December 15, 2023 **Rapport de projet C++**

Université de Strasbourg

Master 1 CSMI

Simulation numérique d'un dispositif de refroidissement

Auteur :
Adama Dieng

Contents

Chapter 1

Contexte du projet

1.1 Présentation

Ce projet a pour but de réaliser un programme C++ permettant d'étudier le comportement thermique d'un dispositif de refroidissement d'un micro-processeur. Un des moyens généralement utilisé pour réguler la température d'un processeur est l'utilisation d'un ventilateur. Le soufflage d'air va permettre d'évacuer la chaleur par convection sur la surface du processeur. Pour améliorer ce processus, le processeur est généralement attaché à un dissipateur (composé de plusieurs ailettes) qui est un élement trés conducteur de chaleur.

Pour ce projet, nous allons nous intéresser uniquememt à la simulation thermique d'une seule ailette du dissipateur. Les données du problème sont décrite par les longueurs L_x, L_y et L_z, le flux de chaleur Φ_p généré par le processeur et la température ambiante Te. De plus, nous supposons que l'ailette est suffisamment mince pour considérer le problème unidimensionnel. La température T en un point donné dépendra uniquement de sa position selon x et de l'instant t.

En prenant en compte les pertes latérales par convection avec l'air et en supposant connaitre le flux de chaleur Φ_p dégagé par le processeur en $x = 0$ et l'hypothèse quee le transfert de chaleur en $x = L_x$ est négligeable par rapport au flux apporté sur la section longitudinale l'equation de la chaleur dans une ailette definie sur le domaine $x = [0, L_x]$ est donnée par :

$$\rho C_p \frac{\partial T}{\partial t} - K \frac{\partial^2 T}{\partial x^2} + \frac{h_c p}{S}(T - T_e) = 0 \qquad \forall x \in]0, L_x[\tag{1.1}$$

$$-K \frac{\partial T}{\partial x} = \Phi_p \qquad \forall x = 0 \tag{1.2}$$

$$-K \frac{\partial T}{\partial x} = 0 \qquad \forall x = L_x \tag{1.3}$$

1.2 Paramètres Physiques et géométriques

Les valeurs des paramètres utilisés pour les simulations sont données dans la table 1. On a choisi les propriétés physiques en considérant que l'ailette est constituée d'un alliage d'aluminium.

Ces paramètres Pourront être modifiés dans diefferents.

Paramètres	Valeurs	Unités
L_x	0.04	m
L_y	0.004	m
L_z	0.05	m
h_c	200	W/m2/K
p	2*(L_y + L_z)	m
S	L_y * L_z	m2
ρ	2700	kg/m3
C_p	940	J/kg/K
T_e	20	°C
Φ_p	1.25e5	W/m2
K	164	W/m/K

Table 1.1: Paramètres physiques et géométriques

1.3 Modèle Stationnaire

Nous commençons par traiter le cas stationnaire, c'est-à-dire par calculer la distribution de température lorsque le temps t tend vers l'infini. On enlevera alors la dérivée en temps présente dans les ´equations (1.1) et (1.2)

En utilisant la méthode des différences finies, on resoudre l'équation. Pour cela, nous décomposons le segment $[0, L_x]$ en M intervalles de longueur $h = \frac{L_x}{M}$. Nous obtenons un maillage de $M + 1$ points $x_i = ih$ pour $i = 0, ..., M$. La solution numérique sera alors decrite par les valeurs $M + 1$ points et on notera T_i la valeur de la température en x_i.

