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

**INTRO**

Le programme écrit permet de visualiser le comportement d’un dispositif de refroidissement d’un microprocesseur. Le dispositif étudié est constitué d’un ventilateur et d’un dissipateur de chaleur rattaché au processeur [fig 1].

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

Avec RHO la densité, CP la chaleur spécifique a pression constante, HC la conductivité thermique, HC le coefficient de transfert de chaleur surfacique, S = LyLz l'aire d'une section transversale et p = 2(Lx + Ly) le périmètre d'une section transversale. Lx, Ly, et Lz étant les longueurs de l’ailette. En x=0, le flux dégage par le processeur vaut PHI\_P. En x=Lx, le flux est supposé négligeable par rapport au flux apporte par la section longitudinale.

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

**CONCLUSION**

Le projet a bien permis de simuler le comportement thermique d’une ailette du dissipateur de chaleur, et de voir que la connaissance du transfert de chaleur sur cette ailette nous permet d’imaginer le comportement du système tout entier. On a donc constaté que le dissipateur se charge de dissiper des températures très élevées lorsque le ventilateur est éteint, ce qui peut causer son disfonctionnement, peu importe si le flux de chaleur est activé et désactivé toute les 30s. D’autre part, l’augmentation de la taille dissipateur (ceci en augmentant la longueur Lx 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 hc pour avoir de bonnes performances du microprocesseur en vue d’un surcadençage par exemple. Mais cela vient à des couts puisque le ventilateur devient alors très bruyant. On se demande donc si la méthode de refroidissement liquide n’est pas à envisager dans ces cas.

**DESCRIPTION**

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.

**EXECUTION**

Plusieurs fichiers sont nécessaires à l’exécution du programme :

* Les fichiers headers (stationnaire.h et instationnaire.h) : ils contiennent la déclaration des classes Stationnaire et Instationnaire ;
* Les fichiers sources (cooling.cpp, stationnaire.c, et instationnaire.c) : Ils contiennent la définition des classes. Le fichier cooling.cpp est 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\_stat.dat, gnuplot\_command\_insta\_const.dat, gnuplot\_command\_insta\_non\_const.dat) : ils 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 et géométriques, ainsi que ceux de la simulation avec Paraview du problème qu’on veut résoudre et visualiser. Ces 17 paramètres peuvent être lues dans n’importe quel ordre, à condition de bien les nommer, (l’unité de longueur étant le mètre).

Une fois tout ces fichiers dans le même répertoire, 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).

**FONCTIONS**

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 factoLU de la matrice A, un attribut de la classe Stationnaire ;
* ***void Stationnaire::descenteRemontee() :*** Résous le problème Ax = b pour une matrice A sous forme LU
* ***void Stationnaire::searchInterval() :*** Localise les points xi00 dans le maillage 1D, nécessaire pour effectuer l’interpolation linéaire.
* ***void Stationnaire::interpolationLineaire() :*** Calcule l’interpolant linéaire aux différents points xi00
* ***bool Stationnaire::ecriture3D(int indice\_temps, bool flux\_de\_chaleur\_constant) :*** Permet d’écrire les fichiers .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 le cas instationnaire, et qui vaut -1 dans le cas stationnaire). Ce numéro sera nécessaire pour nommer le fichier créer. La fonction prend aussi en entrée un booléen disant si on traire le problème instationnaire au scenario 1 (=1) ou au scenario 2 (=0). Il 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 écris 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 fonction **solve()** du cas instationnaire. Rempli la matrice A, appelle les fonction nécessaires pour résoudre le probelme, et écris les solution instationnaire a diverses iterations dans les fichier .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).

**DONNEES GENEREES**

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 calcule et 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 :*** donnes 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 seconds (Scenario 2) en 3 points donnes (x=0, x=M/2, et x=M)
* ***stationnaire.vtk***: donnes du modèle stationnaire pour la visualisation 3D avec Paraview
* ***instationnaire\_constant.\*.vtk***: donnes du modèle instationnaire au scenario 1 pour la visualisation 3D avec Paraview
* ***instationnaire\_non\_constant.\*.vtk***: donnes du modèle instationnaire au scenario 2 pour la visualisation 3D avec Paraview

**VERIFICATION DU CODE**

Nous nous servirons des visualisations 1D et 3D pour montrer que nos solutions convergent bien vers la solution exacte calculeee de façon analytique.

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. (On utilise respectivement les fichiers de configuration simu\_0.cfg et simu\_1.cfg) pour ces deux simulations.

Fig2

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.

Cas Instationnaire

*Scenraio 1*

Nous utilisons ici le fichier de configuration simu\_2.cfg

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 ci-haut.

En 3D avec Paraview, on observe le mem résultat lorsqu’on observe l’évolution de la température sur l’ailette

*Scenario 2*

Les hypothèses du scenario 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 scenario 2 ne converge donc pas forcement 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.

*LONGUEUR Lx Variable*

*Scenario 1*

*Pour trois longueurs Lx différentes (40mm, 80mm, 200mm), visualisons le résultat en 1D puis en 3D.* Les fichiers de configuration ***simu\_4.cfg***, ***simu\_5.cfg***, et ***simu\_6.cfg*** permettent de réaliser ces simulations.

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, le point x=0 (qui correspond à la température du processeur) est moins chaud quand Lx augmente.

*Scenario 2*

On utilise ici les fichiers de configuration ***simu\_7.cfg***, ***simu\_8.cfg***, et ***simu\_9.cfg***. La visualisation 3D 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.

Dans ce scenario. 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 Lx augmente.*

**Utilisation ou non du ventilateur**

*Ventilateur allumé (hc = 200)*

Ce scenario correspond aux multiples simulations représentées précédemment. Nous reprenons donc le cas Lx = 40 mm dans le scénario 1 (simu\_4.cfg, figure 13), et dans le scenario 2(simu\_7.cfg, , figure 14).

Ventilateur éteint (hc = 10)

Nous utilisons ici les fichiers de configuration simu\_10.cfg et simu\_11.cfg

Les comparaisons des températures finales (en 3D à droite, sur un intervalle de 20oC a 250oC) montrent que lorsque le ventilateur est éteint, les températures augmentent grandement sans toutes fois être évacuées. On obtient ainsi une température fulgurante de 250oC 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 oC sur l’ailette, cela au bout de 300 secondes (figure 16).

Il se charge d'évacuer les chaleurs.

allumé

Température à

A

Mais cela vient avec un inconvénient majeur qui est que le ventilateur devient très bruyant.