Image Processing using Graph Laplacian Operator

David Wobrock
david.wobrock@gmail.com

ALPINES Team - INRIA Paris
Laboratoire Jacques-Louis Lions - Sorbonne Université
KTH, Stockholm
INSA Lyon

April 24, 2018

Table of Contents

Background

- Millions of pictures shared daily
- Image processing using spectral methods on smartphones (denoising, sharpening, ...)
- Involves eigenvalue problems, linear algebra and solving dense linear systems
- Opportunity for high-performance computing and parallelism

Objective

- Not necessarily improving image processing
- Analyse the behaviour of solving large dense systems

- Large: N^2, N the number of pixels in the input image
- Dense: affinity and Laplacian matrices are dense because global filter

Image processing - Global filter algorithm

- z output image, y input image, W data-dependent global filter of size N^2
- $z_i = \sum_{j=1}^{N} W_{ij} y_j$
- A vector of weights for each pixel
- Global filter: $z = Wy$

Image processing - Image as graph

- Think of the image as a graph
- Each pixel is a node
- The graph is complete and the edges are weighted
- The weight represents the similarity between two pixels/nodes
- Similarity can be measured by weighting together spatial and color closeness of pixels
- Bilateral kernel: $exp(-\frac{||x_i - x_j||^2}{h_x^2}) \cdot exp(-\frac{||z_i - z_j||^2}{h_z^2})$

Image processing - Filter and Laplacian

- K affinity matrix, $\mathcal{L}$ Laplacian matrix, W filter matrix and f a function
- Filter defined as $W = I - f(\mathcal{L})$
- Let $d_i = \sum_j K_{ij}$, $D = diag\{d_i\}$ and $\bar{d} = \frac{1}{N}\sum_i d_i$

Laplacian	Formula of $\mathcal{L}$	Symmetric	Spectral Range
Un-normalised	$D - K$	Yes	[0, n]
Normalised	$I - D^{-1/2}KD^{-1/2}$	Yes	[0, 2]
Random walk	$I - D^{-1}K$	No	[0, 1]
"Sinkhorn"	$I - C^{-1/2}KC^{-1/2}$	Yes	[0, 1]
Re-normalised	$\alpha(D - K)$, $\alpha \approx \bar{d}^{-1}$	Yes	[0, n]

Table: Overview of different graph Laplacian operator definitions.

Image processing - Sampling and Nyström extension

- Sample p pixels of the image ($\leq 1\%$ of pixels)
- With $K_A \in \mathbb{R}^{p\times p}$,$K_B \in \mathbb{R}^{p\times N-p}$ and $K_C \in \mathbb{R}^{N-p\times N-p}$
  $K = \begin{bmatrix} K_A & K_B \\ K_B^T & K_C \end{bmatrix}$
- Approximate K by $\tilde{K} = \tilde{\Phi}\tilde{\Pi}\tilde{\Phi^T}$
- With submatrix $K_A = \Phi_A \Pi_A \Phi_A^T$,
  $\tilde{\Pi} = \Pi_A$
  $\tilde{\Phi} = \begin{bmatrix} \Phi_A \\ K_B^T \Phi_A \Pi_A^{-1} \end{bmatrix}$
- Finally, $\tilde{K} = \begin{bmatrix} K_A & K_B \\ K_B^T & K_B^T K_A^{-1} K_B \end{bmatrix}$

Image processing - Eigenvalues

- For the filter, only the largest eigenvalues are needed because the smallest tends to 0
- There is an an equivalence between largest eigenvalues of W and the smallest ones of $\mathcal{L}$
- The smallest eigenvalues can be computed with the inverse power method, which requires solving systems of linear equations
- But the Nyström extension only works for leading eigenvalues.

