Advanced Design Optimization (MANE 6963)

Independent Study:
Optimization of Temperature Profile of billet in Extrusion Process

Sagar Bhatt

Contents

List of Figures

Executive Summary

Microstructures in a polycrystalline material are responsible for many of its mechanical properties. In this project, the temperature profile of a cross-section of the billet is optimized to minimize the standard deviation of the grain size. This will result in a uniform grain growth. To implement this, Monte Carlo simulation based Grain Growth Package was used where a frictional heat equation solver was added. A surrogate model of initial temperature profile was created using 10 design variables and the whole system was optimized using the `fmincon` *solver of MATLAB. Final solution results in uniform grain structure and a reduction in standard deviation from 2-4 to 0.9 units of length.*

1. Introduction

Several critical mechanical properties such as yield, fatigue and creep depend on the grain size and distribution of microstructures in polycrystalline material. Although the relation between structure, properties and the processes is understood very well, the control of these microstructure variations and properties is much more complex. Currently no method exists to dynamically control material processing to achieve target microstructure. This project is a step in that direction. The problem under consideration is hot extrusion of a copper billet. The objective is to optimize the temperature field on the cross-section of the billet by minimizing the standard deviation of the grain sizes. This will result in a uniform microstructure evolution. For simplicity, 2D case is being considered at the point where copper is flowing through the die as shown in fig. 1. At this stage of extrusion process is nearly complete. To further simplify the model, only heat generated due to friction is being considered since deformation has already taken place and the dynamic source of heat at this point is just friction. Also, modeling deformation and material flow would be way beyond the scope of this project. This model goes on to show that for problems of larger scale, similar concepts can be applied to achieve the initial temperature field across the billet. In further sections, the concepts are presented that have been used to model the analysis section of the problem.

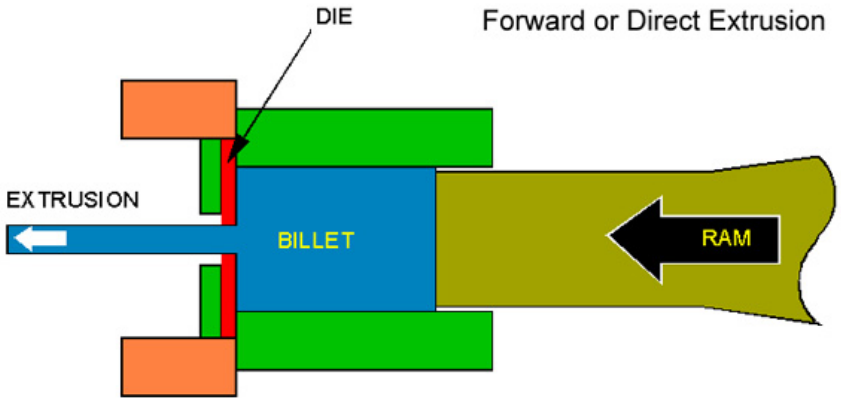

Figure 1: *Schematic of the extrusion process*

Image Source: `http:www.eling.rsslikeAl%20profili13_ForwardorDirectExtrusion.jpg`

1.1. Grain Growth Using Monte-Carlo Simulations

One of the most widely used methods to simulate grain growth in metals is the Monte Carlo (MC) method based on the Potts model. A simulation package was developed at the Computational Solid Mechanics laboratory at RPI by Y.Tan *et. al*[4] . The methodology goes as follows:

1. A lattice site i located at position x_i, where $i = 1, N_L$ and N_L is the total number of lattice points, is selected with a Site Selection Probability(SSP) function $P(x_i)$.

2. The orientation at the selected lattice site i, denoted S_i^0 , is randomly switched to one of its neighbors orientations S_i^n , where the neighbor is selected with uniform probability, and the change in energy ΔE due to the switch is computed according to

$$\Delta E = J \sum_{j=1}^{N_n} [\delta(S_i^0, S_i^j) - \delta(S_i^n, S_i^j)]$$

   where N_n is the number of neighboring lattice sites considered, δ is the Kronecker delta function, which equals one if the orientations compared are the same and zero otherwise.

