Heat diffusion

v1.0 — October 6th 2022

1 Introduction

As you are aware, microprocessors are prone to heating up. In 2023, the most powerful processors absorb more than 200 Watts and dissipate it as heat. Evacuating this heat is a critical issue: if not managed well, the rise in temperature will destroy the processor. Put simply, if the processor exceeds a certain temperature T_{max} (for example 100°C), it will be destroyed.

The *heat flux density* (i.e. the number of Watts transferred per square meter) of processors is extremely high. A modern high-end CPU can produce 280 W on 10 cm^2, which makes 280 kW/m^2. This is ten times more than a hotplate, and it's about half as much as a fuel rod in a nuclear reactor.

For this reason, dissipators in the form of metal heatsinks are fitted against the processors. They allow for the efficient transfer of heat from the processor to an external environment. In mainstream machines, a fan is used to evacuate the heat of these metal dissipators by forced convection[1]. In computing centers, the heatsink is often cooled by a fluid (*water-cooling*). Finally, in the case of less powerful processors, the phenomenon of natural convection[2] may be sufficient to remove heat from the heatsink.

We assume here that the heatsink is a rectangular parallelepiped (a *cuboid*). Its surface, its thickness and the choice of material it is made of all affect its thermal characteristics.

This project consists in parallelizing a numerical simulation that determines the temperature reached by a recent CPU (an AMD EPYC "rome") on which we place an aluminum plate of 15 cm × 12 cm × 0.8 cm. The $z = 0$ face of the heatsink is forcibly held at $T = 20\,^{\circ}\text{C}$ by some cooling device, and we assume that the entire heatsink is initially at this temperature. The ambient air is also assumed to be at $T = 20\,^{\circ}\text{C}$. The general idea is to turn on the processor at time $t = 0$ and simulate the diffusion of heat in the heatsink until an approximately steady state is reached.

Will the processor overheat? What if we reduce/increase the thickness/area of the What if we reduce/increase the thickness/area of the heatsink? What if we replace the aluminium with copper, iron or... gold?

[1] To increase the contact surface with the air flow and maximize heat transfer, the heat sinks are usually equipped with with fins.

[2] The air in contact with the heatsink absorbs heat, heats up, rises and makes room for fresh air.

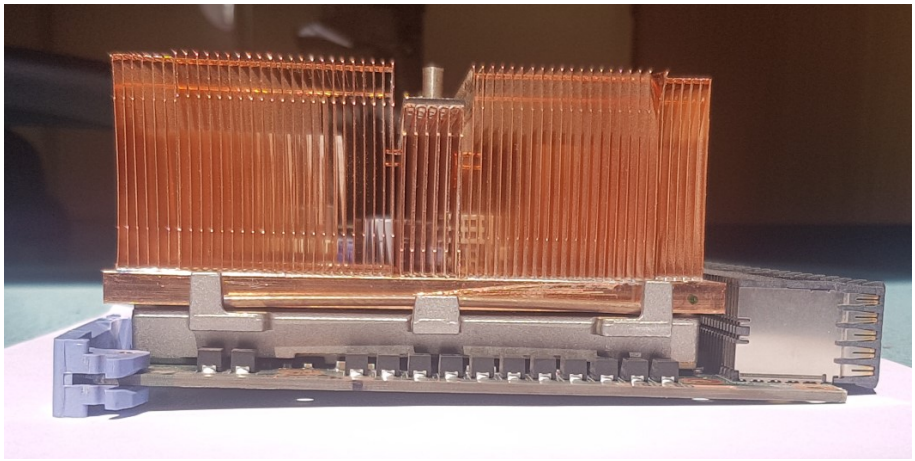

Figure 1: A copper heat sink (with fins) placed on a board containing a CPU.

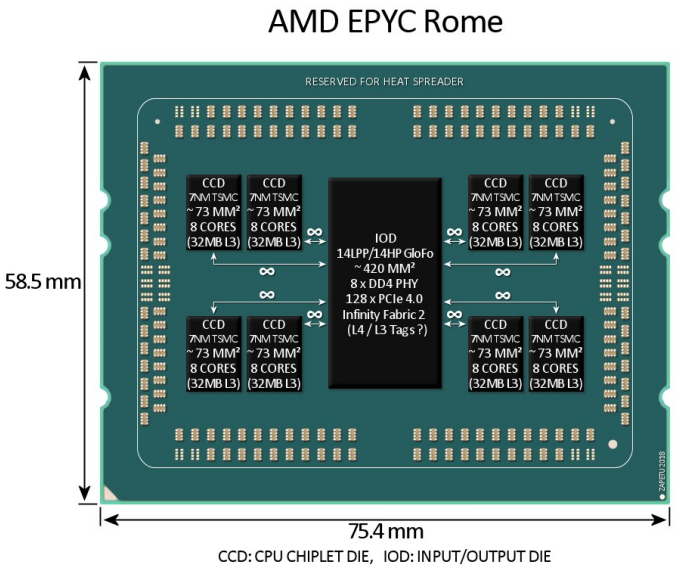

Figure 2: The CPU under the simulated heatsink. Real size. Power dissipation: 280W. Only the black parts are in contact with the heatsink.

