

# PTfS-CAM

Project: Modelling 2D steady-state heat equation

Part 1



# Overview

---

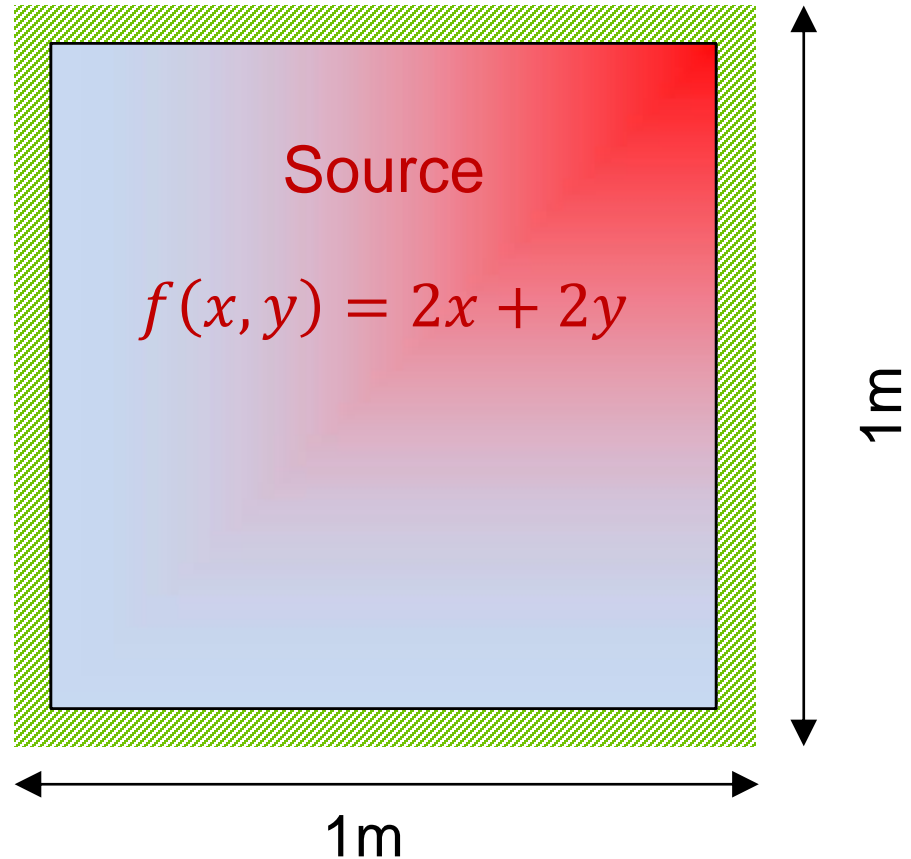
- Background
- What are your tasks
- Questions

# Scenario: Heat dissipation on a rectangular plate

Find steady state temperature distribution inside the plate!

Boundary

$$T(\varphi) = 0$$




# Steady-state heat equation

$$-k\Delta u = f \quad \forall (x, y) \in \Omega$$

$$u(x, y) = 0 \quad \forall (x, y) \in \partial\Omega$$

where  $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$

Assume (w.l.o.g.)  $k = 1$  (“diffusivity” or “conductivity”)


$$\Rightarrow -\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f$$

But this is in the continuous world... how to solve for  $u$  with a computer?



