

ATMI 2.0 manual

Pierre Michaud
pmichaud@irisa.fr

August 17, 2009

- ATMI is an **analytical** model of **temperature** in **microprocessors**, implemented as a set of C functions.
- ATMI is not a power model. ATMI takes power numbers as input, and returns temperature numbers. The user must provide power numbers.
- ATMI does not model a particular packaging and heat-sink. These are ATMI inputs defined by the user.

1 ATMI model

The processor and its packaging are idealized as depicted in Figure 1. ATMI models two layers of different materials. Typically, layer 1 is silicon and layer 2 is metal. Both layers have horizontal dimensions $L \times L$, with L the width of the metal layer (or the square root of its area). That is, ATMI assumes a large silicon chip, which neglects the impact of chip edges. The interface material between the two layers is considered infinitely thin and is modeled by a conductance h_1 in W/m^2K . If the actual thickness of the interface is d_i and its thermal conductivity k_i , the equivalent conductance is

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

The ambient medium is assumed to have a fixed and uniform temperature T_{amb} . Planes $z = 0$, $x = \pm L/2$ and $y = \pm L/2$ are assumed thermally insulated. Heat can only escape through the top plane $z = z_2$, where

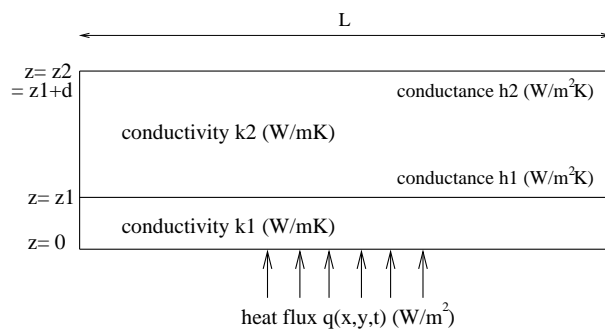


Figure 1: ATMI model

parameter	unit (SI)	meaning
z_1	m	layer 1 thickness
$d = z_2 - z_1$	m	layer 2 thickness
k_1	W/mK	Layer 1 thermal conductivity
k_2	W/mK	Layer 2 thermal conductivity
α_1	m^2/s	layer 1 thermal diffusivity
α_2	m^2/s	layer 2 thermal diffusivity
h_1	W/m^2K	conductance between layers 1 and 2
h_2	W/m^2K	conductance between layer2 and ambient
L	m	width

Table 1: *The 9 ATMI parameters*

the conductance is h_2 in W/m^2K . For example, to simulate a heat-sink of thermal resistance R in K/W and width L , we define the equivalent conductance

$$h_2 = \frac{1}{R \times L^2}$$

The heat generation by transistors and wires is simulated by a 2-dimensional power density $q(x, y, t)$ in the plane $z = 0$. That is, we assume that heat is generated in an infinitely thin layer. For more information on ATMI theory, see references [9, 8]. **All temperature numbers output by ATMI are on the plane $z = 0$ and are relative to the ambient T_{amb} .** The value of T_{amb} is not part of the ATMI model. Hence to get absolute temperatures, the user should add her/his own T_{amb} value to all temperature numbers output by ATMI functions. T_{amb} is a local ambient, defined as the temperature of the medium on top of layer 2. It is assumed uniform and constant. Hence if layer 2 represents the base-plate of a heat-sink cooled by convection of air, T_{amb} is the temperature inside the computer case, at the CPU fan inlet. It is typically several degrees Celsius higher than the room temperature [7].

2 ATMI parameters

The 9 ATMI parameters are listed in Table 1. They are defined in a C structure :

```
typedef struct {
    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 thermal conductance (W/m^2K) */
    double h2; /* layer 2/ambient thermal conductance (W/m^2K) */
    double L; /* width (m) */
    double MAXR; /* not an ATMI parameter, for internal use */
} atmi_param;
```

ATMI works with SI units. In particular, distances are in meters and temperatures in kelvins.

The user must set the 9 parameter values corresponding to her/his problem. For instance, modeling a server processor will generally require different parameter values than modeling a laptop processor.

