SIMULATION THERMIQUE D'UNE AILETTE D'UN DISSIPATEUR DE CHALEUR

Roussel Desmond Nzoyem Ngueguin

16 décembre 2019

Introduction

Le programme écrit permet de visualiser le comportement du dispositif de refroidissement d'un microprocesseur. Le dispositif étudié est constitué d'un ventilateur et d'un dissipateur de chaleur rattaché au processeur **(figure 1)**.

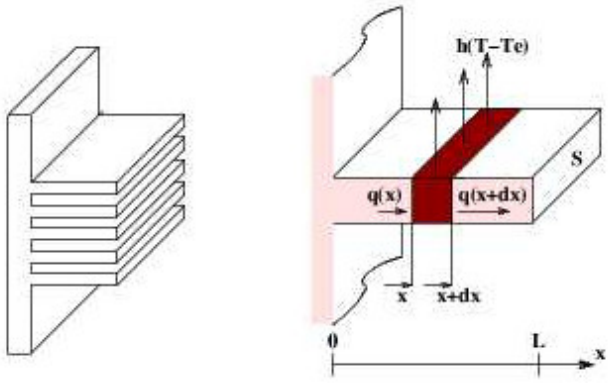


FIGURE 1 – Modelisation d'une ailette du dissipateur

L'équation de la chaleur sur une ailette supposé suffisamment mince pour que la température en un point donné ne dépende que sa position x a un instant t donné est :

$$\rho C_p \frac{\partial T}{\partial t} - \kappa \frac{\partial^2 T}{\partial x^2} + \frac{h_c p}{S}(T - T_e) = 0 \tag{1}$$

Avec ρ la densité, C_p la chaleur spécifique a pression constante, $\varkappa$ la conductivité thermique, h_c le coefficient de transfert de chaleur surfacique, T_e la température ambiante, $S = L_y L_z$ l'aire d'une section transversale et $p = 2(L_x + L_y)$ le périmètre d'une section transversale. L_x, L_y, et L_z étant les longueurs de l'ailette. En x=0, le flux dégagé par le processeur vaut Φ_p. En x=L_x, le flux est supposé négligeable par rapport au flux apporté par la section longitudinale.

Après avoir décrit la structure du programme C++, nous effectuerons des simulations sur une des ailettes du dissipateur.

1 Description de la structure du programme

Une classe « Stationnaire » est définie pour traiter le modèle stationnaire, et une classe « Instationnaire » (dérivée de « Stationnaire ») est définie pour traiter le modèle instationnaire. Le point d'entrée du programme se charge de lire le fichier de configuration et de créer les objets qui vont contenir le problème et le résoudre.

1.1 Exécution du programme

Plusieurs fichiers sont nécessaires à l'exécution du programme :

- **Les fichiers headers** (stationnaire.hpp et instationnaire.hpp) : ils contiennent les déclarations des classes Stationnaire et Instationnaire ;
- **Les fichiers sources** (cooling.cpp, stationnaire.cpp, et instationnaire.cpp) : ils contiennent les définitions des classes. Le fichier cooling.cpp contient la fonction main() qui lit les paramètres. Avec ces données, elle crée un nouveau problème (un objet), qui va "se résoudre". Puis elle supprime l'objet créé.
- **Les commandes gnuplot** (gnuplot_command_*.dat) : ces fichiers contiennent les commandes à exécuter par le programme préinstallé "gnuplot" pour visualiser les solutions 1D.
- **Les fichiers de configuration** (simu_0.cfg, simu_1.cfg, . . . , simu_11.cfg) : localisés dans le répertoire "config_files", ils contiennent les paramètres physiques, géométriques, et de simulation du problème. Ces 17 paramètres peuvent être lus dans n'importe quel ordre, à condition de bien les nommer (l'unité de longueur étant le mètre).

Une fois chacun de ces fichiers dans le répertoire approprié, l'exécution de la commande :

```
g++ cooling.cpp stationnaire.cpp instationnaire.cpp -o cooling && ./cooling simu_1.cfg
```

devrait lancer la simulation 1. Le programme s'assurera de la résolution du problème, de l'écriture des solutions dans des fichiers .csv et .vtk, puis de l'affichage 1D de la solution correspondant au problème (avec "gnuplot").

