Image Filtering on .NET-based Desktop Grids

Christian Vecchiola[1], Krishna Nadiminti[2] and Rajkumar Buyya[2]

[1]DIST – Department of Communication Computer and System Sciences
The University of Genoa, Italy
Email: christian@dist.unige.it

[2]**Gri**d Computing and **D**istributed **S**ystems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email: {kna, raj}@csse.unimelb.edu.au

Abstract

Image filtering is the use of computer graphics algorithms to enhance the quality of digital images or to extract information about their content. However rendering very large size digitial images on a single machine is a performance bottleneck. To address this we propose parallelising this application on a desktop Grid environment. For parallelizing this application we use the Alchemi Desktop Grid environment and the resulting framework is referred to as ImageGrid. ImageGrid allows the parallel execution of linear digital filters algorithms on images. We observed acceptable speed up as the result of parallelising filtering through ImageGrid. We run the tests on different data sets by varying the dimension of the images and the complexity of the filters. Results demonstrate potential of Grid computing for desktop applications and that the speed up obtained is more consistent for large images and complex filters.

1. Introduction

Digital image processing [1] has nowadays become a common activity for every kind of users. If we went out with your digital camera we will most likely to have hundreds of pictures whose size is normally about 3 megabytes; this means that if we want to retouch or adjust them we will have to deal with hundreds of megabytes. If we consider image processing for scientific purposes we will have terabytes of data and, probably, days of processing time. Digital image management plays an important role in astronomy (earth observation, space probes) and medicine (medical imaging as Magnetic Resonance Imaging and microscopy): in these cases large datasets of huge images are produced daily. For example radar images are normally 25K x 5K pixels while microscope images can range from 40K x 40K to 100K x 100K pixels. This means having to process images whose size ranges from hundreds of megabytes to gigabytes. The common tasks performed on these images range from image enhancement to features extraction and content retrieval and they basically rely on some sort of image filtering. Image filtering is a CPU intensive task and processing images of the above dimensions becomes prohibitive even on a fast workstation. Fortunately, it is possible to take advantage of distributed systems like computational Grids, to reduce the filtering processing time or to rely on wide network storage.

Computational Grids [2] are a particular kind of distributed systems which use the resources of many separate computers connected through the Internet and expose them as a virtual computer architecture that is able to distribute process execution across a parallel infrastructure. Grids can provide different kind of resources to the user: CPU cycles (*Computational Grids*), disk space (*Data Grids*) and services (*Service Grids*). A Grid is an intrinsically dynamic system: resources constituting the system change during time and normally come from different domains and organisations. When these resources are spread across an enterprise, provide services to users within that enterprise and are managed by a single organisation, we are considering an *Enterprise Grid* (which is popularly called as a Desktop Grid).

The “Grid concept” is now considerably established in the IT and there are many grid infrastructures that can be used (Globus [3], Gridbus [4], Achemi [5], Condor [6], NetSolve [9], Harness II [7], and H20 [8]). The real challenge now lies in making Grid computing infrastructures easily accessible and usable to the end users by seamlessly integrating their desktop applications with Grids on demand (whenever they need a huge computing power). In other words the Grid should be used as a service in any kind of application. At the moment Grid computing is already being employed widely as a service in e-Science and e-Business applications. In particular, in the case of e-Science, Grids are used to process large amounts of data generated by scientific experiments that evaluate models, and to share large datasets among researchers. Some projects that have been actively used by research communities include:, PlanetLab [10], myGrid [11], MediGrid [12], MammoGrid [13], and BIRN [14].

These large-scale efforts predomently focused on the use of high-end computing systems such as clusters and supercomputers to build computational Grids for scientific applications. Our work complements them by demonstraing how a light weight Grids can be established (by leveraging existing technologies) and easily harnessed for performing image filetering operaton. Filetering is the basis for many image manipulation tasks performed by any imaging application running on either desktop or workstation. Nowadays desktop users deal with ever-growing image sizes with the proliferation of digital cameras and the demand for more computing power to quickly perform image editing tasks has been growing. It is no more uncommon to manipulate images as large as 10K x 2K pixels on desktop computers. Almost all professional and semi-professional imaging applications allow extending their features with the use of plug-ins: by developing a plug-in we can easily enhance such applications and make them Grid aware. A smooth integration of desktop applications with enterprise grids (desktop grids) rapidly enhances adoption of Grids for common day-to-day applications.