Image processing - Algorithm recap

Algorithm 1 Image processing using approximated graph Laplacian operator

Input: y an image of size N, f the function applied to $\mathcal{L}$

Output: $\tilde{z}$ the output image by the approximated filter

{Sampling}

Sample p pixels, $p \ll N$

{Kernel matrix approximation}

Compute K_A (size $p \times p$) and K_B (size $p \times (N - p)$)

Compute the Laplacian submatrices $\mathcal{L}_A$ and $\mathcal{L}_B$

{Eigendecomposition}

Compute the m smallest eigenvalues Π_A and the associated eigenvectors Φ_A of $\mathcal{L}_A$

{Nyström extension and compute the filter}

See methods of solution proposed by Fowlkes, 2004

$\tilde{z} \leftarrow \tilde{W} y$

Parallel implementation details

- C language
- PETSc (Toolkit for scientific computing in parallel)
- SLEPc, Elemental

Full matrix computation result

Figure: Left: input image. Right: sharpened image.

Runtime of full matrix computation

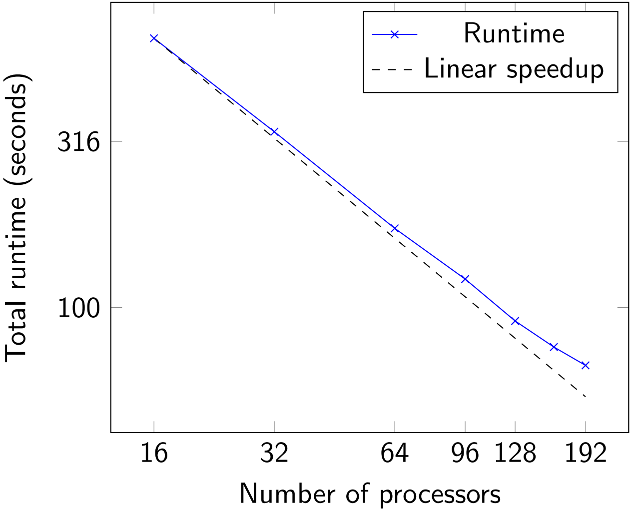


Figure: Total runtime of the algorithm with entire matrix computation (log scale).

Inverse subspace iteration

Algorithm 2 Inverse subspace iteration

Input: A the matrix of size $p \times p$, m the number of required eigenvalues, ε a tolerance

Output: X_k the desired invariant subspace

Initialise m random orthonormal vectors X_0 of size p

$R_0 \leftarrow (I - X_0 X_0^T) A X_0$

For k=0, 1, 2, . . .