Avec un developpement de Taylor, on obtient l'approximation suivante :

$$\frac{\partial^2 T}{\partial x^2}(x_i) = \frac{T_{i-1} - 2T_i + T_{i+1}}{h^2} \tag{1.4}$$

Nous obtenons ainsi les conditions discrètes du problemesur tout les noeuds internes:

$$-K\frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} + \frac{h_c p}{S}(T_i - T_e) = 0 \quad \forall i \in [1, M-1] \tag{1.5}$$

$$-K\frac{\partial T}{\partial x}|_{x=0} = -K\frac{T_1 - T_0}{h} = \Phi_p \tag{1.6}$$

$$-K\frac{\partial T}{\partial x}|_{x=L_x} = -K\frac{T_{M-1} - T_M}{h} = 0 \tag{1.7}$$

On obtient ainsi un système linéaire. On peut l'écrire sous la forme matricielle $AX = F$ avec A une matrice tridiagonale de taille $M \times M$, F un vecteur de taille M et X le vecteur solution de taille M(la température en tout les points x_i pour $i = 0, ..., M$).

On a donc :

$$\begin{bmatrix} b_0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_2 & b_2 & c_2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{M-1} & b_{M-1} & c_{M-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_M & b_M \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ T_{M-1} \\ T_M \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ F_{M-1} \\ F_M \end{bmatrix}$$

Avec :

$$a = (a_1, \dots, a_{M-1}, a_M) = (-\frac{K}{h^2}, \dots, -\frac{K}{h^2}, -\frac{K}{h})$$

$$b = (b_0, b_1, \dots, b_{M-1}, b_M) = (\frac{K}{h}, \frac{h_c p}{S} + \frac{2K}{h^2}, \dots, \frac{h_c p}{S} + \frac{2K}{h^2}, \frac{K}{h})$$

$$c = (c_0, c_1, \dots, c_{M-1}, c_{M-1}) = (\frac{-K}{h}, \frac{-K}{h^2}, \dots, \frac{-K}{h^2})$$

$$F = (F_0, F_1, \dots, F_{M-1}, F_M) = (\Phi_p, \frac{h_c p}{S} T_e, \dots, \frac{h_c p}{S} T_e, 0)$$

Pour Resoudre ce systeme, nous utilisons la méthode de la décomposition LU, ensuite nous calculons la solution en utilisant la méthode de remontée comme decrit dans le projet.

La matrice A est une matrice tridiagonale, on peut donc ecrire $A = LU$ où:

$$L = \begin{bmatrix} b_0^* & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_1 & b_1^* & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_2 & b_2^* & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{M-1} & b_{M-1}^* & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & a_M & b_M^* \end{bmatrix} \text{ et } U = \begin{bmatrix} 1 & c_0^* & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & c_1^* & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & c_2^* & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & c_{M-1}^* \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}$$

Avec :

$$b_0^* = b_0$$

$$c_0^* = \frac{c_0}{b_0^*}$$

Pour $i = 1, \dots, M-1$:

$$b_i^* = b_i - a_i . c_{i-1}^*$$

$$c_i^* = \frac{c_i}{b_i^*}$$

et

$$b_M^* = b_M - a_M . c_{M-1}^*$$

Une fois la décomposition LU effectuée, on peut résoudre le système $AX = F$ en deux étapes :

- Résoudre le système $LY = F$ en utilisant la méthode de descente.

$$Y_0 = F_0 / b_0^*$$

Pour $i = 1, \dots, M$:

$$Y_i = \frac{F_i - a_i . Y_{i-1}}{b_i^*}$$

- Résoudre le système $UX = Y$ en utilisant la méthode de remontée.

$$X_M = Y_M$$

Pour $i = M-1, \dots, 0$:

$$X_i = Y_i - c_i^* . X_{i+1}$$

Des fonctions ainsi que des classes ont été implémentées pour résoudre ce problème en utilisant les etapes decrites ci-dessus. **Voir section2**

1.4 Modèle instationnaire

Nous souhaitons maintenant suivre l'évolution en temps de la distribution de température dans l'ailette pendant une durée notée $tfinal$. Pour cela, nous devons prendre en compte la dérivée en temps dans l'équation aux dérivées partielles.

Nous commençons par échantillonnner l'intervalle de temps $[0, tfinal]$ avec une discrétisation régulière composée de N+1 temps discret. Le pas de temps (i.e. la durée entre 2 temps discrets), sera donc constant et noté Δt. Ainsi, l'échantillonnage en temps est décrit par l'ensemble $tn = n\Delta t, n = 0, .., N$. On utilisera pour cette étude $tfinal = 300s$ et $N = 600$. On note T_i^n la température discrète calculée au points x_i et à l'instant t_n.