3. The final orientation is selected with a probability function $\check{P}(\Delta E, T(x_i))$ that depends on the change in energy and local temperature $T(x_i)$ at lattice site i. Here, we use a function of the form

$$\check{P}(\Delta E, T(x_i)) = \begin{cases} p_m(T(x_i)) & if \quad \Delta E \leq 0, \\ p_m(T(x_i)) exp\left(-\frac{\Delta E}{k_B T_s}\right) & if \quad \Delta E > 0, \end{cases}$$

where p_m is the probability of switching if the change is energetically favorable, k_B is Boltzmanns constant and T_s is the simulation temperature.

MC simulations can be scaled to physical processes by fitting experimental data. In MC simulations, the evolution of average grain size $\bar{D}$ follows:

$$\bar{D}^m - \bar{D}_0^m = \lambda^m C t_{mc}$$

where λ is the corresponding physical length of the lattice spacing, $\bar{D}_0$ is the initial grain size, and C and m are parameters fitted from simulations.

Site Selection Probability is defined as:

$$P(x) = \frac{exp\left(-\frac{Q}{RT(x)}\right)\left[\left(\frac{\bar{D}_0}{\lambda}\right)^m + Ct_{mc}(x^*)\right]^{\frac{n}{m}-1}}{exp\left(-\frac{Q}{RT(x^*)}\right)\left[\left(\frac{\bar{D}_0}{\lambda}\right)^m + Ct_{mc}(x)\right]^{\frac{n}{m}-1}}$$

Where, n, t, R, and Q represent the grain growth exponent, time, gas constant, and activation energy for grain boundary migration, respectively. As we can see that this probability, which basically governs how much a particular grain will grow is a function of temperature. Hence, in this work, a temperature field due to friction is provided to this package. More on the temperature field solution in the next section.

2. Methodology

MATLAB's Optimization Toolbox's solver `fmincon` [1] was used to for optimization. The main simulation package as well as majority of the analysis code was written in C++, the simulation was run by a script in MATLAB which communicates with the package by means of data files. MATLAB script runs the analysis and receives the grain sizes and `fmincon` tweaks the design variables until the standard deviation of the grain sizes drops below a certain value. Ideally, the minimization of standard deviation of grain size could be allowed to go through the optimization process constrained only by the bounds on Temperature field, but owing to smaller computational resource, a hard limit of standard deviation was set to 0.9. The specifics of the solution methodology is explained in the subsequent sub-sections.

2.1. Heat Equation Solution:

Two-dimensional heat equation for isotropic material can be written as:

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T + \frac{Q}{\rho c_p}$$

This equation can be discretized using finite difference as follows:

$$\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} = \kappa \left(\frac{T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n}}{(\Delta x)^2} + \frac{T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n}}{(\Delta y)^2} \right) + \frac{Q_{i,j}^{n}}{\rho c_p}$$

Rearranging,

$$T_{i,j}^{n+1} = T_{i,j}^{n} + s_x \left(T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n} \right) + s_y \left(T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n} \right) + \frac{Q_{i,j}^{n}}{\rho c_p}$$

where,

$$s_x = \frac{\kappa \Delta t}{(\Delta x)^2} \quad and \quad s_y = \frac{\kappa \Delta t}{(\Delta y)^2}$$

Since this is an explicit scheme, the stability of this method is guaranteed only if the following CFL condition is satisfied:

$$\frac{2\kappa \Delta t}{min((\Delta x)^2, (\Delta y)^2)} \leq 1$$

Heat Generation due to Friction

As described by Saha [3], the heat generation by friction between the extruded part and the die can be expressed as:

$$q_f = \frac{\tau_f V}{J}$$

where V is the relative velocity between the die and the extrude and is taken as $1cm/s$ [3], J is the mechanical equivalent of heat and is a constant ($4186.8W.s/Kg$) and τ_f is the frictional stress which is given by

$$\tau_f = \frac{\bar{\sigma}}{\sqrt{3}}$$

Where, $\bar{\sigma}$ is the flow stress of copper. Flow stress varies with temperature but for our purposes, it is assumed to hover around $200MPa$ at the temperatures under consideration. Dynamic flow stress can of course can be modeled separately. The Q computed thus is applied on the edges of the 2D square under consideration, hence becoming Neumann boundary condition.

