

Computational Acceleration of Image Inpainting Alternating-Direction Implicit (ADI) Method Using GPU CUDA

Mutaqin Akbar

Magister Teknik Informatika
Universitas Atma Jaya Yogyakarta
Yogyakarta, Indonesia
mutaqin.akbar@gmail.com

Pranowo

Magister Teknik Informatika
Universitas Atma Jaya Yogyakarta
Yogyakarta, Indonesia
pran@mail.uajy.ac.id

Suyoto

Magister Teknik Informatika
Universitas Atma Jaya Yogyakarta
Yogyakarta, Indonesia
suyoto@mail.uajy.ac.id

Abstract—This paper presents a computational acceleration of image inpainting using parallel processing based on Graphics Processing Unit (GPU) Compute Unified Device Architecture (CUDA). We use parabolic partial differential equation (PDE) called heat equation as the model equation. The heat equation is discretized numerically using Finite Difference method. Semi-algebraic equation that formed then solved by using Alternating-Direction Implicit (ADI) scheme. The numerical algorithm is implemented in GPU CUDA parallel computing to speed up the computational time. The computational process of the inpainting can be done using larger time-step. The computational time can be accelerated to 5.86 times faster using an image with 2736x1824 resolution.

Keywords—image inpainting, PDE, ADI, parallel computing, GPU CUDA

I. INTRODUCTION

The field of digital image processing refers to processing digital images by means of a digital computer [1]. Some fields in digital image processing are image enhancing, image filtering, edge detection, and the newest one is image inpainting. Inpainting is a term that begun from art literature. Long time ago inpainting is done by an artist manually, to achieve its main goal which is restoring damaged area on a painting to looks like it first created.



Fig 1. Inpainting example, the damaged painting (top-left) and the inpainting result (bottom-right) (<https://projects.library.villanova.edu/paintingrestoration/>).

This technique is then brought into digital scope [2]. Different from conventional inpainting that needs thoroughness from an artist, digital inpainting use information from the surrounding of damaged area to fill in the damaged area. This process is done automatically, where user just needs to mark the area to be inpainted.

The technique fills in damaged area by propagating the edges into the damaged target area through the diffusion process of partial differential equations of physical heat flow. In this technique, there are 2 schemes presented, explicit scheme and implicit scheme. The explicit scheme first done by Bertalmio et al [2]. Implicit scheme have advantages which are resulting unconditionally stable [3] and convergent solution although using a larger time-step. Somehow, the implicit scheme's solution procedure is very time-consuming. Alternating-Direction Implicit (ADI) method can overcome this by splitting the solution [4].

Other technique of partial differential equations image inpainting is by implementing phase-field model based on Cahn-Hilliard equation. This technique first done by Bertozzi et al [5] and then updated by Darae Jeong et al [6] and Schonlieb [7].

All techniques above require long computational time done by a sophisticated Central Processing Unit (CPU). The use of GPU for general purpose can tackle this obstacle. Past research by Prananta prove that GPU can accelerate fourth order PDE equation Image Inpainting [8].

This paper presents the implicit scheme of heat equation which is discretized using Finite Difference method. Alternating-Direction Implicit (ADI) scheme is used to solved Semi-algebraic equation that formed. Parallel computing is implemented to speed up the computational time. Parallel computing on GPU is implemented using CUDA, an Application Programming Interface (API) developed by NVIDIA.

II. EQUATION, DISCRETIZATION, AND GPU IMPLEMENTATION

We use Crank-Nicolson method to solve heat equations. By using the Finite Difference approach, the images pixel can be associated to finite difference grids and intensity of image is

associated to temperature. Images used in this paper is grayscale images with various resolutions. Generally, The heat equation for the inpainting process can be written as follows:

$$\frac{\partial I}{\partial t} = \alpha \left(\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \right) + \lambda(f - I) \quad (1)$$

$$I^{n+1}_{(i,j)} = I^n_{(i,j)} + \Delta t \cdot I_t^n_{(i,j)}, \forall (i,j) \in \Omega \quad (2).$$

Image intensity denoted as $I_{(i,j)}$, where (i,j) are pixel coordinates and I is the intensity of current pixel coordinate, α is a constant value, t is time-step, x and y are space-step in x (abscissa) and y (ordinate), λ is mask image, and f is damaged image.

