20-24 September
MICCAI 2009
London UK

MICCAI-Grid Workshop

Medical imaging on GRID, HPC and GPU Infrastructures
Interoperability Highlights on NeuroSciences

Proceedings of the MICCAI-Grid 2009 Workshop

Edited by

Giovanni Frisoni,

Chiara Barattieri

and

David Manset

with Johan Montagnat and Silvia D. Olabarriaga

MICCAI-Grid Workshop
http://proton.polytech.unice.fr/MICCAI-Grid/

20-24 September
MICCAI 2009
London UK

Preface

Medical image processing raises new challenges related to the scale and complexity of the required analyses, for example in studies that involve the federation of large data sets or in complex modelling and data mining. Grid and HPC technologies are addressing such problems by syndicating computing resources and providing tools for exploiting them, while additionally serving as vectors for structuring user communities. On a different scale but similarly, GPU-based processing is gaining more attention given its immediate benefits to users and close integration in imaging environments and practices.

In the area, Grids provide a foundational layer that can be utilized e.g., to build patient specific models, reduce computing time to meet clinical practice constraints, algorithms validation and optimization, or collaborative studies on rare diseases. Specific Grid initiatives are thus emerging worldwide, demonstrating a growing interest from the health community for such infrastructures and impacting on the way medical research is conducted. However, deploying medical image analysis applications on such infrastructures requires a proper understanding of the corresponding specific needs.

The MICCAI-Grid workshop aims at bringing together researchers using GRID, HPC and GPU infrastructures to address problems in medical image processing. The scientific objectives are twofold, (1) to demonstrate the current achievements of GRID, HPC and GPU technologies through concrete examples and (2) to precisely identify the fundamental problems limiting the adoption of existing systems and their interconnections from both a data and processing interoperability standpoint.

Thematic

Thus far, the MICCAI-Grid workshop has created a unique place where image processing researchers can debate on these aspects and become familiar with such technologies and international infrastructures being developed.

MICCAI-Grid stimulates the community to build new collaborations and leverage convergence by taking advantage of the sharing and social capabilities of distributed computing and infrastructures.

We kindly invite authors to submit original papers addressing one or more topics from the following list:

- Medical image processing using Grid, HPC and / or GPU infrastructures,
- Medical image pipelines / workflows, from authoring, to planning and enactment on Grid, HPC and / or GPUs,
- Distributed and heterogeneous medical data representation and annotation,
- Large scale and statistical studies,
- Content-based retrieval and data mining,
- Methods evaluation and parameter sweep studies,
- Medical data visualization using Grid, HPC and / or GPU resources,
- Dedicated Grid, HPC, GPU infrastructures,
- Standards for exchanging data and algorithms,
- Success stories and show stoppers.

Important Dates

Workshop venue	**September 24, 2009**
Deadline for submission	**June 14, 2009**
Acceptance notification	**July 19, 2009**
Camera ready	**August 14, 2009**

MICCAI-Grid Workshop
http://proton.polytech.unice.fr/MICCAI-Grid/

Program Committee

- **Alan C. Evans,** *Departments of Neurology and Neurosurgery Biomedical Engineering, Medical Physics, Montreal Neurological Institute at McGill University, Montreal, Canada*
- **Giovanni B. Frisoni,** *Epidemiology and Neuroimaging Laboratory, I.R.C.C.S Fatebenefratelli, Brescia, Italy*
- ***Alex Zijdenbos,*** *Prodema Informatics, Montreal, Canada*
- **Johan Montagnat,** *CNRS / I3S, Sophia Antipolis, France*
- **Alfredo Tirado-Ramos,** *Wallace H. Coulter Department of Biomedical Engineering Georgia Institute of Technology, Emory University School of Medicine and Center for Comprehensive Informatics, Woodruff Health Sciences Center Emory University, Atlanta, USA*
- **Diane Lingrand,** *I3S / Polytech'Sophia, Sophia Antipolis, France*
- **Silvia D. Olabarriaga,** *University of Amsterdam, Academic Medical Center, The Netherlands*
- **Alejandro Frangi,** *Pompeu Fabra University, Barcelona, Spain*
- **Christian Barillot,** *INSERM/INRIA IRISA, France*
- **Marco Antonio Gutierrez,** *Heart Institute, Sao Paulo, Brazil*
- **Ron Kikinis,** *Harvard University, Harvard Medical School, USA*
- **Toshiharu Nakai,** *National Center for Geriatrics and Gerontology, Japan*
- **Alexey Tsymbal,** *SIEMENS AG Corporate Technology, Erlagen, Germany*
- **Vipin Chaudhary,** *Computer Science and Engineering, SUNY at Buffalo, USA*
- **Dagmar Krefting,** *Institute of Medical Informatics of the Charity - Universitats Medizin Berlin, Germany*
- **Leiguang Gong,** *IBM T. J. Watson Research, Hawthorne, New York, USA*
- **Tony Solomonides,** *University of the West of England (UWE), Bristol, UK*
- **Richard McClatchey,** *University of the West of England (UWE), Bristol, UK*
- **Bob W. van Dijk,** *VU Medisch Centrum, Amsterdam, The Netherlands*
- **Tristan Glatard,** *CREATIS-LRMN, INSA, Lyon, France*
- **Yannick Legre,** *HealthGrid Association, Clermont-Ferrand, France*
- **Jerome Revillard,** *Biomedical Applications Department, maat Gknowledge, Archamps, France*
- **David Manset,** *Biomedical Applications Department, maat Gknowledge, Archamps, France*

MICCAI-Grid Workshop
http://proton.polytech.unice.fr/MICCAI-Grid/

Sponsor

This year the MICCAI-Grid workshop is organized and sponsored by the neuGRID project.
www.neugrid.eu

Recently launched by the Research Infrastructure Unit as part of the 7th Framework Program of the European Commission, neuGRID aims to establish a distributed e-Infrastructure interconnecting major clinical research centres in Europe, ultimately supplying neuroscientists with the most advanced Information and Communication Technologies to defeat Alzheimer's disease and neurodegenerative pathologies in general.

Actively pursuing collaboration and convergence with other projects in the field, neuGRID is delighted to organize this year's edition of the MICCAI-Grid Workshop.

Table of Contents

20-24 September
MICCAI 2009
London UK

MICCAI-Grid Workshop

Medical imaging on GRID, HPC and GPU Infrastructures
Interoperability Highlights on NeuroSciences

Part 1 – GPU-Based Image Processing

MICCAI-Grid Workshop
http://proton.polytech.unice.fr/MICCAI-Grid/

2D/3D registration at 1Hz using GPU splat rendering

Release 1.2

Christelle Gendrin[1] Jakob Spoerk [1] Christoph Weber[1] Michael Figl[1]
Dietmar Georg[2] Helmar Bergmann[1] and Wolfgang Birkfellner[1]

August 11, 2009

[1]Center for Biomedical Engineering and Physics, Medical University Vienna, Austria
[2]University Clinic for Radiotherapy, Division of Medical Physics, Medical University Vienna, Austria

Abstract

Nowadays, radiation therapy systems incorporate kilo voltage imaging units which allow for real-time acquisition of intrafractional X-ray image of the patient with high details and contrast. This technology enables real-time image based tumor motion monitoring. For tumor tracking implanted markers or sensors provide a method of choice but require an intervention. 2D/3D intensity based registration is an alternative non-invasive method but the procedure must be sped up to the update rate of the device. In this paper we compare the computation time and accuracy of 2D/3D registrations using a GPU based wobbled splatting algorithm implemented with the Cross-Correlation (CC) or Stochastic Rank Correlation (SRC) merit functions. SRC demonstrates better accuracy and superior robustness than CC algorithm. Registration at a frequency of 1Hz is achieved, which is by magnitudes faster than the currently reported time expenses for 2D/3D registration.

Contents

1 Introduction

As its name implies, Image-Guided Radiation Therapy (IGRT) uses an imaging modality during the therapy sessions of a patient. CT, MRI and PET allow to localise the tumor and to determine the treatment plan for maximum dose escalation in the target volume while sparing surrounding tissues. Thanks to mega voltage (MV) on-board imaging units in the radiation therapy systems, daily patient position can be checked and corrected, changes affecting the tumor such as size reduction or displacement might be monitored and the treatment plan might thus be updated accordingly [1] but the resulting image suffers from poor contrast. Recently, additional kilo voltage imaging units have been incorporated into the therapy systems allowing the acquisition of high contrast planar x-ray images. These technologies also enable IGRT with the possible monitoring of tumor motion in a real time .

So far, tumor tracking has been attempted by modeling periodic motion such as breathing and by tracking implanted fiducial markers on radiographs with the drawback of additional surgery [2]. 2D/3D registration is an non-invasive alternative method. 2D/3D registration was first introduced by Lemieux et al [3] to correct the patient position; the basic method is shown in Figure 1. An x-ray image of the patient is acquired and compared to Digitally Reconstructed Radiographs (DRRs) rendered from the pre-interventional 3D CT scan. A cost function measures the similarity between the base image and the DRR; it is minimized in order to find the best match. To achieve real-time tumor motion monitoring using intensity-based registration, this procedure must be sped up to the update frequency of the device which is, in our case, 5Hz with the Elekta Synergy suite.

Today, powerful graphics cards are commercially available at a low price. Thanks to high-level shading programs, they are easily programmable and the scientific community can now use them as a parallel stream processor. The computation time is thus drastically reduced, especially in the field of image processing and medical imaging where voxels can often be processed independently. General-purpose computations on graphics units (GPGPU) are particulary suited for DRR rendering and are essential to achieve real-time 2D/3D registration [4, 5]. In this paper we present high-speed 2D/3D registrations at a 1 Hz rate using a GPGPU version of the rendering, the wobbled splatting algorithm [6] previously developped by our group, with the merit function calculated on the CPU.

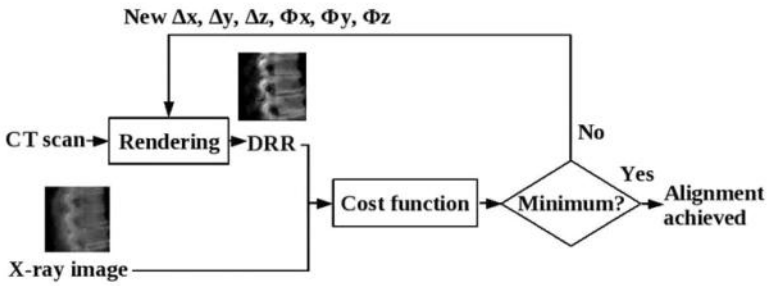


Figure 1: Framework for 2D/3D image registration.

2 Methods and materials

2.1 Wobbled Splatting algorithm

Wobbled splatting [6] has been proposed as a computationally efficient algorithm to render DRR while conserving image quality. Splatting is based on perspective projection of translated and rotated voxel data to the image plan and is mathematically defined by equation 1:

$$\vec{x}_p = \mathbf{P}.\mathbf{V}.\vec{x} \tag{1}$$

with

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/f & 1 \end{bmatrix} \qquad \mathbf{V} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where $\vec{x}$ is the original voxel position and $\vec{x_p}$ is the rotated and translated voxel position projected onto the DRR imaging plane, $\mathbf{P}$ is the projection to the X-Y plane alongside Z-axis with an X-ray focus at distance z=f. $r_{00}, ..., r_{22}$ describe the components of the rotations and t_x, t_y and t_z give the translation of the voxel coordinates. The main advantage is the fact that only pixels above a given intensity are effectively rendered; this dramatically reduces the number of pixels being used by the algorithm. The drawback of the splatting algorithm is the resulting aliasing effect. In the original version of the algorithm, a 3D Gaussian kernel is calculated for each voxel and projected into the image plan [7]. The projections of the kernels (the so-called footprints) are computed as a preprocessing step and stored in a look up table which is a very time consuming step. Our approach consists in introducing blurring by stochastic Gaussian motion, either to all voxels or to the focal spot, in the splatting process. Since the algorithm has already been described, the reader is referred to [6] for more information.

2.2 GPU implementation

Wobbled splatting has been successfully implemented on the GPU [8] using a vertex shader program developed with the Cg language as a first approach. Certainly the code could be rewritten using CUDA library and would further improve the velocity of the computation

The vertex shader program is provided in Figure 2. It performs the rotation and translation of the voxel points and the splatting using the focus-wobbled method (Figure 2 lines 25-28). As no random generator is provided for the shader program, a function given by equation (3) is used to introduce the pseudorandom numbers [8] (Figure 2 lines 18-19).

$$r = D\sin(p_i \pm p_j)\sin(p_k \pm p_l) \tag{3}$$

where D is the maximum displacement of the focal spot and $p_{i,j,k,l}$ is voxel coordinates and its intensity value in varying condition.

In our algorithm the intensity values of the voxels projected into the same image pixel are summed using the blending functionality of OpenGL. Blending is initially used to create effects like transparency fog or liquid through weighted summation of two or more object color. In our application new fragments are first scaled to the range [0 1]. They are subsequently linearly added to the values already stored in the frame buffer using alpha blending. Since any pixel intensity in the frame buffer outside the range [0 1] is clipped to that range, a second scaling applied to the the alpha channel is necessary to avoid that the sum of the pixel

intensities exceed 1. In our case, the maximum number of voxels that can be potentially summed into the same image pixel is the voxel number present along the diagonal dimension of the CT volume. In practice, using the ratio of this number leads to poorly constrasted image. This ratio is thus multiplied by the square root of the mean intensity in order to get an appropriate second scaling factor.

```
1 void vs_main ( //input parameter: vertex array and color array
2 in float3 vs_iPosition : POSITION,
3 in float vs_iColor,
4 //the following matrix is the transformation matrix (perspective projection,
5 //rotation and translation of pixels)
6 in uniform float4x4 TransMat,
7 //extremValues contains the scaling factors
8 in uniform float extremValues[3],
9 //Output of the vertex program: the  pixel position and color
10 out float4 vs_oPosition : POSITION,
11 out float4 vs_oColor : COLOR )
12
13{
14 //Matrix multiplication of rotation-translation matrix and projection matrix
15 float4x4 wobbleMat;
16 float4x4 tmp = mul(parameterMatrix[1],parameterMatrix[0]);
17
18 //Calculate the parameters for focus wobbling
19 float v1 = 1.0f * sin(vs_iPosition.y - vs_iColor) * sin(vs_iPosition.z + vs_iPosition.y);
20 float v2 = 1.0f * sin(vs_iPosition.z + vs_iColor) * sin(vs_iPosition.x - vs_iPosition.z);
21 float w0 = tmp._m00 * v1 + tmp._m01 * v2;
22 float w1 = tmp._m10 *v1 + tmp._m11 * v2;
23 float invF = tmp._m32;
24
25 //Calculate the new projection matrix
26 wobbleMat._m00_m10_m20_m30 = tmp._m00_m10_m20_m30;
27 wobbleMat._m01_m11_m21_m31 = tmp._m01_m11_m21_m31;
28 wobbleMat._m02_m12_m22_m32 = float4(tmp._m02+ invF * w0,tmp._m12 + invF*w1, 0, tmp._m32);
29 wobbleMat._m03_m13_m23_m33 = float4(tmp._m03+ w0, w1 + tmp._m13, 0 , tmp._m33);
30
31 //Calculate the position of the projected voxels and scale to the space in the range [-1 1]
32 float4 vs_pos = float4(vs_iPosition.x,vs_iPosition.y,vs_iPosition.z,1.0f);
33 float4 tmpVek = mul(wobbleMat, vs_pos);
34 tmpVek = tmpVek / tmpVek.w;
35 float scale = 1.0f/128.0f;
36 vs_oPosition = float4(tmpVek.x * scale, tmpVek.y * scale, 0.0f, 1.0f);
37
38 //Scale the color between 0 and 1 (only the red component is used in our implementation)
39 //and put appropriate scaling factor in the Alpha channel to avoid pixel overflowed.
40 vs_oColor = float4((vs_iColor - extremValues[0]) * extremValues[1], 0,0, extremValues[2]);
41
42 return;
43}
```

Figure 2: The vertex shader program for wobbled splatting algorithm

In our code, the necessary scaling values for avoiding overflow of the final pixel intensities are stored in the float table extremValues (Figure 2, line 7). extremValues[0] stores the minimum voxel intensity of the CT volume. extremValues[1] stores the fraction $\frac{1.0}{MaxInt - MinInt}$; with MaxInt and MinInt the maximum and

minimum voxel intensity of the CT volume respectively. This first two values are used to scale one projected voxel intensity in the range [0 1]. Then, extremValues[2] stores $\frac{\sqrt{MeanGray}}{MaxVox}$ with MeanGray the mean intensity of the CT volume and MaxVox the maximum number of voxels that can be potentially projected into the same pixel. This second scaling avoid pixel overflow when adding the intensities on the frame buffer.
For the transmission of data from the CPU to the GPU, the voxel positions and intensities are stored as point like element in two separate arrays. Each voxel position is described by a float triplet (x,y,z) and each intensity by one unsigned short. Each array is linked to the corresponding Cg input parameter and enabled in the OpenGL state using the Cg command cgGLEnableClientState(). The actual drawing is performed by issuing the OpenGl command: glDrawArrays() with each vertex considered as one point. The blending effect is previously switched on by calling the function glEnable(GL_BLEND).

2.3 Merit functions

In a context of real-time 2D/3D registration, not only the rendering but also the merit function should be calculated in a short time and should feature good convergence property to achieve the best alignment as possible. Differences in the intensities of the CT-scan and the X-ray image is an issue for the minimization of the merit function. For example, Cross-Correlation (CC) [3, 9] supposes the existence of a linear relationship between the intensities of the base and match images. Mutual Information (MI) [10] has been especially developed for intermodal imaging data and is well-suited for 3D/3D registration but has been demonstrated to fail in the case of 2D/3D registration due to sparse histogram population [11]. To overcome the problem of differences in the image intensities, our laboratory has recently published a novel cost function called Stochastic Rank Correlation (SRC). The idea of SRC is to assess a correlation between data not by using their actual value but their ranks in an ordered list of intensities. Since this method requires the sorting of data, a time consuming task, only a fraction of data point is actually employed in the merit function calculation. The reader is referred to [11] for additional details on this technique. The merit functions are implemented on the CPU.

2.4 Software, hardware and data-set

For initial testing of our 2D/3D registration based on the GPU, a software was developed in C++ using gcc 4.3 under the Fedora 9 Linux distribution. User interfaces were programmed using Qt 4.3 toolkit. The minimizer chosen for the optimization was the Downhill simplex algorithm and was taken from the GSL package (Free Software Foundation, Boston, MA). As already stated, the GPU was programmed through OpenGL interface and Cg language.
The programm was run on a personal computer mounted with a Intel-Core 2 Duo CPU of 3 GHZ each and a NVIDIA GeForce 8800 GTS series graphics card with a memory of 640 MB and 96 streaming processors. The CPU implementation did not exploit the 2 cores of the processor.
The CT-volume of a spine cord featuring 5 segments was used as a test set [12]. The data set had a dimension of 196*256*196 voxels. The X-ray image was obtained using 120 KV energy. The intensities of the CT-volume were scaled to grayscale values in the range 0-255.

2.5 Implementation evaluation

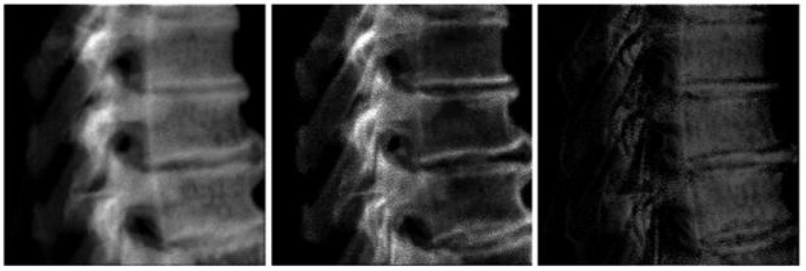
Our research effort aims at achieving on-line tumor motion using 2D/3D registration. Besides registration speed, its accuracy is of crucial importance and both parameters must be compared for GPU and CPU registrations. The accuracy of the registration depends mainly on the merit function, therefore two merit

functions: CC and SRC (with 1%, 3% and 5% of the image content called respectively SRC1, SRC3, and SRC5 in the following of the article) were also evaluated. Since the registration speed is also dependent of the number of projected data points of the CT-scan different thresholds were tried out. The results obtained with a threshold of 70 and 90, which were the more relevant, are reported in this paper.
The tests were conducted on a diagnostic CT-Scan of the spine. The starting points for the registration were chosen in a displacement range of 5° for the rotations and 10 pixels for the translations. 100 registrations were done. The distance between the final positions and the initial guess was computed and the mean and standard deviation over all the registrations were estimated.

3 Results and discussion

3.1 Digitally Reconstructed Radiographs

Figure 3 shows two DRR obtained using CPU (Figure 3(a)) or GPU implementation (Figure 3(b)) and the image of the difference (Figure 3(c)). With our data, the contrast of the two DRRs is obviously different. The explanation of this result lies in the use of the scaling factor on the alpha channel, calculated before the rendering in the GPU implementation, to avoid that final pixel intensity with intensity outside the allowed range are automatically clipped. On the CPU, the voxel intensities are first added and the resulting image is scaled afterwards to grayscale values in the range 0-255. Differences in the DRR intensity might lead to differences in the registration accuracy depending on the employed merit function.



(a) CPU rendering (b) GPU rendering (c) Difference image

Figure 3: Digitally Reconstructed Radiographs (DRR) rendered from the pre-interventional 3D CT scan. Differences in the intensity of the DRR are evident between CPU and GPU rendering

3.2 Registration time and accuracy

Table 1 gives the mean and standard deviation of the number of iterations necessary to minimize the cost function and the total registration time. With a threshold of 70, the GPU implementation divides the total registration time by a factor of 8 for CC and SRC1. When the percentage of pixels used to compute the SRC increases, the factor decreases to 7 (SRC3) and 5 (SRC5).With a threshold of 90 the gain in computation time is a factor of 5 to 3 because the number of pixels to be projected is lower. The matrix multiplication performed on Figure 2, line 16, could be done once for all on the CPU to further speed-up the computation. GPU greatly accelerates the rendering process [8] but the computation of the merit functions on the CPU, the number of iterations and also the data transfer between CPU and GPU becomes the bottleneck of high speed

registration. However, in our experiment we achieve registration below 1Hz with the SRC1 algorithm. Regarding registration accuracy (Table 2), SRC merit function performs better than CC. Especially, CC results are worst when GPU rendering is used whereas SRC does not show significant difference between GPU and CPU implementation. Clearly, CC is more sensitive to difference among pixel intensity. With our data, the SRC merit functions perform similar whatever the percentage of pixel used to calculate the ranks. It is demonstrated here that accurate and fast registration (below 1Hz) is possible. Even if these first results still need to be confirmed with further investigations and phantom studies, we believe that on-line tumor monitoring based on 2D/3D registration shall be possible with further hardware and software development and the help of a locally distributed environment.

	Threshold		Merit function			
			CC	SRC (1%)	SRC (3%)	SRC (5%)
CPU	70	Reg. Time	25.38 ± 6.33	17.47 ± 3.87	21.05 ± 4.84	22.58 ± 5.58
		# Iter.	82 ± 26	53 ± 16	60 ± 19	64 ± 21
	90	Reg. Time	7.50 ± 1.70	3.76 ± 0.64	6.71 ± 1.22	8.84 ± 1.68
		# Iter.	74 ± 20	49 ± 13	54 ± 14	60 ± 15
GPU	70	Reg. Time	3.02 ± 0.92	1.98 ± 0.40	3.04 ± 0.63	4.90 ± 1.20
		# Iter.	96 ± 34	58 ± 16	66 ± 18	74 ± 23
	90	Reg. Time	1.33 ± 0.40	0.89 ± 0.15	1.80 ± 0.33	3.48 ± 0.66
		# Iter.	83 ± 28	51 ± 13	61 ± 19	64 ± 16

Table 1: Mean and standard deviation of the running time and number of iterations obtained with 100 registrations of an X-ray image of the spine to the diagnostic CT scan. Registration as a speed of 1Hz is achieved.

	Threshold	Result scatter	Merit function			
			CC	SRC (1%)	SRC (3%)	SRC (5%)
CPU	70	$\Delta(\phi)$	1.82 ± 2.02	2.16 ± 2.18	2.02 ± 1.93	1.95 ± 1.80
		$\Delta X - Y[mm]$	3.31 ± 2.61	2.93 ± 2.07	3.00 ± 2.14	3.07 ± 2.20
		$\Delta Z[mm]$	2.16 ± 1.82	1.96 ± 1.61	1.90 ± 1.53	1.92 ± 1.71
	90	$\Delta(\phi)$	1.85 ± 1.81	2.21 ± 2.14	1.97 ± 1.93	2.02 ± 2.10
		$\Delta X - Y[mm]$	2.97 ± 2.27	2.82 ± 1.97	2.85 ± 2.02	2.93 ± 2.03
		$\Delta Z[mm]$	2.01 ± 1.69	1.85 ± 1.60	1.85 ± 1.62	1.90 ± 1.54
GPU	70	$\Delta(\phi)$	2.27 ± 3.84	2.09 ± 2.06	1.89 ± 1.90	2.09 ± 2.83
		$\Delta X - Y[mm]$	4.19 ± 4.85	2.92 ± 2.19	2.79 ± 2.04	3.04 ± 2.59
		$\Delta Z[mm]$	2.91 ± 4.31	1.87 ± 1.50	1.96 ± 1.65	1.96 ± 1.66
	90	$\Delta(\phi)$	1.94 ± 1.93	2.26 ± 2.05	1.96 ± 1.83	2.04 ± 1.96
		$\Delta X - Y[mm]$	3.11 ± 2.46	2.90 ± 2.11	2.98 ± 2.15	2.98 ± 2.17
		$\Delta Z[mm]$	2.06 ± 1.81	1.76 ± 1.53	1.84 ± 1.54	1.83 ± 1.58

Table 2: Mean and standard deviation of the difference between final displacement and initial guess obtained with 100 registrations of an X-ray image of the spine to the diagnostic CT scan. SRC is more robust than CC

4 Conclusion

In this paper we have investigated the speed and accuracy of intensity based 2D/3D registration using a GPU implementation of the wobbled splatting algorithm and two merit functions, namely CC and SRC. Compared to a CPU implementation, the GPU one was 3 to 8 times faster depending on the CT threshold and the image content used to calculate the SRC algorithm. GPU and CPU implementations generated DRR with different contrast but the SRC merit function provided same registration accuracy. The fastest registrations were carried out in less than 1s which renders image-based intrafractional tumor motion monitoring within reach.

5 Acknowledgments

We acknowledge the financial support of the Austrian FWF Translational Research Program (Project P19931 and L503).

References

[1] L. A. Dawson and M. B. Sharp. Image-guided radiotherapy: rational, benefits, and limitations. *The Lancet Oncology*, 7:848–858, 2006. 1

[2] H. Shirato, S. Shimizu, K. Kitamura et al. Organ motion in image-guided radiotherapy: lessons from real-time tumor-tracking radiotherapy. *International Journal of Clinical Oncology*, 12:8–16, 2007. 1

[3] L. Lemieux, R. Jagoe, D. R. Fish et al. A patient-to-computed-tomography image registration method based on digitally reconstructed radiographs. *Medical Physics*, 21(11):1749–1760, 1994. 1, 2.3

[4] M. Grabner, T. Pock, T. Gross et al. Automatic Differentiation for GPU-accelerated 2D/3D Registration. In *Advances in Automatic Differentiation*, 259–269. Springer, 2008. 1

[5] A. Kubias, F. Deinzer, T. Feldmann et al. 2D/3D image registration on the GPU. *Pattern Recognition and Image Analysis*, 18:381–389, 2008. 1

[6] W. Birkfellner, R. Seemann, M. Figl et al. Wobbled splatting a fast perspective volume rendering method for simulation of x-ray images from CT. *Physics in Medicine and Biology*, 50(9):N73–N84, 2005. 1, 2.1, 2.1

[7] L. Westover. Footprint evaluation for volume rendering. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques, SIGGRAPH'90*, 367–376. 1990. 2.1

[8] J. Spoerk, H. Bergmann, F. Wanschitz et al. Fast DRR splat rendering using common consumer graphics hardware. *Medical Physics*, 34(11):4302–4308, 2007. 2.2, 3.2

[9] G. P. Penney, J. Weese, J. A. Little et al. A comparison of similarity measures for use in 2D-3D medical image registration. *IEEE Transactions on Medical Imaging*, 17:586–595, 1998. 2.3

[10] J. Kim, J. A. Fessler, K. L. Lam et al. A feasibility study of mutual information based setup error estimation for radiotherapy. *Medical Physics*, 28(12):2507–2517, 2001. 2.3

[11] W. Birkfellner, M. Stock, M. Figl et al. Stochastic Rank Correlation - a robust merit function for 2D/3D registration of image data obtained at different energies. *Medical Physics*, accepted for publication, 2009. 2.3

[12] E. B. van de Kraats, G. P. Penney, D. Tomazevic et al. Standardized evaluation methodology for 2-D-3-D registration. *IEEE Transactions on Medical Imaging*, 24:1177–1189, 2005. 2.4

A Parallel Annealing Method For Automatic Color Cervigram Image Segmentation

Edward Kim, Wei Wang, Hongsheng Li, Xiaolei Huang

Image Data Emulation and Analysis Laboratory
Lehigh University

Abstract

The accurate and automatic segmentation of tissue regions in cervigram images can aid in the identification and classification of precancerous regions. We implement and analyze four GPU (Graphics Processing Unit) based clustering algorithms: K-means, mean shift, deterministic annealing, and spatially coherent deterministic annealing. From our results, we propose a novel parallel algorithm using the CUDA programming language for digital cervigram segmentation and clustering. The first step of our fully automatic method is to compute the number of modes in the feature space of a color cervigram image using the mean shift clustering algorithm. Next, we use the number of modes in a novel spatially coherent deterministic annealing optimization technique to produce an approximate optimal solution for the clustering problem. Our GPU based methods perform approximately 38x (deterministic annealing), 134x (mean shift), and 276x (spatially coherent deterministic annealing) faster than an equivalent CPU solution. Our implementation decreases the computational time of an annealing method on a 1280x872 pixel image from 5 hours 3 minutes to 72.12 seconds, enabling the use of this optimization method in clinical settings and on large cervigram datasets.