In this article we present our work, called‘ImageGrid’, an application that has been developed as a proof of concept to demonstrate the advantages desktop Grid-based image filtering and to show how desktop applications can easily exploit Grid-services. ImageGrid allows performing basic image editing operations and let the user run them either locally or remotely by executing the filters on an Alchemi Grid. The integration between Alchemi and ImageGrid is seamless and does not require the user to learn much about the Grid. The user just has to provide his/her credentials and the host name/IP address of the Alchemi Grid Manager. For these reasons, ImageGrid is a good point to start and to learn from, if we wish to make Grid-aware professional imaging applications.

2. Image Filtering: Basics in Brief

A *digital image* is a representation of a two-dimensional image as a finite set of digital values, called picture elements or *pixels*. Digital images are commonly represented with 2D matrices whose elements a_{xy} maintain the color information (values for the red, green, and blue channels and transparency) of the corresponding point in the image at the given coordinates (x,y)[1].

Digital image processing is the use of computer algorithms to perform processing on digital images or make modifications to them. In particular, image filtering is the process of applying computer algorithms – called digital filters – to an image in order to create a new one. Image filtering allows performing basic image editing tasks such as image smoothing, sharpening, blurring, edge detection, mean removal and embossing. All these operations can be implemented by a particular class of filters called *linear filters*. Linear filters compute each pixel-value as a linear combination of the values of a set of pixels in the image. Usually this set is defined by the pixels contained in a square region centered on the pixel to be evaluated. In this case, the coefficients corresponding to each pixel can be arranged in a matrix, called the *kernel,* whose dimensions are defined by the previous square region. If the filter is described by a kernel it is possible to express the value of each new pixel as the result of the following 2D discrete convolution operation:

$$P_{filter}(x,y) = \sum_{0,kx}^{N-1}\sum_{0,ky}^{N-1} K[kx,ky] \cdot P(x-((N-1)/2)+kx, y-((N-1)/2)+ky)$$

Expression 1. Pixel Convolution

[1] Hereafter we will omit the term *digital* that is always implied.

In the expression N is the dimension of the kernel matrix K and P_{filter} and P are the functions which return the corresponding pixel information, given the coordinates x and y. Figure 1 describes the entire process of determining the new value of a pixel. The summations in Expression 1 can be easily translated into a two nested *for-loops* and by iterating this expression for all the pixels we can implement the filtering algorithm. Actually, the real implementation of the filtering algorithm has to take into account some issues that are not captured by the previous expression. These are : *pixel value underflow, overflow*[2] and *edge-pixel filtering*.