La dérivé en temps est discrétisée à l'aide d'un schéma d'Euler:

$$\frac{\partial T}{\partial t}(x_i, t_n) \approx \frac{T_i^{n+1} - T_i^n}{\Delta t}$$

Nous obtenons ainsi les équations discrètes du problème instationnaire définies sur tous les noeuds internes à chaque temps t_n, $n = 1, .., N$:

$$\rho C_p \frac{T_i^{n+1} - T_i^n}{\Delta t} - K \frac{T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}}{h^2} + \frac{h_c p}{S}(T_i^{n+1} - T_e) = 0 \qquad \forall i \in [1, M-1]$$

et les conditions aux limites :

$$-K \frac{\partial T}{\partial x}|_{x=0} \approx -K \frac{T_1^{n+1} - T_0^{n+1}}{h} = \Phi_p$$

$$-K \frac{\partial T}{\partial x}|_{x=L_x} \approx -K \frac{T_{M-1}^{n+1} - T_M^{n+1}}{h} = 0$$

Nous choisissons que la température initiale est uniforme et égale à la température ambiante $T_i^0 = T_e$ pour tout $i = 0, .., M$.

Nous obtenons ainsi un système linéaire. On peut l'écrire, comme pour le cas stationnaire, sous la forme matricielle $AX^{n+1} = mX^n + F$ avec A une matrice tridiagonale de taille $M \times M$, F un vecteur de taille M, X^{n+1} le vecteur solution de taille M(la température en tout les points x_i pour $i = 0, ..., M$) à l'instant t_{n+1} et x^n celui à l'instant t_n.

On a donc :

$$\begin{bmatrix} b_0 & c_0 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_2 & b_2 & c_2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{M-1} & b_{M-1} & c_{M-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_M & b_M \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ \vdots \\ T_{M-1}^{n+1} \\ T_M^{n+1} \end{bmatrix} = \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_{M-1} \\ m_M \end{bmatrix}^T \begin{bmatrix} T_0^n \\ T_1^n \\ T_2^n \\ \vdots \\ T_{M-1}^n \\ T_M^n \end{bmatrix} + \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ F_{M-1} \\ F_M \end{bmatrix}$$

Avec :

$$a = (a_1, \dots, a_{M-1}, a_M) = (-\frac{K}{h^2}, \dots, -\frac{K}{h^2}, -\frac{K}{h})$$

$$b = (b_0, b_1, \dots, b_{M-1}, b_M) = (\frac{K}{h}, \frac{h_c p}{S} + \frac{2K}{h^2} + \frac{\rho C_p}{\Delta t}, \dots, \frac{h_c p}{S} + \frac{2K}{h^2} + \frac{\rho C_p}{\Delta t}, \frac{K}{h})$$

$$c = (c_0, c_1, \ldots, c_{M-1}) = (\frac{-K}{h}, \frac{-K}{h^2}, \ldots, \frac{-K}{h^2})$$

$$m = (m_0, m_1, \ldots, m_{M-1}, m_M) = (0, \frac{\rho C_p}{\Delta t}, \ldots, \frac{\rho C_p}{\Delta t}, 0)$$

et

$$F = (F_0, F_1, \ldots, F_{M-1}, F_M) = (\Phi_p, \frac{h_c p}{S} T_e, \ldots, \frac{h_c p}{S} T_e, 0)$$

Pour Resoudre ce systeme, nous utilisons la méthode de la décomposition LU, ensuite nous calculons la solution en utilisant la méthode de remontée comme decrit dans le projet.

Attetion: Pour chaque instant t_n, nous devons calculer le second membre $mX^n + F$. Il faut donc calculer mX^n à chaque itération. Nous avons fait des fonctions pour bien gerer ce cas. **Voir section 2**

1.5 Visualisation 3d de la solution

Nous souhaitons maintenant visualiser la solution sur la geometrie 3D de l'ailette qui est un pavé droit paramétré par les longueurs L_x, L_y et L_z. Nous commencons par dicretiser les intervalles $[0, L_x]$, $[0, L_y]$ et $[0, L_z]$ en $M_x + 1$, $M_y + 1$ et $M_z + 1$ points respectivement.

