TELKOMNIKA, Vol.14, No.3, September 2016, pp. 1009~1015
ISSN: 1693-6930, accredited A by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v14i3.3412

GPU CUDA Accelerated Image Inpainting using Fourth Order PDE Equation

Edwin Prananta*[1], Pranowo[2], Djoko Budianto[3]
Departement of Information Technology, Faculty of Industrial Technology,
University Atma Jaya Yogyakarta, Jln Babarsari no 44, Yogyakarta, DIY, Indonesia
*Corresponding author, e-mail: edwinprananta@gmail.com[1], pran@mail.uajy.ac.id[2],
djoko.bdy@gmail.com[3]

Abstract

This paper describes the technique to accelerate inpainting process using fourth order PDE equation using GPU CUDA. Inpainting is the process of filling in missing parts of damaged images based on information gleaned from surrounding areas. It uses the GPU computation advantage to process PDE equation into parallel process. Fourth order PDE will be solved using parallel computation in GPU. This method can speed up the computation time up to 36x using NVDIA GEFORCE GTX 670.

***Keywords**: Inpainting, PDE, GPU-CUDA, parallel, computation*

1. Introduction

The Inpainting problem is the term for what researchers in image processing call “image interpolation”. Digital Image restoration technique using information gleaned from surrounding areas of image to fill the damages or missing parts of image is called inpanting. The origin of inpainting was begun in art world, it was used in restoration old oil painting. This term was firstly introduced into digital image processing in the work of Bertalmio, et al., [1].

There are many applications of image inpainting. It can be used to remove scratches automatically from digital image or film, digital restoration of ancient painting, text erasing, and connect roads in satellite image [2].

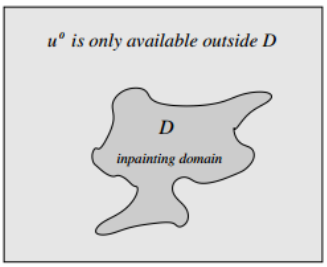


Figure 1. Inpainting Problem

In 1984, Geman and Geman used statistical approach for image restoration tasks [3]. This approach was to remove noise in the image, if the image structure was missing, this approach would be difficult to give fair result. The work of Bertalmio, et. al., introduced image inpainting as a new research area of digital image processing [1]. This work not only can remove noise, but also can repair structure of the image. The model is based on nonlinear PDE. When the damage area has large distances and complicated texture, this method will have difficulties to repair image.

A different approach to inpainting was proposed by Chan and Shen [4]. They introduced the idea that well-known variational image denoising and segmentation models can be easily

Received January 21, 2016; Revised May 16, 2016; Accepted June 3, 2016

adapted to the inpainting task by a simple modification. The model can successfully propagate sharp edges into the damaged domain. However, this model exacts a penalty on the length of edges, this technique cannot connect contours across large distances. One year after that, 2002, Chan, Kang, and Shen introduced a new variational image inpainting model that addressed the shortcomings of the the total variation [5]. This method makes improvement in inpainting longer distances than before. Esedoglu and Shen adapted image segmentation to the inpainting problem [6]. This model can be solved rather quickly but this model cannot connect image across large distance.

In 2007, modified Cahn Hilliard being used for inpainting. It shown that this method give fast inpainting of binary images [2]. In 2009, Schönlieb, make research in modern PDE inpainting [7]. This research show us the advantage of fourth-order inpainting models over models of second differential order is in the smooth continuation of image contents even across large distance in the image. The research emphasise to make more research about faster numerical solvers for inpainting with higher-order equations than before.

The most recent approach to inpainting is based on fourth order PDE method [8]. There are lots of researches trying to improve the quality and to reduce processing time of inpainting process using high order PDE method [5, 8]. A number of methods were suggested to solve inpainting problem, many of which are based on advanced mathematical techniques [1, 2], [4-6], 8, 9]. This research is trying to accelerating inpainting process using different approach than advance mathematical techniques.