2.2. Design Variables

A surrogate model based approach was taken for this work. Entire temperature field cannot be taken as design variables as it would make the problem unnecessarily expensive. Hence, The temperature field was formulated as follows

$$T(x, y) = T_{min} + \sum_{k=1}^{5} \frac{T_k sin(\frac{x\pi}{L})}{k} + \sum_{j=1}^{5} \frac{T_j sin(\frac{y\pi}{H})}{j}$$

where, L and H are the length and height of the cross-section respectively and T_{min} is the minimum temperature that the temperature field is allowed to have (in this case $400°C$). Thus, design variable becomes a vector of length 10 with upper half corresponding to x-axis and the lower half corresponding to y-axis. This a truncated sine series and 10 variables were chosen arbitrarily so as not to increase the computational cost too much. Higher accuracy can be achieved with more design variables.

2.3. Objective Function

As described in section 1, the objective of the project is to minimize the standard deviation of the grain size. Hence, the objective function includes an input of the design variables, and passes the initial temperature field to the grain growth package. It reads the final grain size file, computes standard deviation and returns the same.

2.4. Constraints

The following constraints were considered:

- **Lower Bound:** The vector containing the design variables with a value of 0(i.e. no variation in temperature field, just a constant value throughout).
- **Upper Bound:** The vector containing the design variables with a value of 600(i.e. the temperature field just shy of the melting point of copper at $1085°C$).
- **Additional constraint on the minimizer:** Considering the computational resources required for this problem, a constraint of 0.9 was put on standard deviation.(i.e. optimization stops at this value).

3. Results

Figure 2 shows the contour plot of temperature field with no optimization. Starting from constant temperature, the central region still has the minimum initial temperature of $400°C$ while near the boundary, the temperature has risen due the friction between the die and the extruded part. The distance traveled by the extrude here was the thickness of the die which again from Saha's work was taken to be 3 mm tick[3]. Given the velocity of $1cm/s$ implies that this whole process lasted $0.3s$. Hence, the relatively small temperature gain.