A. Implicit Scheme

For the 2 dimensional space, the Crank-Nicolson equation can be written below [9]:

$$\frac{I^{n+1}_{(i,j)} - I^n_{(i,j)}}{\Delta t} = \frac{\alpha}{2} \left(\frac{I^{n+1}_{(i-1,j)} - 2I^{n+1}_{(i,j)} + I^{n+1}_{(i+1,j)}}{\Delta x^2} + \frac{I^{n+1}_{(i,j-1)} - 2I^{n+1}_{(i,j)} + I^{n+1}_{(i,j+1)}}{\Delta y^2} + \frac{I^n_{(i-1,j)} - 2I^n_{(i,j)} + I^n_{(i+1,j)}}{\Delta x^2} + \frac{I^n_{(i,j-1)} - 2I^n_{(i,j)} + I^n_{(i,j+1)}}{\Delta y^2} \right) \quad (3)$$

To obtain a direct solution to this scheme, it will be difficult and computationally inefficient. So the usage of Alternating-Direction Implicit (ADI) method is well-known to overcome the difficulty of this scheme. ADI method known to its better performance compared to other implicit methods. Well defined and structured access pattern can be achieved by splitting the problem which changes the sparse matrix memory access pattern. [10]

B. Alternating-Direction Implicit (ADI)

The main idea of the ADI for the 2 dimensional problem is to split the computations in two steps. In the first step, we sweep through x -direction by applying an implicit method in the x -direction and an explicit method in the y -direction, producing an intermediate solution for time. In the second step, we sweep through y -direction by applying an implicit method in the y -direction and an explicit method in the x -direction.

So ADI implementation of Crank-Nicolson's solution [4] discretized below and the illustration shown in Fig. 2.

$$\frac{I^{n+\frac{1}{2}}_{(i,j)} - I^n_{(i,j)}}{\Delta t} = \frac{\alpha}{2} \left(\frac{I^{n+\frac{1}{2}}_{(i-1,j)} - 2I^{n+\frac{1}{2}}_{(i,j)} + I^{n+\frac{1}{2}}_{(i+1,j)}}{\Delta x^2} + \frac{I^n_{(i,j-1)} - 2I^n_{(i,j)} + I^n_{(i,j+1)}}{\Delta y^2} \right) \quad (4)$$

$$\frac{I^{n+1}_{(i,j)} - I^{n+\frac{1}{2}}_{(i,j)}}{\Delta t} = \frac{\alpha}{2} \left(\frac{I^{n+\frac{1}{2}}_{(i-1,j)} - 2I^{n+\frac{1}{2}}_{(i,j)} + I^{n+\frac{1}{2}}_{(i+1,j)}}{\Delta x^2} + \frac{I^{n+\frac{1}{2}}_{(i,j-1)} - 2I^{n+\frac{1}{2}}_{(i,j)} + I^{n+\frac{1}{2}}_{(i,j+1)}}{\Delta y^2} \right) \quad (5)$$

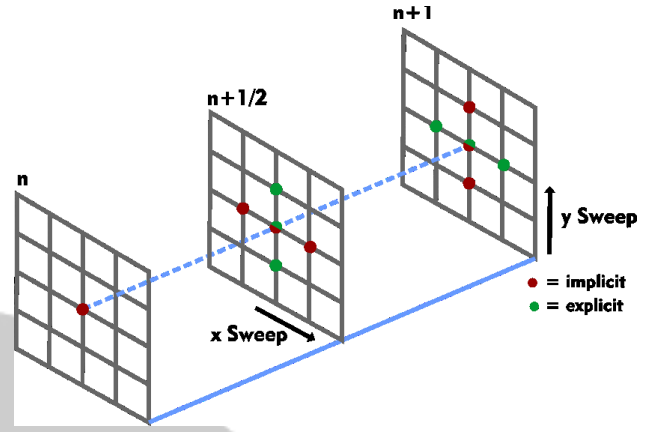


Fig 2. ADI illustration in the grid system.

The ADI solution to heat equation will form two tridiagonal matrix system, which can be solved efficiently using Thomas's algorithm.

C. GPU and CUDA

GPUs are first-known as graphics accelerators. Recently demand of processing large number of data (or even big data) makes GPUs evolved to be powerful, general-purpose, fully programmable, task and data parallel processors, ideally suited to tackle massively parallel computing problems. The use of GPUs is already embraced by many scientific field, such as fluid-dynamics, aero-dynamics, artificial intelligence, machine learning/deep learning, image processing, etc.

Basically, data is stored one-dimensionally. Even when a logical multi-dimensional view of data is used, it still maps to one-dimensional physical storage. CPU processes data sequentially from the first until the end of data (finished one data to execute the next data). While GPU can allocate every threads it has to processes every single one of data at the same time, depends on the capability of the GPU itself.