Contents

1 Introduction

Clustering is an important technique in image segmentation. In consideration of an archive of 60,000 color uterine cervix images created by the National Library of Medicine (NLM) and the National Cancer Institute (NCI), we require a fast, accurate, and automatic segmentation method to analyze these images. Specifically in the area of cervigram image analysis, the most important observed area to segment is the Acetowhite (AW) region, which is caused by the whitening of potentially malignant regions of the cervix epithelium. The segmentation of this area and other tissue regions is particularly challenging due to high variability where tissue color distributions frequently overlap with different classes [16, 7]. Segmentation using simple clustering algorithms like K-means and fuzzy c-means [9], run quickly but are highly sensitive to initialization. Mean shift [4] is a more advanced algorithm able to detect the number of modes in an image, but similarly cannot guarantee a globally optimal clustering. Other more complex methods can achieve a globally optimal clustering (e.g. deterministic annealing (DA) [13]), but are computationally expensive and require many iterations, preventing their practical use in clinical segmentation or for processing large medical image archives. Indeed, in image processing techniques, it is typical to have a trade-off between complexity, robustness, automation, and speed [8]. Fast segmentation methods are often not sufficiently robust or require large amounts of user interaction, whereas, complex methods are often robust, but are computation and time prohibitive.

Thus, some researchers have explored parallelization of clustering techniques to achieve fast execution times. Several implementations of a parallel K-means clustering [14, 5, 2, 15] have been analyzed and have achieved good speed up results. Other research has attempted to exploit algorithmic parallelism of mean shift [1, 15, 18] and deterministic annealing [12] on grid computing or SIMD architectures. On medical images, Gammage *et. al* [6] has used fuzzy connectedness to perform a segmentation using OpenMP on a SMP system resulting in a speed up of nearly 50x using four processors. Although many grid or multi-core implementations scale fairly well, they become power, cost, and resource limited at high speed up factors. For instance, in [15], the speed up factor of several parallelized algorithms is linearly related to the number of processors. On the other hand, K-means [14] and mean shift [18] have been analyzed on GPU architectures, but unanswered questions remain regarding the adoption of complex clustering algorithms on the newest CUDA (Compute Unified Device Architecture) enabled GPU architectures.

In this paper, we use the CUDA [10] programming language and GPU infrastructure to implement four clustering algorithms: K-means, mean shift, deterministic annealing (DA), and a novel spatially coherent adaptive deterministic annealing method (spatial DA). Next, we analyze these methods to realize trends and identify where computational bottlenecks exist. Finally, we combine several of these methods together to produce a robust algorithm called *Parallel Mode Deterministic Annealing* (PMDA) and apply PMDA to our cervigram image segmentation problem. The proposed approach uses a mean shift algorithm to detect the modes of an image and a spatial DA method to render a globally optimal clustering, unaffected by cluster initialization. Our method resembles [3], but incorporates spatial continuity features and exploits the inherent parallelism of mean shift and spatial DA. Our spatial DA method has roots in [17, 9], but the previous methods presented a spatial constraint that is either too general (single scalar for the entire image) or too specific (constrained to a small window). Thus, our contributions are to,

1. provide a comparative analysis of four clustering algorithms on the GPU using CUDA
2. utilize the massively parallel architecture of GPUs to produce results nearly 38x (DA), 134x (mean shift), and 276x (spatial DA) faster than equivalent CPU implementations for a 1720x1172 pixel image
3. present PMDA, an unsupervised, automatic segmentation method for color medical images which is able to achieve an accurate globally optimal clustering in a time frame suitable for clinical use and on large scale medical image data sets
4. introduce an adaptive, spatially coherent clustering GPU extension to DA

2 Compute Unified Device Architecture

CUDA is a parallel programming model and software environment that is able to leverage the computational power of the latest generation Nvidia GPU processors.

CUDA Hardware Model: A GPU is a collection of stream processors that can be thought of as a massively multi-core processor. In the case of our experiment, one of the GPUs we used was a Nvidia Geforce 280 GTX. This card has a total of 240 streaming processing cores and 1 GB of total graphics memory. Each stream processor executes the same instruction but on different areas of memory. Additionally, each multiprocessor has local register memory, fast shared memory, and unrestricted access to texture and global memory. However, use of different types of memory incur different access times. One of the more recent modifications for GPUs is the addition of 64-bit support which has shown to be crucial when dealing with high precision algorithms (e.g. deterministic annealing).

CUDA Software Model: The CUDA software model and execution model can be most easily described from the bottom up. The most basic computational elements are *threads*. A collection of threads, called a *block*, runs on a multiprocessor and the block size is defined by the user. Threads within the block are able to share resources such as registers and shared memory. The combination of all the blocks makes up the *grid*. Typically, the grid covers the entire area of computation for your GPU program (in our case, the image size). A single GPU program is called a *kernel*. Kernel programs are frequently launched by a standard CPU program where many parallel operations can be executed. These kernel programs do the bulk of our computation which we describe in detail in later sections.

3 Clustering Methods on the GPU

In this section we first describe our implementation of four clustering methods on the GPU. Then, we present our proposed PMDA method for a fast, automatic segmentation of cervigram images. For all of our clustering methods, we use $L^*a^*b^*$ color space because the $L^*a^*b^*$ space has been shown to be better for clustering and classification [4]. Each color channel is stored in texture memory as a 2D texture. The reason for using texture memory rather than global or constant memory is because *texture fetches*, i.e. reads from texture memory, can exhibit higher bandwidth. The GPU architecture is better able to hide the latency of the addressing calculations and there is 2D spatial locality in texture fetches.

3.1 K-means and CUDA implementation

The K-means algorithm is a very popular clustering algorithm that is quite fast; however, it is sensitive to outliers, initialization of cluster centers, and needs to be given a predetermined number of clusters.

K-means CUDA Implementation and kernel description: The K-means CUDA implementation begins with a random initialization of cluster centers. Each pixel is assigned a thread on the GPU and each thread belongs to the kernel grid. The dimensions of the grid for the K-means kernel encompass the entirety of the cervigram image. The kernel is defined as,

1. over the entire grid, compute the distance of each pixel to each cluster center in feature space
2. choose the minimum distance centroid (cluster center) and store all $width \times height$ labels in a 1D GPU global memory array of size $width \times height$
3. on the CPU, update the centroids based upon the minimum distance labels

3.2 Mean Shift Mode Detection and CUDA implementation

The mean shift [4] procedure is a non-parametric density estimation method that finds the *modes*, or local maxima of the empirical probability density function (p.d.f). This is a more complex clustering method that has the advantage of not needing to know the number of clusters *a priori*. It uses a density distribution kernel

to shift a window of size h in the direction of higher density, where the direction is defined by the mean shift vector. Thus, the kernel will eventually converge at the local density maxima in feature space. Our radially symmetric profile at data point x is defined as,

$$g(x) = -\frac{1}{2}\exp\left(-\frac{1}{2}x\right) \qquad x \geq 0 \tag{1}$$

Using $g(x)$, the kernel used in our mean shift procedure, $G(x)$ is defined as,

$$G(x) = c_{g,d}\, g(\|x\|^2) \tag{2}$$

where $c_{g,d}$ is the normalization constant. The mean shift procedure involves successive computation of the mean shift vector $m_{h,G}(x)$ and translation of the kernel window $G(x)$ by $m_{h,G}(x)$. This process repeats until the magnitude of the mean shift vector converges to 0. Given n data points x_i, $i = 1,\ldots,n$, and window size h, the mean shift vector can be defined as,

$$m_{h,G}(x) = \frac{\sum_{i=1}^{n} x_i g(\|\frac{x-x_i}{h}\|^2)}{\sum_{i=1}^{n} g(\|\frac{x-x_i}{h}\|^2)} - x \tag{3}$$

where x is the center of the kernel window.

Mean Shift CUDA Implementation and kernel description: The dimensions of the grid for the mean shift kernel encompass the entirety of the cervigram image. Each pixel is assigned a thread on the GPU and each thread runs the mean shift procedure to convergence;

1. over the entire grid, compute the mean shift vector $m_{h,G}(x)$ as in equation (3) and translation of the kernel window $G(x)$ in the color feature space
2. store all $width \times height$ modes in a 1D GPU global memory array of size $width \times height$
3. on the CPU, merge all $width \times height$ modes within a provided bandwidth parameter, h

3.3 Deterministic Annealing (DA)

The DA method [13] is a globally optimal clustering method that introduces an element of randomness to a clustering problem to avoid being stuck in local minima. Similar to simulated annealing, the procedure does not always take the greedy decision and is able to jump out of local minima. The amount of jumping can be thought of as the temperature and as the temperature cools and decreases the randomness, the clustering should converge to a global minimum. The mass-constrained clustering DA method that is independent of cluster center initializations is defined as the minimization of,

$$F = D - TH \tag{4}$$

where H is defined as the Shannon entropy in equation (5), T, temperature, is the Lagrange multiplier, and D is the distortion in equation (6). H is defined as,

$$H(X,Y) = -\sum_{x}\sum_{y} p(x,y)\log p(x,y) \tag{5}$$

where $x \in X$ is the set of data points and $y \in Y$ is the set of clusters. D is the expected distortion,

$$D = \sum_{x}\sum_{y} p(x,y)d(x,y) = \sum_{x} p(x)\sum_{y} p(y|x)d(x,y) \tag{6}$$

Our distortion measure $d(x,y)$ was defined as the Euclidean sum of squared differences between x and the center of cluster y in each color channel. Similar to fuzzy membership, our x data points are associated with

(a) Original (b) Deterministic Annealing (c) Spatially Coherent DA

Figure 1: *Example of DA clustering without (b) and with (c) spatial constraints.* $\sigma_2 = 1.0$.

some probability to every cluster y with probability magnitude $p(y|x)$. When T is a large value, the equation maximizes the entropy; however, as T is lowered, the equation minimizes D. Given y_j as the distinct cluster center, the association probabilities are calculated by,

$$p(y_j|x) = \frac{p(y_j)\exp\left(-\frac{d(x,y_j)}{T}\right)}{Z_x} \tag{7}$$

where Z_x is defined as the normalization constant,

$$Z_x = \sum_i^k p(y_i)\exp\left(-\frac{d(x,y_i)}{T}\right) \tag{8}$$

Here, k is the number of cluster centers. To minimize the distortion with respect to the cluster center, y, the gradients can be set to zero, yielding,

$$\sum_x p(x)p(y|x)\frac{d}{dy}d(x,y) = 0 \qquad \forall y \in Y \tag{9}$$

3.4 Spatially Coherent Deterministic Annealing (spatial DA)

In many image clustering problems, including our cervigram application, regions of similar features tend to appear together spatially [17]. However in the basic DA method, these spatial features are ignored in the global optimization problem. To incorporate spatial coherency to the DA method, we introduce a weighting function that influences the DA distortion measurement to incorporate the membership of the current pixel as well as its surrounding 3x3 neighborhood. The idea is that neighboring pixels can influence the current pixels' cluster membership based on their feature space similarity. Thus, neighboring pixels with similar features have a greater probability to belong to the same cluster. Let (r,s) represent a pixel location and (r_n,s_n) represent the neighbor. The weighting function of the neighboring pixels on (r,s) is defined as,

$$\lambda^{r,s}(\delta) = \frac{1}{1+\exp\left(\frac{-(\delta-\mu)}{\sigma_1}\right)} \tag{10}$$

where μ shifts λ and is defined in equation (13), and σ_1 specifies the steepness of the sigmoid curve. The δ value is computed by,

$$\delta_{(r,s),(r_n,s_n)} = (x_{r,s} - x_{r_n,s_n})^T(x_{r,s} - x_{r_n,s_n}) \tag{11}$$

where $x_{r,s}$ is the feature vector and x_{r_n,s_n} is the neighbors' feature vector. This equation (11), computes the Euclidean distance between the two feature vectors. We can now use the following distortion term as a substitute for $d(x,y)$ in equations (7),(8), and (9),

$$d'(x,y) = \frac{1}{8}\sum_{l_1=-1}^{1}\sum_{l_2=-1}^{1}\left[d(x_{r,s},y)\lambda^{r,s}_{l_1,l_2} + d(x_{r+l_1,s+l_2},y)(1-\lambda^{r,s}_{l_1,l_2})\right] \tag{12}$$

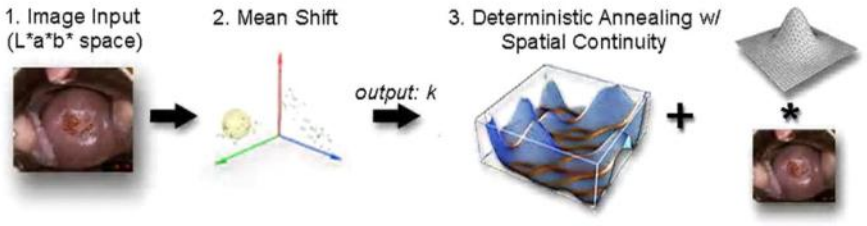


Figure 2: *PMDA Method*

where $\lambda_{i,j}^{r,s} = \lambda(\delta_{(r,s),(r_i,s_j)})$.

Research presented in [17, 9], describes spatial constraints either too general (single scalar for the entire image) or too specific (constrained to small window). Our method approximates μ, while parameterizing a Gaussian smoothing technique to encompass as large an area as desired. This term is able to be precomputed and stored in texture memory for fast fetching on our GPU implementation. Our μ is defined as,

$$\mu_{(r,s)} = 1 - || \bigtriangledown G_{\sigma_2} * I_{(r,s)} ||_2 \tag{13}$$

where our image is convolved with a smoothing Gaussian,G_{σ_2}. There is an inverse relationship between pixel busyness and image gradient. If the gradient at a pixel is high, this can be thought of as high busyness and thus the μ would be small. The high gradient would be present in non-homogeneous regions and would shift the weight influence on the center pixel to a smaller value. Given our parameterized method, we can adjust the amount of spatial continuity desired at each pixel. Fig. 1 contrasts the results between the basic DA method and the new spatial DA method.

3.5 DA and spatial DA CUDA Implementation

Because the framework of the DA and spatial DA methods differ only by their distortion terms, we represent both methods in this section. At each temperature state, the DA algorithm consists of three steps,

1. fix the cluster centers and use equation (7) in a GPU kernel to compute the association probabilities.
2. fix the association probabilities and optimize the cluster centers using (9). The maximum association probabilities are stored in a 1D GPU global memory array of size $width \times height$, one array for each color channel. The current individual cluster centers are stored in a 1D GPU global memory array of size k.
3. use the CUBLAS libraries (CUDA Basic Linear Algebra) to compute the parallel reduction of the arrays to find the final vector positions and probabilities.

This kernel is computed $\forall y \in Y$ until the cluster centers converge. Upon convergence, the cooling schedule will determine the next value of T and will continue until $T < \varepsilon$.

3.6 PMDA Methodology

Our *Parallel Mode Deterministic Annealing* (PMDA) method is a combination of a mean shift mode detection followed by a spatial DA clustering algorithm. Since we cannot assume a fixed number of clusters for our image segmentation, we use a mean shift algorithm to detect the number of color modes and we use this characteristic, k, to initialize our spatial DA clustering method. The method consists of three steps (Fig. 2),

1. input the image in $L^*a^*b^*$ color space
2. compute the modes, k, of an image by a mean shift clustering algorithm.
3. compute the optimal clustering using spatial DA with parameter, k.

(a) Several image examples (b) Cervigram image (c) Feature space

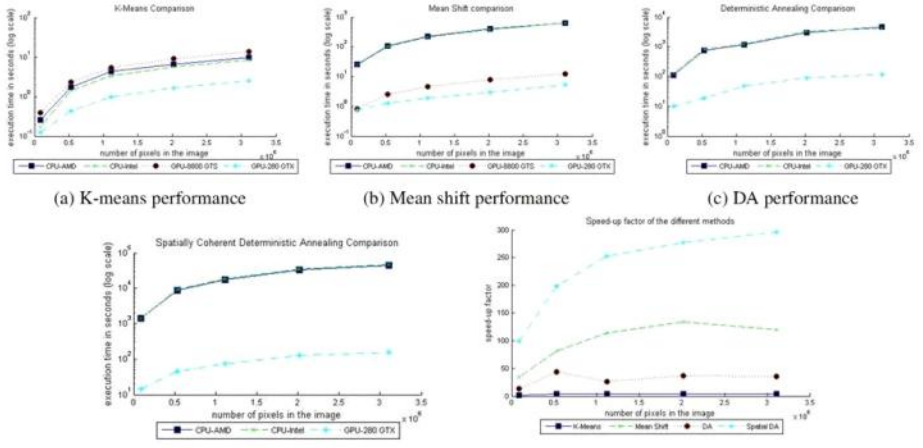
Figure 3: *(a) Example cervigram images used in our experiments depicting the high variability present in the data set. (b) a sample image with the corresponding $L^*a^*b^*$ feature space plot in (c).*

4 Experimental Results

For our experiments, we evaluate our method on 18 uterine cervix images obtained from the NLM/NCI archive (Fig. 3). The hardware for these experiments consisted of several machines equipped with different GPUs. Our first machine is an AMD 4600+ dual core with a Nvidia 8800 GTS. The second machine is an Intel Xeon 3.0 Ghz dual core with a Nvidia 280 GTX. We evaluate our results on several aspects: computational speed, analysis, optimality, and validity.

4.1 Computational Speed and Analysis

We present our timing and speed up results in Fig 4. Similar to the work by Cao [2] *et. al*, we find a comparable speed up of 3-8x when implementing K-means on the GPU. With mean shift, we are able to greatly outperform other parallelized methods [1, 15], mainly because these methods are implemented for multi-core CPUs. Because their performance scales linearly to the number of processing cores, to match the speed up factor of our GPU implementation, these systems would need nearly two hundred cores (based on a 1720x1172 image). Similarly with DA, the performance of [12] scales approximately linearly. Our GPU based method dominates these other methods in performance because of the number of streaming processors and high memory bandwidth that are present on GPUs.

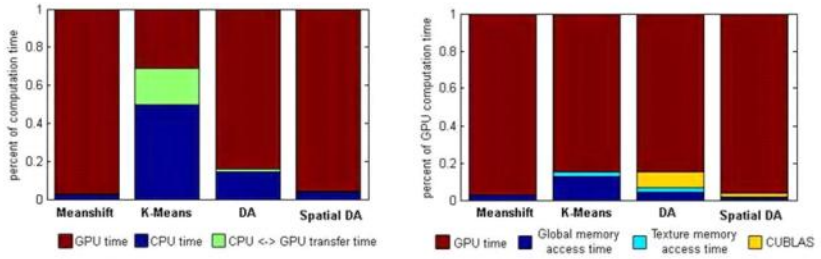


(a) K-means performance (b) Mean shift performance (c) DA performance

(d) Spatial DA performance (e) GPU Speed-up

Figure 4: *Execution times of the four clustering implementations when mean shift determined 11 modes, k=11 for all other methods. (e) Speed-up factor determined from slowest CPU to fastest GPU.*

Next, we analyze each method at each stage of the execution in a 880x600 pixel image to identify where the computational bottlenecks exist. The CPU and GPU execution times of the individual methods are compared in Fig. 5(a). From our results, we can deduce several key observations when parallelizing these clustering algorithms. First, it can be seen from the utilization graphs of both the mean shift and spatial DA methods that when the GPU time is highly utilized, the algorithm is offloading more computations to the GPU and can attain an overall greater speed up factor. Second, highly iterative methods can perform extremely well on GPU architectures when data transfer between the CPU and GPU can be minimized. For example, in our K-means implementation, this data transfer is unavoidable, which limits our speed up factor. Third, the utilization of specific memory structures, such as shared memory and texture memory rather than global memory can help in lowering the total execution time. We frequently benefit from texture memory in our implementations but would like to explore shared memory in future work. A detailed breakdown of the GPU executions is presented in Fig. 5(b).



(a) Comparison of Computation Time (b) Breakdown of GPU execution time

Figure 5: *Percentage of execution time for each process*

4.2 Optimality of Cluster Centers

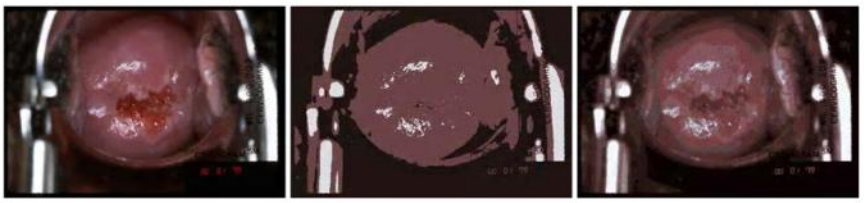
We define an optimal clustering as the minimization of the Euclidean sum of squared differences (SSD) between the pixel cluster assignment and location of the final cluster center in feature space. We compare different clustering methods and present our results in Table 1. From our mean shift clustering with window sizes, $h_l = 0.08, h_a = 0.03, h_b = 0.03$, we detect 17 modes in a 1280x872 cervigram image. In 250 K-means trials we randomly initialized the cluster centers, set the $k = 17$ and the convergence criteria to $\varepsilon < 0.001$. For the DA method, we ran 50 randomized inputs, set the $k = 17$ and the cooling schedule to $0.8T$, and computed the annealing steps until $T < 5 \times 10^{-5}$. We also ran our spatial DA method on 50 randomized inputs with $\sigma_1 = 0.25$ and $\sigma_2 = 1$ to demonstrate its optimality. The presented SSD energy is slightly higher than the spatial distortion energy, but for direct comparability, the SSD energy is reported.

Because the mean shift method takes every pixel as an input, there is no initialization randomization to the method and the average energy is static. From our results, the simplistic K-means method has very high variance, and is highly sensitive to its initialization as seen in Fig. 6(b). Conversely, our DA algorithm always produces an approximate globally optimal result regardless of initialization (Fig. 6(c)).

Table 1: Energy comparisons between different methods, (σ = standard deviation)

Method	# of Trials	Min Energy	Max Energy	Average Energy	σ = sd
Mean Shift	n/a[1]	n/a[1]	n/a[1]	9,019.59	n/a[1]
K-means	250	1,521.43	10,824.08	3,325.92	1,695.15
Determinstic Annealing	50	884.51	1,264.13	1,044.14	86.28
Spatially Coherent DA[2]	50	1,276.59	2,780.63	1,791.61	375.01

[1] *Mean shift is initialized with every pixel, thus, there is no deviation between trials*
[2] *Spatially Coherent DA distortion energy is calculated differently than the other methods. Basic SSD shown here.*



(a) Original (b) K-Means (c) Spatially Coherent DA

Figure 6: *K=13 for the different methods. (b) K-means result from a poor initialization. (c) Spatial DA method with the same initialization as K-means but reaches optimal clustering.*

4.3 Validity

To measure the validity of our clustering results, we compared selected AW tissue clusters with expertly segmented ground truths obtained from the NLM/NCI. We evaluated the p and q (sensitivity and specificity) and also the dice similarity coefficient (DSC) for 18 randomly chosen samples. The calculations of p, q, and DSC can be found in Popovic *et. al* [11]. Depending on the variation within each image, the number of modes detected by our PMDA method varied from 4 to 19. On average, our results are as follows: for the basic DA method, p = 0.711, q = 0.892, and DSC = 0.726. When we applied spatial coherency, our averages improved across all measurements. This was typically because the spatial method eliminated outliers and better delineated boundary conditions as seen in Fig. 7(c)(d). The most significant improvement was a 5.2% increase in the sensitivity. Our spatial coherent DA results were p = 0.748, q = 0.898, and DSC = 0.733.

5 Conclusion

In conclusion, we implemented and analyzed four clustering algorithms on the GPU using the CUDA programming language. We combined several of these methods and introduced a robust method, PMDA, for automatically segmenting cervigram images. We exploited the parallelism inherent in two algorithms, mean shift and deterministic annealing, using the GPU to achieve significant speed increases. We were able to achieve an approximate globally optimal clustering with relatively few resources (time and hardware). Also, a new spatially coherent extension to the deterministic annealing algorithm was introduced. Although we used cervigram images for our process, this clustering method can be extended for use on many different applications (e.g. histology images). In our future work, we would like to extend our algorithm to automatically classify cluster results and explore the use of *a priori* knowledge to more accurately segment and cluster image regions.

References

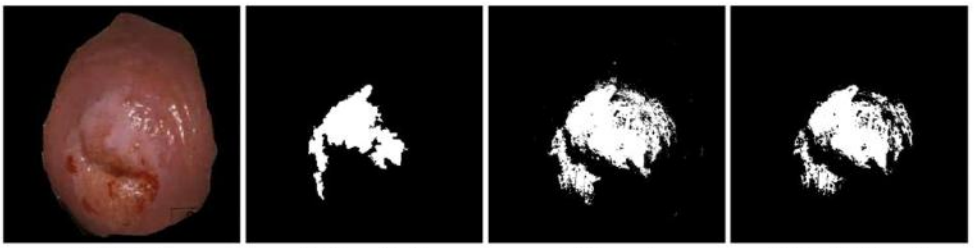
[1] J.G. Allen, R.Y.D, Xu, and J.S. Jin. Mean shift object tracking for a SIMD computer. *Third Int'l Conference on Information Technology and Applications*, 1:692–697, 2005. 1, 4.1

[2] F. Cao, A. Tung, and A. Zhou. Scalable clustering using graphics processors. *Advances in Web-Age Information Management*, 4:372–384, 2006. 1, 4.1

[3] W. Cho, S. Park, and J. Park. Segmentation of color image using deterministic annealing EM. *15th Int'l Conference on Pattern Recognition*, 3:646–649, 2000. 1

[4] D. Comaniciu and P. Meer. Mean shift: A robust approach towards feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24:603–619, 2002. 1, 3, 3.2

[5] E. Gabriel, V. Venkatesan, and S. Shah. Towards high performance cell segmentation in multispectral fine needle aspiration cytology of thyroid lesions. *HP-MICCAI Workshop*, 2008. 1



(a) Original (b) Ground Truth (c) DA result (d) Spatial DA result

Figure 7: *(a) Original image with (b) ground truth and selected regions (c)(d) identified as Acetowhite.*

[6] C. Gammage and V. Chaudhary. On optimization and parallelization of fuzzy connected segmentation for medical imaging. *Int'l Conf. on Advanced Information Networking and Applications*, 2:623–627, 2006. 1

[7] S. Gordon, G. Zimmerman, R. Long, S. Antani, J. Jeronimo, and H. Greenspan. Content analysis of uterine cervix images: Initial steps towards content based indexing and retrieval of cervigrams. *In Proc. of SPIE medical imaging*, 6144:1549–1556, 2006. 1

[8] C. Kulikowski and L. Gong. Progress in high performance medical imaging. *Int'l Conf. on Multimedia and Expo*, pages 284–287, 2007. 1

[9] A.W.C. Liew, S.H. Leung, and W.H. Lau. Fuzzy image clustering incorporating spatial continuity. *IEEE Proc. Visual Image Signal Processing*, 147:185–192, 2000. 1, 3.4

[10] Nvidia. Cuda: Compute unified device architecture programming guide. *Technical Report*, 2008. 1

[11] A. Popovic, M. Engelhardt D. La, and K. Radermacher. Statistical validation metric for accuracy assessment in medical image segmentation. *Int'l Journal of Computer Assisted Radiology and Surgery*, 2:169–181, 2007. 4.3

[12] X. Qiu, G. Fox, H. Yuan, S. Bae, and G. Chrysanthakopoulos. Performance of multicore systems on parallel data clustering with deterministic annealing. *Int'l Conf. on Computational Science*, 5101:407–416, 2008. 1, 4.1

[13] K. Rose. Deterministic annealing for clustering, compression, classification, regression and related optimization problems. *Proc. IEEE*, 86:2210–2239, 1998. 1, 3.3

[14] H. Takizawa and H. Kobayashi. Hierarchical parallel processing of large scale data clustering on a pc cluster with GPU co-processing. *Journal of Supercomputing*, 36:219–234, 2006. 1

[15] H. Wang, J. Zhao, H. Li, and J. Wang. Parallel clustering algorithms for image processing on multi-core CPUs. *International Conference on Computer Science and Software Engineering*, 2008. 1, 4.1

[16] W. Wang, X. Huang, Y. Zhu, D. Lopresti, L.R. Long, S. Antani, Z. Xue, and G. Thoma. A classifier ensemble based on performance level estimation. *In Proc. of the IEEE Int'l Symposium on Biomedical Imaging (ISBI)*, 2009. 1

[17] Z.M. Wang, Q. Song, and Y.C. Soh. MRI brain image segmentation by adaptive spatial deterministic annealing clustering. *In Proc. of the IEEE ISBI*, pages 299–302, 2006. 1, 3.4, 3.4

[18] L. Zhu, C. Wang, G. Zhu, B. Han, H. Wang, P. Huang, and E. Wu. Image spatial diffusion on GPUs. *Circuits and Systems, IEEE Asia Pacific Conference*, pages 610–613, 2008. 1

Using GPUs for fast computation of functional networks from fMRI activity

A. Ravishankar Rao, Rajesh Bordawekar, Guillermo Cecchi
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA
ravirao@us.ibm.com, bordaw@us.ibm.com, gcecchi@us.ibm.com

August 6, 2009

Abstract

The recent deployment of functional networks to analyze fMRI images has been very promising. In this method, the spatio-temporal fMRI data is converted to a graph-based representation, where the nodes are voxels and edges indicate the relationship between the nodes, such as the strength of correlation or causality. Graph-theoretic measures can then be used to compare different fMRI scans.

