VECTORS

Kt=zeros(ndof);
fl=zeros(ndof, 1);
Kb=zeros(ndof);
C=zeros(ndof);
timevector= zeros(1, time);
tempvector=zeros(1, time);
Ne= [2 1; 1 2];
fb=zeros(ndof, 1);
Ne3 = 1/2*[1; 1];
Kb=zeros(ndof);
```

```
K2=zeros(ndof*2);
Sigma0=zeros(3, nelm);
Sigma=zeros(4, nelm);
ebsilon=zeros(3, nelm);
fdeltaT= zeros(ndof*2, 1);
D = eye(2);
```

Stationary Temperature Distribution

```
%%%%%%%%%%%%%%%%%%%%%%%%% SOLVER FOR TEMPERATURE DISTRIBUTION
%%% Stiffness matrix  K = Kt+ Kb
%%% force vector      f = fl + fb

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Element matrices Kte and fe

for el = 1:nelm
    [Kte, fe] = flw2te(Ex(el,:), Ey(el,:), thick, D, eq);

    if t(4, el) == 3
        Kte= Kte*kSI;

    elseif t(4,el) == 2 || t(4,el) == 4
        Kte= Kte*kCU;

    else
        Kte= Kte*kAG;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Kt
    indx = edof(el,2:end);
    Kt(indx,indx) = Kt(indx,indx)+Kte;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% fl
    if t(4, el) == 3
        indx = edof(el,2:end);
        fl(indx) = fl(indx) + fe;
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Boundary force vector %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Element matrices Kbe and fbe
er = e([1 2 5],:);
conv_segments = [14 1 17];
edges_conv = [];

for i = 1:size(er,2)
    if ismember(er(3,i),conv_segments)
        edges_conv = [edges_conv er(1:2,i)];
    end
end
```

```
edges_conv2 = transpose(edges_conv);


for i = 1:length(edges_conv)
    dx = coord(edges_conv(2, i), 1) - coord(edges_conv(1, i), 1);
    dy = coord(edges_conv(2, i), 2) - coord(edges_conv(1, i), 2);
    l = sqrt(dx^2 +dy^2);
    Kbe = Ne*thick*alphac*l/6;
    fbe= Ne3*thick*alphac*l*TRum;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Kb = assem2(edges_conv2(i,:),Kb,Kbe);
    fb = assem3(edges_conv2(i,:),fb,fbe);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Stiffness matrix K %%%%%%%%%%%%%%%%%%%%
K = Kb+Kt;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Force vector f %%%%%%%%%%%%%%%%%%%%%%%%
f= fb+fl;

a = solveq(K, f);
ed = extract(edof)
```

Transient Temperature Distribution

Run with the code for the stationary temperature distribution solver.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SOLVER FOR TRANSIENT HEAT FLOW
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% C matrix

for el = 1:nelm

    if t(4, el) == 3

        Ce =thick*plantml(Ex(el,:), Ey(el,:), cpSI*rawSI);

    elseif t(4,el) == 2 || t(4,el) == 4
        Ce =thick*plantml(Ex(el,:), Ey(el,:), cpCU*rawCU);

    else
        Ce =thick*plantml(Ex(el,:), Ey(el,:), cpAG*rawAG);
    end

    indx = edof(el,2:end);
    C(indx,indx) = C(indx,indx)+Ce;
end

%%%%%%%%%%%%%%%%%%%%%%%% Trapezoidal to get time derivative %%%%%%%%%%

dt=1;
time=20*60;
a1 = T0*zeros(ndof,time);
aold = T0*ones(ndof,1);
fnew=f;
a1(:, 1) = aold;

for i = 1: dt: time-1
    anew = (C+dt*K)\(C*aold+ dt * fnew);
    a1(:, i+1) = anew;
    aold = anew;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Boundary conditions

for sec = 1:1:time-1
    amin= a1(:,sec);
```

```
    maxamin=max(amin);
    tempvector(1, sec) = maxamin;
end

figure(3)
plot(tempvector)

a1min= a1(:,1*60);
a10min= a1(:,10*60);
a20min= a1(:,20*60);
a = solveq(K, f);
ed1 = extract(edof, a1min);
ed10 = extract(edof, a10min);
ed20 = extract(edof, a20min);


ex = [-flip(Ex) ; Ex];
ey = [flip(Ey) ; Ey];
ed1 = [flip(ed1); ed1];
ed10 = [flip(ed10); ed10];
ed20 = [flip(ed20); ed20];

figure(4)
patch(ex',ey',ed1')
colormap(hot)

figure(5)
patch(ex',ey',ed10')
colormap(hot)

figure(6)
patch(ex',ey',ed20')
colormap(hot)
```

