Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**

High Performance Computing **2017**

Instructor: Prof. Olaf Schenk TA: Juraj Kardos, Radim Janalik

Assignment 4

Due date: 7 November 2017, 13:30

Mini-app

The simple OpenMP exercises such as the pi computation are good for basic understanding the OpenMP constructs, but these are very far from practical applications. On the contrary, typical HPC applications are very complicated. We are going to use a mini-app in this assignment, which is a nice example of an application that solves something sophisticated, but the code is still short and easy to understand. The mini-app solves a reaction diffusion equation known as Fisher's Equation (equation 1). It can be used to simulate traveling waves and simple population dynamics.

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

We solve the equation on a rectangular domain with fixed value $s = 0$ on the boundaries while a circular region in the lower left corner is initialized to $s = 0.1$, as you can see in the figure 1.

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

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

Equation 2 can be expressed as the following nonlinear problem:

$$f_{i,j} = [-(4+\alpha)s_{i,j} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{i,j}(1 - s_{i,j})]^{k+1} + \alpha s^k_{i,j} = 0. \tag{3}$$

We have one nonlinear equation for each grid point. Together, the equations form a system of $N = nx * ny$ equations. We solve the system with Newton's method. Each iteration of the Newton's method has to solve a linear system. This is solved with a matrix-free Conjugate Gradient solver.

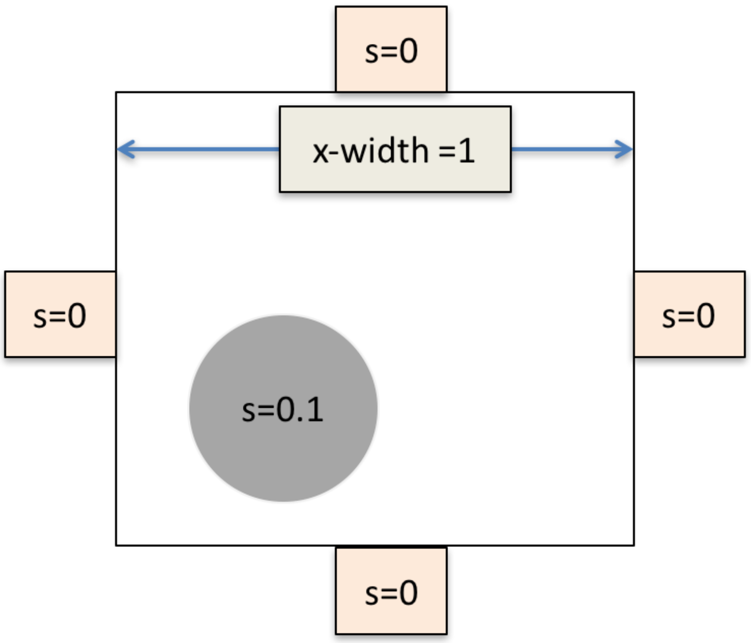
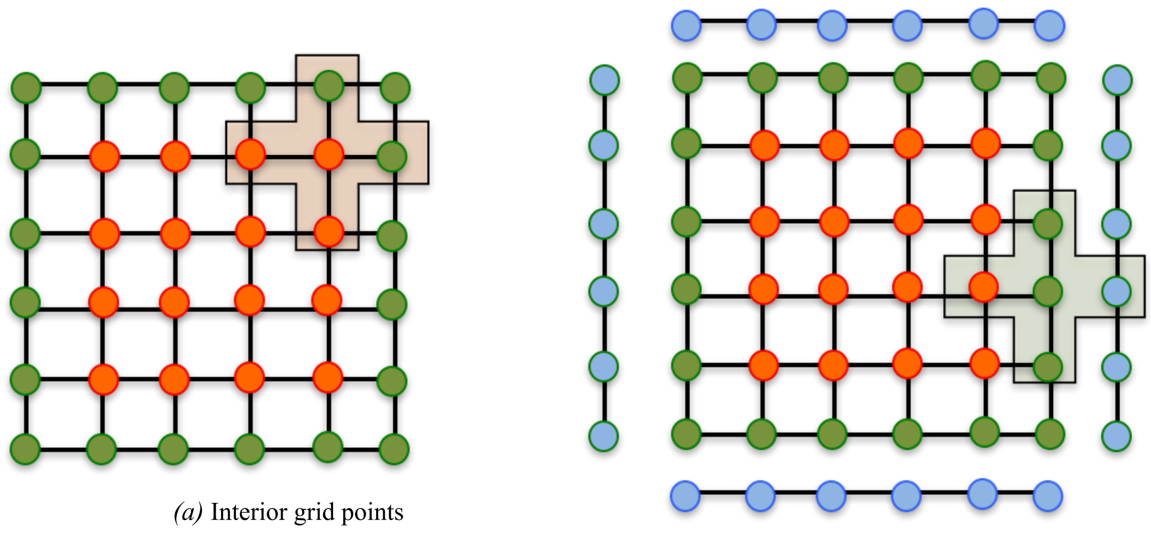


