Introduction to the Summer School MiniApp

Ben Cumming, CSCS
July 14, 2019

Overview

In this session we will cover:

1. What is a miniapp?
2. The summer school miniapp overview.
3. First look at the code.
4. Compile, run and visualize the miniapp.

CSCS

ETHzürich

What is a HPC miniapp?

- Full HPC applications are complicated.
  - Difficult to model/understand performance behavior.
- A miniapp is a smaller code that aim to characterize performance of larger applications.
  - simpler to understand and benchmark than full applications.
  - can be used to test different hardware, languages and libraries.
  - good for learning new techniques!

The Summer School Miniapp

- Throughout the summer school we will be using a miniapp to reinforce the lessons.
  - During talks there will be small programming exercises to test out what you learn.
  - Then you will get the opportunity to apply the techniques to the miniapp.
- We will start with a serial version that has no parallel optimizations.
- By the end of the course we will have different versions, one for each technique.

ETH *zürich*

The Application

- The code solves **Fisher's equation**, a **reaction diffusion** model:

$$\frac{\partial s}{\partial t} = D \left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} \right) + Rs(1-s).$$

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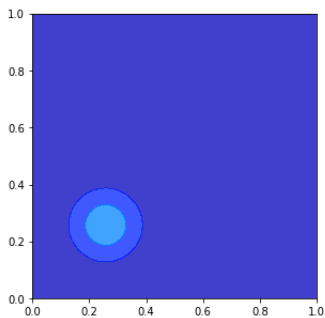 - The species reproduces to a maximum of $s = 1$.

CSCS

ETHzürich

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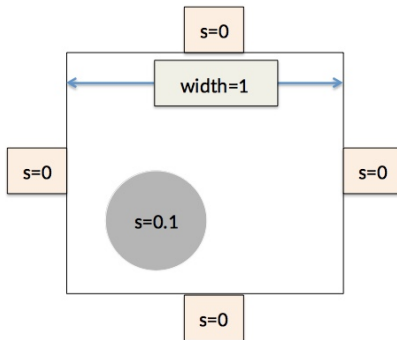
The domain is rectangular, with fixed value of $s = 0$ on each boundary, and a circular region of $s = 0.1$ in the lower left corner initially.
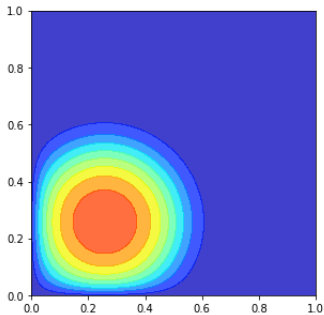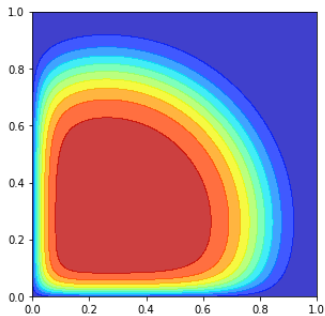
Time Evolution of the Solution

$$t = 0.001$$

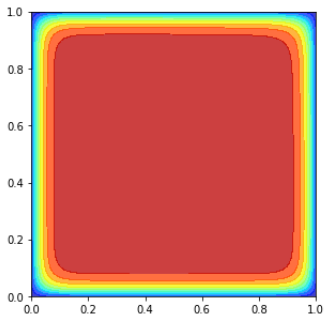# Time Evolution of the Solution

$$t = 0.005$$

Time Evolution of the Solution

$$t = 0.01$$

CSCS

ETH *zürich*

Time Evolution of the Solution

$$t = 0.02$$

Numerical Solution

- The rectangular domain is discretized with a grid of dimension $nx \times ny$ points.
- A finite volume discretization and method of lines gives the following ordinary differential equation for each grid point

$$\frac{ds_{i,j}}{dt} = \frac{D}{\Delta x^2} \left(-4s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1} \right) + Rs_{i,j}(1 - s_{i,j})$$

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

Numeric Solution

- One nonlinear equation for each grid point:
  - together they form a system of $N = nx \times ny$ equations
  - solve with Newton's method
- Each iteration of Newton's method solves a linear system
  - use a matrix-free Conjugate Gradient solver
- Solve the nonlinear system at each time step
  - requires in the order of between 5–10 conjugate gradient iterations

- Don't worry if you don't understand everything.
- We dont need a deep understanding of the mathematics or domain problem to optimize the code.
  - I often work on codes with little domain knowledge.
- The miniapp has a handful of kernels that can be parallelized.
- And care was taken when designing it to make parallelization as easy as possible.
- So lets look a little closer at each part of the code...

The Code

- The application is written in **C++**.
- It could be faster. . .
  - We avoid aggressive optimization to make the code easier to understand.
  - It is not a fine example of design.

