

ATMI:
Analytical Model of Temperature
in Microprocessors

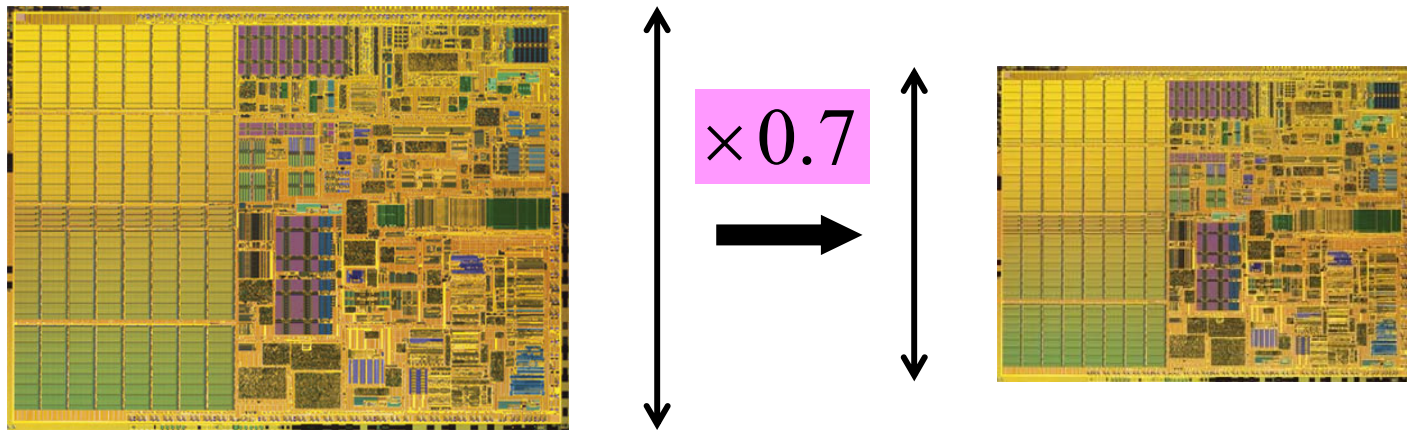
Pierre Michaud
IRISA/INRIA

Yiannakis Sazeides
University of Cyprus

Temperature must be limited

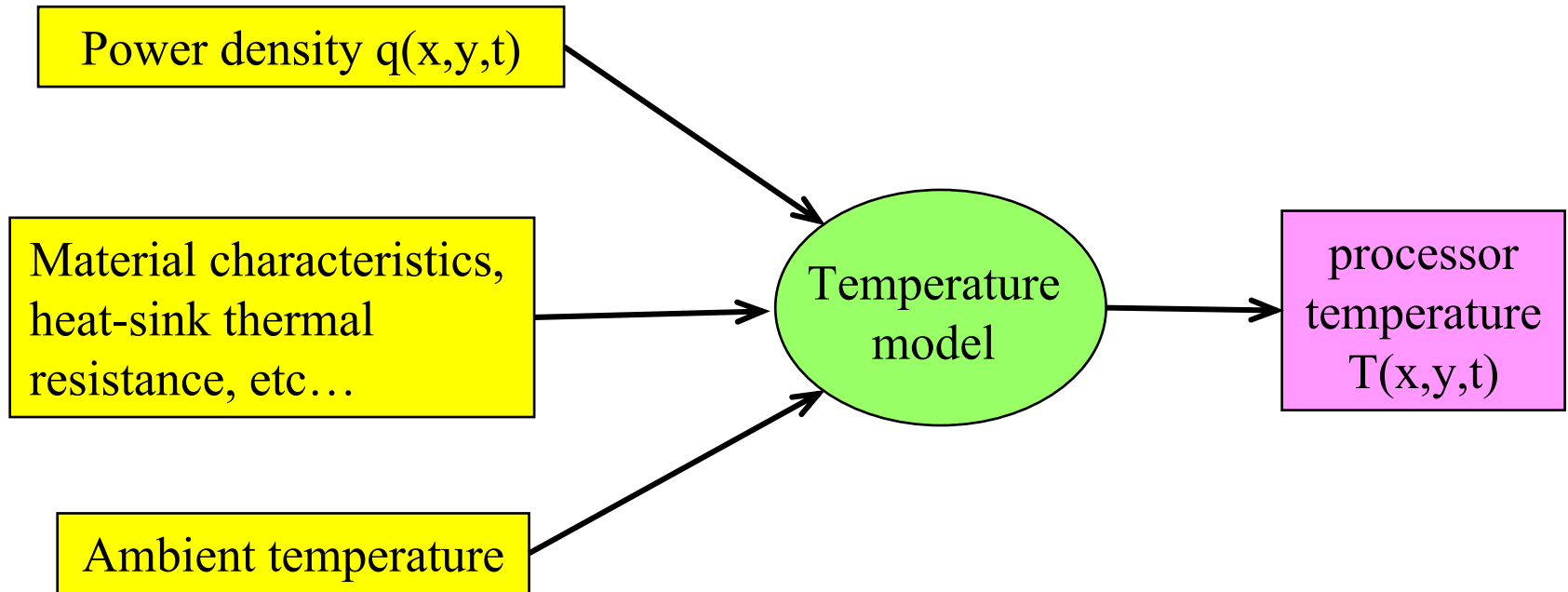
- Typically $< 100\text{ }^{\circ}\text{C}$
- Microprocessors age faster with higher temperature
 - $\rightarrow +10^{\circ}\text{C}$ halves mean-time to failure
- Subthreshold leakage increases with temperature
 - \rightarrow static power consumption
- High temperature \rightarrow circuit delays increase