High performance computing on GPU using CUDA library will be used to solve image Inpainting problem. Parallelized execution of high order PDE in image Inpainting would be a feasible optimization method. We will take the advantages of GPU multicore processor to increase the application performance by executing them in GPU cores [10]. The research of Guo and He, is one of the example that the use of GPU has been significantly increase the the lattancy time in digital image processing. They can speedup fractal image compresion process to 123 times faster compare to the traditional method [11]. That research gives an opportunity another image processing algoritm to use the power of GPU to speedup computation process.

This research will answer how significant the uses of GPU to increase the lattecy time in digital image inpainting using fourth order PDE.

2. Inpainting Method

Inpainting digital image is essentially interpolation process on filling missing parts of an image based on surrounding areas. The solution of this problem is using fourth order PDE equation. The inpainting problem is to to reconstruct the original image u in (damaged) domain $D \subset \Omega$ (Figure 1). Image f represent some given image defined on an image domain Ω. The discretization in space used finite difference and spectral methods, i.e., the fast Fourier transform, to simplify the inversion of the Laplacian Δ for the computation of u_{k+1}. The fourth order PDE inpainting equation:

$$u_t = \Delta\left(-\epsilon\Delta u + \frac{1}{\epsilon}F'(u)\right) + \lambda(f-u)\,, in\ \Omega, \quad (1)$$

The Equation (1) is the work of Betrozzi, et al., using the fourth order PDE to solve inpainting problem [2]. Based on the work of Equation (1), Schönlieb, et al., [9] built another fourth order PDE equation:

$$\frac{u_{k+1}-u_k}{\Delta t} + C_1\Delta\Delta u_{k+1} + C_2 u_{k+1} = C_1\Delta\Delta u_{k+1} - \Delta\left(\nabla.\left(\frac{\Delta u_k}{|\Delta u_k|}\right)\right) + C_2 u_k + \lambda(f-u) \quad (2)$$

The constants C_1 and C_2 are positive, and need to be chosen large enough to make this equation convex. The constants C_1 and C_2 must fulfill the condition of $C_1>1/\epsilon$, $C_2>\lambda\_0$.

This equation has been proved by Schönlieb, et al., [9] have consistency, unconditional stability, and convergence.

3. GPU CUDA

Commodity computer graphics chips, known generically as Graphics Processing Units or GPUs, are probably today's most powerful computational hardware for the dollar. Researchers and developers have become interested in harnessing this power for general-purpose computing, an effort known collectively as GPGPU (for "General-Purpose computing on the GPU") [10].

CUDA is a parallel programming model developed by NVIDIA was started at 2006. The first CUDA SDK was released in the early 2007. A parallel system using CUDA consists of a host (CPU) and a device (GPU). The computation of tasks is done in GPU by a set of threads that run in parallel. The GPU architecture for threads consist of two-level hierarchy, namely block and grid (Figure 2). Block is a set of tightly coupled threads where each thread is identified by a thread ID, while grid is a set of loosely coupled of blocks with similar size and dimension [10].

The usage of GPU shows improvement performance in wide area such as Physical based simulation, signal processing, data processing, image segmentation, computer vision and image processing [10]. An example of the usage GPU for image processing is the work of Guo and Hei, they worked fractal image compression in GPU [11]. Their work can accelerate 123 times faster than before. Not only in image processing the usages of GPU become popular, but also in data processing. It shown in the work of Xu and Xu, they worked a hybrid shorting algoritm with the power of GPU and CPU. This work can short one billion 32-bit float in no more than 5 seconds [12].

