Parallel and High Performance Computing : PROJECTS

Vincent Keller
Nicolas Richart
Vittoria Rezzonico

1 Introduction

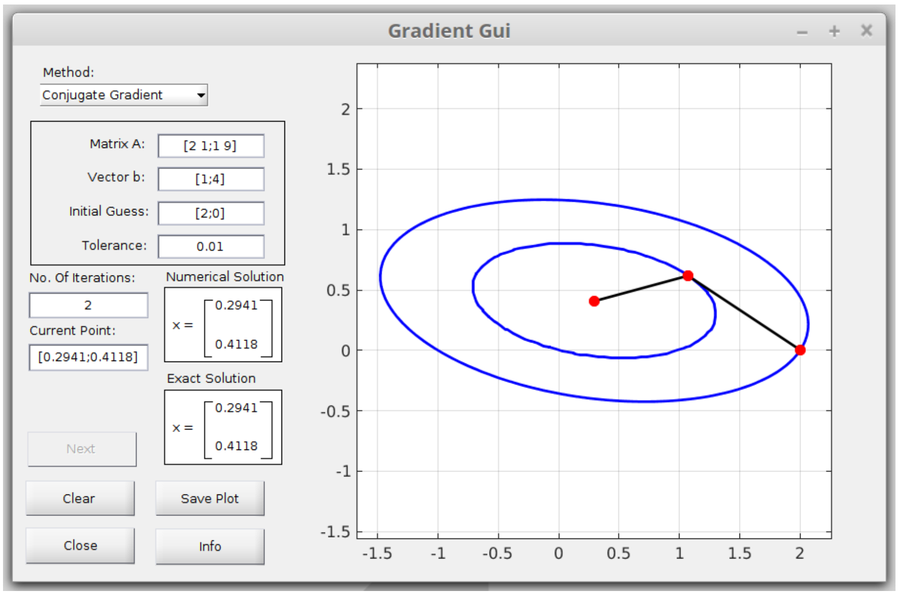
- 60 hours of personal and indivdual work
- We suggest 3 subjects
- **PhD students have the possibility to propose their own projects**
- MPI or CUDA (OpenMP/hybrid/MPI-IO are a plus)
- All the deliverables have to be handed in via the Moodle

Important dates :

Deliverable	**Due date**
Fixing topic	April 23, 2018
Theoretical Analysis	May 14, 2018
Final Report	June 8, 2018
Exam (15' pres + 5' Q&A)	July 5, 2018 8:15-18:00 and July 6, 2018 8:15-18:00

2 The projects

2.1 Conjugate Gradient Method (CG) for solving Ax = b


Method: Conjugate Gradient

Matrix A: [2 1;1 9]

Vector b: [1;4]

Initial Guess: [2;0]

Tolerance: 0.01

No. Of Iterations: 2

Current Point: [0.2941;0.4118]