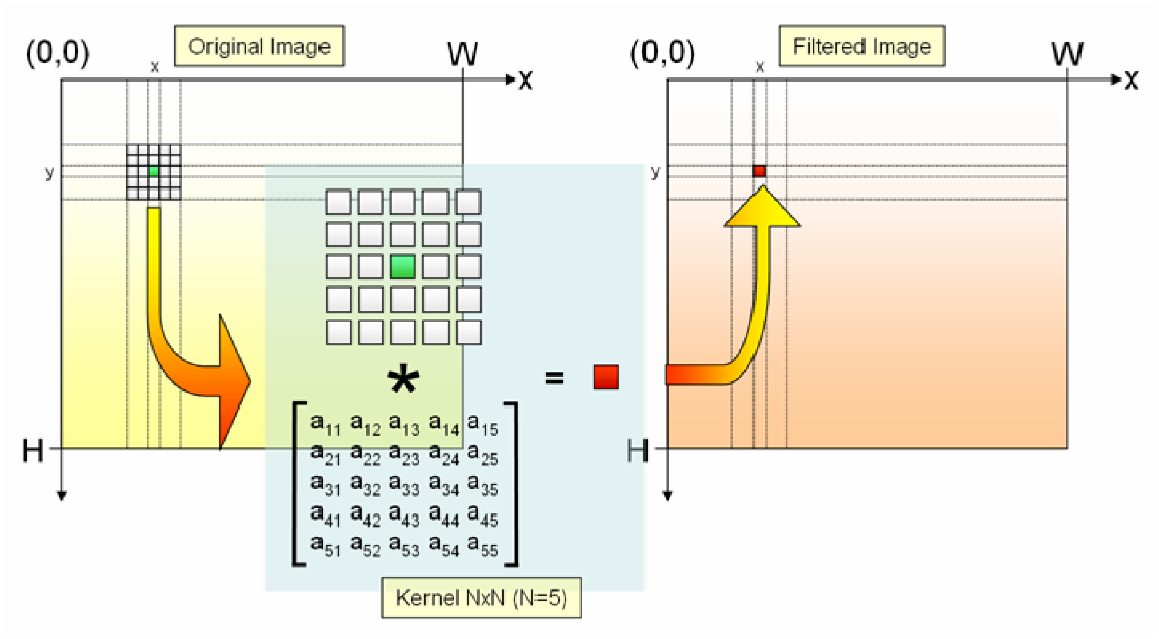


Figure 1- Computing the value of the filtered pixel.

2.1. Pixel Value Overflow and Underflow

So far we have not looked into the structure of the data maintained for each pixel and have implicity assumed that pixel color information is represented by a scalar value. Actually, the structure of such data strictly depends on the image encoding format and the scalar value has to be manipulated in order to extract the pixel information. If we consider images using a 24bppRGB encoding format, then the scalar value represented by the 24 bits has to be separated into the three corresponding bytes with each byte representing a single colour channel. This means that the previous expression has to be computed for each colour channel. Moreover, since the encoding assigns only one byte per channel, a pixel value for each channel ranges from 0 to 255, and the possible overflows or underflows have to be rounded to the range limits and the resulting value has to cast into a byte value.

2.2. Edge-pixel Filtering

Pixels on the edges of the image cannot be evaluated with the discussed algorithm since the square region required by the kernel is not properly defined. In other words, when we want to compute the value of $P_{filter}(0,0)$ we need the information of $P(-(N-1)/2, -(N-1)/2)$ which does not exist. In order to solve this problem the original image is enlarged by $(N-1)/2$ pixels on each side and the new regions are filled according to a given algorithm[3].

[2] The terms *underflow* and *overflow* respectively identify the condition in which a quantity goes out of an established range by assuming a value smaller than the minimum or bigger than the maximum. The expression *pixel value underflow (overflow)* means that the numeric value of the pixel is out of range.

[3] There are normally three different techniques used: *zero fill*, *mirror fill*, and *stretch fill*. The first strategy assigns to each pixel the black color value. The second one mirrors the pixel values by taking the original image borders as symmetry axis while the third one just replicates the pixel value of edge pixels.

3. Grid-based Image Filtering

Given an image the performance of linear filtering heavily depends on the kernel dimension. For example, given a kernel dimension of *N,* for each pixel channel we will have:

- *NxN* sums
- *NxN* products

Hence the complexity for each pixel is $O(N^2)$. This means that for large images the filtering process can take a lot of time and the operation is computationally intensive. A possible solution to reduce the processing time and the CPU workload is trying to parallelise the process and to take advantage of distributed computing infrastructures such as computational Grids.

This operation is actually possible because linear filtering is a *local* operation; this means that in order to compute the value of one pixel we need the information of only the nearby pixels and not of the entire image. More precisely, given a kernel of dimension *N,* in order to determine the value of pixel (x,y), we need to access the pixels contained in the square region centered on that pixel which has an edge of *N* pixels. Normally an even value of *N* ranging from 3 to 9 is chosen. A value of 3 is for fast filters while a value of 9 leads to more accurate but more computationally intensive filters.

Due to this locality property of linear filtering, we can parallelise the execution by dividing the image into several adjacent rectangular regions and model the filtering of each region as a separate Grid task. After all the tasks are executed the filtered regions are recomposed into the resulting image. Figure 2 describes the entire process.

