TAF MCE: Numerical methods

Mini Project: Parallel computing in Julia

Lucas Drumetz
lucas.drumetz@imt-atlantique.fr

Outline

Outline

Introduction

What is parallelization?

Using multiple physical cores of the processor to run several tasks at the same time, to obtain a gain in running time

Embarrassingly vs non embarrassingly parallel problems:

An embarrassingly parallel problem is a problem in which several independent tasks have to be carried out before aggregating the results. Examples:

- Generating N realizations of a random variable
- Running a for loop doing the same thing over several data samples
- Applying a convolution mask over each pixel of an image ($\rightarrow$ GPUs in deep learning)

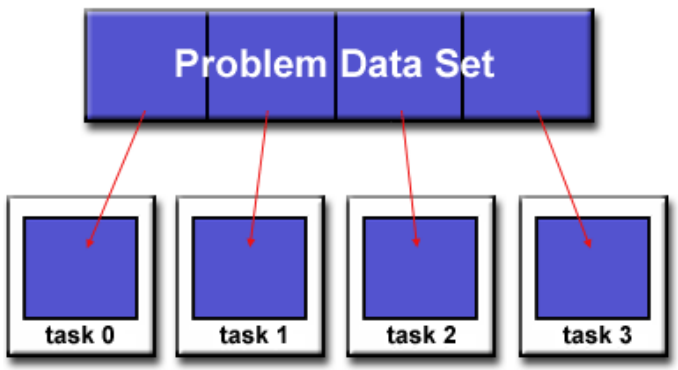
A non embarrassingly parallel problem is a problem where tasks run in parallel have to share information.

- Computing the solution of a PDE over a 2D grid
- Denoising an image using spatial information

Introduction (cont'd)

Here we will focus on embarrassingly parallel problems only.

Typically, if we have access to K physical cores, we will work with K processes, and expect at best a factor K of improvement.



The word "overhead" denotes the computation time spent communicating between processes. If there is too much overhead the parallel version might not be much faster (or even slower) than the serial version!

Outline

Parallel computing in Julia

Julia is particularly adapted to parallel computing (more than Python – for parallelization on CPUs at least).

Julia 1.2 is sufficient for this project but Julia 1.5 is the latest version and there are new parallel computing functionalities. Check it out...

Useful Notions (see notebook)

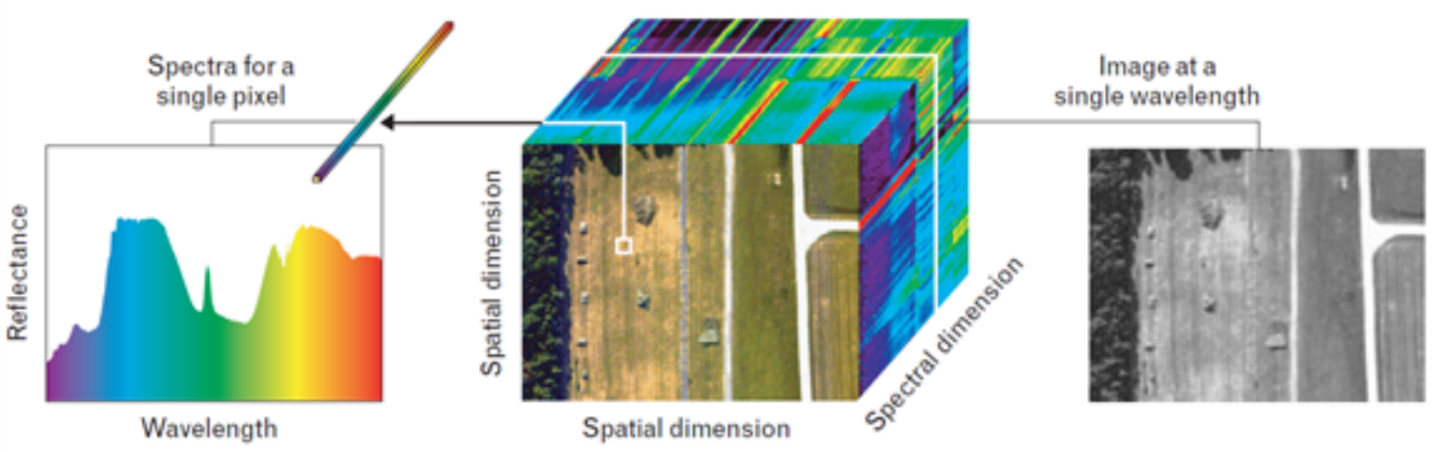
- How to handle processes and workers
- Remote calls and fetch commands
- Make functions accessible to all workers
- The map and pmap function
- Shared and Distributed arrays
- Synchronous vs asynchronous calls