There are two ways to set ATMI parameters values. The first function sets parameters values directly :

```
void atmi_set_param ( atmi_param *p ,
                    double val_z1 ,
                    double val_d ,
                    double val_k1 ,
                    double val_a1 ,
                    double val_k2 ,
                    double val_a2 ,
                    double val_h1 ,
                    double val_h2 ,
                    double val_L );
```

The second function sets parameters corresponding to silicon (layer 1) and copper (layer 2) :

```
void atmi_fill_param ( atmi_param *p ,
                    double celsiuszone , /* celsius */
                    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 */ );
```

Silicon thermal characteristics depend on temperature. However, ATMI is a linear model (like many temperature models) and neglects this effect. A solution that is often used is to linearize the problem in a temperature range, estimated *a priori*. Parameter *celsiuszone* should be set approximately to the temperature of the point where we want to be accurate. If only a temperature range is known, *celsiuszone* may be set to the middle of this range. For example, if temperature on the chip is expected to be between 50°C and 100°C, *celsiuszone* may be set to 75.

With function *atmi_fill_param*, parameters h_1 and h_2 are set indirectly by providing the interface material thickness and thermal conductivity, and the heat-sink thermal resistance in K/W (more precisely, the sink-to-ambient thermal resistance).

If the second layer is not copper but another material, it is possible to use *atmi_fill_param* to set silicon characteristics, and then to correct layer 2 parameters with *atmi_set_param*. For example :

```
atmi_param p;
atmi_fill_param (&p , 85 , 0.3 , 0.07 , 0.005 , 0.0005 , 0.0001 , 4);
atmi_set_param (&p , p.z1 , p.d , p.k1 , p.a1 , 200 , 0.0001 , p.h1 , p.h2 , p.L );
```

Remark. ATMI does not model a particular packaging. There is not a single interface material, or a single heat sink. It depends on the type of system (embedded, laptop, desktop, server, etc.) and the associated economic and technological constraints. Note that the sink-to-ambient thermal resistance depends on fan speed. Moreover, the interface material characteristics degrade with usage [10].

3 ATMI core functions

We recall that all temperature numbers output by ATMI are on the plane $z = 0$ and are relative to the ambient. The ATMI model is based on a set of core functions that give transient temperature responses to a step power.

```
double atmi_rectangle ( atmi_param *p ,
                        double q, /* power density (W/m^2) */
                        double a, /* rectangle width (m) */
                        double b, /* rectangle height (m) */
                        double x,
                        double y,
                        double t, /* time (s), used if steady=0 */
                        char steady );
```

This functions gives the relative temperature generated at time t 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 $t \geq 0$. The rectangles sides are parallel to the x and y axes, and 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 relative to the rectangle center (the rectangle center has coordinates $(0, 0)$, the rectangle vertices have coordinates $(\pm a/2, \pm b/2)$). If parameter *steady* is null, the function gives temperature at time t , otherwise it gives the steady-state temperature ($t \rightarrow \infty$).

```
double atmi_images ( atmi_param *p ,
                    double power, /* power (W) */
                    double t, /* time (s), used if steady=0 */
                    char steady );
```

This function gives the temperature contribution from the finite value of L (this contribution becomes null as $L \rightarrow \infty$). It is assumed that most power sources are located close the center of the $L \times L$ square, which is generally the case as the heat-sink is wider than the chip and the chip is mounted at the center of the base-plate. The value returned by *atmi_images* must be added to values returned by *atmi_rect*.

For example, the steady-state temperature at a vertex of a $1\text{ mm} \times 1\text{ mm}$ square source dissipating a power density 3 W/mm^2 can be obtained as follows :

```
double a = 0.001; // source length
double q = 3e6; // power density
double temperature = atmi_rectangle(&p,q,a,a,a/2,a/2,0,1)
                    + atmi_images(&p,q*a*a,0,1);
```

4 Principle of superposition

4.1 Steady state

The relative temperature generated by a set of power source can be obtained by summing the relative temperature generated by each source considered separately. For example, the steady-state temperature at the center of a square source in the presence of a second identical source can be obtained as follows :

```

double a = 0.001; // sources length
double b = 0.005; // distance between sources centers
double q = 3e6; // power density
double temperature = atmi_rectangle(&p,q,a,a,0,0,0,1)
                    + atmi_rectangle(&p,q,a,a,-b,0,0,1)
                    + atmi_images(&p,2*q*a*a,0,1);