Dans la suite de ce projet, On prends $M_x = 50$, $M_y = 10$ et $M_z = 30$. On obtiendra alors un maillage de $(M_x+1).(M_y+1).(M_z+1)$ points. On note $T^n_{i,j,k}$ la température discrète calculée au point $P_{i,j,k} = (x_i, y_j, z_k)$ de la discretisation volumique où i, j et k sont les indices de la discretisation de $[0, L_x]$, $[0, L_y]$ et $[0, L_z]$

Il nous faudra donc interpoler la solution calculée sur le maillage 1D sur le maillage 3D. Pour cela, nous utilisons la méthode de l'interpolation linéaire. On a alors :

$$T^n_{i,j,k} = \hat{T}_i \forall i = 0, .., M_x \text{ et } \forall j = 0, .., M_y \text{ et } \forall k = 0, .., M_z$$

avec $\hat{T}_i$ un interpolant linéaire de la solution 1D au point x_i. Il depend uniquement de la position x car $x_i00 = x_{ijk} \forall j = 0, .., M_y$ et $\forall k = 0, .., M_z$.

La construction de l'interpolant linéaire $\hat{T}_i$ est donnée par l'algotithme suivant :

Algotithme de calcul de $\hat{T}_i$

---

for i = 0,.., Mx **do**
-Localiser x_{i00} dansle maillage 1D(i.e. trouver k tel que $x_{i00} \in [x_k, x_{k+1}]$)
-calculer les coefficients de la droite lineaire $y = ax + b$ passant par les points (x_k, T_k) et (x_{k+1}, T_{k+1})
-évaluer $\hat{T}_i = ax_{i00} + b$
end for

Les resultats correspondant aux simulations stationnaire et instationnaire sont donnés dans la **section 3**.

Chapter 2

Description du programme et des dossiers associés

2.1 Compilation et Execution

La compilation du programme se fait en se plaçant dans le Repertoire **CPP**. Esuite il faudra faire la commande **g++ *.cpp -o MonExecutable** pour compiler le programme. Cette commande permettra entre autre de compiler tous les fichiers **.cpp** du repertoire et de creer un executable nommé **MonExecutable** que nous pourrons executer en faisant la commande **./MonExecutable**.

Attention: Il faudra pas oublier de faire la commande ***.cpp** sinon le programme ne pourra pas compiler et aura probablement des erreurs du genre **reference indefinie vers ...**.

2.2 Structure du dossier

La structure de ce projet, nommé **Rendu projet**, est organisée de manière claire et modulaire. Voici une description plus détaillée de chaque élément du dossier du projet:

- **HPP** : Ce répertoire est dédié aux fichiers d'en-tête **(.hpp)** conçus pour le projet. Les fichiers d'en-tête contiennent les déclarations des classes, fonctions et variables utilisées dans le projet. Ils sont inclus dans les fichiers source pour permettre la compilation.
- **CPP** :Ce répertoire contient les fichiers sources **(.cpp)** spécifiquement conçus pour le projet. Les fichiers source contiennent les implémentations réelles des classes et fonctions déclarées dans les fichiers d'en-tête.
- **CFG** :Répertoire contenant le fichier de configuration **simu.cfg** qui contient les paramètres physiques et géométriques du problème.
- **CSV** :Répertoire contenant les fichiers **.csv** générés par le programme lors de l'exécution.

- **VTK** :Comme pour les fichiers csv, ce répertoire contient les fichiers **.vtk** générés.

- **TesterLesFonctions** :Enfin ce répertoire contient presque tous les fichiers **.cpp** utilisés pour tester les fonctions et classes implémentées au fur et à mesure de notre travail.

