Image Processing using Graph Laplacian Operator

David Wobrock
david.wobrock@gmail.com

March 28, 2018

Under the responsability of:

Supervisor: Frédéric Nataf
ALPINES Team - INRIA Paris
Laboratoire Jacques-Louis Lions - Sorbonne Université

INSA Supervisor: Christine Solnon
Département Informatique
INSA de Lyon

Abstract

The latest image processing methods are based on global data-dependent filters. These methods involve huge affinity matrices which cannot fit in memory and need to be approximated using spectral decomposition. The inferred eigenvalue problem concerns the smallest eigenvalues of the Laplacian operator which can be solved using the inverse iteration method which, in turn, involves solving linear systems. In this master thesis, we detail the functioning of the spectral algorithm for image editing and explore the behaviour of solving large and dense systems of linear equations in parallel, in the context of image processing. We use Krylov type solvers, such as GMRES, and precondition them using domain decomposition methods to solve the systems on high-performance clusters. The experiments show that Schwarz methods as preconditioner scale well as we increase the number of processors on large images. However, we observe that the limiting factor is the Gram-Schmidt orthogonalisation procedure.

Keywords image processing, graph Laplacian operator, eigenvalue problem, high performance computing

Résumé

Les méthodes récentes de traitement d'images sont basées sur des filtres globaux dépendant des données. Ces méthodes impliquent le calcul de matrices de similarités de grandes tailles qui ne peuvent pas être contenues en mémoire et nécessitent ainsi d'être approchées par une décomposition spectrale. Le problème aux valeurs propres qui en découle concerne les plus petites valeurs propres du Laplacien qui peuvent être calculées grâce à la méthode de la puissance inverse qui implique la résolution de systèmes linéaires. Dans cette thèse de master, nous détaillons le fonctionnement de l'algorithme spectral pour l'édition d'images et étudions le comportement de la résolution de systèmes d'équations linéaires sur de grandes matrices pleines en parallèle, dans le cadre du traitement d'images. Nous utilisons des solveurs de type Krylov, tel que GMRES, et nous les préconditionnons avec des méthodes de décomposition de domaines pour résoudre les systèmes sur des clusters haute performance. Les expériences montrent que les méthodes de Schwarz comme préconditionneur passent bien à l'échelle quand on augmente le nombre de processeurs sur des images de grandes tailles. Toutefois, on observe que le facteur limitant est la procédure d'orthogonalisation de Gram-Schmidt.

Mots-clés traitement d'images, Laplacien graphe, problème aux valeurs propres, calcul haute performance

1 Introduction

1.1 Background

The talk [1] and articles [2] [3] by Milanfar, working at Google Research, about using the graph Laplacian operator for nonconformist image processing purposes awakes curiosity.

Indeed, Milanfar reports that these techniques to build image filters are used on smartphones, which implies a reasonable execution time with limited computational resources. Over 2 billion photos are shared daily on social media [1], with very high resolutions and most of the time some processing or filter is applied to them. The algorithm must be efficient to be deployed at such scale.

1.2 Objective

The aim of this degree project is not to explore and improve the state of image processing. Instead, the spectral methods used in the algorithm will be our focus point. Those will inevitably expose eigenvalue problems, which may involve solving systems of linear equations.

Concerning the challenges about solving linear systems, on the one hand, the size of the systems can be large when considering high-resolution images with millions of pixels, or even considering 3D images. We process huge matrices of size N^2, with N the number of pixels of the input image. On the other hand, those huge affinity matrices are dense, thus the linear systems are dense. Often, linear systems result from discretising partial differential equations (PDEs) yielding sparse matrices, and therefore most linear solvers are specialised in sparse systems.

The aim of this work is to explore the performance of linear solvers on dense problems, their stability and convergence. This will include preconditioning the linear systems, especially using domain decomposition methods, and analyse their behaviour on dense systems.

2 Image processing using graph Laplacian operator

A global image filter consists of a function which results in one pixel, taking all pixels as input and applying weights to them.

As a practical notation, we say that, with W the matrix of weights and y and z respectively the input and output images as vectors,

$$z = Wy.$$

The filter matrix W considered here is data-dependent and built upon the input image y. A more mathematical notation would consider W as a nonlinear function of the input image such as $z = W(y) \cdot y$.

Image as graph We convert the image to a graph by considering each pixel as a node which is linked to every other node by an edge, thus to every pixel. The graph is therefore complete and we assign weights to the edges through a similarity[1] measure of the two concerned nodes. This measure will typically be a weighted average of the spatial and colour closeness of the pixels.