```

The ATMI software provides some functions that automatize the principle of superposition. The following function gives the steady-state temperature for a power density map consisting of a set of rectangle sources :

```

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

```

More precisely, this functions returns in the array *temperature* the steady-state temperature at the center of each rectangle. Each rectangle geometry is described with the following structure :

```

typedef struct {
    double x1;
    double y1;
    double x2;
    double y2;
} atmi_rect;

```

where (x_1, y_1) and (x_2, y_2) are the coordinates of opposite vertices of the rectangle, i.e., those defining a diagonal. The previous example could be treated as follows :

```

double a = 0.001; // sources length
double b = 0.005; // distance between sources centers
double q[2] = {3e6,3e6}; // power density of each source
atmi_rect rc[2] = {
    {-a/2,-a/2,a/2,a/2},
    {b-a/2,-a/2,b+a/2,a/2}
};
double temp[2];
atmi_steady_rect(&p,2,rc,q,temp);
double temperature = temp[0];

```

However, the computation time of function *atmi_steady_rect* increases quickly with the number of rectangles. If the power density map is very detailed, then it is better to use the following function :

```

void atmi_steady_grid(atmi_param *p,
                    int nx, /* number of blocks in the x direction */
                    int ny, /* number of blocks in the y direction */
                    double gridunit, /* block length (meters) */
                    atmi_grid q, /* power density */
                    atmi_grid temperature);

```

where a grid is defined as a 2-dimensional array :

```
#define ATML_GRIDMAX 1024
typedef double atmi_grid [ATML_GRIDMAX][ATML_GRIDMAX];
```

In function *atmi_steady_grid*, the power density map consists of a grid of $nx \times ny$ square blocks whose side length is *gridunit*, with *nx* and *ny* less than ATML_GRIDMAX. The function returns in the *temperature* grid the steady-state temperature at the center of each block.

4.2 Transient

The principle of superposition does not apply solely to steady-state temperature, it also applies to transient temperature. For example, the transient temperature at the center of a square source which is on for 1 *ms* and off the rest of the time can be obtained as follows :

```
double a = 0.001; // source length
double q = 3e6; // power density
double duration = 0.001; // power is on for 1 ms

double step_response(double t)
{
    if (t <= 0) {
        return 0;
    } else {
        return atmi_rectangle(&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);
}

double t;

for (t=0; t<=0.01; t+=1e-4) {
    printf("%f \t%f\n",t,temperature(t));
}
```

The ATMI software provides functions that automatize the generalized principle of superposition. These functions can be used to simulate the temperature in a microprocessor whose power density map can be described by a set of rectangle sources. The user provides a set of *nrect* rectangles numbered from 0 to *nrect* - 1. Some of these rectangles can be defined as *sensors*, meaning that the thermal simulator will compute the temperature at the center of each of these rectangles. For example, to define a sensor in rectangles #0 and #4, we do as follows :

```
atmi_rectset sensors;
atmi_rectset_init(&sensors);
```

```

atmi_rectset_add(&sensors ,0); // sensor 0 in rectangle 0
atmi_rectset_add(&sensors ,4); // sensor 1 in rectangle 4

```

The following function initializes the thermal simulator :

```

void atmi_simulator_init(atmi_simulator *ts ,
                        const char *filename ,
                        atmi_param *p ,
                        double timestep ,
                        int nrect , /* number of rectangle sources */
                        atmi_rect rc [] , /* rectangles coordinates */
                        atmi_rectset *sensors ,
                        double q []); /* warm-up power density */

```

Parameter *timestep* is the time-step, which means that power density in each rectangle is considered constant between times $n \times timestep$ and $(n + 1) \times timestep$, n being integer. The thermal simulator will give temperature only in the rectangles declared as sensors via the parameter *sensors*.