Figure 1. Mini-app: rectangular domain with zero boundary conditions and initial conditions $s = 0.1$ in the circle.

Code Walkthrough

There are three files of interest:

- `main.cpp`: initialization and main time stepping loop
- `linalg.cpp`: the BLAS level 1 (vector-vector) kernels and conjugate gradient solver
  - This file defines simple kernels for operating on 1D vectors, including
    - dot product: $x \cdot y$: `ss_dot()`
    - linear combination: $z = \text{alpha} * x + \text{beta} * y$: `ss_lcomb()`
    - $\ldots$
  - The kernels of interest start with `ss_xxxxx`
- `operators.cpp`: the stencil operator for the finite volume discretization
  - This file has a function/subroutine that defines the stencil operator


(a) Interior grid points

(b) Boundary grid points

Figure 2. Stencil: interior and boundary grid points

```
for $j = 2 : ydim - 1$ do
    for $i = 2 : xdim - 1$ do
        $f_{i,j} = [-(4+\alpha)s_{i,j} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{i,j}(1 - s_{i,j})]^{k+1} + \alpha s^{k}_{i,j} = 0$
    end
end
```

Algorithm 1: Stencil: interior grid points

```
$i = xdim$
for $j = 2 : ydim - 1$ do
    $f_{i,j} = [-(4+\alpha)s_{i,j} + s_{i-1,j} + s_{i+1,j} + s_{i,j-1} + s_{i,j+1} + \beta s_{i,j}(1 - s_{i,j})]^{k+1} + \alpha s^{k}_{i,j} = 0$
end
```

Algorithm 2: Stencil: boundary grid points

Compiling and running mini-app

Log-in to icsmaster with X-forwarding. Add switch `-X` or `-Y` to the `ssh` command.

```
$ ssh —Y icsmaster
```

Load gcc and python modules

```
$ module load gcc/6.1.0 python/2.7.12
```

Go to mini-app directory

```
$ cd HPC_2017/Assignment4/Miniapp_openmp/
```

Use makefile to compile the code

```
$ make
```

Connect to compute node

```
$ salloc
```

Run the app on the compute node with selected parameters, e.g. domain size $128 * 128$, 100 time steps and simulation time $0 - 0.01$ s

```
$ ./main 128 128 100 0.01
```

After you implement the first part of the assignment, the output of the mini-app should look like this:

```
========================================================================
                      Welcome to mini—stencil!
version   :: C++ Serial
mesh      :: 128 * 128 dx = 0.00787402
time      :: 100 time steps from 0 .. 0.01
iteration :: CG 200, Newton 50, tolerance 1e—06
========================================================================
________________________________________________________________________________
simulation took 1.06824 seconds
7703 conjugate gradient iterations, at rate of 7210.92 iters/second
866 newton iterations
________________________________________________________________________________
Goodbye!
```

1. Implementing linear algebra functions and the stencil operators *(40 Points)*

In the directory `HPC_2017/Assignment4/miniapp_openmp/` there is a serial version of the mini-app with some code missing. Your first task is to implement the missing code to get a working mini-app.

Open file `linalg.cpp` and implement the functions `ss_xxxxx()`. Follow the comments in the code. It should help you with the implementation.

Open file `operators.cpp` and implement the missing stencil kernels.