2.3 Contenu des répertoires et description des fichiers

Dans la plupart des fichiers **.cpp** et **.hpp** utilisés, notament les fonctions et classes, nous avons mis des commentaires pour expliquer leur fonctionnement. Cependant nous allons décrire brièvement le contenu de chaque fichier.

2.3.1 Repertoire HPP

- **parametre.hpp** : Contient une classe **Parametres**qui va nous permettre de lire un fichier de configuration et d'initialiser les paramètres du problème par les valeurs lues. Il contient notament un contructeur qui prend en paramètre le nom du f ichier de configuration.

- **Matrice.hpp** : Nous avons pensé à creer ici une classe **MatriceTridiagonale** pour initialiser et stocker des matrices de type tridiagonale dans les differents cas du probleme(stationnaire et instationnaire).

- **Vecteur.hpp** : Comme pour la classe Matrice, nous avons créé une classe **Vecteur** pour initialiser et stocker des vecteurs. Il contient aussi deux constructeurs, notement un qui prend en paramètre en entier(la taille du vecteur) etc un vecteur de type double. Des operateurs tels que **[]**, +, * sont aussi surchargés pour faciliter les calculs.

- **resolution.hpp** : Ce fichier contient des fonctions générales et communes aux deux modèles(stationnaire et instationnaire). Il contient notament:

  - Une fonction **MatriceTridiagonale** DecompositionLU(const MatriceTridiagonale A)**: cette fonction permettra d'effectuer la decomposition d'une matrice tridiaognale en un produit de LU par la mehtode de Gauss.

  - Une fonction **void ResolutionLYF(Vecteur& Y, const MatriceTridiagonale& L, const Vecteur& F);**: pour la resolution de LY = F (c'est la descente).

  - Une fonction **void ResolutionUXY(Vecteur& X, const MatriceTridiagonale& U, const Vecteur& Y);**: pour la resolution de UX = Y (c'est la remontée).

- **stationnaire.hpp** : Ce fichier contient une grande classe **ModeleStationnaire**. Cette classe contient toutes les fonctions et variables necessaires pour resoudre le probleme stationnaire.

- Methode **void A_stationnaire**: pour definir la matrice A pour le cas stationnaire.

- Methode **void StationnaireF()**: pour definir le vecteur F pour le cas stationnaire

- Methode **void ResolutionNumerique()**: pour la resolution numerique du modele stationnaire.

- Methode **Vecteur* Solution_T_exact()**: pour la solution exacte du modele stationnaire Texact

- Methode **void EcrireCSV();** : pour ecrire dans un fichier CSV la solution numérique 1D et la solution exacte afin de les visualiser.

- methode **void EcrireVTK();** : Pareil que la methode EcrireCSV mais pour les fichiers VTK afin de visualiser la solution en 3D avec Paraview.

- **instationnaire.hpp**: Comme dans le cas stationnaire, ce fichier contient une grande classe **ModeleInstationnaire** qui gere le cas instationnaire. Il contient notament:

  - Methode **void A_instationnaire()**: definir la matrice A pour le cas instationnaire

  - Methode **void instationnaire_F()**pour definir le vecteur F pour le cas instationnaire

  - Methode **void Resolution_Instationnaire()**: pour la resolution numerique du modele instationnaire

  - Methode **void EcrireCSV_Instationnaire();** : pour ecrire dans un fichier CSV la solution numérique 1D du modèle instationnaire au temps $t = 15$, $t = 30$, $t = 60$, $t = 90$, $t = 150$ et $t = 210$ afin de les visualiser.

  - Methode **void EcrireVTK_Instationnaire();** : Pour ecrire la solution 3D du modèle instationnaire aux memes temps t ci dessus.

2.3.2 Repertoire CPP

Ce répertoire contient les fichiers **.cpp** respectivement associés aux fichiers **.hpp** du répertoire **HPP**. Il contient notament les implémentations des fonctions et classes déclarées dans les fichiers **.hpp**.

