JOHANNES GUTENBERG UNIVERSITÄT MAINZ

GEO SCIENCES MAINZ

Pseudo-transient Iterations in coupled non-linear problems

Evangelos Moulas

evmoulas@uni-mainz.de

A toy problem based on physics

We will start with a simple problem that has a solid physical foundation. That is, heat diffusion with radiogenic heat production.

In the simplest case (and in 1d) that is: $\rho C \frac{dT}{dt} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) + Q$

And for constant parameters (and no advection) it becomes:

$$\frac{\partial T}{\partial t} = \underbrace{\frac{k}{\rho C}}_{D} \frac{\partial^2 T}{\partial x^2} + \underbrace{\frac{Q}{\rho C}}_{S}$$

A toy problem based on physics

The previous can be discretized as follows (finite differences):

$$\frac{T_i^{k+1} - T_i^k}{\Delta t} = D \frac{\left(T_{i-1}^k - 2T_i^k + T_{i+1}^k\right)}{\Delta x^2} + S$$

By solving for T_i^{k+1} explicitly we have:

$$T_i^{k+1} = T_i^k + \frac{\Delta t D}{\Delta x^2}\left(T_{i-1}^k - 2T_i^k + T_{i+1}^k\right) + \Delta t S$$

This is the simplest of all solutions (can be unstable – CFL condition)

A toy problem based on physics

From now on we can write the previous equation as follows:

$$T_i^{k+1} = T_i^k + \Delta t \dot{R}$$

where:

$$\dot{R} = \underbrace{\frac{D}{\Delta x^2}\left(T_{i-1}^k - 2T_i^k + T_{i+1}^k\right) + S}_{\text{original } RHS}$$

A toy problem based on physics

Now, if we perform iterations until $\Delta tR \approx 0$, it means that

$$T_i^{k+1} = T_i^k + \sim 0$$

And therefore we would have solved the elliptic problem (numerically)

$$0 = \underbrace{D\frac{\partial^2 T}{\partial x^2} + S}_{\approx \dot{R}}$$

We now have a way to solve elliptic equations. In fact, this can easily be adjusted for non-linear coefficients (formula needs to be rederived)

A toy problem based on physics

In summary, we took the original (time-dependent problem) and we solved for the steady state. Thus, if we are interested in the final state of T, we do not care about the physical time (for elliptic problems). The timestep is important only for stability.

$$\frac{\partial T}{\partial \boldsymbol{\tau}} = D\frac{\partial^2 T}{\partial x^2} + S \qquad \boldsymbol{\tau}: \text{Pseudo} - \text{time}$$

Why to do all this work?

- Because the explicit method can be quite fast in the GPUs and WE DO NOT NEED TO BUILD A MATRIX (we save a lot of memory in 3d problems)
- Keep in mind that geological problems have many degrees of freedom per node.

A toy problem based on physics

The fact that the discretized equation resembles the original form is very advantageous. This has led some researchers to refer to pseudo-transient iterations as “physics-inspired iterations”

In other fields of computational mechanics this method (in principle) is called the “Dynamic-Relaxation method”

$$\frac{\partial T}{\partial \boldsymbol{\tau}} = D\frac{\partial^2 T}{\partial x^2} + S \qquad \boldsymbol{\tau}: \text{Pseudo} - \text{time}$$

In this case, it is trivial to solve a non-linear problem
(e.g. using conservative finite differences).

This is actually known for a while…

JOURNAL OF COMPUTATIONAL PHYSICS **135,** 118–125 (1997)
ARTICLE NO. CP975716

A Numerical Method for Solving Incompressible Viscous Flow Problems*

Alexandre Joel Chorin

Courant Institute of Mathematical Sciences, New York University, New York, New York 10012

A numerical method for solving incompressible viscous flow problems is introduced. This method uses the velocities and the pressure as variables and is equally applicable to problems in two and three space dimensions. The principle of the method lies in the introduction of an artificial compressibility δ into the equations of motion, in such a way that the final results do not depend on δ. An application to thermal convection problems is presented.

where $R = UD/v$ is the Reynolds number. Our purpose is to present a finite difference method for solving (1a)–(1b) in a domain D in two or three space dimensions, with some appropriate conditions prescribed on the boundary of D.

The numerical solution of these equations presents major difficulties, due in part to the special role of the pressure in the equations and in part to the large amount of computer time which such solution usually requires, making it

Extending the problem

What will happen if we want an actual diffusion equation? How to consider the actual physical time?

$$\underbrace{\frac{\partial T}{\partial \tau} = D\frac{\partial^2 T}{\partial x^2} + S}$$

Before we wanted RHS=0

The solution is straightforward. We now need to modify the right-hand side and include the time derivative (no advection).

$$\underbrace{\frac{\partial T}{\partial \tau} = -\frac{\partial T}{\partial t} + D\frac{\partial^2 T}{\partial x^2} + S}$$

Now we want this RHS=0

At the end of the iterations, we satisfy the original diffusion equation (for a given step)

A toy problem based on physics

Our previous (discretized) equation was:

$$\frac{T_i^{k+1} - T_i^k}{\Delta\tau} = \underbrace{D\frac{\left(T_{i-1}^k - 2T_i^k + T_{i+1}^k\right)}{\Delta x^2} + S}_{\text{we had to satisfy this}}$$

$$\frac{T_i^{k+1} - T_i^k}{\Delta\tau} = \underbrace{-\frac{T_i^k - T_i^{OLD}}{\Delta t} + D\frac{\left(T_{i-1}^k - 2T_i^k + T_{i+1}^k\right)}{\Delta x^2} + S}_{\text{now we need to satisfy this}}$$