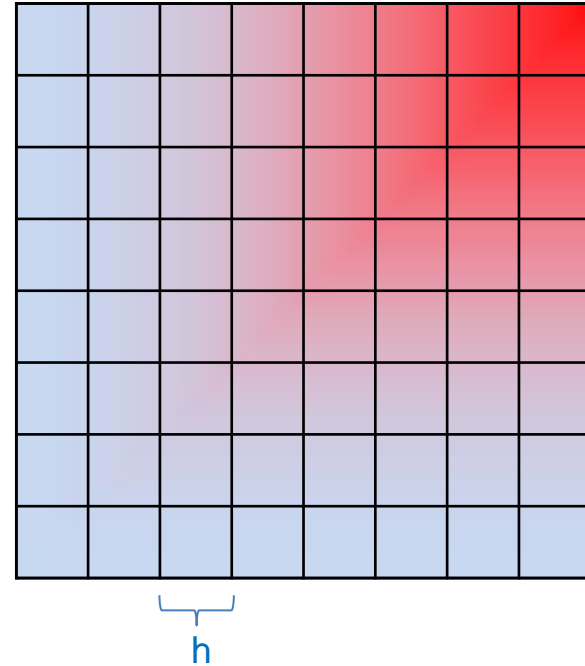
# Discretization

$$-\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f$$

Use Finite Difference Method (FDM) for discretization (2D)

$$\Rightarrow -\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)(x, y) \approx$$

$$\frac{1}{h^2} (4u(x, y) - u(x - h, y) - u(x + h, y) - u(x, y - h) - u(x, y + h))$$



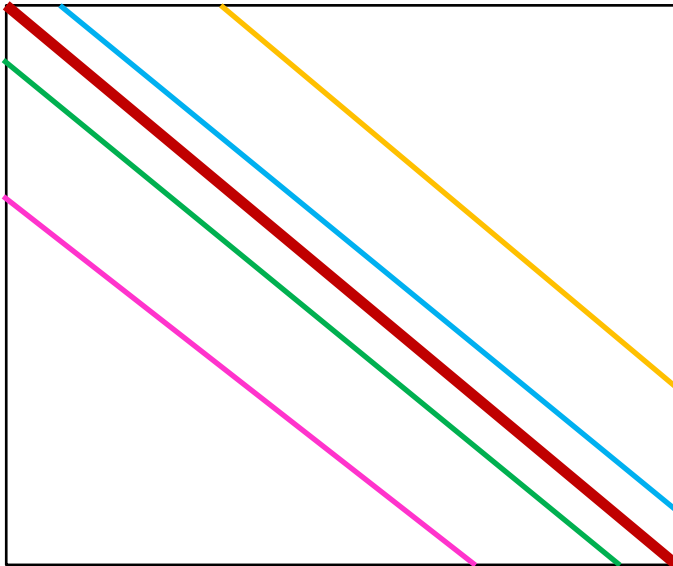
# Linear system of equations

Instead of a general point  $(x, y)$ , we have one equation for every particular choice of  $x$  and  $y$

$$\Rightarrow -\Delta u \approx \frac{1}{h^2} (4 u(x, y) - 1 u(x - h, y) - 1 u(x + h, y) - 1 u(x, y - h) - 1 u(x, y + h)) = f$$

$$\Rightarrow A u = f$$

$A =$



$u =$

$u(0,0)$
$u(0,h)$
$u(0,2h)$
$\vdots$
$u(h,0)$
$u(h,h)$
$u(h,2h)$
$\vdots$
$\vdots$
$\vdots$

$f =$

$f(0,0)$
$f(0,h)$
$f(0,2h)$
$\vdots$
$\vdots$
$\vdots$
$f(h,0)$
$f(h,h)$
$f(h,2h)$
$\vdots$
$\vdots$
$\vdots$

# Solving the linear system

Solve for  $u$  :  $A u = f$

1. Use Conjugate Gradient (CG)
2. Use Preconditioned Conjugate Gradient (PCG) with symmetric Gauss-Seidel preconditioning

Implementations of these algorithms are already given, and is not the focus of this project.