We can compute from this graph the affinity matrix[2] K which represents the similarity of each pixel to every other pixel in the image. Consequently, this matrix is symmetric and of size $N \times N$ with N the number of pixels in the image. Using this affinity matrix, we obtain the graph Laplacian $\mathcal{L}$, used to build the filter matrix W.

Building the filter Multiple graph Laplacian definitions, more or less equivalent, exist and can have slightly different properties. In the case of image smoothing, the filter W is roughly defined such as $\mathcal{L} = I - W$ [1] and so $W = I - \mathcal{L}$. To get various filters using one Laplacian, we can apply some function f to $\mathcal{L}$ and obtain $W = I - f(\mathcal{L})$ which gives us more possibilities on the filter computation.

However, all these matrices K, $\mathcal{L}$ and W represent huge computational costs. Only storing one of these matrices is already a challenge since they have a size of N^2.

[1] Also called affinity.

[2] Or similarity matrix, or kernel matrix

A modern smartphone picture, taken with a 10 megapixel camera, involves matrices that contains 10^{14} elements, meaning 800 TB of memory for each matrix.

Approximation by sampling and Nyström extension To avoid storing any of those huge matrices, approximation will be necessary. Following [4], we define the Nyström extension. It starts by sampling the image and only select a subset of p pixels, with $p \ll N$. Numerically, p should represent around 1% or less of the image pixels. We only need to compute of subset of the affinity matrix K and then compute the eigendecomposition of a submatrix. The eigenvectors of the submatrix are extended to the entire matrix and used to reconstruct it.

Therefore, we will only need to compute two submatrices. For our previous example, for the 10 megapixel image, each matrix needs 8 TB of memory, which is still a lot of memory, but the sampling rate can be lower than 1% as suggested by [4] and [2].

Eigendecomposition We need to compute the largest eigenvalues of the filter W.

It can easily be shown that the largest eigenvalues of the submatrix filter is equivalent to computing the smallest eigenvalues of the corresponding Laplacian operator.

To compute the latter, we use the inverse power method which involves solving systems of linear equations.

The main drawback of the Nyström method for us, is that it approximates the leading eigenvalues but we compute the trailing ones of the Laplacian $\mathcal{L}$. However, we stick to compute the smallest eigenvalues even if it means the algorithm cannot compute the output image for now.

Below a summary of the complete algorithm using spectral decomposition of the matrix to approximate it:

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 $\Pi_A$ and the associated eigenvectors $\Phi_A$ of $\mathcal{L}_A$
  {Nyström extension and compute the filter}
  See methods of solution proposed by [4]
  $\tilde{z} \leftarrow \tilde{W}y$
```

3 Implementation

3.1 Parallel implementation

We implemented our algorithm using the C language the Portable, Extensible Toolkit for Scientific Computation (PETSc) [5]. This library is built upon the message passing interface (MPI) and contains distributed data structures and parallel scientific computation routines.

The implementation associated to this project is open source and can be found on GitHub[3].

3.2 Inverse subspace method

The used algorithm to compute the smallest eigenvalues is the inverse subspace iteration inspired by [6].

Solving the systems of linear equations is done using the Krylov type solvers and the preconditioners included in PETSc. As a standard approach, we use GMRES as our solver and the RAS method as preconditioner, without overlap and 2 domains per process. Each subdomain is solved using the GMRES method also.

On each outer iteration, we must compute the residuals to see if we converged. This requires multiple matrix-matrix products and computing a norm, so communication cannot be avoided here. We implemented a parallel Gram-Schmidt routine for orthogonalisation, based on the classical sequential one.

3.3 Entire matrix computation

We start by showing the result of the computation using the full matrices in order to see the image processing result. We limit ourselves to grayscale images for the beginning. As stated before, computing the entire matrices can only be done on small images. The image below contains 135 000 pixels, so each matrix needs around 145 GB.

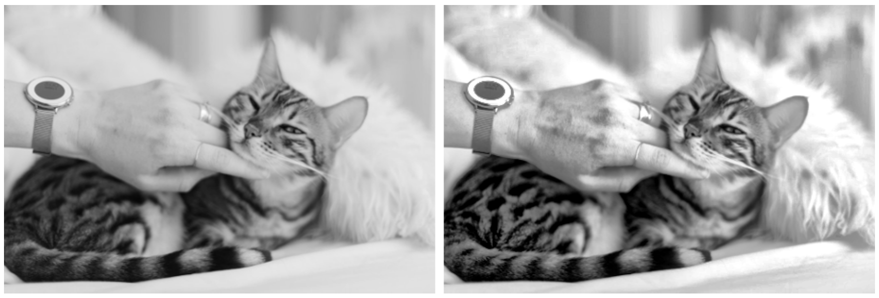

Figure 1 – Left: input image. Right: sharpened image.

[3]`https://github.com/David-Wobrock/image-processing-graph-laplacian/`

As figure 1 shows, the cat's fur on the left-hand side, the person's hand and the cushion on the right-hand side appear to be more detailed. We observe that the already sharp part of the image, such as the cat's head, stays nice and is not over-sharpened. We obtained this filter by defining $f(\mathcal{L}) = -3\,\mathcal{L}$ in the output image $z = (I - f(\mathcal{L}))y$.

This corresponds to the adaptive sharpening operator defined in [1] as $(I + \beta\,\mathcal{L})$ with $\beta > 0$. This approach remains a simple application of a scalar and doesn't require any eigenvalue computation.

3.4 Approximate matrix computation

As a reminder, we used GMRES to solve the linear systems with RAS preconditioning and using GMRES on the subdomains. We sample 1% of the pixels of an image with $4 \cdot 10^5$ pixels using spatially uniform sampling. The performances of the inverse subspace iteration for 50 and 500 eigenvalues:

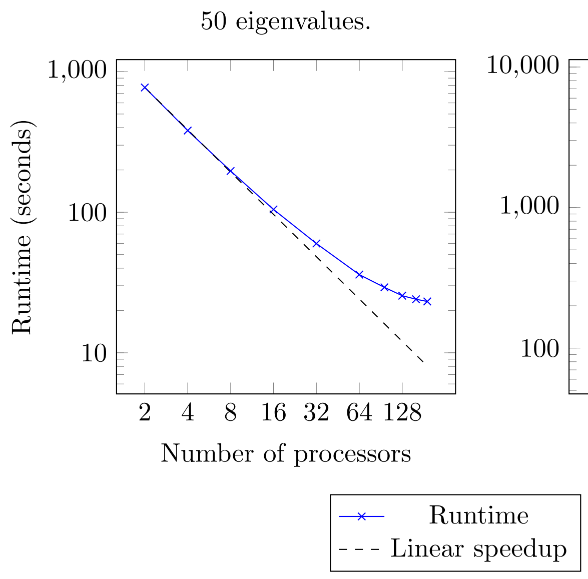

Figure 2 – Runtime of the inverse subspace iteration part of the algorithm (log scale).

When increasing the number of processors, we see an improvement of the performances in both cases of figure 2. But the runtime stagnates slowly for 50 eigenvalues and quickly for 500 eigenvalues. We even observe a raise of the runtime for 500 eigenvalues. The algorithm reaches its parallelisation limit and the communication overhead takes over. For 500 eigenvalues, the runtime for 2 processors is over 7000 seconds, and the fastest runtime is reached for 64 processors and is of 3700 seconds.

A study of the internal steps of the inverse power method allows an insight on the origins of the problem. The algorithm consists of iteratively solving m linear systems, orthonormalising the vectors and computing the residual norm. The limiting factor is the Gram-Schmidt orthogonalisation which requires many communications. It is a well-known problem that will be difficult to overcome completely.