Temperature constraint $>$ power constraint



To keep the same temperature,
power consumption must be $\sim \times 0.7$

Processor temperature model



Errors on inputs does not mean that temperature numbers are random

The model must be consistent with physics

The heat equation

Thermal conductivity ($\text{WK}^{-1}\text{m}^{-1}$)

$$g + \vec{\nabla} \cdot (k \vec{\nabla} T) = C \frac{\partial T}{\partial t}$$

3D power density (Wm^{-3})

Heat capacity ($\text{JK}^{-1}\text{m}^{-3}$)

- Simplify

- uniform material characteristics, independent of temperature
- 2D power density ($g=0$)



$$\nabla^2 T = \frac{1}{\alpha} \frac{\partial T}{\partial t}$$

Thermal diffusivity (m^2s^{-1})

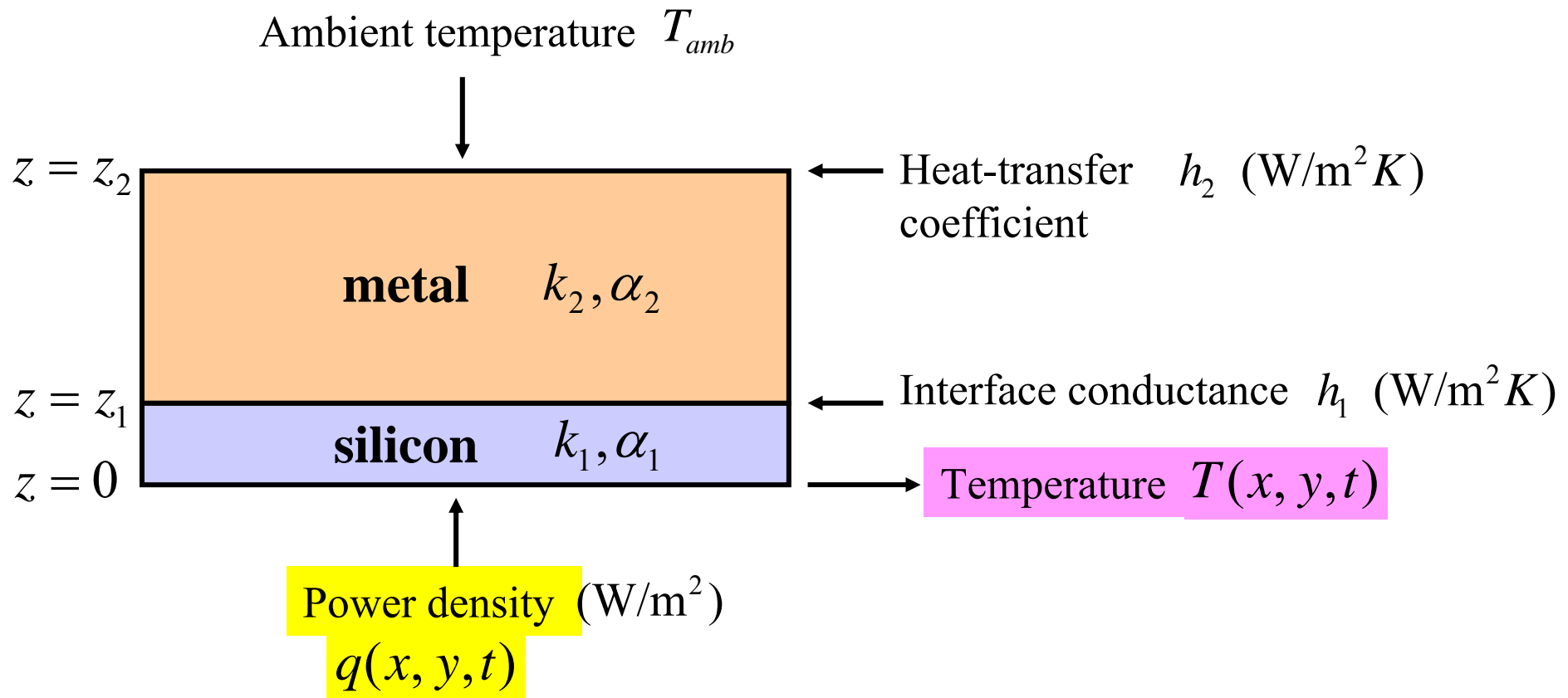
Solving the heat equation with boundary conditions