Thermal Expansion

Run with the code for the stationary temperature distribution solver.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SOLVER FOR DISPLACEMENT
%%%%%%%%%%%%%%%%%%%% Stiffness matrix K2
a0= T0*ones(length(a),1);
deltaT = (a-a0)*1/3;


for el = 1:nelm

    Telement = deltaT(t(1,el)) +deltaT(t(2,el))+deltaT(t(3,el));

    if t(4, el) == 3
        D = ESI/((1+vSI)*(1-2*vSI))*[1-vSI vSI 0; vSI 1-vSI 0; 0 0 1/2*(1-2*vSI)];
        K2e = plante(Ex(el,:), Ey(el,:), [2 thick], D);
        Debs0 = (alphaSI*ESI*Telement)/(1-2*vSI)*[1; 1; 0];

    elseif t(4,el) == 2 || t(4,el) == 4
        D = ECU/((1+vCU)*(1-2*vCU))*[1-vCU vCU 0; vCU 1-vCU 0; 0 0 1/2*(1-2*vCU)];
        K2e = plante(Ex(el,:), Ey(el,:), [2 thick], D);
        Debs0 = (alphaCU*ECU*Telement)/(1-2*vCU)*[1; 1; 0];
    else
        D=EAG/((1+vAG)*(1-2*vAG))*[1-vAG vAG 0; vAG 1-vAG 0; 0 0 1/2*(1-2*vAG)];
        K2e = plante(Ex(el,:), Ey(el,:), [2 thick], D);
        Debs0= (alphaAG*EAG*Telement)/(1-2*vAG)*[1; 1; 0];
    end

    indx = edof_S(el,2:end);
    K2(indx,indx) = K2(indx,indx)+K2e;

    Sigma0(:,el) = Debs0;
end

Sigma0= transpose(Sigma0);

%%%%%%%% Internal element force vector fdeltaT in a triangular element
%%%%%%%% in plain strain


for el = 1:nelm
    fe2= plantf(Ex(el,:), Ey(el,:), [2 thick], Sigma0(el,:));
    fdeltaT = insert(edof_S(el,:), fdeltaT ,fe2);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Boundary along x = 0

conv3_segments = [18 19 20 21];
edges_conv3 = [];

for i = 1:size(er,2)
    if ismember(er(3,i),conv3_segments)
        edges_conv3 = [edges_conv3 er(1:2,i)];
    end
end

uniquefree1 = unique(edges_conv3);

for nodnr = 1 : length(uniquefree1)
    uniknod = uniquefree1(nodnr);
    for i = 1 : nelm
        for j = 2 : 7
            x = edof_S(i, j);
            if x == uniknod
                if mod(j, 2) == 0
                else
                    xnod = edof_S(i, j-1);
                    uniquefree1(nodnrs)=xnod;
                end
            end
        end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Boundary bottom boundaries

conv4_segments = 6; % Choosen boundary segments
edges_conv4 = [];

for i = 1:size(er,2)
    if ismember(er(3,i),conv4_segments)
        edges_conv4 = [edges_conv4 er(1:2,i)];
    end
end

uniquefree2 = unique(edges_conv4);
```

```
for nodnr = 1 : length(uniquefree2)
    uniknod = uniquefree2(nodnr);