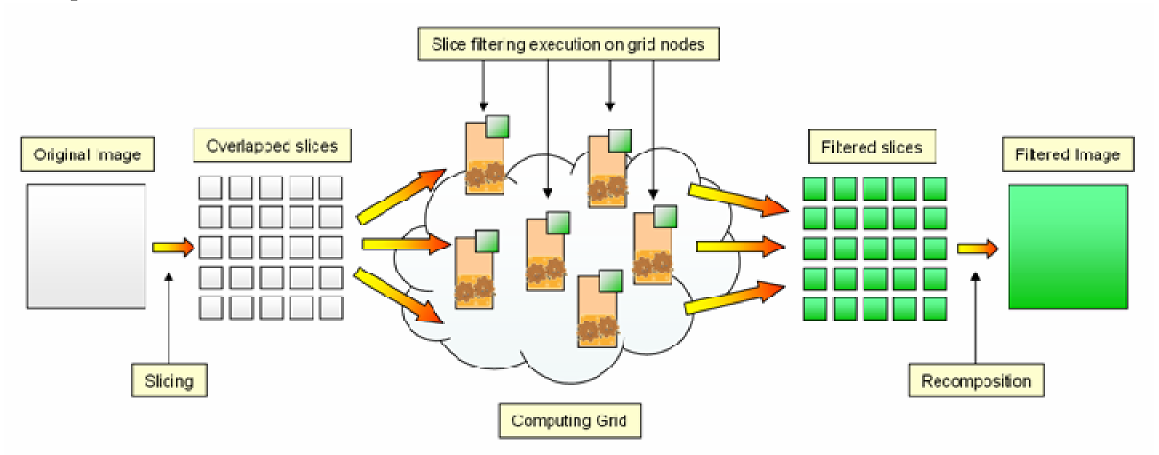


Figure 2 - Grid-based image filtering.

The task of filtering a large image can be broken down into a set of filtering tasks performed on smaller images. The operation performed on each rectangular region is the same as that for the entire image. If we overlap the rectangular regions by half of the kernel dimension the filtering process becomes an embarrassingly parallel problem and there will be no need of inter-task communication. Grid-based filtering adds additional time to the entire filtering task since we need to perform the following operations:

- Divide the image into rectangular regions
- Connect to the Grid and send tasks
- Recompose the filtered regions into the resulting image after the tasks are completed

The time required to perform these sub-tasks depends on the number of regions we decide to create; for these reason choosing the right number of regions can greatly influence the overall computation time. Nonetheless, the time required to perform these tasks is only a fraction of the entire computation time and is far less than the time required to perform filtering. This is particularly true for large images. Moreover, we can observe that image recomposition can be performed while the filtering process is still running without spending additional time. The reason behind this is that as soon as the grid tasks are completed the corresponding sub-region in the resulting image is filled with the filtered data. The recomposition time is then negligible in comparison to the slicing and the applicaction setup times. Thus the recomposition time can be excluded for computing the overall filtering time.

4. Exploiting the Power of the Grid: ImageGrid Implementation

ImageGrid is an application that allows users performing basic filtering operations on digital images. It relies on the Alchemi Grid computing infrastructure to perform filtering. ImageGrid allows you to run image filters in three different modes:

- *Default*: on the local node without parallel execution
- *Threaded*: on the local node with parallel execution
- *Alchemi*: on multiple nodes by exploiting the services of Alchemi

The application keeps track of the execution timing along with a wide range of statistics for the parallel execution modes. In particular, it is possible to see the timing for each slice and to collect the maximum, the minimum and the average execution time. By using this historical data the user can be supported in selecting the best execution mode for a given image.

ImageGrid is developed using the .NET Framework 2.0 and does not require anything more than the framework and the Alchemi libraries available for free download from http://www.alchemi.net.

4.1. Alchemi

Alchemi is an open source, .NET-based enterprise Grid computing framework developed by researchers at the GRIDS laboratory, in the Computer Science and Software Engineering Department at the University of Melbourne, Australia. It lets the user to painlessly aggregate the computing power of networked machines into a virtual supercomputer and develop applications to run on the Grid with no additional investment and no discernible impact on users. Alchemi supports the Microsoft Windows operating system and the main features offered by the framework are:

- Virtualisation of compute resources across the LAN/internet
- Ease of deployment and management
- Web Services interface for interoperability with Grid meta-schedulers

Three components constitute the architecture of an Alchemi Grid (see Figure 3):

