ATMI : Analytical Model of Temperature in Microprocessors

Pierre Michaud
IRISA/INRIA
Rennes, France
pmichaud@irisa.fr

Yiannakis Sazeides
University of Cyprus
Nicosia, Cyprus
yanos@cs.ucy.ac.cy

Members of HiPEAC

Abstract

As microprocessors become increasingly thermally constrained, microarchitecture and operating-system research will require temperature models that are reliable, fast, and easy to use. This goal is difficult to achieve with general-purpose tools like those based on finite elements. We have developed a computationally efficient temperature model, called ATMI, using analytical methods. ATMI is specialized and is based on some idealizations of the microprocessor chip and its packaging. The computation speed and ease of use of ATMI make it appropriate for research. This paper presents the ATMI model, gives an overview of its software implementation, and provides some example applications.

1. Introduction

The relentless increase of power density has made temperature an important constraint in the design of high performance microprocessors. Packaging has long been concerned with temperature issues, but packaging alone is no longer sufficient. Now temperature issues concern circuit designers, microarchitects, and operating-system developers. Consequently, researchers need reliable models for studying temperature-related problems. We propose ATMI, a model of temperature in microprocessors. Compared to general-purpose tools, ATMI is computationally efficient and easy to use. These advantages were obtained by sacrificing generality and by idealizing the microprocessor chip and its packaging.

Accurate temperature modeling requires to solve a boundary-value problem. First, the physical system is modeled (object geometry, material properties, boundary conditions,...), then the heat equation is solved with mathematical means. There are different methods for solving the heat equation. Ideally, one would like a method that is general, reliable, fast, and easy to use. To our knowledge, no existing method has all these qualities. The most widely used methods for solving the heat equation are finite difference (FDM) and finite element methods (FEM). These methods permit modeling arbitrarily detailed physical systems. However, using FDM or FEM for modeling microprocessor temperature requires a large number of nodes [7]. This means a long computation time, especially when modeling time-varying temperature.

The oldest method for solving the heat equation is the analytical method [9, 4]. Unlike other methods, analytical methods provide explicit solutions, generally in the form of an infinite sum or an integral, which can be evaluated quickly and accurately on modern computers. However, analytical methods are not as general as FDM and FEM since they require more idealization of the physical system. Nonetheless, when applicable, analytical methods are reliable and computationally efficient. We believe that analytical methods are appropriate for researchers that have interest in temperature issues in microprocessors. Our arguments in favor of analytical methods are as follows :

- In microarchitecture and operating-system (OS) research related to thermal issues in microprocessors, the greatest source of inaccuracy does not come from physical idealization, but from not knowing parameters value (e.g., heat sink thermal resistance, interface material thickness and thermal conductivity, ...). This does not necessarily mean that the qualitative conclusions are wrong (if an idea were dependent on precise parameters values, this idea would probably have little applicability). Having a model with few but important parameters makes sense, but it is essential that the model be consistent with physics when we vary parameters values. This goal is compatible with analytical methods.
- For microarchitecture and OS research, computation speed is very important. Analytical methods are generally very fast compared with FEM and FDM.
- Limit studies are useful when trying to gain some understanding (e.g., what if the heat source is very small?

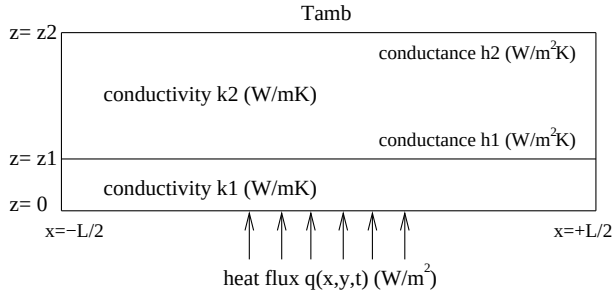

Figure 1: ATMI model : two layers of different materials. Heat transfer between the two layers is modeled by a conductance h_1. Heat transfer from layer 2 to the ambient medium is modeled by a conductance h_2. Heat generation is modeled as a prescribed heat flux on the plane $z = 0$.

what if the heat sink is very large ? ...). Analytical methods do not rely on space discretization and make limit studies very easy.

The rest of the paper is organized as follows. Section 2 presents the ATMI physical model. Section 3 gives an overview of the implementation of the ATMI software. Section 4 provides some applications of ATMI. Finally, Section 5 concludes the paper.

2. The ATMI physical model

Figure 1 depicts the ATMI physical model. It consists of two layers of different material. Each layer is parallel to the "horizontal" (x, y) plane and perpendicular to the z direction. Layer 1 corresponds to $z \in [0, z_1]$ and represents the silicon die, with thermal conductivity k_1. Layer 2 corresponds to $z \in [z_1, z_2]$ and represents the heat spreader and/or the heat sink base plate, with thermal conductivity k_2. The material used for layer 2 (for instance, copper) generally has a high thermal conductivity for good heat spreading. The two layers have the same horizontal dimensions : each layer is a square of side length L, where L is the heat-sink width. Hence one of ATMI limitations is that it does not model chip edges. [1] The power dissipated in transistors and wires is modeled by a prescribed heat flux $q(x, y, t)$ depending on time t and representing the 2D power density in the plane $z = 0$.