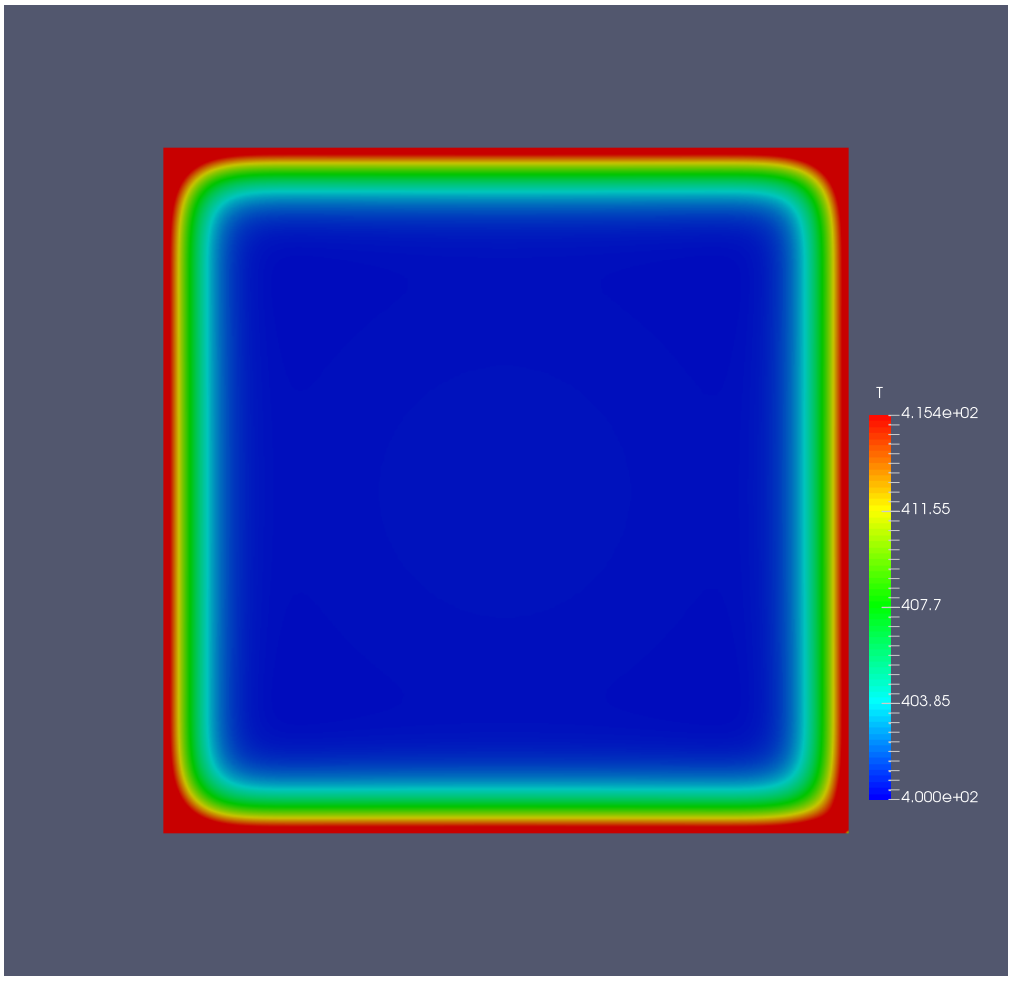

Figure 2: *Temperature evolution due to friction*

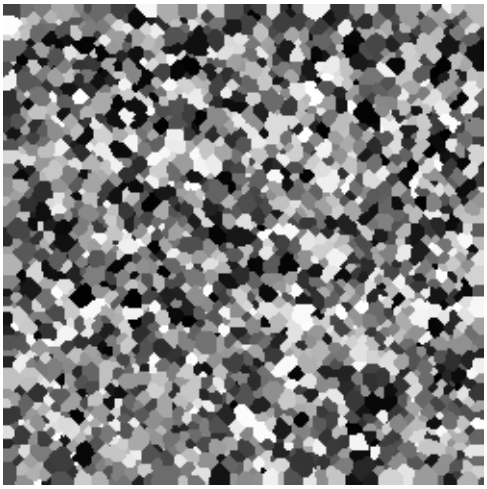

Figure 3: *Non-Uniform grain growth*

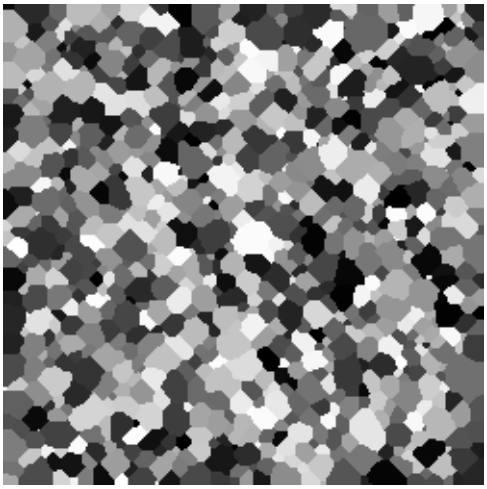

Figure 4: *Uniform grain growth*

Figure 3 Shows grain growth without optimization and we can see from the figure that there is significantly more growth near the boundaries on the other hand, fig. 4 shows a uniform growth after optimization. Numerically, unoptimized grain growth showed a standard deviation of 2-4 units of length(i.e. dy) along vertical lines while optimized grain growth showed standard deviation of just under 0.9 (because of our constraint).

4. Conclusion

The final converged solution satisfies all the constraints and results in a uniform grain structure. The optimized initial temperature profile may not have been uniform. In fact, the optimized temperature field is of no consequence to this problem , as long as the standard deviation is minimized. As we keep adding components to the heat equation solver like material anisotropy, deformation energy etc, the temperature profile will get more and more unpredictable. Hence, the optimized temperature profile has not been given any weightage in the results. The point to note is that the solution can be improved by using more design variables and having lower tolerance on the standard deviation.

References

[1] Matlab Documentation Center. Optimization Toolbox, Constrained Optimization, fmincon.