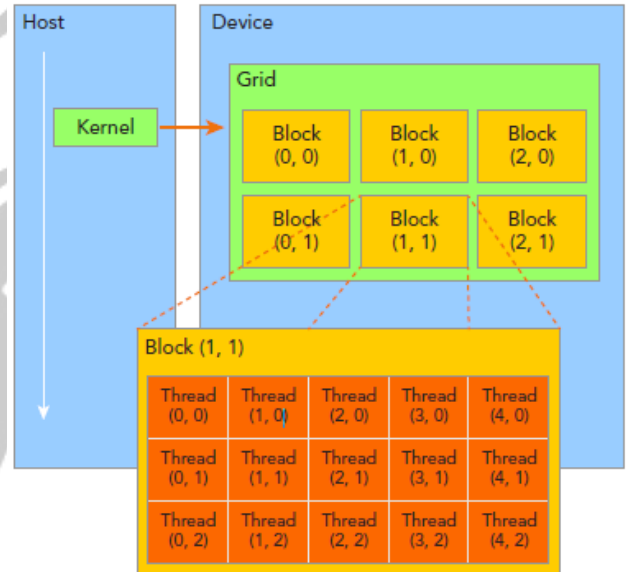


Fig 3. Grid, Block, and Thread in a GPU.

CUDA is a general-purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way [11].

D. Inpainting Flowchart

Fig. 4 describes the algorithm of the inpainting process using GPU CUDA implementation.

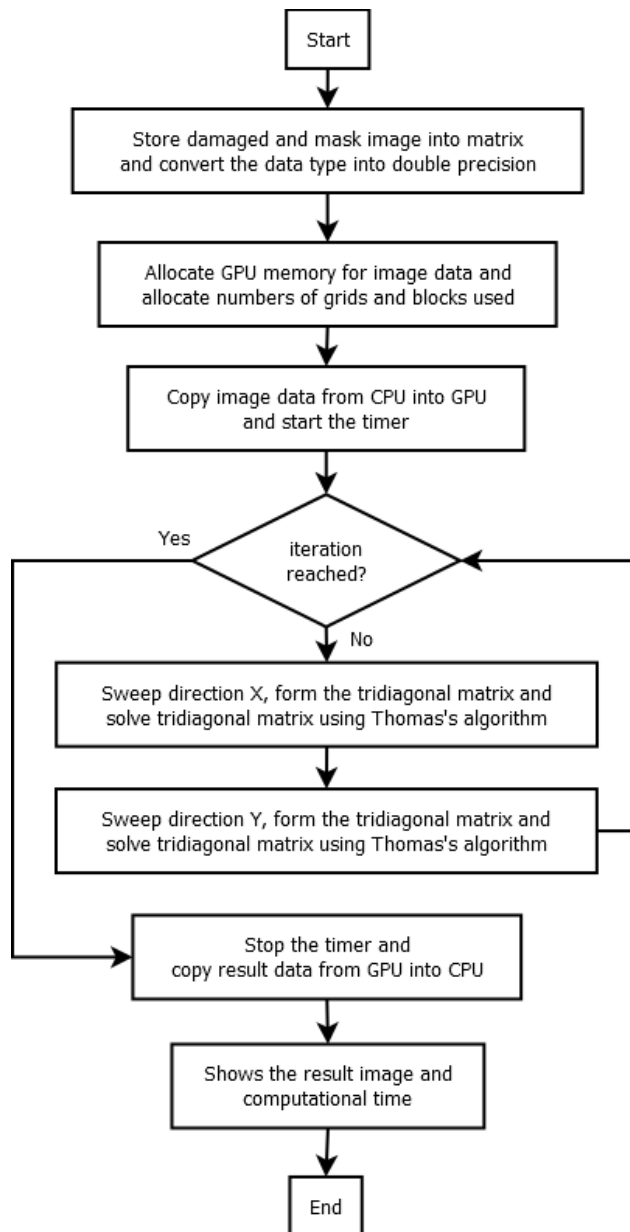


Fig 4. Inpainting Based on GPU Flowchart.

III. RESULTS AND ANALYSIS

Simulation has been done using a personal computer with specification described below:

- CPU Intel(R) Core(TM) i7-3770K @3.50GHz
- GPU GeForce GTX660Ti 2GB RAM
- RAM 16GB

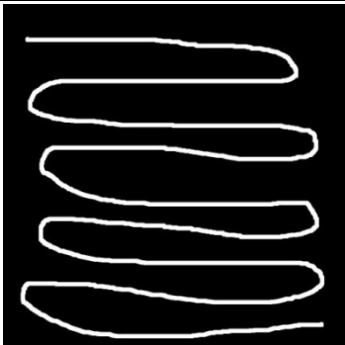
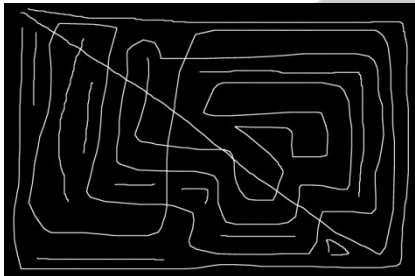
Images used are grayscale images with various resolutions, shown in Table I and the mask images shown in Table II.



