

# PDC Summer School: OpenMP Advanced Project

## About this exercise

The aim of this exercise is to give hands-on experience in parallelizing a larger program, measure parallel performance and gain experience in what to expect from modern multi-core architectures.

Your task is to parallelize a finite-volume solver for the two dimensional shallow water equations. Measure speed-up and if you have time, tune the code. You do not need to understand the numerics in order to solve this exercise (a short description is given in Appendix A). However, it assumes some prior experience with OpenMP, please refer to the lecture on shared memory programming if necessary.

## Algorithm

For this exercise we solve the shallow water equations on a square domain using a simple dimensional splitting approach. Updating volumes  $Q$  with numerical fluxes  $F$  and  $G$ , first in the  $x$  and then in the  $y$  direction, more easily expressed with the following pseudo-code:

```
for each time step do
  Apply boundary conditions
  for each Q do
    Calculate fluxes F in the x-direction
    Update volume Q with fluxes F
  end
  for each Q do
    Calculate fluxes G in the y-direction
    Update volumes Q with fluxes G
  end
end
```

In order to obtain good parallel speed-up with OpenMP, each sub-task assigned to a thread needs to be rather large. Since the nested loops contains a lot of numerical calculations the solver is a perfect candidate for OpenMP parallelization. But as you will see in this exercise, it is fairly difficult to obtain optimal speed-up on today's multi-core computers. However, it should be fairly easy to obtain some speed-up without too much effort. The difficult task is to make a good use of all the available cores.

Choose to work with either the given serial C/Fortran 90 code or, if you think you have time, write your own implementation (but do not waste time and energy). Compile the code by typing `make` and execute the program `shwater2d` with `srun` as described in the general documentation.

## 1. Parallelize the code

A serial version of the code is provided here: [shwater2d.c](#) or [shwater2d.f](#). Remember not to try parallelising everything. Add OpenMP statements to make it run in parallel and make sure the computed solution is correct. Some advices are provided below.

### Tasks and questions to be addressed

1. How should the work be distributed among threads?
2. Add OpenMP statements to make the code in parallel without affecting the correctness of the code.
3. What is the difference between

```
!$omp parallel do
  do i=1,n
    ...
!$omp end parallel do
!$omp parallel do
  do j=1,m
    ...
!$omp end parallel do
```

and

```
!$omp parallel
  !$omp do
    do i=1,n
      ...
    !$omp end do
  !$omp do
    do j=1,m
      ...
    !$omp end do
!$omp end parallel
```

*Hint: How are threads created/destroyed by OpenMP? How can it impact performance?*

## 2. Measure parallel performance

In this exercise, we explore parallel performance refers to the computational speed-up  $S_n = (\Delta T_1 / \Delta T_n)$ , where  $n$  is the number of threads.

### Tasks and questions to be addressed

1. Measure run time  $\Delta T_n$  for  $n = 1, 2, \dots, 24$  threads and calculate the speed-up.
2. Is it linear? If not, why?
3. Finally, is the obtained speed-up acceptable?
4. Try to increase the space discretization (M,N) and see if it affects the speed-up.

Recall from the OpenMP exercises that the number of threads is determined by an environment variable `OMP_NUM_THREADS`. One could change the variable or use the shell script provided in Appendix B.

## 3. Optimize the code

The given serial code is not optimal, why? If you have time, go ahead and try to make it faster. Try to decrease the serial run time. Once the serial performance is optimal, redo the speedup measurements and comment on the result.

For debugging purposes you might want to visualize the computed solution. Uncomment the line `save_vtk`. The result will be stored in `result.vtk`, which can be opened in ParaView, available on Tegner after module `add paraview`. Beware that the resulting file could be rather large, unless the space discretization (M,N) is decreased.

## A. About the Finite-Volume solver

In this exercise we solve the shallow water equations in two dimensions given by

$$\begin{aligned} h_t + (hu)_x + (hv)_y &= 0 \\ (hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x + (huv)_y &= 0 \\ (hv)_t + (huv)_x + \left(hv^2 + \frac{1}{2}gh^2\right)_y &= 0 \end{aligned} \tag{1}$$

where  $h$  is the depth and  $(u,v)$  are the velocity vectors. To solve the equations we use a dimensional splitting approach, i.e. reducing the two dimensional problem to a sequence of one-dimensional problems

$$\begin{aligned} Q_{ij}^* &= Q_{ij}^n - \frac{\Delta t}{\Delta x} \left( F_{i+1/2,j}^n - F_{i-1/2,j}^n \right) \\ Q_{ij}^{n+1} &= Q_{ij}^* - \frac{\Delta t}{\Delta y} \left( G_{i,j+1/2}^* - G_{i,j-1/2}^* \right) \end{aligned} \tag{2}$$

For this exercise we use the Lax-Friedrich's scheme, with numerical fluxes  $F, G$  defined as

$$\begin{aligned} F_{i-1/2,j}^n &= \frac{1}{2} (f(Q_{i-1,j}^n) + f(Q_{ij}^n)) - \frac{\Delta x}{2\Delta t} (Q_{ij}^n - Q_{i-1,j}^n) \\ G_{i,j-1/2}^* &= \frac{1}{2} (g(Q_{i,j-1}^*) + g(Q_{ij}^*)) - \frac{\Delta y}{2\Delta t} (Q_{ij}^* - Q_{i,j-1}^*) \end{aligned} \quad (3)$$

where  $f$  and  $g$  are the flux functions, derived from (1). For simplicity we use reflective boundary conditions, thus at the boundary

$$h = h \quad u = -u \quad v = -v$$

## B. Run scripts for changing OMP\_NUM\_THREADS

Bash:

```
#!/bin/bash

for n in `seq 1 1 16`; do
    OMP_NUM_THREADS=$n srun -n 1 ./a.out
done
```

C shell:

```
#!/bin/csh

foreach n (`seq 1 1 16`)
    env OMP_NUM_THREADS=$n srun -n 1 ./a.out
end
```