1.2 Fonctions majeures du programme

Toutes les fonctions du programmes (hormis main())sont des méthodes définies dans les fichiers stationnaire.cpp et instationnaire.cpp. Parmi elles, les plus importantes sont les suivantes :

- **void Stationnaire : :factoLU()** : effectue la factorisation LU de la matrice A
- **void Stationnaire : :descenteRemontee()** : résout le problème AX = F , avec A = LU
- **void Stationnaire : :searchInterval()** : localise les points x_{i00} dans le maillage 1D, nécessaire pour effectuer l'interpolation linéaire
- **void Stationnaire : :interpolationLineaire()** : calcule l'interpolant linéaire aux différents points x_{i00}
- **bool Stationnaire : :ecriture3D(int indice_temps, bool flux_de_chaleur_constant)** : Permet d'écrire un fichier .vtk dans le répertoire "paraview_data" en vue de la visualisation 3D. Elle prend en entrée l'indice de l'itération à laquelle la solution a été calculé (compris entre 0 et N pour les cas instationnaires, et qui vaut -1 dans le cas stationnaire). Ce numéro sera nécessaire pour nommer le fichier créé. La fonction prend aussi en entrée un booléen disant si on traire le problème instationnaire au scénario 1 (=true) ou au scénario 2 (=false). Elle nous retourne un booléen disant si l'écriture a été un succès ou pas.
- **void Stationnaire : :solve()** : fonction virtuelle qui remplit la matrice A, appelle les fonction nécessaires pour résoudre le problème, et écrit la solution stationnaire dans le fichier .vtk
- **void Stationnaire : :solveAnalytiqueStationnaire()** : calcule la solution stationnaire exacte
- **void Stationnaire : :displayPlot() const** : fonction virtuelle qui écrit la solution dans le fichier .csv et effectue la visualisation 1D du problème stationnaire
- **void Instationnaire : :solve()** : fonction qui surcharge la méthode solve() du cas stationnaire, rempli la matrice A, appelle les fonction nécessaires pour résoudre le probelme, et écris les solution instationnaire à diverses itérations dans les fichiers .vtk
- **void Instationnaire : :displayPlot() const** : surcharge la fonction displayPlot() pour écrire dans le fichier .csv, et puis affiche ces solutions (aux différents instants, ou à différentes positions, en fonction du scénario traité)

1.3 Strucure UML

Les autres fonctions et attributs sont representés dans le diagramme de classe UML **(figure 2)**.

1.4 Données generées

Les donnes générées sont les fichiers .csv et .vtk pour visualiser respectivement les solutions 1D et 3D. Les fichiers .csv se trouvent dans le répertoire courant (celui contenant le fichier cooling.cpp), alors que les fichiers. vtk sont créer dans le répertoire "paraview_data".

- **data_stationnaire.csv** : données pour la visualisation 1D de la solution stationnaire calculée et de la solution analytique exacte
- **data_instationnaire_const.csv** : données pour la visualisation 1D de la température du modèle instationnaire à flux constant (Scenario 1) à 6 instants distinct (t=15s, t=30s, t=60s, t=90s, t=150s, et t=210s)
- **data_instationnaire_non_const.csv** : données pour la visualisation de l'évolution de la température du modèle instationnaire avec activation et désactivation du flux de chaleur venant du processeur toute les 30 secondes (Scenario 2) en 3 points donnés (x=0, x=M/2, et x=M)
- **stationnaire.vtk** : données du modèle stationnaire pour la visualisation 3D avec Paraview
- **instationnaire_constant.*.vtk** : données du modèle instationnaire au scénario 1 pour la visualisation 3D avec Paraview
- **instationnaire_non_constant.*.vtk** : données du modèle instationnaire au scénario 2 pour la visualisation 3D avec Paraview