Numerical Solution $x = \begin{bmatrix} 0.2941 \\ 0.4118 \end{bmatrix}$

Exact Solution $x = \begin{bmatrix} 0.2941 \\ 0.4118 \end{bmatrix}$

Figure 1: screenshot from http://m2matlabdb.ma.tum.de

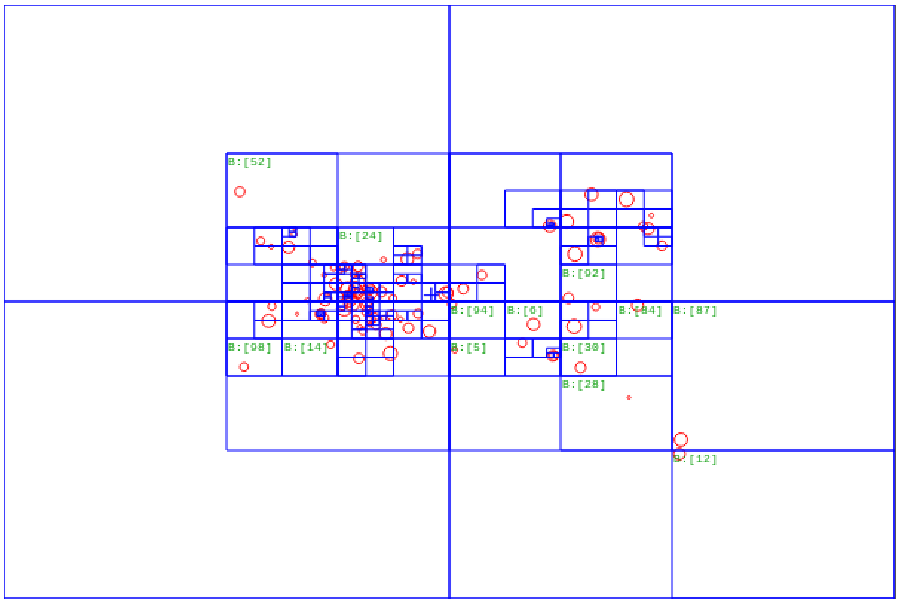
Problem description

- Solve $Ax = b$ with the "iterative" CG without pre-conditioning
- A is a full, real, symmetric, positive-definite matrix ($x^T Ax > 0.\ \forall x \in \Omega$)
- CG generates a solution x_k at iteration k with $x_{k+1} = x_k + \alpha_k p_k$ where
  - p_k is the conjugate vector at iteration k (or *search direction*)
  - α_k is the step length at iteration k
- $r_k = b - Ax_k$ is the residual
- p_k is always conjugated with the previous p_{k-1}. i.e. define β_k's such $\beta_k = \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}} = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$

Basic sequential algorithm

```
Data: A,b,$x_0$
Result: x such $Ax = b$
initialization:
usually $x_0 = 0$
$r_0 := b - Ax_0$
$p_0 = r_0$
$k := 0$
while $||r_{k+1}|| > 0$ do
    $\alpha_k := \frac{r_k^T r_k}{p_k^T A p_k}$
    $x_{k+1} := x_k + \alpha_k p_k$
    $r_{k+1} := r_k + \alpha_k A p_k$
    $\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
    $p_{k+1} := -r_{k+1} + \beta_k p_k$
    $k := k + 1$
end
```

2.2 N-Body Problem



Problem description

The n-body problem aims at simulating a dynamical system of particles under the influence of physical forces. We'll restrain on the gravity field applied on celestial bodies:

$$F_{ij} = \frac{Gm_i m_j(q_j - q_i)}{||q_j - q_i||}$$

where G is the gravitational constant, m_i and m_j the masses of the i-th and j-th bodies and q_i and q_j their positions.

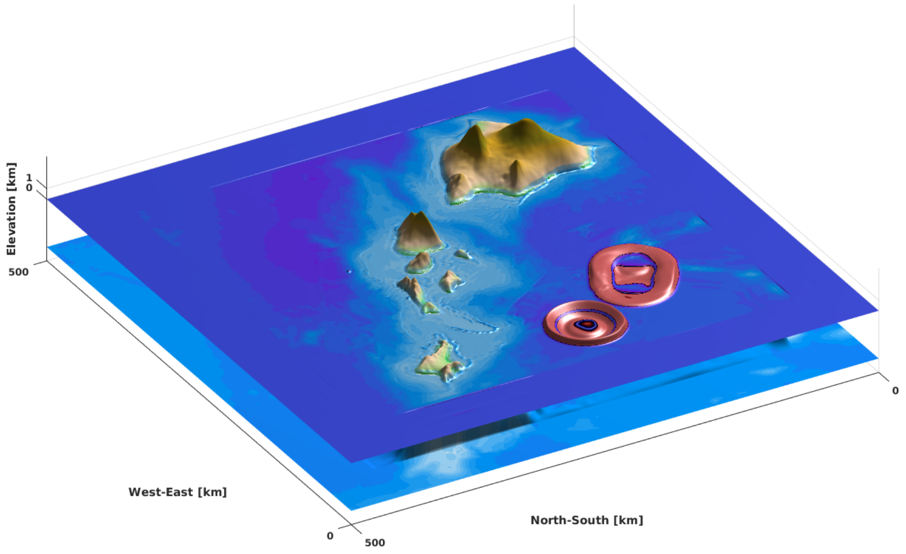
Possible algorithms

- **Particle-Particle Method** : this brute force algorithm will not be accepted
- Barnes-Hut
- Particle-based mesh methods
- Fast Multipole