CSCS

ETH zürich

Code Walkthrough

There are three main files of interest:

1. `main.cpp`: Initialization and time stepping code.
2. `linalg.cpp`: BLAS level-1 vector-vector operations and conjugate gradient solver.
3. `operators.cpp` The stencil kernel.

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

CSCS

ETH *zürich*

Linear Algebra: linalg.cpp

- This file defines simple kernels for operatiing on vectors, e.g.:
  - dot product $x^T y$ or $x \cdot y$: `ss_dot`.
  - linear combination $z = \alpha x + \beta y$: `ss_lcomb`.
- The kernels of interest are named `ss_xxxx`.
- Each will have to be parallelized using CUDA, MPI and OpenACC.
- The `ss_cg` function implements conjugate gradient using the vector and stencil operations.

Stencil operator: operators.cpp

This file has the function that applies the stencil operator:

```
for i=2:nx-1
    for j=2:ny-1
        S(i,j) = -(4. + alpha) * U(i,j)
                              + U(i-1,j) + U(i+1,j)
                              + U(i,j-1) + U(i,j+1)
                              + alpha * x_old(i,j)
                              + dxs * U(i,j) * (1.0 - U(i,j));
    end
end
```
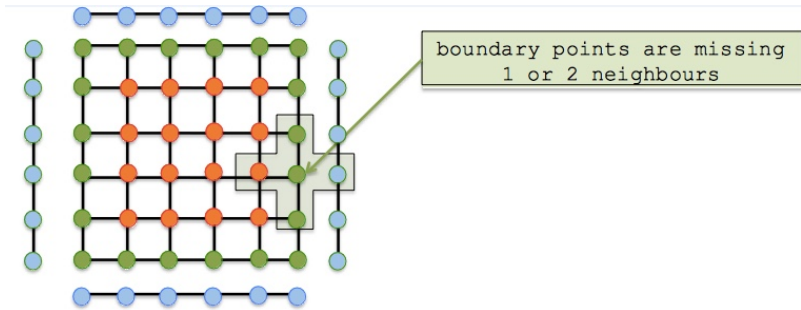
Stencil operator: Interior grid points

interior points have all neighbours available
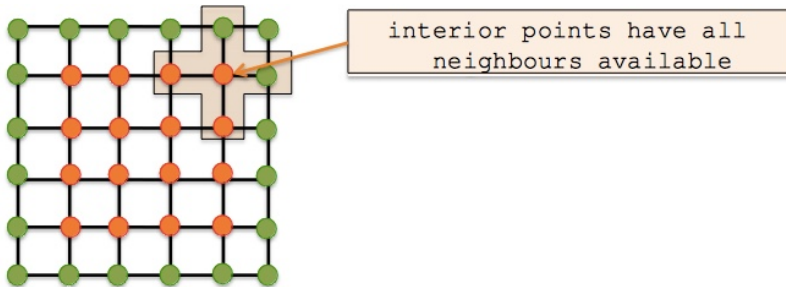
```
S(i,j) = -(4+alpha)*U(i,j)
       + U(i-1,j) + U(i+1,j) + U(i,j-1) + U(i,j+1) + ...
```

Stencil operator: Boundary grid points

boundary points are missing 1 or 2 neighbours

Points on the boundary need to use one or two external boundary points.

```
S(i,j) = -(4+alpha)*U(i,j)
       + U(i-1,j) + bndE[j] + U(i,j-1) + U(i,j+1) + ...
```

CSCS

ETHzürich

Testing the Code

Get the code and compile miniapp

```
> git clone git<at>github.com:eth-cscs/SummerSchool2019.git
> cd SummerSchool2019/miniapp/serial
> module load daint-gpu
> module swap PrgEnv-cray PrgEnv-gnu
> make
```

Run the miniapp

```
> srun -Cgpu --reservation=course ./main 128 128 100 0.01
==================================================
Welcome to mini-stencil!
version   :: C++ serial
mesh      :: 128 * 128 dx = 0.00787402
time      :: 128 time steps from 0 .. 0.01
iteration :: CG 200, Newton 50, tolerance 1e-06
==================================================
--------------------------------------------------
simulation took 1.07502 seconds
7439 conjugate gradient iterations, at rate of 6919.88 iters
    /second
959 newton iterations
--------------------------------------------------
```

Exercise: run the miniapp

- Run with 4 different resolutions
  - `128 128 100 0.01`
  - `256 256 200 0.01`
  - `512 512 200 0.01`
  - `1024 1024 400 0.01`
- For each case record:
  1. the number of CG iterations.
  2. the number of CG iterations per second.
- We will refer to these results when testing the MPI and GPU versions of the code.

Exercise: visualize the reults

- The application generates two data files with the final solution: `output.bin` and `output.bov`.
- There is a Python script that will show a contour plot of the solution.
- Now is a good time to test if X-windows is working.

```
> module load PyExtensions/2.7.15.1-CrayGNU-18.08
> python2 ../../scripts/plotting.py
```

Questions?