3.5 Skipping some orthogonalisations

Fundamentally, the orthogonalisation is used to stabilise the algorithm. To accelerate our algorithm further, we try to orthogonalise the vectors X_k less often and skip some Gram-Schmidt procedures.

Below the runtime for 2 and 64 processors of the inverse subspace algorithm depending on the frequency of orthogonalisation is given:

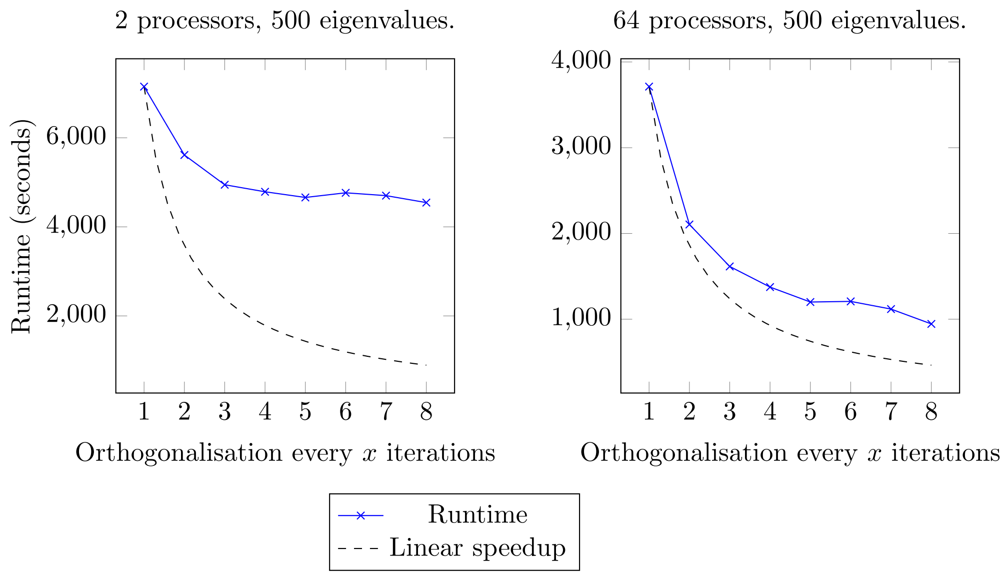

Figure 3 – Runtime of the inverse subspace iteration depending on the amount of Gram-Schmidt procedures.

For 2 processors, we see in figure 3 that the speedup stagnates quickly when skipping the Gram-Schmidt orthogonalisation more and more often. Indeed, the orthogonalisation represents approximately a quarter of the execution time for 2 processors, whereas it represents over 85% for 64 processors. We observe from figure 3 a speedup of the inverse subspace iteration that is nearly linear for 64 processors. The less we stabilise the algorithm through orthogonalisation, the faster it gets in this case. The number of outer iterations between not skipping the Gram-Schmidt algorithm and applying it every 8 iterations only varies from 38 to 42 which explains the resulting runtime.

3.6 Linear solver performances

In this section will be studied the behavior of the linear solver using domain decomposition methods for our inverse iteration algorithm. We remind that we used GMRES and the RAS domain decomposition method with 2 domains per process for the previous examples. We compare the runtimes of solving 50 linear systems with different number of processors, with and without preconditioner on a large image:

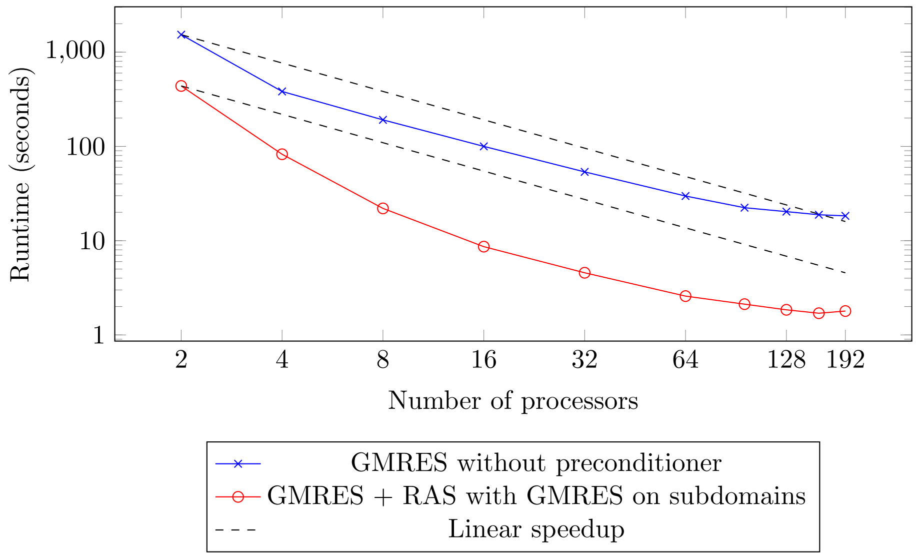

Figure 4 – Runtime of the linear solver for 50 eigenvalues for a 14400 × 14400 matrix (log scale).

Figure 4 still shows more explicitly the benefits of preconditioning the linear system through domain decomposition methods. Without preconditioner, solving 50 systems iteratively on 2 processors takes more than 1500 seconds, whereas with the restricted additive Schwarz method, it only takes around 400 seconds. When scaling up to 192 processors, solving the systems takes about 18 seconds without preconditioner and 1.8 seconds with the domain decomposition method.

In the end, we observe that for large and dense systems of linear equations in the context of image processing, domain decomposition methods present a good and naturally parallel preconditioner.

4 Conclusion

4.1 Discussions

Linear solvers & domain decomposition methods on dense matrices The presented experiments show that the resolution of linear systems scales with respect to the number of processors. Domain decomposition methods improve the performances of solving dense systems of linear equations, even without overlap, in the context of image processing. Indeed, these methods are naturally parallel and precondition the matrix appropriately. This becomes clear when observing the performances improvement on large matrices.

Gram-Schmidt process The orthogonalisation process is difficult to parallelise efficiently. Skipping the Gram-Schmidt procedure tends to improve the runtime when a large amount of processors is involved. However, the stability of the algorithm diminishes when omitting numerous orthogonalisations. This scalability problem is well-known and one of the biggest limitations for scaling diverse algorithms to a large number of processors.

The Gram-Schmidt procedure orthogonalises a set of vectors by sequentially substracting from a vector the projections on the previously orthogonalised vectors. The inner product of two vectors is computed frequently, because of the projection, and since each vector is shared over all processors, a lot of communication is involved in this operation. Attempts for parallel implementation are numerous, like [7], but they either still have many communications or suggest a different memory distribution schema.