- The Manager
- The Executor
- The User application

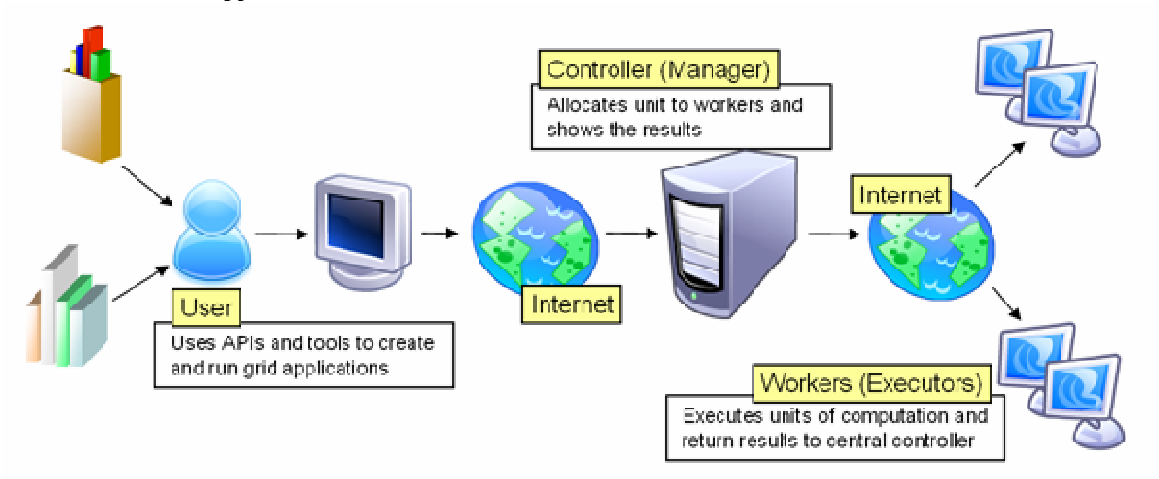

Controller (Manager)
Allocates unit to workers and shows the results
Internet
User
Uses APIs and tools to create and run grid applications
Internet
Workers (Executors)
Executes units of computation and return results to central controller

Figure 3 – Alchemi architecture.

The Manager node is a computer with the Alchemi Manager installed. Its main function is to service user requests for application distribution. On receiving a user request, the Manager authenticates it, and distributes the workload across the various executors that are connected to it. The Executor node is the node that actually performs the computation. By using the Alchemi Software Developer's Kit users can easily create the applications and run and monitor their execution on the Grid.

Alchemi offers two different programming models:

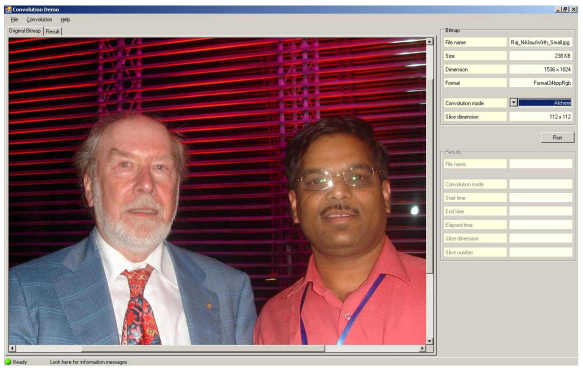
- Object-oriented G*rid thread* programming model. This model is suitable for Grid application development: a Grid application consist of a set of Grid threads which define the tasks performed by the application and are executed on the Grid.

- File-based G*rid job* model. This model allows legacy applications running on the Grid. In this case the users submit a job to the Grid which consists of an executable that will be run on the Grid Executor nodes.

By using the first model we take advantage of all the APIs available with the .NET framework while the second model is useful when we want to Grid-enable legacy/existing applications without changing their codebase. Alchemi is widely used for a variety of applications. It has been used for teaching, setting up test Grids and also for some commercial applications.

4.2. ImageGrid

ImageGrid is the GUI application which allows users performing basic image editing using predefined filters. Users can load images from the file system, apply the filters and save the results. Along with the basic filtering operations (edge detection, smoothing, Gaussian blur, sharpening, mean removal, and emboss) users can define their own linear filter by defining the kernel matrix through the settings dialog. The settings dialog also allows users providing the connection parameters to the Alchemi Grid and other properties for the parallel execution such as the slices dimension of the slices and recomposition mode.



Bitmap

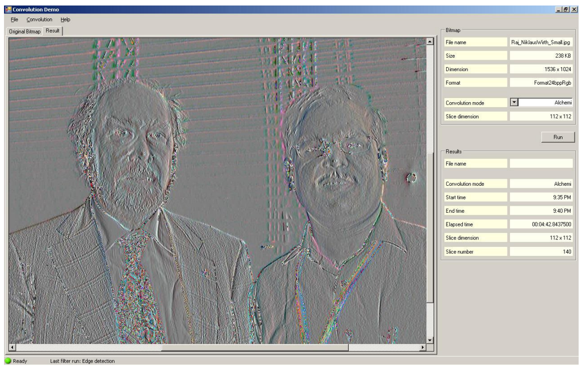
File name	Raj_NiklausWirth_Small.jpg
Size	238 KB
Dimension	1536 x 1024
Format	Format24bppRgb
Convolution mode	Alchemi
Slice dimension	112 x 112