The thermal simulator initialization phase may take a long time if there are many rectangles and many sensors. This is the time necessary to compute the temperature responses for each (source,sensor) pair. These temperature responses depend only on the ATMI parameters. So if we want to run several simulations with the same ATMI parameters, it is possible to compute the thermal responses once 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 applying warm-up power densities *q[]* (one value per rectangle). This initial power density is applied for a time that is long enough to reach a steady state. The warm-up power density must be set carefully to obtain meaningful simulation results. For instance, if warm-up power densities are null, it means that the initial temperature is the ambient temperature. But it may not be what we want to simulate. Often, we want the initial thermal state to be close to a steady state, because the simulation will not be long enough to reach a steady state. Moreover, modern processors feature thermal throttling mechanisms that prevent temperature to exceed a certain limit. The corresponding warm-up power density can be computed with the following function which computes approximately the steady-state effect of thermal throttling :

```

void atmi_steady_throttle(atmi_param *param ,
                        int nrect ,
                        atmi_rect rc [] ,
                        atmi_rectset *sensors ,
                        atmi_rectset domain [] ,
                        double wmax ,
                        double q []);

```

Here, *param*, *nrect*, *rc[]* and *sensors* are the same parameters that are passed as input to *atmi_simulator_init*. Each sensor is associated with a *domain*. A domain is a set of rectangles that are throttled whenever the relative temperature at the sensor exceeds *wmax*. The domain *domain[i]* for the i^{th} sensor is set with *atmi_rectset_add*. Function *atmi_steady_throttle* takes as input the power density *q[]* corresponding to when throttling is not engaged, and it returns in *q[]* the power density that should be used as warm-up power density.

For example, let us assume that we model a dual-core processor with 4 rectangles, where rectangles #0 and #1 model the first core and rectangles #2 and #3 model the second core. Let us assume that the ambient

temperature is 35°C and that each core has an independent throttling mechanism that prevents temperature in any rectangle to exceed 90°C. We can do as follows :

```

double q[4] = {1.3e6,5e5,1e6,5e5}; // power density when no throttling
atmi_rectset sensors;
atmi_rectset domain[4]; // one domain per sensor
atmi_rectset_init(&sensors);
for (i=0; i<4; i++) {
    atmi_rectset_add(&sensors , i);
    atmi_rectset_init(&domain[ i ]);
}
// sensors 0 and 1 throttle rectangles 0 and 1
atmi_rectset_add(&domain[0],0);
atmi_rectset_add(&domain[0],1);
atmi_rectset_add(&domain[1],0);
atmi_rectset_add(&domain[1],1);
// sensors 2 and 3 throttle rectangles 2 and 3
atmi_rectset_add(&domain[2],2);
atmi_rectset_add(&domain[2],3);
atmi_rectset_add(&domain[3],2);
atmi_rectset_add(&domain[3],3);
// compute warm-up power density
atmi_steady_throttle(&p,4,rc,&sensors ,domain,90-35,q);

```

Once the thermal simulator is initialized with *atmi_simulator_init*, each call to the following function simulates a time-step :

```

void atmi_simulator_step(atmi_simulator *ts ,
                        double q[] /* power density */);

```

where parameter q[] is the power density in each rectangle during this particular time-step. After each execution of *atmi_simulator_step*, the temperature at each sensor is stored in the array *temperature[]* of the *atmi_simulator* structure, that is, *ts.temperature[i]* gives temperature at the *ith* sensor.

The example at the beginning of Section 4.2 could be treated as follows :

```

double a = 0.001; // source length
double q = 3e6; // power density
double duration = 0.001; // power is on for 1 ms
atmi_simulator ts;
atmi_rect rc[1] = {{-a/2,-a/2,a/2,a/2}};
double q0[1] = {0}; // start with null temperature (=ambient)
double qvar[1] = {0};
atmi_rectset sensors;

atmi_rectset_init(&sensors);
atmi_rectset_add(&sensors ,0);

atmi_simulator_init(&ts ,NULL,&p,1e-4,1,rc,&sensors ,q0);

```

```

while (ts.t <= 0.01) {
    printf("%f %f\n", ts.t, ts.temperature[0]);
    qvar[0] = (ts.t < duration)? q : 0;
    atmi_simulator_step(&ts, qvar);
}

atmi_simulator_freemem(&ts);

```

The function

```

void atmi_simulator_freemem(atmi_simulator *ts);

```

permits freeing the dynamically allocated memory (it should be used if we want to re-initialize the same thermal simulator).