However, there is a significant computational bottleneck, as the computation of the functional network takes several hours on conventional machines with single CPUs. The study in this paper shows that a GPU can be advantageously used to accelerate the computation, such that the network computation takes a few minutes. Though GPUs have been used for the purposes of displaying fMRI images, their use in computing functional networks is novel.

We describe specific techniques such as load balancing, and the use of a large number of threads to achieve the desired speedup. Our experience in utilizing the GPU for functional network computations should prove useful to the scientific community investigating fMRI as GPUs are a low-cost platform for addressing the computational bottleneck.

1 Introduction

Research investigating the creation and analysis of fMRI imaging has grown dramatically in the past decade. The application of brain imaging to clinical studies of patients is an important goal in this field. Indeed, Illes *et al.*, observe that "Nearly 2000 fMRI empirical studies of depression, bipolar disorder and schizophrenia can be retrieved from Pub-Med today." [9].

With the clinical applications of fMRI imaging in mind, we create the following list of desirable attributes that a total fMRI solution should have. By a total fMRI solution, we mean a combination of the right analysis techniques on an appropriate hardware platform.

1. The fMRI solution should utilize analysis techniques that are applicable across a variety of experimental protocols, spanning the range from block designs to resting state data.

2. The fMRI solution should utilize analysis techniques that permit the measurement of relevant features from the fMRI data that are useful in a clinical monitoring and diagnosis setting.

3. The computational performance of the fMRI solution should be near real-time to enable rapid analysis and diagnosis, preferably while the patient is still in the scanner room.

4. The analysis technique should be implementable on low cost hardware, preferably such that the box that performs the processing can be co-located with the scanner. This reduces data transmission needs, and ameliorates patient privacy issues.

The main contribution of this paper is to present a promising approach that combines the advantages of a functional network approach [5, 6] with that of a low-cost hardware realization to deliver an effective solution that meets the above desirable requirements.

A recent approach that is gaining importance is the use of functional network based analysis, which is an alternative to standard GLM-based methods. However, the computational load of functional network based techniques is high. We overcome this by presenting an implementation of the delayed correlation algorithm [4] on a GPU platform to compute a directional network from raw fMRI data. GPU architectures are receiving increasing attention in many fields. Currently they're being used mainly for image rendering and visualization. However they can also be used for more advanced analysis of fMRI images.

Once the functional networks are computed, they can be subjected to different types of network motif analysis, which highlight statistical properties of basic network structures. A fruitful direction to explore is to examine properties related to cycles in the networks, as done in [3].

We present an approach to parallelizing the computation on a GPU of a very basic operation, that of extracting directed graphs from fMRI images. We highlight specific challenges that need to be addressed, such as memory access considerations, methods of optimizing parallelization, and the appropriate use of the various parameters offered by the GPU platform.

2 Background

The field of fMRI image analysis is dominated by techniques that use the general linear model, the GLM approach. Grinband *et al.* [7] identified 170 papers published in the first six months of 2007 on this approach alone in leading journals. Though the GLM approach is useful and has resulted in many insights into brain function, it has limited use in being applied across a wide variety of experimental protocols, especially in the use of resting state data [10]. Hence, the development of functional connectivity-based methods has been receiving increasing attention in the field [4, 6, 10].

A useful direction is to explore the graph-theoretic properties of such functional networks. This approach [5, 6] is based on the analysis of the structure of pair-wise correlations between voxels in fMRI, resulting in a graph structure whose nodes are the voxels and edges linking them are defined by the covariance exceeding a threshold. Investigations based on this approach demonstrate [6, 18] that the resulting networks have universal statistical topological properties, like scale-free connectivity and small-worldness [19], which are also present in other biological networks. A similar approach has been used to analyze and discriminate a variety of functional and dysfunctional brain states, confirming the utility of the functional network approach [2, 13, 16].

The primary application of GPUs in brain imaging appears to be in the image acquisition and rendering phases. Petrovic *et al.* [14] present a GPU-based solution to rendering and visualizing whole brain diffusion tensor tractography to allow interaction while maintaining visual quality. The system presented by [12] computes plausible fiber tracts from DTI data using a probabilistic model implemented on a GPU. Stone *et al.* [17] describe a method for improving image reconstruction quality in MRI.

Other applications of the GPU have been to speed up basic signal processing algorithms, such as the FFT [15].

3 Experimental methods

In this section we describe the creation of functional networks from fMRI time series data. We discuss GPU-specific implementation details that illustrate how such networks can be efficiently computed. We higlight major differences between GPU environments and MPI (message-passing interface) environments. (Further information about MPI is available at www.mcs.anl.gov/mpi/).

Both GPU CUDA threads and MPI threads are concurrent logical execution contexts that are mapped onto physical processors. The GPU threads are designed to operate in a massively data-parallel manner and conmmunicate via variables stored in shared address spaces. This makes them lightweight in terms of the state they carry. Furthermore, GPU threads operate in a lock-step manner whereby multiple threads execute a single instruction at every cycle. The thread scheduling algorithms are executed by the hardware. In contrast, MPI threads are designed to operate independently in a distributed memory environment and communicate via explicit messages. It is the user's responsibility to coordinate the threads, involving scheduling and synchronizataion. This requires MPI threads to carry more state. The lightweight nature of GPU threads enables one to run thousands of such threads on a single GPU device. In MPI, overheads such as the the cost of communication increase with the number of threads. Hence it is not advantageous to use a large number of MPI threads.

3.1 Creation of functional networks

We define a functional network by considering all functional voxels $\{v_i\}$ as possible nodes. Their covariance determines whether a binary functional link, or edge, exists between them: $c_{ij} = \langle (v_i(t) - v_i)(v_j(t) - v_j)\sigma_i^{-1}\sigma_j^{-1}$, where $v_i = \langle v_i(t)\rangle_t$ and $\sigma_i^2 = \langle (v_i(t) - v_i)^2\rangle$, such that if the correlation between i and j exceeds a threshold, C_T, a functional link is considered, and none otherwise: **if** $c_{ij} > C_T$ **then** $d_{ij} = 1$, **else** $d_{ij} = 0$. This approach was extended by Cecchi *et al.*in [4] to determine a possible directionality for the edge by considering the delayed or lagged covariance: $c_{ij}(\tau) = \langle (v_i(t+\tau) - v_i)(v_j(t) - v_j)\sigma_i^{-1}\sigma_j^{-1}$. We reason as follows: if there is a significant peak of the covariance between i and j at zero lag, then there is a potential binary symmetric link between them, as before. However, if the significant peak is not at zero lag, then we consider that the preceding voxel, and only it, has a directed link pointing to the succeeding one. That is, **if** $c_{ij}(\tau = 0) > C_T$ **then** $d_{ij} = d_{ji} = 1$; **else if** $c_{ij}(\tau < 0) > C_T$ **then** $d_{ij} = 1$ **and** $d_{ji} = 0$; **else** $d_{ij} = 0$.

In other words, two voxels whose activity is highly correlated and simultaneous are considered to be symmetrically linked. A voxel that is highly correlated with the future of another one will be considered as a "source", and the latter as a "sink". Though this approach can break the symmetry of the covariance, it cannot deal with the problem of transitivity as described in [4]. Nevertheless, this approach is useful in identifying most of the directional links.

The construction of the graph network from fMRI data is described in Figure 1. The typical data measurements consist of time traces of activity at each scanned voxel in the brain. This can be considered to be a time series, as shown in Figure 1. In Figure 1, we show the time traces of three voxels, V1, V2 and V3. Both V1 and V2 are driven by a common source, while V3 is driven by V1 with a delay. The right panel plots the delayed covariance analysis, by applying the equations above. Through this plot, we determine that there is a lagged correlation between V1 and V3.

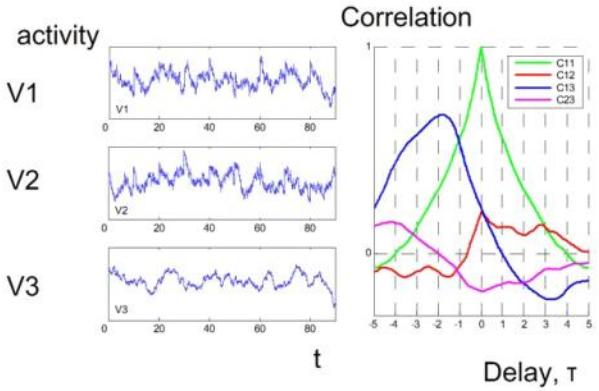
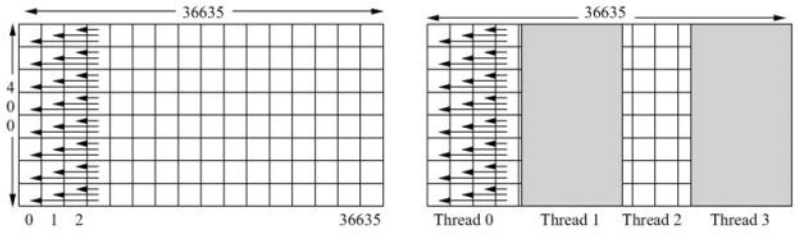


Figure 1: Illustrating the computation of the delayed correlation. The time traces of activities in three voxels, V_1, V_2 and V_3 is considered. Their delayed correlation is considered for different values of the delay, τ. In this case, both V1 and V2 are driven by a common source, while V3 is driven by V1 with a delay. This makes the lagged correlation between V1 and V3 evident, as shown by the curve C13. Positive values of τ are placed to the left of the origin, and negative values of τ are to the right of the origin. The correlation curve C13, indicates that the correlation between voxels 1 and 3 shows a significant peak at $\tau = -2$. We compute the significance by comparing the correlation value with a threshold, which is $CT = 0.5$. As can be seen, only the curve C13 shows a significant peak at a nonzero value of τ. Hence, we can assign a directional link from voxel 1 to voxel 3.

3.2 GPU-specific implementation

The Nvidia GTX280 GPU (Graphics Processing Unit) has a 240-core system-on-a-chip processor with 1 GB of off-chip, on-card device memory. GTX280's 240 cores are arranged as a group of 30 cores, called Symmetric Multiprocessors, with 8 processors each. In addition to the global device memory, the GTX280 GPU also supports a hierarchy of on- and off-chip memories, e.g., on-chip shared-memory, on-chip texture cache, and off-chip local and constant memories. The GTX280 supports single-instruction multiple thread (SIMT) programming model using its CUDA programming system. The CUDA programming language enables users to develop parallel applications for a GPU-based hybrid system. A section of the CUDA program executes on the host CPU whereas the computationally-intensive kernel is executed on the GPU using hundreds of threads. The CUDA runtime system then schedules the GPU application threads on 30 symmetric multiprocessors (240 hardware threads). Threads executing on a symmetric multiprocessor can share a 16 KB shared-memory. However, in practice, most applications have working sets larger than 16 KB, so majority of memory accesses are made to the global memory. Since accessing off-chip global memory is expensive, any GPU application usually uses a large number of threads to hide memory latency via overlapping computation and memory accesses.

Figure 2 illustrates the differences between parallelizing the correlation problem using massive data parallelism on the GPUs and coarse-grained parallelism using MPI. In the first approach, every column of the matrix is assigned to a separate thread. Thus the total number of threads used in the application equals the number of columns in the matrix (e.g., 36635 threads for a typical data set). The MPI application, in contrast, uses far fewer threads (16 to 32), where each thread processes a set of columns. The GPU application uses the shared device memory to communicate among the threads, whereas the MPI threads communicate



(a) GPU Execution using 36635 threads (b) MPI Execution over 4 threads

Figure 2: In this Figure, the time series data are organized such that each column represents a voxel, and contains a single time series over 400 time slices. The interval between successive slices is 2 seconds. There are 36635 voxels shown. The horizontal arrows indicate that the delayed correlation operations are performed between the i^{th} voxel and all the preceding voxels, from voxel number 1 to voxel number $i-1$. (a) Shows how the correlation is computed by using massive data parallelism on the GPUs. (b) Shows the same computation being performed via coarse-grained parallelization using MPI. Each thread processes a given set of columns, indicated by the shading.

via message-passing primitives.

Figure 3 shows a typical CUDA program, which consists of roles for both the CPU and GPU in performing the required computations. A CUDA program is a hybrid application, where part of the code runs on the host CPU and the compute intensive portion, called the kernel, executes on the GPU device. The CPU-specific component reads the input data, performs the pre-processing to generate the time series elements and stores them into data structures to be copied into the GPU device memory. Then it sets up the GPU execution parameters, e.g., number of participating threads and their mapping, initializes and copies GPU-specific data structures to the GPU device and invokes the GPU kernel. The GPU kernel first maps the data structures on the threads, executes the per-thread computation and copies the results to the device memory. Once the GPU kernel returns, the CPU component copies the data back to the host memory and returns.

4 Experimental Results

In this section we compare the results of applying parallel computation to functional network creation on different platforms.

Figure 4 illustrates the performance of the parallel fMRI code developed using MPI as the number of processors was varied from 1 to 8. As shown in Figure 4, the MPI version of the application demonstrates linear scalability as the number of processors is increased from 1 to 8: the execution time is reduced from 794 minutes to 101 minutes.

Figure 5 shows the effect of varying the number of GPU threads on the performance of the fMRI CUDA code on the Nvidia GTX280 GPU. The CUDA version of the fMRI code invokes the correlation function on the GPU. As shown in Figure 5 as the number of threads is increased from 1145 to 36635 (the thread count was chosen to match input image characteristics), the performance of the CUDA code improves from 1160 seconds to 348 seconds. As the input image data is large (around 200 MB), it has to be stored in the slower global memory. As a result, using larger number of threads is beneficial as it helps in hiding latencies of expensive memory accesses.

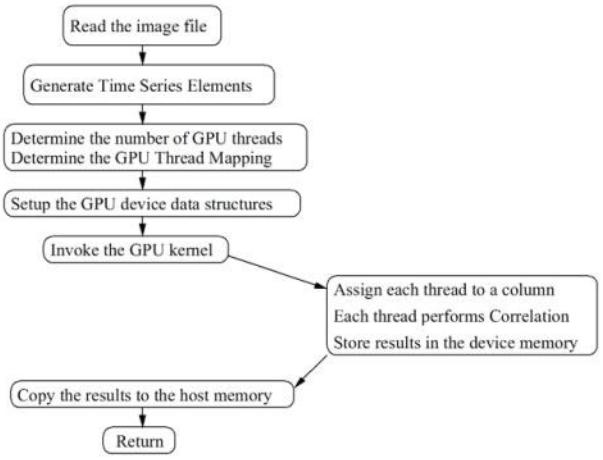


Figure 3: This flowchart shows how the CUDA code is run both on the host CPU and the GPU device.

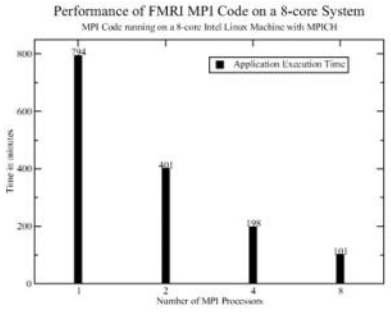


Figure 4: Performance of the fMRI MPI Code. The code was written in C, and ran on an 8-core Intel Linux machine that implemented the MPI (message-passing interface) standard. The specific implementation used was MPICH. The plot shows that execution time decreases as the number of processors utilized is increased.

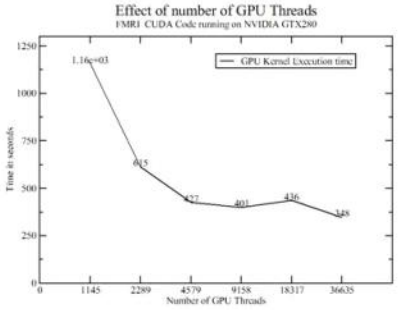


Figure 5: Effect of number of threads on the performance of the fMRI CUDA Code. Here, the CUDA code is run on the GPU platform as shown in Figure 3

CUDA also allows users to organize the application threads into groups of thread blocks, where each thread block executes on a different symmetric multiprocessor. Figure 6 illustrates the effect of arranging 36635 threads into thread blocks of sizes 64, 128, 256, and 512. In general, the more the number of threads in a thread block, the more likely it is to hide expensive memory accesses. Hence, the best performance is observed when the number of threads in the thread block equals 512.

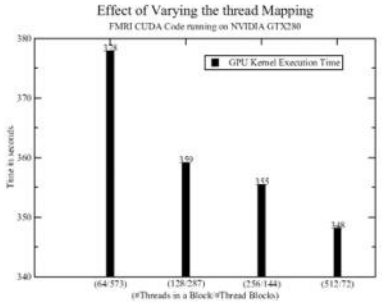


Figure 6: Effect of varying thread mapping on the performance of the fMRI CUDA Code. The execution of the CUDA code is described in the flowchart in Fig. 3

To summarize, using a single CPU thread, and 36635 threads on an Nvidia GTX280, the CUDA version of the fMRI application took about 6 minutes, whereas the MPI version, using 8 threads on a 8-core Intel Linux machine, took 101 minutes. This translates to a 17x accelaration with the fMRI GPU CUDA code over the MPI code. A single threaded version took 794 minutes, which gives the fMRI GPU CUDA code a 132x accelaration. This demonstrates the significant speedup that is achievable with a low-cost GPU platform.

5 Discussion

In this section, we discuss ways in which the GPU architecture can be efficiently utilized. We specially emphasize considerations in processing data loads typical of fMRI applications.

5.1 Memory intensiveness of fMRI applications

We define an fMRI kernel to be the block of code on the GPU that usually performs the correlation operations on large matrices (typically of size 400x40000 floats, or 64 MB in size). The correlation operation is implemented as a doubly-nested loop whose working set can match the entire matrix. The double-nesting arises from the need to compute pairwise correlations between voxels over multiple time lags. This makes the fMRI application memory-intensive. As the number of threads is increased, the application becomes even more memory-dominant. One way to improve memory performance is exploit the read-only texture cache of the GPU. Since, most of the arrays used in the fMRI kernel are read-only, they can be accessed via the texture cache. Unfortunately, the size of the texture cache is also small, which requires that the inner loop be carefully re-ordered to improve access locality. Furthermore, the correlation computation has an unusual access pattern that can cause load imbalance among participating GPU threads. The effect of imbalance becomes prominent when the number of GPU threads increases. As a result, the doubly-nested loop needs to re-ordered as per the thread parameters (i.e., number of threads and thread mapping) to balance the workload.

5.2 Impact of arithmetic precision on the computations

All NVIDIA GPUs with compute capability 1.3 (e.g., GTX280) follow the IEEE-754 standard for binary floating point arithmetic with certain deviations, such as the absence of a dynamically configurable rounding mode and absolute value/negation non-compliance. However, these issues do not affect the accuracy of the single precision fMRI imaging computations. The most recent GPUs, such as the GTX280 that we used in our experiments, also supports double precision floating point arithmetic. However, the gains in double precision performance are not as significant as the gains achieved through single precision performance. We expect that even when double precision is used, the GPU would perform significantly better than CPUs due to improved computational capabilities. For further details, the reader is referred to Appendix A2 in [1].

5.3 Guidelines for efficient GPU utilization

We provide the following guidelines to enable the judicious development of GPU applications.

1. Identify a computationally-intensive kernel that can be ported on the GPU. Ideally, this kernel should be computationally intensive and should have memory footprint that can fit into GPU's device memory.
2. The kernel should use regular data structures (e.g., arrays) that are amenable to massive data parallelism.
3. The kernel should be parallelized using the maximum mumber of (logical) threads possible.
4. The logical threads should be then mapped onto the GPU threads such that each symmetric multiprocessor (SM) uses the largest number of threads.
5. The computation should be reordered such that threads within an SM share data. The shared data structures can then be stored in the shared memory in each SM.
6. The inner loops should be unrolled either explicitly or by using the # pragma unroll feature to hide memory latencies.
7. Further performance improvement can be achieved by using the texture memory or the constant memory.

Overall, the performance of a GPU application depends on many inter-related parameters: number of threads, thread mapping, application working set, memory access patterns, etc. To obtain best throughput, the right parameters and configuration of the GPU need to be utilized. Our results show that appropriate parameter selection is critical to achieving success, as the performance is adversely affected otherwise.

The results we have obtained should provide guidance to other researchers in achieving a similar performance speedup on the GPU. For further documentation on developing GPU applications, the reader is referred to www.gpgpu.org. For NVIDIA specific information, the site forums.nvidia.com is useful. The CUDA SDK 2.1 in [1] has several sample applications.

5.4 Implications for future fMRI research

The development of fast GPU-based algorithms can signifcantly impact research in the fMRI imaging field. Functional network analysis techniques are promising, but are limited in practical applications due to their

high computational cost. The techniques presented in this paper offer a cost-effective route to overcome this limitation.

The results presented in this paper have been obtained by performing correlation analysis between voxels in a single brain scan. This can be trivially extended to compute the correlation between voxels in two different brains. Indeed, there is a growing need for such computations, as exemplified by the work of Hasson *et al.* [8]. Hasson *et al.* used movie sequences, and showed that there is an ordering of temporal receptive field sizes for different cortical regions. In addition, their work brings up the issue of computing *inter-subject* correlation matrices. In this light, our GPU based algorithm presented in this paper becomes even more significant, as it can be used for inter-subject studies across large populations, as well as for traditional intra-subject studies. The availability of fast, cost-effective implementations of the correlation computations should enable such large population studies to be carried out efficiently.

5.5 Patient privacy issues

The heavy computational load associated with fMRI image processing poses special challenges. If a total fMRI solution is desired, as explained in the introduction, patient privacy becomes an important issue. As mentioned by Li *et al.* [11], special care has to be taken in storing and transmitting medical images. Typically, medical institutions are reluctant to transmit medical images to remote servers or compute farms. This makes it necessary for the entire fMRI solution to be available on site, preferably co-located with the scanner, so that the need for image transmission is eliminated. A solution that achieves this will have the dual advantage of solving patient privacy issues as well as having a smaller footprint in terms of physical machine space. Indeed, the GPU solution that we have provided achieves this objective, as all it requires is a small GPU card to be plugged into the host computer.

6 Conclusion

This paper demonstrates the significant potential of applying GPU-based computing to fMRI image analysis. The combination of functional network analysis on the GPU platform meets several desirable requirements of a total fMRI solution that is intended for clinical use. The versatility of functional network analysis in handling data from a variety of fMRI experimental protocols can now be exploited on a low cost, high-throughput hardware platform offered by GPUs. We have offered several useful guidelines towards optimizing the computation of functional networks on a GPU architecture. Our algorithm and implementation will be useful for both inter-subject and intra-subject analysis of temporal fMRI sequences, and specifically for functional network analysis approaches.

References

[1] Nvidia cuda programming guide, version 2.1. http://www.nvidia.com/object/cuda develop.html.

[2] S Achard, R Salvador, B Whitcher, J Suckling, and E Bullmore. A resilient, low-frequency, small-world human brain functional network with highly connected association cortical hubs. *Journal of Neuroscience*, 26(1):63–72, 2006.

[3] G.A. Cecchi, A. Ma'ayan, A.R. Rao, J. Wagner, R. Iyengar, and G. Stolovitzky. Ordered cyclic motifs contributes to dynamic stability in biological and engineered networks. *PNAS*, 105:19235–19240, 2008.

[4] Centeno MV Baliki M Apkarian AV Chialvo DR Cecchi GA, Rao AR. Identifying directed links in large scale functional networks: application to brain fmri. *BMC Cell Biology, 8(1):S5*, 2007.

[5] S Dodel, JM Herrmann, and T Geisel. Functional connectivity by cross-correlation clustering. *Neurocomputing*, 44:1065–1070, 2002.

[6] V M Eguiluz, D R Chialvo, G A Cecchi, M Baliki, and A V Apkarian. Scale-free functional brain networks. *Physical Review Letters*, 94(018102), 2005.

[7] Jack Grinband, Tor D. Wager, Martin Lindquist, Vincent P. Ferrera, and Joy Hirsch. Detection of time-varying signals in event-related fmri designs. *NeuroImage*, 43(3):509–520, 2008.

[8] Uri Hasson and *et. al.* A hierarchy of temporal receptive windows in human cortex. *J. Neuroscience*, 28(10):2539–255, 2008.

[9] J. Illes, S. Lombera, J. Rosenberg, and B. Arnow. In the minds eye: Provider and patient attitudes on functional brain imaging. *Journal of Psychiatric Research*, 43(2):107–114, 2008.

[10] Kaiming Li, Lei Guo, Jingxin Nie, Gang Li, and Tianming Liu. Review of methods for functional brain connectivity detection using fmri. *Computerized Medical Imaging and Graphics*, 33(2):131–139, 2009.

[11] M. Li, R. Poovendran, and Y. Narayanan. Protecting patient privacy against unauthorized release of medical images in a group communication environment. *Computerized Medical Imaging and Graphics*, 29:367–383.

[12] Tim McGraw and Mariappan Nadar. Stochastic dt-mri connectivity mapping on the gpu. *IEEE Transactions On Visualization And Computer Graphics*, 13(6):1504–1511, 2007.

[13] S Micheloyannis, E Pachou, CJ Stam, M Breakspear, P Bitsios, M Vourkas, S Erimaki, and M Zervakis. Small-world networks and disturbed functional connectivity in schizophrenia. *Schizophrenia research*, 87(1-3):60–66, 2006.

[14] Vid Petrovic, James Fallon, and Falko Kuester. Visualizing whole-brain dti tractography with gpu-based tuboids and lod management. *IEEE Transactions On Visualization And Computer Graphics*, 13(6):1448–1495, 2007.

[15] Thomas Sangild Srensen, Tobias Schaeffter, Karsten stergaard Noe, and Michael Schacht Hansen. Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. *IEEE Transactions On Medical Imaging*, 27(4):538–547, 2008.

[16] CJ Stam, BF Jones, G Nolte, M Breakspear, and Ph Scheltens. Small-world networks and functional connectivity in alzheimer's disease. *Cerebral Cortex*, (Advanced Online Publ.), 2006.

[17] S.S. Stone and *et al*. How gpus can improve the quality of magnetic resonance imaging. In *Proc. 1st Workshop on General Purpose Processing on Graphics Processing Units, http://www.gigascale.org/pubs/1175.html*, 2007.

[18] M.P. van den Heuvel, C.J. Stam, M. Boersma, and H.E. Hulshoff Pol. Small-world and scale-free organization of voxel-based resting-state functional connectivity in the human brain. *NeuroImage*, 43(3):528–539, 2008.

[19] D Watts and S Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998.

A Self-Optimizing Histogram Algorithm for Graphics Card Accelerated Image Registration

Tom Brosch[1] and Roger Tam[2]

August 6, 2009

[1]MS/MRI Research Group, Faculty of Computer Science, University of Magdeburg
[2]MS/MRI Research Group, Department of Radiology, University of British Columbia

Abstract

This paper presents a new method to speed up image registration using graphics cards. In recent years, using graphics cards to implement image registration algorithms has become a reasonable way to reduce the computation time. While transforming images on the graphics card is rather simple, the computation of histograms as part of mutual information computation remains challenging. The performance of most algorithms depends greatly on the gray value distribution and is optimized for a small range of images only. In this paper a self-optimizing algorithm for histogram computation on the graphics card is presented which uses distribution information of the input images gained during a previous calculation step in order to adapt the histogram bin sizes to maximize the use of fast but limited local memory and avoid access collisions during subsequent calculations. The proposed approach has been evaluated using PD, T1 and T2 weighted MRI images of the brain revealing a four times speed-up on average compared to other graphics card implementations.