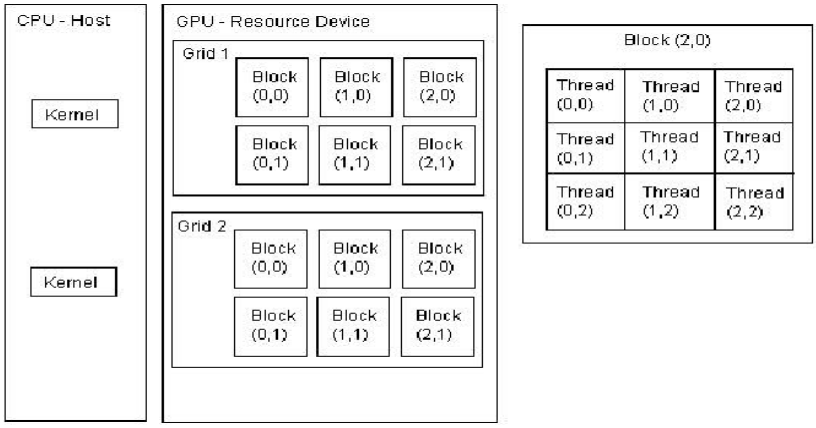


Figure 2. CUDA Architecture

4. GPU Combined Parallel Image Inpainting

The acceleration inpainting process is a challagne to researchers. They are trying to use advance matemathical teqniques to improve the computation time but there another way to accelerate inpainting process using parallel execution in GPU. We realize that the use of GPU in inpainting problem will reduce the computation latancy time.

Image source and mask inpainting should be inisialize in the CPU memory. In GPU memory, data only can be process in one dimensional array. Image data should be process to one dimensional array data. We will assign image soruce and mask to variable that can be transfer to GPU memory.

The threads of a block are processed in the same streaming processor (SM), and the parallelization depends on the number of cores in the SM. The GPU architecture has a limit in the number of threads per block. In this work, a NVIDIA GeForce GTX-670 card with up to 1,024 threads per block was used.

The computations of fast Fourier transform use the CUDA library. The step to call CUFFT library is: create plan, execute plan and destroy plan. The computations need double presicion so we will use parameter CUFFT_Z2Z, which indicate double complex to double complex computation.

Step 1 :	Image and mask inisialization
Step 2 :	Preprocessing data to transfer GPU
Step 3 :	Computation Variable inisialization
Step 4 :	Transfer data from CPU to GPU Memory
	---- Begin parallel data computation in GPU ----
Step 5 :	GPU variable and workload inisialization
Step 6 :	Inizialisation Calculation Parameter
Step 7 :	Parallel image and mask FF transform
Step 8 :	Parallel calculation curve
Step 9 :	Paralel curve FF transform
Step 10 :	Parallel Fourth Order PDE Inpainting in spectral domain
Step 11	Parallel Inverse FF transform of inpainting result
Step 12 :	Inisialisation new curve
	Step 9-12 will be repeated until fair image condition
Step 13 :	Transfer data from GPU to CPU Memory
Step 14 :	Celan GPU Memory
	---- End parallel data computation in GPU ----
Step 15 :	Preprocessing data to CPU
Step 16 :	Show and save image *Inpainting Result*

Figure 3. Algoritm parallel Inpainting forth order PDE in GPU

We use the data variable from the research of Schönlieb, et al., [9]. The C1 variable is set to 200 and C2 variable is 100. We save this variable in register memory of GPU. The curve calculation result and the fourth order inpainting result will be saved in global memory.

After the iteration complete, the GPU will begin to transfer the data to CPU. The data will be process from the one dimensional array to two dimensional image array.

5. Result Inpainting Fourth Order GPU

The proposed method was simulated with an image of white rectangular on black background having the damage in the middle of the image. The damage area had gray colour. This image resolution was 100 x 100 pixels. The image test can be seen in Figure 3(a) while the required result should be seen as in Figure 3(f). This damage area can be easily seen in Figure 3(a) with gray colour in the middle of the image. This initial test image had high MSE (9,795.40), low PSNR (8.22) and low SSIK (0.81). These indicators indicated that the initial image had high damage and small similarity.

Table 1. Latency *Inpainting* process CPU and GPU

	Latency (sec)		
Iterations	CPU	GPU	Speedup
10,000	155.99	4.31	36.19
50,000	780.00	21.65	36.03
100,000	1,560.16	43.37	35.98
200,000	3,120.28	86.50	36.07

The latency of inpainting process using fourth order PDE has been noted both in CPU and GPU to repair the damage area. As shown in Table 1, the CPU computation time was slower than GPU computation time. The computation time for 10,000 iterations in CPU was 155.99s and in GPU was 4.31s. These were the computation time for 100,000,000 data (100 x 100 x 10,000 iterations). The computation time for 200,000 iterations in CPU was 3,120.28s and in GPU was 86.50s. These were the computation time for 2,000,000,000 data (100 x 100 x 200,000 iterations). The parallel computation using GPU was showing 36x latency speedup than using CPU.