    for i = 1 : nelm
        for j = 2 : 7
            x = edof_S(i, j);
            if x == uniknod
                if mod(j, 2) == 0
                    findnod = edof_S(i, j+1);
                    uniquefree2 = [uniquefree2; findnod];
                else
                    findnod = edof_S(i, j-1);
                    uniquefree2 = [uniquefree2; findnod];
                end
            end
        end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Boundary conditions %%%%%%%%%%%%%%%%%%

uniquefree = [uniquefree1; uniquefree2];
uniquefree = unique(uniquefree);

bc = zeros(length(uniquefree), 1);
bc = [uniquefree, bc];

a = solveq(K2, fdeltaT, bc);
ed = extract(edof_S, a);

mag = 100; % Magnification (due to small deformations)
exd = Ex + mag*ed(:,1:2:end);
eyd = Ey + mag*ed(:,2:2:end);


ex = [-flip(Ex) ; Ex];
ey = [flip(Ey) ; Ey];
exd = [-flip(exd) ; exd];
eyd = [flip(eyd) ; eyd];


figure(7)
```

```
patch(ex',ey',[0 0 0],'EdgeColor','none','FaceAlpha',0.3)
hold on
patch(exd',eyd',[0 0 0],'FaceAlpha',0.3);
axis equal
title('Displacement field [Magnitude enhancement 100]')



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VON MISES %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% Stress vector holds from plants

for el = 1:nelm

   Telement = deltaT(t(1,el)) +deltaT(t(2,el))+deltaT(t(3,el));

    if t(4, el) == 3
        D = ESI/((1+vSI)*(1-2*vSI))*[1-vSI vSI 0; vSI 1-vSI 0; 0 0 1/2*(1-2*vSI)];
        [es,et]=plants(Ex(el,:), Ey(el,:), [2 thick], D, ed(el,:));
        es = es - (alphaSI*ESI*Telement)/(1-2*vSI)*[1 1 0];
        eszz = vSI*(es(1,1)+ es(1,2)) - (alphaSI*ESI*Telement);

    elseif t(4,el) == 2 || t(4,el) == 4
        D = ECU/((1+vCU)*(1-2*vCU))*[1-vCU vCU 0; vCU 1-vCU 0; 0 0 1/2*(1-2*vCU)];
        [es,et]=plants(Ex(el,:), Ey(el,:), [2 thick], D, ed(el,:));
        es = es - (alphaCU*ECU*Telement)/(1-2*vCU)*[1 1 0];
        eszz = vCU*(es(1,1)+ es(1,2)) - (alphaCU*ECU*Telement);

    else
        D=EAG/((1+vAG)*(1-2*vAG))*[1-vAG vAG 0; vAG 1-vAG 0; 0 0 1/2*(1-2*vAG)];
        [es,et]=plants(Ex(el,:), Ey(el,:), [2 thick], D, ed(el,:));
        es = es - (alphaAG*EAG*Telement)/(1-2*vAG)*[1 1 0];
        eszz = vAG*(es(1,1)+ es(1,2)) - (alphaAG*EAG*Telement);

    end

    Sigma(1:3,el) = es;
    Sigma(4,el) =eszz;
end

Sigma = transpose(Sigma);

for i= 1:nelm
    vonMise = sqrt(Sigma(i,1)^2 + Sigma(i,2)^2 + Sigma(i,4)^2 - Sigma(i,1)*Sigma(i,2
    Seff_el(i,1)= vonMise;
```

```
end

for i=1:size(coord,1)
    [c0,c1]=find(edof_S(:,2:4)==i);
    Seff_nod(i,1)=sum(Seff_el(c0))/size(c0,1);
end

[maxStresss, maxnod]= max(Seff_nod);

Ed2 = extract(edof, Seff_nod');
Ex1 = [-flip(Ex) ; Ex];
Ey1 = [flip(Ey) ; Ey];
Ed21 = [flip(Ed2) ; Ed2];

figure (7)
patch(Ex1',Ey1',Ed21', 'EdgeColor','none')
colormap(jet(5000))
```