5 ATMI validation

The validation of ATMI consisted mainly in validating the mathematical solution. Having an exact mathematical solution to a boundary-value problem does not guarantee the accuracy of the numerical evaluations on a computer (numerical integration, truncation of infinite sums, etc...). Several examples are provided in the ATMI package, that serve to validate the implementation :

- The program **example_Goh.c** reproduces the results published in the appendix of reference [6]
- The program **example_Xu.c** reproduces some of the results published in [11]
- The program **example_Fisher** reproduces Figure 3 in reference [4]

6 ATMI limitations

If there are not too many rectangles and sensors, ATMI is generally faster than traditional numerical methods like finite-differences or finite-elements. However, the speed and ease-of-use of ATMI was obtained by sacrificing generality. The main approximation in ATMI is that coming from the model itself, described in Section 1. In particular ATMI does not model chip edges. More precisely, it is assumed that layer 1 width equals layer 2 width, that is, L . For example, to model a $2\text{ cm} \times 2\text{ cm}$ chip with a 10 cm -wide heat-sink, parameter L must be set to 0.1. The chip dimensions are ignored. One advantage is that the user does not need to worry about the origin when specifying rectangle coordinates. What matters is the relative position of rectangles. However, the user must be aware of this limitation of ATMI.

Another limitation is the number of layers in ATMI, i.e., two layers (or four, considering that conductances h_1 and h_2 simulate thin layers). Also, the heat-sink fins are not modeled explicitly, but with an effective heat-transfer coefficient h_2 (this simplification is commonly used when modeling temperature in microprocessors [11, 5]). Another limitation is the linear approximation coming from fixed thermal characteristics for silicon. These are not the only approximations in ATMI (among others : ignoring the heat path from the chip pins to the board, assuming a constant and uniform ambient temperature, etc...).

7 Advanced use

With a little physical intuition, it is possible to adapt the model to situations that depart from the strict ATMI model.

The ATMI package provides a few examples of this. The program **example_Xu.c** reproduces results of [11]. However, the physical system modeled in [11] does not fit exactly the ATMI model. The system modeled in [11] features more layers than ATMI, and the heat-sink base features a vapor chamber. Nevertheless, the results output by **example_Xu.c** are reasonably close to those published in [11]. The presence of the vapor chamber allows to define an isothermal plane whose temperature equals the total power times the heat-sink thermal resistance. We use this temperature as if it were the ambient T_{amb} , and we add to it the relative temperature numbers given by ATMI. Layers that exceed the number of layers in the ATMI model are collapsed in a single infinitely thin layer modeled with parameter h_2 .

The program **example_Goh.c** shows that neglecting the chip dimensions has little impact on temperature, provided the measure point is not too close to a chip edge. However, this may not be the case if the measure point is close to an edge.

The program **example_chipedges.c** in the ATMI package gives the steady state temperature at the center of a square source on a $1\text{ cm} \times 1\text{ cm}$ chip. The chip and packaging dimensions are given on Figure 2. Other ATMI parameters are $k_1 = 120\text{ W/mK}$, $k_2 = 400\text{ W/mK}$ and $h_2 = 200\text{ W/m}^2\text{K}$.

We consider 3 possible source dimensions, 1 mm , 2 mm and 3 mm square side, 3 possible source locations, center, edge, and corner (cf. Figure 3), and 3 different values for the interface material thermal conductivity, $k_i = 1, 3, 10\text{ W/mK}$. The power dissipated by the source is constant and equal to 2 W whatever the source dimensions. Figure 5 shows the temperature obtained with ATMI (**example_chipedges.c**) and that obtained with the finite-element tool FF3D [2] (the mesh was generated with *gmsh* [1]).

For ATMI, we simulated the chip edges as described in Figure 4. When the source is located at the chip center, we use directly the temperature given by ATMI, as we are far from chip edges. However, when the source is located close to an edge, temperature in the source is underestimated by ATMI. The method used in **example_chipedges.c** is based on the method of images [3]. By adding a source image, we create a symmetry that forces the heat flux to be parallel to the plane corresponding to the chip edge, which simulates the presence of an edge (see Figure 4). However, if we compute the temperature contribution from the image source using the normal value of h_2 , we increase the global heat-sink temperature, which is not what a chip edge is supposed to do (a local modification of the geometry or material characteristics generally has little impact on temperature at remote locations). Instead, we compute the image source contribution using a very large value for h_2 . This way, the contribution we compute is mainly due to the chip edge. This intuitive method is approximative.. Nevertheless, as can be seen on Figure 5, it gives a good estimation of the impact of edges.