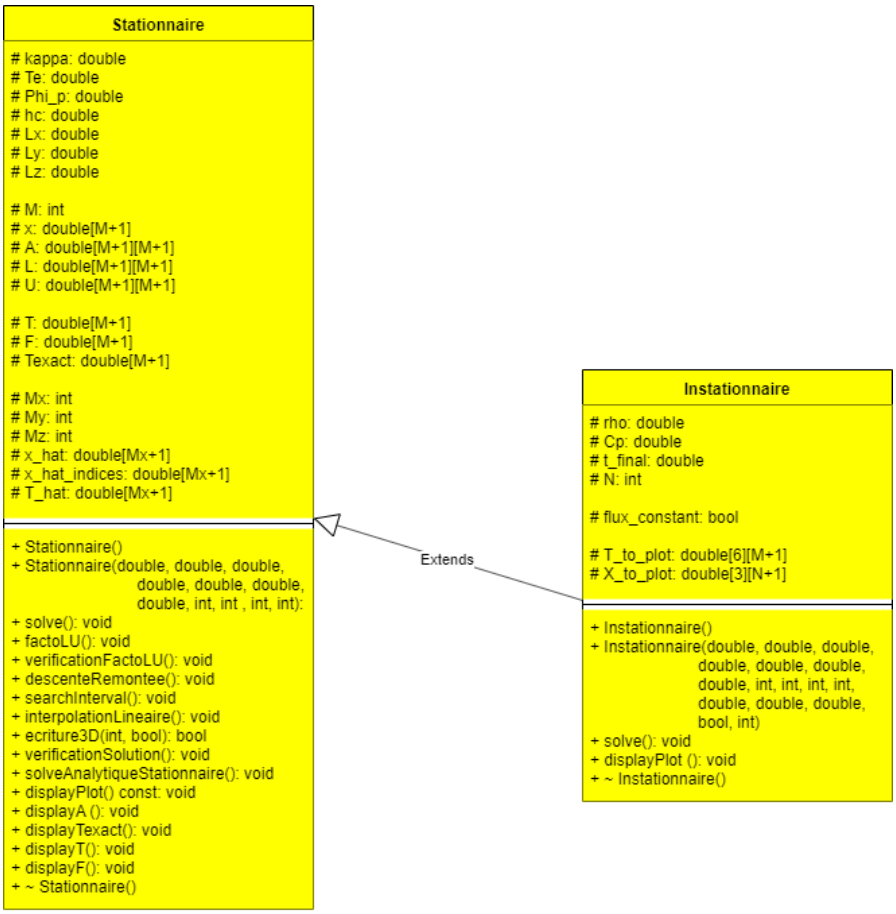


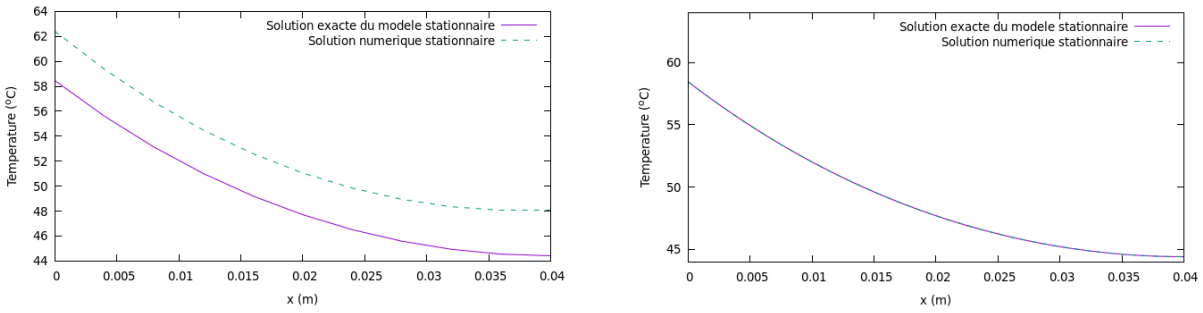
FIGURE 2 – Digramme de classes du programme

2 Simulations

2.1 Vérification du code de calcul

2.1.1 Cas Stationnaire

On utilise les fichiers de configuration simu_0.cfg et simu_1.cfg pour visualiser les températures de l'ailette en 1D respectivement pour M = 10 et M=10000. M étant la taille du maillage utilisé pour la discrétisation du problème.



(a) M = 10 (b) M = 10000

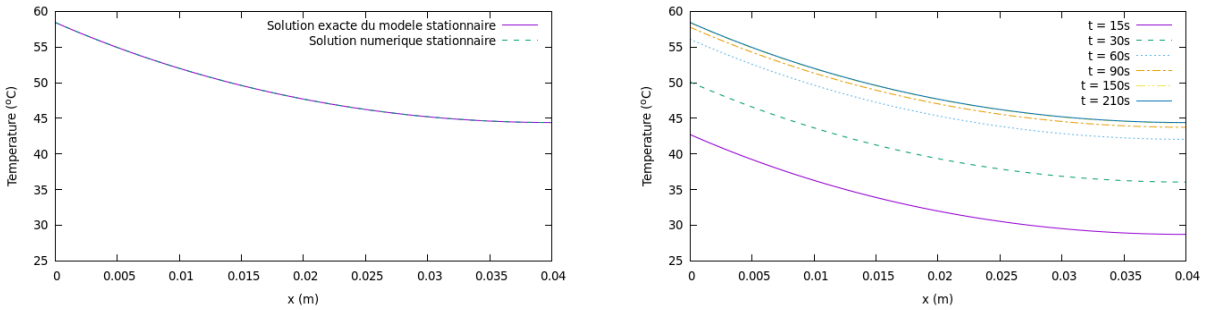
FIGURE 3 – Température dans le modele stationnaire

On constate que la solution calculée par résolution du système linéaire est correcte. Plus le nombre de points de notre maillage est grand, plus les calculs sont précis, et ainsi la solution calculée se rapproche de la solution analytique trouvée a la main.