Conductance h_1, in W/m^2K, represents the interface material. In the ATMI physical model, the interface is considered infinitely thin. This physical idealization provides a good approximation of the actual behavior [21]. If one knows the interface thickness d_i and thermal conductivity k_i, the corresponding conductance is

1 Neverthless, the ATMI manual [1] describes a rule of thumb, based on the method of images, for estimating the impact of the silicon layer width being less than L.

locus	boundary condition
$z = 0$	$-k_1 \frac{\partial T_1}{\partial z} = q(x, y, t)$
$z = z_1$	$-k_1 \frac{\partial T_1}{\partial z} = -k_2 \frac{\partial T_2}{\partial z}$ $= h_1(T_1 - T_2)$
$z = z_2$	$-k_2 \frac{\partial T_2}{\partial z} = h_2(T_2 - T_{amb})$
$x = \pm L/2$	$\frac{\partial T_1}{\partial x} = \frac{\partial T_2}{\partial x} = 0$
$y = \pm L/2$	$\frac{\partial T_1}{\partial y} = \frac{\partial T_2}{\partial y} = 0$
$t = 0$	$T_1 = T_2 = T_{amb}$

Table 1: ATMI boundary conditions.

$$h_1 = \frac{k_i}{d_i} \tag{1}$$

Conductance h_2 is an effective heat transfer coefficient [10, 23]. For example, for a conventional heat-sink of thermal resistance R_{hs} (K/W) and width L, the equivalent heat transfer coefficient is

$$h_2 = \frac{1}{R_{hs}L^2} \tag{2}$$

In each layer, we assume material characteristics that are uniform and independent of temperature. In reality, silicon characteristics are dependent on temperature. To linearize the problem and make the analytical approach tractable, material characteristics should be set according to the temperature range *a priori*. After linearization, the heat equation can be written

$$\nabla^2 T_i = \frac{1}{\alpha_i} \frac{\partial T_i}{\partial t}$$

for $i \in \{1, 2\}$ and with the boundary conditions listed in Table 1. In particular, temperature T_{amb} of the medium on top of layer 2 is assumed uniform and constant. T_1 and T_2 are temperatures in layers 1 and 2 respectively, and α_1 and α_2 are thermal diffusivities in m^2/s.

We obtained an explicit analytical solution for a point source using classical methods [4]. The detailed derivation of the solution is provided in [19]. The mathematical expression gives temperature on the plane $z = 0$ as a function of the distance from the point source and as a function of time (a power-step is applied to the point source).

Because the ATMI model is linear, the temperature for any power density $q(x, y, t)$ can be obtained from the point-source solution by superposition. More precisely, we define the temperature relative to T_{amb} (or *relative temperature* for short)

$$u = T - T_{amb}$$

The principle of superposition can be stated as follows. Let $u_1(x, y, t)$ be the temperature on the plane $z = 0$ generated

by any power density $q_1(x, y, t)$ and $u_2(x, y, t)$ the temperature generated by any power density $q_2(x, y, t)$. The temperature generated by power density

$$q(x, y, t) = \beta_1 q_1(x, y, t) + \beta_2 q_2(x, y, t)$$

is

$$u(x, y, t) = \beta_1 u_1(x, y, t) + \beta_2 u_2(x, y, t) \qquad (3)$$

where β_1 and β_2 are any fixed factors. The principle of superposition is intuitive and powerful. It is the basis of ATMI.

2.1. Spatial and time convolutions

Essential for a microprocessor temperature model is the measurement of the effects of the different temperature sources on a chip over time. This can be computed using space and time convolutions as described below. We define $H(t)$ as

$$H(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}$$

and we define $u_p(x, y, t)$ as the temperature generated on the plane $z = 0$ when a point source located at $(0, 0, 0)$ dissipates a power $H(t)$ (watts) and assuming $L = \infty$. The temperature $u_H(x, y, t)$ generated by any power density $q(x, y) \times H(t)$ is

$$u_H(x', y', t) = \iint q(x, y) u_p(x' - x, y' - y, t) dx dy \qquad (4)$$

If we consider any function $p(t)$, the temperature $u(x', y', t')$ generated by any power density $q(x, y) \times p(t)$ is

$$u(x', y', t') = \int_0^{t'} p(t) \frac{\partial u_H}{\partial t}(x', y', t' - t) dt \qquad (5)$$

which is known as Duhamel's theorem. Both (4) and (5) stem directly from the principle of superposition (3).

2.2. Method of images

Formula (4) is valid for $L = \infty$. The impact of the finite heat sink width L can be modeled by the method of images [4, 15]. The sides of the base plate are located at $x = \pm L/2$ and $y = \pm L/2$. Heat escaping by convection through the sides of the base plate is negligible compared with that escaping through the top. Hence we can assume insulated walls at $x = \pm L/2$ and $y = \pm L/2$. The effect of insulated walls can be simulated rigorously by adding the temperature contributions of an infinite number of image copies of the power source, as illustrated on Figure 2. The principle is to create a symmetry that forces the heat flux in the planes $x = \pm L/2$ and $y = \pm L/2$ to be parallel to these planes. In practice, because the heat-sink is wider than the chip and the chip is mounted at the center of the heat-sink base plate, it is accurate and computationally efficient to model each image as a point source (temperature a few centimeters away from a source is practically independent from the source geometry and very close to that of a point source).

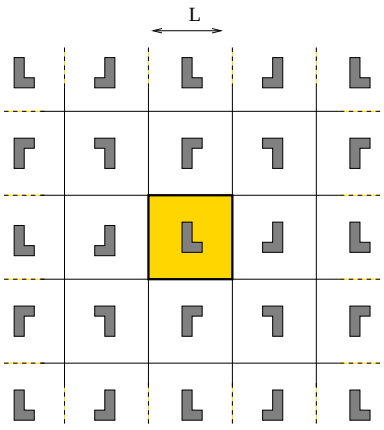

Figure 2: Method of images : an infinite number of image copies of a power source in the plane $z = 0$ simulate insulated walls at $x = \pm L/2$ and $y = \pm L/2$.

3. ATMI software implementation

The model described in Section 2 could be generalized to an arbitrary number of layers and various boundary conditions [4, 13, 6, 17]. Such model could be used for applications other than microprocessor temperature. However, by specializing the model, we could tailor the model to our needs and obtain computation speed and ease of use.