- General purpose numerical methods
 - Finite differences
 - Finite elements
 - ...
- Ad hoc analytical model → [ATMI](#)

ATMI

- Simplified chip and package model
- Simplified boundary conditions
- Explicit analytical solution to the heat equation
 - No discretization of the temperature field
- `atmi.c` + `atmi.h` → ~ 2000 lines of C
 - Released under GNU GPL
 - Uses the GNU Scientific Library

ATMI physical model



Principle of superposition

Define $u = T - T_{amb}$

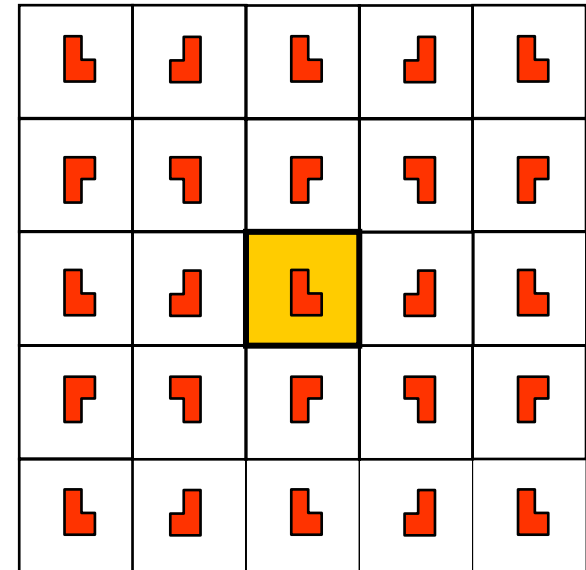
$$q_1(x, y, t) \rightarrow u_1(x, y, t)$$

$$q_2(x, y, t) \rightarrow u_2(x, y, t)$$

$$\beta_1 q_1 + \beta_2 q_2 \rightarrow \beta_1 u_1 + \beta_2 u_2$$

Method of images

- Heat-sink width simulated with infinite number of copies of power sources
 - In practice, finite number of copies
- Advantage: can assume silicon and metal layers extend to infinity



Point source solution

$$\delta(x)\delta(y)H(t) \rightarrow u_p(x, y, t)$$

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

$$S_1 = \sqrt{\sigma^2 + \frac{s}{\alpha_1}} \quad S_2 = \sqrt{\sigma^2 + \frac{s}{\alpha_2}}$$

$$r = \sqrt{x^2 + y^2}$$

$$E = \frac{k_2 S_2 - h_2}{k_2 S_2 + h_2} e^{-2(z_2 - z_1)S_2}$$

$$F = \frac{k_1 S_1}{h_1} + \frac{k_1 S_1}{k_2 S_2} \times \frac{1 + E}{1 - E}$$

$$G = \frac{F - 1}{F + 1} e^{-2z_1 S_1}$$

$$\text{Laplace transform } U_p(r, s) = \int_0^{+\infty} u_p(r, t) e^{-st} dt = \frac{1}{2\pi k_1 s} \int_0^{+\infty} \frac{1 + G}{1 - G} \times \frac{J_0(r\sigma)}{S_1} \sigma d\sigma$$