2.1.2 Cas Instationnaire

Scenario 1 : Flux constant

Nous utilisons ici le fichier de configuration simu_2.cfg



(a) Modele stationnaire

(b) Solution iterative du modele instationnaire

FIGURE 4 – Convergence pour le modele instationnaire vers la solution du modele stationnaire

En 1D, nous pouvons voir que la solution du modèle instationnaire dessiné à différents instants de la simulation se rapproche de la solution du modèle stationnaire **(figure 4)** .

En 3D avec Paraview, on observe le meme résultat lorsqu'on observe l'évolution de la température sur l'ailette **(figures 5 et 6)**.

Scenario 2 : Flux activé et desactivé toutes les 30 secondes

Les hypothèses du scénario 2 diffèrent grandement de celles du modèle stationnaire. En effet, on suppose (implicitement) le flux de chaleur émis par le processeur constant pour le modèle stationnaire. La solution dans le scénario 2 ne converge donc pas forcément vers la solution du modèle stationnaire. Mais elle s'y rapproche néanmoins dans certaines circonstances (quoique lentement), en particulier lorsque le ventilateur est éteint, comme nous le montrerons par la suite.

2.2 Longueur Lx variable

Scenario 1 : Flux constant

Pour trois longueurs L_x différentes (40mm, 80mm, 200mm), visualisons le resultat en 1D puis en 3D. Les fichiers de configuration simu_4.cfg, simu_5.cfg, et simu_6.cfg permettent de réaliser ces simulations **(figures 7, 8, et 9)**.

Dans ce cas stationnaire, on remarque que l'évacuation de chaleur est bien meilleure avec des longueurs Lx de plus en plus grandes. En effet, tous les points (et en particulier le point x=0 qui correspond à la température du processeur) sont moins chauds quand Lx augmente.

Scenario 2 : Flux activé et desactivé toutes les 30 secondes

On utilise ici les fichiers de configuration simu_7.cfg, simu_8.cfg, et simu_9.cfg. La visualisation 3D ici correspond à la température de l'ailette au moment ou le point x=0 est le plus chaud, c'est-à-dire t = 270 s **(figures 10, 11, et 12)**.

Dans ce scénario, on remarque aussi que la température maximale atteinte par le point le plus chaud du système diminue lorsque Lx augmente.

En somme, l'augmentation de la longueur Lx améliore de façon considérable l'évacuation de chaleur dans notre dispositif. Ceci est du à l'augmentation de la surface d'échange (par convection) avec l'air ambient qui augmente lorsque L_x augmente.

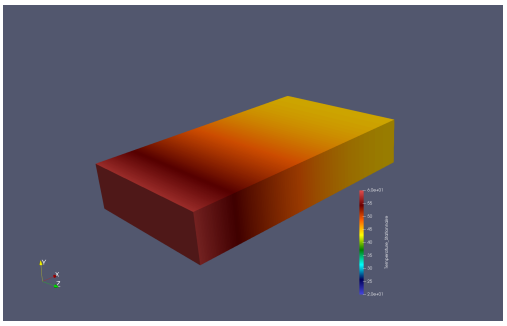
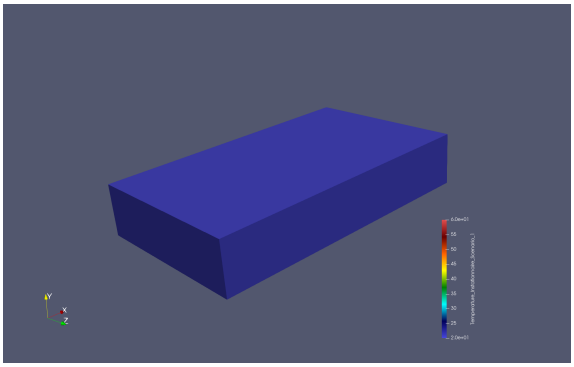
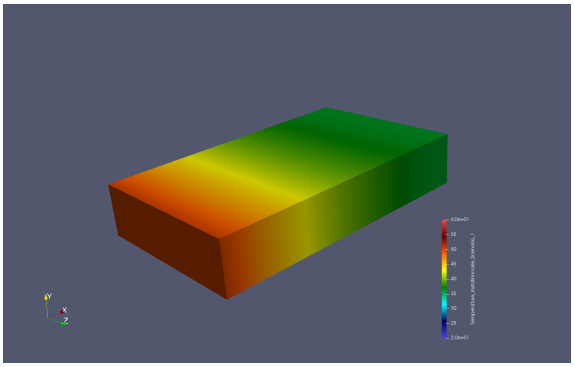