Chapter 3

Analyse des résultats

3.1 Modèle Stationnaire

3.1.1 Solution en dimension 1D

Nous avons obtenus deux solutions avec le modèle stationnaire:

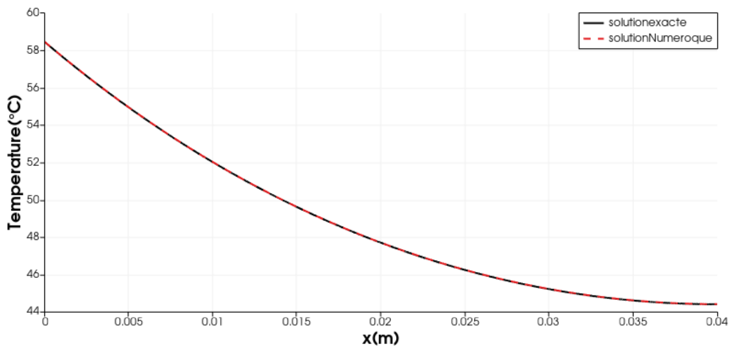

Figure 3.1: Solution 1D avec $L_x = 40mm$

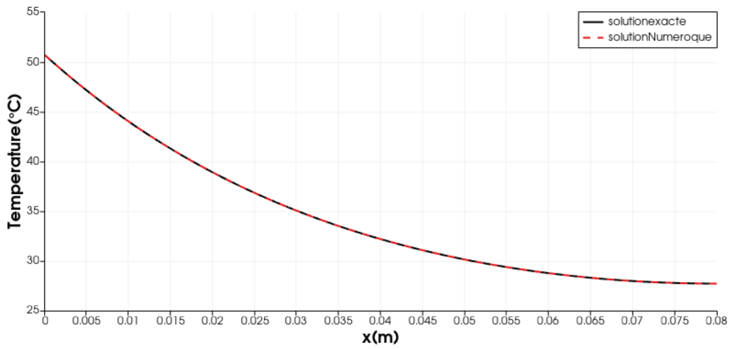

Figure 3.2: Solution 1D avec $L_x = 80mm$

Nous avons deux courbes dans chacune des figures ci-dessus. Les courbes en bleu représentent les solutions numériques et les courbes en rouge représentent les solutions exactes calculées à l'aide de la fonction **Vecteur* SolutionTexact()** de la classe **ModeleStationnaire**. Analytiquement, la solution exacte est donnée par :

$$T(x) = T_e + \frac{\Phi_p}{K} \frac{\cosh(\sqrt{a})L_x}{\sqrt{a}\sinh(\sqrt{a})L_x} \frac{\cosh(\sqrt{a}(L_x - x))}{\cosh(\sqrt{a}L_x)} \qquad \text{avec } a = \frac{h_c p}{SK}$$

Interpretation:

Dans un premier regard, on observe que les solutions calculées semblent correspondre visuellement aux solutions exactes. En poursuivant l'analyse, il semble tout à fait cohérent que la température à gauche de l'ailette, près du processeur, soit plus élevée, diminuant à mesure que l'on s'éloigne. Ceci indique clairement un processus de dissipation de chaleur. De plus, il est à noter que plus la longueur de l'ailette est importante, plus les températures semblent diminuer. Par exemple, pour une longueur Lx égale à $40mm$, les températures se situent approximativement entre 44 et 59 °C, tandis que pour une longueur Lx égale à $0.08m$, elles varient entre 25 et 51 °C.

3.1.2 Solution en dimension 3D

Pour ce cas, on commence d'abord par definir le maillage qui comporte $M_x.M_y.M_z$ points avec $M_x = 50$, $M_y = 10$ et $M_z = 30$. On a obtenu:

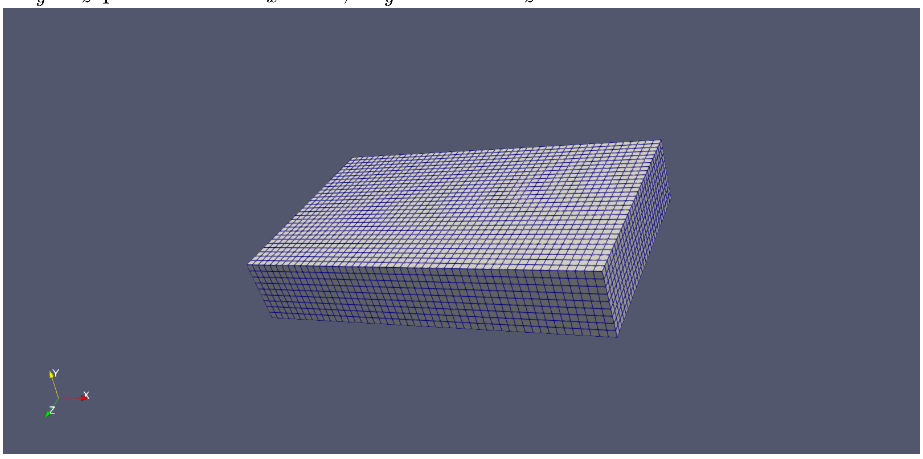

Figure 3.3: Maillage 3D

Et ensuite on a obtenu la solution suivante:

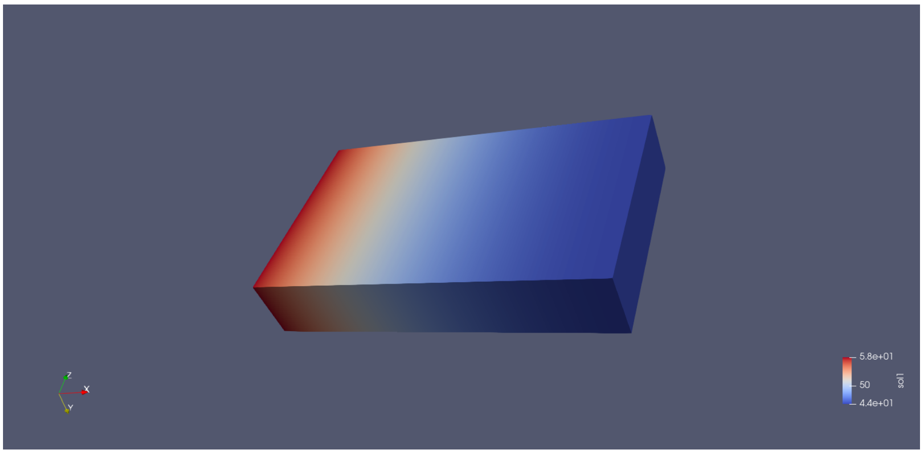

Figure 3.4: Solution 3D ModeleStationnaire

Interpretation: On observe que plus on s'eloigne du bord gauche de l'ailette, plus la temperature diminue. Ceci est tout à fait cohérent avec ce que nous avons observé dans le cas 1D.

3.2 Modèle Instationnaire

Pour ce modele, normalement nous avons deux cas à traiter; le cas où le flux est constant et le cas où il est activé et desactivé. **Nous avons traité le cas où le flux est constant**. Nous avons donc pris $\Phi_p = 1.25e5$ et nous avons les resultats suivants.

3.2.1 Solution en dimension 1D

Nous avons tracé la solution numérique 1D du modèle instationnaire aux temps $t = 15$, $t = 30$, $t = 60$, $t = 90$, $t = 150$ et $t = 210$.