This method was not implemented as an ATMI function. We do not feel chip edges are a fundamental constraint. If one is doing research to solve a severe temperature problem, it should be possible to assume that the chip floorplan can be reorganized to put high power-density regions away from edges, or that the chip can be enlarged to have a strip of non-dissipating silicon around. Still, if one wants an approximate estimation of the impact of chip edges, one can manipulate ATMI core functions as illustrated in program **example_chipedges.c**.

References

- [1] Gmsh. <http://www.geuz.org/gmsh/>.

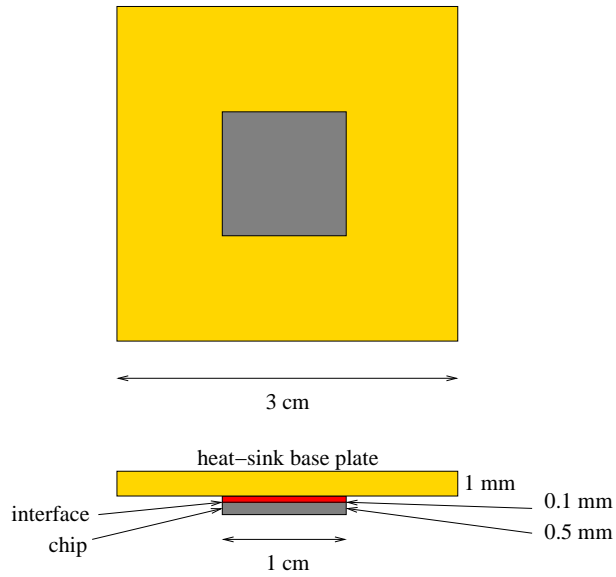


Figure 2: Example chip and packaging.

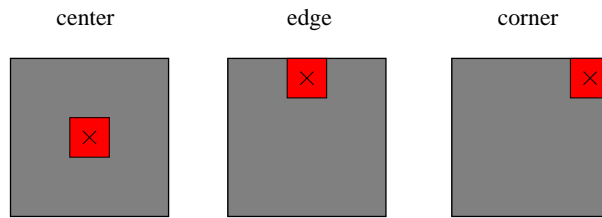


Figure 3: Square source at different locations on the chip.

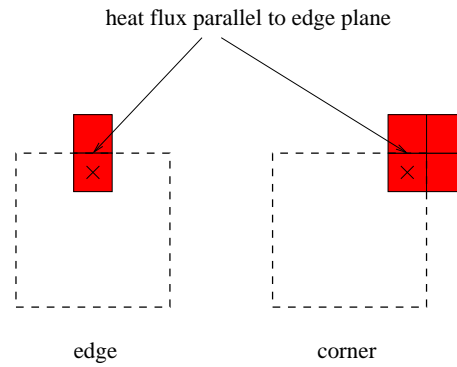


Figure 4: How one can simulate chip edges with ATMI : compute temperature as if there were no edges, then set $h_2 \rightarrow \infty$, add extra sources (method of images), and add the resulting temperature to the previous value.

