Investigating the heat dissipation in microprocessors

Kei Tsun Yeung (CID: 01054596)

20-12-2017

Number of words: 2482

Abstract

The general heat transport equation in steady state in the form of a Poisson equation for temperature was solved for a microprocessor with a ceramic case, for the cases with and without a heat sink and for natural and forced convection, in search for an optimum structure of heat dissipation model of the microprocessor. The temperature in the microprocessor was found to decrease with an increasing height, separation and number of fins, and also forced convection helped in decreasing the average temperature by a large extent than natural convection.

1 Introduction

A typical microprocessor in a computer produces around 95 W of thermal power, which is a huge amount of heat that can increase the temperature of the microprocessor to a point where it stops working [1]. Therefore, an effective way of dissipating this heat at a high enough rate is necessary for a microprocessor to work.

In this project, an optimum solution to this was achieved through investigating into a number of scenarios and varying the parameters, including adding a heat sink with different number, width and height of the fins, and also with natural and forced convection by adding a wind current at the surface.

2 Theory

2.1 Heat transport equation

The general heat transport equation in steady state gives the relationship between the heat flux through an area ϕ (units W/mm^2) of an object and the heat generated or consumed q (units W/mm^3) at a given point in space $\underline{r}$ and is given by [1]

$$\nabla\phi = q(\underline{r}). \tag{1}$$

Fourier's law of heat conduction expresses the heat flux ϕ as the derivative of temperature T in a material [2] as given by

$$\phi = -k\nabla T, \tag{2}$$

where k (units W/(mm K)) is the thermal conductivity of the material.

Combining equations 1 and 2 gives

$$-k\nabla^2 T = q(\underline{r}), \tag{3}$$

or

$$-k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = q(\underline{r}) \tag{4}$$

in two dimensions.

2.2 Boundary conditions

At the surface of an object, heat dissipates into another object in contact with it or the ambient. The ambient case is where boundary conditions are applied. In the analysis a Neumann boundary condition is given in terms of the heat flux ϕ (hence the derivative of temperature) by

$$\phi_s = -k\left(\frac{\partial T}{\partial x} + \frac{\partial T}{\partial y}\right) = h(T_{\text{surf}} - T_{\text{a}}), \tag{5}$$

where h (units W/(mm^{2}K)) is the heat transfer coefficient whose value depends on the convection mechanism, and T_{a} is the ambient temperature, taken as 20°C = 293K. For natural convection without any driving wind currents, h_{natural} is given by

$$h_{\text{natural}} = 1.31 \times 10^{-6}(T_{\text{surf}} - T_{\text{a}})^{1/3}. \tag{6}$$

For forced convection with wind currents, h_{forced} is given by

$$h_{\text{forced}} = (11.4 + 5.7v) \times 10^{-6}, \tag{7}$$

where v is the wind speed.

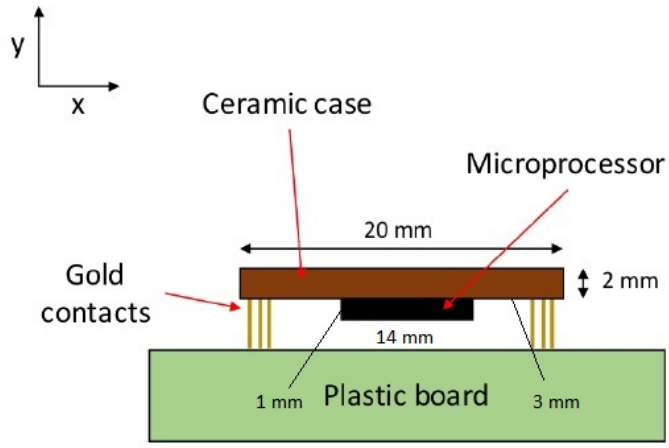

Figure 1: Diagram showing the microprocessor-ceramic case system without the heat sink [1].

