

Introduction to Project 4 (OpenMP and Nonlinear PDE)

Tim Holt, Olaf Schenk

Overview

- An overview of Project 4.
- First look at the code.
- Run the code and visualize output.

The HPC PDE Application

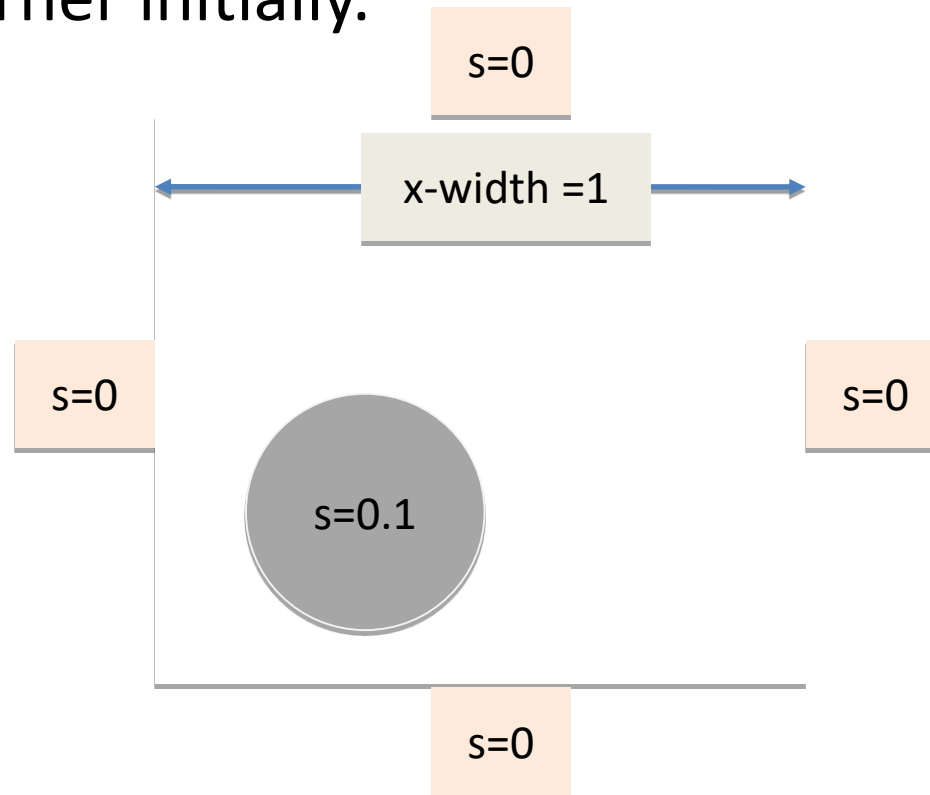
- The code solves a reaction diffusion equation known as Fischer's Equation

$$\frac{\partial s}{\partial t} = D \overbrace{\left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} \right)}^{\text{diffusion}} + \overbrace{Rs(1-s)}^{\text{reaction/growth}}$$

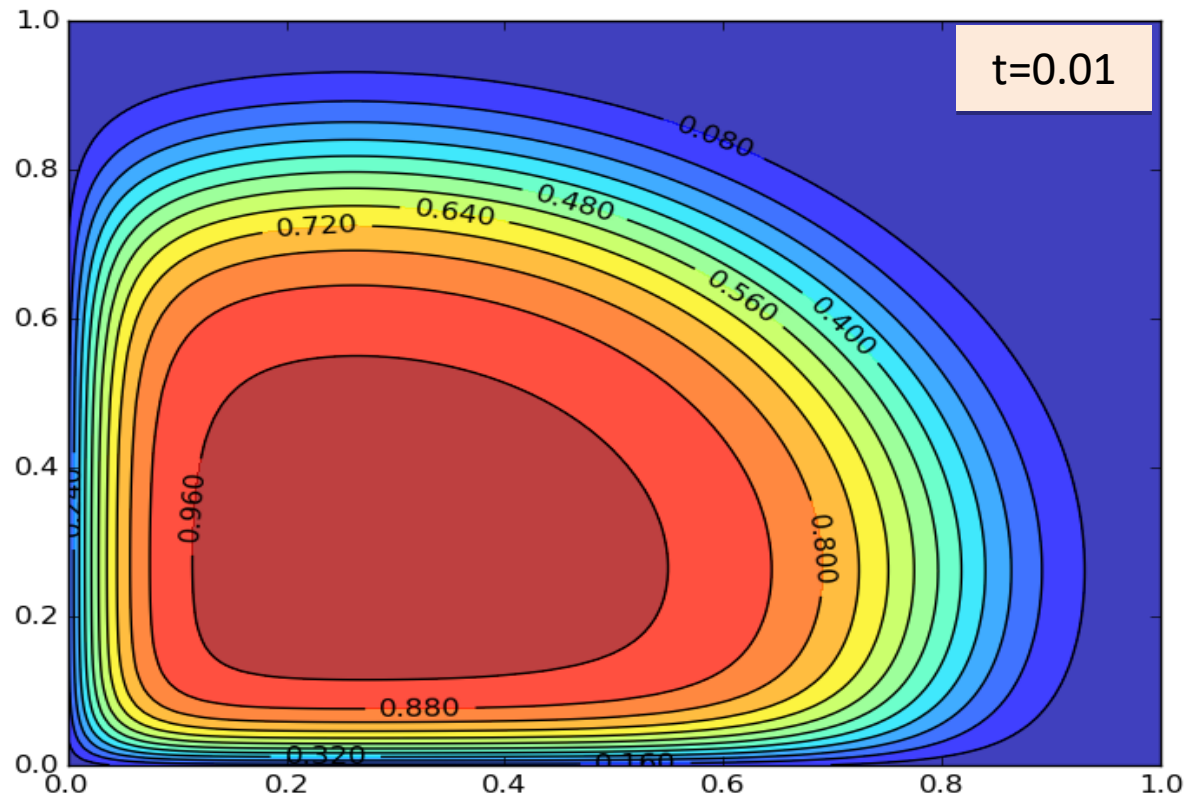
- Used to simulate travelling waves and simple population dynamics
 - The species s diffuses
 - And the population grows to a maximum of $s=1$