$u_p(r, t)$ obtained with Gaver-Stehfest numerical Laplace transform inversion

$$\text{Steady state: } \lim_{t \rightarrow +\infty} u_p(r, t) = \lim_{s \rightarrow 0} s \times U_p(r, s)$$

Spatial convolution

Apply principle of superposition on space variables

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

Implementation of spatial convolution

- 2 methods used in ATMI
 1. Rectangles with uniform power density
 - → Temperature at center of rectangles declared as “sensors”
 - Computation time proportional to number of rectangles times number of “sensors”
 2. Detailed power density map ([steady-state](#))
 - $N \times M$ identical squares with uniform power density
 - Discrete convolution using [Fast Fourier Transform](#)
 - → Temperature at center of each square

Time convolution

Apply principle of superposition on time variable

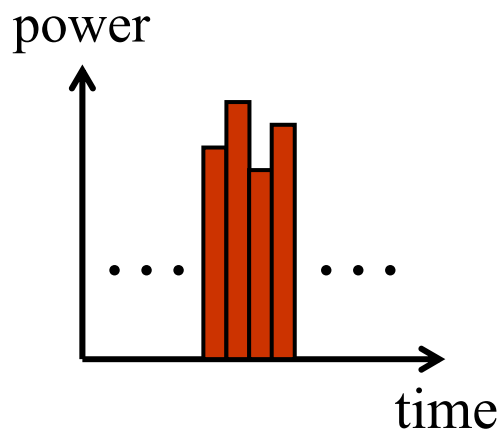
$$q(x, y)p(t) \rightarrow \int_0^{t'} p(t) \frac{\partial u_H}{\partial t}(x', y', t' - t) dt$$

- Modern microprocessors feature thermal sensors
- Thermal throttling → power density depends on temperature
- ATMI does not use FFT for time convolutions
- Original method: [event compression](#)

Event compression: principle

Time discretization: define time step τ

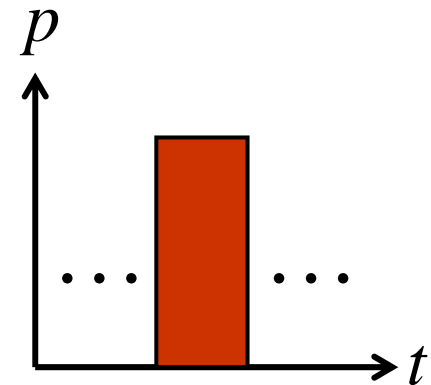
$$\begin{aligned}
 u(m\tau) &= \sum_{n=0}^{m-1} \int_{n\tau}^{(n+1)\tau} p(t) \frac{\partial u_H}{\partial t}(m\tau - t) dt \\
 &= \sum_{n=0}^{m-1} p(n\tau) \times [u_H((m-n)\tau) - u_H((m-n-1)\tau)]
 \end{aligned}$$



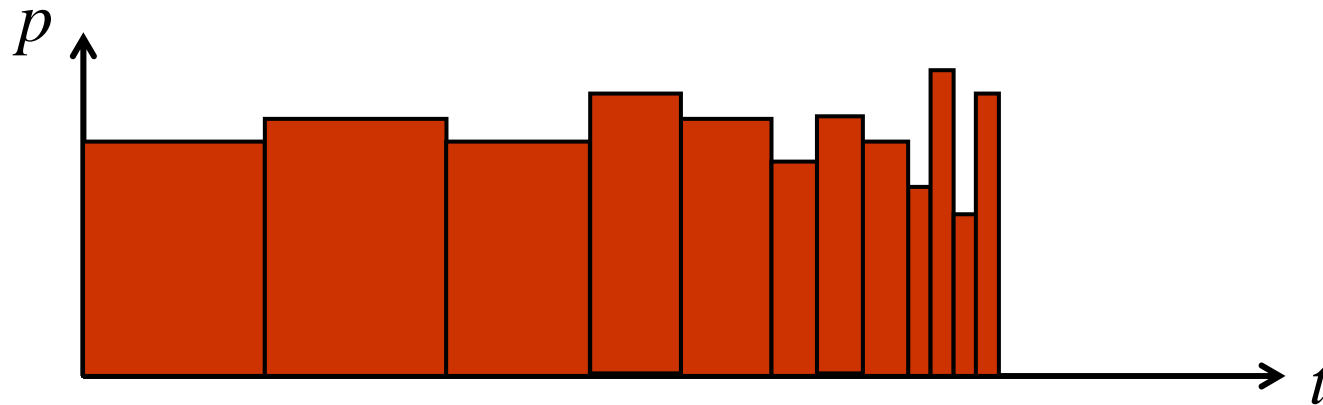
Merge several
consecutive power events
into a single event



