Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

High-Performance Computing Lab for CSE **2024**

Student: Benedict Armstrong Discussed with: Tristan Gabl

Solution for Project 3

Due date: Monday 15 April 2024, 23:59 (midnight)

1. Implementing the linear algebra functions and the stencil operators

1.1. Linalg functions

Implementing the eight linalg function outlined in `linalg.cpp` was relatively straighforward. Each followed a similar pattern of component wise iteration over the input array(s) and then performing the required operation. An example of the `copy` function is shown in Listing 1.

```
for (int i = 0; i < N; i++)
{
    y[i] = x[i];
}
```

Listing 1: Linalg copy function

1.2. Stencil operators

The next task was Implementing the stencil operator. Listing 2 shows how we calculate the value for each grid cell.

```
f(i, j) = -(4. + alpha) * s_new(i, j)
        + s_new(i - 1, j) + s_new(i + 1, j)
        + s_new(i, j - 1) + s_new(i, j + 1)
        + beta * s_new(i, j) * (1.0 - s_new(i, j))
        + alpha * s_old(i, j);
```

Listing 2: Stencil operator

1.3. Plotting the results

Finally we can plot the results with the following parameters:

- $nx = ny = 128$
- $nt = 100$
- $t = 0.005$

The output of the serial version is shown in Listing 3.
The results are shown in Figure 1.

```
$ ./build/main 128 100 0.005
================================================================================
                        Welcome to mini-stencil!
version   :: C++ Serial
mesh      :: 128 * 128 dx = 0.00787402
time      :: 100 time steps from 0 .. 0.005
iteration :: CG 300, Newton 50, tolerance 1e-06
================================================================================
--------------------------------------------------------------------------------
simulation took 0.15112 seconds
1514 conjugate gradient iterations, at rate of 10018.5 iters/second
300 newton iterations
--------------------------------------------------------------------------------
### 1, 128, 100, 1514, 300, 0.15112 ###
Goodbye!
```

Listing 3: Running the serial version of the mini-app

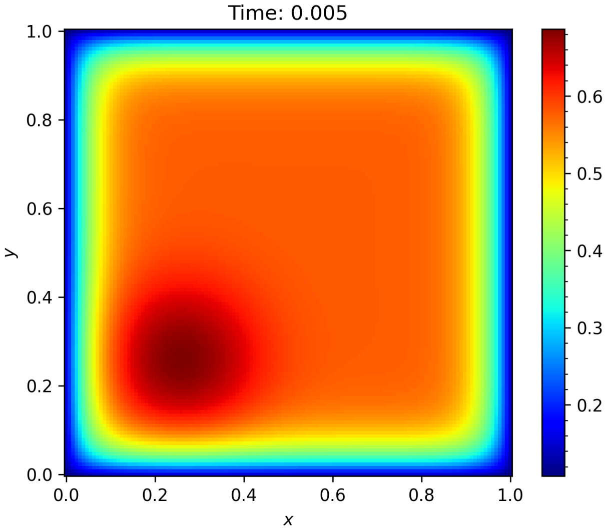

Figure 1: Results of the nonlinear PDE mini-app

2. Adding OpenMP to the nonlinear PDE mini-app

First I reconfigured the project welcome message. If `_OPENMP` is defined, we use `omp_get_max_threads()` to get the number of threads. The code welcome message is shown in Listing 4.

```
#ifdef _OPENMP
    std::cout << "version   :: C++ OpenMP" << std::endl;
    std::cout << "threads   :: " << threads << std::endl;
#else
    std::cout << "version   :: C++ Serial" << std::endl;
#endif
```

Listing 4: New OpenMP welcome message

Next I added parallelised versions of the linalg functions. For most of the functions, I simply added the *#pragma omp parallel for* directive before the loop. For the dot product and the norm functions, I used the `reduction` clause to ensure the correct result. An example of the copy function is shown in Listing 5.

```
#pragma omp parallel for shared(y, x)
    for (int i = 0; i < N; i++)
    {
        y[i] = x[i];
    }
```

Listing 5: OpenMP copy function

Finally, I added the `#pragma omp parallel for collapse(2)` directive to the stencil operator to parallelise the nested loops. The updated stencil operator is shown in Listing 6.

```
#pragma omp parallel for collapse(2) shared(s_new, s_old, f)
        for (int j = 1; j < jend; j++)
        {
            for (int i = 1; i < iend; i++)
            {
                f(i, j) = -(4. + alpha) * s_new(i, j)
                    + s_new(i - 1, j) + s_new(i + 1, j)
                    + s_new(i, j - 1) + s_new(i, j + 1)
                    + beta * s_new(i, j) * (1.0 - s_new(i, j))
                    + alpha * s_old(i, j);
            }
        }
```

Listing 6: OpenMP stencil operator