2 Physical principle of the numerical simulation

2.1 General comments

Heat is not to be confused with *temperature*. Heat is a transfer of thermal energy. When two bodies are brought into contact, an exchange of thermal energy takes place until both bodies are at the same temperature. Temperature is measured in Celsius degrees (°C) or Kelvin degrees (K, where $0\,\mathrm{K} = -273.1\,^\circ\mathrm{C}$), while quantities of energy (thermal or otherwise) are measured in joules (J).

When heat energy is exchanged, it is convenient to measure the *flow rate*. Heat *flow* is the amount of energy transferred per unit time, and is expressed in Watts (W) — it is the equivalent of some power. For example, if a constant heat flow Φ is transferred for t seconds, the amount of thermal energy transferred is $Q = \Phi t$. In the case of our CPU, the heat flow from the processor to the heatsink is $\Phi = 280\,\mathrm{W}$.

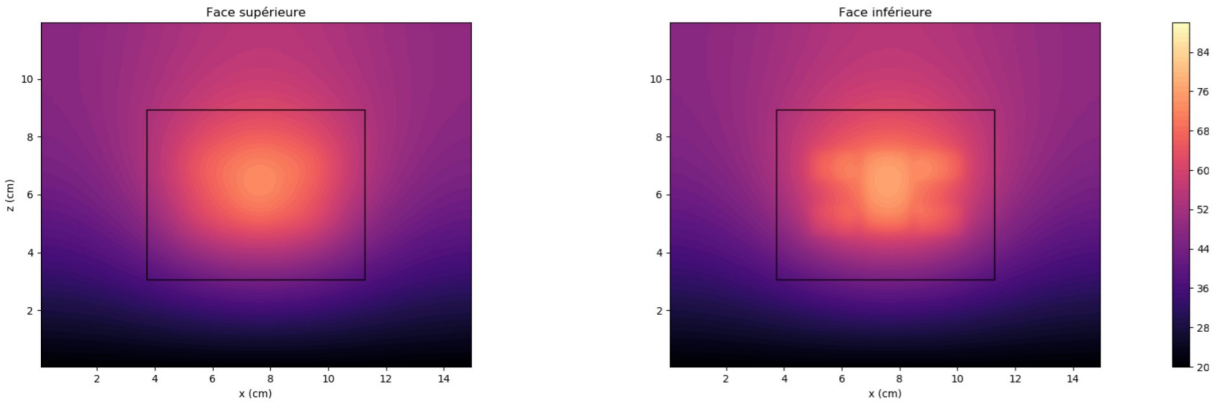

Figure 3: Result of the numerical simulation in stationary regime.

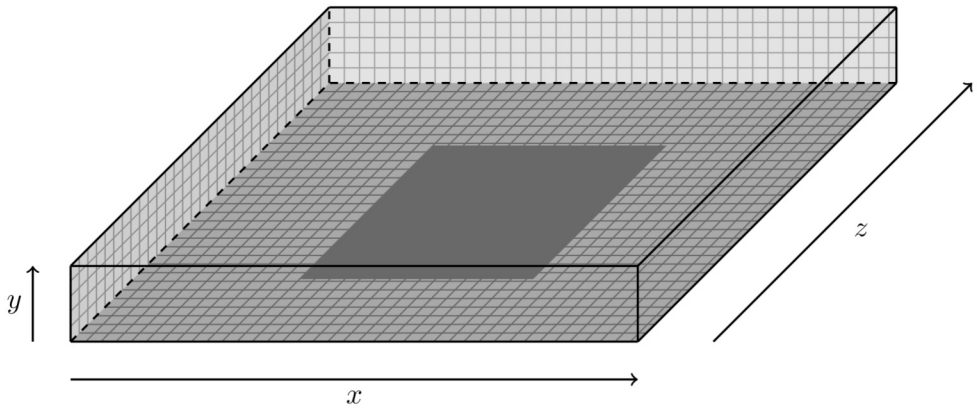

Figure 4: Schematic illustration of the heat sink.