examples: loop parallelization

Outline

Outline

Hyperspectral imaging

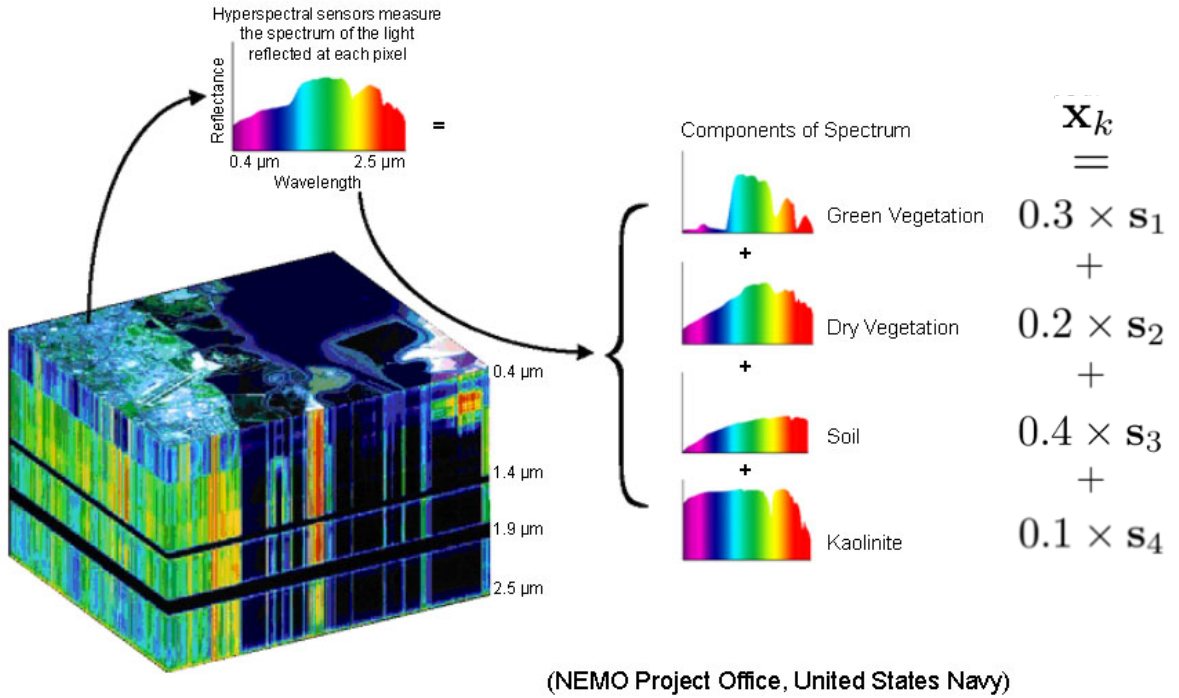


Hyperspectral imaging concept.

- Each pixel: full reflectance spectrum for many contiguous and narrow wavelengths (visible and near-IR).
- HSI: also a collection of gray-level reflectance images for each wavelength.
- The spectrum characterizes a material, but because of the limited spatial resolution, mixed pixels are present.

Applications in: **Remote sensing** – environment monitoring, geology, precision agriculture, planetary science, defense ...
but also food processing, chemometrics, document analysis...

Spectral Unmixing



Linear Spectral Unmixing

The unmixing problem

A blind source separation problem whose goals are to:

- extract the signatures of the *pure materials* (***endmembers***) present in the image
- estimate their *relative proportions* (***fractional abundances***) in each pixel.

Spectral Unmixing

Linear Mixing Model (LMM)

$$\mathbf{x}_n = \sum_{p=1}^{P} a_{pn}\mathbf{s}_p + \mathbf{e}_n = \mathbf{S}\mathbf{a}_n + \mathbf{e}_n$$

with two constraints:

- Abundance non-negativity constraint (ANC) $a_{pn} \geq 0, \forall\{p, k\}$
- Abundance sum-to-one constraint (ASC) $\sum_{p=1}^{P} a_{pn} = 1, \forall n$

In a matrix form:

$$\mathbf{X} = \mathbf{S}\mathbf{A} + \mathbf{E}$$

with $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{L\times N}$, $\mathbf{E} \in \mathbb{R}^{L\times N}$, $\mathbf{S} \in \mathbb{R}^{L\times P}$ and $\mathbf{A} \in \mathbb{R}^{P\times N}$.