A toy problem based on physics

The previous can also be written as:

$$T_i^{k+1} = T_i^k + \boldsymbol{\Delta\tau}\dot{R}$$

Note that we have two different timesteps. The numerical and the physical.

where:

Also note that once the iterations converge, this method is equivalent to implicit method.

$$\dot{R} = \underbrace{-\frac{T_i^k - T_i^{old}}{\boldsymbol{\Delta t}} + \frac{D}{\Delta x^2}\left(T_{i-1}^k - 2T_i^k + T_{i+1}^k\right) + S}_{\text{modified } RHS}$$

A toy problem based on physics

The previous iteration strategy is equivalent to solving:

$$\frac{\partial T}{\partial \tau} = \overbrace{-\frac{\partial T}{\partial t} + D\frac{\partial^2 T}{\partial x^2} + S}^{F}$$

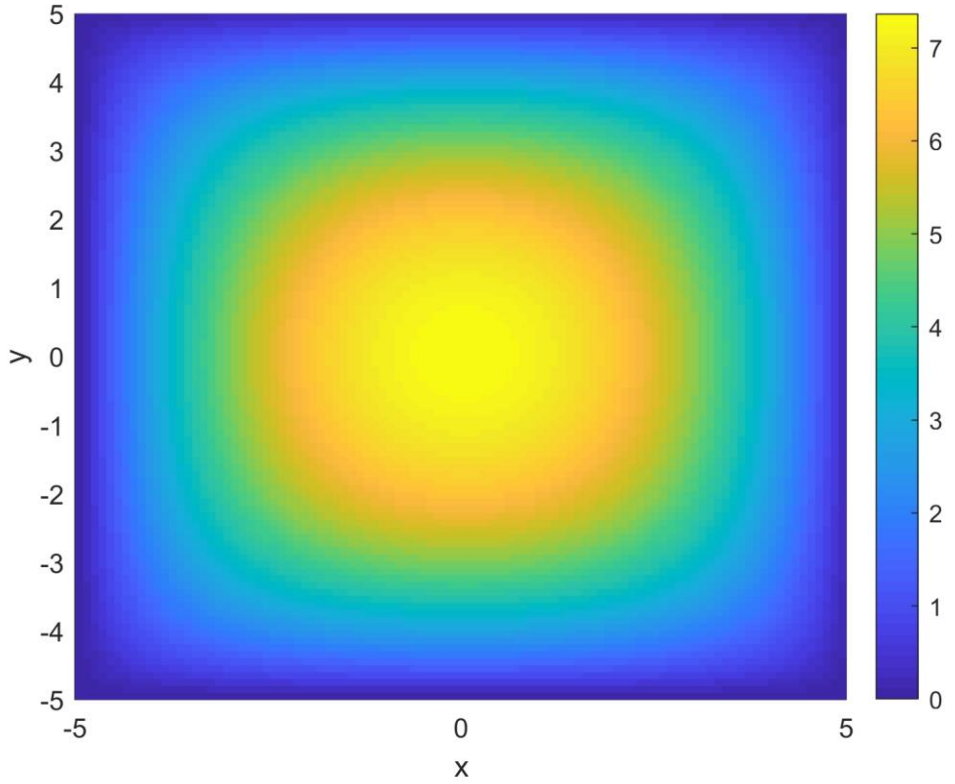
That can be solved following:

$$T^{k+1} = T^{k} + \Delta\tau \cdot (F)$$

This approach is extremely easy to implement, to teach, and to use for real world problems (more details in the Julia User Group Meeting).

A code example

```
clear,clc
%Physical params
D       =   1.0;        %Diffusivity
S       =   1.0;        %Source Term
Lx      =   [10 10];    %Length of domain
%Numerical Parameters
nx      =  100*[1 1];   %Numerical Resolution
CFL     =   0.2;        %CFL condition
Etol    =   1e-14;      %Error tolerance
%Make Grid
dx      =   Lx./(nx-1); %step
[x  y]  = ndgrid([-Lx(1)/2:dx(1):Lx(1)/2],...
                 [-Lx(2)/2:dx(2):Lx(2)/2]);
%Initial Conditions
Rdot    = zeros(nx(1)-2,nx(2)-2); %Initialize Residual
T       = zeros(nx(1),nx(2));     %Initialize T
ErrT    = 1e23; it      = 0;      %Initialize Error and Iterations
dt      = min(dx)^2/D*CFL;        %Timestep restriction
while ErrT>Etol
    it   = it + 1;
    qx   = -D*diff(T(:,2:end-1),1,1)/dx(1);
    qy   = -D*diff(T(2:end-1,:),1,2)/dx(2);
    Rdot = - diff(qx,1,1)/dx(1)...
           - diff(qy,1,2)/dx(2)...
           + S;
    T(2:end-1,2:end-1) = T(2:end-1,2:end-1) + dt*Rdot;
    ErrT = max(abs(dt*Rdot(:)));
end
```



This approach allows the implementation of non-linearities in a trivial manner (relaxation may be needed).

That can also become GPU compatible

The conversion of the previous code so that it can run in GPUs is easy.

one needs to keep in mind reducing the copying/reading from host to device.

This is particularly important for Geosciences applications where we have many dofs per node.