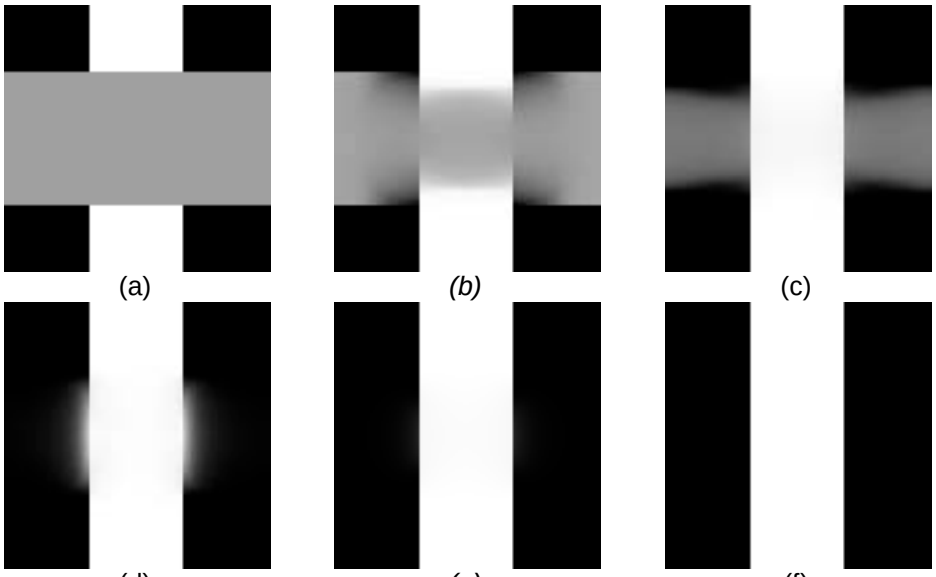


Figure 4. (a)Test image with damage in the middle (b). Inpainting 10.000 iterations (c) Inpainting 50.000 iterations (d). Inpainting 100.000 iterations (e). Inpainting 200.000 iterations (f) Test image without damage

The GPU have been applied to solve inpainting problem using fourth order PDE equation in Equation (2). The result of the 10,000 iterations is shown in Figure 3(b). This image was trying to reconstruct the white rectangular in the middle of the image. In 10,000 iterations, the rectangular almost connected. In 50,000 iterations as shown in Figure 3(c), the white rectangular already connected and the black background was reconstructed. After 100,000 iterations, we can see only small area of black background that hasn't repaired like in Figure 3(d). In 200,000 iterations, Figure 3(e) already has the same structure like in Figure 3(f). This is proved by the SSIK value of 0.98.

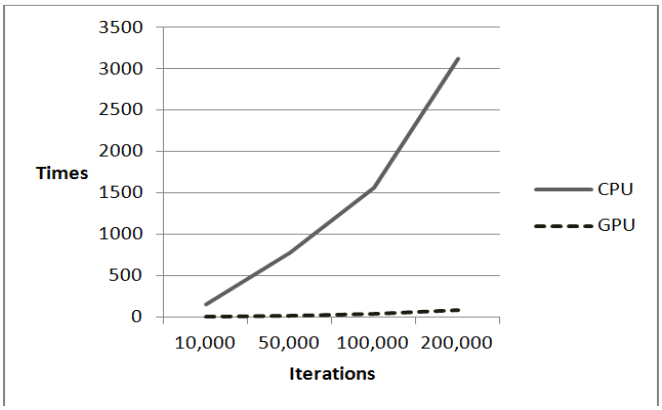


Figure 5. Line chart of latancy comparison GPU and CPU

Table 2. Image Quality Metric

Iterations	MSE	PSNR	SSIK
0	9,795.40	8.22	0.81
10,000	7,628.24	9.30	0.74
50,000	2,663.89	13.88	0.84
100,000	663.59	19.91	0.88
200,000	29.25	33.47	0.98