When heat is transferred occurs across a surface, we can also examine how the heat flow rate varies along that surface. The *heat flux density* φ indicates what heat flux passes through a given point of the surface. If it is constant over a surface S, then the total heat flux is $\Phi = \varphi S$. Still in the case of our processor, its contact surface with the heatsink is about $10\,\mathrm{cm}^2$, which gives $\varphi = 280\,\mathrm{kW\,m^{-2}}$.

When a material receives thermal energy, its temperature rises. Its *mass thermal capacity* (or *specific heat capacity*), generally denoted by c, represents the amount of energy (measured in J) required to raise the temperature of 1kg of the material by 1 degree. For example, we have $c = 897\,\mathrm{J\,kg^{-1}\,K^{-1}}$ for aluminium, so if we put a 1Kg block of aluminum on our processor, then we know that it will heat up by $\approx 0.31\,^\circ\mathrm{C}$ per second (that is 280/897). Without external cooling, it will go from $20\,^\circ\mathrm{C}$ to $100\,^\circ\mathrm{C}$ in 256.3 seconds.

2.2 Thermal transfers

It is assumed that there are four different types of heat transfer that take place. In each case, the heat flux (the flow rate of the transfer) is directly proportional to the surface area where the transfer occurs. Therefore we only provide the heat flux density.

1. Thermal *conduction* from the CPU to the heatsink. It occurs solely at points of contact, which all reside on the $y = 0$ plane. We have already calculated that $\varphi = 2.8 \times 10^5\,\mathrm{W\,m^{-2}}$.

2. Heat *conduction* within the heatsink: heat propagates in the metal from the hot parts to the cold parts. This transfer obeys Fourier's law: $\vec{\varphi} = -\lambda \cdot \overrightarrow{\mathrm{grad}}\, T$, where λ (the thermal conductivity) characterizes the capacity of the medium to conduct heat: the greater λ is, the faster the thermal transfer. Conductivity is expressed in Watts per meter per Kelvin and represents the heat flux that circulates between two points when the temperature gradient is 1 degree per meter.

   More concretely, if we have a wall of thickness e whose two sides are of homogeneous temperature T_1 and T_2, then the heat flux density at any point of the wall (from 1 to 2) is $\varphi = \lambda(T_1 - T_2)/e$.

3. *Convection* on the edges of the heatsink: the metal heats the ambient air which enters in movement and evacuates the heat. If a solid wall of surface S and temperature T_1 is in contact with a fluid (air) of temperature T_2, the heat flux density is $\varphi_{1\to 2} = h(T_1 - T_2)$, where h is the *convection heat transfer coefficient* of the material. We observe experimentally that $h \approx 10\,\mathrm{W\,m^{-2}\,K^{-1}}$ for air.

4. *Radiation* emitted on the edges of the heatsink: the metal releases an electromagnetic wave whose frequency depends on the temperature (it can belong to the visible light spectrum: "hot-white" metal). Under reasonable assumptions (black body radiation), the density of thermal flux emitted towards the outside is $\varphi = \sigma T^4$, where σ is the Stefan-Boltzmann constant (its value is $5.6703 \times 10^{-8}\,\mathrm{W\,m^{-2}\,K^{-4}}$; be careful that here the temperature must be in Kelvin).

The proposed numerical simulation claims a certain realism, but it is not perfect: the temperature of the medium is assumed to be fixed, the black body hypothesis is a simplification, and it is assumed that the heat sink does not receive thermal energy from the surrounding medium by radiation, which is impossible.

2.3 Numerical simulation

To simulate all these thermal transfers, we discretize time and space. We slice the heatsink in *cells* (small cuboids) of size $\Delta_x \times \Delta_y \times \Delta_z$, and we suppose that during a sufficiently short time step, the temperature is homogeneous within each cell. The time steps are small enough so that the propagation of thermal energy is relatively slow, and in particular so that it cannot "cross" several cells at the same time (thus the more the metal is a good thermal conductor, the smaller time steps have to be).

We denote by $T(i, j, k)$ the temperature of the cell of coordinates (i, j, k) at time t. The problem consists in computing the temperature of all the cells at time $t + \Delta_t$.

To do this, we must determine the heat flux Φ received by each cell, and then we can calculate the variation of its temperature thanks to the mass heat capacity of the metal and the mass of the cells — fortunately we know the density ρ of the metal and the volume of the cells. We will thus have $\Delta T = \Delta Q/(c\Delta_x\Delta_y\Delta_z\rho)$.