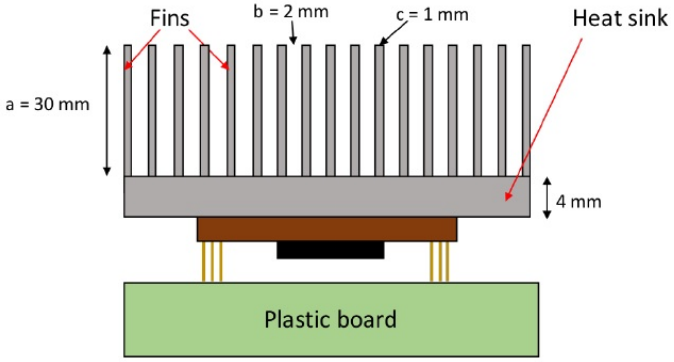

Figure 2: Diagram showing the microprocessor-ceramic case system with the heat sink [1]. The number of fins, the height a and the width b of the fins were varied to test different designs.

3 Method

3.1 The system under consideration

Figures 1 and 2 show the microprocessor and ceramic case system under consideration, without and with a heat sink to help dissipate heat respectively [1]. The gold contacts and the plastic board were not included in the analysis for simplicity.

The microprocessor had a width of 14 mm and a height of 1 mm, and in the third dimension the objects were assumed to be infinite in size and ignored in the analysis. The objects were assumed to be centred around each other so the ceramic case with length 20 mm extended 3 mm in each direction from the microprocessor in the x-axis, for example. The coordinate system was as defined in Figure 1 and the zero point on each axis was taken as the leftmost point and the lowest point in the system respectively. For example, in Figure 1, $x = 0$ on the fictitious points to the left of the ceramic case and $y = 0$ on the fictitious points below the microprocessor.

The heat sink was characterised by a few variable parameters, the number of fins, the height of a fin a and the separation between fins b as in Figure 2. The base of the heat sink had a fixed height of 4 mm and all fins had a fixed width c of 1 mm. The heat sink was assumed to have a greater length than the ceramic case in all cases and so the positions of the ceramic case and the microprocessor were shifted to centre around the heat sink. Natural convection and forced convection were considered for the situation with a heat sink by varying the formula for h as in equations 6 and 7.

3.2 Finite difference scheme

To evaluate the second and first order derivatives as given in equations 4 and 5 respectively, a finite difference scheme was used, where the derivatives in temperatures were expressed as a finite difference between neighbouring temperature values. In particular, the central difference scheme (CDS) which uses values from neighbouring points in both directions was used to obtain a better estimate. In this scheme, the derivatives are given by [3]

$$\frac{\partial T}{\partial x} = \frac{T(x + \Delta x) - T(x - \Delta x)}{2\Delta x} \tag{8}$$

and

$$\frac{\partial^2 T}{\partial x^2} = \frac{T(x + \Delta x) - 2T(x) + T(x - \Delta x)}{(\Delta x)^2} \tag{9}$$

respectively, where Δx is the distance between neighbouring points in the mesh grid (see section 3.3).

3.3 Mesh grid and fictitious points

3.3.1 Mesh grid - `meshval` and `data`

To implement the finite difference scheme on a computer, a mesh grid which represented the points in space of the system under consideration was generated using an array to store the temperatures at the different points (data attribute

`meshval` in classes `meshclass` and `multimesh`). The distance between neighbouring mesh points (step size) in this analysis was 0.2 mm. When an object is initialised, an estimate of the temperature `Tguess` must be given and this was stored as the initial values of the internal points in `meshval`. The relevant index of a point was then used and values at neighbouring points were obtained by changing the index by 1.

Another array of the same size was also created to store the nature of a point in the mesh (data attribute `data` in classes `meshclass` and `multimesh`). The identification numbers used in this matrix are given in Appendix A.

3.3.2 Fictitious points

As seen in equations 8 and 9, the evaluation of the derivatives using CDS required values from both neighbouring points. In order to make this possible for the boundary points, points immediately outside the boundaries, or fictitious points, had to be added to the mesh grid. For a boundary point with temperature $T_{0,j}$, substituting equation 8 into 5 gives the temperature for the fictitious point [4]

$$T_{-1,j} = T_{1,j} - \frac{2h\Delta x}{k}(T_{0,j} - T_{\mathrm{a}}) \qquad (10)$$

in the x-direction, where $T_{\mathrm{surf}} = T_{0,j}$.

The values for all fictitious points were calculated and appended into `meshval` using the function `updatebc`. The initial values were calculated using `Tguess` for the internal and surface points $T_{1,j}$ and $T_{0,j}$ in equation 10. After one iteration of the Jacobi method yielded a new set of values for the internal points, the new values at these points were recalculated using `updatebc`. This was repeated using the `iterateJacobi` function until the change in the average temperature in the microprocessor region was less than 1×10^{-5}, the tolerance.