Energy is conserved



Event compression applied progressively



Power events get merged as they get old

ATMI data 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 (m2/s) */  
    double k2; /* layer 2 thermal conductivity (W/mK) */  
    double a2; /* layer 2 thermal diffusivity (m2/s) */  
    double h1; /* layer 1/layer 2 (W/m2K) */  
    double h2; /* layer 2/ambient (W/m2K) */  
    double L; /* width (m) */  
    ...  
};
```

The user provides 7 parameters

```
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) */
```

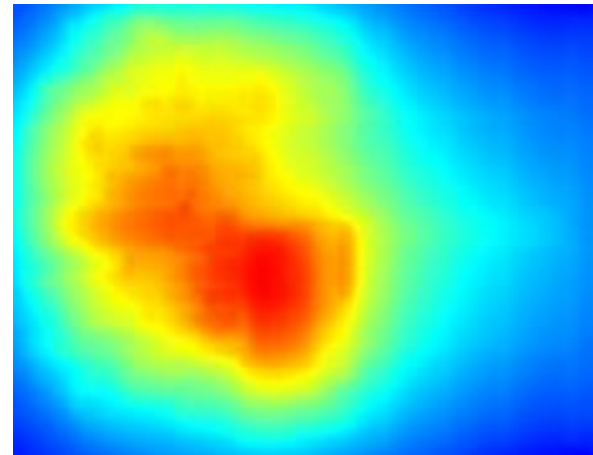
Steady-state temperature

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

```
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);
```

Example: 230 x 180 squares

→ ~ 2 seconds on a 3 Ghz Pentium 4



Time-dependent temperature

Thermal simulator initialization:

```
void atmi_simulator_init(  
    struct atmi_simulator *ts,  
    const char *filename,  
    struct atmi_param *p,  
    double timestep, /* time step */  
    int nrect, /* number of rectangle sources */  
    int nsens, /* number of "sensors" */  
    struct atmi_rect rc[], /* rectangles coord */  
    int sensor[], /* which rectangles are sensors */  
    double initq[],  
    double wmax);
```

Each call to this function simulates a time step:

```
void atmi_simulator_step(  
    struct atmi_simulator *ts,  
    double q[]); /* power density in each rectangle */
```

Example: 2 rectangles

```

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

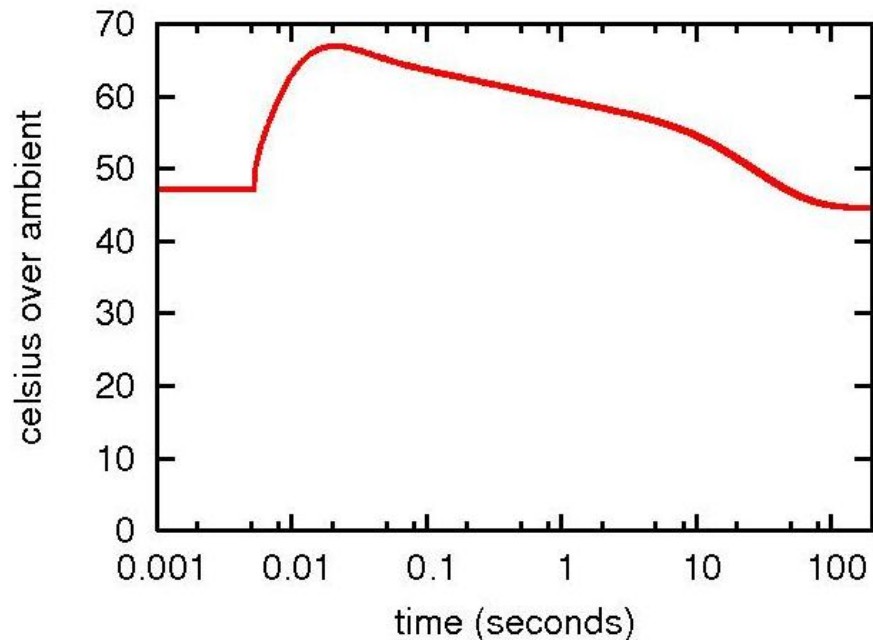
#define XS 0.002
#define XL 0.008
#define Q1 1e6
#define Q2 4e6

