02616 – ASSIGNMENT 2: IMAGE FILTERING

S090763 – Olivier Jais-Nielsen

CONTENTS

1. Mathematical context

a. Introduction

The algorithm studied here is an image filtering algorithm. Its purpose is to filter an image to remove the noise while preserving the edges.

b. Continuous System

The image filtering algorithm implemented here is a simplified version of steady-state resolution of a single diffusion-reaction equation. The original equation system is the following (1) with f the original image and $g(\|\nabla u\|^2)$ being the Perona-Malik diffusivity, that is $g: s \longmapsto \left(1+\frac{s}{\lambda}\right)$, with λ and β positive parameters:

$$\frac{\partial u}{\partial t} = \nabla^T(g(\|\nabla u\|^2)\nabla u) + \beta(f-u)$$

The boundary conditions are the following: $\frac{\partial u}{\partial n} = 0$ where $\frac{\partial u}{\partial n}$ is the spatial derivative in the direction normal to the boundary.

The simplification transforms the preceding equation into two successive steady-state resolutions of diffusion reactions with β_1, β_2 and γ three positive parameters:

$$\frac{\partial u}{\partial t} = \nabla^T(\gamma\nabla u_1) + \beta_1(f-u_1)$$

$$\frac{\partial u}{\partial t} = \nabla^T(g(\|\nabla u_1\|^2)\nabla u_2) + \beta_2(f-u_2)$$

They can be rewritten as follows with k_1 and k_2 positive parameters:

$$-k_1\Delta u_1 + u_1 = f$$

$$-\mathrm{k}_2\nabla^T(g(\|\nabla u_1\|^2)\nabla u_2) + u_2 = f$$

c. Discretized System

Using the finite central differences discretization scheme, we get the following two linear systems with b being the original image pixels, $\tilde{b}$ being an image composed of the original image and additional 1-pixel-wide-zero-valued edges, and A_1 and A_2 being two sparse matrices:

$$A_1x_1 = \tilde{b}$$

$$A_2x_2 = b$$

The reason for solving the first diffusion on a domain bigger than the original image is to be able to evaluate the gradient and the Perona-Malik diffusivity of the result at any location of the original image, including the edges.

The values of A_1 are such that if (i,j) are inner image coordinates, we have the following five-point stencil formula with K_1 a positive parameter:

$$[A_1x]_{i,j} = (4K_1+1)x_{i,j} - K_1\left[x_{i-1,j} + x_{i,j-1} + x_{i+1,j} + x_{i,j+1}\right]$$

For image coordinates corresponding to edges, we define the undefined terms following the boundary conditions, that is, with (M,N) the dimensions of the considered domain:

$$\begin{cases} x_{M+1,j} \overset{\text{def}}{=} x_{M-1,j} \, for \; 1 \le j \le N \\ x_{0,j} \overset{\text{def}}{=} x_{2,j} \, for \; 1 \le j \le N \\ x_{i,N+1} \overset{\text{def}}{=} x_{i,N-1} \, for \; 1 \le i \le M \\ x_{i,0} \overset{\text{def}}{=} x_{i,2} \, for \; 1 \le i \le M \end{cases}$$

The values of A_2 are such that if (i,j) are image coordinates, we have the following five-point stencil formula with K_2 a positive parameter and G the Perona-Malik diffusivity evaluated on the output of the first resolution:

$$\begin{aligned} [A_2 x]_{i,j} = {} & (4K_2 G_{i,j} + 1) x_{i,j} \\ & - K_2 \left[\left(\frac{G_{i-1,j} - G_{i+1,j}}{4} + G_{i,j} \right) x_{i-1,j} + \left(\frac{G_{i+1,j} - G_{i-1,j}}{4} + G_{i,j} \right) x_{i+1,j} \right. \\ & \left. + \left(\frac{G_{i,j-1} - G_{i,j+1}}{4} + G_{i,j} \right) x_{i,j-1} + \left(\frac{G_{i,j+1} - G_{i,j-1}}{4} + G_{i,j} \right) x_{i,j+1} \right] \end{aligned}$$

As before, for image coordinates corresponding to edges, we define the undefined terms following the boundary conditions.

d. Resolution

Both resolutions use the conjugate gradient method as both sparse matrices are symmetric positive definite. A diagonal preconditioner is used in both cases. In both cases, the initial values are determined by setting the first estimate to zero.