The model of Section 2 is already specialized as it considers a fixed number of layers, fixed boundary conditions (apart from $q(x, y, t)$), and considers temperature only on the plane $z = 0$. Moreover, assuming that power sources are located close to the center of the base-plate, as explained in Section 2.2, facilitated the implementation of the method of images.

3.1. Accelerating time-convolutions

Computing spatial convolution (4) and time convolution (5) may require a lot of time. Thus our model requires additional specialization. Fast Fourier Transform (FFT) could be used to speed up convolutions. The ATMI software provides a FFT-based spatial convolution for steady-state temperature. However, for transient temperature, we used a different method. Instead of considering arbitrarily detailed power density maps, we assumed a microprocessor power density map could be modeled with a moderate number of rectangle power sources.

As for time convolution, it was not possible to use the FFT in the general case. The FFT can speed up convolutions

provided power density as a function of time is known a priori. In particular, power density should not depend on temperature. However, there are two reasons why this is not the case. First, modern microprocessors feature on-chip thermal sensors that trigger thermal throttling when the temperature limit is exceeded, and thermal throttling changes power density. Second, static power consumption is a function of temperature.

Consequently, we used a different method to speed up convolutions. We define a fixed time-step τ, and we assume that power density is constant in time intervals $[n\tau, (n+1)\tau[$ where n is any integer. The time convolution (5) becomes a discrete convolution :

$$u(m\tau) = \sum_{n=0}^{m-1} \int_{n\tau}^{(n+1)\tau} p(t) \frac{\partial u_H}{\partial t}(m\tau - t)dt$$
$$= \sum_{n=0}^{m-1} p(n\tau)[u_H((m-n)\tau) - u_H((m-n-1)\tau)] \quad (6)$$
$$= p(0)u_H(m\tau) + \sum_{n=1}^{m-1} [p(n\tau) - p((n-1)\tau)]u_H((m-n)\tau) \quad (7)$$

where we used the fact that $u_H(0) = 0$. As computing $u_H(t)$ involves a large number of operations, u_H is computed for a few values of t according to a geometric series, and interpolation is used for other values. Moreover, some $u_H(n\tau)$ values are memoized for further speed-up. Yet, computation time is still proportional to m^2.

To solve this problem, we implemented a method that we call *event compression*. To our knowledge, this is an original method. Event compression tries to decrease the number of summands in the discrete convolution. Indeed, in equation (7), if there exists an n such that $p(n\tau) = p((n-1)\tau)$, the corresponding summand is null. The basic principle of event compression is to merge consecutive events. More precisely, we replace consecutive $p(n\tau)$ values with their common average value. Merging events preserves the total energy (when events have different durations because they result from previous merging, we compute a weighted average).

We used two event-compression methods and combined them. The first event-compression method is based on the observation that, when $p(n\tau) \approx p((n-1)\tau)$, merging the two events keeps temperature approximately the same (cf. Equation (6)).

The second event-compression method is based on the observation that if $u_H((m-n)\tau) - u_H((m-n-1)\tau)$ is constant for consecutive time-steps n and $n+1$, merging the events does not change the overall sum in equation (6). Following this observation, we merge the events when $u_H((m-n)\tau) - u_H((m-n-1)\tau)$ is approximately constant for several consecutive time-steps. This happens for large values of $m-n$, as the second derivative of $u_H(t)$ converges to 0. As time increases, old power events are progressively merged thanks to the second compression method. As old events get merged, this creates new opportunities for further compression with the first method.

A physical interpretation of event compression is that, as power events get old, what matters is the quantity of energy that these events represent, not the precise times at which energy was released.

In the ATMI software, the loss of accuracy introduced by event-compression is controlled with a parameter, i.e., we merge events only when the error introduced does not exceed the parameter value. The speed-up provided by event-compression depends on the parameter value : the larger the error tolerance, the faster.

3.2. The ATMI software

The ATMI software is a library written in C. It is publicly available under the GNU General Public Licence [1]. ATMI compiles under Linux and requires the GNU Scientific Library [2]. This section provides an overview of the software. A more detailed description is given in the ATMI manual [1].

The ATMI model features 9 physical parameters (cf. Figure 1) : z_1, $d = z_2 - z_1$, k_1, k_2, α_1, α_2, h_1, h_2 and L. These 9 parameters are gathered in a C structure :

```
struct atmi_param {
  double z1; /* layer 1 thickness (m)                    */
  double d;  /* layer 2 thickness (m)                    */
  double k1; /* layer 1 thermal conductivity (W/mK)      */
  double a1; /* layer 1 thermal diffusivity (m^2/s)      */
  double k2; /* layer 2 thermal conductivity (W/mK)      */
  double a2; /* layer 2 thermal diffusivity (m^2/s)      */
  double h1; /* layer 1/layer 2 (W/m^2K)                 */
  double h2; /* layer 2/ambient (W/m^2K)                 */
  double L;  /* width (m)                                */
  ...
};
```

ATMI works with SI units. In particular, distances are expressed in meters, times in seconds, and temperatures in kelvin or degrees Celsius. The 9 parameters can be set either directly, with the function

```
atmi_set_param(...)
```

or with the function

```
void atmi_fill_param(
    struct atmi_param *p,
    double celsiuszone,              /* (C)     */
    double heatsink_resistance,      /* (K/W)   */
    double heatsink_width,           /* (m)     */
    double copper_thickness,         /* (m)     */
    double bulk_silicon_thickness,   /* (m)     */
    double interface_thickness,      /* (m)     */
    double interface_thermal_cond); /* (W/mK)  */
```

which sets parameters k_1,k_2,α_1 and α_2 corresponding to silicon (layer 1) and copper (layer 2). The value of h_1 is set by providing the interface material thickness and thermal conductivity (cf. equation (1)). The value of h_2 is set by providing the heat-sink width and thermal resistance (cf. equation

(2)). Parameter *celsiuszone* is a temperature used to determine silicon characteristics. For example, if we expect temperature on the chip to be between 40°C and 100°C, we may set *celsiuszone* to 70.

