# OpenMP Advanced project

PDC Summer School
Introduction to High-Performance Computing

## 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.

In the exercise you will use a dual hexadeca-core shared memory Intel Xeon E5-2698v3 Haswell node. There will be several nodes available on the Cray for interactive use during the lab and each group will have access to a node of their own. Running the program should therefore give you realistic timings and speedup characteristics.

Your task is to parallelize a finite-volume solver for the two dimensional shallow water equations. Measure speedup and if you have time, tune the code. You don't 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 speedup with OpenMP, each sub task assigned to a thread needs to be rather larger. 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's fairly difficult to easily obtain optimal speedup on today's multi-core computers. However, it should be fairly easy to obtain some speedup 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 don't waste time and energy). Compile the code by typing `make` and execute the program `shwater2d` with `aprun` as described in the general instructions (`https://www.pdc.kth.se/support/documents/courses/summerschool.html`).

1. Parallelize the code.
   Start with the file `shwater2d.(c/f90)`, add OpenMP statements to make it run in parallel and make sure the computed solution is correct. Some advice are given below

   - How should the work be distributed among threads
   - Don't parallelize everything
   - What's the difference between

   ```
                               $!omp parallel
   $!omp parallel do           $!omp do
   do i=1,n                    do i=1,n
   ...                         ...
   $!omp end parallel do       $!omp end do
   $!omp parallel do           $!omp do
   do j=1,m                    do j=1,m
   ...                         ...
   $!omp end parallel do       $!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, parallel performance refers to the computational speedup $S_n = T_1/T_n$, using $n$ threads. Measure run time $T$ for $1, 2, \ldots, 16$ threads and calculate speedup. Is it linear? If not, why? Finally, is the obtained speedup acceptable? Also, try to increase the space discretization (`M,N`) and see if it affect the speedup.

   Recall from the OpenMP exercise that the number of threads is determined by an environment variable `OMP_NUM_THREADS`. One could change the variable or use the provided shell script 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 the lab computers after the module has been loaded `module add paraview`. Beware 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 vector. To solve the equations we use a dimensional splitting approach, reducing the two dimensional problem to a sequence of one-dimensional problems.

$$\begin{aligned} Q^*_{ij} &= Q^n_{ij} - \frac{\Delta t}{\Delta x}\left(F^n_{i+1/2,j} - F^n_{i-1/2,j}\right) \\ Q^{n+1}_{ij} &= Q^*_{ij} - \frac{\Delta t}{\Delta y}\left(G^*_{i,j+1/2} - G^*_{i,j-1/2}\right) \end{aligned} \tag{2}$$

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

$$\begin{aligned} F^n_{i-1/2,j} &= \frac{1}{2}\left(f(Q^n_{i-1,j}) + f(Q^n_{ij})\right) - \frac{\Delta x}{2\Delta t}\left(Q^n_{ij} - Q^n_{i-1,j}\right) \\ G^*_{i,j-1/2} &= \frac{1}{2}\left(g(Q^*_{i,j-1}) + g(Q^*_{ij})\right) - \frac{\Delta y}{2\Delta t}\left(Q^*_{ij} - Q^*_{i,j-1}\right) \end{aligned} \tag{3}$$

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

$$h = h \qquad u = -u \qquad v = -v$$

## B Run script

```
#!/bin/csh
foreach n (`seq 1 1 16`)
    env OMP_NUM_THREADS=$n aprun -n 1 -d $n ./a.out
end
```