2. Implementation

a. Parallelization

The first level of parallelization is implemented using MPI. The domain is divided following a Cartesian grid where each process computes the values corresponding to a block of the domain.

In order to compute the Perona-Malik diffusion, the second sparse matrix and the five-point stencil formula, each process needs to exchange with each of its neighbors the values of the considered matrix situated on the common edge of their domains. To allow the computation and the communications to overlap, each send and receive operation is asynchronous; as soon as the values to be sent are computed, the sending process initiates the send operation and as soon as its receiving buffer is unused, it initiates the receive operation.

At some points of a conjugate gradient iteration, there is the need to compute dot products; these dot products have to be performed on the entire domain and therefore, each process computes the dot product of the vectors limited to their own block and then they all reduce their contribution to obtain the global value. This is a synchronous operation.

The second level of parallelization is implemented using OpenMP. Each process uses several threads to perform the matrix operations on their own block.

b. Workflow

i. Initialization

One particular process starts by reading the input data. It communicates the domain's dimensions to the other processes. Each process initializes the Cartesian communicator resulting from these dimensions.

Additionally each process computes the ranks of its neighboring processes. The process that read the data dispatches the different blocks to the appropriate processes.

The same particular process synchronizes all the processes and starts a timer.

ii. Computation

Each process performs the computations.

The sparse matrix, the preconditioner and its inverse for the first pass are computed. The matrices involved in the conjugate gradient resolution are then initialized. The iterative resolution loops starts; at each iteration the norm of the residual is compared to the convergence criterion and the loop is interrupted if convergence is reach before the maximum number of iterations.

The Perona-Malik diffusivity is then computed using the previously computed solution. The sparse matrix, the preconditioner and its inverse for the second pass are computed using this diffusivity. The second resolution is then performed, similarly to the first one.

iii. Finalization

The previously mentioned particular process synchronizes all the processes and stops the timer, storing information about the computation into a log. A particular process gathers all the solution blocks from all the processes, assembles them and save the result as an image.

c. Program Structure

The program is written in C and consists of the following files.

- ***common.h***
  This file contains common macros and include statements as well as the parameters: the parameters of both diffusions, the parameters of the Perona-Malik diffusivity, the residual's norm value under which a conjugate gradient algorithm is considered converged, the maximum number of iterations. It also contains the names of the input image file, of the output image file and of the log file.
- ***matrix.c***
  This file contains the function for manipulating matrices.
- ***matrix.h***
  This file contains the declaration of the functions in the previous file and defines the matrix data structure.
- ***util.c***
  This file contains various routines concerning memory management, debugging, initialization of the Cartesian communicator and timing.
- ***util.h***
  This file contains the declaration of the routines in the previous file.
- ***conjugate_gradient.c***
  This file contains routines to initialize, iterate and finalize a conjugate gradient resolution. It also contains routines to exchange the appropriate data between neighbor processes and to enforce Neumann boundary conditions.
- ***conjugate_gradient.h***
  This file contains the declaration of the routines in the previous file.
- ***main.c***
  This file contains the main workflow.
- ***main.h***
  The declarations of the routines in the previous file as well as the declaration of the global variables were put in this file for readability purposes.