ATMI is built around a set of core functions providing heat-equation solutions. We describe in this document only the two most useful ones.

```
double atmi_rect (
    struct atmi_param *p,
    double q,              /* power density (W/m^2) */
    double a, double b,    /* rect size (m) */
    double x, double y,    /* point coord (m) */
    double t,              /* time (s) */
    char steady);
```

This function gives the relative temperature on the plane $z = 0$ at time t generated by a rectangle source **when** $L = \infty$, i.e, when layers 1 and 2 are infinite in the x and y directions. The source is assumed to dissipate no power for $t < 0$, and a constant and uniform power density q (in W/m^2) for $t \geq 0$. Parameters a and b are respectively the width (x) and height (y) of the rectangle source. Parameters (x, y) are the coordinates of the measure point, taking the rectangle center as the origin. If parameter *steady* is null, the function gives temperature at time t, otherwise it gives the steady-state temperature ($t \to \infty$).

The impact of the heat-sink width is modeled with a separate function :

```
double atmi_images (
    struct atmi_param *p,
    double power,          /* (W) */
    double t,              /* time (s) */
    char steady);
```

This function gives the temperature contribution from the finite value of L as a function of time. This contribution becomes null as $L \to \infty$. Here, *power* is the total power (in watts) dissipated on the chip : it is null for $t < 0$ and constant for $t \geq 0$. This contribution must be added to the temperature values obtained with function *atmi_rect()*.

Here is an example (partial) program :

```
struct atmi_param p;
atmi_set_param(&p,...);
double a1 = 1e-3;  /* side 1st square (m)          */
double a2 = 2e-3;  /* side 2nd square (m)          */
double b = 5e-3;   /* distance between squares (m) */
double q = 1e6;    /* power density (W/m^2)        */
double Tamb = 45;  /* local ambient (C)            */
double t = 0.1;    /* time (s)                     */
double pw = q*(a1*a1+a2*a2); /* power (W)          */

double T1 = Tamb + atmi_rect(&p,q,a1,a1,0,0,t,0)
                 + atmi_rect(&p,q,a2,a2,-b,0,t,0)
                 + atmi_images(&p,pw,t,0);
```

This example considers two square sources whose centers are $b = 5\,mm$ away from each other, and with the same power density ($q = 0$ for $t < 0$, $q = 1\,W/mm^2$ for $t \geq 0$). The temperature of the air hitting the heat sink is $T_{amb} = 45$°C. The example computes the temperature in °C at the center of the first square after $t = 0.1\,s$. It should be noted that this example uses the principle of superposition.

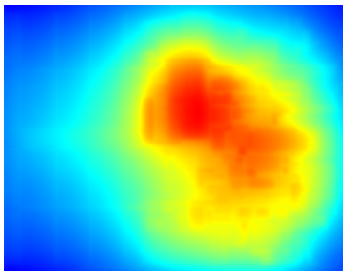

Figure 3: Example of steady-state temperature map obtained with the *atmi_steady_grid()* function.

The ATMI software provides some functions that automate the use of the principle of superposition. The following function takes as input a set of rectangle sources, each source being specified by its coordinates and power density :

```
void atmi_steady_rect(
    struct atmi_param *p,
    int nrect,             /* number of rectangles  */
    struct atmi_rect rc[], /* rectangle coordinates */
    double q[],            /* power density         */
    double temperature[]);
```

It returns in the array *temperature[]* the steady-state relative temperature at the center of each rectangle. Here, *nrect* is the number of rectangles, *rc[]* are the rectangles coordinates and *q[]* are the power densities in each rectangle. Nevertheless, the computation time of *atmi_steady_rect()* increases as the square of the number of rectangles. For very detailed power density maps, we have implemented a FFT-based spatial convolution in the following function :

```
void atmi_steady_grid(
    struct atmi_param *p,
    int nx, int ny,        /* number of squares */
    double gridunit,       /* squares size      */
    atmi_grid q,           /* power density     */
    atmi_grid temperature);
```

The *atmi_grid* type is defined as

```
typedef double atmi_grid [1024][1024];
```

Here, we consider $nx \times ny$ square power sources ($nx \leq 1024$, $ny \leq 1024$), all squares having the same side length *gridunit*. The *atmi_steady_grid()* function takes as input the power density grid q and returns in the *temperature* grid the steady-state relative temperature at the center of each square.

Figure 3 shows an example of steady-state temperature map obtained with *atmi_steady_grid()*. In this example, the grid consists of 180×230 squares, and $atmi\_steady\_grid()$ takes about 2 seconds to execute on a 3 GHz Pentium 4 with the default (conservative) settings of the ATMI 1.0.6 release.