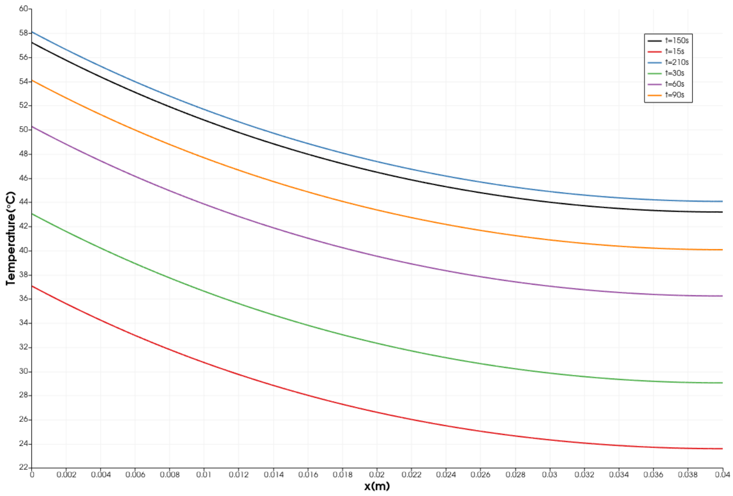

Figure 3.5: Solution 1D Modele instationnaire

Interpretation: On observe initialement que, similairement au modèle stationnaire, la température décroît à mesure que l'on s'éloigne du processeur (c'est-à-dire lorsque x augmente). En outre, l'augmentation du temps entraîne une élévation de la température de l'ailette, ce qui est cohérent étant donné que le flux de chaleur est maintenu constant. Bien que la solution semble atteindre un plateau après un certain temps, il est important de noter que l'ailette ne se refroidira pas.

3.2.2 Solution en dimension 3D

Toujours dans le cas où le **flux est constant** et avec le maillage de la figure3.3, nous avons abtenus les resultats suivants:

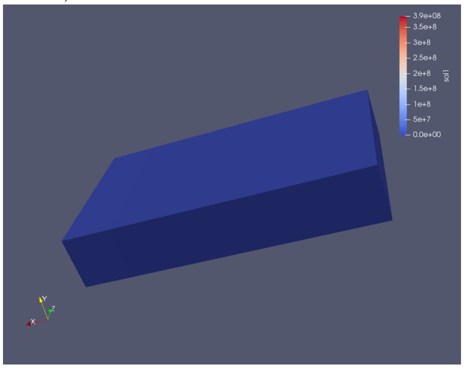

Figure 3.6: Solution à t = 0

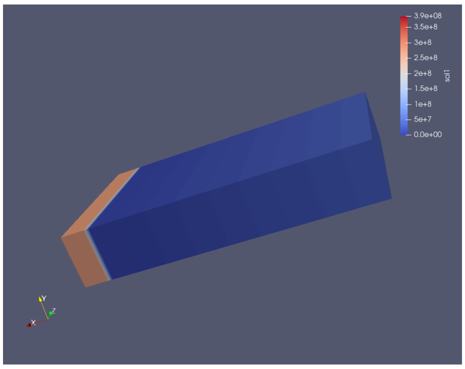

Figure 3.7: Solution à t = 15

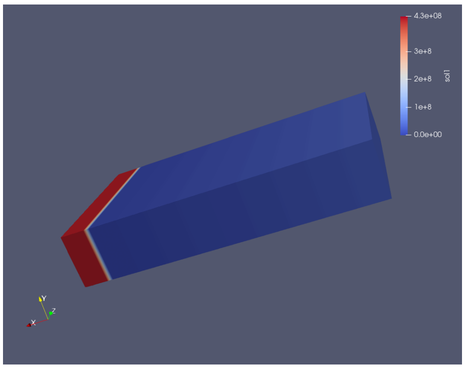

Figure 3.8: Solution à t = 150

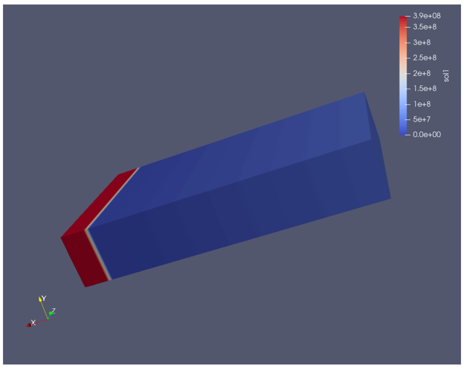

Figure 3.9: Solution à t = 210

Interpretation:

On a obtenu ces simulations. **Cependant** les resultats ne sont pas satisfaisants et semblent pas corrects. On note des valeurs inattendues très grandes. Cela est peut être dû à une erreur dans l'implémentation du code correspondante. Nous n'avons pas pu trouver l'erreur