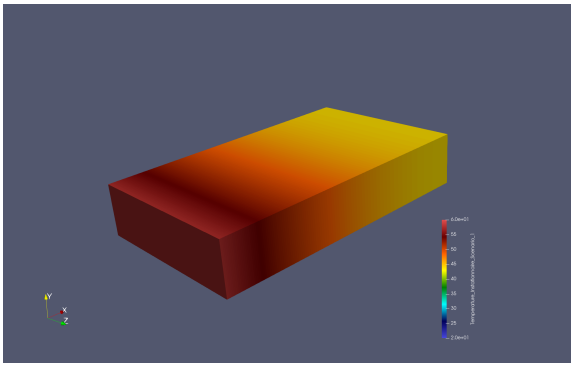
FIGURE 5 – Visualisation 3D de la solution du modele stationnaire



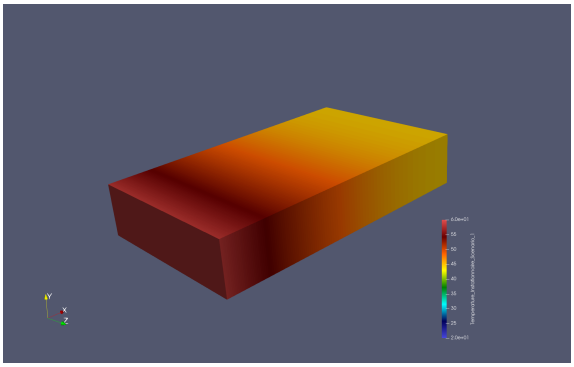
(a) t = 0s



(b) t = 30s



(c) t = 90s



(d) t = 300s

FIGURE 6 – Visualisation 3D de la convergence de la solution du modele instationnaire avec flux constant

2.3 Utilisation ou non du ventilateur

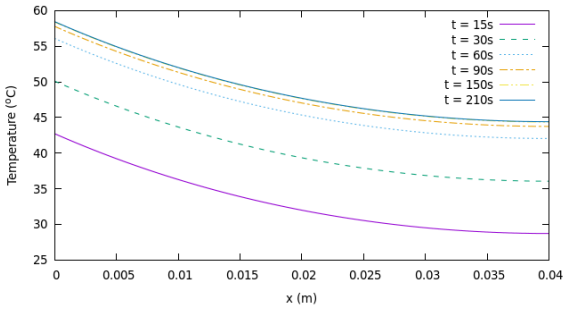
Cas 1 : Ventilateur allumé (h_c = 200 W/(m^2/K))

Ce scénario correspond aux multiples simulations représentées précédemment. Nous reprenons donc le cas Lx = 40 mm dans le scénario 1 (simu4_.cfg, **figure 13**), et dans le scénario 2(simu_7.cfg, **figure 14**).

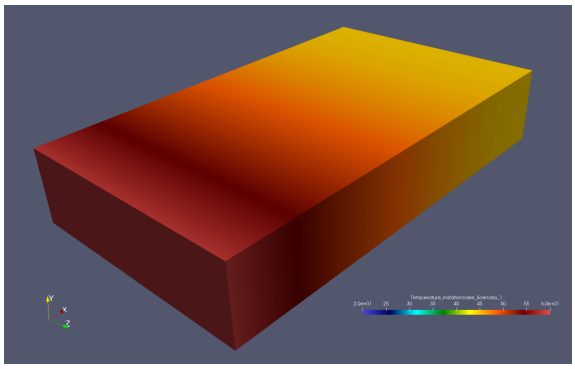
Cas 2 : Ventilateur étteint (h_c = 10 W/(m^2/K))

Nous utilisons ici les fichiers de configuration simu_10.cfg (**figure 15**) et simu_11.cfg (**figure 16**).

Les comparaisons des températures finales (en 3D à droite, sur un intervalle de 20 °C à 250 °C) des cas 1 et 2 montre que lorsque le ventilateur est éteint, les températures augmentent grandement sans toutes fois être évacuées. On obtient ainsi une température fulgurante de 250 °C lorsque le flux de chaleur venant du processeur est constant (**figure 15**). Même lorsque le flux de chaleur est activé et désactivé toutes les 30 secondes, on atteint toujours des températures de l'ordre de 180 °C sur l'ailette, cela au bout de 300 secondes (**figure 16**).