3.3.3 Combining mesh grids

The `meshclass` module was designed for a single object. To generalise to a larger number of objects, a `multimesh` object, `system`, was created from two `meshclass` objects, `micro` and `ceramic`, using the coordinates of the two objects to work out the coordinates of the combined grid and storing temperature values in the corresponding grid space. The `multimesh` object allowed for internal points with different k, q and `Tguess` values for different objects to be iterated at the same time as opposed to only one value allowed in the `meshclass` module. The k, q and `Tguess` values for different objects in a `multimesh` were stored in the attribute `values` in an array form.

The `combine` function can be called for a `multimesh` object to include a third `meshclass` object and so on. It was called to include the base and the fins of the heat sink. The function was similar to the initialisation of the `multimesh` except that care was taken when shifting the `multimesh` mesh grid to adjust to the new object introduced in order to keep the shape unchanged. The shifts in both directions were evaluated using `xshift` and `yshift` in the function.

3.4 Jacobi method

3.4.1 Theory

The iterative Jacobi method was used to solve equation 4 in two dimensions [5]. A matrix $\mathbf{A}$ can be separated into a diagonal $\mathbf{D}$, a lower $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$, i.e. $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$. Expressing the equation $\mathbf{A}\underline{x} = \underline{b}$ in this form and rearranging gives

$$\underline{x}_{n+1} = -\mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U}) \cdot \underline{x}_n + \mathbf{D}^{-1} \cdot \underline{b}, \qquad (11)$$

where n is the number of iterations. From a starting guess x_0, equation 11 can be iterated for a large number of times until a convergence criterion is met. The one used in this analysis was the fractional change in the norm of the solution vector [5]

$$\varepsilon = \left| \frac{||\underline{x}_{n+1}|| - ||\underline{x}_n||}{||\underline{x}_n||} \right|. \qquad (12)$$

When this dropped below a specified tolerance, the iteration was stopped and the solution x_n was accepted.

3.4.2 Implementation

Two ways of implementing the Jacobi method were tested, which were the pictorial operator method and the matrix roll method.

The pictorial operator shows the dependence of a point on the neighbouring points. For the

Jacobi method, the iterating equation in pictorial operator form is given by

$$T_{i,j}^{n+1} = \frac{1}{4}\begin{bmatrix} & 1 & \\ 1 & 0 & 1 \\ & 1 & \end{bmatrix} T_{i,j}^{n} - \frac{\Delta^2}{4}\rho_{i,j}, \qquad (13)$$

where Δ is the step size in either direction and $\rho_{i,j} = -q_{i,j}/k_{i,j}$ as in equation 4 is the source term. A `for` loop was run over all the internal points in the mesh grid and this was repeated until the convergence criterion was met.

The matrix roll method was derived from the pictorial operator as well. Instead of taking each point, the function `np.roll` was used to shift the rows and columns once in both directions on the x- and y-axes separately. At the same position in the mesh grid, the original value was replaced by that of one neighbouring point. Doing this four times and adding them up yielded the same value as using the pictorial operator. This method was predicted to be faster as the values of all the points in the mesh grid can be obtained at the same time using only 4 `roll` operations and a matrix addition.

The `Jacobi` and `Jacobiroll` functions were used to implement the above two methods respectively. The tolerance used was to measure the norm of the change in the solution vector of the whole mesh, until it dropped below 5×10^{-6}. The whole mesh was used here instead of only the microprocessor region because the Jacobi iteration was more related to finding the solution of the whole mesh grid while the final `iterate` function was more related to the physical problem of finding the temperature of the microprocessor so it could be stopped whenever the temperature inside the microprocessor was not changing by much.

4 Discussion and Results

4.1 Numerical accuracy

The main source of uncertainty in this analysis was in the numerical accuracy of the data storage in Python. Firstly, there were only a finite number of points in the mesh grid and so the system cannot be fully represented by the mesh grid. Using the finite difference scheme gave a truncation error in the order of Δ^2 [3]. When evaluating the positions of objects in the grid, the indices had to be rounded to the nearest grid point as well.