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]);
    }
}

```

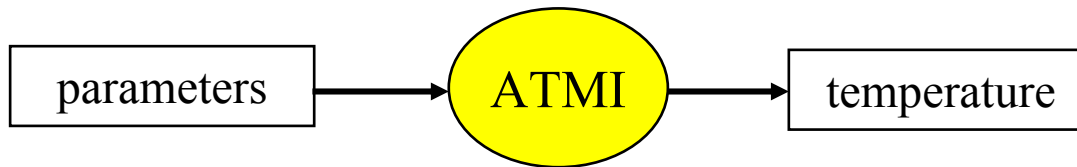


Time step = 0.2 ms

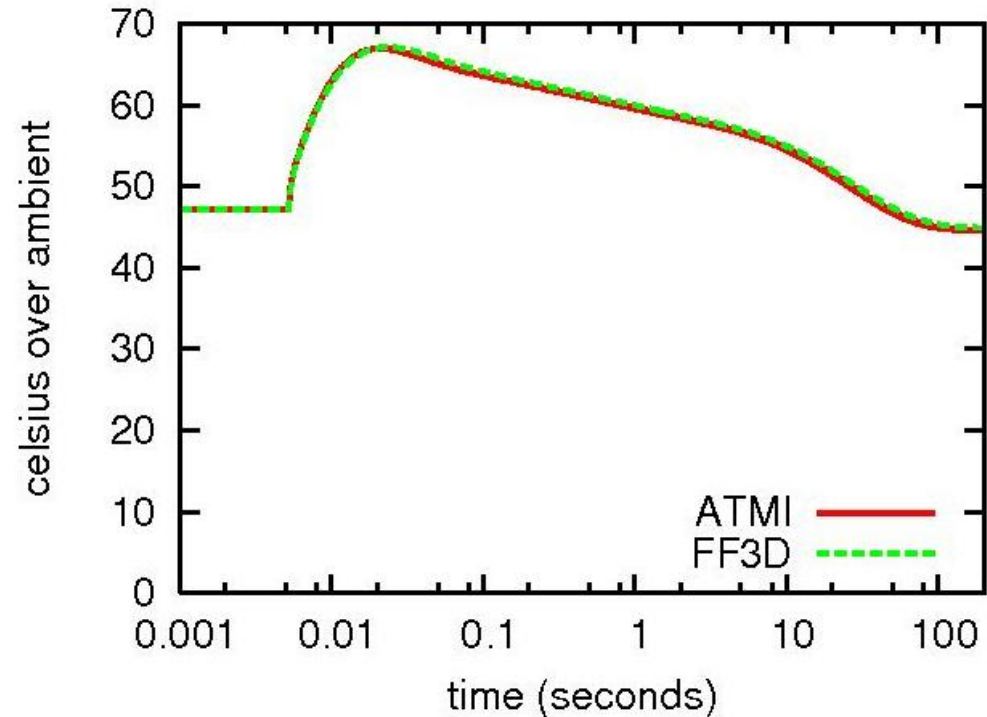
At each time step

- update power densities
- update temperature

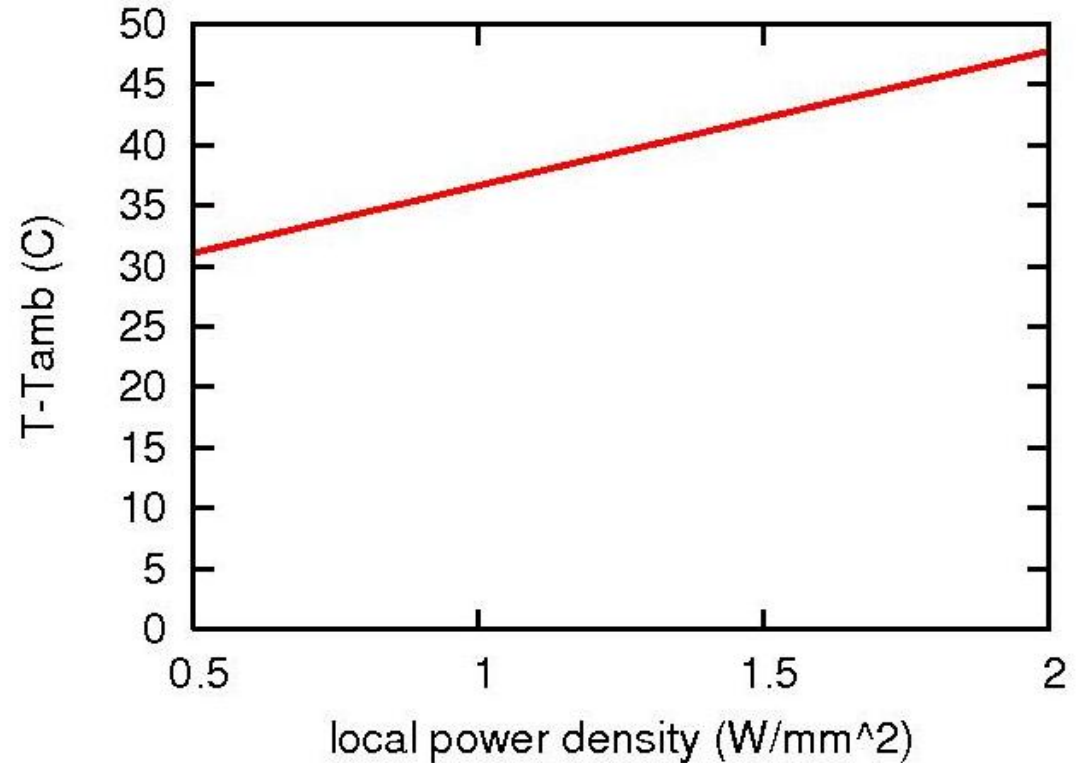
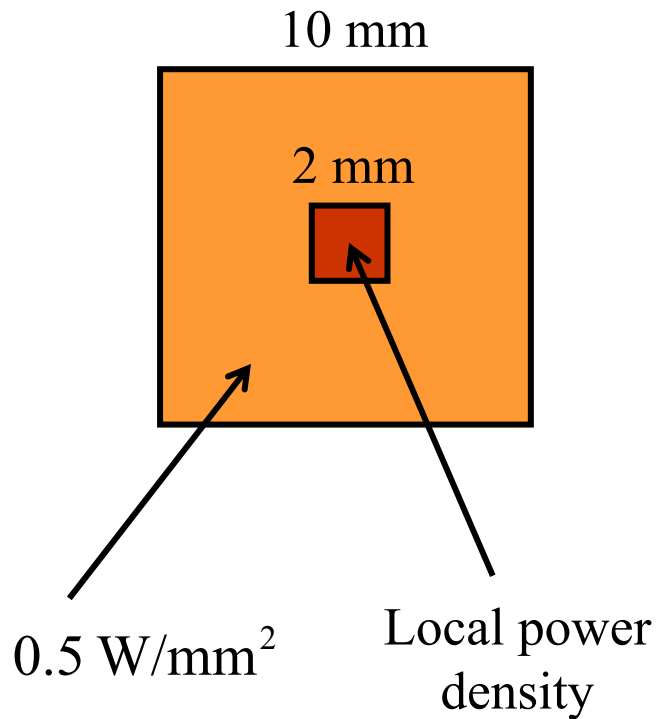
Validation



Many comparisons with
finite elements



One last example



Local Power density

0.5 → 2

×4

Total power

50 → 56

×1.12

Temperature

31 → 48

×1.5

Conclusion

- Our studies based on ATMI
 - Thread migration on thermally-constrained multicores (TCMC)
 - Priority-aware thread scheduling policies on TCMC
- ATMI is maintained, but no plan for further development

<http://www.irisa.fr/caps/projects/ATMI>

- Funding for this work
 - INRIA
 - University of Cyprus
 - Cyprus Research Promotion Foundation
 - French Ministry of Foreign Affairs
 - HiPEAC NoE