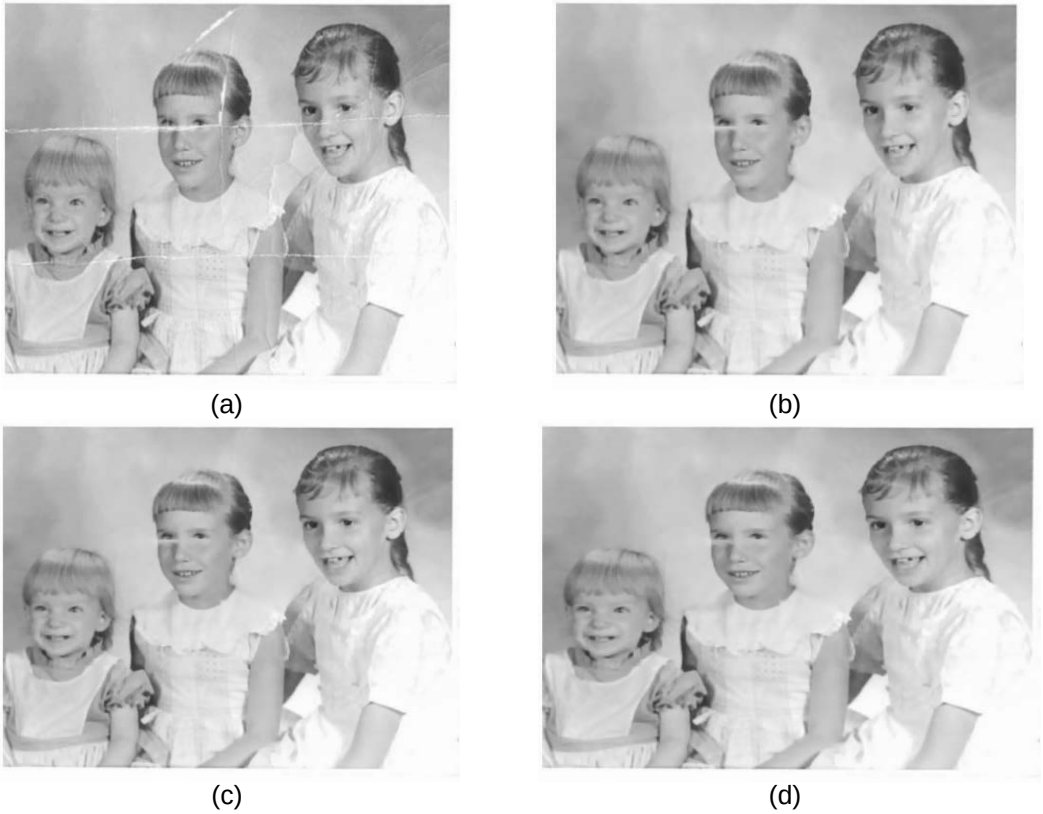
The outcome of inpaiting process using fourth order PDE as simulated by the authors is shown in Figure 3(e) and the required result was shown is Figure 3(f). It was found that the iterations in GPU and the quality of the image are correlated. If we increase the iteration, the quality of the image will increase. It was shown in Table 2, that after 200,000 iterations the MSE decreased from 9,795.40 to 29.25 ,PSNR increased from 8.22 to 33,47 and SSIK increased from 0.81 to 0,98.

We use sample image to test the real problem reparing the damage picture. We used the three girls old grayscale portrait that have folded damage like in Figure 6(a). This picture dimension is 483 x 405 pixels. We are running the algoritm execution in both CPU and GPU.