Results

File name	
Convolution mode	
Start time	
End time	
Elapsed time	
Slice dimension	
Slice number	

Ready Look here for information messages

(a) Original Image



Bitmap

File name	Raj_NiklausWirth_Small.jpg
Size	238 KB
Dimension	1536 x 1024
Format	Format24bppRgb
Convolution mode	Alchemi
Slice dimension	112 x 112

Results

File name	
Convolution mode	Alchemi
Start time	9:35 PM
End time	9:40 PM
Elapsed time	00:04:42.8437500
Slice dimension	112 x 112
Slice number	140

Ready Last filter run: Edge detection

(b) Emboss Filetred Image

Figure 4 – A snapshot of ImageGrid GUI. Image used in creating emboss is a photo of ones of the authors (Buyya) taken with Prof. Niklaus Wirth who is the inventor of Pascal language.

Figure 4 shows the structure of the GUI and its usage in *emboss* operation. The working area hosts a tabbed interface in which the user can compare the filtered image and the original one. All the information about the current filtering task (image dimensions, size, filtering mode, and kernel) are mantained into a property page. ImageGrid records the execution times of each filters and provides a history of all the filters run: this feature allows users to compare the different runs quickly.

The GUI acts as the front-end of the imaging library which actually performs image filtering. All the filters must implement the *IFilter* interface which defines the basic operation each filter should support. In order to integrate the GUI with Alchemi we developed a filter class implementing the *IFilter* interface which connects to the Alchemi Grid computing infrastructure and applies filtering as described in Figure 2.

5. Performance Evaluation

We have run some tests in order to compare the performance of the different execution modes. We set up different tests by varying the following parameters:

- Image dimensions
- Slice dimensions
- Kernel dimensions

In order to run the tests we used a Pentium 4, 2.80 GHz with 2 GB RAM as local machine, while the Alchemi Grid was composed by 6 - 1 manager and 5 executors - Dell OPTIPlex GX 2f0 Pentium IV 3.40 GHz, 1.5 G of RAM connected through a 100 Mbps LAN. All the machines used in the test were running Windows XP SP2 and .NET framework 2.0.

Table 1 presents the timining statistics (hh.mm:ss.d) for the different filtering modes which have been tested with four different image sizes and four different kernel dimensions. As mentioned earlier in the case of parallel filtering only the slicing time is taken into account.