The cells which are on the surface yield energy to the external medium by radiation. For example, on the face $z = 1$, a cell of temperature T transfers to the outside a heat flux of $\Delta_x\Delta_y\sigma T^4$ Watts (indeed, the exchange surface is $\Delta_x\Delta_y$).

The cells that are on the surface also give up energy to the outside environment by convection. For example, on the face $x = 0$, a cell of temperature T transfers $\Delta_y\Delta_z h(T - 293.1)$ Watts.

Each of the cells in contact with the CPU receives a thermal flux of $\Delta_x\Delta_y 2.8 \times 10^5$.

Finally, a cell exchanges thermal energy with each of its neighbors. Cell a receives from cell b (its right-hand neighbor on the x axis, say) a heat flux $\lambda\Delta_y\Delta_z(T_b - T_a)/\Delta_x$.

2.4 Implementation

To simplify matters, we assume that cells are cubic ($\Delta_x = \Delta_y = \Delta_z$). In the program, we denote by `dl` the length of the cell sides.

Representation in memory One (minor) difficulty is that we have to represent the three-dimensional array T in memory. Let us assume that it is of dimension $n \times m \times o$ (according to the axes x, y, z). We flatten it on one dimension using the following convention (*row-major*, usual in the C language): the xy planes are represented contiguously in memory, and within these planes the x lines are represented contiguously. In other words, if the temperature of cell (i, j, k) is in $T[u]$, then:

- $T[u + 1]$ contains the temperature of the cell $(i + 1, j, k)$;
- $T[u + n]$ contains the temperature of the cell $(i, j + 1, k)$;
- $T[u + nm]$ contains the temperature of the cell $(i, j, k + 1)$.

The temperature of the cell (i, j, k) is therefore stored in $T[i+nj+knm]$. The simulation works according to the pseudo-code in figure 5.

3 What you Have to Do

We provide you with a `heatsink.c` sequential `C` program that runs the simulation and dumps the end result to the standard output. We also provide a few python scripts that do visual renderings of the result.

```
1: procedure SIMULATION(...)
2:     Allocate two arrays R and T of size nmo
3:     T[:] ← 293.15                                  ▷ all cells are initially at 20 °C
4:     t ← 0
5:     convergence ← 0
6:     while convergence = 0 do
7:         for 0 ≤ k < o do                           ▷ Processes the kth xy-plane
8:             for 0 ≤ j < m do
9:                 for 0 ≤ i < n do
10:                    R[knm + jn + i] ← UPDATETEMP(i, j, k)
11:                                                   ▷ Accesses to up to 6 neighbor cells
12:                end for
13:            end for
14:        end for
15:        if t is an integer then
16:            T_max = max R[:]
17:            ε = Σ_{u=0}^{omn} (R[u] − T[u])^2
18:            if √ε/Δ_t < 0.1 then
19:                convergence ← 1
20:            end if
21:            Print t and T_max to keep the user waiting...
22:        end if
23:        t ← t + Δ_t
24:        T ← R
25:    end while
26:    Save T in a file for the forthcoming graphical rendering
27: end procedure
```

Figure 5: Principle of the simulation

Your primary goal is to run the simulation in "CHALLENGE" mode, as fast as possible.

There is only one rule: *the use of OpenMP is forbidden.* You may use the external libraries that are usually available in computing centers. We expect that you end up with a parallel program capable of running on more than one compute node.

Some things that you can do:

- Try to use non-blocking communications to overlap communications with computation. Does it bring any improvement? (if so, how much, if not, why?)
- Try to experiment with several kinds of domain decomposition and choose the best one (e.g. 1D, 2D, 3D).
- Try to provide a theoretical model of the performance of your code, in terms of processing and network speed; check this model against reality.
- Try to implement some form of *checkpointing* — the program has to save periodical checkpoints ; if it is interrupted, then it can restart from the last checkpoint.

Whatever you do, you **must** :

1. *Describe* what you have done.
2. *Measure* the performance of your implementation and its scalability (both strong- and weak-scaling are fair game).
3. *Comment* these results: are they good or not? If not, why? Can you design an experiment that would confirm your opinion? Would it have been possible to tell in advance what has happened?

Your work must be submitted by Sunday, November 12th before 23:59 (using Moodle). You must submit your source code, a `Makefile` that compiles your code (without errors nor warnings, even with `-Wall`), and a ≈ 5 pages report in `.PDF` format that explains what you have done.

Please follow our guidelines about writing reports (on Moodle).

You must work in pairs (submit one report with both names)

You **MUST** respect the grid5000 usage policy. Do big computations at night.

If you believe that you have found an error in our programs (it does happen), or if you and your classmates are facing a common technical problem, don't hesitate to contact us.