CopytoGPU.m

```
Rdot  = gpuArray(Rdot);
T     = gpuArray(T);
dx    = gpuArray(dx);
D     = gpuArray(D);
S     = gpuArray(S);
Etol  = gpuArray(Etol);
ErrT  = gpuArray(ErrT);
it    = gpuArray(it);
```

CopytoCPU.m

```
T     = gather(T);

```

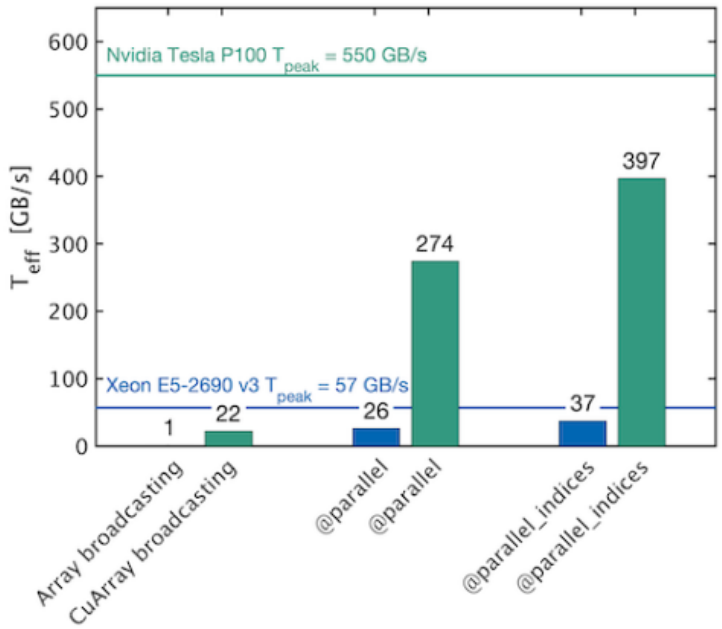
julia packages

JG|U GEO SCIENCES MAINZ

ParallelStencil.jl

CI passing

ParallelStencil empowers domain scientists to write architecture-agnostic high-level code for parallel high-performance stencil computations on GPUs and CPUs. Performance similar to CUDA C / HIP can be achieved, which is typically a large improvement over the performance reached when using only CUDA.jl or AMDGPU.jl GPU Array programming. For example, a 2-D shallow ice solver presented at JuliaCon 2020 [1] achieved a nearly 20 times better performance than a corresponding GPU Array programming implementation; in absolute terms, it reached 70% of the theoretical upper performance bound of the used Nvidia P100 GPU, as defined by the effective throughput metric, `T_eff` (note that `T_eff` is very different from common throughput metrics, see section Performance metric). The GPU performance of the solver is reported in green, the CPU performance in blue:

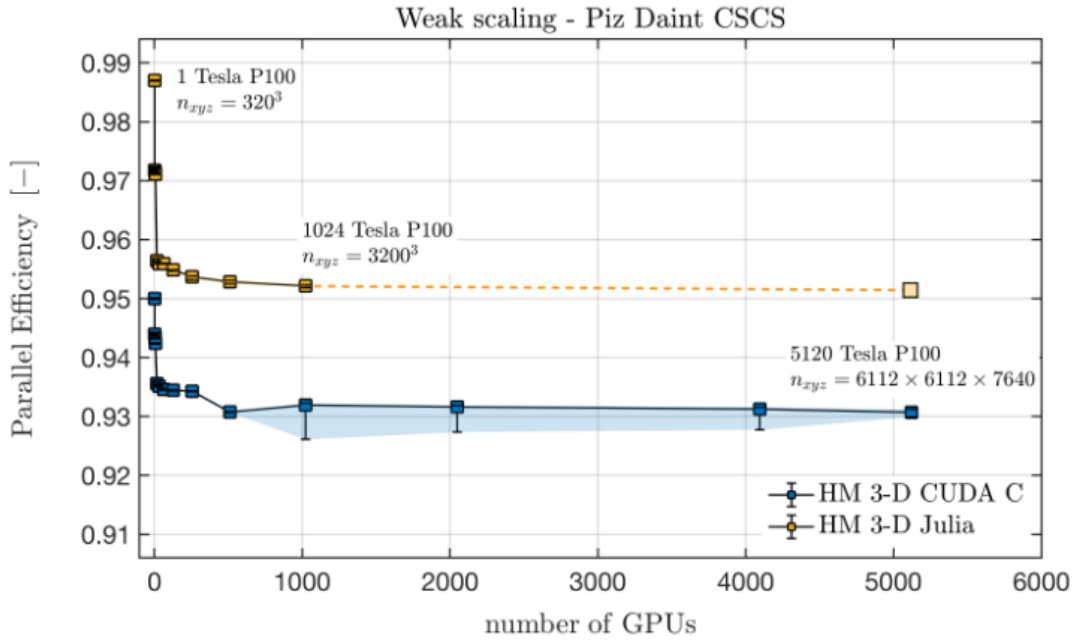


README BSD-3-Clause license

ImplicitGlobalGrid.jl

CI passing codecov unknown

ImplicitGlobalGrid is an outcome of a collaboration of the Swiss National Supercomputing Centre, ETH Zurich (Dr. Samuel Omlin) with Stanford University (Dr. Ludovic Räss) and the Swiss Geocomputing Centre (Prof. Yuri Podladchikov). It renders the distributed parallelization of stencil-based GPU and CPU applications on a regular staggered grid almost trivial and enables close to ideal weak scaling of real-world applications on thousands of GPUs [1, 2, 3]:



Multiphysics applications

$$\frac{1}{\rho}\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\frac{\partial v_k}{\partial x_k}$$

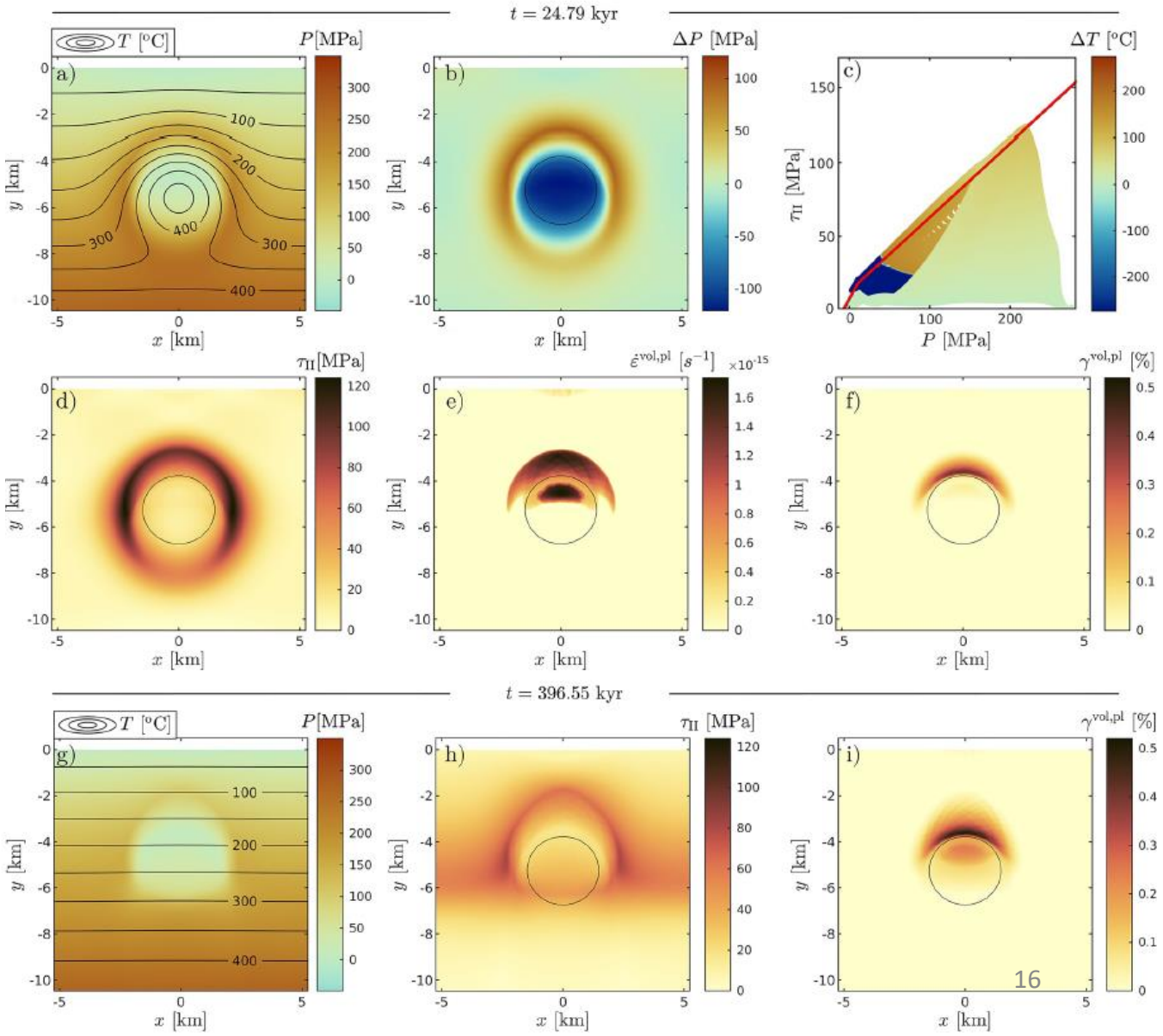
$$0 = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i$$

$$\rho C_P \frac{\mathrm{d}T}{\mathrm{d}t} = \alpha T \frac{\mathrm{d}P}{\mathrm{d}t} + \frac{\partial}{\partial x_j}\left(\lambda \frac{\partial T}{\partial x_j}\right) + \sigma_{ij}\left(\dot{\varepsilon}_{ij} - \dot{\varepsilon}_{ij}^{\mathrm{el}}\right) + Q_r$$

$$\frac{\partial v_k}{\partial x_k} = \alpha \frac{\mathrm{d}T}{\mathrm{d}t} - \beta \frac{\mathrm{d}P}{\mathrm{d}t} + \dot{\varepsilon}_{kk}^{\mathrm{vol,pl}}$$

$$\dot{\varepsilon}_{ij}^{\mathrm{dev}} = \frac{\tau_{ij}}{2\eta\left(\dot{\varepsilon}_{\mathrm{II}}^{\mathrm{dev,vis}}, T\right)} + \frac{1}{2\mu}\frac{\mathrm{d}\tau_{ij}}{\mathrm{d}t} + \dot{\varepsilon}_{ij}^{\mathrm{dev,pl}},$$

$$\sigma_{ij} = -P\delta_{ij} + \tau_{ij}$$



From Kiss, Moulas, Kaus & Spang, (2023), JGR

All works well until....