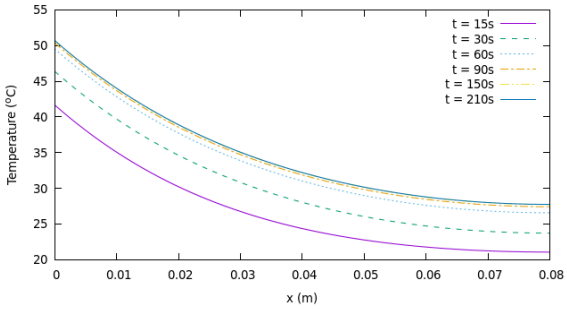


(a) Evolution de la température

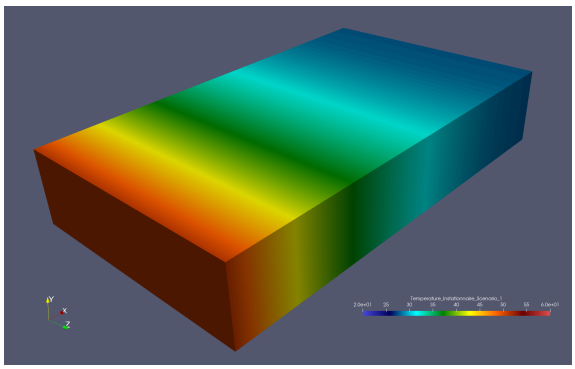


(b) Température finale (à t = 300s)

FIGURE 7 – Scénario 1 - Lx = 40 mm

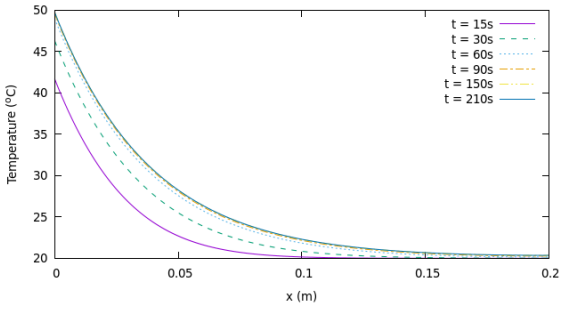


(a) Evolution de la température

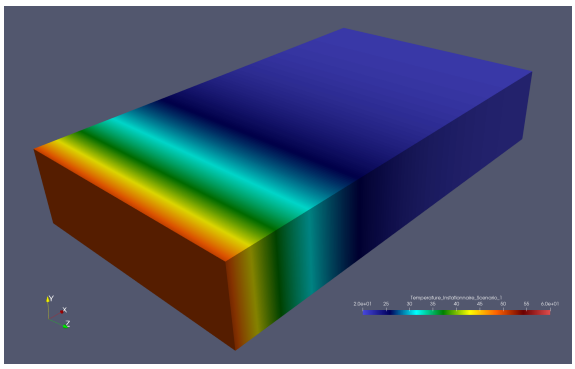


(b) Température finale (à t = 300s)

FIGURE 8 – Scénario 1 - Lx = 80 mm



(a) Evolution de la température



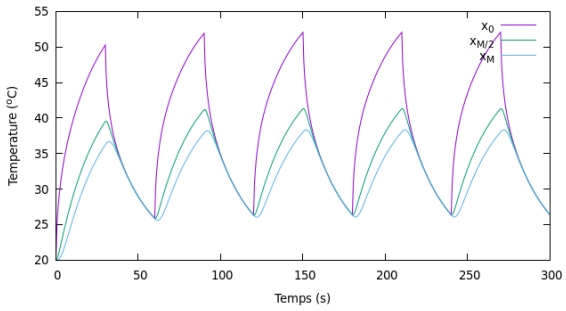
(b) Température finale (à t = 300s)

FIGURE 9 – Scénario 1 - Lx = 200 mm

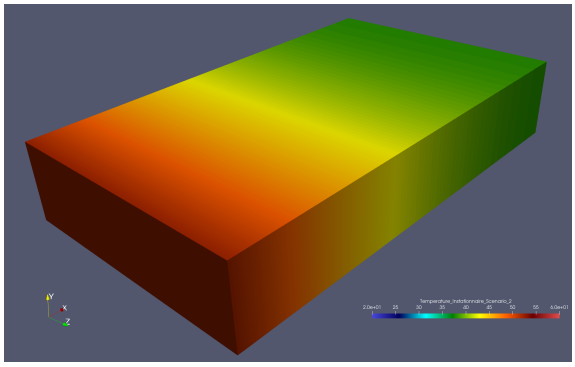
Conclusion

Le projet a bien permis de simuler le comportement thermique d'une ailette du dissipateur de chaleur. La connaissance de la distribution des températures sur cette ailette nous permet d'imaginer le comportement du système tout entier. On a constaté que le dissipateur doit évacuer des chaleurs très élevées lorsque le ventilateur est éteint, ce qui peut causer son disfonctionnement, peu importe que le flux de chaleur soit activé et désactivé toute les 30s. D'autre part, l'augmentation de la taille du dissipateur (ceci en augmentant la longueur L_x de ses ailettes) permet d'assurer une meilleure évacuation de la chaleur.

Le ventilateur a donc une très grande influence sur le système. Il est donc important d'avoir un ventilateur ayant un grand coefficient de transfert surfacique h_c pour avoir de bonnes performances du microprocesseur (en vue d'un surcadençage par exemple). Mais cela vient avec un inconvénient majeur qui est que le ventilateur devient très bruyant. On se demande donc si la méthode de refroidissement liquide n'est pas à envisager dans ces cas.