Initial and boundary conditions

- The domain is rectangular, with fixed value of $s=0$ on each boundary, and a circular region of $s=0.1$ in the lower left corner initially.



Time Evolution of Solution



Numerical Solution

The rectangular domain is discretized with a grid of dimension $(n+2)*(n+2)$ points

- A **second order finite difference** discretization gives the following approximation for the spatial derivatives

$$\left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2}\right)_{i,j} \approx \frac{1}{\Delta x^2} (-4s_{i,j} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1})$$

- A **first order finite difference** discretization is used for the approximation of the temporal derivative

$$\left(\frac{\partial s}{\partial t}\right)_{i,j}^k \approx \frac{1}{\Delta t} (s_{i,j}^k - s_{i,j}^{k-1})$$

Numerical Solution

- Putting these together one obtains

$$\frac{1}{\Delta t}(s_{i,j}^k - s_{i,j}^{k-1}) = \frac{D}{\Delta x^2}(-4s_{i,j}^k + s_{i-1,j}^k + s_{i+1,j}^k + s_{i,j-1}^k + s_{i,j+1}^k) + Rs_{i,j}^k(1 - s_{i,j}^k)$$

- Reformulate problem as

$$f_{i,j}^k := [-(4 + \alpha)s_{i,j} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{i,j}(1 - s_{i,j})]^k + \alpha s_{i,j}^{k-1} = 0$$

$$\alpha := \Delta x^2 / (D\Delta t)$$

$$\beta := R\Delta x^2 / D$$

Numerical Solution

- One nonlinear equation for each grid point
 - together they form a system of $N=n*n$ equations
 - Solve with Newton's method
 - $$\mathbf{y}^{l+1} = \mathbf{y}^l - [\mathbf{J}_f(\mathbf{y}^l)]^{-1} \mathbf{f}(\mathbf{y}^l)$$
- Each iteration of Newton's method has to solve a linear system
 - Solve with matrix-free Conjugate Gradient solver

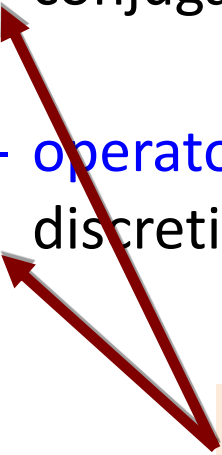
$$\begin{aligned} [\mathbf{J}_f(\mathbf{y}^l)] \delta \mathbf{y}^{l+1} &= \mathbf{f}(\mathbf{y}^l) \\ \mathbf{y}^{l+1} &= \mathbf{y}^l - \delta \mathbf{y}^{l+1} \end{aligned}$$

Numerical Solution

- Most of the code is already implemented
- The focus is on the parallelization of the code using OpenMP
- So let's look a little closer at each part of the code

Code Walkthrough

- There are three modules of interest
 - `main.cpp` : initialization and main time stepping loop
 - `linalg.cpp` : the BLAS level 1 (vector-vector) kernels and conjugate gradient solver
 - `operators.cpp` : the stencil operator for the finite difference discretization