while $\|R_k\| > \varepsilon$ **do**

  for i=1 **to** m **do**

    Solve $AX_{k+1}^{(i)} = X_k^{(i)}$

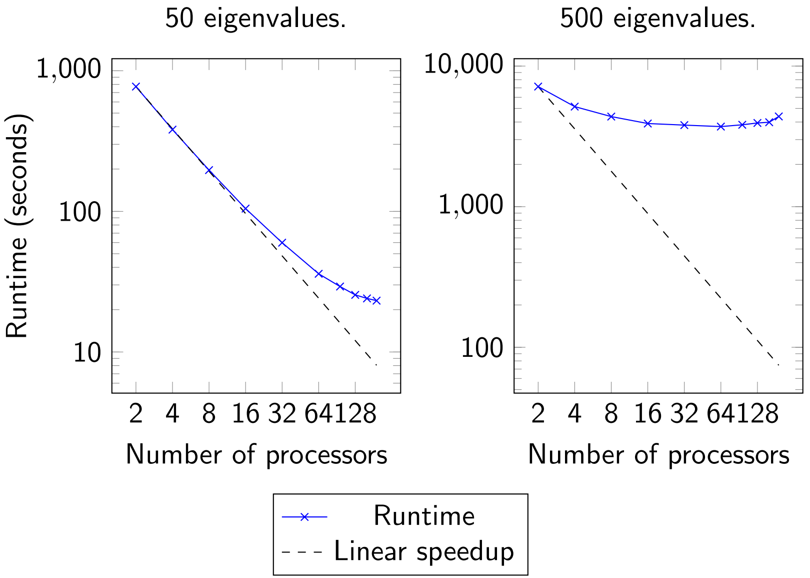
  end for

  $X_{k+1} \leftarrow \text{Orthonormalise}(X_{k+1})$

  $R_{k+1} \leftarrow (I - X_{k+1} X_{k+1}^T) A X_{k+1}$

end while

Approximation runtime of approximated computation



Runtime in the inverse subspace iteration algorithm

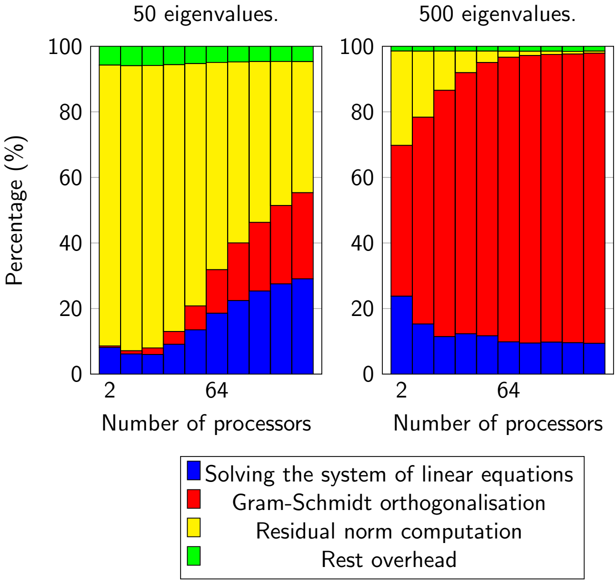
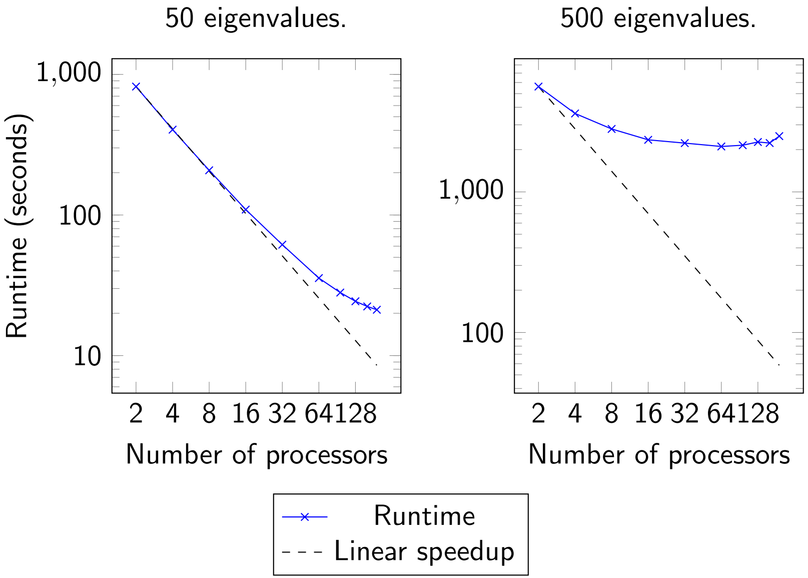


Figure: Proportion of each step in the inverse subspace iteration.