The principle of superposition as stated by equation (3), is very general and applies to any time-varying power density. The following example gives the transient temperature at the center of a square source which is on for $1\,ms$ and off the rest of the time :

```
double a = 0.001; // square side
double q = 1e6; // power density
double duration = 0.001;

double step_response(double t) {
  return (t<=0)? 0 : atmi_rect(&p,q,a,a,0,0,t,0) +
       atmi_images(&p,q*a*a,t,0);
}

double temperature(double t) {
  return step_response(t)-step_response(t-duration);
}
```

However, for more complex cases, manipulating ATMI core functions directly becomes impractical. The ATMI software also provides functions that automate the principle of superposition for time-varying rectangle sources. The following function initializes a *thermal simulator* :

```
void atmi_simulator_init(
    struct atmi_simulator *ts,
    const char *filename,
    struct atmi_param *p,
    double timestep,
    int nrect,
    int nsens,
    struct atmi_rect rc[],
    int sensor[],
    double initq[],
    double wmax);
```

Parameter *timestep* corresponds to the time-step τ in equation (7) and is expressed in seconds. Power density in each rectangle is considered constant between times $n \times timestep$ and $(n+1) \times timestep$, n being any integer. The thermal simulator gives only the temperature at the center of rectangles, and only for rectangles that are declared as *sensors*. The number of sensors *nsens* must not exceed the number of rectangles *nrect*. The list of sensors is declared in the array *sensor[]*. For example, if the power density map consists of $nrect = 4$ rectangles and we want temperature only in the first and last rectangles, we should declare $nsens = 2$ and $sensor[2] = \{0, 3\}$. The thermal simulator initialization phase may take a long time if $nrect \times nsens$ is large. This is the time necessary to compute the temperature responses $u_H(x, y, t)$ for each source and each sensor. These temperature responses depend only on ATMI parameters and rectangles coordinates. If we want to run several simulations with the same ATMI parameters and the same set of rectangles, it is possible to compute the thermal responses once for all and store them in a file whose name is *filename*. The file is automatically created at the first execution of *atmi_simulator_init*. Before starting the simulation, the simulated chip must be put in a meaningful thermal state. This is done by passing initial power densities *initq* corresponding to each rectangle. This initial power density is applied for a time that is long enough to reach a steady state. When simulating a thermally-constrained processor with automatic thermal management, the temperature on the chip cannot exceed a certain value. This is what parameter *wmax* is for. For example, if the ambient temperature is 30°C and the maximum allowed temperature on the chip is 85°C, parameter *wmax* should be set to 55 ($= 85 - 30$) to guarantee a consistent initial thermal state. Once the thermal simulator is initialized, each call to the following function simulates a time-step :

```
void atmi_simulator_step(
    struct atmi_simulator *ts,
    double q[]);
```

where *q[]* is the power density in each rectangle during this particular time-step. After each execution of *atmi_simulator_step()*, the relative temperature at each sensor is stored in the array *temperature[]* of the *atmi_simulator* structure. The event-compression method described in Section (3.1) is used in *atmi_simulator_step()*.

```
#include <stdlib.h>
#include <stdio.h>
#include "atmi.h"

#define XS 0.002
#define XL 0.008
#define Q1 1e6
#define Q2 4e6

struct atmi_rect rc[2] = {
  {-XS/2,-XS/2,XS/2,XS/2},
  {-XL/2,-XL/2,XL/2,XL/2}
};
int sensor[1] = {0};
double initq[2] = {0,Q1};
double q[2] = {0,0};
struct atmi_param p;
struct atmi_simulator ts;

main()
{
  atmi_fill_param(&p,75,0.3,0.07,5e-3,5e-4,5e-5,4);
  atmi_simulator_init(&ts,NULL,&p,2e-4,2,1,rc,sensor,
       initq,1e10);
  while (ts.t <= 200.) {
    q[0] = (ts.t < 5e-3)? 0 : Q2;
    q[1] = (ts.t < 5e-3)? Q1 : 0;
    atmi_simulator_step(&ts,q);
    printf("%8.3e %8.3e\n",ts.t,ts.temperature[0]);
  }
}
```

Figure 4: Example of program using the ATMI thermal simulator. The data produced by this program is plotted in Figure 5.

Figure 4 shows a full program using the ATMI thermal simulator. This programs simulates 2 square sources, a large one ($8\,mm$) and a small one (2 mm). The small square is at the center of the large square. From $t = -\infty$ to $t = 5\,ms$, power density in the large square is $1\,W/mm^2$, while the small square is turned off. At $t = 5\,ms$, power density in the small square is set to $4\,W/mm^2$ while the large square is turned off. Figure 5 shows the relative temperature as a function of time.

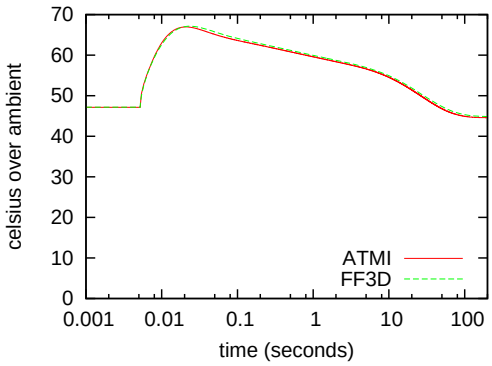

Figure 5: Temperature as a function of time obtained with the C program of Figure 4. The second curve, almost indistinguishable, was obtained by solving the heat equation with the finite-element solver FreeFEM3D [3].

3.3. Validation and limitations

In order to check the correctness of the software implementation, we have reproduced with ATMI some temperature numbers published in [8, 23, 11]. Moreover, we did some comparisons with the finite-element solver FreeFEM3D [3] (for instance, Figure 5 shows the curve obtained by solving the heat equation with FreeFEM3D, which is almost indistinguishable from that obtained with ATMI). These multiple verifications give us confidence that ATMI is correctly implemented **under** the physical approximations made.

Some of the physical approximations were mentioned explicitly in Section 2, in particular chip edges that are not modeled. There are some other approximations, among which modeling the heat-sink as a cuboid, neglecting the heat transfer through the chip pins, and not modeling the material heterogeneity in transistors and connections layers.

The qualitative behaviors obtained with ATMI are consistent with Fourier's law of heat conduction. However, the temperature numbers given by ATMI are that of an idealized physical system whose parameters may need proper adjustment in order to imitate a real system and obtain quantitative accuracy. For instance, a straightforward setting of parameters T_{amb} and h_2 allowed us to reproduce with ATMI some of the temperature numbers published in [23] for a heat-sink with a vapor chamber.