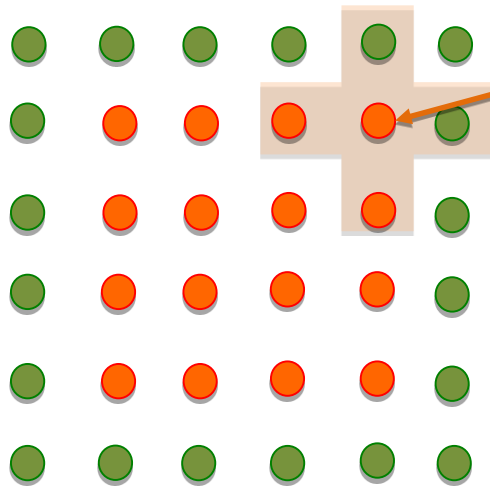


the vector-vector kernels and diffusion operator are the only kernels that have to be parallelized

Linear algebra: linalg.cpp

- This file defines simple kernels for operating on 1D vectors, including
 - dot product : $\mathbf{x}^T \mathbf{y}$: `hpc_dot()`
 - linear combination : $\mathbf{z} = \alpha * \mathbf{x} + \beta * \mathbf{y}$: `hpc_lcomb()`
- The kernels of interest start with `hpc_XXXXX()`
 - hpc == HPC Lab for CSE
- For each parallelization approach that we will see (OpenMP, and later on MPI), each of these kernels will have to be considered.

Stencil: Interior Grid Points



interior points have all neighbours available

interior points

```
for j=2:dim-1
```

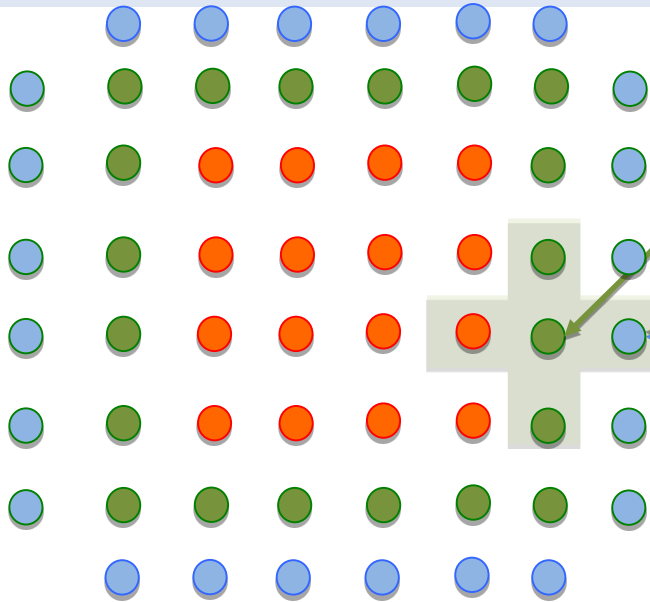
```
  for i=2:dim-1
```

$$f_{ij} = \left[- (4 + \alpha) s_{ij} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{ij} (1 - s_{ij}) \right]^{k+1} + \alpha s_{ij}^k = 0$$

```
  end
```

```
end
```

Stencil: Boundary Grid Points



boundary points are missing 1 or 2 neighbours

create 4 halo buffers, that hold "ghost" buffers
bndN, bndE, bndS, bndW

east boundary

i=dim

for j=2:dim-1

$$f_{ij} = [-(4 + \alpha) s_{ij} + s_{i-1,j} + \text{bndE}_i + s_{i,j-1} + s_{i,j+1} + \beta s_{ij} (1 - s_{ij})]^{k+1} + \alpha s_{ij}^k = 0$$

end

Testing the code

- Get the code, by checking it out from github

```
> git clone https://github.com/oschenk/HPC2020/Projects/Project4
```

- Compile and run

```
[user@login]$ salloc  
[user@icsnodeXX]$ make  
[user@icsnodeXX]$ export OMP_NUM_THREADS=1  
[user@icsnodeXX]$ ./main 128 100 0.01
```

It is possible to choose parameters that will make the simulation fail to converge! The code should tell you gracefully that it was unable to converge. Increasing the spatial resolution may require increasing the number of time steps.

Output

=====

version :: C++
mesh :: 128
time :: 100 t

The number of conjugate gradient iterations, which should always be constant for a given mesh size and time parameters. Can be used to check that changes to the code are still getting the correct result. There will be small variations due to the imprecise nature of floating point operations.

=====

step 1 required 4 iterations for residual 7.21951e-07

step 2 required 4 iterations for residual 7.9975e-07

...

step 99 required 12 iterations for residual 9.36586e-07

step 100 required 12 iterations for residual 9.44772e-07

time to solution

simulation took 1.58408 seconds

8127 conjugate gradient iterations, at rate of 5130.43 iters/second

920 newton iterations

Goodbye!

best way to compare different implementations

Thank you for your attention
and
have fun with the Project!