# Solving the linear system

## PCG example

“Iterative method”  
i.e. starting with  
some  $u = u_0$

Matrix-free implementation,  
i.e., stencil updates

initialization

```
r = f - A u
res_norm = <r, r>
z = P r0
alpha0 = <r, z>
p = z
```

iteration

```
while( (iter<niter) && (res_norm > tol*tol) )
    v = A p
    lambda = alpha0 / <v, p>
    u = u + lambda p
    r = r - lambda v
    ...
```



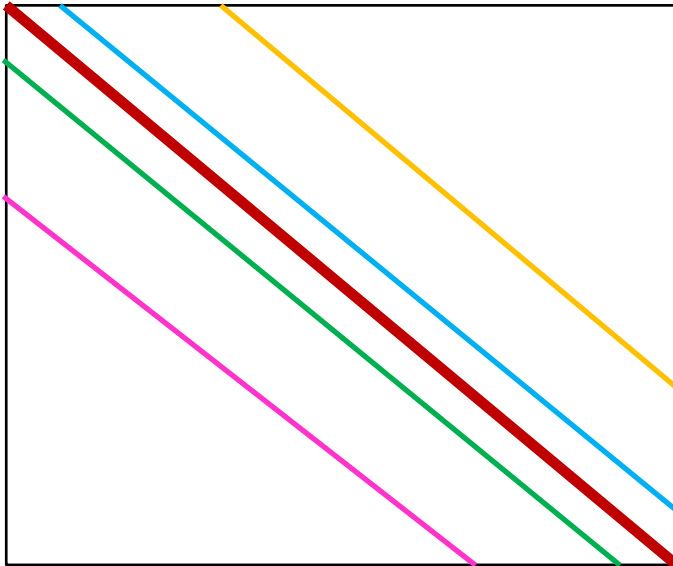
# Linear system of equations

$$\Rightarrow -\Delta u \approx \frac{1}{h^2} (4u(x, y) - 1u(x-h, y) - 1u(x+h, y) - 1u(x, y-h) - 1u(x, y+h)) = f$$

$$\Rightarrow A u = f$$

2d-5pt stencil

$A =$



$u =$

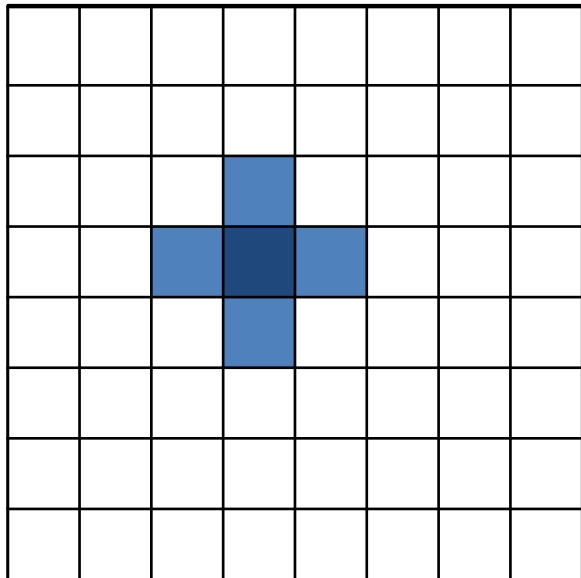
$u(0,0)$
$u(0,h)$
$u(0,2h)$
$\vdots$
$\vdots$
$\vdots$
$u(h,0)$
$u(h,h)$
$u(h,2h)$
$\vdots$
$\vdots$
$\vdots$

$f =$

$f(0,0)$
$f(0,h)$
$f(0,2h)$
$\vdots$
$\vdots$
$\vdots$
$f(h,0)$
$f(h,h)$
$f(h,2h)$
$\vdots$
$\vdots$
$\vdots$

# 2d-5pt stencil

Computing  $Au$  efficiently:  
How to take advantage of knowing the pattern of  $A$  ahead of time?



```
for j=1,jmax
  for k=1,kmax
    Au[j,k] = (1/h2)*( 4*u[j,k] - u[j-1,k]
                       -u[j+1,k]-u[j,k-1]-u[j,k+1] )
  enddo
enddo
```