4. Applications of ATMI

This section provides three example applications using ATMI. The first one is a simple example showing that neither power nor local power density can be used as a substitute for temperature. The second example shows the impact of static power being dependent on temperature. The third example studies the placement of thermal sensors, which is an important issue for thermally-constrained microprocessors. This problem has already been studied (e.g., [14, 20]), but we provide an original alternative approach. Another ATMI application for OS research can be found in [18].

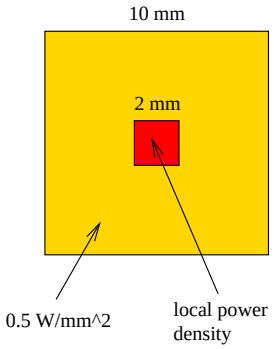

Figure 6: Example of Section 4.1.

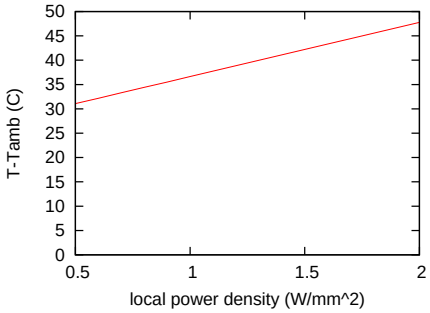

Figure 7: Cf. Figure 6. Steady-state relative temperature at the square center as a function of local power density. ATMI parameters are set as in Figure 4.

4.1. Temperature is neither power nor power density

Temperature at a given point depends both on the total power dissipated by the chip and on the power density around the point. Figure 6 illustrates this situation. The chip is modeled as a $1\,cm \times 1\,cm$ square. A $2\,mm \times 2\,mm$ square representing a functional unit is located at the center of the chip. Power density on the chip is $0.5\,W/mm^2$ except in the small square. We set the ATMI parameters as in Figure 4 and we plot on Figure 7 the steady-state relative temperature at the square center. When the local power density in the small square goes from $0.5\,W/mm^2$ to $2\,W/mm^2$, the total power on the chip goes from $50\,W$ to $56\,W$, i.e., a 12% increase. Yet, the relative temperature increase is about 50%. This example shows that neither power nor local power density alone can be used as a substitute for temperature. Temperature is a function of the whole power density distribution.

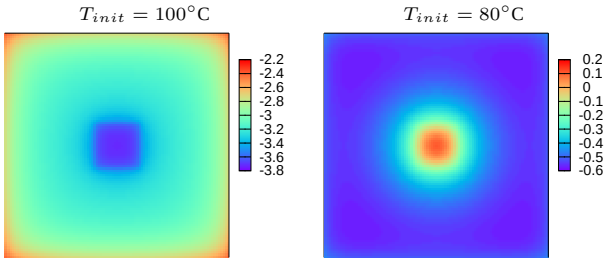

Figure 8: Same configuration as in Figure 6 but power density depends on temperature. Function *atmi_steady_grid()* is used iteratively. The picture shows the difference between the final temperature map and the first temperature map computed with power density corresponding to T_{init} (100°C and 80°C).

4.2. Power density depending on temperature

The subthreshold leakage current of MOSFETs depends on temperature. If an accurate static power consumption model is available, one can obtain a more accurate steady-state temperature by iterating *atmi_steady_grid()* until temperature converges.

To illustrate this, we have defined a 100×100 grid on the same example as in Section 4.1 but with power density computed as follows. We assume that, at 100°C, power density is 50% static and 50% dynamic, and that the total power density is $2\,W/mm^2$ in the small square and $0.5\,W/mm^2$ elsewhere. To model the dependence of static power on temperature, we used the model given in [16], which is based on BSIM4 gate leakage and subthreshold leakage model. We used the parameters given in [16] for logic circuits in $65\,nm$ technology. We assume $V_{dd} = 1\,V$ and $T_{amb} = 45$°C.

First we compute the power density map based on a fixed initial temperature T_{init}. We compute the temperature map corresponding to this power density. Then we recompute the power density map with the detailed temperature map, and we compute a new temperature map, and so on. We iterate until the maximum difference between consecutive temperature maps does not exceed 0.001 °C. In practice, temperature converges very quickly.

Figure 8 shows the difference between the final temperature map and the first temperature map computed with power density corresponding to T_{init}. On this example, the actual (final) temperature on the chip is between 65°C and 90°C. When static power is computed with $T_{init} = 100$°C, the initial temperature map overestimates the actual temperature : the actual temperature is 2.2°C to 3.8°C lower than indicated by the first temperature map. As can be seen in Figure 8, taking $T_{init} = 80$°C gives a first temperature map which is within 1°C of the actual temperature.

This example suggests that a rough guess of the final average temperature may be sufficient to obtain a reasonably accurate temperature map.

4.3. Thermal sensors placement

When designing a thermally-constrained microprocessor, one must decide where to put temperature sensors [14, 20].

The power density map generated when an application executes on a processor depends on the application characteristics. A systematic method for placing thermal sensors requires to collect a set of power density maps by simulating a large number of real-life applications. This set may be extended with some power density maps corresponding to worst-case scenarios. Ideally, we would like to put a sensor at every location that may be a hot spot. However, in practice, we want to minimize the number of sensors. For example, if a processor consists of N functional units that may be independently active or inactive, there exists 2^N distinct configurations, hence 2^N distinct power density maps. If we place a sensor at the hottest location for each configuration, we may need up to 2^N sensors.