There was an unexpected behaviour while observing the change in the norm of the solution vector: this value decreased initially, then asymptoted and increased again. It was also noted that the faster the convergence in the beginning, the earlier the asymtotic behaviour occurred. This can be attributed to the numerical precision to which Python can store the floating point values. It was for this reason that the above tolerance levels had to be chosen carefully so that it was above this asymptotic value, so that a solution can be obtained. However, this made the tolerance a large value compared to accepted ones (of the order 1×10^{-14} [5]) and so this means that the solution had not fully converged yet and the values obtained would be less accurate.

4.2 Spectral radius and the convergence of the Jacobi method

From the matrix $\mathbf{A}$ as in the equation $\mathbf{A}\underline{x} = \underline{b}$, whether the Jacobi method can converge for this matrix can be known by checking whether $\mathbf{A}$ is strictly diagonally dominant. This will mean that the spectral radius of the update matrix $\mathbf{T} = \mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U})$, which is defined as the largest magnitude of its eigenvalues, is smaller than 1 [5]. For a second derivative, from equation 9, the diagonal elements of $\mathbf{A}$ are -4 while the off-diagonal elements are 1. So, it is obvious that the matrix for a second derivative equation is diagonally dominant and so it should converge using the Jacobi method.

4.3 Run time

For a system with a large mesh grid, optimising the code can lead to a huge reduction in the run time. The pictorial operator method described in 3.4.2 was found to be slow as it involved a `for` loop that ran over the whole mesh grid, which means one operation for each point, so the running time went up quickly as the number of points in the mesh grid increased. The rate of one iteration was around 1 per second. After changing to the matrix roll method, the rate increased to about 7-8 iterations per second as it involved matrix addition and it updated all values in the mesh grid at the same time.

The main contribution to the running time in the code was found to be the `for` loop that searched for fictitious points over the whole mesh grid and updating their values in the `updatebc` function. However, due to time constraints, it was not possible to obtain a faster method than this to look for fictitious points other than manually inputting their locations.

Using an iterative method like Jacobi normally allowed for a quicker run time than an analytical method to find the inverse of the matrix, such as LU decomposition [5]. This is because in methods like LU decomposition, forward and backward substitutions were carried out which means one operation per point on the mesh grid, while only a few matrix additions per iteration was required for iterative methods.

4.4 Code validation

Two extreme cases were used to validate the code for the microprocessor and ceramic case system: when $h = 0$, i.e. no convection and $h = 1 \times 10^{10}$, i.e. near infinite rate of convection. The temperature in the microprocessor cannot converge for the small h case, which was as expected because without any heat dissipation, the temperature of the system was expected to increase continually and steady state cannot be reached. It was found to be close to the ambient temperature for the large h case, which was as expected if the rate of convection was high enough, any heat from the microprocessor can be dissipated much faster than can be produced.

4.5 Without heat sink and natural convection

The situation with natural convection without a heat sink is as shown in Figure 3. As expected, the heat spreaded out from the middle of the microprocessor towards the air and the ceramic case. Therefore, the inner part of the microprocesssor was hotter than the outside and the ceramic case. The average temperature of the whole system was around 1000 K = 727 °C. Considering the fact that the microprocessor required a temperature of $60 - 80$°C to function normally, natural convection without a heat sink was not sufficient to allow the microprocessor to operate.

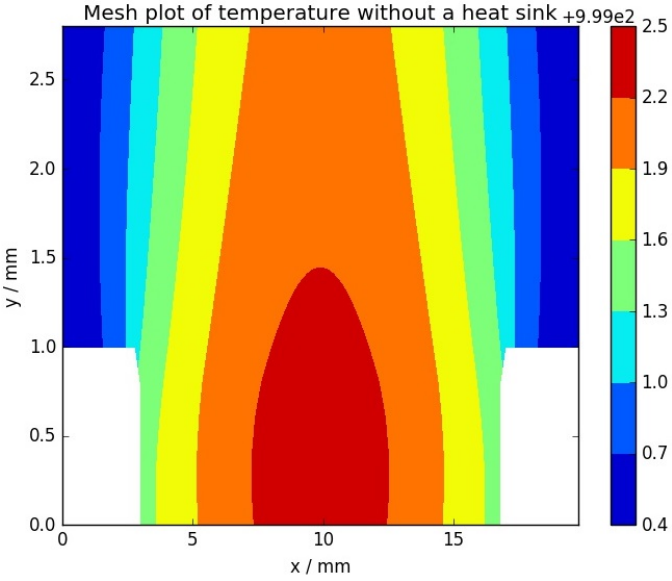

Figure 3: A contour plot of the temperatures over the microprocessor-ceramic case system without a heat sink. The temperature scale on the right is + 999 K.