Image Size	Kernel	Default (Single Thread)	Parallel Thread											
			128x128			256x256			512x512			1024x1024		
			Threaded	*Alchemi*	*Slicing*	*Threaded*	*Alchemi*	*Slicing*	*Threaded*	*Alchemi*	*Slicing*	*Threaded*	*Alchemi*	*Slicing*
10205x1752	*9x9*	01.26:03.6	50:53.6	12:38.0	03:04.2	45:32.7	09:26.4	00:52.9	44:54.6	08:45.8	00:13.1	45:02.7	08:57.5	00:03.7
	7x7	00.41:42.5	32:15.0	09:44.2	03:03.5	28:31.6	06:15.6	00:48.3	27:46.5	05:37.5	00:18.1	27:43.8	05:45.7	00:04.5
	5x5	00.21:19.1	18:40.2	07:50.5	03:05.6	15:36.4	03:51.3	00:48.2	14:52.7	03:13.6	00:16.0	14:41.7	03:09.8	00:04.8
	3x3	00.08:14.2	11:19.3	08:08.5	02:58.1	06:57.4	02:19.0	00:52.9	06:07.4	01:30.8	00:14.4	05:54.4	01:21.6	00:04.0
3072x2304	*9x9*	00.29:54.4	22:00.9	04:21.9	00:28.0	18:50.2	03:42.1	00:07.3	18:58.8	03:28.9	00:02.0	18:27.8	04:52.4	00:01.0
	7x7	00.19:09.5	12:53.8	03:03.2	00:27.1	12:00.0	02:26.5	00:06.4	11:59.6	02:21.1	00:02.2	11:44.9	03:02.2	00:01.0
	5x5	00.08:56.5	07:39.8	02:12.8	00:29.8	06:46.4	01:28.5	00:07.8	06:23.9	01:19.5	00:02.4	06:38.3	01:40.2	00:00.9
	3x3	00.03:48.5	02:57.0	01:45.5	00:28.9	02:25.4	00:48.7	00:07.6	02:21.6	00:38.1	00:02.7	02:22.1	00:45.7	00:00.9
2048x1365	*9x9*	00.12:22.0	07:27.9	01:37.1	00:04.7	07:10.6	01:27.0	00:01.3	07:11.0	01:39.1	00:00.5	07:06.0	02:42.4	00:00.4
	7x7	00.07:33.0	04:44.9	01:12.1	00:06.5	04:32.6	00:55.8	00:01.5	04:30.6	00:55.9	00:00.5	05:00.3	01:31.9	00:00.3
	5x5	00.04:16.9	02:43.7	00:49.2	00:05.2	02:29.5	00:34.2	00:01.8	02:23.4	00:34.1	00:00.7	02:25.9	00:55.1	00:00.3
	3x3	00.01:41.6	01:00.6	00:36.7	00:05.3	00:55.7	00:18.8	00:01.4	00:55.9	00:16.1	00:00.4	00:54.7	00:23.1	00:00.2
1024x683	*9x9*	00.02:20.7	01:52.1	00:25.5	00:00.4	01:50.4	00:25.3	00:00.2	01:56.9	00:37.7	00:00.1	02:13.2	01:37.3	00:00.2
	7x7	00.01:21.9	01:10.1	00:17.6	00:00.4	01:08.0	00:16.7	00:00.1	01:09.8	00:24.6	00:00.1	01:25.6	01:01.6	00:00.2
	5x5	00.00:42.6	00:37.5	00:12.8	00:00.4	00:36.1	00:10.2	00:00.2	00:37.7	00:14.5	00:00.1	00:44.9	00:32.6	00:00.1
	3x3	00.00:16.8	00:14.7	00:09.9	00:00.5	00:13.8	00:06.4	00:00.1	00:13.8	00:07.3	00:00.1	00:16.4	00:15.8	00:00.2

Table 1 – Filter timing (hh.mm:ss.d).

The data represented in Table 1 shows that the use of Alchemi as computing backbone helps speeding up the image filtering process. This performance gain becomes more evident as the size of the image increases. The best results for the given configuration of Alchemi have been obtained with 256x256 and 512x512 slices. Figure 5 shows the comparison of execution times of the default mode with the worst runs of the parallel modes along with the time required to divide the image into slices: it can be observed that Alchemi gives always the best performance. In this case the time spent to divide the image in slices is just a small fraction of the overall execution time of the filter. Even when the two parallel filtering modes (*Threaded* and *Alchemi*) give comparable processing times the use of Alchemi has an advantage in that it does not cause a high percentage of CPU usage on the local machine.