After completion of the above steps, the mini-app should produce correct results. Compare the number of conjugate gradient iterations and the newton iterations with the reference output above. If the numbers are about the same, you have probably implemented everything correctly. Now try to plot the solution with the script `plotting.py`. It should look like in the figure 3.

```
$ ./plotting.py
```

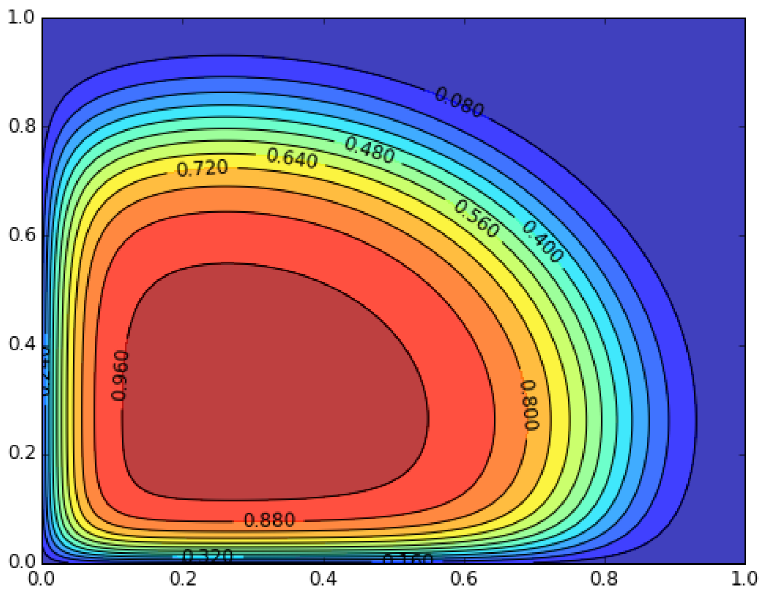


Figure 3. Output of the mini-app for domain size $128 * 128$, 100 time steps and simulation time $0 - 0.01$ s

2. Adding OpenMP to the mini-app *(60 Points)*

When the serial version of the mini-app is working we can add OpenMP directives. This allows you to use all cores on one compute node. On icsmaster there are two 10-cores CPUs on each compute node, so you can use 20 threads.

Hints

Before starting adding OpenMP parallelism, find two sets of input parameters that converge for the serial version

- You can use these parameters, or choose anything you like
  - `./main 128 128 100 0.01`
  - `./main 256 256 100 0.01`
- Write down the time to solution
  - You want this to get faster as you add OpenMP
  - But you might have to add write a few directives before things actually get faster
- Write down the number of conjugate gradient iterations
  - Use this to check after you add each directive that you are still getting the right answer
  - Remember that there will be some small variations because floating point operations are not commutative

Replace welcome message in main.cpp

Replace the welcome message in `main.cpp` with a message that informs the user about the following points:

- That this is the OpenMP version
- How many threads it is using

For example:

```
$ export OMP_NUM_THREADS=10
$ ./main 128 128 100 0.01
========================================================================
                      Welcome to mini—stencil!
version   :: C++ OpenMP
threads   :: 10
...
```

Linear Algebra

Open `linalg.cpp` and add OpenMP directives to all functions `ss_XXXX()` except for `ss_cg()`.

- Recompile frequently and run with 10 threads to check that you are getting the right answer

Once finished with this file, did your changes make any performance improvement?

- Compare the 128x128 and 256x256 results

The diffusion stencil

The final step is to parallelize the stencil operator in `operators.cpp`

- The nested for loop is an obvious target
- It covers $nx * ny$ grid points
- How about the boundary loops?

Testing

How does it scale at different resolutions?
Plot time to solution for these grid sizes for 1-10 threads.

- 64x64
- 128x128
- 256x256
- 512x512
- 1024x1024

Bonus

Can you implement first touch memory allocation?

- Requires adding just one OpenMP directive

Create the graphs in task 2 Testing for 1-20 threads.

- Do you see good scaling?
- Compare before and after implementing the first touch memory allocation.

Submission:

Submission: Submit the source code files in an archive file (tar, zip, etc) and show the TA the results. Furthermore, summarize your results and the observations for all exercises by writing an extended Latex summary. Use the Latex template from the webpage and upload the extended Latex summary as a PDF to iCorsi.