4.6 With heat sink and natural convection

The initial test case with the heat sink was with 7 fins, height $a = 30$ mm and $b = 5$ mm, as shown in Figure 4. The average temperature had dropped to around 300 K = 27 °C. This would allow the microprocessor to work normally. With the heat sink, the temperature was spread out in a greater range (about 7.5 K from 295.5 to 303.0 K) than without (about 2.1 K). So, the heat was dissipated much faster with the heat sink.

This was then compared with several test cases. Firstly the number of fins was changed to 8 and 10 fins, shown in Figures 5 and 6 respectively. The temperature in the middle of the microprocessor and ceramic case was lower with an increasing number of fins, as expected since this increased the contact area with the air.

The height a was changed to 20 mm and 50 mm and this is as shown in Figures 7 and 8 respectively. It was found that the greater the height a, the lower the average temperature in the middle, again because of a greater contact area.

The separation between fins b was then varied to 8 mm and 11 mm, shown in Figures 9 and 10 respectively. As the separation increased, the middle part of the system cooled down quickly and the maximum temperature when $b = 11$ mm

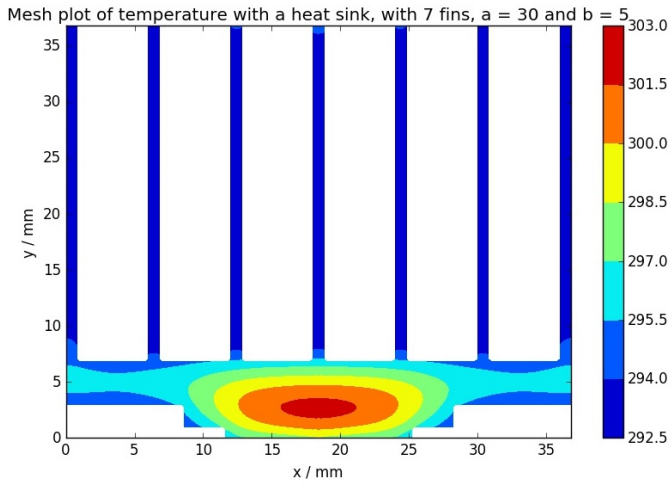

Figure 4: A contour plot of the temperatures of the system with a heat sink, with 7 fins, $a = 30$ mm and $b = 5$ mm.

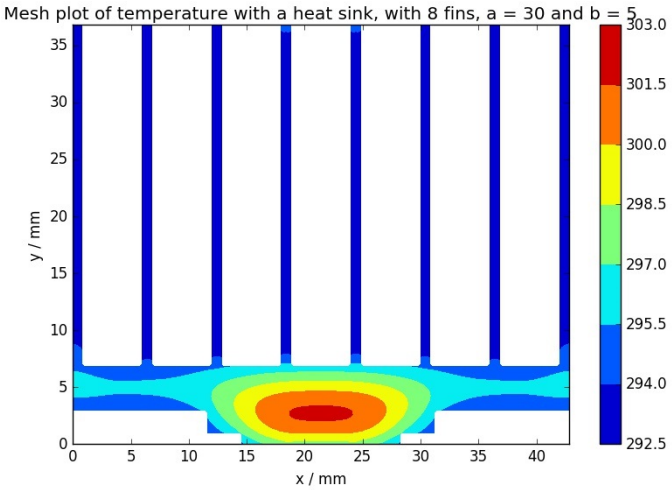

Figure 5: A contour plot of the temperatures of the system with a heat sink, with 8 fins, $a = 30$ mm and $b = 5$ mm.