# Solving the linear system

## PCG example

“Iterative method”  
i.e. starting with  
some  $u = u_0$

Matrix-free implementation,  
i.e., stencil updates

initialization

```
r = f - A u
res_norm = <r, r>
z = P r0
alpha0 = <r, z>
p = z
```

$$P = (D + U)^{-1}D(D + L)^{-1}$$

Think of solving this efficiently, since also not explicitly stored! See `PDE::GS_precon` defined in `PDE.cpp`

iteration

```
while( (iter<niter) && (res_norm > tol*tol) )
    v = A p
    lambda = alpha0 / <v,p>
    u = u + lambda p
    r = r - lambda v
    ...
```

# Your tasks

1. Clone the code from Github:  
`git clone https://github.com/RRZE-HPC/PTfS-CAM-Project.git`
2. Build the code using the given Makefile, i.e., just type `CXX=icpx make`
3. To switch on LIKWID measurement (for part 2) set the LIKWID flag to 'on', i.e.,  
`LIKWID=on CXX=icpx make`
4. Check for code correctness using the `test` executable: `./test`
5. To run the actual code use the `perf` executable:  
`./perf num_grids_y num_grids_x`
6. If all tests pass, **parallelize building blocks using OpenMP**. Always observe correctness!
7. Are there any possible performance optimizations that you could do in the CG and PCG solver implemented in `SolverClass::(P)CG (Solver.cpp)`? If so, implement them!

Do these now!  
Don't wait to get familiar with code.

# Walkthrough

---

1. No need for separate, manual compilation and linking
  - Everything can be done with a simple **make**
2. Important files
  - Grid.cpp (.h)
  - PDE.cpp (.h)
  - Solver.cpp (.h)
3. “./test” executable for correctness checking, “./perf” executable for... performance
  - After making optimizations, tests should always pass!
  - ./perf gives us performance w/ and w/out preconditioning
    - Also, routine specific timers (part 2). See Grid.cpp

# Things to take care

---

- Think to use **loop fusion** wherever necessary.
- For debugging please compile code as: `CXX=icpx make EXTRA_FLAGS=-DDEBUG`
- Sometimes it's useful for debugging to visualize your arrays. Use the function `writeGnuplotFile` and plot using `splot` in gnuplot if needed.
- Take particular care with parallelizing the Gauss-Seidel preconditioner.
- Use Fritz (Ice Lake) for getting your performance results.
- Fix clock frequency to 2.0 GHz
- Check if the measurements are reproducible. (Think of **pinning**, **scheduling**, and **clock frequency**).
- Request a dedicated node for measuring performance.

# Reminders

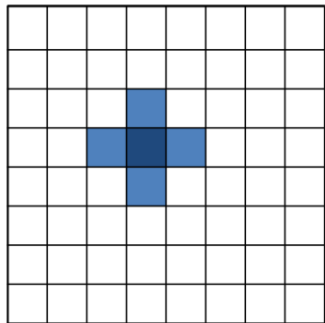
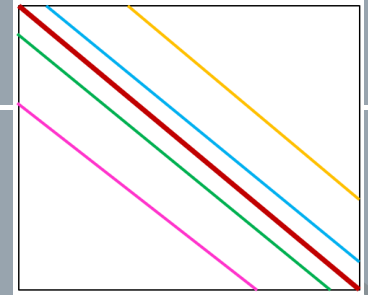
---

- Clone from git and build code sooner rather than later!
- Office hours
  - Thursdays 12:00 – 14:00
  - Blaues Hochhaus, Room 04.139
  - **Not on next two slots: May 30<sup>th</sup> or June 6<sup>th</sup>**

# Questions?

$$\Rightarrow -\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)(x, y) \approx$$

$$\frac{1}{h^2} (4u(x, y) - u(x - h, y) - u(x + h, y) - u(x, y - h) - u(x, y + h))$$



```

for j=1, jmax
  for k=1, kmax
    Au[j,k] = (1/h^2) * ( 4*u[j,k] - u[j-1,k]
                        -u[j+1,k]-u[j,k-1]-u[j,k+1] )
  enddo
enddo
    
```