4.2 Perspectives

Image processing Numerous possibilities for improving the algorithm result from the applied simplifications. An important matter would be to complete the final part of the algorithm computing the output image. This requires to compute the inverse of the square root of the matrix.

A way to improve the filtering is multiscale decomposition. As explained in [3], instead of applying a linear function to all eigenvalues such as $f(W) = \phi f(\Pi)\phi^T$, we can actually use a polynomial function f. This is interesting because each eigenpair captures various features of the image and one can apply different coefficients on different aspects of the image.

For the state-of-the-art, the article [8] proposes an enhancement of global filtering. They argue that the eigendecomposition remains computationally too expensive and show results of an improvement. The presented results and performances are astonishing; however, the method is hardly described and replicating it would be difficult. This is understandable since this algorithm seems to be in the latest Pixel 2 smartphone by Google and they surely want to preserve their market advantage in the field of image processing.

An improvement of the algorithm of the present case, would be to formulate a method for extending the trailing eigenvectors of the sampled Laplacian $\mathcal{L}_A$ and not only the leading ones as the Nyström extension supports. This way, it would be possible to apply the spectral decomposition of the Laplacian, and thus apply a filter to the input image, avoiding the computation of the inverse square root of the matrix.

To conclude, various possibilities remain to be exploited by future work.

References

[1] Peyman Milanfar. *Non-conformist Image Processing with Graph Laplacian Operator*. 2016. URL: `https://www.pathlms.com/siam/courses/2426/sections/3234`.

[2] Hossein Talebi and Peyman Milanfar. "Global Image Denoising". In: *IEEE Transactions on Image Processing* 23.2 (Feb. 2014), pp. 755–768. ISSN: 1057-7149, 1941-0042. DOI: `10.1109/TIP.2013.2293425`. URL: `http://ieeexplore.ieee.org/document/6678291/`.

[3] H. Talebi and P. Milanfar. "Nonlocal Image Editing". In: *IEEE Transactions on Image Processing* 23.10 (2014), pp. 4460–4473. ISSN: 1057-7149. DOI: `10.1109/TIP.2014.2348870`.

[4] Charless Fowlkes et al. "Spectral grouping using the Nystrom method". In: *IEEE transactions on pattern analysis and machine intelligence* 26.2 (2004), pp. 214–225. URL: `http://ieeexplore.ieee.org/abstract/document/1262185/`.

[5] Satish Balay et al. *PETSc Web page*. `http://www.mcs.anl.gov/petsc`. 2017. URL: `http://www.mcs.anl.gov/petsc`.

[6] G. El Khoury, Yu. M. Nechepurenko, and M. Sadkane. "Acceleration of inverse subspace iteration with Newton's method". In: *Journal of Computational and Applied Mathematics*. Proceedings of the Sixteenth International Congress on Computational and Applied Mathematics (ICCAM-2012), Ghent, Belgium, 9-13 July, 2012 259 (Mar. 2014), pp. 205–215. ISSN: 0377-0427. DOI: `10.1016/j.cam.2013.06.046`. URL: `http://www.sciencedirect.com/science/article/pii/S0377042713003440`.

[7] Takahiro Katagiri. "Performance Evaluation of Parallel Gram-Schmidt Re-orthogonalization Methods". In: *High Performance Computing for Computational Science — VECPAR 2002*. Ed. by José M. L. M. Palma et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 302–314. ISBN: 978-3-540-36569-3.

[8] Hossein Talebi and Peyman Milanfar. "Fast Multilayer Laplacian Enhancement". In: *IEEE Transactions on Computational Imaging* 2.4 (Dec. 2016), pp. 496–509. ISSN: 2333-9403, 2334-0118. DOI: `10.1109/TCI.2016.2607142`. URL: `http://ieeexplore.ieee.org/document/7563313/`.