3. Results

a. Correctness

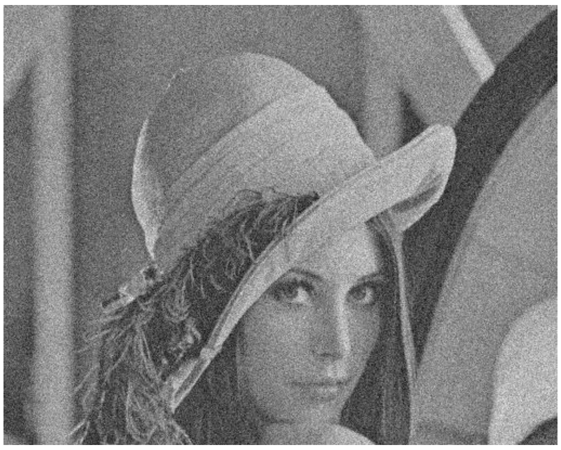
FIGURE 1 - NOISY INPUT IMAGE

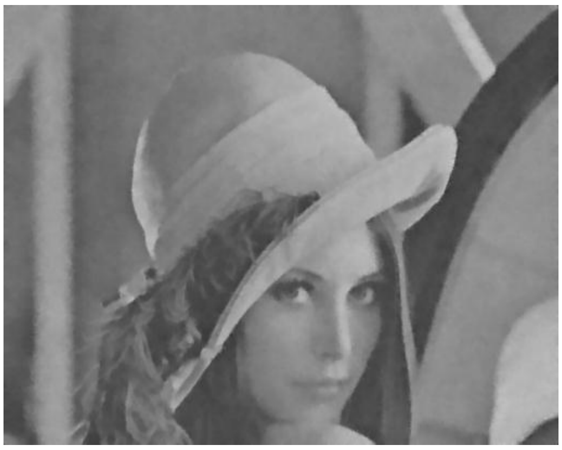
FIGURE 2 - FILTERED IMAGE

As shown in Figure 1 and Figure 2, the output of the algorithm is rather satisfactory: most of the noise is removed while the edges are respected, avoiding the blurry aspect that a simple diffusion algorithm would have created.

b. Speed

Tests have been performed on two different input images, of respectively 2,795,520 and 9,980,928 pixels. The parameters that were used are 10 for the two diffusion parameters, 0.0001 for the Perona-Malik diffusivity parameter and 0.1 for the maximum converged residual norm.

Both resolutions converged in about 30 iterations each, for both images.