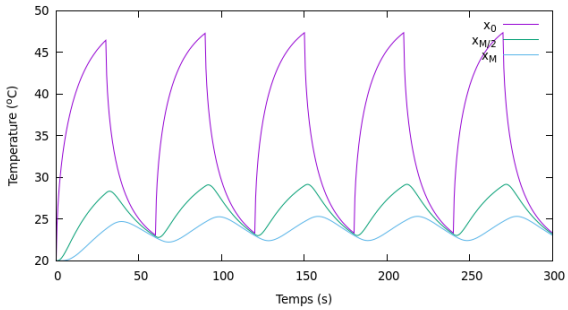


(a) Evolution de la température en 3 points

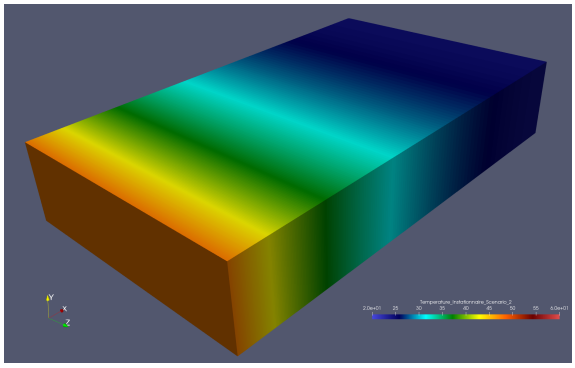


(b) Température de l'ailette à l'instant t = 270s

FIGURE 10 – Scénario 2 - Lx = 40 mm

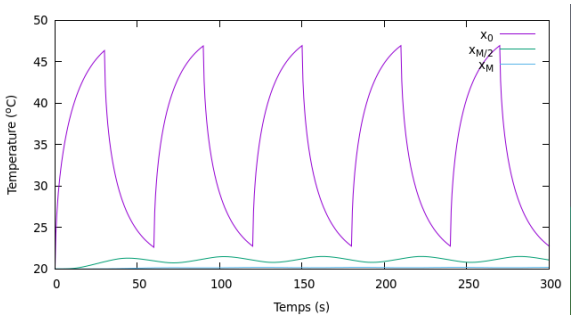


(a) Evolution de la température en 3 points

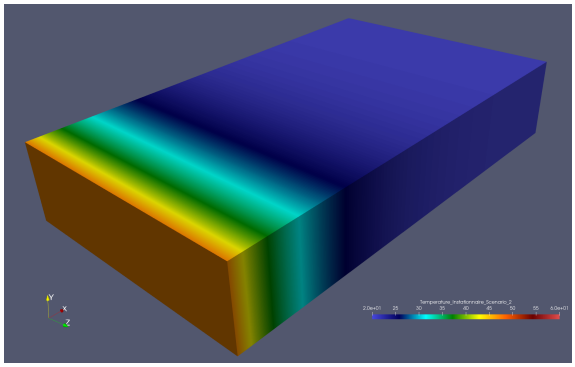


(b) Température de l'ailette à l'instant t = 270s

FIGURE 11 – Scénario 2 - Lx = 80 mm

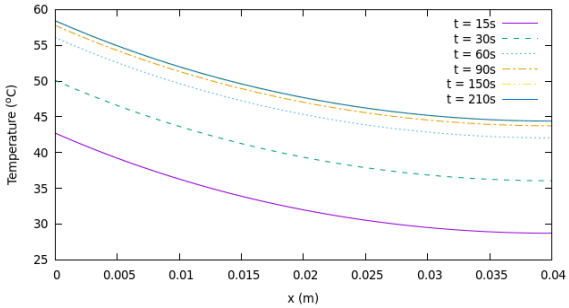


(a) Evolution de la température en 3 points

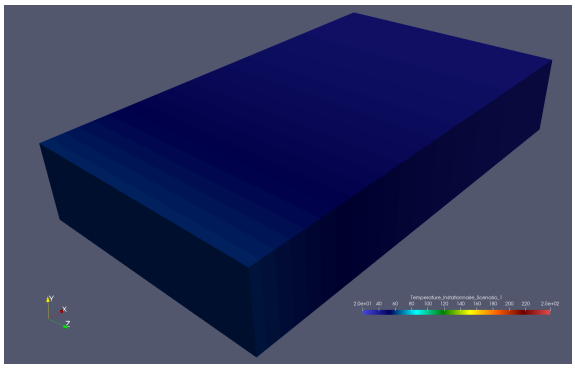


(b) Température de l'ailette à l'instant t = 270s

FIGURE 12 – Scénario 2 - Lx = 200 mm

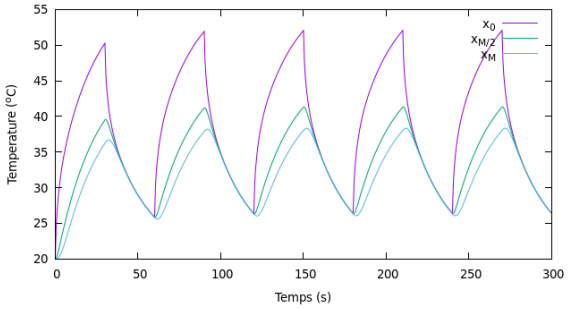