The previous approach is easy to implement and to be used in teaching and simple applications.

However, with increasing problem size this method takes very long to converge.

We need to accelerate the iteration procedure.

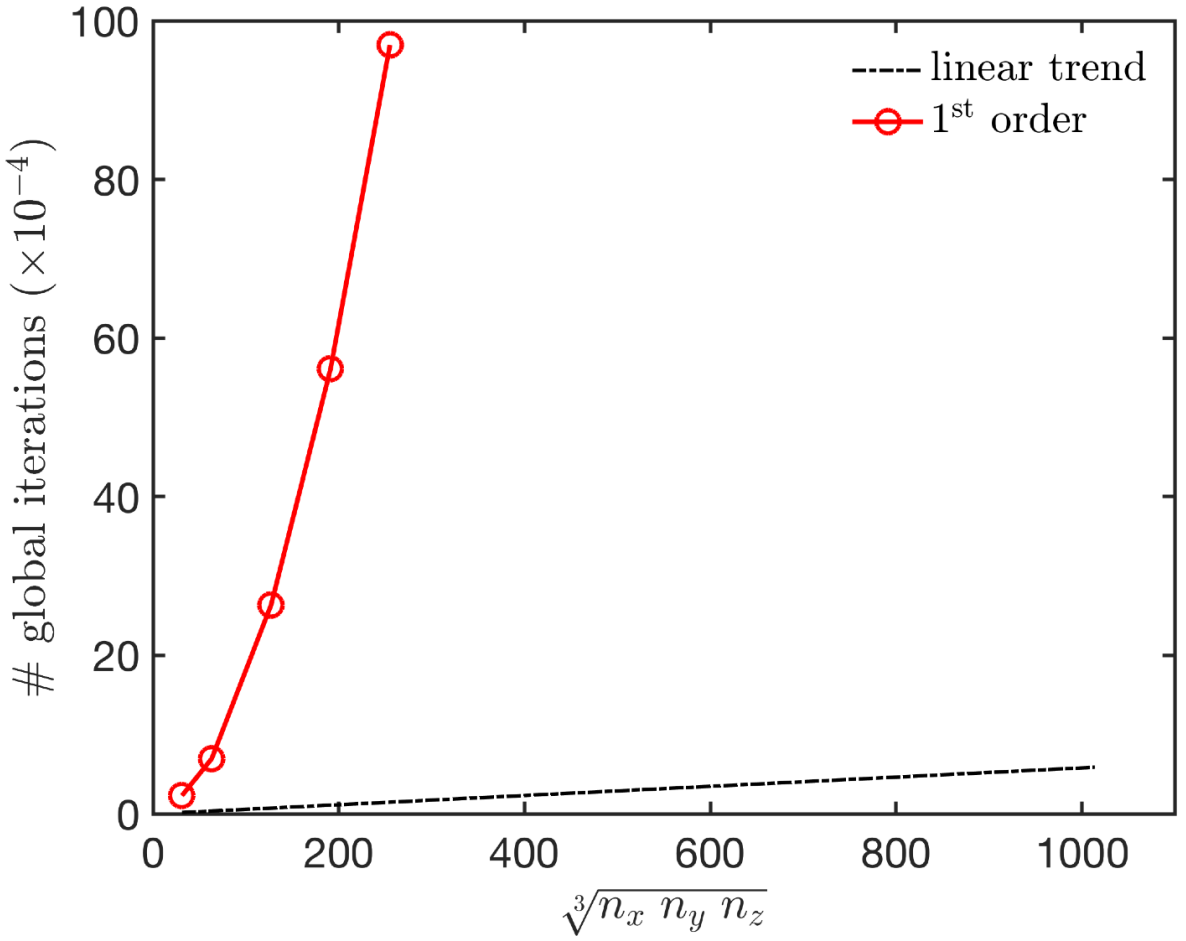


figure courtesy of Ludovic, Räss

Again, the inspiration comes from physics

Following Fourier (1822), the original system describing heat flow is:

$$J = -k\frac{\partial T}{\partial x} \quad , \quad \rho C\frac{dT}{dt} = -\frac{\partial J}{\partial x}$$

Following Cattaneo (1948)*, the system describing heat flow is:

$$J = -k\underbrace{\left(\frac{\partial T}{\partial x} - \alpha\left(\frac{\partial \dot{T}}{\partial x}\right)\right)}_{\text{Taylor expansion}} \quad , \quad \rho C\frac{dT}{dt} = -\frac{\partial J}{\partial x}$$

Or we could just write it as: $J = -k\Lambda\nabla T$ where $\Lambda = \left(1 - \alpha\frac{d}{dt}\right)$

*after Müller et al. (1998)

Again, the inspiration comes from physics

In that case, the expression for the Temperature equation becomes

$$\frac{dT}{dt} = D\Lambda \frac{\partial^2 T}{\partial x^2}$$

By assuming that the following approximation holds (small timescales):

$$\Lambda^{-1} = \left(1 - \alpha \frac{d}{dt}\right)^{-1} \approx \left(1 + \alpha \frac{d}{dt}\right)$$

we obtain:

$$\alpha \frac{d^2 T}{dt^2} + \frac{dT}{dt} = D \frac{\partial^2 T}{\partial x^2}$$

Introducing second-order iterations

By ignoring the advection (for now), the iteration scheme can be modified as (second order):

$$\alpha \frac{\partial^2 T}{\partial \tau^2} + \frac{\partial T}{\partial \tau} = F, \text{where} \quad F = \text{Residual}$$