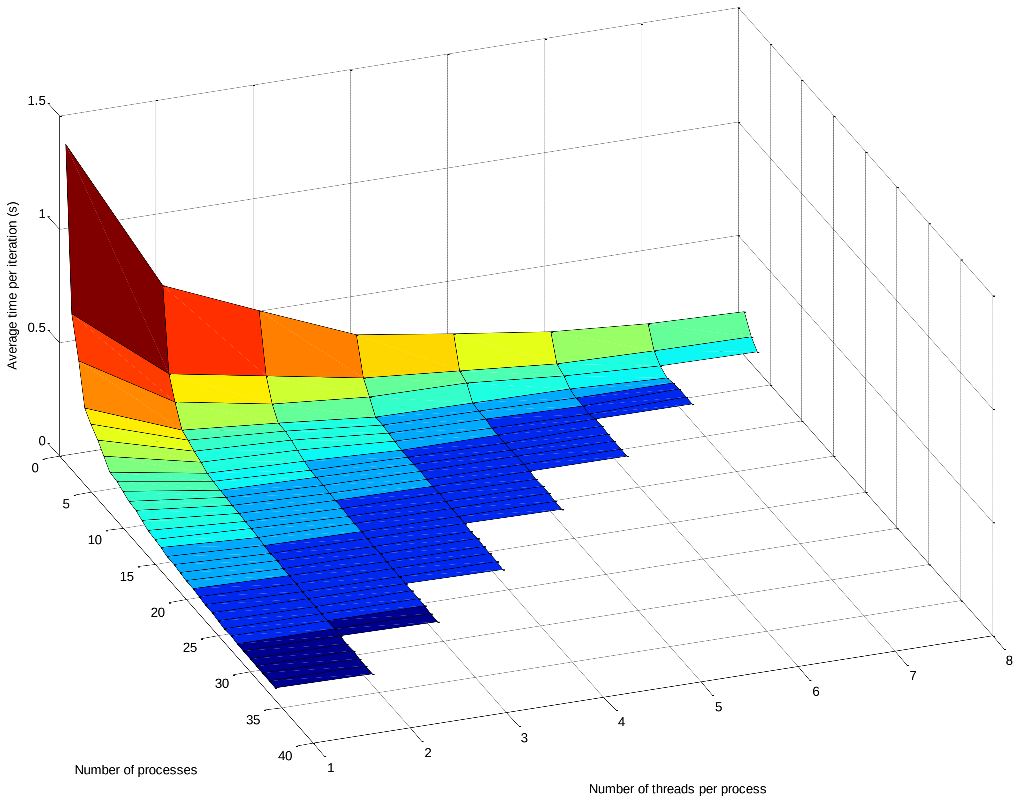

FIGURE 3 - AVERAGE COMPUTING TIME PER ITERATION FOR DIFFERENT NUMBER OF PROCESSES AND THREADS PER PROCESSES FOR AN INPUT OF 2,795,520 PIXELS