TABLE I. IMAGES USED IN SIMULATIONS

Image Resolution	Damage Image
483 x 405	
1024 x 1024	
1532 x 1021 2189 x 1459 2736 x 1824	

TABLE II. MASK IMAGES USED IN SIMULATIONS

Image Resolution	Mask Image
483 x 405	

1024 x 1024	
1532 x 1021 2189 x 1459 2736 x 1824	

1024 x 1024	
1532 x 1021 2189 x 1459 2736 x 1824	

Restoration rate (α) used is 1.0 and time-step (Δt) equals 0.5. Table III below, shows the results of image inpainting in 10th iteration and Table IV shows the results for 50th iteration. In the 10th iteration the result images are not yet inpainted and in the 50th iteration, the solution is already convergent. In the girls and cars image, the result image are well-inpainted. But in the lena image, the result is not as good as two others image, even though the solution is already convergent.

TABLE III. INPAINTING RESULTS FOR 10TH ITERATION




Image Resolution	10 th Iteration
483 x 405	

TABLE IV. INPAINTING RESULTS FOR 50TH ITERATION

Image Resolution	50 th Iteration
483 x 405	
1024 x 1024	


1532 x 1021 2189 x 1459 2736 x 1824	
---	---

TABLE V. THE PERFORMANCE OF CPU AND GPU

Image Resolution	Number of Iterations	CPU Time (ms)	GPU Time (ms)	Speed Up (x)
483 x 405	50	26224	15318	1.71
1024 x 1024	50	146343	39481	3.70
1532 x 1021	50	211692	54491	3.88
2189 x 1459	50	437023	85997	5.08
2736 x 1824	50	685278	116807	5.86

By comparing the result done by computing on CPU and GPU, we can say that the bigger the resolution of a given image used, the bigger the speed up achieved. The speed up for GPU implementation reaches 5.86 when the image resolution is 2736x1824.

IV. CONCLUSION

This paper presents a computational acceleration of image inpainting using parallel computing based on GPU CUDA. Using Crank-Nicolson's solution to heat equation and ADI's solution to two-dimensional problem, the computational process of image inpainting can be done using larger time-step with stable and convergent result. The computational process time can be accelerated (speed up) using CUDA implementation to 5.86 times faster using an image with 2736x1824 resolution.

REFERENCES

- [1] Gonzales R C, Woods R E. Digital Image Processing. 3rd Edition. New Jersey: Pearson Education, Inc. 2008; 1-2.
- [2] Bertalmio M, Sapiro G, Caselles V, & Ballester C. Image Inpainting. SIGGRAPH ACM. 2000; 417-424.
- [3] Fjelland B. Thesis on applications of the Alternating Direction Implicit method. Thesis. Copenhagen: Copenhagen Business School; 2012.
- [4] Hoffmann K A, Chiang S T. Computational Fluid Dynamics. Fourth Edition. Kansas: A Publication of Engineering Education System. 2000; 76-80.
- [5] Bertozzi A L, Esedoglu S, Gillette A. Inpainting of Binary Images Using the Cahn-Hilliard Equation. IEEE Transactions on Image Processing. 2007; 16(1): 285-291.
- [6] Darae J, Yibao L, Hyun G L, Junseok K. Fast and Automatic Inpainting of Binary Images Using a Phase-Field Model. J. KSIAM. 2009; 13(3): 225-236.
- [7] Schonlieb C-B. Modern PDE Techniques for Image Inpainting. Thesis. Cambridge: University of Cambridge; 2009.
- [8] Prananta E, Pranowo, Budianto D. GPU CUDA Accelerated Image Inpainting using Fourth Order PDE Equation. TELKOMNIKA. 2016; 14(3): 1009-1015.

- [9] Araujo A, Neves C, Sousa E. An alternating direction implicit method for a second-order hyperbolic diffusion equation with convection. Elsevier. Applied Mathematics and Computation 239. 2014; 17-28.
- [10] Laszlo E. Parallelization of Numerical Methods on Parallel Processor Architectures. Thesis. Hungary: Pázmány Péter Catholic University; 2016.
- [11] Cheng J, Grossman M, McKercher T. Professional CUDA C Programming. Indiana: John Wiley & Sons, Inc. 2014; 14-31.