Contents

1 Introduction

Image registration is an important preprocessing step in medical image analysis. The task of image registration is to find a transformation T which applied to the floating image v, aligns the floating image with the base image u. Viola and Wells formulated this problem as an optimization problem [6] defined as

$$\hat{T} = \arg\max_{T} I(u, T(v)) \tag{1}$$

with I the *mutual information* (MI) between the base image and the transformed floating image. There are several ways to compute mutual information[3]. One definition is given by

$$I(A,B) = \sum_{a}\sum_{b} p_{A,B}(a,b) \log \frac{p_{A,B}(a,b)}{p_A(a)p_B(b)} \tag{2}$$

where $p_{A,B}$ is the joint probability density function (PDF) and p_A and p_B are the marginal PDFs. A good way to estimate PDFs are histograms. Joint histograms are used to estimate joint PDFs. After calculating the joint histogram, the histograms of image A and B can be computed by summing all bins along the rows and columns of the joint histogram. Hence, it is sufficient to calculate the joint histogram only in order to compute mutual information.

The optimization problem is solved by iteratively transforming the floating image and calculating mutual information. Depending on the size and dimension of the images used and the number of degrees of freedom of the transformation, registration can take several minutes up to several hours. Since image registration is a time-consuming task it makes great demands on the underlying hardware.

In the recent decade, graphics cards have advanced to powerful processing units and have become well accepted in the field of scientific computations – especially for complex simulations. This reduces the hardware costs by reducing the need for large computer clusters, but the development of GPU programs has still been labor-intensive. With the introduction of the *Compute Unified Device Architecture* (CUDA), NVIDIA made it easy to use the computational power of today's graphics cards for a large field of applications while reducing the development costs. So GPU implementations have found their way into medical image processing applications like image registration. Popular examples come from Shams, who presented in [4] a method for image registration on the GPU, which is 25 times faster than image registration on a single same-generation CPU. Other examples come from Chu [1]. He presented a method for non-rigid registration of breast MRI images using an adapted version of Shams' method for calculating histograms. This way, a 40 times speed-up compared to the original CPU implementation has been reached.

With CUDA, NVIDIA provides a technology to perform general purpose calculations on their graphics cards. The focus is to support the user in using this technique for highly parallel computing. The higher the number of independent operations that can be processed in parallel the higher the performance. There are also different memory units on a graphics card. The type of memory requiring the most consideration for optimization is the shared memory, which is the fastest memory but typically limited to only 16 KB. The remaining graphics card memory, that is usually refered to as global memory, is much larger (256 MB and more) but slower. Detailed information about CUDA can be found in the "Programing Guide" [2].

In this paper, a self-optimizing algorithm for computing a series of histograms, such as those used in image registration, is presented. The algorithm uses the knowledge of previous calculations to speed up the computation of following histograms by using the intensity distributions of the input images to adapt the histogram bin sizes in order to maximize the use of shared memory and avoid access collisions. This way, joint histograms can be computed up to five times faster on the GPU than other current GPU algorithms. In the next Section, we give a brief explanation of Shams' method [5], which is a recently proposed approach

for calculating histograms on the GPU. In Section 2 we present our approach, followed by a validation of our method and results in Section 3. The last section summarizes our conclusions.

2 Method

2.1 Histograms on the GPU

The calculation of histograms without parallelization is simple. The first step is to initialize all bins of the histogram to zero. Then for each pixel of the image the corresponding bin counter is incremented. A principal algorithm for the GPU is very similar. Instead of looping through the image, one thread per pixel is started which reads the pixel value and increments the corresponding bin counter. If two threads read the same intensity value they will increment the same bin counter so these operations need to be synchronized.

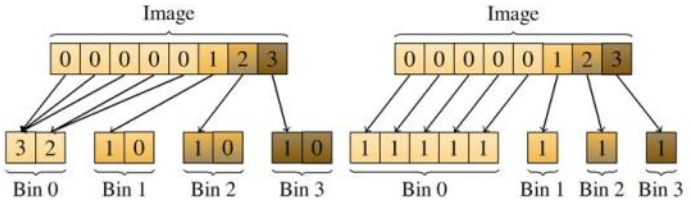
There are two principal strategies to overcome this issue. The most obvious solution is to let each thread calculate its own partial histogram. This method was first presented in [5]. After the calculation of partial histograms is finished all partial histograms need to be summed up to get the final histogram. In order to minimize slow global memory access all partial histograms should be placed in shared memory. The number of bits b that can be used for each bin can be calculated with Equation 3 ([5]).

$$b = \frac{s_{\max} \times 32}{B \times N_b} \tag{3}$$

where $s_{\max}$ is the number double words that fit into shared memory, B is the number of bins of the histogram and N_b is the number of threads.

Most of todays graphics cards have a shared memory of 16 KB which would fit 4096 double words. To maximize computational power the program should not have less than 128 threads, which corresponds to the number of calculation units typically available. For accurate MI computation with most medical image data, such as MRI, 128 bins per histogram would be considered a minimum, which would result in $128^2 = 16384$ bins for the joint histogram. Thus, there is less than one bit available for each bin so the fast shared memory cannot be used in a straightforward manner.

In order to maximize the use of limited memory, Shams presented a second method [5] where several threads share one bin. This involves the need for synchronizing thread access of conjointly used memory, which is not possible in general without synchronization primitives. However, since all 32 threads of a collective thread unit, called a thread warp, perform the same operation at any time, it is possible to simulate a mutex for groups of 32 threads. Every time a thread tries to write to shared memory the counter will be tagged with the thread ID. After writing, the written value will be read back and by checking the thread tag, each thread can check whether the writing operation succeeded and repeat it if necessary. The storage of tags reduces the number of bits that can be used for the counter itself. To comply with the shared memory alignment requirements, every bin is stored in a double word. Shams pointed out that the algorithms performs best with eight thread warps each calculating their own partial histogram. With a shared memory of 16 KB and eight histograms only 512 bins would fit into shared memory compared to 128^2, which are needed for accurate MI computation. To overcome this problem, only a specific bin range is calculated and the algorithm calculates each bin range iteratively.



(a) Without adaptive bin size (b) With adaptive bin size

Figure 1: Comparison of histogram configurations with and without adaptive bin size. Histograms with adaptive bin size use the available memory more efficient resulting in less thread collisions and less overflows.

2.2 Histograms with Adaptive Bin Size

Usually, the distribution of gray values is non-uniform. That means that some bins of the histogram need to be incremented more frequently than others resulting in many overflows and global updates for bins that are frequently used and many empty or nearly empty bins. Our algorithm adapts the histogram bin sizes to fit the image data. Bins that are more likely updated are allocated more bytes than others. Bins that are nearly never used are not cached in shared memory at all saving space for other bins.

This idea is illustrated in Fig. 1. In the example, a histogram is calculated with eight threads for an image with eight pixels. The first case shows a histogram with equally sized bins. Each bin is allocated two bytes, and each byte is filled independently. Because there are five pixels with the gray value zero, the threads collide while updating bin zero. On the other hand the bins one, two and three each occupy one byte that is not used. A histogram with adaptive bin size is able to balance bin updates much better avoiding collisions and unused memory completely.

2.3 Algorithm Details

The algorithm can be divided into two major parts – the calculation of an optimal histogram configuration and the calculation of histograms itself. The histogram configuration contains for each bin the number of bytes that are used to cache it in shared memory. Since the calculation of an optimal histogram configuration needs an estimate of the probability density function the first histogram calculation is not optimized. The optimization of the histogram configuration is time consuming, but since the histograms typically do not change very much during the registration process, the histogram configuration needs to be calculated only once at the beginning. So when many histograms are computed in succession the costs for the first calculation and histogram optimization amortizes well over the complete registration process. To facilitate an explanation of the optimization process, we first describe how we calculate histograms with adaptive bin sizes.

Calculate Histograms An overview of the algorithm is given in Algorithm 1. The first step is to initialize the histogram in global and shared memory. After initializing the memory the histogram is updated for each pixel of the image, with the number of available threads determining the number of pixels that can be processed simultaneously. Each bin has a *logical bin position*, which is determined by the gray value of the current pixel and the number of gray values which fall into the same bin. Then, the *bin parameters* for the current bin are read. These parameters specify whether a bin is cached in shared memory and if so how

many bytes are used to store it as well as the position in shared memory. In case a bin is spread over more than one byte each thread group increments a separate byte which is identified by the bin parameters and the current thread ID.

Before incrementing the bin it is checked for a potential overflow. The maximum value of the bin depends on the number of threads that share a bin, because some bits of the bin are used for synchronization. The more threads that share one byte the more bits are used for synchronization, reducing the maximum value of the bin. If an overflow needs to be handled each thread writes zero to the bin and the incremented value is updated in global memory, otherwise the bin is incremented in shared memory. A unique thread ID is also written in the upper bit range. If two threads try to write to the same memory location only one thread will succeed. After reading the written value the thread ID can be checked to test if the last writing operation succeeded. If the write operation does not succeed the procedure is repeated.

Algorithm 1 Calculation of histograms with adaptive bin size

```
1: initialize all bins of the histogram to zero
2: for all p, p ∈ image do {This loop is processed in parallel}
3:    get logical bin position b_l of p and the corresponding bin parameters
4:    if bin is cached then
5:       get physical bin position b_p depending on the bin parameters and thread ID
6:       repeat
7:          if bin b_p is full then
8:             try to set bin b_p to zero
9:             if last writing operation succeeded then
10:               update global memory
11:            end if
12:         else
13:            try to increment bin b_p
14:         end if
15:      until last writing operation succeeded
16:   else {bin is not cached}
17:      update global memory
18:   end if
19: end for
20: write cached values in shared memory to global memory
```

Histogram Configuration When incrementing a bin, each bin creates costs. The greatest costs are caused by collisions in shared or global memory and by global memory updates. The aim of the optimization is to minimize the expected costs. To achieve this, a bin level l_i is calculated for each bin i. The bin level indicates how many bytes are used for caching a bin in shared memory. A level of 0 indicates that a bin is not cached at all otherwise the used bytes are given by $b_i = 2^{l_i - 1}$. The expected costs are defined as

$$\text{Costs} = \sum_{i=0}^{n_\text{b}-1} c_{\text{gu},i} + c_{\text{sc},i} + c_{\text{gc},i} \tag{4}$$

with n_b number of bins, $c_{\text{gu},i}$ costs caused by global updates of bin i, $c_{\text{sc},i}$ costs caused by collisions in shared memory and $c_{\text{gc},i}$ costs caused by collisions in global memory by bin i. The total costs of global memory updates are estimated as the costs for one update c_gu times the expected number of global updates and given

by

$$c_{\mathrm{gu},i} = \begin{cases} c_{\mathrm{gu}} \frac{h_i}{b_{\max,i}} & \text{if } l_i > 0 \\ c_{\mathrm{gu}} h_i & \text{if } l_i = 0 \end{cases} \tag{5}$$

with h_i the value of bin i and $b_{\max,i}$ the number of different values that can be stored in bin i. The maximum number that can be stored in bin i depends on the number of bits that are used to store the bin counter. With a bin level of 1 32 threads share one bin so $\log_2(32) = 5$ bits are used to store the thread tag leaving only 3 bits to store the bin counter. Hence, $2^3 = 2^1 \times 2^2$ differnt values can be stored in a bin with a bin level 1. Increasing the bin level by 1 doubles the number of bytes that are used to cache a bin and halves the number of threads that share one bin reducing the number of bits used to store the thread tag by 1. Thus, one more bit can be used to store the bin counter doubling the number of different values of the bin. From this it follows that $b_{\max,i} = 2^{l_i} \times 2^2 = 2^{l_i+2}$.

Next, we calculate the expected number of collisions in shared memory. Consider we are given n threads which try to access the same bin i each with a probability of p. Then the probability $u_{k,i}$ that exactly k threads try to update bin i is given by

$$u_{k,i} = \mathrm{B}(k \mid p,n) = \binom{n}{k} p^k (1-p)^{n-k} \tag{6}$$

The expected number of updates of bin i is

$$\mathrm{E}[u_i] = \sum_{i=0}^{n} i\mathrm{B}(i \mid p,n) = np \tag{7}$$

This yields to the expected number of collisions given by

$$\mathrm{E}[c_i] = \sum_{i=1}^{n} (i-1)\mathrm{B}(i \mid p,n) = np + (1-p)^n - 1 \tag{8}$$

The number of threads n which try to access the same bin depends on the bin level. With a bin level of 1 $32 = 2^5 = 2^{6-1}$ threads share one bin. Increasing the bin level by 1 halves the number of threads that share one bin, so $n = 2^{6-l_i}$. The expected costs of bin i with respect to its bin level l_i is given by

$$c_{\mathrm{sc},i} = \begin{cases} c_{\mathrm{sc}} h_{\mathrm{sum}} \frac{np+(1-p)^n-1}{n} & \text{if } l_i > 0 \\ 0 & \text{if } l_i = 0 \end{cases} \tag{9}$$

with $h_{\mathrm{sum}} = \sum_i h_i$ and $p = h_i / h_{\mathrm{sum}}$.

Finally, we need the expected costs caused by global memory collisions. Analogous to the costs caused by collisions in shared memory these costs are given by

$$c_{\mathrm{gc},i} = \begin{cases} 0 & \text{if } l_i > 0 \\ c_{\mathrm{gc}} h_{\mathrm{sum}} \frac{np+(1-p)^n-1}{n} & \text{if } l_i = 0 \end{cases} \tag{10}$$

with $n = n_{\mathrm{T}}$, the total number of threads, and $p = h_i / h_{\mathrm{sum}}$. The costs caused by collisions in global memory due to overflows in shared memory are negligible and therefore ignored.

In order to find an optimal histogram configuration, each bin level is set to zero and the expected costs of each bin are calculated. The number of available bytes are set to the number of bytes used to store histograms

in shared memory. Then the *relative benefit* defined as the expected decrease of costs divided by the bytes that are needed to increase the bin level are computed for each bin. As long as there are bytes available, the bin level of the bin which promises the greatest relative benefit is incremented and the number of available bytes is decremented by the number of bytes used for the last optimization step. An optimal histogram configuration is found when there are no more bytes available or all bins have reached the maximum bin level 6 which means that no bins are shared by more than one thread.

2.4 Calculating Mutual Information

As outlined in the introduction, the computation of mutual information requires an estimate of the PDFs of both images and the joint PDF. Histograms and joint histograms can be used to provide such an estimate. In the previous section, we have shown a method to calculate histograms on the graphics card. Since the calculation of joint histograms is separable in each dimension, this method can be used to calculate joint histograms as well. Therefore, the 2D bin index of the joint histogram (x, y) is mapped to the 1D index i by $i = w \times y + x$ where w is the width of the joint histogram. The PDFs themselves are given by dividing each bin counter by the number of voxels of one image. Then, mutual information can be computed using Equation 2.

3 Results

In this section, we present the results of two experiments, one to compare the overall registration speed between our new algorithm and Shams' methods, and the other to analyze how much of the runtime of each registration iteration is occupied by histogram calculation. For both tests, we used a data set consisting of PD, T1 and T2 weighted MRI images of the brain of two patients with a resolution of $256 \times 256 \times 60$ voxels. All image pairs were formed using all combinations of images of the same patient. Between 300 and 400 iterations were necessary in order to register two images. Details about the graphics card that were used to perform all tests are summarized in Appendix A, Table 1.

During the first test, joint histograms are calculated for different numbers of bins for all image pairs of the test set using our approach and the two methods presented by Shams. Then, all results for the same number of bins and the same method have been averaged to get the mean runtime depending on the number of bins and algorithm. Since the first calculation of the method with adaptive bin size takes a lot more time (up to three seconds) than all following calculations, the joint histogram is calculated twice and only the runtime of the second calculation was taken into account. The results are summarized in Fig 2. The runtime of Shams' first and second method increases with increasing number of bins, while the runtime of our approach is nearly constant. With Shams' first method, only a fixed number of bin can be calculated at a time. So with more bins more iterations are needed to calculate the complete histogram resulting in a longer runtime. With the second method, all bins are filled during one iteration, but the amount of fast shared memory that can be used for each bin decreases with increasing number of bins, so the overall runtime increases too.

Our proposed method performed best during this test. Because only one iteration is needed to calculate all bins, it is faster than Shams' first method. In addition, the shared memory is used much more efficiently than with Shams' second method resulting in a shorter runtime when many bins are calculated. The more complicated calculation due to the interpretation of the bin configuration introduces some algorithmic overhead, which could have led to a worse runtime. Our tests have proven that this overhead is small in comparison to the benefit of an optimal histogram configuration so the algorithm is faster even with a very small number of bins. However, the difference becomes much more clear as more bins are used.

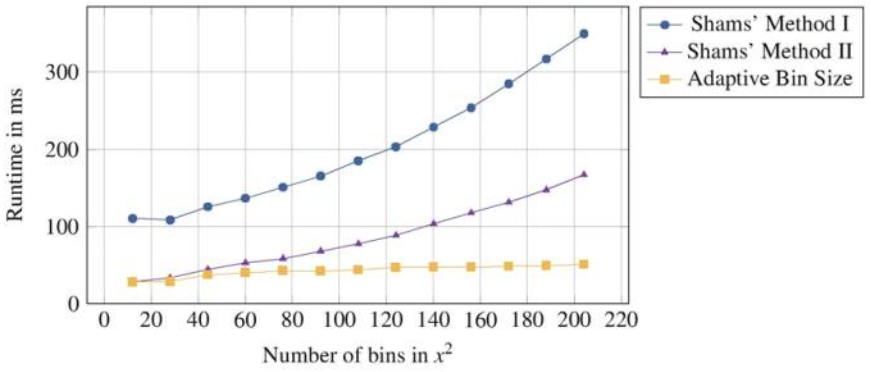


Figure 2: Comparison of the time need to compute mutual information using different histogram algorithms for increasing number of bins.

In the second test, we measured the proportion of the total runtime of the registration process required for histogram computation, and compared it to that required by transformations and computation of mutual information. Therefore, we registered each image pair of our data set using different histogram algorithms. Each joint histogram was calculated using 240×240 bins. The optimization algorithm used to maximize mutual information was a combination of Powell's method which uses Brent's method to perform the line optimization step. During each optimization step, the floating image needs to be transformed according to the new transformation parameters. Then the joint histogram of the base image and the transformed floating image was computed as a part of the mutual information computation. For each step, three intermediate times were recorded: 1) the time needed to calculate the joint histogram (this includes the time needed to calculate the optimal histogram configuration in case of our method), 2) the time needed to calculate mutual information including the time needed to calculate the joint histogram and 3) the two simple histograms and the time needed for one complete optimization step including the time needed to transform the floating image and to calculate mutual information. Then, for each method, the averaged time over all iterations and all images to perform each of these steps has been calculated. The results are illustrated in Figure 3.

The first observation is that for all algorithms the total runtime is dominated by the time needed to compute joint histograms. The time needed to compute MI excluding the time needed to calculate joint histograms and the time needed to transform the images are nearly negligible when performing these operations on the graphics card. In Section 2.3, we claimed that the costs for the first calculation and histogram optimization amortizes well over the complete registration process. Figure 3 supports this claim because, even though the calculation of the first histogram and the histogram calculation is included in the averaged histogram time, it is much faster compared to Shams' methods. Simply by using our approach to calculate joint histograms, image registration can be performed three to four times faster compared to other graphics card implementations.

4 Summary

Other publications have shown that GPU implementations can greatly enhance the performance of image registration compared to CPU implementations due to the vast computational power of today's graphics cards. While the calculation of transformations is rather simple on the GPU, counting and gathering al-

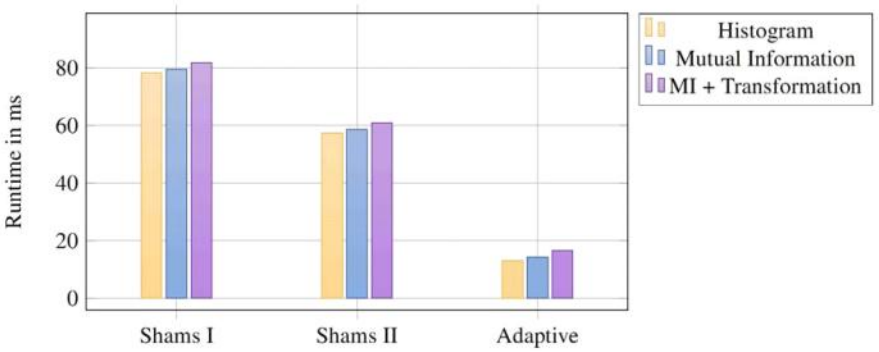


Figure 3: Averaged time to compute histograms, mutual information and one optimization iteration including the computation of mutual information and transforming the image for different histogram algorithms.

gorithms like calculating histograms are challenging due to the lack of synchronization primitives. There are some good solutions that depend greatly on the very limited amount of fast memory, and we have used this previous work as a starting point for designing our new algorithm. As with Sham's methods, our algorithm also computes histograms and joint histograms on the graphics card. The main difference is the way the shared memory is used. Shams used the shared memory to cache all bins of the histogram, no matter whether they are updated frequently or not. Our new approach tries to overcome this issue by calculating a histogram configuration based on a previously calculated histogram. Depending on the parameter difference between different optimization steps, the histograms are very similar. In case of large initial misregistation, a new histogram configuration calculation might be necessary after moving a specific distance in order to have optimal caching even at the end of the registration process. However, our evaluations have shown that for common registration problems such a resetting step is not necessary.

The difference in performance between CPU and GPU implementation is dominated by the differences in the computational power of CPUs and GPUs so such a comparison might not be appropriate to judge the quality of a GPU implementation itself. Therefore, we compared our implementation with other GPU implementations in order to provide informative results. We have shown, that it is possible to speed up common GPU implementations by improving the memory usage. It is possible to calculate an optimal histogram configuration, if a good estimate of the distribution is available. In common registration tasks, the calculation of many similar histograms is required, therefore a previous histogram can provide a good estimate of optimal memory usage for subsequent histogram calculations. Furthermore, we showed that calculating histograms is the bottleneck of image registration using mutual information. Even though the calculation of an optimal histogram configuration is very slow compared to the histogram calculation, the overall runtime could be reduced significantly by reducing the computation time of histograms. So implementing the calculation of an optimal histogram configuration on the GPU will further the reduce the runtime in the future.

A Appendix

Table 1: Graphics Card Specification

Model	NVIDIA GeForce 9600 GT
Processor Cores	64
Graphics Clock	650 MHz
Processor Clock	1625 MHz
Graphics Memory	1024 MB
Shared Memory	16 KB
Max number of threads per block	512
Warp Size	32

References

[1] Mei Yi Chu. *Mutual Information based Non-Rigid Image Registration using Adaptive Grid Generation: GPU*. PhD thesis, University of Texas at Arlington, 2008. 1

[2] C. NVIDIA. Programming Guide 2.0. http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf, 2008. 1

[3] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever. Mutual-information-based registration of medical images: a survey. *IEEE transactions on medical imaging*, 22(8):986–1004, 2003. 1

[4] R. Shams and N. Barnes. Speeding up mutual information computation using NVIDIA CUDA hardware. In *Proc. Digital Image Computing: Techniques and Applications (DICTA)*, pages 555–560, Adelaide, Australia, December 2007. 1

[5] R. Shams and R. A. Kennedy. Efficient histogram algorithms for NVIDIA CUDA compatible devices. In *Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS)*, pages 418–422, Gold Coast, Australia, December 2007. 1, 2.1, 2.1

[6] Paul Viola and William M. Wells III. Alignment by Maximization of Mutual Information. *International Journal of Computer Vision*, 24(2):137–154, 1997. 1

GPU-Based Elasticity Imaging Algorithms

Nishikant Deshmukh, Hassan Rivaz, Emad Boctor

Johns Hopkins University, 3400 N. Charles Street, Computer Science Department, 224 NEB, Baltimore, Maryland 21218, nishikant@jhu.edu

Abstract. Quasi-static elastography involves estimating the displacement field of the tissue undergoing slow compression. Since elastography is computationally expensive, many compromises have been made to perform it in real time. Parallelized algorithms and implementations for freehand palpation elastography are proposed in this paper to speed the computation. We present parallel implementations for Normalized cross correlation elastography algorithm and for Dynamic programming elastography algorithm. Both methods are implemented in CUDA® using Graphics Processing Unit (GPU). We are able to achieve above 300 images per second for NCC and around 30 images per second for DP elastography algorithm.

Keywords: Dynamic Programming, Normalized Cross Correlation, GPU, Parallelization, Ultrasound Elastography, Real time strain imaging.

1 Introduction

Elastography, the computation of the spatial variation of the elastic modulus of tissue, is an emerging medical imaging method with medical applications such as tumor detection [1].This paper focuses on static elastography, a well known technique that applies quasi-static compression of tissue and simultaneously images it with ultrasound. Through analysis of the ultrasound images, a tissue displacement map can be obtained [2, 3]. A least squares technique is then typically used to generate a low noise strain estimate from the displacement map [3].

In Ultrasound Elastography we initially take RF data sample with Ultrasound probe just above the surface of the tissue, we call this image as uncompressed image and the next sample we take by compressing the tissue and we call this image as compressed image. The resulting images are 2D matrices with columns representing the axial direction along the path of RF waves emitted out of the probe; we refer to it as RF lines.

In Normalized cross correlation (NCC) method we compare RF lines from compressed and uncompressed images for displacement in axial direction, since the

pressure is applied in axial direction and also since resolution is superior in this direction. We select a vectors of size $1 \times r$ from RF line in uncompressed image and vectors of size $1 \times s$ from RF line in compressed image such that $s > r$. Each window has 85% overlapping over each other along the RF line. These computations for small windows are independent of each other and can be computed in parallel. After calculating NCC we apply cosine fit interpolation, to get more approximate values, in the same thread in which NCC was calculated reducing the need to spawn a separate thread. The parallel algorithm will be discussed shortly.

Dynamic Programming (DP) is a numerical method for fast numerical optimization of the algorithms which are computationally intensive and complex in implementation. The most popular technique in DP is the top down and bottom up approach in which, in terms of Finite Automata, the present state depends on the previous state. This approach is particularly good for algorithms involving energy or cost functions which depend on the result from the previous stage. DP is hard to parallelize because of its dependence on the data from previous steps. With the introduction of many cores GPU's by NVidia and flexibility of NVidia CUDA programming environment it has become increasingly easy to parallelize existing implementations by using these environment. We thought of parallelization of NCC and DP Elastography on this hardware and utilize the computation power of these fast emerging compute capabilities and help the cancer research efforts in detecting and monitoring of tumors in real time.

2 The Parallel Algorithms

2.1 Parallel Algorithm for Normalized Cross-correlation

In NCC, every search window for comparison of the n RF lines in uncompressed data set (window size r) to the n RF lines in compressed data set (window size s) is computed in separate threads on the GPU [5]. The number of sample points per RF line is m. These threads since doing the same work are run in parallel. After applying cosine fit method to get sub sample interpolation, the result is a single pixel which is independent of other interpolation output pixel from other threads [5]. This output data independence eliminates the need to perform thread synchronization which makes the task highly inefficient [5]. The calculation of echo-strain, by applying median filter and low pass filter, and strain estimation, using least square methods, is calculated per pixel with every GPU thread assigned to every pixel and this method again gives data independence in output data pixels which ensures that the write operation can be done by each thread without interference [5]. To increase the

throughput we performed volume rendering by processing multiple images at the same time by combining them horizontally along the axial direction [5].

Depending on size of window and percentage of overlapping we get number of samples k in output image. So we spawn $k \times n$ number of threads to calculate strain which is also dimension of the output image. For volume rendering with p number of images the number of thread becomes $k \times n \times p$. We used fast Normalized cross correlation [10] given by

$$\gamma(u,v) = \sum_{x,y} \frac{\left[f(x,y) - f'_{u,v}\right]\left[t(x-u,y-v) - t'\right]}{\left\{\sum_{x,y}\left[f(x,y) - f'_{u,v}\right]^2 \sum_{x,y}\left[t(x-u,y-v) - t'\right]^2\right\}^{0.5}} \quad (1)$$

where f is the image and sums over (x, y) under t which is the matching template and t' is the mean of template and $f'_{u,v}$ is the mean of $f(x, y)$ in the region under the feature[10].

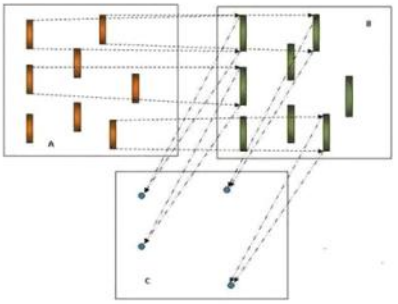


Fig. 1. A is uncompressed image, B is compressed image and C is output image. The windows in A and B are compared with each other giving out the pixel in C. All this comparisons are carried out in parallel on separate threads of GPU.

2.2 Parallel Algorithm for Dynamic Programming (DP) Elastography

DP Elastography is a known technique for calculating 3D echo strain image in medical imaging [4]. We focus on static elastography which applies quasi static compression of tissue [4]. DP Elastography is important since it has several advantages over the previous known elastography techniques. It reduces noise effect due to signal decorrelation between precompression and post-compression images and also reduces need for using large windows to reduce variance [4].