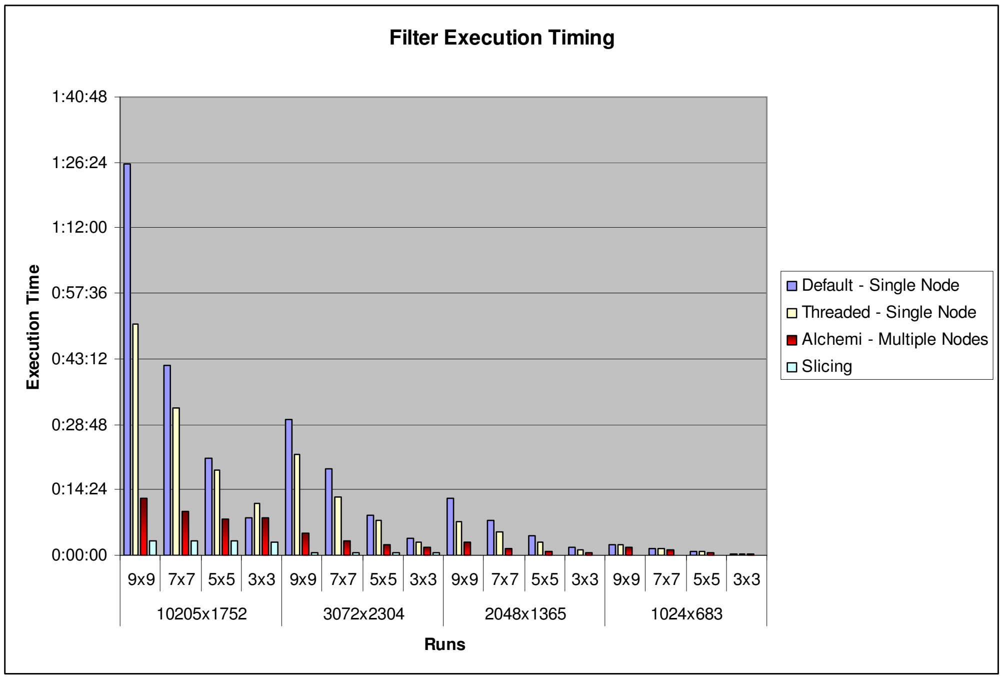


Figure 5 – Filter timing histogram.

6. Conclusion and Future Work

The use of Alchemi for image filtering – and of Grid computing in general – is a real advantage and its integration into ImageGrid has been a seamless task. ImageGrid is a proof of concept effort demonstraing seamless integration of desktop applications with light-weight Grids. It also demonstrates the strategy used and how it can be adopted in existing imaging software like PaintShop Pro or Photoshop. Another interesting idea is to try to plug-in Grids into the Paint.NET open source project. Paint.NET is a .NET-based imaging application which uses an architecture similar to ImageGrid to implement image filters. Paint.NET is a popular imaging software and the introduction of such a feature into its code base would really contribute to making the Grid computing resources available to end users with no burden.

References

[1]. R.C. Gonzales and R.E. Woods. Digital Image Processing 2nd Ed. Prentice Hall, 2002.

[2]. Ian Foster, Carl Kesselman, and Steve Tuekle, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International J. Supercomputer Applications*, Vol 15, No. 3, 2001.

[3]. Ian Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779, 2-13pp, 2006.

[4]. Rajkumar Buyya and Srikumar Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report", *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, April 23, 2004, Seoul, Korea, 19-36pp, IEEE Press, NJ, USA.

[5]. Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Alchemi: A .NETBased Enterprise Grid Computing System", *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)*, June 27-30, 2005, Las Vegas, USA.

[6]. Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience", *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, 323-356pp, February-April, 2005

[7]. Mauro Migliardi, Dawid Kurzyniec, and Vaidy Sunderam. "Standards Based Heterogeneous Metacomputing: The Design of Harness II", *International Parallel and Distributed Processing Symposium (IPDPS-HCW)*, April 2002, Ft. Lauderdale, FL.

[8]. Dawid Kurzyniec, Tomasz Wrzosek, Dominik Drzewiecki, and Vaidy Sunderam, "Towards Self-Organizing Distributed Computing Frameworks: The H2O Approach", *Parallel Processing Letters*, Vol. 13, No. 2, 273–290pp, 2003.

[9]. Keith Seymour, Asim YarKhan, Sbhishek Agrawal, and Jack Dongarra, "NetSolve: Grid Enabling Scientific Computing Environments", *Grid Computing and New Frontiers of High Performance Processing*, Grandinetti, L. eds. Elsevier, *Advances in Parallel Computing*, 14, 2005.

[10]. Andy Bavier et. al., "Operating System Support for Planetary-Scale Network Services", *First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 29-31, 2004, 253-266pp.

[11]. Robert D. Stevens, Alan J. Robinson, and Carole A. Goble, "myGrid: Personalised Bioinformatics on the Information Grid", *Proceedings of 11th International Conference on Intelligent Systems in Molecular Biology*, June 29–July 3, 2003, Brisbane, Australia.

[12]. Johan Montagnat, Vincent Breton, and Isabelle E. Magnin, "Using Grid Technologies to Face Medical Image Analysis Challenges", *Biogrid'03, Proceedings of the IEEE CCGrid03*, Tokyo, Japan, May 2003

[13]. Salvator Roberto Amendolia et. al., "MammoGrid: A Service Oriented Architecture based Medical Grid Application", *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04)*, October 20-23, 2004, Wuhan, China.

[14]. Jeffrey S. Grethe et. al., "Biomedical Informatics Research Network: Building a National Collaboratory to Hasten the Derivation of New Understanding and Treatment of Disease", *Studies in Health Technology and Informatics*. Vol 112, 100-109pp, 2005, ISBN 978-1-58603-510-5, IOS Press, Amsterdam, The Netherlands.