[2] Jason E. Hicken. Tutorial 6 slides in Advanced Design Optimizaton (MANE6963). Rensselaer Polytechnic Institute, Fall 2017.

[3] Pradip K. Saha. Thermodynamics and tribology in aluminum extrusion. *Wear*, 218(2):179–190, 1998.

[4] Y Tan, A M Maniatty, C Zheng, and J T Wen. Monte Carlo grain growth modeling with local temperature gradients. *Modelling and Simulation in Materials Science and Engineering*, 25, 2017.

5. Appendix

5.1. Main Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                         %
% Main file for Advanced Design Optimization Independent %
%                                                         %
%Study:  Optimizes Temperature field in a copper billet  %
%
% This will keep executing Grain Growth package with  %
%different temperatures until constrains are satisfied   %
                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%





 clear all;
 clc;

nx=250;   %grid size
L=nx;
x=1:1:nx;
y=x;
 TInit=400;    %min T
T1=ones(10,1);    %initial guess of design var
% T=TInit*ones(nx);

lb = 0*T1;     %upper and lower bound of design variable
ub= 600*T1;

fun=@(T1)obj(T1,TInit);
options=optimoptions(@fmincon,'Algorithm', 'sqp', 'Display', 'Iter','
    ObjectiveLimit',0.9);
[T1,dev,flag,output]=fmincon(fun,T1,[],[],[],[],lb,ub,[],options);
```

5.2. Objective Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                        %
% Ojective function for Independent Study       %
%                                                        %
% % Need:
% minimum temp TInit
% Array of design Variables T1
%                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[dev]=obj(T1,TInit)
nx=250;
x=1:1:nx;
y=x;
L=5;
for i=1:nx
    for j=1:nx
        for k=1:size(T1,1)/2
            T(i,j)=TInit+T1(k)*(sin(x(i)*pi/L))/k+T1(k+(size(T1,1)/2))*(sin(y(
    j)*pi/L))/k;
        end
    end
end
file='temp.dat';    %Writing temp field to file
dlmwrite(file,T,' ');

% system command to execute the C++ code with abouve temp field

[status,cmdout]=system('mpirun -n 4 ./2DfitMc/parallel_MC.out --nonstop 2
    trian.000.dat 480 120 0 1 100 100');

% Array of grain boundary Loacations
g=vertcat(load('grainSize1.dat'),load('grainSize2.dat'));


%Array of grain sized
k=1;
for f=1:size(g,1)-1
    if(g(f+1,2)-g(f,2) >0 )
        grainSizeArray(k)=g(f+1,2)-g(f,2);k=k+1;
    end
end

% Computing Standard deviation of grain sizes
dev=std(grainSizeArray);

end
```