Important remarks

- 2D or 3D
- non-uniform initial distribution must be tested to stress the application
- how to handle collisions and "lost bodies in the far space"

2.3 Shallow Water Wave Equation with a finite volume solver



An oceanographic researcher has written a simple predictive tool to simulate Tsunamis as they move over oceans and overflow land. Unfortunately, the Matlab code is prohibitively slow! Running the high-resolution simulation takes several hours on a workstation. You are asked to write new code that can simulate the Tsunami much faster. Use the new techniques that you have learned in MATH454 and the computational resources available at SCITAS so to help the researcher. The Matlab code and data is available in the project folder, and below you will find a short introduction to the model and the mathematics.

Introducing the Model

The shallow water wave equation is a non-linear hyperbolic system of coupled partial differential equations often used to model various wave phenomenons. Simulation of ocean waves, river flows, hydraulic engineering and atmospheric modeling are among the many areas of application. The researcher has used the two dimensional version of the equations along with a right hand side source term as his model

$$\begin{cases} h_t + (hu)_x + (hv)_y = 0 \\ (hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x + (huv)_y = -ghz_x \\ (hv)_t + (huv)_x + \left(hv^2 + \frac{1}{2}gh^2\right)_y = -ghz_y \end{cases}$$

In the above, $h := h(x,y,t)$ denotes water height, $u := u(x,y,t)$ and $v := v(x,y,t)$ water velocity in x and y direction respectively, $z := z(x,y)$ topography and $g = 9.82\mathrm{m/s}^2$ the gravitational constant.

Introducing the Numerical Scheme

Finite volume schemes are a popular approach for computing an approximate solution to hyperbolic equations as the underlying physics is represented in a natural way. Let $I_{i,j} = \left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right] \times \left[y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}\right]$ define a structured rectangular uniform mesh. In a finite-volume scheme, one seeks to find the cell average $\bar{q}_{ij}(t_n)$ that approximates $q(x_i, y_j, t_n)$ at every cell $I_{i,j}$ for a given time-step t_n in the sense

$$\bar{q}_{ij}(t_n) = \frac{1}{\Delta x \Delta y} \int_{I_{i,j}} q(x,y,t_n)\, dydx$$

The researcher has used the Lax–Friedrichs method to discretize the problem. The method is explicit meaning that the solution $\bar{q}_{ij}^{n+1}$ for all i,j at timestep t_{n+1} may be computed directly from the solution at the previous timestep $\bar{q}^n_{ij}$ without solving any system of equations using the following three equations