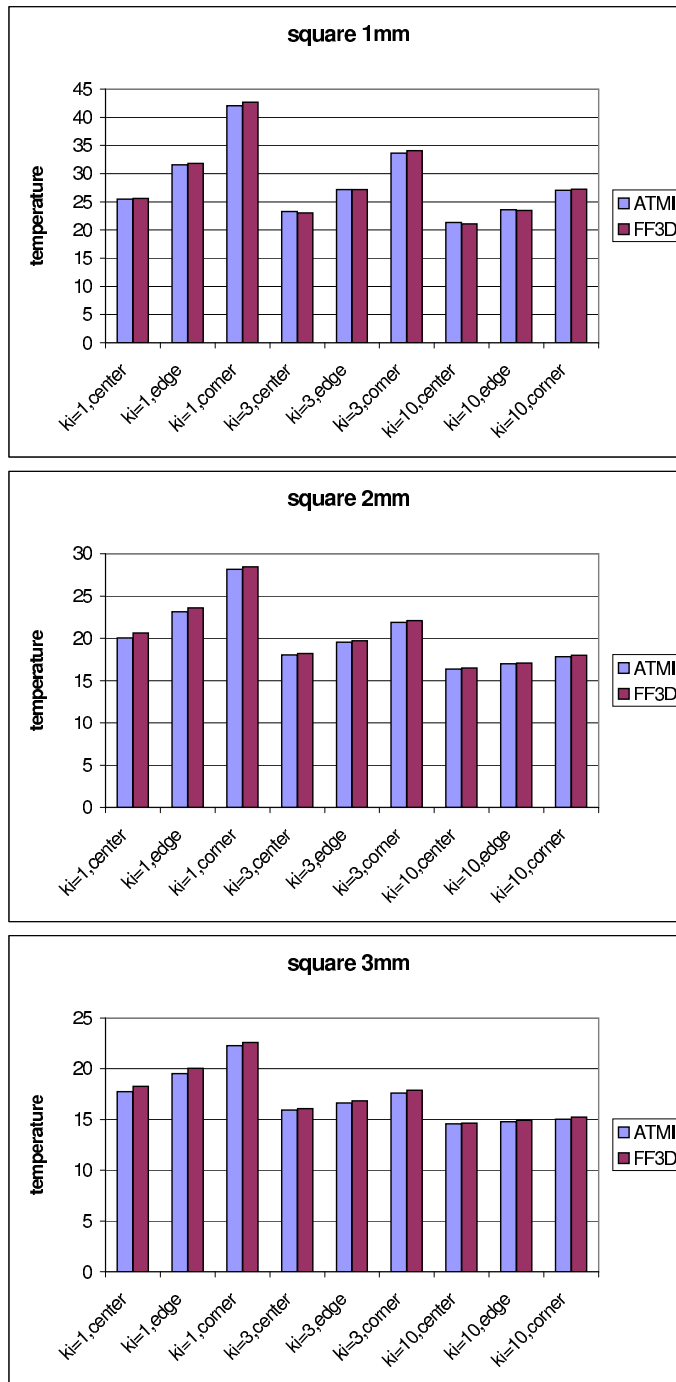


Figure 5: Temperature at the source center, for different source dimensions (1 mm, 2 mm, 3 mm square side), different source locations (center, edge, corner), and different values of the interface material thermal conductivity ($k_i = 1, 3, 10 \text{ W/mK}$). Comparison between ATMI (example_chippedges.c) and the finite element tool FF3D.

- [2] freeFEM3D. <http://www.freefem.org/ff3d/>.
- [3] H.S. Carslaw and J.C. Jaeger. *Conduction of heat in solids*. Oxford University Press, 1959.
- [4] T.S. Fisher, C.T. Avedisian, and J.P. 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), February 1996.
- [5] 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.
- [6] T.J. Goh, K.N. Seetharamu, G.A. Quadir, Z.A. Zainal, and K.J. Ganeshamoorthy. Thermal investigations of microelectronic chip with non-uniform power distribution : temperature prediction and thermal placement design optimization. *Microelectronics International*, 21(3):29–43, December 2004.
- [7] Intel. *Intel Core i7 Extreme Edition and Intel Core i7 Processor and LGA1366 Socket, Thermal / Mechanical Design Guide*, March 2009. Section 5.3.1.
- [8] P. Michaud and Y. Sazeides. ATMI : an analytical model of temperature in microprocessors. atmi-1.0 package, 2006.
- [9] P. Michaud, Y. Sazeides, A. Sez nec, T. Constantinous, and D. Fetis. An analytical model of temperature in microprocessors. Research report PI-1760, IRISA, November 2005.
- [10] E.C. Samson, S.V. Machiroutu, J.-Y. Chang, I. Santos, J. Hermerding, A. Dani, R. Prasher, and D.W. Song. Interface material selection and a thermal management technique in second-generation platforms built on Intel Centrino Mobile Technology. *Intel Technology Journal*, 9(1), 2005.
- [11] 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.