In 2D DP elastography [4] firstly we calculate the difference between two image signals g (i) and g' (i) along RF line and lateral direction.

$$\Delta(i, j, d_a, d_l) = \left| g_j(i) - g'_{j+d_l}(i + d_a) \right| \qquad (2)$$

Where $d_{a,\min} \le d_l \le d_{a,\max}$ is the displacement at sample *i* in axial direction (along the RF line) and $d_{l,\min} \le d_l \le d_{l,\max}$ is the displacement in lateral direction, and j = 1 to n and i = 1 to m, where m is the number of samples per RF-line and n is the number of RF lines [4].

A cost function is defined as

$$C_j(d_a, d_l, i) = \min_{\delta_a, \delta_l} \{ \frac{C_j(\delta_a, \delta_l, i-1) + C_{j-1}(\delta_a, \delta_l, i)}{2} + \alpha R(d_a, d_l, \delta_a, \delta_l) \} + \Delta(d_a, d_l, i) \qquad (3)$$

Where $R(d_{a_i}, d_{l_i}, d_{a_{i-1}}, d_{l_{i-1}}) = (d_{a_i} - d_{a_{i-1}})^2 + (d_{l_i} - d_{l_{i-1}})^2$ is an axial and lateral direction smoothness regularization term, α is a weight for the regularization, j is the sample number for i[th] RF line, δ_a and δ_l are values that minimizes the cost function are stored for d_a, d_l and i. The cost function is minimized at i = m and the d_i values that have minimized the cost function are traced back to i = 1, giving the d_i for all samples. More information is in [4].

Our approach involves vertical partitioning of the input image of size $m \times n$ with a specific width w. The width can be set arbitrarily depending on the degree of parallelization needed. We prefer vertical partitioning since comparison is done along RF lines between two images and compression is applied along the RF lines. After dividing the image we get a new set of images which are (1/w) [th] size of the original image or in other words we get $\lceil n/w \rceil$number of subsets. The starting RF line of these subsets now acts as the pivot by applying independent cost function on them. On these subsets we apply the DP Elastography. But applying this approach will result in formation of straps like output since at the start of each subset; the application of cost function which does not depend on previous cost function will make an independent disparity from previous RF lines and these two disparities will vary greatly. To reduce this effect we propose the following approach.

Stage I: We calculate the disparity until a fixed depth d for every subset and preserve the cost function for this RF line.

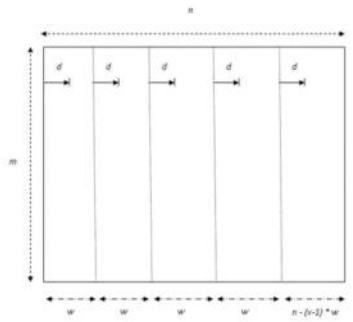


Fig. 2. Shows the Stage I, wherein we do DP steps until depth d at every interval of width w.

Stage II: Now considering the RF line in Step I at depth d for every subset as a new starting point and using its cost value as a pivot value we calculate disparity in both forward and backward directions (separate steps). We can say our original subset in Stage I got shifted right by distance d RF lines.

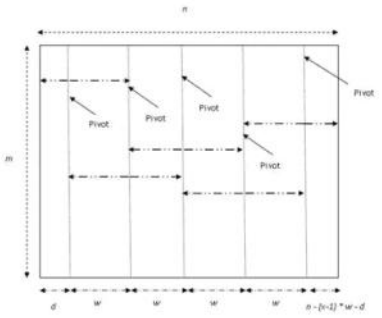


Fig. 3. Shows Stage II of DP Programming, after we have determined the new pivot values from Step I we apply DP algorithm forward and backward from these new points.

Stage III: Excluding the first subset and last subset, the intermediate subsets have two disparity values. We now calculate mean and weighted average of these disparities to get a fair disparity.

By using this approach we have successfully divided the data into $\lceil n/w \rceil$number of subsets giving us data independence. The forward and backward movement as in Stage II can be similarly run in parallel. The weighted average can be calculated

independently for each pixel for output disparity. Coming back to the DP algorithm, the value of $\Delta(i, j, d_a, d_l)$ is independent for every value of i and j, and hence can be calculated in parallel with $m \times n \times d_a \times d_l$ threads. On GPU random memory access is very expensive incurring 400+ clock cycles [11], this latency can be effectively hided by spawning large amounts of thread. $C_j(d_a, d_l, i)$ can be calculated for every j value across all RF lines for i = 1 to m, we have to repetitively call the kernel in this loop since cost function depends on cost function of i -1. Hence the number of threads called by each kernel is $n \times \lceil n/w \rceil \times d_a \times d_l$. Finally the cost function minimization can be done for each individual subset in parallel in $\lceil n/w \rceil$ number of threads. To improve the performance we have used software managed caching feature of GPU.

To get maximum throughput we do multivolume processing of data by feeding multiple images for processing. So if numbers of streams are p then the overall threads will increase by multiplying p with number of threads in every stage.

3 Results and Discussion:

We are using NVidia Tesla C1060 GPU card for our experiments which has 240 Streaming Processor Cores and 4GB of DDR3 RAM.

3.1 Normalized Cross Correlation

In GPU the memory access is fastest in case of cached memory followed by sequential access and then the random memory access [9]. NCC involves accessing a window of size r and s in uncompressed and compressed image respectively along the RF lines. Because of the availability of small cache inside GPU we decided to approach this problem by performing serial access over the image. To do so we read in the input image in transpose forms making the RF line a row instead of column. In this way we perform serial access on the RF lines in each thread.

Table 1. Figure shows comparison timings of Normalized cross correlation method using CUDA with corresponding C implementation for $k = 139$ samples, $m=641$ and $n=73$.

	CUDA		C	
No. of images (*p*)	**time***	**time/image***	**time***	**time/image***
1	0.0035	0.0035	0.0324	0.0324
125	0.3776	0.0030	3.8171	0.0305
250	0.7696	0.0031	7.6362	0.0305
375	1.1267	0.0030	11.4495	0.0305
500	1.8251	0.0037	15.2657	0.0305
625	1.8723	0.0030	19.0773	0.0305
750	2.3079	0.0031	22.9005	0.0305

*All timings in seconds

Table 1 shows there is approximately 8X to 10X performance improvements in CUDA implementation compared to standard C implementation.

Table 2. Figure shows timing comparison for samples for corresponding number of frames.

	1	125	250	375	500	625	750
32	0.0044	0.4977	0.9987	1.4934	2.0052	2.4910	2.9969
61	0.0047	0.5607	1.1313	1.6846	2.2937	2.8096	3.3883
69	0.0053	0.6353	1.2808	1.9121	2.6222	3.1576	3.8247
139	0.0035	0.3776	0.7696	1.1267	1.8251	1.8723	2.3079
147	0.0053	0.5696	1.1512	1.7001	2.5980	2.8279	3.4453
192	0.0077	0.8683	1.7581	2.5812	3.9171	4.3041	5.2716

*All timings in seconds, horizontal heading shows number of images p and vertical shows number of samples(k).

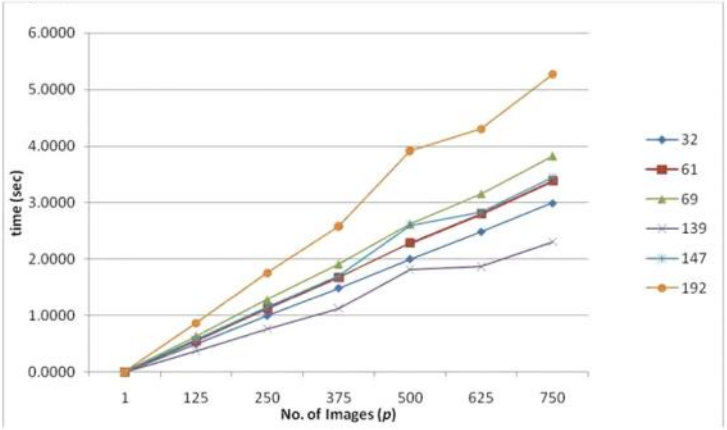


Fig. 4. Diagram shows plot for Table 2. We can see that we get optimal performance for k = 139 samples.

3.2 Dynamic Programming

GPU's global memory access is not through cache. So it incurs a memory access penalty of 400+ clock cycles for every Random access to the global memory [11]. It has a memory with small cache known as texture memory and additional constant memory with cache. Access to this constant memory cache is very fast [11] compared to the Global memory. Component $R(d_{a_i}, d_{l_j}, d_{a_{i-1}}, d_{l_{j-1}})$ to calculate C_j is calculated only once and is same for the whole image, according to (3) this component is used

extensively. Hence we copied this array into constant memory and achieved 1.5X performance. We did not needed to read the input image in transpose form because from (2) calculation of $\Delta(i, j, d_a, d_l)$ is performed in separate threads which masks the memory access latency. We tested our program with CUDA profiler and found that all of our kernels are performing 0 non-coherent reads which means all our multiprocessors are accessing memory in coalesced form and all threads are accessing consecutive memory locations [11] which means we have a very efficient kernel design and configuration. More information on NVidia Cuda is in [8].

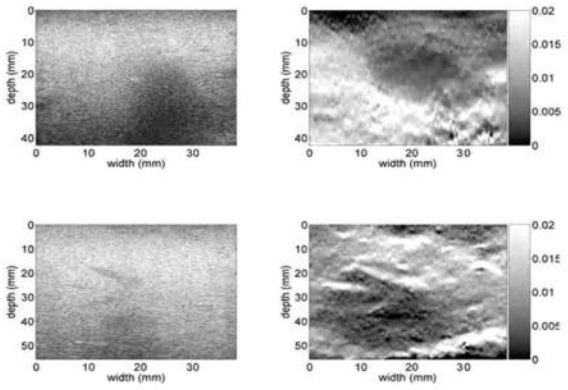


Fig. 5. On the right column, B-mode images of liver for two patients, and on the right column, the corresponding elasticity images are shown (from [6]).

Table 3. Table shows running DP code for different number of images in parallel and we can see that 200 images at an instance will give best performance.

	Timings in sec					
No. of Images	Stage I	Stage II forward	Stage II backward	Stage III	Total time	time/image
1	0.2860	0.4006	0.4294	0.0017	1.1177	1.1177
50	0.5866	0.8003	0.8743	0.0707	2.3319	0.0466
100	0.9427	1.2511	1.3934	0.1242	3.7114	0.0371
200	1.7547	2.1969	2.5555	0.2565	6.7636	0.0338
300	2.9633	3.5530	4.2858	0.3830	11.1852	0.0373
400	4.0215	5.0171	5.9079	0.5035	15.4500	0.0386
500	5.0205	6.3884	7.3990	0.6519	19.4599	0.0389

We ran our code for image of size m=1000 and n=100 with d_a = 20, d_l = 2, width w = 15 and depth d =10 and the result is in Table 3. We ran the code for C implementation for image of size m = 1000 and n=100 and the running is 0.2648 sec/image. From

Table 3 we can see that CUDA implementation gives improvement of approximately 6X – 8X compared to the corresponding C implementation.

4 Acknowledgements

The author would like to thank Mr. Matthew Bolitho, Dr. Philipp Stolka and Dr. Randal Burns for useful discussions.

5 Conclusion

GPU based parallelization provides opportunity for real time imaging and we have shown that we can successfully parallelize the present known techniques of Elastography for medical imaging. Sometimes it is important to expand the solution to get more opportunity for parallelization as seen in the case of Dynamic Programming Elastography. The results from Normalized Cross-correlation are encouraging with 300 images taking less than a second which is approximately 8 to 10 times the corresponding C implementation and Dynamic Programming based Elastography is having performance of 30 images per second which is 6 to 8 times the corresponding C implementation. GPU provides promising results but we have take in mind that memory access based computation can be performance degrading and special care needs to be taken to improve the performance of GPU based implementations using software managed caching and proper kernel configuration to get coalesced read.

6 References

[1] B. Garra, E. Céspedes, J. Ophir, S. Spratt, R. Zuurbier, C.Magnant, and M. Pennanen, "Elastography of breast lesions: Initial clinical results," Radiology, vol. 202, pp. 79–86, 1997.

[2] J. Ophir, S. Alam, B. Garra, F. Kallel, E. Konofagou, T. Krouskop, and T. Varghese, "Elastography: Ultrasonic estimation and imaging of the elastic properties of tissues," Annu. Rev. Biomed. Eng., vol. 213, pp. 203–233, Nov. 1999.

[3] J. Greenleaf, M. Fatemi, and M. Insana, "Selected methods for imaging elastic properties of biological tissues," Annu. Rev. Biomed. Eng., vol. 5, pp. 57–78, Apr. 2003.

[4] Rivaz, H., Boctor, E., Foroughi, P., Zellars, R., Fichtinger, G., Hager, G., "Ultrasound Elastography: a Dynamic Programming Approach", IEEE Trans. Medical Imaging Oct. 2008, vol. 27 pp 1373-1377

[5] E. M. Boctor, N. Deshmukh, M. S. Ayad, C. Clarke, K. Dickie, M. A. Choti, E. C. Burdette, "Three-dimensional heat-induced echo-strain imaging for monitoring high-intensity acoustic ablation", Vol. 7265, Medical Imaging 2009: Ultrasonic Imaging and Signal Processing.

[6] Rivaz, H., Fleming, I., Assumpcao, L., Fichtinger, G., Hamper, U., Choti, M., Hager, G., Boctor, E., "Ablation Monitoring with Elastography: 2D In-vivo and 3D Ex-vivo Studies", Medical Image Computing and Computer Assisted Intervention, MICCAI, New York, NY, Sept. 2008, pp 458-466

[7] J. Ophir, I. Cespedes, et.al. Elastography: A Quantitative Method for Imaging the elasticity of Biological Tissues. Ultrasonic imaging 1991; 13(2):111-34

[8] Cuda 2.2 Programming Guide and Reference Manual.

[9] GPU Gems 2, Chapter 32: Taking the Plunge into GPU Computing.

[10] J. P. Lewis, Fast Normalized Cross-Correlation.

[11] Victor Adrian Prisacariu, Ian Reid, fastHOG - a real-time GPU implementation of HOG, Technical Report No. 2310/09

20-24 September
MICCAI 2009
London UK

MICCAI-Grid Workshop

Medical imaging on GRID, HPC and GPU Infrastructures
Interoperability Highlights on NeuroSciences

Part 2 – GRID-Based Image Processing

MICCAI-Grid Workshop
http://proton.polytech.unice.fr/MICCAI-Grid/

Automatic annotation of 3D multi-modal MR images on a Desktop Grid

Curzio Basso[1,3], Marco Ferrante[1,2], Matteo Santoro[1,3] and Alessandro Verri[1]

August 3, 2009

[1]DISI, Università di Genova, Italy
[2]CSITA, Università di Genova, Italy
[3]CAMELOT Biomedical Systems Srl, Genova, Italy

Abstract

We present a distributed version of an algorithm recently proposed for the automatic annotation of 3D multi-modal MR images. The implementation takes advantage of the extremely parallelizable nature of the algorithm, both during the training and evaluation phases. Furthermore, the use of Python as implementation language allowed for the implementation of a framework allowing the easy conversion from the local, sequential version of the algorithm to the distributed one. The algorithm is deployed on a Desktop Grid infrastructure shared by a group of universities, made up of commodity desktop PCs.

Contents

In the last ten years supervised machine learning methods have proved considerably successful at tackling diverse types of problems, as long as they can be framed as either classification (assign a label to a previously unseen object x belonging to an input space $\mathcal{X}$) or regression (assign a scalar value rather than a label) tasks. The automatic annotation of medical images, that is the segmentation and classification of regions corresponding to specific tissues or anatomic structures, can be posed as a voxel classification task, and is therefore amenable to be approached with supervised learning methods. Some examples of these types of

works can be found in [4, 11, 15]. The key idea of supervised learning is to estimate a solution based on known examples; in the classification setting, we are given a *training set* of N labeled data $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathcal{X}$ are the data and y_i the labels. During the training phase, the training set $\mathcal{T}$ is used to estimate a solution $f : \mathcal{X} \to \mathcal{Y}$ which maps new data to the most likely labels, thereby *generalizing* the examples provided in $\mathcal{T}$.

The drawback of such methods is that, in order to provide bias-free solution and to avoid over-fitting, the training has to be repeated several times with different parameters, thereby requiring a considerable amount of computation. In the analysis of medical images this might be especially daunting, if more modalities and/or 3D data are considered. The starting point of our work, is the observation that this procedure is easily parallelizable, since it is a typical *parameter sweep* case, consisting of one or two nested loops of cross-validation. In fact, in the case of automatic annotation also the evaluation phase is conceptually easy to parallelize, since the classification of each voxel is independent from the classifications of all others.

We focus on an algorithm recently proposed in [3], aimed at the automatic annotation of inflamed synovia in multi-modal 3D MR images of wrists. The algorithm was originally implemented and tested in Python, and in this work we present its porting to a distributed environment. The distributed version of the algorithm has been deployed on a Desktop Grid infrastructure shared by a group of universities, made up of commodity desktop PCs.

Although conceptually easy, the actual adaptation of a pre-existing processing pipeline and its deployment on a distributed infrastucture is, in practice, a challenging endeavour. Indeed, in a typical situation the infrastructure and the pipeline have been developed in two different contexts, by two different research communities, with different goals in mind, and under different constraints. One of our goals is to explore mechanisms enabling a smooth and easy adaptation, under appropriate conditions, of legacy Python software to distributed infrastructures, and possibly a transparent switch between different ones, even if this result in a loss of functionality.

The software project that comes closest to satisfy our needs is PyMW [13], a Python module for parallel master-worker architecture. Altough having similar goals, the architecture of PyMW in its first release was quite different from the one implemented in our framework. Specifically, tasks to be distributed in PyMW are generic Python scripts or functions, tasks are submited one at time, and input and output seem oriented to primitive data types. Hovewer, latest version of PyMW, not yet available during our test, extends its feature in a direction similar to ours and in future we might consider some form of convergence.

The work presented here, as well as being an example of how medical image analysis can benefit from distributed computing, is also a proof-of-concept for a more general framework acting as a middle-layer between Python legacy software and a generic distributed infrastructure.

1 The Algorithm

We consider the algorithm presented in [3]. The system is aimed at the automatic segmentation of the inflamed synovia in multi-modal 3D MR images of wrists.

The whole pipeline is composed of four modules: multi-modal registration, intensity standardization, cues extraction and supervised voxel classification. Although all the steps might gain from computation on a distributed environment, our focus is on the last step, where we expect to reap the highest benefit. In this section we will provide a general overview of the classification method, with a special attention to the details most important for our topic, and refer to the original work for further details.

1.1 Rationale and Materials

The system works on MR images of pediatric patients affected by Juvenile Idiopathic Arthritis (JIA), a form of arthritis with unknown etiology. The segmentation of the inflamed synovia in volumetric images is a prerequisite for the quantitative measurement of its volume, which in turn is thought to be a marker of disease activity and whose monitoring might improve the patients' treatment and outcome.

The imaging protocol includes different modalities, all volumetric, performed with a Philips Medical Systems Achieva 1.5T scanner. Each MR study includes three T1-weighted 3D sequences: an FFE-SENSE acquired without contrast, followed by the injection of a contrast bolus and the acquisition of two 3D SPIR-SENSE, acquired at 3 and 10 minutes after the contrast injection respectively.

1.2 Voxel Classification

Due to the small number of positive voxels (belonging to the inflamed synovia) compared to the negative ones, the classifier is implemented as a two-stage cascade. At the first stage, S1, the voxels are coarsely classified based on their intensities and positions (*threshold-and-trim*); voxels passing the first stage are further classified with S2, a more advanced classifier, based on a sparse set of image features, selected from a large data-dependent dictionary of features.

S1 works by rejecting voxels not satisfying both of the following two conditions:

1. their intensities in the first post-contrast modality are higher than a lower threshold;
2. their position relative to the center of the image is within a given bounding box.

The optimal values of the parameters of S1, the lower threshold and the bounding box, are estimated during the training phase. In order to limit the risk of false negatives of S1, the threshold is chosen such that, on the training data, it achieves a 99.5% sensitivity, while the bounding box is chosen such that it includes all positive examples in the training set.

The second stage of the cascade classifier is a function $f(x)$ with positive values when a voxel x belongs to the tissue of interest and negative values otherwise. In practice the voxel x is described by a set of k image cues φ^j, possibly derived from more than one imaging modality: $x = \{\varphi^1, \ldots, \varphi^k\}$. As an example we might have a simple two-modality setting with two cues φ^1 and φ^2, being the intensities in each modality. For the case at hand, the cues representing the voxels are: (1) the intensities of the voxel and the set of its first neighbors; (2) the position of the voxel in millimeters, relative to the center of the acquisition volume; (3) the multiscale 2-jets as defined in [10], computed by convolution with the appropriate derivative of a Gaussian kernel; (4) the vesselness measure as defined in [12]. All cues except positions and vesselness are computed from all modalities, resulting in 14 different cues per voxel.

As explained in the introduction, the optimal classifying function is learned from the examples, that is we have access to a set of n exemplar voxels $x_i = \{\varphi_i^1, \ldots, \varphi_i^k\}$ for which we know the right labels y_i, equal to $+1$ if the voxel belongs to the inflamed synovia and -1 otherwise. We make the fundamental assumption that the classification function can be represented, without loss of generality, as the linear combination of $k \times n$ basis functions

$$K_i^j(x) = \kappa^j(\varphi^j, \varphi_i^j), \tag{1}$$

where each $\kappa^j(x, x')$ is a kernel function satisfying some basic condition ensuring that the result of the training is well-behaved. A well-known example of such a function, and the one we used, is the Gaussian

kernel

$$\kappa^j(\varphi^j, \varphi_i^j) = \exp\left\{-\frac{\|\varphi^j - \varphi_i^j\|^2}{2\sigma_j^2}\right\}. \tag{2}$$

The set of basis functions can be seen as a data-dependent dictionary $\mathcal{D}$ of features; the training phase will consist in the selection of the most relevant features in $\mathcal{D}$, and in finding their optimal combination for solving the classification problem.

Given the dictionary $\mathcal{D}$, the discriminant function will have the form

$$f(x; \alpha, b) = \sum_{j=1}^{k} \sum_{i=1}^{n} \alpha_i^j K_i^j(x) + b. \tag{3}$$

where b is a bias term and the coefficients of the sum build the vector $\alpha = (\alpha^1 \dots \alpha^k) \in \mathbb{R}^{kn}$, with each subvector α^j holding the coefficients relative to the basis functions of the j-th image cue. To simplify things, in the following we get rid of the bias term in (3) by assuming the data are centered.

Since the dictionary $\mathcal{D}$ is in general over-complete, we look for a method capable of selecting only a sparse subset of the basis functions in order to build our classifier. Denote by y the labels vector $y = (y_i)_{i=1}^n$, and by K the $n \times kn$ design matrix

$$K = \left(K^1 \cdots K^k\right), \tag{4}$$

where each K^j is a cue-specific $n \times n$ design matrix with elements

$$K_{il}^j = \kappa^j(\varphi_i^j, \varphi_l^j). \tag{5}$$

The goal is to approximate y with $K\alpha$ where α is sparse. There is growing literature on such methods, known as feature selection or sparse signal recovery. We decided to use the Orthogonal Matching Pursuit (OMP) algorithm, an iterative method which begins with $\alpha = \mathbf{0}$, and then improves the approximation by adding the basis function (a column of K) with the largest projection on the current residual; the algorithm stops when it reaches the required number of basis functions. We refer to [8] for a detailed description of the algorithm.

The feature selection stage is used to restrict the set of basis functions used for classification. As suggested in [5], in order to improve the classification performance we use the m selected basis to solve an ordinary ℓ_2-regularized least-squares problem. Denoting by $\widehat{K}$ the $n \times m$ matrix obtained by discarding the columns of K corresponding to null coefficients, and by $\widehat{\alpha}$ the m-dimensional vector of non-null coefficients, the optimal classifier is estimated by solving the problem

$$\min_{\widehat{\alpha}} \frac{1}{n} \|y - \widehat{K}\widehat{\alpha}\|_2^2 + \lambda \|\widehat{\alpha}\|_2^2. \tag{6}$$

1.3 Model Selection (Validation) and Testing

Both S1 and S2 depends on parameters estimated during training; the former requires the choice of the lower intensity threshold and of the bounding box, while the latter the selection of a subset of the dictionary $\mathcal{D}$ and the corresponding optimal coefficients $\widehat{\alpha}$. Clearly, the choice of these paramters depends on the training data, and in order to evaluate the performance of the algorithm one has to test it on data not used for training. This is done by performing a cross-validation procedure,

Normally, cross-validation is performed by splitting a certain number of times the training set, consisting in our case of voxels. However, given the variability across the MR images, it might not be safe to consider

Algorithm 1 Leave-one-out procedure for model selection and testing. Stages 1 and 2 of the cascade are denoted as S1 and S2.

for all patients $i \in \{1,\ldots,N\}$ **do** {outer **test** loop}
  train S1 on $\{I_l | l \neq i\}$
  for all patients $j \in \{1,\ldots,N\}/\{i\}$ **do** {inner **validation** loop}
    apply S1 to $\{I_l | l \neq i,j\}$
    draw a random sample of positives and false positives from step (4)
    for all $m \in \{\ldots\}$ and $\lambda \in \{\ldots\}$ **do** {selection and training}
      select m features with OMP
      train S2 by RLS with regularization coefficient λ
      validate S2 on I_j
    end for
  end for
  choose optimal parameters $m^\star$ and $\lambda^\star$
  apply S1 to $\{I_l | l \neq i\}$
  draw a random sample of positives and false positives from step (13)
  select $m^\star$ features with OMP
  train S2 by RLS with regularization coefficient $\lambda^\star$
  test S2 on I_i
end for

the voxels belonging to two different images as drawn from the same distribution. Therefore, our cross-validation procedure splits the training set patients-wise (so-called *leave-one-patient-out* scheme).

In order to obtain un-biased estimates, we need to adopt a slighly more complex schema. Indeed, the parameters of S2 actually depends on the choice of other two parameters, the number of basis functions m and the regularization parameter λ, whose optimal values are unknown. This results in a further cross-validation loop nested within the main one, which performs the selection and training procedure for S2 with different values of the pair (m,λ).

To summarize, each experiment is carried out with two nested cross-validation (CV) iterations, the outer one for estimating the performance (test) and the inner one for estimating the optimal number of features and the regularization parameter λ (validation). Both CV loops iterates over the patients, leaving all data of a patient aside for testing or validation, and using the others for training. This can be seen in algorithm 1.

2 Distributed Implementation

An important part of our work is the development of a light-weight Python-based framework that requires only minor effort by the developer to adapt their software to distributed infrastructures. Our experience showed that this is a valuable tool for the developers of client applications. The framework works under the assumption that each task in a job is independent from the others.

2.1 A general framework

Our framework is based on two Python classes, `Job` and `Grid`, which take care of all the logic involved in creating and launching jobs on the Grid infrastructures. What is left to the developer of the client application

Algorithm 2 Leave-one-out procedure for distributed model selection and testing. Stages 1 and 2 of the cascade are denoted as S1 and S2.

for all patients $i \in \{1,\ldots,N\}$ **do** {outer **test** loop}
  train S1 on $\{I_l | l \neq i\}$
  for all patients $j \in \{1,\ldots,N\}/\{i\}$ **do** {inner **validation** loop}
    apply S1 to $\{I_l | l \neq i,j\}$
    draw a random sample of positives and false positives from step (4)
    store the data for the task
  end for
  launch Grid job with $N-1$ tasks, one for each validation step
end for
for all patients $i \in \{1,\ldots,N\}$ **do** {outer **test** loop}
  collect validation results
  choose optimal parameters $m^\star$ and $\lambda^\star$
  apply S1 to $\{I_l | l \neq i\}$
  draw a random sample of positives and false positives from step (13)
  select $m^\star$ features with OMP
  train S2 by RLS with regularization coefficient $\lambda^\star$
  test S2 on I_i
end for

is the implementation of a subclass of `Job`, mostly just a wrapper of the code to be distributed, and to insert in the main function a few calls to methods of `Job`.

The methods implemented in `Job` are used to

- define per-task input data `Job.add_task()`;
- dump the data to a file `Job.dump()` in order to ship it to the nodes;
- create an archive with all Python modules the code depends on `Job.pack_deps()`, again for later shipping to the nodes.

The dependencies of the Python code, have to be defined by the developer in the subclass.

The duty of writing a job description and of launching the job is demanded to the subclass of `Grid` specific to the underlying infrastructure, through the methods `Grid.write_description()` and `Grid.launch()`. Note that due to the so-called *duck-typing* used in Python, there is no real need for a parent `Grid` class as there would be, e.g., in C++ in order to exploit a polymorphism mechanism. However, there still might be an advantage in sharing common functionalities between different Grid infrastructures.

The possibility of creating a job grouping a number of tasks, which is not implemented in PyMW, allows our framework to rely on the capabilities offered by some schedulers to exploit *data affinites* or other optimization strategies.

2.2 Application to the Automatic Annotation

The previous framework has been employed to take advantage of the Grid infrastructure. In the modified version each single training step with a fixed pair of parameters (m,λ) is executed as a single task on one

node of the Grid. In practice we substituted the selection and training loop starting at 6 in algorithm 1 with the steps 6 and 8 in algorithm 2.