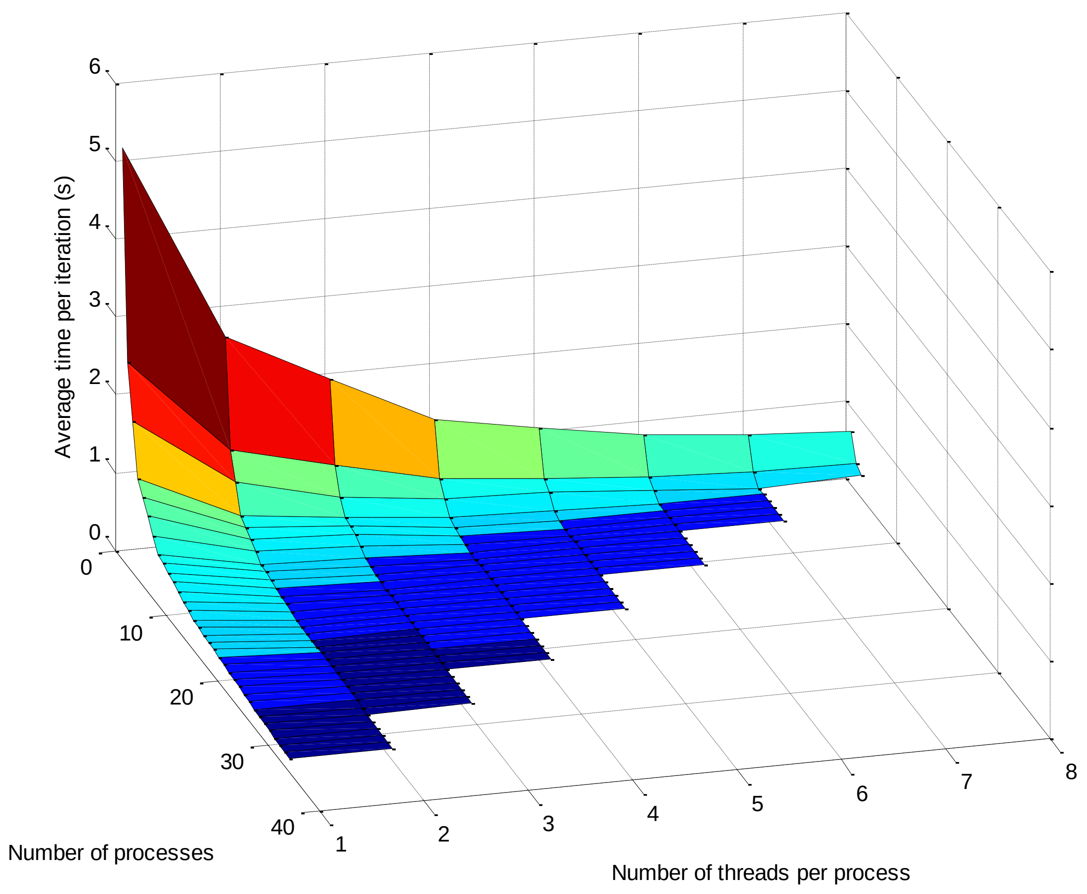

FIGURE 4 - AVERAGE COMPUTING TIME PER ITERATION FOR DIFFERENT NUMBER OF PROCESSES AND THREADS PER PROCESSES FOR AN INPUT OF 9,980,928 PIXELS

As shown in Figure 3 and Figure 4, the parallelization is very profitable for these amounts of data and reduces dramatically the computing time. If we consider small numbers of “workers”, we can see that the computing time is inversely proportional to the number of “workers”, which shows that the non-parallelizable overhead is negligible in comparison to the parallelizable part of the algorithm.

c. Parallelization levels efficiency

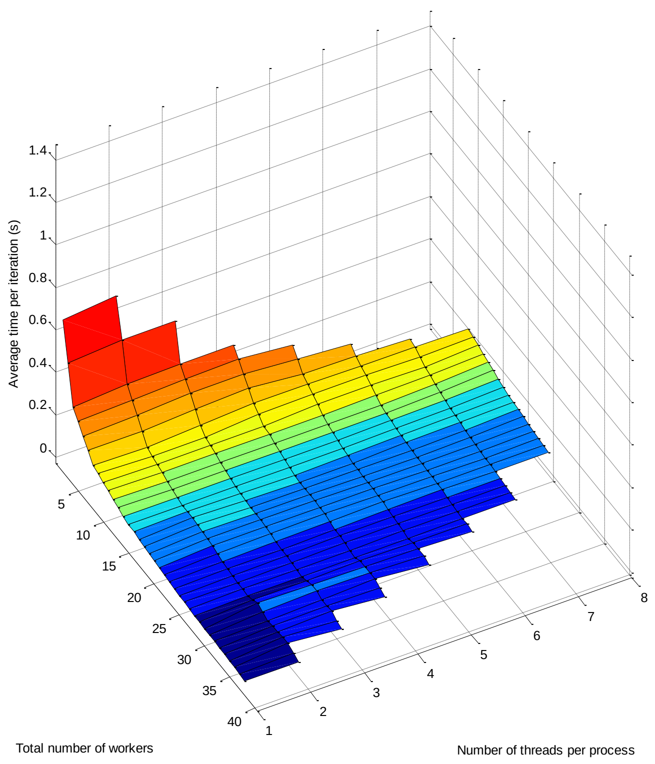

FIGURE 5 - AVERAGE COMPUTING TIME PER ITERATION FOR DIFFERENT NUMBER OF WORKERS AND THREADS PER PROCESSES FOR AN INPUT OF 2,795,520 PIXELS