The previous can be solved by introducing damping $\left(1 - \frac{\nu}{nx}\right)$

$$T^{k+1} = T^k + \Delta\tau \left(F^k + \left(1 - \frac{\nu}{nx}\right) F^{k-1} \right)$$

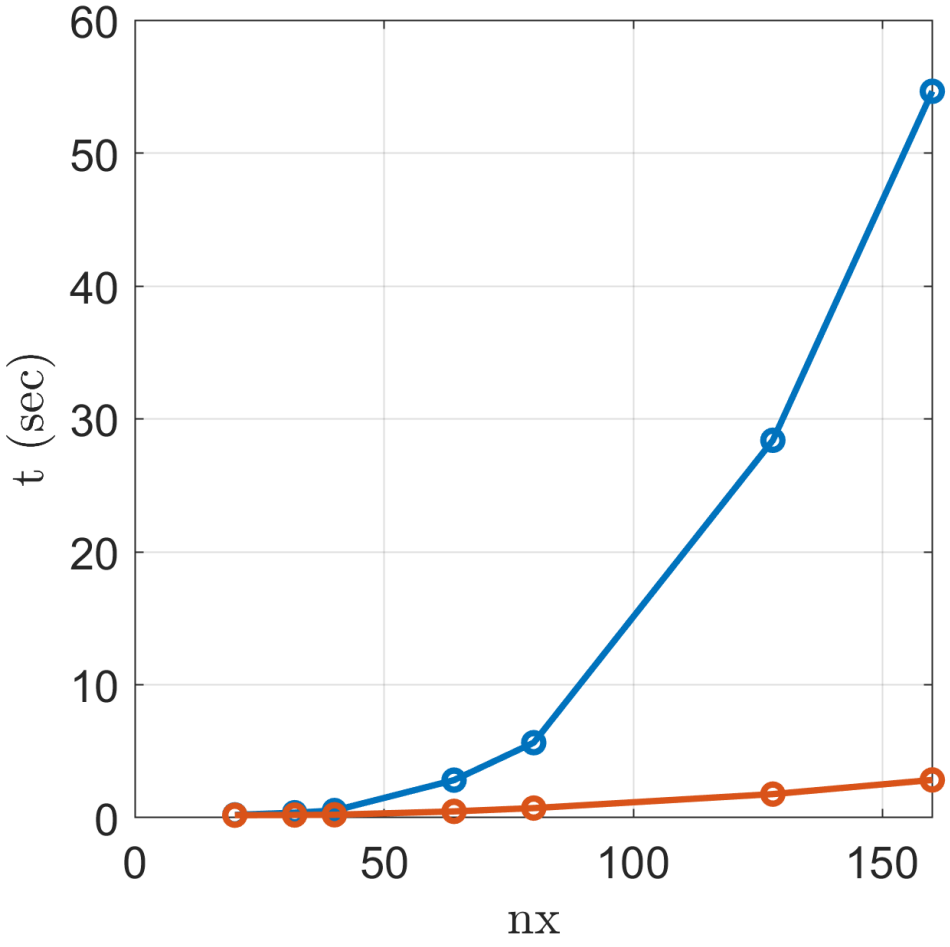
This step improves the convergence dramatically!

Introducing second-order iterations

The previous procedure allows very easy implementation of complex physical equations by modifying the residual (F)

$$T^{k+1} = T^k + \Delta\tau \left(F^k + \left(1 - \frac{\nu}{nx}\right) F^{k-1} \right)$$