$$\begin{aligned}
\bar{h}^{n+1}_{i,j} =& \frac{1}{4}\left[\bar{h}^n_{i,j-1} + \bar{h}^n_{i,j+1} + \bar{h}^n_{i-1,j} + \bar{h}^n_{i+1,j}\right] \\
&+ \frac{\Delta t^n}{2\Delta x}\left[\bar{hu}^n_{i,j-1} - \bar{hu}^n_{i,j+1} + \bar{hv}^n_{i-1,j} - \bar{hv}^n_{i+1,j}\right] \\
\bar{hu}^{n+1}_{i,j} =& \frac{1}{4}\left[\bar{hu}^n_{i,j-1} + \bar{hu}^n_{i,j+1} + \bar{hu}^n_{i-1,j} + \bar{hu}^n_{i+1,j}\right] - \Delta t^n g \bar{h}_{i,j}^{n+1} z_y \\
&+ \frac{\Delta t^n}{2\Delta x}\left[\frac{\left(\bar{hu}^n_{i,j-1}\right)^2}{\bar{h}^n_{i,j-1}} + \frac{1}{2}g\left(\bar{h}^n_{i,j-1}\right)^2 - \frac{\left(\bar{hu}^n_{i,j+1}\right)^2}{\bar{h}^n_{i,j+1}} - \frac{1}{2}g\left(\bar{h}^n_{i,j+1}\right)^2\right] \\
&+ \frac{\Delta t^n}{2\Delta x}\left[\frac{\bar{hu}^n_{i-1,j}\bar{hv}^n_{i-1,j}}{\bar{h}^n_{i-1,j}} - \frac{\bar{hu}^n_{i+1,j}\bar{hv}^n_{i+1,j}}{\bar{h}^n_{i+1,j}}\right] \\
\bar{hv}^{n+1}_{i,j} =& \frac{1}{4}\left[\bar{hv}^n_{i,j-1} + \bar{hv}^n_{i,j+1} + \bar{hv}^n_{i-1,j} + \bar{hv}^n_{i+1,j}\right] - \Delta t^n g \bar{h}_{i,j}^{n+1} z_y \\
&+ \frac{\Delta t^n}{2\Delta x}\left[\frac{\bar{hu}^n_{i,j-1}\bar{hv}^n_{i,j-1}}{\bar{h}^n_{i,j-1}} - \frac{\bar{hu}^n_{i,j+1}\bar{hv}^n_{i,j+1}}{\bar{h}^n_{i,j+1}}\right] \\
&+ \frac{\Delta t^n}{2\Delta x}\left[\frac{\left(\bar{hv}^n_{i-1,j}\right)^2}{\bar{h}^n_{i-1,j}} + \frac{1}{2}g\left(\bar{h}^n_{i-1,j}\right)^2 - \frac{\left(\bar{hv}^n_{i+1,j}\right)^2}{\bar{h}^n_{i+1,j}} - \frac{1}{2}g\left(\bar{h}^n_{i+1,j}\right)^2\right]
\end{aligned}$$

Note the index n on Δt above. After each simulation time-step, we need to compute a new time-step length! This is needed to make sure that the CFL condition is satisfied. The time-step Δt^n

$$\Delta t^n = \min_{i,j} \frac{\Delta x}{\sqrt{2}\nu^n_{i,j}}$$

is found computing first ν^n using

$$\nu^n_{i,j} = \sqrt{\max\left(\left|\frac{\bar{hu}^n_{i,j}}{\bar{h}^n_{i,j}} + g\bar{h}^n_{i,j}\right|, \left|\frac{\bar{hu}^n_{i,j}}{\bar{h}^n_{i,j}} - g\bar{h}^n_{i,j}\right|\right)^2 + \max\left(\left|\frac{\bar{hv}^n_{i,j}}{\bar{h}^n_{i,j}} + g\bar{h}^n_{i,j}\right|, \left|\frac{\bar{hv}^n_{i,j}}{\bar{h}^n_{i,j}} - g\bar{h}^n_{i,j}\right|\right)^2}$$

After computing a new time-step $\bar{q}^{n+1}_{ij}$ from $\bar{q}^n_{ij}$, cells that are below a certain water threshold are made inactive

$$\bar{h}^{n+1}_{i,j} \leq 0 \rightarrow \bar{h}^{n+1}_{i,j} = 10^{-5} \qquad \bar{h}^{n+1}_{i,j} \leq 10^{-4} \rightarrow \bar{hv}^{n+1}_{i,j} = 0,\ \bar{hv}^{n+1}_{i,j} = 0$$

The Matlab Code

In the project folder you will find several files.

- Matlab files.
  - compute.m : A Matlab script to run the simulation
  - visualize.m : A Matlab script to visualize the result
- Data files with initial condition.
  - Data_nx2001_500km_T0.2_h.bin: Water height
  - Data_nx2001_500km_T0.2_hu.bin: Flow-rate x direction
  - Data_nx2001_500km_T0.2_hv.bin: Flow-rate y direction
  - Data_nx2001_500km_T0.2_Zdx.dat: Bathemetry slope in x direction
  - Data_nx2001_500km_T0.2_Zdy.dat: Bathemetry slope in y direction

The data files are also given in higher resolution, 4001x4001 and 8001x8001. In order to use the code, first execute compute.m to solve the problem. Then execute visualize.m to visualize the solution.

Study the Matlab code in compute.m and translate it to c++. Can your sequential code beat Matlab's speed? How much faster can you make it using the parallel computing techniques you've learned in MATH454?