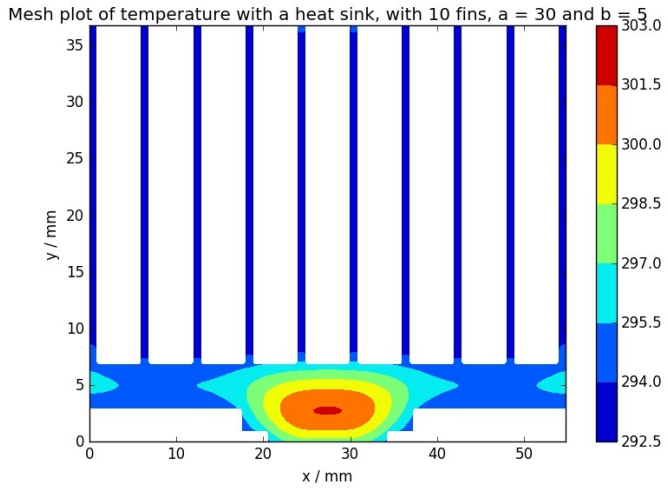

Figure 6: A contour plot of the temperatures of the system with a heat sink, with 10 fins, $a = 30$ mm and $b = 5$ mm.

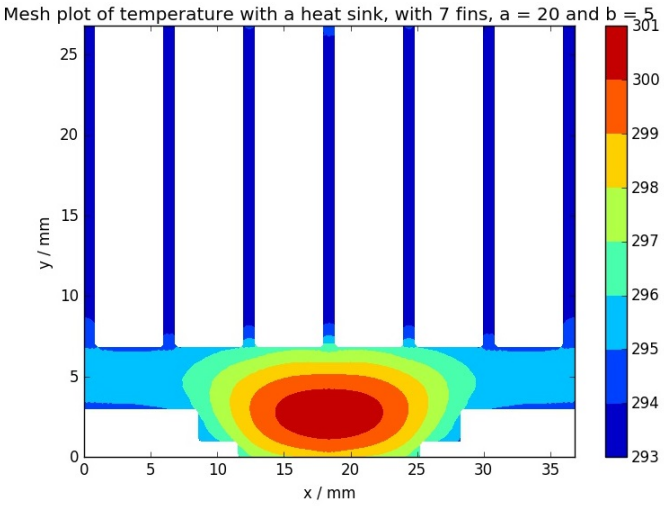

Figure 7: A contour plot of the temperatures of the system with a heat sink, with 7 fins, $a = 20$ mm and $b = 5$ mm.

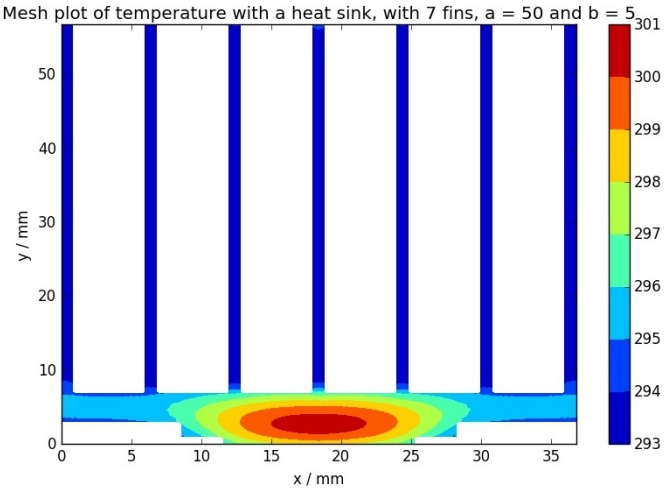

Figure 8: A contour plot of the temperatures of the system with a heat sink, with 7 fins, $a = 50$ mm and $b = 5$ mm.

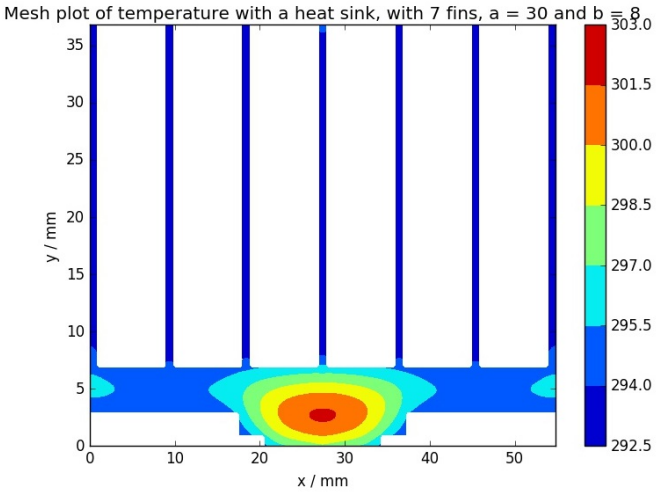

Figure 9: A contour plot of the temperatures of the system with a heat sink, with 7 fins, $a = 30$ mm and $b = 8$ mm.