We subclassed `Job` into a `SelectionTrain` class, for which we also defined its dependencies. For each single task we define as input data, via `SelectionTrain.add_task()`, the sampled positive and negative examples and the parameters of the training process; the task input data were dumped to disk as soon as they were added.

After having defined all tasks, the job object is passed as argument to `Grid.launch()`, in order to be submitted to the Grid. The method takes care of writing the job description specific to the underlying infrastructure and to submit the description for execution. In the case at hand, using the OurGrid infrastructure, the file was written in JDF format, as the example we report here; the syntax is pretty similar to the one used by other Grid software such as Condor. In this example `X` stands for the job id defined by the developer (we used the index of the outer testing loop, i in algorithm 2), and we assume just two tasks corresponding to an inner 2-fold validation loop.

```
job :
  label : select_train
  requirements : ( os == linux && mem >= 1024 )

  init :
    put select_train_data_X_0.dmp select_train_data_X_0.dmp
    put select_train_data_X_1.dmp select_train_data_X_1.dmp
    put select_train.py select_train.py
    put select_train_libs.tgz libs.tgz

  final :
    get output.log results/$JOB-$TASK-$PROC.stdout.log
    get error.log results/$JOB-$TASK-$PROC.stderr.log
    get results.tgz results/$JOB_$TASK_result.tgz

task :
  remote : python select_train.py select_train_data_X_0.dmp > output.log 2> error.log

task :
  remote : python select_train.py select_train_data_X_1.dmp > output.log 2> error.log
```

Note that the code performing the selection and training is not already present on the nodes, and it is rather shipped to them (file `select_train.py` in the `init` clause) together with the input data and other Python modules which it depends on (packed in the `libs.tgz` file).

The output of each single task is stored into a file which is shipped back to the client, named according to the job and the task identifying numbers. All outputs are then collected (step 11) and the algorithm proceeds as in the original version.

3 Test-bed Infrastructure

Our parallelization approach produces a set of indipendent tasks, suitable to run on an inexpensive, both in terms of equipments and skills, *Desktop Grid* (DG) infrastructure, which harvests idle time of commodity PCs such as the ones used in homes or in library/lab computer facilities [6]. Hovewer, the requirements of

the distributed version of the algorithm include the capabilities to deploy libraries on-the-fly and to execute arbitrary applications or code fragments.

In well known free-to-join *Volunteering Computing* DGs the research tasks must coexists with the owner activities, hence concerns about security and intrusiveness have suggested to limit the execution to trusted code. Some DG services, expecially BOINC based ones [1], execute only a single specific reseach activity, such as searching for prime numbers or protein folding simulations.

Few DG middlewares [14] [9] support the execution of arbitrary applications, under the assumption that all the hosts are managed by a single or a trusted organization. As Grid middleware, we choose OurGrid [7] since (1) it is multiplatform in both clients and computation nodes, (2) it can deploy applications on-the-fly, (3) does not rely on a shared filesystem, and (4) is easy to setup. Nevertheless, OurGrid implements a very common Grid architecture and present results could be extended to other middlewares with little or no effort.

OurGrid is used by the ShareGrid project [2], which involves several universities in the North-West Italy. In this way, our implementation can exploit up to 50 PCs in the students lab in our department, and more than 150 hosts in other partner sites. The exact number of available hosts can vary, and at the time of the experiments an average of 54 PCs were available to us.

A common issue that prevents the use of DGs in many scientific projects is the lack of boundary in task completion. The majority of DG client implementations execute the task during idle time only (usually as the screensaver) and suspend or kill the execution when the local user preempt the console. In busy environments such as student labs, this can result in tasks that frequently restart and never be accomplished.

To overcome this problem and security concerns, we developed a solution based on virtual machines. Each PC hosts an installation of free available VMware server running as a low priority background service [20]. As the majority of PCs produced in the recent past, the hosts are equipped with dual-core CPUs but the software used for educational purposes usually does not exploit multithreading, hence one core is usually underused and the impact on performance experienced by the local user is negligible. The guest virtual machines are deployed by cloning a template virtual disk file, installed with a Debian-based Linux distribution and provided with a base of familiar scientific software, such as GNU Octave, R, Python [16, 19, 18]. Scientific users can temporarily deploy other arbitrary scientific software inside virtual machines in a completly transparent way to the host environment. Host and guest configurations are arranged to avoid interference in either way, from local host user to the virtual machine and from the remote user of the virtual machine to the local host. Guest machines are *frozen* and every modification persists until the reboot only.

Another challenge was posed by networking requisites. Commonly, Grid implementations assume that node hosts are reachable from the public network. On the contrary, for several reasons, PCs in lab facilities can usually access the Internet through a proxy but cannot be contacted from the outside of the lab local network. OurGrid is based on a structured Peer-to-Peer network, where Grid nodes comunicate only with a super-peer also for file transfer. In this way, only the super-peer needs to comunicate with the outer networks. We decided to connect all virtual machines to the central super-peer using a Virtual Private Network based on opensource OpenVPN [17]. OpenVPN provides mechanisms for NAT-traversal, traffic encryption, mutual authentication and other techniques to deal with networking issues, so that the Grid middleware does not need to implement these features. OpenVPN has also the capability of starting an application, the node daemon in our case, once the VPN is estabilished. Since the virtual machines are cloned between PCs, all of them stored the same certificate(s). Altought the most natural choice would be to exploit the possibility offered by recent releases of OpenVPN to share a single X.509 digital certificate between hosts, this approach cannot be pursued since it does not guarantee a persistent pairing of distinct hosts with a network address. The problem can be solved by storing in the virtual machine prototype all the possible certificates and then picking the right one at runtime depending on the host machine. This does not

decrease system security with respect to the solution with a shared certificate.

4 Conclusion

The implementation of the distributed version would have required writing the appropriate description files for the given Grid infrastructure and modifying the legacy Python code. This process has been simplified by the general framework described in section 2.1 and it boiled down to a relatively small amount of changes in the code.

We first tested our system replicating one of the experiments normally performed to evaluate the results of the annotation algorithm. A full experiment with 5-fold outer and 4-fold inner cross-validation, ran locally on a quad-core Intel PC with 4GB RAM, took approximately 25 hrs and 45 min. Distributing the selection and training subtasks to the Grid, the same type of experiment took approximately 16 hrs and 31 min (wall-clock time). The distributed subtasks took virtually no time from the viewpoint of the client, since the results where ready in a shorter time then it took the client to be ready to process them for the final training step. Moreover, this last step was run in both cases locally on the client, and it took on average 11 hrs and 30 min; then, issuing 20 selection-training tasks to the Grid reduced to less than a third (29.8%) the computational time required for them.

Although this might seem a limited gain, it becomes definetely larger in experiments with more tasks. As a demonstration, we ran an experiment with both cross-validation loops as leave-one-out, resulting in $15 \times 14 = 210$ tasks. In this case, distributing the selection and training tasks on the Grid took 13 hrs 24 min wall-clock time, less than 10% of what we estimate (approximately 138 hrs) it would have taken to execute it locally in a sequential fashion. Note how an increase of more than ten times the number of tasks, resulted only in an increase of less than three times wall-clock time on the Grid infrastructure.

As a further remark, we note that most of the time has been spent in the sequential part of the algorithm that prepares the data for the tasks. Indeed, examining the statistics of the task one realizes that they typically took slightly less than one hour each. This would be the maximum performance in an optimal setting with no bottleneck in preparing the data and a sufficient number of nodes, with a best-case gain reduction to less than 1% of computation time.

In the experiments we ran, the percentage of tasks not completed because of the Grid workstations outages was extremely low, and all jobs finished successfully. This is a fairly interesting result, since this type of failures in Desktop Grids is quite common because of the non-dedicated nature of the workstations. We argue that the main reason for such success is the use of virtual machines as a stable and isolated excecution environment for the Grid nodes.

In the future we are interested in further exploit distributed infrastructures by also implementing the testing part of the algorithm as Grid jobs. This will be easier thanks to the framework we developed.

Acknowledgments

The research has been supported by the EU-funded project Health-e-Child, IST-2004-027749.

References

[1] D.P. Anderson. Boinc: a system for public-resource computing and storage. pages 4–10, Nov. 2004. 3

[2] C. Anglano, M. Canonico, M. Guazzone, M. Botta, S. Rabellino, S. Arena, and G. Girardi. Peer-to-peer desktop grids in the real world: The sharegrid project. pages 609–614, May 2008. 3

[3] C. Basso, M. Santoro, A. Verri, and M. Esposito. Segmentation of inflamed synovia in multi-modal 3D MRI. In *Proc. IEEE ISBI*, Boston, MA, USA, June 28 - July 1 2009. (document), 1

[4] P Bourgeat, J Fripp, P Stanwell, S Ramadan, and S Ourselin. MR image segmentation of the knee bone using phase information. *Medical Image Analysis*, 11(4):325–335, Jan 2007. (document)

[5] E Candes and T Tao. The Dantzig selector: Statistical estimation when p is much larger than n. *Ann Stat*, 35(6):2313–2351, Jan 2007. 1.2

[6] SungJin Choi, HongSoo Kim, EunJoung Byun, MaengSoon Baik, SungSuk Kim, ChanYeol Park, and ChongSun Hwang. Characterizing and classifying desktop grid. pages 743–748, May 2007. 3

[7] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006. 3

[8] G Davis, S Mallat, and M Avellaneda. Adaptive greedy approximations. *Constr Approx*, 13(1):57–98, Jan 1997. 1.2

[9] Gilles Fedak, Cecile Germain, Vincent Neri, and Franck Cappello. Xtremweb: A generic global computing system. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 582, Washington, DC, USA, 2001. IEEE Computer Society. 3

[10] LMJ Florack, BMT Romeny, JJ Koenderink, and MA Viergever. Scale and the differential structure of images. *Image Vision Comput*, 10(6):376–388, Jan 1992. 1.2

[11] J Folkesson, EB Dam, OF Olsen, PC Pettersen, and C Christiansen. Segmenting articular cartilage automatically using a voxel classification approach. *IEEE TMI*, 26(1):106–115, 2007. (document)

[12] AF Frangi, WJ Niessen, KL Vincken, and MA Viergever. Multiscale vessel enhancement filtering. In *Medical Image Computing and Computer-Assisted Interventation - MICCAI'98*, volume 1496, pages 130–137, Jan 1998. 1.2

[13] Eric M. Heien, Adam Kornafeld, Yusuke Takata, and Kenichi Hagihara. Pymw - a python module for parallel master worker computing. Technical Report pymw_tech_2008, Hagihara Laboratory, Department of Computer Science, Osaka University, 2008. (document)

[14] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988. 3

[15] RA Ochs, JG Goldin, F Abtin, HJ Kim, K Brown, P Batra, D Roback, MF McNitt-Gray, and MS Brown. Automated classification of lung bronchovascular anatomy in CT using AdaBoost. *Medical Image Analysis*, 11(3):315–24, Jun 2007. (document)

[16] GNU Octave. `http://www.gnu.org/software/octave`. 3

[17] OpenVPN. `http://openvpn.net`. 3

[18] Python Programming Language. `http://www.python.org`. 3

[19] R project. `http://www.r-project.org`. 3

[20] VMWARE. `http://www.vmware.com`. 3

A comparison between ARC and gLite for medical image processing on Grids

Tristan Glatard[1], Xin Zhou[2], Sorina Camarasu–Pop[1], Oxana Smirnova[4] and Henning Müller[2,3]

August 19, 2009

[1] Creatis–LRMN, University of Lyon, France
[2]Medical Informatics, University of Geneva, Switzerland
[3]University of Applied Sciences Western Switzerland, Sierre, Switzerland
[4]NDGF and Institute of Physics, Lund University, Sweden

Abstract

Medical imaging tasks often require large amounts of computing power or they could be improved if more computing power were available. Many medical institutions do not have any dedicated computing infrastructure for research and a way to cope with this is the use of computational Grids. These Grids can be used internally if the data can not leave the hospital network or from external infrastructure providers. Choosing/maintaining a Grid infrastructure can be a tedious tasks for researchers, as well as adapting existing applications for parallel computation on the Grid. Based on medical imaging use-cases, this article compares two widely-used middleware solutions, namely gLite and ARC (Advanced Resource Connector). Interoperability is enabled at the application level and the resulting setup is demonstrated on two use–cases combining resources from both Grids. In addition, experimental results show a simple performance comparison of data transfers and job submissions.

1 Introduction

Medical imaging is an essential part of medical diagnosis and treatment planning but processing large amounts of medical imaging data can be computationally very expensive. Only few medical institutions currently have large-scale computing infrastructures destined for imaging research, which led to the use of computational Grids in the medical imaging field [3, 15]. A variety of Grid middleware projects have been conducted over the past 20 years, from Condor [11], to Globus [6], gLite [7], and ARC (Advanced Resource Connector) [5]. For a researcher not familiar with computational Grids it is difficult to choose a particular middleware among the available solutions and most often the available resources determine this choice. Middleware comparisons, in particular for a concrete task (in this case medical imaging) are rare.

On the other hand, there are many ongoing efforts currently targeting middleware interoperability, so jobs can be exchanged, potentially easing the development of applications [20]. Regarding interoperability, problems range from very low middleware layers (e.g. interoperability among batch queues to build Grids) to higher levels (interoperability among Computing Elements to federate Grids [10]). At the application–level, there are also several motivations for interoperability:

- Sharing of applications to limit the Grid porting effort. Applications ported to a particular Grid platform can be run on another. In particular, this can be useful for widely adopted software tools.
- Sharing of data to enhance the accuracy of applications requiring large amounts of data for really meaningful results (e.g. content–based image retrieval [14]).
- Sharing of resources without an additional maintenance cost (e.g. to access very specific resources such as large clusters or clusters of Graphical Processing Units (GPUs)).

This article presents our early attempts towards application–level interoperability between ARC and gLite. Our goal is to provide a qualitative comparison for medical imaging applications. Experiments reported here are run from execution environments aiming at facilitating Grid access to non–expert users, i.e. medical image analysis researchers. Two specific environments are targeted, one being deployed at the HUG (University Hospitals of Geneva[1]) to interface with an ARC–enabled Grid resource and the other being deployed at CREATIS–LRMN (Centre de REcherche et d'Applications en Traitement de l'Image et du Signal — Laboratoire de Résonance Magnétique Nucléaire)[2] to give access to the EGEE Grid running gLite. In addition to the facilities provided by ARC and gLite, both execution environments include a workflow manager for application porting and an application–level job submitter. Execution environments and methods for application–level interoperability are first presented in section 2. Experiments for data–sharing and job submission are then reported in section 3.

2 Methods

One group (HUG) targeted data sharing and attempted to use data stored on EGEE from ARC resources in a content–based image retrieval (CBIR) application. A second group (CREATIS–LRMN) targeted resource sharing and attempted to run with ARC a radiotherapy simulation application originally ported to a gLite–based environment. This section first presents the two execution environments in 2.1. Setups for interoperability are then detailed in 2.2 (for data) and 2.3 (for jobs).

2.1 Execution environments

Both execution environments are mainly composed of a workflow (WF) description tool and a workflow engine enabling job submission, input selection, and data piping between jobs. Figure 1 summarizes the components adopted by the partners and shows how they interact with the grid middleware. A more detailed description follows.

HUG Grid setup

The HUG group has a particular Grid setup to assure that computation of data is also possible inside the hospital itself to avoid the transmission of sensitive medical information. Thus a small setup inside the hospital makes available computational power based on virtualization, Condor as computing node software and ARC to manage the jobs [16]. To ease the creation of parallel applications and gridify them the Taverna workflow system is used [17]. This also includes an ARC plugin to automatically submit the created jobs, following XRSL (eXtended Resource Specification Language) [23]. Besides the use of internal submission

[1] http://www.sim.hcuge.ch/medgift/
[2] http://www.creatis.insa-lyon.fr/

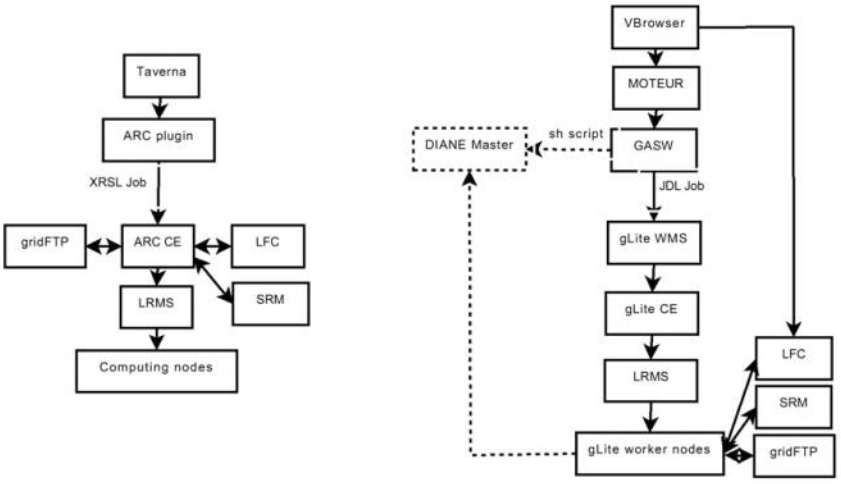


Figure 1: Overview of the Grid environments used by HUG (left) and CREATIS-LRMN (right).

interfaces the created applications can simply be submitted to external resources such as the KnowARC[3] virtual organization (VO) of the NorduGrid[4] infrastructure using the exactly same submission interface. In this case, jobs are handled by the ARC Computing Element (CE) and delegated to a Local Resource Management System (LRMS) that eventually schedules them on computing resources. In/output data is handled directly by the CE which pre-/post-stages files from/to storage systems. Supported file protocols include gridFTP, Logical File Catalog (LFC) and the Storage Resource Management (SRM).

CREATIS–LRMN Grid setup

The workflow description in the second setup relies on the Scufl (Simple Concept Unified Flow Language) language, generated using the Taverna workbench as a workflow editor [17]. Code is wrapped into workflow components using the Generic Application Service Wrapper (GASW [8]), which provides a basic command–line description language enabling input file staging, parameter specification, output file naming and transfer, as well as dependency specification. Workflows are then executed on gLite using MOTEUR [9] that generates, submits and monitors jobs on gLite complying to the Job Description Language (JDL) as figured by plain lines on figure 1. The main difference with ARC regarding job submission is that in ARC the site selection (matchmaking) is performed by the client whereas in gLite it is delegated to a global matchmaker (called Workload Management System — WMS). gLite's strategy is supposed to yield better scheduling while ARC's ensures better scalability. Moreover, with gLite, data has to be transferred by the job itself once it reaches the computing resource. A comparison of ARC and gLite job life–cycles is reported in [10].

Alternatively (dashed lines on figure 1), MOTEUR can also submit tasks to the DIANE pilot–job framework [13][5], offering a *pull* execution model supposed to improve performance on high–throughput systems.

[3] http://www.knowarc.eu/
[4] http://www.nordugrid.org
[5] http://cern.ch/diane/

In DIANE, tasks are no longer pushed to computing resources but generic pilots are submitted. Once running, pilots connect back to a central pool, fetching tasks when available and dying otherwise. Workflow input files and results are graphically browsed and selected on Grid storage resources using the Virtual Resource Browser (VBrowser) [18, 19]. Eventually, jobs execute on resources of the `biomed` EGEE VO, external to the institution.

Data is stored on gLite Storage Elements (SE) equipped with the Storage Resource Management interface (SRM) [1]. Data files are indexed in the Logical File Catalog (LFC), which maps application–specific logical file names to their physical locations.

2.2 Data interoperability for content–based image retrieval

Part of the medical data sets used for imaging cannot be deported outside of the hospital network for privacy reasons. On the other hand, CBIR relies on databases that can be stored on external resources, potentially belonging to another Grid and VO. The goal of this subsection is to enable the execution of applications developed on ARC with databases stored on EGEE servers.

Accessing the data stored on EGEE from the protected hospital network is not straightforward. Outbound connections in the HUG have several constraints: connections can only use port 80 (HTTP/HTTPS) and are always passing through a restrictive proxy server. Thus, communication with a Grid server outside of the hospital is hardly possible by default. To help scientific projects we were allowed to circumvent some of these restrictions by using a VPN (Virtual Private Network) connection towards the network of the University of Geneva. This network then has much lower security restrictions. The group also has two servers for data processing on the University network that are used for accessing other Grid networks.

The tested CBIR application can be divided into two parts: (i) downloading the data from the EGEE Grid servers to the University network, and (ii) executing the medical image analysis application on the internal hospital Grid. The latter is not related to data interoperability, we thus focus on the first step with the purpose of evaluating feasibility and possible overhead. Two client tools are tested for data transfer: the Java LFC client of VBrowser [18] and the ARC standalone client, which is also interfaced with LFC. Other related candidate tools for evaluation include the Grid Storage Access Framework (GSAF)[6] and JavaGAT [22].

2.3 Resource sharing for radiotherapy simulation

Computationally expensive simulation experiments often require large amounts of resources that may not be available on a single Grid at a given time. For instance, radiotherapy simulation [2] benefits from hundreds ($\geq$300) of concurrent CPUs (Central Processing Units). The goal of this section is to enable the execution of applications developed for EGEE on ARC resources. ARC resources under consideration are the ones provided by the NorduGrid infrastructure.

Two solutions can be envisaged for this execution using the execution environment described in section 2.1: (i) DIANE submits pilot-jobs to ARC (MOTEUR still submits tasks to DIANE), or (ii) MOTEUR directly submits jobs to ARC (DIANE is not used). Solution (i) provides more interoperability since every application relying on DIANE could then be executed both on gLite and on ARC. Besides this, the solution would easily enable a joint exploitation of ARC and gLite resources for a single application. On the other hand, (ii) provides better performance since the job generation can be adapted to a particular middleware. In practice, implementing (i) raises several technical issues.

[6] http://grid.ct.infn.it/twiki/bin/view/PI2S2/GSAF

Firstly, since tasks are only fetched when the job reaches a computing resource (so–called *late binding*), pre–staging of files cannot be implemented easily in a pilot–job framework. As a consequence, files need to be transferred onto the computing node by the task itself, which not only underexploits the features of ARC but is also technically heavy to implement since neither the data transfer client nor the user proxy are present on the computing node by default. In addition, it may lead to an unnecessary and uncontrollable overload of the storage service.

Secondly, in all cases, task generation by MOTEUR has to be adapted to the execution on an ARC computing node to accommodate, e.g., syntax differences in data clients. This is problematic given the late binding of tasks provided by pilot–jobs.

These reasons led us to implement solution (ii). The GASW was extended to support submission to ARC clusters. Beyond minor changes in job submission, monitoring, and status syntax, this required the adaptation of the job description format (from JDL to XSRL) and of the job content (from explicit to automatic data transfers).

Data transfers from EGEE to NorduGrid were performed using ARC's support for LFC (LFC locations can be specified in XRSL, the files being automatically transferred to/from EGEE resources). Because of the numerous ambiguities, only non–DPM EGEE SEs (Storage Elements) could be used, though[7]. A more important issue is that the VO–specific physical locations are not automatically generated by the generic ARC client, whereas it is done by the gLite LFC client for the registered EGEE VOs. The SRM output directory path thus has to be explicitly suggested in the configuration of the workflow manager that uses ARC, while only the SE host has to be specified for the EGEE infrastructure. This is potentially problematic in case of changes in the configuration of an EGEE SE (e.g. upgrade leading to change of the directory hierarchy or permissions).

Authorization of an EGEE user on NorduGrid clusters was easily performed by registering the X509 certificate in the `knowarc.eu` VO. However, being a member of two VOs led to some technical issues when information about the VOs is stored in the proxy itself (i.e. the proxy contains an extension obtained from the VO Management Service — VOMS). Due to the lack of a relevant specification that would formalize processing of multiple VOMS extensions, proxies containing two or more VOMS extensions are treated in an arbitrary manner by SRM services, often leading to data transfer errors. Since submission to ARC clusters does not require any VOMS extension, we coped with this issue by using a proxy with an EGEE VOMS extension only.

It has to be mentioned that the ease of installation of the ARC client greatly facilitated this implementation. The ARC client and gLite UI (User Interface) are easily able to be installed on a single Linux box. Well–packaged distributions like the one maintained by the Dutch VL–e (Virtual–Lab for eScience) project[8] now allow installing and configuring a gLite UI in ca. 20 minutes, including download and configuration. Because of its reduced dependencies, only 2 minutes were required for installation of an ARC client. This process was also less invasive and several Linux flavors are supported.

[7] SRM standard currently does not allow to identify neither the transfer protocol, nor the necessary end-point details such as port number, leading to incompatible implementations

[8] http://poc.vl-e.nl/

	data on EGEE		data on ARC	
	download	upload	download	upload
VBrowser	4523	1022	X	X
ARC–client	4451	997	4301	911

Table 1: Comparison of transfer speed (KB/s) to ARC computing resources for data stored on EGEE and ARC clusters with a catalog service.

	data on EGEE		data on ARC	
	download	upload	download	upload
VBrowser	339	116	X	X
ARC–client	361	110	345	112

Table 2: Comparison of transfer speed (KB/s) to ARC computing resources for data stored on EGEE and ARC clusters with Catalog service, using VPN in both cases.

3 Experiments and results

3.1 Evaluation of the access to data on EGEE and ARC clusters

The ARC standalone client is a Linux command line tool that offers not just ARC–specific job management functionality, but also some fundamental data transfer commands. Its interoperability with various data management services — either plain GridFTP servers, SRM or LFC — is demonstrated in [12]. However, in HUG, users are used to Windows–like graphical interfaces. VBrowser provides such an interface for data management. It also adapts necessary protocols to access the data on both gLite and ARC.

Two virtual organizations(VO) are used for this test: the `Biomed` VO based on the gLite middleware and the `knowarc.eu` VO of the EU KnowARC project. The test file comprises 40MB of a compressed image collection; physically it is located in Italy (the gLite server, `Biomed` VO) and Hungary (the ARC server, `knowarc.eu` VO). Two different data indexing services were used: the Globus Replica Location Service (RLS) [4] for ARC and the LFC catalog and indexing service for EGEE gLite. ARC can use both LFC and RLS for data indexing, while gLite currently supports only LFC. Incidentally, VBrowser cannot deal with RLS either, thus only the other interoperability possibilities were tested.

Tests are performed both on university network (Table 1) and using a VPN from the hospital network (Table 2). Client tools are installed both on a server located in University of Geneva and a workstation inside the HUG. For reasons explained beforehand, the communication from the HUG has to be through a VPN. A VPN encrypts the communication in both directions, which reduces the download/upload speed. When performing the tests from the university network the difference of speed is not significant and depends on the network speed (EGEE sites are network–wise closer to the HUG than the ARC ones). It should be pointed out that ARC offers a light–weight storage element named Smart Storage Element, which uses the HTTPS protocol for the data transfer. This can help reducing the communication overhead.

Regarding the comparison between VBrowser and ARC client, no significant difference was detected in terms of overhead. Comparison tests on larger data sets are part of our future work. Client tools with a windows–like GUI are regarded as more user–friendly. The ARC client, though lacking a GUI, supports more existing URL formats, which can be an advantage for users who want to access different storage systems.

	input archive	GATE release	GATE wrapper script	Output archive
size	1.6 MB	28 MB	7.9 KB	697 B
SE location	DE	GR	NL	BG

Table 3: GATE input/output file sizes and locations.

3.2 Joint execution of radiotherapy simulations on EGEE and ARC clusters

Using the setup described in section 2.3 we were able to execute on ARC resources a workflow initially developed on EGEE. The underlying application is GATE, a Monte Carlo simulation code currently used by more than 1000 users[9] and used here for radiotherapy simulation as described in [21]. Such Monte Carlo simulations are divisible–load problems, i.e. they can be divided into as many tasks as wanted. We here consider a 3h49'-simulation (average on ARC and gLite clusters used for the experiment) split into 50 jobs.

Each of the 50 tasks requires 3 input files and produces 1 output archive wrapping all the results. The job itself is wrapped into a script performing in/output data transfers (for gLite only), checking execution correctness and writing monitoring information such as total run time in the job console. Four gLite SEs spread all over Europe were used. File sizes and locations are reported in Table 3.

The experiment was repeated 5 times (experiments are coined batch 1 to 5 in the following). For each batch, 50 jobs were simultaneously submitted to ARC and gLite. To have similar matchmaking conditions, job submission was restricted to 3 NorduGrid sites and 3 EGEE sites. MOTEUR was configured to resubmit failed jobs up to 3 times. It should be noted, however, that although matchmaking conditions were comparable, ARC–enabled sites are voluntarily academic community contributions supported on a best–effort basis, while gLite–enabled sites were of a professional HPC grade, offering higher levels of service.

For each successful job, the submission, matchmaking, queuing, input transfer, running, output transfer, worktime and total round–trip times were measured as shown in Table 4. Some of the times were estimated from the job status reported by the Grid Information System (IS) and others were obtained from job or LRMS (Local Resource Management System) logs. In Table 4, the job states refer to the gLite[10] and ARC[11] user guides.