Skipping the Gram-Schmidt procedure



Skipping Gram-Schmidt more often

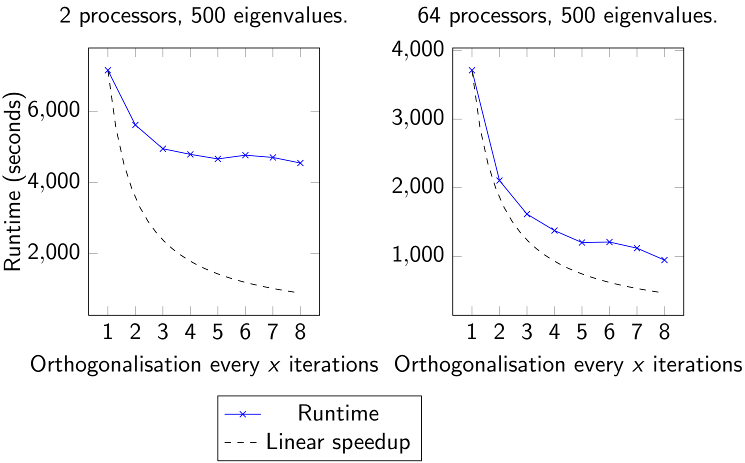


Figure: Runtime of the inverse subspace iteration depending on the amount of Gram-Schmidt procedures.

Linear solver performances

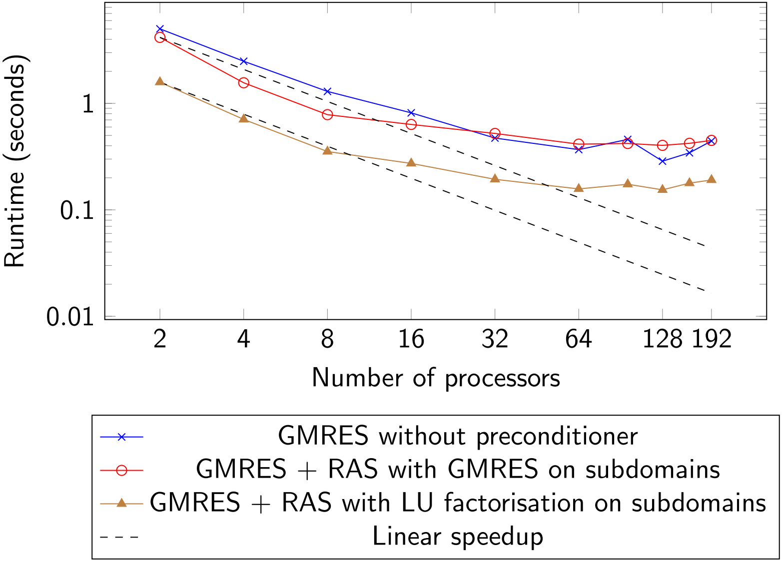


Figure: Runtime of the linear solver for 50 eigenvalues for a 4000 × 4000 matrix (log scale).

Linear solver performances - big image

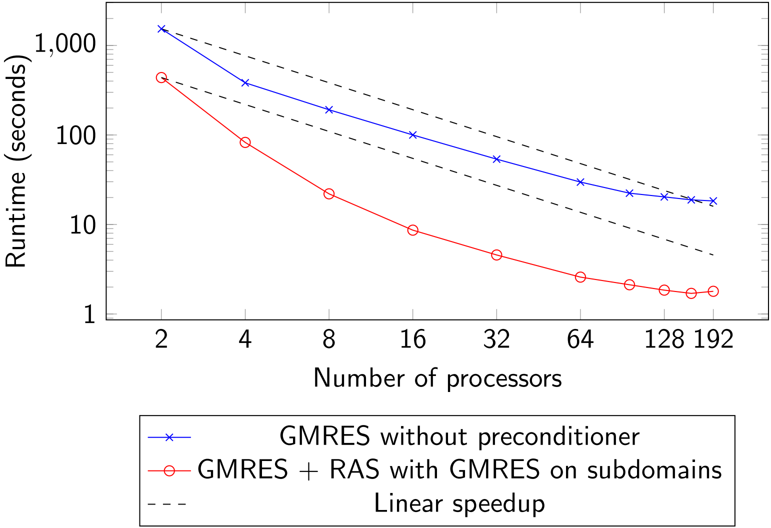


Figure: Runtime of the linear solver for 50 eigenvalues for a 14400×14400 matrix (log scale).

Discussions & perspectives

- Linear solver with domain decomposition methods as preconditioner
- Skipping Gram-Schmidt

- Finished image processing algorithm
- State-of-the-art performances with a similar method
- Explore more solvers, domain decomposition methods and other preconditioner