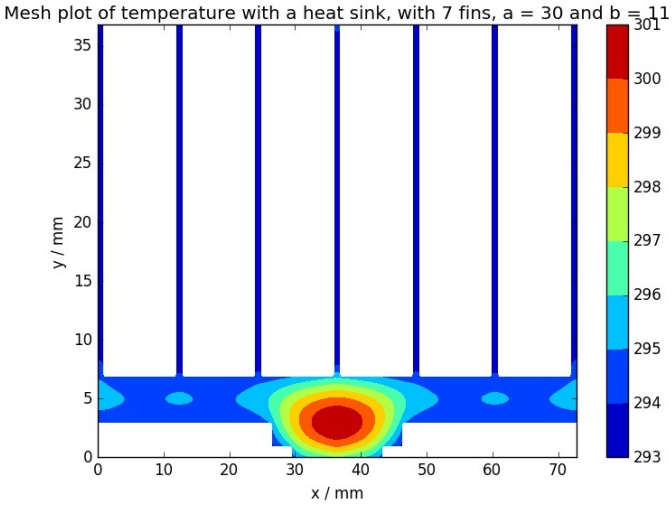

Figure 10: A contour plot of the temperatures of the system with a heat sink, with 7 fins, $a = 30$ mm and $b = 11$ mm.

was lower than the previous plots. This was attributed to the fact that the further the fins were away from each other, the more air there was in between them to convect, hence a higher convection and cooling rate.

4.7 With heat sink and forced convection

The initial test case with forced convection was tested and the resultant contour plot is as shown in Figure 11. Comparing with Figure 4, the average temperature was clearly lower in the microprocessor and ceramic case region, and the heat was more spread out. It showed that forced convection can help in heat dissipation by a great extent.

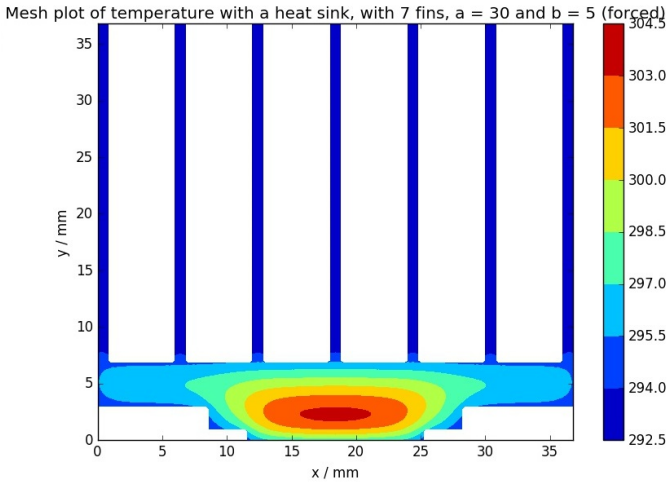

Figure 11: A contour plot of the temperatures of the system with a heat sink and forced convection, with 7 fins, $a = 30$ mm and $b = 5$ mm.

5 Conclusion

The heat transport equation in steady state was solved for a microprocessor with a ceramic case, with and without a heat sink and for both natural and forced convection. The temperature in the microprocessor was found to decrease with an increasing height, separation and number of fins, and also forced convection helped in decreasing the average temperature by a large extent than natural convection.

References

[1] Uchida Y, Scott P, Alonso Alverez D. *Project B4: Heat dissipation in microprocessors.* Imperial College London; 2017.

[2] Gooch JW. *Fourier's Law of Heat Conduction.* Encyclopedic Dictionary of Polymers. Springer; 2011.

[3] Uchida Y. *Finite Difference Methods - Numerical Differentiation.* Imperial College London; 2017.

[4] Alonso Alverez D. *Partial Differential Equations - Elliptic Equations.* Imperial College London; 2017.

[5] Uchida Y. *Matrix Algebra - Iterative Solution - Jacobi Method.* Imperial College London; 2017.

Appendix

A Identification number for nature of mesh points

Table 1 shows the numbering system used in the `data` array.

Number	Nature of point
-2	Ambient (irrelevant) point
-1	Fictitious point
0	Microprocessor
1	Ceramic case
2	Heat sink

Table 1: Numbering system used in the `data` array for classes `meshclass` and `multimesh`.