5.3. Heat Equation Solver

```
// Solves the 2D heat equation with an explicit finite difference scheme

#include <vector>
#include <cmath>

std::vector<std::vector<double>> fricHeat(int dim_x, int dim_y,std::vector<std::vector<double>> &v)
{

    //Physical parameters
    double L        =   static_cast<double>(dim_x);
    double H        =   static_cast<double>(dim_y);
    double kappa    =   401;        //   Thermal Conductivity of Copper      [W/mK]
    double c        =   0.39;        // Specific heat  of Copper [kJ/(Kg K)]
    double J        =   4186.8;       // Mechanical Equivalent of heat [W.s/Kg or J/kg ]
    // Numerical parameters
    int nx      =   dim_x;          //   # gridpoints in x-direction
    int nz      =   dim_y;          //   # gridpoints in z-direction
    int nt      =   1 ;           //   Number of timesteps to compute
    double dx       =   L / static_cast<double>(nx); //   Spacing of grid in x-direction
    double dz       =   H / static_cast<double>(nz); //   Spacing of grid in z-direction


    // Compute stable timestep
    double dt           = dx * dx / (4 * kappa);
    double sx,sz;
    // Setup initial linear temperature profile
    double TInit = 460;
    std::vector<std::vector<double>> T;//(nx, std::vector<double>(nx, TInit));
    std::vector<std::vector<double>> Tnew;//(nx, std::vector<double>(nx, 0));
    T=v;
    Tnew=v;

    double tau = 2e8;       // Assumed flow stress [Pa]
    double V = 1e-2;        // velocity of the ram [m/s]
    std::vector<std::vector<double>> Q(nx, std::vector<double>(nx, 0));


    for (int i = 0; i < nx; ++i)
    {
        Q[i][0] = tau * V / (sqrt(3)*J); // volumetric, hence avoiding rho throughout
```

```
        Q[i][nx-1] = tau * V / (sqrt(3)*J);
        Q[0][i] = tau * V / (sqrt(3)*J);
        Q[nz-1][i] = tau * V / (sqrt(3)*J);
    }


        // Compute new temperature

        sx  = kappa * dt / (dx * dx);
        sz  = kappa * dt / (dz * dz);
        for (int j = 1; j < nx - 1; ++j)
        {
            for (int i = 1; i < nx - 1; ++i)
            {
                Tnew[i][j] = T[i][j] + sx * (T[i][j + 1] - 2 * T[i][j] + T[i][j - 1]) + sz * (T[i + 1][j] - 2 * T[i][j] + T[i - 1][j]) + Q[i][j] * dt / c;
            }
        }
        // Set boundary conditions
        for (int j = 1; j < nx - 1; ++j)
        {
            Tnew[0][j]   =  T[0][j] + sx * (T[0][j + 1] - 2 * T[0][j] + T[0][j - 1]) + sz * (2 * T[1][j] - 2 * T[0][j]) + Q[0][j] * dt / c;
            Tnew[nz-1][j]  =  T[nz-1][j] + sx * (T[nz-1][j + 1] - 2 * T[nz-1][j] + T[nz-1][j - 1]) + sz * (2 * T[nz - 2][j] - 2 * T[nz-1][j]) + Q[nz-1][j] * dt / c;
        }
        for (int i = 1; i < nz - 1; ++i)
        {
            Tnew[i][0] = T[i][0] + sx * (2 * T[i][1] - 2 * T[i][0]) + sz * (T[i + 1][0] - 2 * T[i][0] + T[i - 1][0]) + Q[i][0] * dt / c;
            Tnew[i][nx-1] = T[i][nx-1] + sx * (2 * T[i][nx - 2] - 2 * (T[i][nx-1])) + sz * (T[i + 1][nx-1] - 2 * T[i][nx-1] + T[i - 1][nx-1]) + Q[i][nx-1] * dt / c;

        }
        Tnew[0][0] = Tnew[0][1];
        Tnew[0][nx-1] = Tnew[0][nx-2];
        Tnew[nz-1][0] = Tnew[nz-1][0];
        Tnew[nz-1][nx-1] = Tnew[nz - 2][ nx-1];
        T            =    Tnew;
return Tnew;
}
```

5.4. Update Temperature

```
/*
This Function was added to grain growth package
to updae the temperature at each time step

*/
int niter=1;
template <int dim> void UpdateLocalTmp(MMSP::grid<dim, unsigned long>& grid,
    long double physical_time, double* temp){

vector<int> coords (dim,0);
// std::cout<<x1(grid,0)<<std::endl;
if(dim==2){
  std::vector<std::vector<double> > tempInit(dim_x, std::vector<double>(dim_y,
     0));
  for(int codx=x0(grid, 0); codx < x1(grid, 0); codx++)
     for(int cody=x0(grid, 1); cody < x1(grid, 1); cody++){
       coords[0] = codx;
       coords[1] = cody;

       if (niter>1)
       {
        tempInit[codx][cody]= grid.AccessToTmp(coords)-273;
      }
      else if (niter==1)
      {
         for(int codx=x0(grid, 0); codx < x1(grid, 0); codx++)
            for(int cody=x0(grid, 1); cody < x1(grid, 1); cody++){
                tempInit[codx][cody]=0;
                 }

      }

      }
      niter+=1;
  tempInit= fricHeat(dim_x, dim_y,tempInit);

   for(int codx=x0(grid, 0); codx < x1(grid, 0); codx++)
     for(int cody=x0(grid, 1); cody < x1(grid, 1); cody++){
       coords[0] = codx;
       coords[1] = cody;

        grid.AccessToTmp(coords)=273 + tempInit[codx][cody];

     }
   }
```