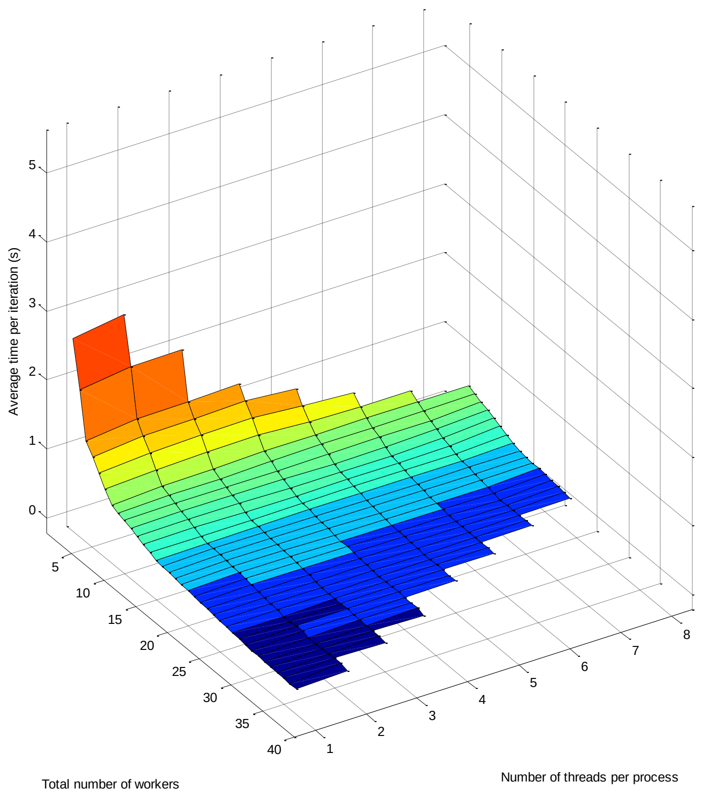

FIGURE 6 - AVERAGE COMPUTING TIME PER ITERATION FOR DIFFERENT NUMBER OF WORKERS AND THREADS PER PROCESSES FOR AN INPUT OF 9,980,928 PIXELS

As shown in Figure 5 and Figure 6, the algorithm adapts very well to both parallelization approaches. Although these approaches are fundamentally different and use completely different technologies and different hardware repartition, it seems there effect is similar: only the total number of "workers" seems to influence the computation time.

d. Scalability

To determine whether an algorithm as a good scalability, we compare its duration of execution as a function of the number of processes with the values provided by Amdahl's law, that is, with t_n the duration for n processes and P the proportion of the program that can be made parallel:

$$\frac{t_n}{t_1} = \frac{1}{(1-P) + P/n}$$

We use the first two measured durations in order to evaluate P, according to the following formula with m and n being two number of processes for which the duration is known:

$$P = \frac{t_m/t_n - 1}{(t_m/t_n - 1)(1 - 1/n) + (1/m - 1/n)}$$

As suggested in 0, the algorithm seems almost entirely parallelizable: the estimate values for P are very close to 1.