Table 3. Latency *Inpainting* old portrait latancy CPU and GPU

	Latency (sec)		
Iterations	CPU	GPU	Speedup
500	124,80	2,58	48,39
1.000	249,61	5,12	48,76
2.000	499,29	10,21	48,91

Similar from the simulation before, the CPU computation time was slower than GPU computation time. The computation time for 500 iterations in CPU was 124,80 s and in GPU was 2,58s. These were the computation time for 97.807.500 data (483 x 405 x 500 iterations). The computation time for 1.000 iterations in CPU was 249,61s and in GPU was 5,12s. These were the computation time for 195.615.000 data (483 x 405 x 1.000 iterations). The computation time for 2.000 iterations in CPU was 499,29s and in GPU was 10,21s. These were the computation time for 391.230.000data (483 x 405 x 2.000 iterations). The parallel computation using GPU was showing 48x latency speedup than using CPU.



(a) (b)

(c) (d)

Figure 6. *(*a) Damage picture; (b) Inpainting 500 iterations; (c) Inpainting 1.000 iterations; (d) Inpainting 2.000 iterations

The GPU have been applied to solve inpainting problem in Figure 6(a) using fourth order PDE equation in Equation (2). The result of the 500 iterations was shown in Figure 6(b). This image was trying to repair small damage in the body and the background, but we can still see the damage in the area around the girl eyes. In 1.000 iterations, the damage in the girl eye reduce this was shown in Figure 6(c). But the face line of the giril is not connected yet. In 2.000 the face line around the girls eye already connected like was shown in Figure 6(d).

6. Conclusion

In this paper, the authors presented a technique for accelerating inpainting problem using GPU as computation tools. This method aims to reduce the heavy computation problem of fourth order PDE and to employ the use of GPU in parallelism to gain computation speed. This improvement was accomplished by using NVDIA GEFORCE GTX 670 to calculate data in parallel computation. The proposed concept proved a speedup to 36x in the simulation model and speedup to 48x in the simulation picture. It was also shown that the use of GPU CUDA can be applied in the image inpainting area.

References

[1] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, Coloma Ballester. *Image inpainting.* Proceedings of the 27th annual conference on Computer graphics and interactive techniques. 2000: 417-424.
[2] Andrea L Bertozzi, Selim Esedoglu, Alan Gillette. Inpainting of binary images using the Cahn-Hilliard equation. *IEEE Transactions on image processing*. 2007; 16(1): 285-291.
[3] Stuart Geman, Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on.* 1984; 6: 721-741.
[4] Tony Chan, J Shen. Mathematical models for local nontexture inpaintings. *SIAM Journal on Applied Mathematics*. 2001; 62(3): 1019-1043.
[5] Tony F Chan, Sung Ha Kang, Jianhong Shen. Euler's elastica and curvature-based image inpainting. *SIAM Journal on Applied Mathematics*. 2002; 63(2): 564-592.
[6] Selim Esedoglu, Jianhong Shen. Digital inpainting based on the Mumford–Shah–Euler image model. *European Journal of Applied Mathematics*. 2002; 13(4): 353-370.
[7] Carola-Bibiane Schönlieb. Modern pde techniques for image inpainting. Doctoral dissertation. University of Cambridge; 2009.
[8] Jessica Bosch, David Kay, Martin Stoll, Andrew J Wathen. Fast Solvers for Cahn--Hilliard Inpainting *SIAM Journal on Imaging Sciences*. 2014; 7(1): 67-97.
[9] Carola-bibiane Schönlieb, Andrea Bertozzi. Unconditionally stable schemes for higher order inpainting. *Communications in Mathematical Sciences*. 2011; 9(2): 413-457.
[10] John D Owens, et al. A Survey of general-purpose computation on graphics hardware. *Computer graphics forum*. 2007; 26(1): 80-113.
[11] Hui Guo, Jie He. A Fast Fractal Image Compression Algorithm Combined with Graphic Processor Unit. *TELKOMNIKA Telecommunication Computing Electronics and Control.* 2015; 13(3): 1089-1096.
[12] Ming Xu, Xianbin Xu, Fang Zheng, Yuanhua Yang, Mengjia Yin. A Hybrid Sorting Algorithm on Heterogeneous Architectures. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2015; 13(4): 1399-1407.