(a) Evolution de la température

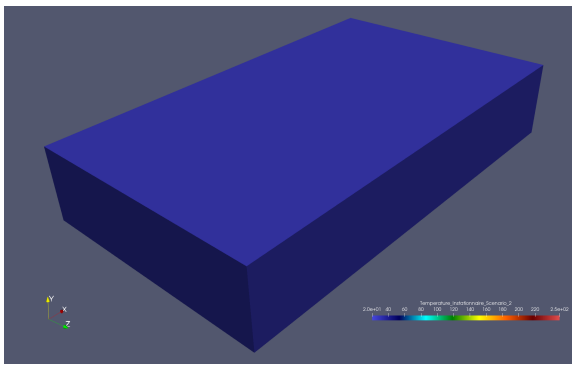


(b) Température finale (à t = 300s)

FIGURE 13 – Ventilateur allumé - Scenario 1

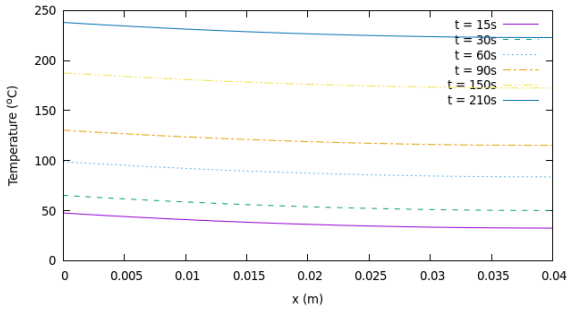


(a) Evolution de la température pour 3 points

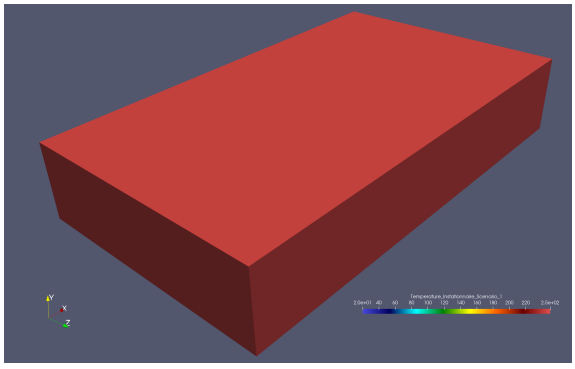


(b) Température finale (à t = 300s)

FIGURE 14 – Ventilateur allumé - Scenario 2

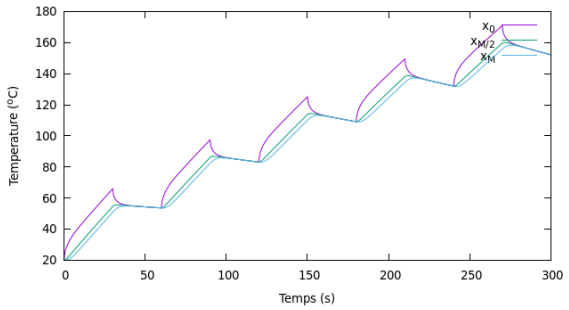


(a) Evolution de la température

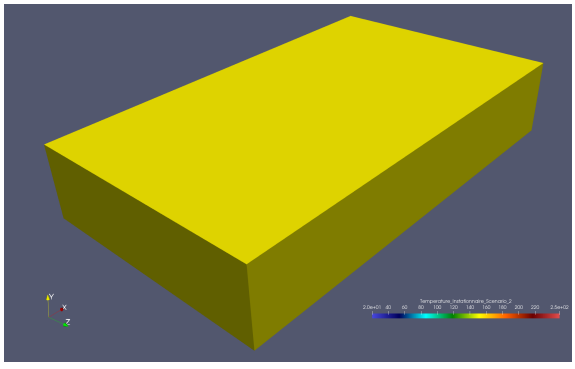


(b) Température finale (à t = 300s)

FIGURE 15 – Ventilateur étteint - Scénario 1



(a) Evolution de la température pour 3 points



(b) Température finale (à t = 300s)

FIGURE 16 – Ventilateur étteint - Scénario 2