L: number of bands
N: number of pixels
P: number of endmembers

$\mathbf{X}$: data matrix
$\mathbf{S}$: endmember matrix
$\mathbf{A}$: abundance matrix
$\mathbf{E}$: additive noise matrix

Outline

Conventional Spectral Unmixing chain

- Estimation of the Intrinsic Dimensionality (ID) of the data
  $\rightarrow$ number of endmembers P
- Extraction of the endmember spectra $\mathbf{S}$ with geometrical approaches.
- Abundance estimation (Δ_P is the unit simplex):

$$\underset{\mathbf{a}_n \in \Delta_P,\ \forall n=1,\ldots,N}{\arg\min} \frac{1}{2}||\mathbf{X} - \mathbf{SA}||_F^2$$

The optimization problem is embarrasingly parallel since

$$\frac{1}{2}||\mathbf{X} - \mathbf{SA}||_F^2 = \frac{1}{2}\sum_{i=1}^{N}||\mathbf{x}_n - \mathbf{S}\mathbf{a}_n||_2^2$$

Outline

Outline

Objectives

Implement the unmixing algorithm in Julia and run it in parallel, to see the gains with respect to a standard serial implementation.

1. Implement an algorithm solving the optimization problem

$$\underset{\mathbf{a}_n \in \Delta_P,\ \forall n=1,\ldots,N}{\arg\min} \frac{1}{2}||\mathbf{X} - \mathbf{SA}||_F^2$$

   incorporating at least the nonnegativity constraint and if possible the sum-to-one constraint as well.

   - Abundance non-negativity constraint (ANC) $a_{pn} \geq 0, \forall \{p, k\}$
   - Abundance sum-to-one constraint (ASC) $\sum_{p=1}^{P} a_{pn} = 1, \forall n$

2. Visualize and interpret the results
3. Implement a parallelized version either by
   - parallelizing the whole algorithm over all pixels
   - dividing the whole image into as many tiles as processes
   - or any other parallelization strategy
4. Study the running times of the algorithms depending on various parameters:
   - The input image size
   - The number of processes
   - The stopping criterion

Outline

Tips and things to try

- A suggested algorithm is a projected gradient algorithm.

  When only the negativity constraint is considered, a projected gradient descent can be performed by a standard gradient descent with a thresholding of negative values of the iterates to zero.

- Start with subsets of the full image to test your algorithms quickly before running everything in parallel

- Compare different versions of that algorithm:
  - a fully global implementation (computing the global gradient over all pixels)
  - a pixelwise implementation using a for loop
  - A parallelized version of the same algorithm

- In gradient descent, a good step size can be set to twice the reciprocal of the smallest Lipschitz constant L of the gradient. For the function $||\mathbf{x}_n - \mathbf{S}\mathbf{a}_n||$, you can check that a value for L is

$$L = |||\mathbf{S}^T\mathbf{S}|||$$

  which denotes the operator norm of $\mathbf{S}^T\mathbf{S}$ (LinearAlgerbra.opnorm in Julia).

- Be creative: think of ways to tune parameters and plots to compare serial and parallel versions of the algorithms.

Projected gradient descent

Improve gradient descent to handle "simple" constraint sets. We want to optimize

$$\arg\min_{\mathbf{x}} f(\mathbf{x})$$
$$\text{s.t. } \mathbf{x} \in \mathcal{S}$$

where $f : \mathbb{R}^D \to \mathbb{R}$ is a convex function, and $\mathcal{S}$ is a convex set.

Algorithm

We add an additional step to each update of gradient descent:

$$x_{k+1} = \mathbf{proj}_{\mathcal{S}}(\mathbf{x}_k - \rho \nabla f(\mathbf{x}))$$

where $\mathbf{proj}_{\mathcal{S}}(\mathbf{x})$ denotes the projection of $\mathbf{x}$ on the set $\mathcal{S}$, i.e. the solution of

$$\arg\min_{\mathbf{y}} ||\mathbf{x} - \mathbf{y}||_2^2$$
$$\text{s.t. } \mathbf{y} \in \mathcal{S}$$

Example: the projection on the nonnegative orthant $\mathbb{R}^{D+} = \{\mathbf{x} \in \mathbb{R}^D, \forall i = 1, \cdots, D, x_i \geq 0\}$ is given by thresholding all the negative values to zero.

Practical Details

- Better to use a computer with at least 4 physical cores for testing...
- But you can write code and test the algorithms on any computer...
- Group work with at least 2 students in each group
- Deliverable: a Jupyter notebook with code and comments. Date of submission: TBD
- Remember: The objective is parallel computing more than a fancy algorithm: better to have a simple algorithm with only the nonnegativity constraint implemented in parallel than just a serial algorithm with all constraints enforced.