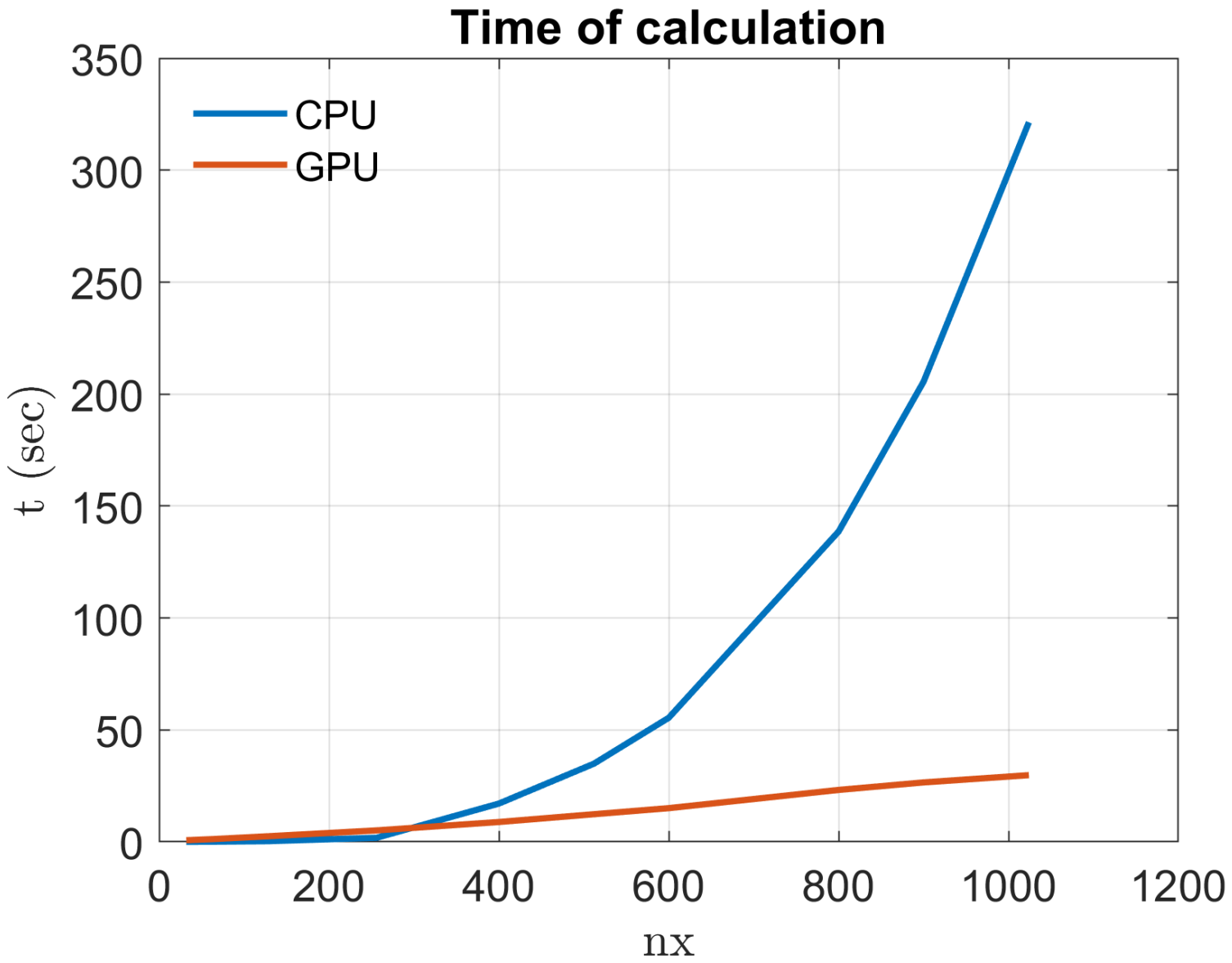
This approach also works for conservative schemes and very non-linear equations.



Performance of the previous MATLAB code (nx: degrees of freedom per dimension)

GPU acceleration

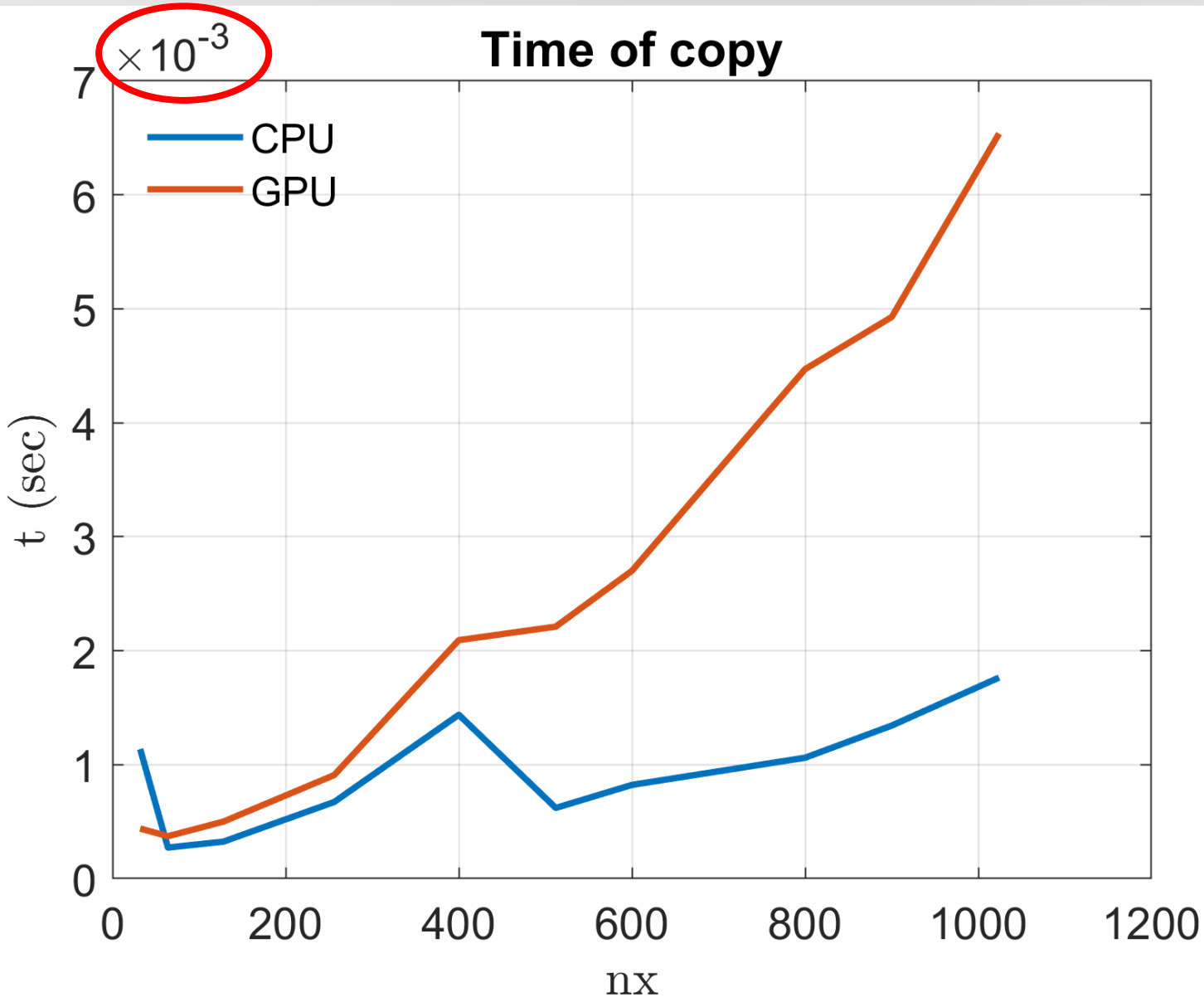
Using GPU arrays and minimizing the copy from the host (CPU) to device (GPU) improves the performance.