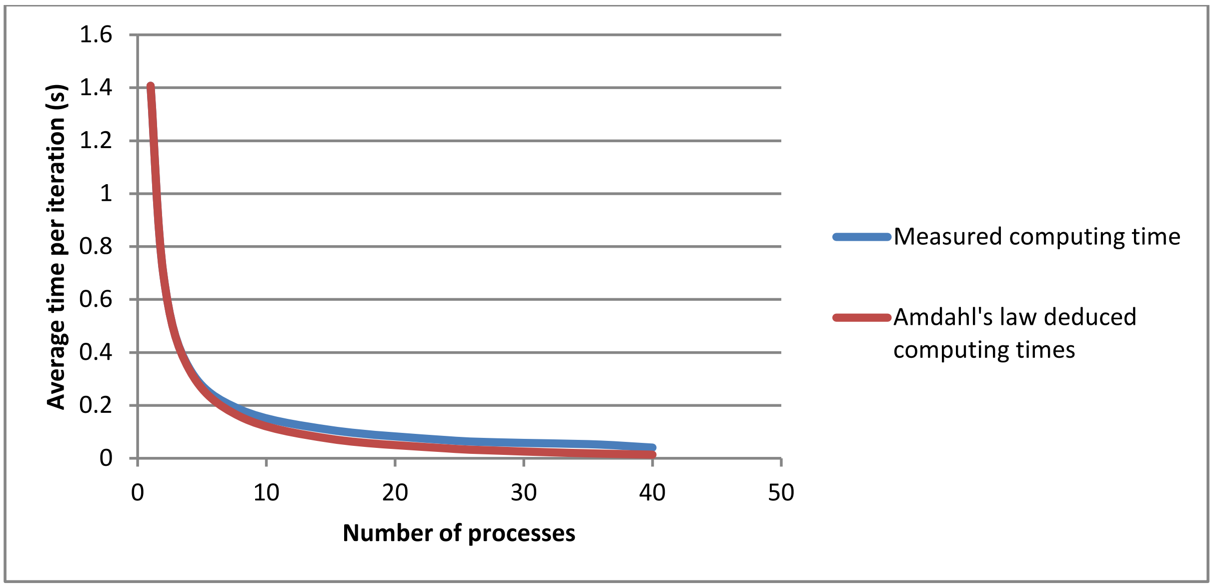

FIGURE 7 - MEASURED AND EXPECTED COMPUTATIONAL DURATIONS FOR DIFFERENT NUMBERS OF PROCESSES WITH ONE THREAD PER PROCESS FOR AN INPUT OF 2,795,520 PIXELS

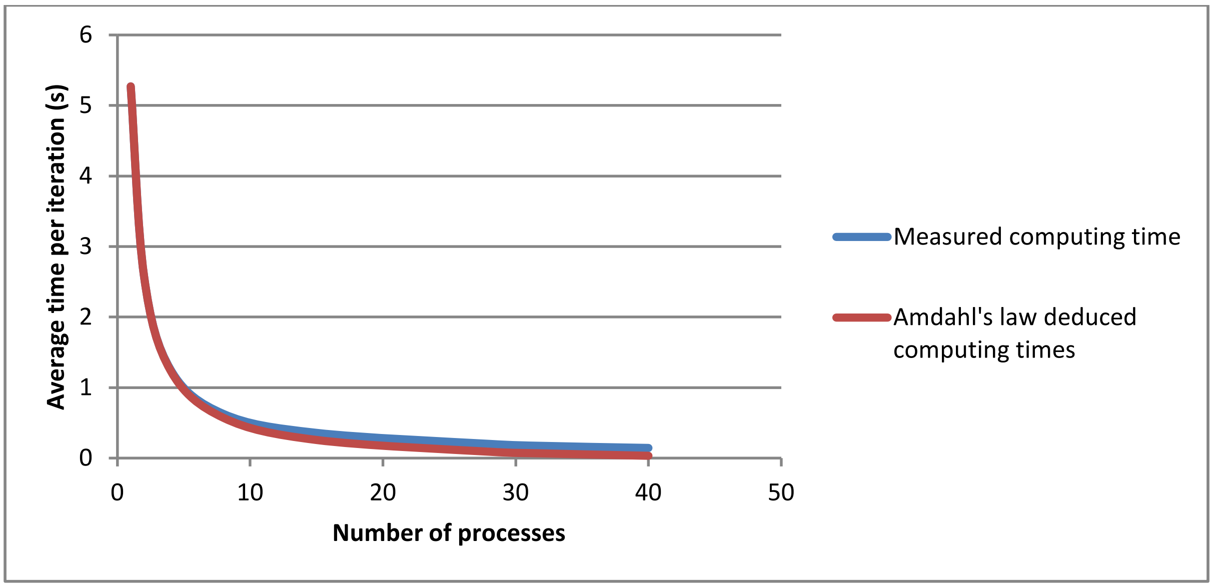

FIGURE 8 - MEASURED AND EXPECTED COMPUTATIONAL DURATIONS FOR DIFFERENT NUMBERS OF PROCESSES WITH ONE THREAD PER PROCESS FOR AN INPUT OF 9,980,928 PIXELS

As shown on Figure 7 and Figure 8, the algorithm scales very well and its computational time is only slightly above Amdahl’s law predicted time. However the shift between both seems to remain constant, which indicates a reliable scalability for larger number of processes.

4. Conclusion

The algorithm is highly parallelizable and scales very well. Moreover, both parallelization strategies work as well although they are of very different natures.

Bibliography

1. **Weickert, Joachim.** *Anisotropic Diffusion in Image Processing.* s.l. : B.G. Teubner Stuttgart, 1998.

Source code