To obtain a reasonable number of sensors, we should work with a temperature margin. We want temperature on the chip to not exceed a fixed value T_{max}, but we assume that sensors trigger thermal throttling when temperature reaches a fixed $T_{lim} < T_{max}$. To simplify the discussion, we assume that power density does not depend on temperature. Moreover, we assume that the ambient temperature is bounded :

$$T_a \leq T_{amb} \leq T_A$$

For each configuration, we compute the maximum relative temperature $max\{u(x,y)\}$, where $u(x,y)$ is the relative temperature when there is no thermal limitation. If $T_A + max\{u(x,y)\} \leq T_{max}$, the configuration is not thermally critical, and we ignore it.

Otherwise, it means that thermal throttling may be triggered on this configuration. We assume that all functional units are throttled with the same duty cycle $\lambda \in [0,1]$, so we can apply the principle of superposition. Under thermal throttling, the temperature is

$$T(x,y) = T_{amb} + \lambda u(x,y)$$

Let us assume that we place the thermal sensor for that configuration at point (x_s, y_s). Under thermal throttling, we have $T(x_s, y_s) \approx T_{lim}$, that is,

$$\lambda = \frac{T_{lim} - T_{amb}}{u(x_s, y_s)}$$

Then,

$$T(x,y) = T_{amb} + \frac{u(x,y)}{u(x_s,y_s)}(T_{lim} - T_{amb})$$

A point (x_s, y_s) is a *potential* sensor location if $max\{T(x,y)\} \leq T_{max}$, that is,

$$u(x_s, y_s) \geq max\{u(x,y)\}\frac{T_{lim} - T_{amb}}{T_{max} - T_{amb}} \quad (8)$$

The set of potential sensor locations gets smaller with smaller values of T_{amb}. Hence we consider the sensor locations corresponding to the lowest value of T_{amb} that requires thermal throttling, which is

$$T_{amb} = max(T_a, T_{max} - max\{u(x,y)\}) \quad (9)$$

Inequation (8), with T_{amb} given by (9), defines a set of *acceptable* sensor locations for that configuration. We compute the set of acceptable sensor locations for each configuration (after eliminating configurations that cannot trigger thermal throttling). Then we associate with each point (x, y) the set $S(x, y)$ of configurations for which (x, y) is an acceptable sensor location. For instance, if configurations are numbered from 1 to M, we have $S(x, y) \subseteq [1, M]$. We try to find a minimum set of points $\{(x_i, y_i)\}$ such that

$$\bigcup_i S(x_i, y_i) = [1, M] \quad (10)$$

Points $\{(x_i, y_i)\}$ are the points where sensors should be placed. The problem of finding a minimum set of points that verifies (10) is an instance of the set covering problem [5], which is NP-hard.

We implemented this method on an artificial example whose purpose is just to show that the method is feasible. [2] We model the processor as a $12\,mm \times 12\,mm$ square consisting of 16 square units, as depicted in Figure 9. We assume that each unit may be active independently, and that the power density in an active unit is $1\,W/mm^2$. Hence there are 2^{16} distinct configurations. To compute the temperature maps, we used the *atmi_steady_grid()* function. The grid size is 60×60. Though a single temperature map can be computed quickly, executing the *atmi_steady_grid()* function 2^{16} times requires hours of computation time. Actually, the 2^{16} temperature maps can be computed much faster than that, by using the principle of superposition. Each of the 2^{16} power density maps is a linear combination of 16 base power density maps, where each base power density map corresponds to a single active unit, all the other units being inactive. We execute *atmi_steady_grid()* on the 16 base power density maps and obtain 16 base temperature maps. From the principle of superposition, the 2^{16} temperature maps are obtained by a linear combination of the 16 base temperature maps. On a 3 GHz Pentium 4, we obtain the 2^{16} temperature maps in less than 1 minute.

2 In practice, the number of distinct power density maps should be smaller than in this example, as functional units have correlated activity rates when the processor executes real-life applications.

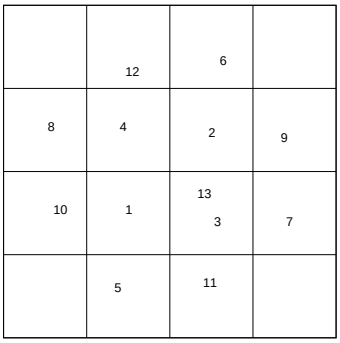

Figure 9: Example of thermal sensors placement using a greedy algorithm. Here, $T_{max} = 85°$C and $T_{lim} = 83°$C.

For this example, we used $T_{max} = 85°$C, $T_a = 5°$C, $T_A = 50°$C, heat-sink thermal resistance $R_{hs} = 0.3\,K/W$ and width $L = 0.07\,m$, copper thickness $d = z_2 - z_1 = 5\,mm$, silicon thickness $z_1 = 0.5\,mm$, interface material thickness $d_i = 50\,\mu m$ and thermal conductivity $k_i = 4\,W/mK$.

We obtained sensor locations $\{(x_i, y_i)\}$ with a greedy heuristic. First, we choose a point (x_1, y_1) that covers the greatest number of configurations, i.e., such that $S(x_1, y_1)$ is largest. We place the first sensor at (x_1, y_1). Then we place the second sensor at the point (x_2, y_2) that covers the greatest number of configurations not already covered by sensor 1. Then we place the third sensor at the point (x_3, y_3) that covers the greatest number of configurations not already covered by sensors 1 or 2. And so on, until all configurations (minus those that do not trigger thermal throttling) are covered. The rank of a sensor reflects its importance : the first sensor covers a large number of configurations, while the last sensor covers a small number of configurations.