GPU acceleration

Using GPU arrays and minimizing the copy from the host (CPU) to device (GPU) improves the performance.

In general, copying to the device is slower. However, this is a small fraction of the calculation.

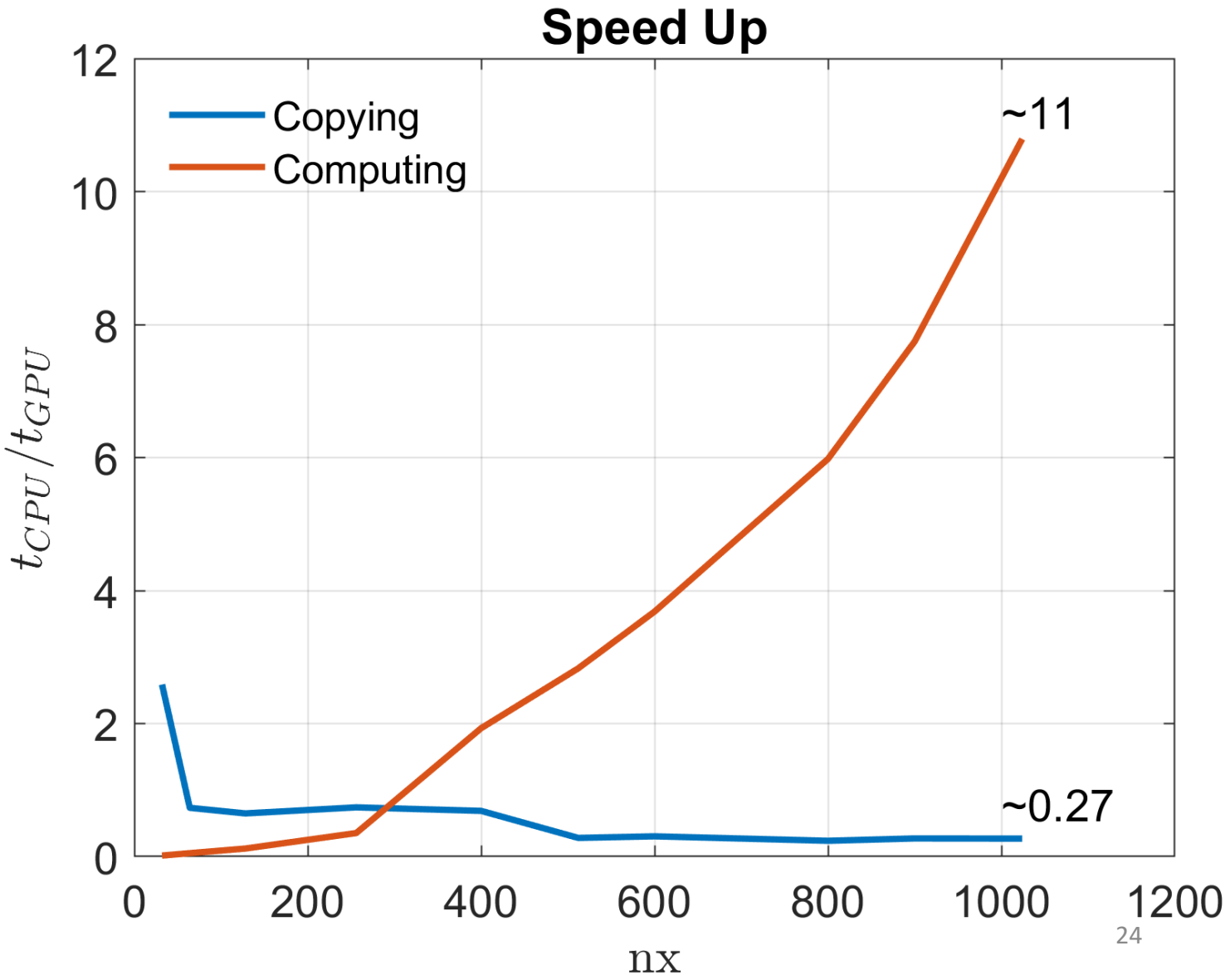


GPU acceleration

Using GPU arrays and minimizing the copy from the host (CPU) to device (GPU) improves the performance.

In general, copying to the device is slower. However, this is a small fraction of the calculation.

The performance increase in this case is about an order of magnitude.



References

- Chorin A (1967), A numerical method for solving incompressible viscous flow problems, Journal of Computational Physics, v.2(1), p.12-26

- Kiss D, Moulas E, Kaus B, Spang A, (2023), Decompression and fracturing caused by magmatically induced thermal stresses. Journal of Geophysical Research – Solid Earth, doi.org/10.1029/2022JB025341

- Müller I, et al. (1998), Rational Extended Thermodynamics, p.12-13