Because KnowARC and EGEE clusters used for the experiment have a different number of nodes and CPU characteristics, the total round–trip times cannot be compared. In particular, job queuing and running times are expected to be largely affected by those differences. Instead, we remove those two values from the total round–trip time to define a comparable Grid overhead defined as $\{5\} - (\{3\} + \{4.b\})$ referring to the notations of Table 4.

This comparable Grid overhead breaks down to the sum of the submission, matchmaking, input transfer, output transfer and *infrastructure overhead* (*ISO*). The latter measures the difference between the *real* job worktime (i.e. obtained from the job and/or LRMS stdout) and the worktime given by the information system, i.e., using notations of Table 4:

$$ISO_{gLite} = \{4\} - (\{4.a\} + \{4.b\} + \{4.c\}) \quad and \quad ISO_{ARC} = \{4\} - (\{4.b\} + \{4.c\})$$

Table 5 reports the comparable overhead of the successful jobs for the 5 batches and how it breaks down to submission, matchmaking, in/output transfer and ISO. The latter accounts for the largest proportion of the

[9] http://www.fgate.fr/
[10] http://glite.web.cern.ch/glite/documentation/userGuide.asp
[11] http://www.nordugrid.org/documents/ui.pdf

	gLite		ARC	
Measured time	Start state in IS	End state in IS	Start state in IS	End state in IS
{1} - Submission	Not submitted	Successfully subm.	Not submitted	Successfully subm.
{2} - Matchmaking	Submitted	Scheduled	not applicable	
{3} - Queuing	Scheduled	Running	Successfully subm.	INLRMSR
{4.a} - Input transfer	Job stdout		LRMS stdout	
{4.b} - Running	Job stdout			
{4.c} - Output transfer	Job stdout		LRMS stdout	
{4} - Worktime	Running	Completed	Running	Finished
{5} - Total round-trip	Not submitted	Completed	Not submitted	Finished

Table 4: Definition of measured times on ARC and gLite.

	Batch 1		Batch 2		Batch 3		Batch 4		Batch 5	
	gLite	ARC	gLite	ARC	gLite	ARC	gLite	ARC	gLite	ARC
Number of jobs	49	50	49	50	48	50	47	49	48	50
Subm. Mean (s)	3.7	15.5	3.6	14.8	4	16.1	4.3	14.4	3.8	13.8
Stdev (s)	0.87	12.2	0.86	9.9	0.97	13.1	1.6	10.3	1.1	10.4
Matchm. Mean (s)	26.6	0	637.4	0	27.9	0	25.2	0	28.8	0
Stdev (s)	6.8	0	862.7	0	7.4	0	6.5	0	6.5	0
In. trsf. Mean (s)	47.7	26.6	44.4	22.4	44.8	25.0	42.9	22.4	42.8	22.8
Stdev (s)	8.9	8.7	6.0	2.8	4.4	6.0	6.2	4.7	6.6	4.1
Out. trsf. Mean (s)	7.9	18.3	12.5	17.4	8.7	14.2	9.4	17	8.0	14.7
Stdev (s)	1.0	4.2	18.4	2.8	2.3	1.5	6.2	9.4	1.8	2.2
ISO Mean (s)	634.3	1255.4	333.7	1280.6	523.5	1257.8	694.2	1321	697.2	1242.3
Stdev (s)	635.3	x	516.0	x	531.4	x	610	x	537.8	x
Comp. over. Mean (s)	719.9	1289.4	1032.1	1312.4	608	1288	776	1352	781	1270
Stdev (s)	721.4	237.4	700.1	202.5	532.1	273.9	611.4	349	536.2	223.4

Table 5: Overhead comparison (mean and standard deviation values) between ARC and gLite on GATE radiotherapy application. Each batch corresponds to a repetition of the experiment. The number of jobs represents the number of successfully completed jobs (after up to 3 resubmissions) among the 50 jobs submitted simultaneously for each experiment. 'x' values are not available.

comparable overhead in both cases. In average, it is close to 10 minutes for gLite (576s) and 20 minutes for ARC (1271s). Note that the ISO also varies significantly, as shown by the standard deviation values (for the gLite infrastructure). If the infrastructure is overloaded, some jobs pass normaly while others suffer a very high ISO. Data transfers have similar performance both on ARC and gLite, which confirms the ability of the ARC client to efficiently handle files stored on EGEE, as shown in section 3.1. As explained in section 2.1, ARC does the matchmaking on the client side, i.e., during the submission process, which explains why the perceived submission time on ARC is higher than on gLite (about 4 times on average). However, the main result is that in all cases, the sum of submission and matchmaking times on gLite are significantly higher than on ARC. It shows that ARC's strategy is globally less penalizing than gLite's in our case. In particular, batch 2 shows that an overloaded WMS dramatically penalizes the experiment, which could not occur on ARC. On the other hand, one should keep in mind that gLite's strategy may lead to better scheduling, thus reducing the job queuing times in LRMS, which is not considered here. Moreover, ARC's strategy may also lead to scalability issues when several experiments are run from the same client.

4 Conclusions

In this work, we successfully implemented data and resource sharing between ARC and gLite. This allowed us to (i) run a CBIR application on ARC resources using data stored on EGEE resources and (ii) easily deploy on ARC an application developed for EGEE. This was tested in high–level graphical execution environments targeting medical imaging researchers. Both ARC and gLite support such solutions that are easy to use and can be of interest for researchers.

Scenarios are different for a research group that is inside a medical institution and research groups being on more open University networks, in particular concerning network connectivity. Data that can be treated inside and outside of medical institutions might also be different. Secondary data use in general is not easy as legal constraints often make it had to acquire data sets.

Beyond application–level interoperability, this setup enabled a comparison between ARC and gLite for medical imaging. Concerning data sharing, the main results are that (i) ARC's data transfer client manages to reach similar performance as gLite's to handle data stored on EGEE, (ii) the performance drop in using the VBrowser Java driver is not significant, (iii) whether data is stored on EGEE or on ARC does not significantly impact transfers and (iv) using VPN solutions to circumvent connectivity limitation in hospitals dramatically penalizes the performance. Regarding resource sharing, results show that (i) both for gLite and ARC, the infrastructure overhead accounts for most of the job latency and (ii) on the tested application, ARC's strategy of implementing matchmaking on the client side yields better performance than gLite's. All in all, this kind of study may be of importance in the current efforts for federating European Grids in a common European Grid Initiative (EGI).

5 Acknowledgements

We thank Piter T. De Boer for technical support on the VBrowser LFC client and David Sarrut for the help given to the porting of the GATE application on EGEE. This work has been funded (supported) by the EGEE–III INFSO–RI–222667 European project. This work was partially supported by the EU 6th Framework Program in the context of the KnowARC project (IST 032691) and by the Swiss National Science Foundation (FNS) in the context of the Talisman–2 project (project 200020 118638).

References

[1] A. Sim, A. Shoshani and others. The Storage Resource Manager Interface (SRM) Specification v2.2. GFD-R-P.129, 2008. 2.1

[2] J. Badel, L. Guigues, and D. Sarrut. ThIS : a Geant4-based Therapeutic Irradiation Simulator. In *1st European Workshop on Monte Carlo Treatment Planning*, 2006. 2.3

[3] V. Breton, R. Medina, and J. Montagnat. DataGrid, Prototype of a Biomedical Grid. *Methods of Information in Medicine*, 42(2):143–148, 2003. 1

[4] A. L. Chervenak et al. Performance and Scalability of a Replica Location Service. In *HPDC'04*, pages 182–191. IEEE Computer Society Press, 2004. 3.1

[5] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. Langgaard Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational Grids. *FGCS*, 23(2):219–240, 2007. 1

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997. 1

[7] F. Gagliardi, B. Jones, M. Reale, and S. Burke. European DataGrid project: Experiences of deploying a large scale testbed for e-science applications. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002*, pages 480–500, 2002. 1

[8] T. Glatard, J. Montagnat, D. Emsellem, and D. Lingrand. A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution. *FGCS*, 24(7):720–730, 2008. 2.1

[9] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *IJHPCA*, 22(3):347–360, 2008. 2.1

[10] M. Gronager, D. Johansson, J. Kleist, C. Sottrup, A. Waananen, L. Field, D. Qing, K. Happonen, and T. Linden. Interoperability between ARC and gLite . In *eScience*, pages 493–500, 2008. 1, 2.1

[11] M. Litzkov, M. Livny, and M. Mutka. Condor — a hunter of idle workstations. In *Proceedings of the 8th international conference on distributed computing*, pages 104–111, 1988. 1

[12] M. Mambelli. OSG Storage Elements and ATLAS DDM, 2008. 3.1

[13] J. Mosciki. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research A*, 502:426429, 2003. 2.1

[14] H. Müller, N. Michoux, D. Bandon, and A. Geissbuhler. A review of content–based image retrieval systems in medicine – clinical benefits and future directions. *Int. Journal of Medical Informatics*, 73:1–23, 2004. 1

[15] M. Niinimäki, X. Zhou, A. Depeursinge, A. Geissbuhler, and H. Müller. Building a community grid for medical image analysis inside a hospital, a case study. In *MICCAI-Grid'08*, pages 3–12, 2008. 1

[16] M. Niinimäki, X. Zhou, A. Depeursinge, A. Geissbuhler, and H. Müller. Building a community grid for medical image analysis inside a hospital, a case study. *FGCS*, 2009. 2.1

[17] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20):3045–3054, 2004. 2.1, 2.1

[18] S. Olabarriaga, P. de Boer, K. Maheshwari, A. Belloum, J. Snel, A. Nederveen, and M. Bouwhuis. Virtual Lab for fMRI: Bridging the Usability Gap. In *e-Science'06*, 2006. 2.1, 2.2

[19] S. Olabarriaga, T. Glatard, K. Boulebiar, and P. de Boer. From 'low-hanging' to 'user-ready': initial steps into a healthgrid. In *HealthGrid'08*, pages 70–79, 2008. 2.1

[20] M. Riedel, editor. *Int. Grid Interoperability and Interoperation Workshop*. IEEE, 2008. 1

[21] D. Sarrut and L. Guigues. Region-oriented CT image representation for reducing computing time of monte carlo simulations. *Med Phys*, 35(4), 2008. 3.2

[22] R. van Nieuwpoort, T.Kielmann, and H. Bal. User-friendly and reliable grid computing based on imperfect middleware. In *SC'07*, 2007. 2.2

[23] X. Zhou, H. Krabbenhöft, M. Niinimäki, A. Depeursinge, S. Möller, and H. Müller. An easy setup for parallel medical image processing: Using Taverna and ARC. In *HealthGrid'09*, 2009. 2.1

Plug–in Grid: A Fully Virtualized Grid Cluster

Release 0.00

Marko Niinimaki[1], Xin Zhou[1], Adrien Depeursinge[1], and Henning Müller[1,2]

July 27, 2009

[1]Medical Informatics, University Hospitals and University of Geneva, Switzerland
[2]Business Information Systems, University of Applied Sciences Western Switzerland, Sierre, Switzerland

Abstract

Medical image processing is known as a computationally demanding and data intensive field. For parallelizing the processing of image data, Grid computing systems and methods have been successfully applied. However, installing and maintaining Grid clusters is a demanding (and often non–rewarding) task for researchers. In this paper, we describe a Grid system that can be deployed completely within virtual machines on standard PC's. The system consists of a Grid/Cluster server and a large number of computing nodes. We discuss its features and demonstrate its performance with real–life tests using several medical imaging applications. Impact of the virtual machine with the computing nodes on the desktop speed are measured and compared on various computers and in various scenarios.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/1338]

Contents

1 Introduction

Modern hospitals produce ever–increasing quantities of data, much of it in the form of digital images, including tomography slices [15]. Medical doctors, analysts and researchers struggle with such an influx of data, particularly in hospitals where there are no dedicated computing resources for research, which is most frequently the case. Cluster computing and combining clusters by so–called Grid middlewares have potential in such environments, especially if the task at hand can be parallelized easily (see [18, 14]).

Harnessing the power of an organization's desktop PCs as a cluster is an intriguing concept, and implemented often using the Condor cluster software [10] in projects such as Greedy [19]. However, a hospital environment often has strict policies that prevent ad–hoc software installations on computers. To overcome this difficulty, projects like Grid Appliance [20] and CoreGrid [11] use a Virtual Machine (VM) within which the software is installed (also solving the problem that the process running in the virtual machine does not have access to the potentially confidential data on the client). As emphasized by Figueiredo et al. [5], this design isolates the application environment from the host PC, improving stability and security. The cluster software is, likewise, run inside the Virtual Machine. The KnowARC[1] project's GridFactory [17] presents a different design, where a cluster software starts Virtual Machine instances in computers, installs software and manages jobs in them. At the MedGIFT[2] research group of the Geneva University Hospitals (HUG), we have tested both approaches using a medical image indexing task as a case [13, 15, 17].

The design presented in this paper follows the same principles as in [15] — namely:

- A cluster of standard hospital PCs is running identical Virtual Machines, containing a compact Linux operating system with tools, applications, and a Condor worker node software (Software distribution is fully automatic via the Microsoft Active Directory–based hospital solution).
- A central server in the hospital runs a Grid middleware, gets jobs from users, sends them to worker nodes (using Condor), receives and stores the results. The results can then be retrieved by the user.

Contrary to our previous implementation, in the scenario described in this paper, we have isolated the Grid server as well, and it is run in a virtual machine. The virtual machine physically resides on an external hard disk, making the system fully portable. Moreover, due to the simple design of the Grid node VM image, it can be copied to any standard hospital PC, thus adding another node into the Grid resources. Another difference to the previous paper are the details tests of the impact of the virtual machine on the desktop performance of the host machine.

The most obvious benefit of this system is that with it the researchers in the hospitals can run analyses that would be impossible without a Grid/Cluster system (see Section Applications). The benefit for using a Grid layer on top of Condor allows the users to run their analysis in another ARC (Advanced Resource Connector) Grid outside of the hospital, if needed. The Taverna workflow engine [16] with its ARC plug–in [8] provides the users with a generic graphical interface for designing analysis tasks, without the need for detailed knowledge of the underlying Grid system.

[1] http://www.knowarc.eu/
[2] http://www.sim.hcuge.ch/medgift/

The rest of the paper is organized as follows. In Section 2 the characteristics of the Grid nodes, the central server and the environment are explained. Test scenarios for running performance and impact tests are described. In Section 3 we present the results of these tests. Section 4 discusses applications that utilise our Grid. Finally, Section 5 contains a short summary and discussion.

2 Methods

This section describes the environment and systems that are the basis for the work described in this paper.

2.1 Existing hospital environment

Like many medical institutions, the HUG do not have a dedicated research computing infrastructure. On the other hand, a very large number of desktop PC's is available, and the renewal cycle for these PCs is 4 to 5 years [21]. Currently (early 2009), around 6,000 computers are available, with the slowest ones containing 1 GB of main memory and a single Pentium IV 2.8 GHz CPU. By mid–2009 around 5,000 of the PC's will have at least 2 GB of main memory and at a minimum 2.6 GHz dual core CPUs. The dual core CPU supports virtualization in hardware, as well [7], leading to a much better performance.

As described in [15], 20 old hospital PCs in a seminar room were made available for us to create an intra–hospital Grid. These computers became computing nodes by virtualization. In addition to these computers, several desktop PC's of the researchers are used as additional nodes in the same way. These are the new generation dual–core PCs.

2.2 Grid setup at the HUG

Condor is used as the cluster software, NorduGrid's Advanced Resource Connector (ARC) [4] 0.6.5 is used as the Grid middleware for job submission and management.

A Debian Linux based Virtual Machine image, called Grid node, and Virtual Machine Player (VMPlayer[3]) are distributed in the hospital to a set of standard PC's running Windows XP. VMPlayer starts the Debian image when the PC is started, though the users of the PC can turn it off if they need more CPU power or memory for a particular task.

The Grid nodes are configured to use 350 MB of memory and a maximum of 2 GB of disk space. They receive an IP (Internet Protocol) address by DHCP (Dynamic Host Configuration Protocol) from the hospital's server.

The Grid nodes communicate with a (pre–configured) central server that runs the *Condor collector* process for sending jobs to and receiving results from Grid nodes. The central server also runs the ARC server. The ARC server manages Grid jobs by getting job descriptions from the users, submitting them to Condor, receiving the results, and storing the results to be retrieved by the user.

Figure 1 shows the overall structure of the infrastructure implemented in the hospitals. A central node stored on an external hard disk as virtual machine is controlling the working nodes that are in different computing rooms of the hospitals, also in virtual machines running Linux on standard desktop PCs.

[3] http://www.vmware.com/products/player/

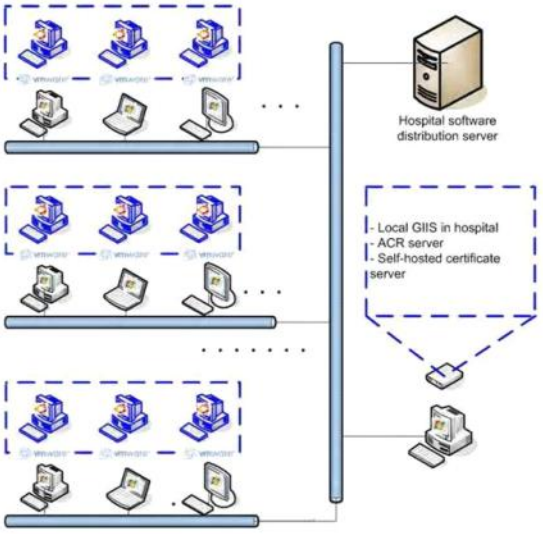


Figure 1: Overview of the infrastructure implemented for a fully virtual Grid, including the central node stored as virtual machine on an external hard disk.

2.3 Evaluating a fully virtual Grid

To test the system's performance and its impact to the computers, we have performed the following tests. It should be noted that here we test single tasks and their impact to individual computers. Section 4, however, discusses the performance of the Grid system as a whole.

- Performance (speed) test: compare the execution time of an image analysis job in a virtual machine with its execution time without virtualization.
- Impact tests on the desktop computers running the virtual machines:
  - Run a benchmark on the host computer (1) when there is no virtual machine running (2) when the virtual machine is running image analysis tasks. The benchmark program (with the virtual machine in case 2) is the only application running in the host computer during this test.
  - Compare the startup times of some typical applications in the host computer (1) when there is no virtual machine running, (2) when the virtual machine is running but idle, and (3) when the virtual machine is running image analysis at full capacity.

These tests were requested by the computer service of the hospitals to estimate the impact of the system on desktop users. Moreover, they allow estimation what kind of desktops could potentially be used for our internal Grid. We also measure network bandwidth and ways to limit this. The goal of this was also to evaluate the impact of such a system on the entire network.

	Internet Explorer 7	Microsoft Office 2003
VM not running	2.7	2.8
VM running but idle	3.2	3.3
VM running analysis	5.8	8.1

Table 1: Startup times of some of the hospital's frequently used applications on a single–CPU PC.

3 Results

This section details the results of the tests with the hospital Grid and its impact.

3.1 Performance measurements

As a simple but representative example, we have measured the execution time of one task in a virtual machine and in a native Linux operating system (installed on an identical computer). The task contains unpacking a *tar* file containing 100 images, compiling a feature extraction program, running the feature extraction for each of the images and combining the results in a *tar* file. This task is a very typical task for image retrieval that is extremely simple to parallelize.

On a single–CPU Pentium IV computer (old desktops used for our Grid) with a native Linux operating system this task takes 3 min 13 seconds, and on a Virtual Machine in the same computer 4 min 15 seconds. Interestingly, on the dual–core CPU computer (researchers personal machines) the execution time is almost identical in native Linux and in a virtual machine (1 min 43 seconds vs. 1 min 45 seconds).

3.2 Impact measurements

A criterion for the usability of the system in the hospitals has been that standard desktop PCs provided for hospital administrative staff and nurses can run Grid nodes so that the users do not notice a large detrimental effect on the performance.

With a single CPU system (our test Grid), this was not the case. Our measurements with the NovaBench[4] benchmark software indicated that the performance of a five–year–old standard hospital PC without the Virtual Machine was 35 MFLOPS (Million FLoating Point Operations per Second), 6.4 M integer operations/s. With the Grid node running but idle, the figures were 34 MFLOPS and 6.3 M integer operations/s. When the Grid node was running image analysis, the performance figures were 18 MFLOPS, 2.3 M integer operations/s [15]. In practice this meant that the startup time of the default WWW browser (Internet Explorer 7) degraded from 2.7 seconds to 5.8 seconds. The effect was even more noticeable in the startup times of Office applications, though hardly noticeable in text processing and spreadsheet usage once the applications were running. Table 1 shows the application startup times on a single CPU system.

With the new generation of dual–core processor PCs (90% of all PCs by mid 2009), Grid nodes run very well in the virtual machine, and they have only a very limited effect on the perceived performance of the PC, since they use only one of the processor cores. The startup times of the most commonly used application software packages are shown in Table 2. The respective figures with the NovaBench2 benchmark were 62 MFLOPS and 34 M integer operations per second when the VM was not running, 60 MFLOPS and 31 M integer operation when the VM was running but idle, and 56 MFLOPS, 28 M integer operations per second

[4] http://novabench.com/

	Internet Explorer 7	Microsoft Office 2003	Microsoft Office 2007
VM not running	2.7	2.4	4.2
VM running but idle	2.7	2.5	4.5
VM running analysis	2.8	3.0	6.1

Table 2: Startup times of some of the hospital's commonly used applications on a dual–CPU PC.

when Grid node was running an analysis job. In reality these differences are not really noticeable for an end user.

The improved performance of desktops has given us the possibility to run the central server inside a virtual machine as well. ARC software is relatively lightweight; an analysis of ARC 0.6.5's memory usage indicates that it consumes about 90 MB of the VM memory in operation (grid infosystem: 20 MB, gridftp 11.5 MB, grid manager 53 MB, scripts communicating with Condor 5 MB). However, *staging* jobs (copying data to and from Condor and making it available for the user) can take large amounts of disk space. Thus, we have prepared the Virtual Machine so that it can use disk space dynamically without limitations. The memory usage is set to 1 GB (50% of the dual–core CPU PC's physical memory). An inexpensive 1 TB external hard disk is used for storage and is also hosting the virtual machine itself.

Grid middlewares usually rely on a certificate–based authentication and authorization framework [6]. For a Grid server, the name of the computer (the fully–qualified hostname) must match with the subject of the computer's certificate — otherwise the communication with this computer is rejected by the Grid client software. In the hospitals' DHCP setup, a specific Virtual Machine is always given the same IP address. Naturally, the names of the computers are determined by the IP addresses. Therefore, we can physically move the external hard disk, containing the Virtual Machine image, to another PC if needed. Thus, we are creating a fully virtual intra–hospital Grid system, where the nodes as well as the controlling nodes can be moved on the standard desktops easily and quickly.

4 Applications

In this section, we present the medical applications that have utilized our hospital Grid and profited from the additionally available computing power. The current applications are limited to medical imaging but other applications such as natural language processing or data mining can easily be adapted to this scenario.

4.1 GIFT feature extraction

The GNU Image Finding Tool (GIFT[5]) has been used as a benchmark for computation in papers by the MedGIFT group as a scenario for content–based medical image retrieval [13]. The 50 0000 source images are from the ImageCLEFmed 2007 collection ([3][6]). For the computation, the collection is divided into packages containing a fixed number of source images, and the image feature extraction software taken from the GIFT software (GNU Image FindingTool). Previously, a 4–CPU local system was used for analysis, with an execution time of 709 minutes. By using the cluster of rather old desktop computers (much slower CPUs), an execution time of 240 minutes was achieved [15].

[5] http://www.gnu.org/software/gift/
[6] http://www.imageclef.org/

4.2 Lung tissue analysis with the TALISMAN software

TALISMAN (Texture Analysis of Lung ImageS for Medical diagnosis AssistaNce) is Java software with a front-end GUI and back-ends for distributed analysis [2]. Two main tasks were griddified for TALISMAN: feature extraction using wavelets, and image classification using Support Vector Machines. The source images that are analysed are high resolution computed tomography (HRCT) images of the chest.

Finding features that reveal lung illnesses is computationally very demanding and thus a distributed solution is much needed. In the analysis, the whole wavelet decomposition (convolutions) and feature calculation (mean and variance of the wavelet coefficients as well as grey level-histograms) were run on the Grid. The features are currently extracted from regions of interest in the images only. The region of interest was defined manually before the analysis (in our more recent version this is automatic).

In the application, we have done feature extraction for a series of images containing 30 slices (on average). The dimensions of each slice are 512x512 pixels. 58 series were used in the test.

The execution time of analysis in a single computer was more than 6 hours. By using ARC and Condor nodes in our cluster, the execution time was cut to 109 minutes.

Eventually, the features should be extracted on "per pixel" basis, creating a much larger need for computing power, currently not even attempted but possible through the Grid.

4.3 SIFT workflows with TAVERNA and ARC

The Scale–Invariant Feature Transform (SIFT) method is often used for feature extraction from medical images as well as for more general stock photography. A workflow process for this task was designed using the Taverna workflow engine [16]. Taverna communicates with ARC using an integration plug–in from the KnowARC project[7] [8]. The implementation was adapted from ImageJ's SIFT plug–in (see [1]). The source images were selected from the ImageCLEFmed 2007 collection. The running times in the cluster varied between 3 to 5 hours (meaning that running the whole analysis as one process would have taken more than 1 week). As SIFT features can have a large variety of parameters, which can largely determine performance, it is important for us to perform this systematic testing. By being able to test new parameters within hours instead of days we have many more options than beforehand. Potential end users of this system are surgeons who contacted us regarding a project on fracture image retrieval. This technology aims at being applied on fracture image retrieval that is described in [12] for a first pilot application.

5 Conclusions and discussion

In this paper, we demonstrate the feasibility of an internal Grid in a hospital environment using standard hospital desktop PCs. The Grid system, including the server, is fully based on virtualization, and standard hospital PC's are used as computing nodes. The benefit of this setup, compared to our previous one, is that is it very portable, does not require a specialized setup in any node, and its performance is still good.

As of now, the system can be best described as functional prototype with a small but active set of users. In order to present the system to larger user community, many political and technical problems would still need to solved, among them providing intuitive interfaces for non–programmers (emphasized e.g. in [9]).

To summarize, the Grid system works as follows:

[7] http://www.knowarc.eu/

- The building blocks of the system are the Condor batch system and NorduGrid ARC grid middleware.
- The Condor execution nodes (Grid nodes) are run on standard hospital PC's in a Virtual Machine image. Currently, a test bed of 20 computers, and several researcher PCs are running the Grid nodes. The images are identical and can be copied to additional PCs for expanding the cluster (a fully automatic solution exists using the standard hospital software distribution system based on Microsoft Active directory).
- The Grid server runs in a virtual machine with large disk space, for staging Grid jobs. Like the Grid nodes, the server can be moved to another location (another host PC) easily as only a single external harddisk needs to be moved from one computer to another one.

We present use case applications and performance measurements of the implemented solution. The impact of the Grid nodes on the performance of the host PC is measured by benchmarks and by startup times of popular applications. In modern dual–CPU host computers, the impact is generally not noticeable by the user. On five–year old desktop PCs on the other hand the performances degrades in an important manner particularly for application startup times.

Our Grid applications consist of parallel image processing tasks for three differing applications. A notable recent improvement for easing the creation of Grid–enabled applications is the use of the Taverna workflow engine in designing and running the tasks. This system allows for a graphical way of combining application blocks and does not require command–line based tools. Taverna's ARC plug–in enables the user to run the tasks in ARC–based Grids.

The experiences show that small Grids within medical institutions are possible and that virtualization techniques work well on new desktop computers. A Grid node running in a virtual machine on a user's desktop does barely slow the use of standard desktop applications. On the other hand, research applications can profit from the availability of more computing power allowing quicker tests of parameters and more complex solutions.

Acknowledgements

This work was partially supported by the EU 6th Framework Program in the context of the KnowARC project (IST 032691) and by the Swiss National Science Foundation (FNS) in the context of the Talisman–2 project (project 200020 118638).