We ran the algorithm with $T_{lim} = 84°$C, i.e., $1°$C below T_{max}. The greedy algorithm found that 29 sensors were necessary, some of these sensors being very close to each other, which means that the number of sensors could be decreased by a slight increase of the temperature margin. We increased the temperature margin by setting $T_{lim} = 83°$C. The result is given on Figure 9, with sensors represented by numbers indicating their rank. This time, 13 sensors are sufficient. It can be noticed that the distribution of sensors is approximately regular, except for sensor #13 (in fact, sensor #13 covers a single configuration. As it is close to sensor #3, one may want to remove it by further enlarging the temperature margin). The cost of having only 13 sensors is a slight performance loss under thermal throttling because the duty cycle is $\frac{T_{lim} - T_{amb}}{u(x_s, y_s)}$ instead of $\frac{T_{max} - T_{amb}}{u(x_s, y_s)}$. The worst-case performance loss is $\frac{T_{max} - T_{lim}}{T_{max} - T_A} \approx 6\%$.

5. Conclusion

ATMI is not the only chip-temperature model available. Some other specialized models have been released, e.g., [22, 12, 24]. However, having several models available gives more choice to the user and permits cross-validating the different models. For example, if a given temperature model generates counter-intuitive results and the user is not sure to use the model correctly, having a second model is useful. The cross-validation is stronger if there is some diversity in the methods used by the different models.

What distinguishes ATMI from other models is that ATMI relies on explicit analytical solutions to the heat equation. In particular, ATMI does not rely on any discretization of the temperature field.

Another distinguishing feature of ATMI is that it is based on the principle of superposition and the method of images. Though using the ATMI software does not necessitate understanding them, these principles, especially the principle of superposition, provide intuition about temperature. Intuition about temperature can also be obtained from simple examples, as illustrated in this paper. Indeed, simple examples often exhibit qualitative behaviors whose generality can be tested with further examples. Yet, these qualitative behaviors can be best understood if one understands the principle of superposition.

Nevertheless, ATMI is compatible with the usual quantitative method in computer architecture studies and can easily be integrated in a performance/power microarchitecture simulator like SimpleScalar/Wattch, to simulate the impact of thermal throttling on performance.

References

[1] ATMI. http://www.irisa.fr/caps/projects/ATMI.

[2] GNU Scientific Library. http://www.gnu.org/software/gsl/.

[3] FreeFEM3D. http://www.freefem.org/ff3d/.

[4] H. Carslaw and J. Jaeger. *Conduction of heat in solids*. Oxford University Press, 1959.

[5] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[6] J.-M. Dorkel, P. Tounsi, and P. Leturcq. Three-dimensional thermal modeling based on the two-port network theory for hybrid or monolithic integrated power circuits. *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, 19(4):501–507, Dec. 1996.

[7] D. Fetis and P. Michaud. An evaluation of HotSpot-3.0 block-based temperature model. In *Proceedings of the Fifth Annual Workshop on Duplicating, Deconstructing, and Debunking*, 2006.

[8] T. Fisher, C. Avedisian, and J. Krusius. Transient thermal response due to periodic heating on a convectively cooled substrate. *IEEE Transactions on Components, Packaging, and Manufacturing Technology - Part B*, 19(1), Feb. 1996.

[9] J. Fourier. *The analytical theory of heat*. Dover Publications.

[10] V. Gektin, R. Zhang, M. Vogel, G. Xu, and M. Lee. Substantiation of numerical analysis methodology for CPU package with non-uniform heat dissipation and heat sink with simplified fin modeling. In *Proceedings of the 9th Intersociety Thermal Phenomena (ITherm) Conference*, 2004.

[11] T. Goh, K. Seetharamu, G. Quadir, Z. Zainal, and K. Ganeshamoorthy. Thermal investigations of microelectronic chip with non-uniform power distribution : temperature prediction and thermal placement design optimization. *Microelectronics International*, 21(3):29–43, Dec. 2004.

[12] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. Mercury and Freon : temperature emulation and management in server systems. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[13] A. Kokkas. Thermal analysis of multiple-layer structures. *IEEE Transactions on Electron Devices*, ED-21(11):674–681, November 1974.

[14] K.-J. Lee, K. Skadron, and W. Huang. Analytical model for sensor placement on microprocessors. In *Proceedings of the International Conference on Computer Design*, 2005.

[15] P. Leturcq, J.-M. Dorkel, A. Napieralski, and E. Lachiver. A new approach to thermal analysis of power devices. *IEEE Transactions on Electron Devices*, ED-34(5):1147–1156, May 1987.

[16] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1042–1053, July 2005.

[17] D. Maillet, S. André, J. Batsale, A. Degiovanni, and C. Moyne. *Thermal quadrupoles - Solving the heat equation through integral transforms*. Wiley, 2000.

[18] P. Michaud and Y. Sazeides. Scheduling issues on thermally constrained processors. Technical Report PI-1822, IRISA, 2006. Also published as INRIA report RR-6006.

[19] P. Michaud, Y. Sazeides, A. Seznec, T. Constantinou, and D. Fetis. An analytical model of temperature in microprocessors. Research report RR-5744, INRIA, Nov. 2005.

[20] R. Mukherjee and S. Memik. Systematic temperature sensor allocation and placement for microprocessors. In *Proceedings of the 43rd Annual Conference on Design Automation*, 2006.

[21] J. Parry, H. Pape, D. Schweitzer, and J. Janssen. Transient performance of common modeling assumptions used in detailed thermal package models. In *Proceedings of the 8th THERMINIC Workshop*, 2002.

[22] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003.

[23] G. Xu, B. Guenin, and M. Vogel. Extension of air cooling for high power processors. In *Proceedings of the 9th Intersociety Thermal Phenomena (ITherm) Conference*, 2004.

[24] Y. Yang, Z. Gu, C. Zhu, R. Dick, and L. Shang. ISAC : Integrated space-and-time-adaptive chip-package thermal analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(1):86–99, Jan. 2007.