References

[1] Wilhelm Burger and Mark J. Burge. *Digital Image Processing, An Algorithmic Introduction Using Java*. Springer, 2008. 4.3

[2] Adrien Depeursinge, Jimison Iavindrasana, Asmâa Hidki, Gilles Cohen, Antoine Geissbuhler, Alexandra Platon, Pierre-Alexandre Poletti, and Henning Müller. Comparative performance analysis of state–of–the–art classification algorithms applied to lung tissue categorization. *Journal of Digital Imaging*, 2009–to appear. 4.2

[3] Thomas Deselaers, Thomas M. Deserno, and Henning Müller. Automatic medical image annotation in ImageCLEF 2007: Overview, results, and discussion. *Pattern Recognition Letters*, 29(15):1988–1995, 2008. 4.1

[4] Mattias Ellert, Michael Grønager, Aleksandr Konstantinov, Balázs Kónya, J. Lindemann, I. Livenson, Jakob Langgaard Nielsen, Marko Niinimäki, Oxana Smirnova, and Anders Wäänänen. Advanced resource connector middleware for lightweight computational Grids. *Future Generation Computer Systems*, 23(2):219–240, 2007. 2.2

[5] R. Figueiredo, P. Dinda, and J. Fortes. A case for Grid computing on virtual machines. In *Proceedings International Conference on Distributed Computing Systems (ICDCS)*, pages 550–559, 2003. 1

[6] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *CCS '1998: Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, San Francisco, California, United States, 1998. 3.2

[7] Intel. Intel virtualization technology. *Intel Technology Journal*, 10(3), 2006. 2.1

[8] Hajo N. Krabbenhöft, Steffen Möller, and Daniel Bayer. Integrating ARC Grid middleware with Taverna workflows. *Bioinformatics*, 24(9):1221–1222, March 2008. 1, 4.3

[9] D. Krefting, J. Bart, K. Beronov, O. Dzhimova, Falkner J., M. Hartung, Hoheisel A., T. A. Knoch, T. Lingner, Y. Mohammed, K. Peter, E. Rahm, U. Sax, D. Sommerfeld, T. Steinke, T. Tolsdorff, M. Vossberg, F. Viezens, and A. Weisbecker. Medigrid: Towards a user friendly secured grid infrastructure. *Future Generation Computer Systems*, 25:326–336, 2009. 5

[10] M. Litzkov, M. Livny, and M. Mutka. Condor — a hunter of idle workstations. In *Proceedings of the 8th international conference on distributed computing*, pages 104–111, San Jose, California, USA, June 1988. 1

[11] A. C. Marosi, P. Kacsuk, G. Fedak, and O. Lodygensky. Using virtualmachines in desktop grid clients for application sandboxing. Technical report, CoreGrid, 2008. 1

[12] Henning Müller, Phuong Anh Do Huang, Adrien Depeursinge, Pierre Hoffmeyer, Richard Stern, Christian Lovis, and Antoine Geissbuhler. Content–based image retrieval from a database of fracture images. In Steven C. Horii and Katherine P Andriole, editors, *Medical Imaging 2007: PACS and Imaging Informatics*, volume 6516 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, page 65160H, San Diego, CA, USA, March 2007. 4.3

[13] Henning Müller, Nicolas Michoux, David Bandon, and Antoine Geissbuhler. A review of content–based image retrieval systems in medicine – clinical benefits and future directions. *International Journal of Medical Informatics*, 73(1):1–23, 2004. 1, 4.1

[14] Henning Müller, Mikko Pitkanen, Xin Zhou, Adrien Depeursinge, Jimison Iavindrasana, and Antoine Geissbuhler. Knowarc: Enabling Grid networks for the biomedical research community. In *Healthgrid 2007*, pages 261–268, Geneva, Switzerland, April 2007. 1

[15] Marko Niinimäki, Xin Zhou, Adrien Depeursinge, Antoine Geissbuhler, and Henning Müller. Building a community grid for medical image analysis inside a hospital, a case study. In Silvia D. Olabarriaga, Diane Lingrand, and Johan Montagnat, editors, *Medical imaging on grids: achievements and perspectives (Grid Workshop at MICCAI 2008)*, pages 3–12, New York, USA, September 2008. 1, 2.1, 3.2, 4.1

[16] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004. 1, 4.3

[17] Frederik Orellana, Marko Niinimäki, Xin Zhou, Peter Rosendahl, Henning Müller, and Anders Wäänänen. Image analysis on gridfactory desktop grid. In *HealthGrid 2009*, Berlin, Germany, July 2009. 1

[18] J. M. Squyres, A. Lumsdaine, and R. L. Stevenson. A toolkit for parallel image processing. In *Proceedings of the SPIE Conference on Parallel and Distributed Methods for Image processing*, pages 69–80, San Diego, CA, July 1998. 1

[19] M. Thiemard and P. Jermini. Grid@EPFL — desktop grid using condor. In *EGEE–06*, Geneva, Switzerland, September 2006. 1

[20] D. I. Wolinsky, A. Agrawal, P. O. Boykin, J. R. Davis, A. Ganguly, V. Paramygin, Y. P. Sheng, and R. J. Figuereido. On the design of virtual machine sandboxes for distributed computing in wide-area overlays of virtual workstations. In *Workshop on Large–Scale and Volatile Desktop Grids (PCGrid)*, March 2007. 1

[21] Xin Zhou, Hajo Krabbenhöft, Marko Niinimäki, Adrien Depeursinge, Steffen Möller, and Henning Müller. An easy setup for parallel medical image processing: Using Taverna and ARC. In *Proceedings of HealthGrid 2009*, Berlin, Germany, 2009. 2.1

A neuroscience Grid-enabled portal for the Portuguese Brain Imaging Network

Ilídio Oliveira[1,2], João Paulo Silva Cunha[1,2], David Pacheco[1,2], José Maria Fernandes[1,2],
Micael Pedrosa[2], Luis Alves[2] and António Sousa Pereira[1,2]

June 5, 2009

[1]Dep. of Electronics, Telecommunications and Informatics, University of Aveiro, Portugal
[2]Institute of Electronics and Telematics Engineering of Aveiro (IEETA), University of Aveiro, Portugal

Abstract

The "Brain Imaging Network Grid" (BING) is a National initiative in Portugal to set up an e-Science network for collaborative research in neuroscience. While dedicate IT infrastructure is being prepared to ensure the proper level of storage and computational services, enabling BING to interface and extend to large-scale production Grids is appealing and an opportunity to develop new collaborations. This paper describes the BING architecture and its early instantiation, including two key components: the MAGI web portal and a Grid interfacing framework designed to integrate production Grid resources within the BING project. The portal brings friendly interactions to end-users running brain imaging analysis, while the Grid enabled framework address the bridging with gLite middleware.

Contents

1 Introduction

Brain Imaging (BI) is supported by a growing R&D multidisciplinary community in Portugal, at the intersection of medicine, biology, physics, mathematics and engineering. Answering a call for proposals held by the Portuguese Ministry of Science, a consortium of universities comprising Aveiro, Coimbra, Minho and Porto was chosen to implement the National Functional Brain Imaging Network (www.brainimaging.pt). This network is the "embryo" of a collaborative cyber-infrastructure composed by a data provider (BI centre located at the University of Coimbra), two integrated data processing and storage provider nodes (at the Universities of Aveiro and Porto) and a clinical neuroscience data access client node (at the University of Minho). All four nodes will be clients of the resulting distributed cyber-infrastructure and collaborative e-Science environment, having Coimbra and Minho a clinical profile and Aveiro and Porto an engineering and physics contribution. The BI center construction in Coimbra is completed and a 3T Siemens Trio MRI machine is already in trial operation on site; the IT infrastructure setup, globally named "Brain Imaging Network Grid (BING[1])" [10], is under way, offering the first services to the Portuguese BI scientific community.

BING use cases include typical e-Science workflows such as the cooperative acquisition of data, its description and preservation, and ensuring advanced computing services to enable virtual, on-demand neuroscience laboratories. While dedicate IT infrastructure is being prepared to ensure the proper level of service, enabling BING to interface and extend to production Grids resources is appealing and a promising path to connect the other neuro-related communities. Medical image processing has been addressed in several Grid projects, to handle the demanding requirements of large images storage and communication, and to enable complex analysis workflows [1-3]. A comprehensive state of the art can be attained from [4]. These approaches provide the ability to seamless aggregate distributed computational power, extensive storage resources and high-bandwidth networking to run on-deman experiments. In addition, Grids also ensure a high security provisions, both at identity (digital certificates) and access (Virtual Organizations management) levels. In the Brain Imaging (BI) area, and more broadly in neurosciences, there are significant efforts to build e-Science infrastructures such as the Biomedical Informatics Research Network (BIRN) [5], VL-e medical applications [6], NeuroGrid [7], NeuroLOG [8] and NeuGrid [9].

In this paper, we present an architecture driven approach towards implementing the BING vision, focusing on current status and early deployments, including the initial BING portal and its underlying grid-interfacing framework, designed to integrate production Grids within the BING project. Domain users, including clinicians and researchers, are able to seamless access Grid services, like storing, sharing and running analysis algorithms on medical images, without having to worry about the underlying complexity of the infrastructure. The current system have been designed primarily to work on top of the gLite middleware [11] (although not restricted to) to harness the production Grids of EGEE [12] and EELA [13], in which our group participates.

[1] The name was defined well before the announcement of Microsoft's Bing search engine.

2 Overall BING architecture

The new BING infrastructure deploys a comprehensive set of resources, from computing infrastructure to domain specific applications (Figure 1). To this end, a previous existing computing center and a newly equipped one, and desirably existing production Grid resources from partner initiatives, should be integrated.

A major challenge in the overall architecture is to provide a long-term, high-quality data repository to serve current demanding data types, but also anticipating future value of subject cases. To enhance the resilience of the solution along time, we have maximized the use of API. For example, data storage services abstract the actual storage technology, allowing the integration of local and Grid storage as a part of the BING data storage solution. With the same principle of separation of concerns, we propose to bring Grid resources into the BING through the usage of a services API (the IEETA Grid Framework). This allows several usage scenarios: a neuroscientist may use the BING by accessing a user-friendly portal to BI applications (MAGI); multidisciplinary research teams may develop specialized modules to explore the BING data catalog in new ways; the IT/Developer may extend the Grid contribution building on the normalized Grid interfacing API (the IEETA Grid Framework - IGF). Still, there are specific requirements for software packages (visualization, image toolkits, library dependencies, etc) that will require a much profiled application environment. To handle these specificities, the concept of a "*Brain Imaging Workstation*" will be explored as an aggregate of the different tools. In this paper, we focus on presenting the BING infrastructure and the Grid interfacing strategy. A pilot client, the MAGI portal, is used to upload data sets and run simple algorithms upon the stored data, and an enabler toolkit, the IGF, provides the Grid interfacing services.

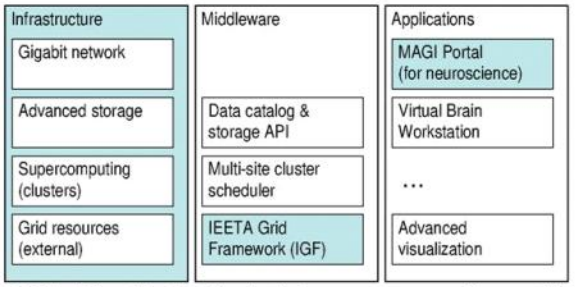


Figure 1: BING building blocks (the shaded components are discussed in this paper).

3 Hardware and network layer

Collaborative e-Science environments build on high-speed networks and advanced IT resources. Such infrastructure is being set-up for the BING and schematically presented in Figure 2. A private Gbps V-LAN connects, through a dark fiber backbone operated by the Portuguese national research and education network (NREN), the four nodes of the network where fiber routers manage data traffic. The main data storage equipment is located at Aveiro node, a NetApp FAS3140, with an overall of 72 TB

storage space. A RAID-6 storage controller is implemented, offering a 5.5 TB fiber-channel storage and 47 TB of storage in SATA technology. In the same node, a 16 blade cage is installed holding 64 CPU cores (on 8 blade computers at the moment). We plan to upgrade this equipment to hold up to 128 cores, as the demand for computer power increases. The Internet access is secured by a "*firewall*" operated at the Aveiro node.

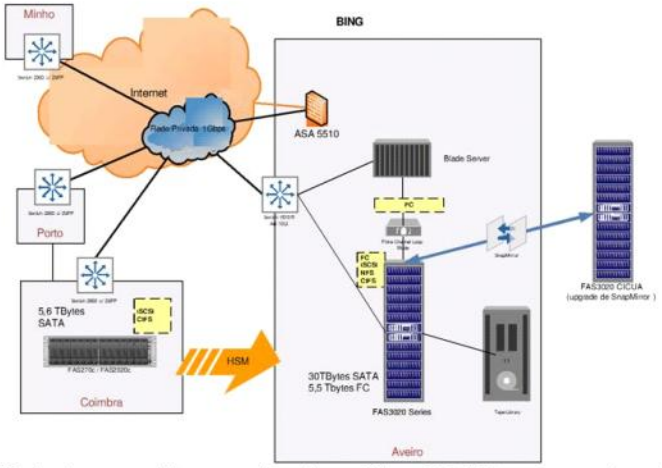


Figure 2: BING hardware architecture. A private Gbps V-LAN connects through a dark fiber backbone the 4 nodes of the network where fiber routers manage data traffic. Different IT equipment is available through Coimbra and Aveiro nodes. A pre-existing cluster operated by the University of Porto will also be connected.

This new infrastructure will function as the core fore the Portuguese brain imaging network. Additional services and specially connectivity to external partners, mapping existing collaboration links, are expected to extend this nucleus.

4 Grid-enabled medical image research portal

Researchers in the BING community, scattered along the country and abroad, will share access to a friendly interface to all the ICT layer features, including accessing datasets and run analysis on-demand. The "portal" abstraction is an intuitive interface for end-users to harness from computing centers and Grid services integrated in the network. Having this objective in mind, we designed a portal called MAGI (Medical Applications Grid Interface). The MAGI offers a rich Internet application interfaces to end-users, abstracting the details regarding the preparation and submission of jobs and input data and results management, either to a Grid or to cluster systems included in the BING. Job submissions and data transfer are triggered by the Portal usage and delegated in the Grid interfacing module, the IEETA Grid Framework (IGF). IGF provides a stable basic API to the BING, allowing future extension to other computational systems (*e.g.*: clusters, Globus middleware, etc) and storage resources (*e.g.*: DICOM servers integration, custom storage, etc.), as contextualized in Figure 1.

Based on our previous work in BI, we were able to generalize basic use cases (Figure 3) and a simple domain model (Figure 4) containing the base concepts to support medical researchers, specially with respect to biosignal processing and medical imaging modalities.

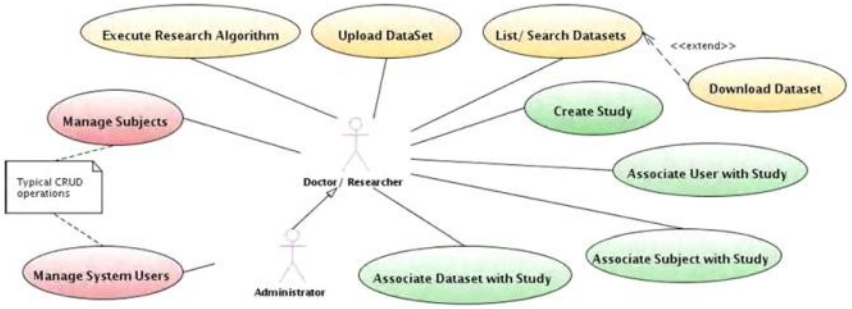


Figure 3: MAGI Portal main use-case diagram.

Our present focus is on modeling common neuroscientist research requirements and support them in the web portal. Consequently, the presented model is expressed at an high-level and while describing basic dataset information, is also able support the datasets and subjects organization within a researcher oriented perspective. The overall idea is that a researcher (**User**) belonging to a specific **Organization** may be responsible for several **Subjects** that he can gather in one or more research studies (**Study**). Most of his research activity is centered in analyzing **Datasets** of a specific **Modality**, correspondent to a **Subject**, obtained using specific **Equipment**. For instance a User can create a Study[2] on, for instance, "children with occipital epilepsy" and associate a selection of Subject and associated Datasets from the BING repository. This schema can be easily extended to accommodate a wide set of views over the data from personal perspective (e.g. "my datasets") to BING level (e.g. "epileptic patients" within BING). This is useful to support several application scenarios, covering different research fields while maintaining a common semantic.

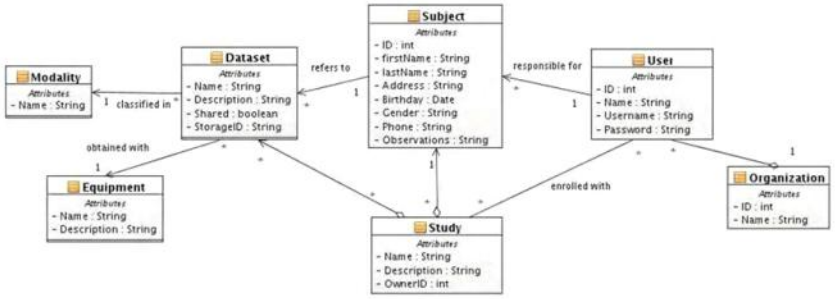


Figure 4: Domain model supporting MAGI semantics.

[2]The meaning of Study is close to the concept Investigation and not related to DICOM world, in which a study is an aggregate of series containing themselves images.

The MAGI web portal was designed to provide the researchers with the same level of productivity and easy-of-use found in other modern on-line applications, avoiding the need to handle low level idiosyncrasies. The portal already supports visual rich interactions, such as intuitive *drag-and-drop* to perform associations (e.g.: drag files into the Grid, datasets into studies) hiding the implementation details, like moving or copying files between specific file system/Grid location, while maintaining the coherence of the semantic data model. Figure 5 illustrates the uploading of data file into the Grid: (1) the user chooses to upload a file; (2) then he/she picks a local file to transfer, (3) enters descriptive file metadata and (4) "drags" the file icon into to the *Grid* container (thus triggering seamlessly underlying Grid operations). The portal can also be fed with "operators" to process the input data. These operators are code fragments, described using XML files, dynamically loaded and listed in the Portal. The user can "drag-and-drop" the input data (in resemblance to the previous example) into these operators, triggering the submission and execution of corresponding Grid jobs.

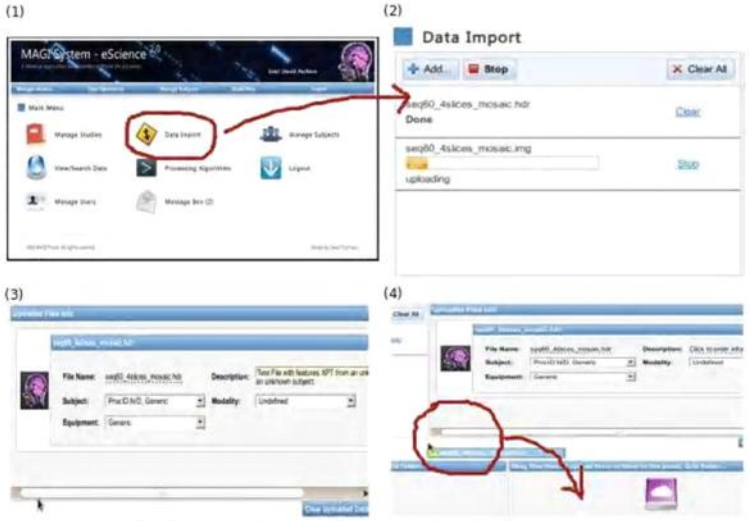


Figure 5: Web portal pages, demonstrating the data import use-case.

5 IGF: IEETA Grid Framework

IGF (IEETA Grid Framework) provides wrapping services to abstract the underlying Grid middleware and is used by the MAGI system. It provides a Java programming framework, using high-level object-oriented abstractions, and, at the present moment, focus specially on storing and sharing information on the Grid. There are other objected oriented programming environments for gLite, but often unstable or under-documented. We decided to isolate BING developments from specific programming frameworks by designing an extensible Java library, able to wrap gLite and to be used in the future with other execution environments (not necessarily Grid).

The package IGDM (IEETA Grid Data Middleware) contains services for data management like copying, retrieving or searching files in the Grid. A Computing Services package is also available to provide the basic services to submit, and monitor jobs; a Proxy Services package to start and destroy user proxies, and a Security Package to provide a higher security layer for the data and computing operations.

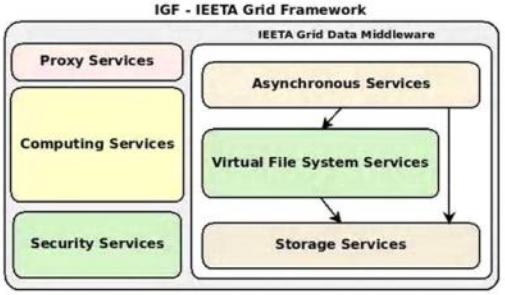


Figure 6: IEETA Grid Framework architecture

The Main IGF Components

The IGDM (IEETA Grid Data Middleware) package is the central to IGF and contains all the necessary logic to hide the complexity from the Grid Storage environment. It is subdivided into three packages:

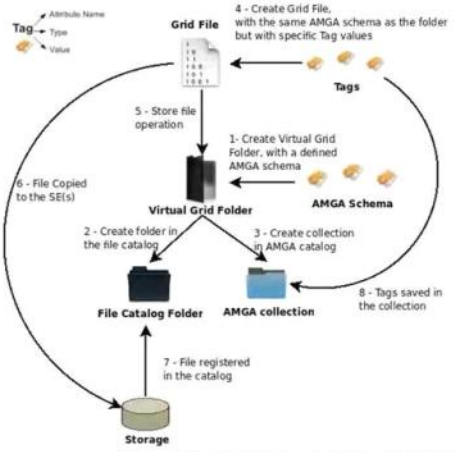


Figure 7: Relationships between Virtual Grid Folder (VGF), Grid Files and Tags, and workflow example of creating a VGF, followed by the copy of a file to that VGF.

Storage Services, Virtual File System Services and Asynchronous Services, each one of them providing a top interface for their operations. The Storage Services package is responsible for dealing with both the File Catalog and Storage Elements (SE) namely all the operations that involve file transfers to and from SEs, and direct access to the File Catalog.

The Virtual File System (VFS) provides an higher-level approach to the Grid storage based on an abstraction close to a regular file system. It allows integrated tagging of files with descriptive metadata. The Virtual Folder concept abstract the coordinated use of real folder in the File Catalog, and a Collection in the AMGA catalog [14]. As depicted in Figure 7, when a new Virtual Folder is created, the user must define a set of Tags to associate with that folder (stored as a AMGA schema), which must be supplied by files stored in that same folder. For instance, assuming we have a Virtual Folder called 'Images', and the defined schema for this folder is composed by two tags, 'pathology' and 'project', then when storing a file inside this folder, the tags 'pathology' and 'project' must be defined. This approach supports file classification for query services and supplies a folder description following the user defined semantics. For some BI dataset, besides the user defined tags, other tags can be automatically extracted as the input data already includes metadata (e.g.: DICOM files tags containing acquisition specific information). The query services rely on the AMGA catalog, to select the respective entries that match the search.

Finally the Asynchronous Services package provides support to asynchronous execution of services either in the Virtual File System or the Storage. As some of the Grid operations can suffer random delays due to various factors, an asynchronous service provider enables the construction of interactive interfaces that do not block on waiting for results from running operations. The asynchronous services associate each individual method call to individual threads. When the corresponding thread finishes, it reports the result of the operation to a controller class that observes the running/executed operations.

Using the IGF API

IGF API was designed to be "lightweight" and "developer-friendly"; it is also supported with documentation, an aspect sometimes disregarded in Grid programming environments. The framework was written in Java language, and uses some well known software design patterns. The API provides suitable error control, so that the developer can quickly debug the applications. To clarify the abstraction level provided, Figure 8 presents an example (using Java pseudo-code) for a simple operation of copying a file to a Virtual Grid Folder (assuming that the virtual folder was created previously).

```
// Creates the folder information
VirtualGridFolder vgf = new VirtualGridFolder();
vgf.setVirtualGridFolderPath("MAGIRoot/UserX");

//Grid File information
GridFile gfile = new GridFile();
gfile.setName("TestFileName.img");
// Sets the location of the temp file in the UI
gfile.setUi_location("/tmp/TestFileName.img");

AMGASchema schema = new AMGASchema();
schema.add(new Tag("Subject", "varchar", "John Doe"));
schema.add(new Tag("Modality","varchar","fMRI"));
schema.add(new Tag("Date", "timestamp", now()));
gfile.setSchema(schema);

IGDMVirtualFSServices vfs =
        IGFServicesFactory.factoryVitualFSServices
                        (IGFService.ASYNC_VIRTUALFS_SERVICES,
                                VFS_CONTEXT );
        IGDMResult res =
                vfs.copyFileToVirtualGridFolder(gfile,vgf,
                                                StoragElementsList);
```

Figure 8: Using the IGF API to upload and describe a file.

6 Conclusions and Future developments

The BING is a virtual collaboration space to support both health sciences and engineering research centers in four Portuguese Universities, to be extended, in the future, to new R&D centers, including centers of excellence across Europe, to whom cooperation links are already in place.

BING shares some common goals with large e-Science infrastructures like BIRN [5] or CaBIG [16], but instantiated at a smaller community. Unlike those initiatives, BING has no major legacy data repositories to integrate; instead, a new scientific instrument (MRI scanner) is being shared and BING is applying solutions to allow partner institutions to build a shared, "future-proof" scientific repository, with emphasis in extendibility. We see BING as a dynamic and evolving entity and for that reason the emphasis was on creating a flexible and extensible architecture, allowing to support both closed (partners-only within the private network) and public usage models (extending collaborations to Grid environments). This paper describes the earlier stages of the already running and deployed BING network. Both MAGI and IGF represent options crucial to ensure a clear and normalized separation between the presentation and service layer (MAGI) and services and storage/computational resources through a extensible service API (IGF) supporting a global virtual file system view - currently mapping Grid resources accessible through gLite.

The MAGI portal (a prototype system) is making the bridge between the users and the Grid. It provides research domain-specific semantics and already makes possible to run basic brain imaging analysis. End-users benefit from a rich web application interface paradigm to have their data and analysis tasks ran on top-resources available from production Grids.

It was an initial option to maintain a limited common research semantics at this stage as we do not intend to over-specify the BING semantics before throughout analysis involving both neuroscientist and IT is performed. As in CaBIG [15] we plan to integrate an ontology layer between IGF and MAGI in order to support more high level information that enables relating concepts and support more complex querying for MAGI. This approach, could provide the abstraction that together with IGF will allow a transparent integration of external data sources. Although BING is intended to support multimodal data repository, either image (e.g. in DICOM, Analyze or NIFTI formats) or other sources formats (e.g. EEG, video) currently only basic file management is performed. We plan to implement features extraction tools, in order to automatically obtained metadata from the files inserted in the repositories. At this time the management of such information is responsibility of the users of the information.

Currently the IGF middleware wrapper plays an essential role to address extensibility and VO reconfiguration. It provides a developer-friendly, documented API to interface the gLite middleware. It does not aim at provide specific neuroimage Grid middleware (like in NeuroLog [9]). While the support for modeling processing workflows is limited, the data operations are well covered. From the implementation perspective, IGF doesn't intend to implement from the ground all the services needed by BING but only define and provide a normalized API that provides gateway for such services. For example, the catalogue and the VFS rely on AMGA [14]. From an IGF perspective, existing solutions providing access to external sources (e.g. MDM to DICOM servers [18]) should be integrated in IGF as a different implementations within the same API.

Acknowledgements

We wish to thank to João Oliveira and Júlio Galvão from Siemens, Ricardo Martins from CICUA and to the support of the Portuguese National Brain Functional Imaging Network (RNICF), namely M. Castelo Branco, N. Sousa and A. Campilho. The present work was partly supported by FCT and FEDER, grants GRID/GRI/81833/2006 and GRID/GRI/81819/2006.

References

[1] V. Breton, K. Dean, T. Solomonides, et. al, "The Healthgrid White Paper," *Studies in Health Technology and Informatics*, vol. 112, 2005, pp. 249-321.

[2] J. Montagnat, F. Bellet, H. Benoit-Cattin, et. al, "Medical Images Simulation, Storage, and Processing on the European DataGrid Testbed," *Journal of Grid Computing*, vol. 2, Dec. 2004, pp. 387-400.

[3] I. Espert, V. Garcáa, and J. Quilis, "An OGSA Middleware for Managing Medical Images using Ontologies," *Journal of Clinical Monitoring and Computing*, vol. 19, Oct. 2005, pp. 295-305.

[4] "Medical imaging on grids: achievements and perspectives. MICCAI-Grid Workshop Proceedings," 2008 , Available from: http://www.i3s.unice.fr/~joha /MICCAI-Grid.

[5] "BIRN - Biomedical Informatics Research Network" , http://www.nbirn.net/ [accessed March 4, 2009].

[6] T. Glatard, K. Boulebiar, and S. Olabarriaga, "Workflow Integration in VL-e Medical," *Computer-Based Medical Systems, 2008. CBMS '08. 21st IEEE International Symposium on*, 2008, pp. 144-146.

[7] J. Geddes, S. Lloyd, A. Simpson, et. al, "NeuroGrid: using grid technology to advance neuroscience," *Computer-Based Medical Systems, 2005. Proceedings. 18th IEEE Symposium on*, 2005, pp. 570-572.

[8] J. Montagnat, A. Gaignard, D. Lingrand, et. al, "NeuroLOG: a community-driven middleware design," *Studies in Health Technology and Informatics, Global Healthgrid: e-Science Meets Biomedical Informatics - Proceedings of HealthGrid 2008*, 2008.

[9] "neuGRID" , http://www.neugrid.eu/pagine/home.php [accessed May 26, 2009].

[10] J.P.S. Cunha, J.M. Fernandes, I. Oliveira, et. al, "The Portuguese BING Network: Towards a Brain Imaging Grid Virtual Community," *IberGrid*, 2009.

[11] "gLite, (Lightweight Middleware for Grid Computing)" , http://glite.web.cern.ch/glite/ [accessed February 21, 2009].

[12] "EGEE: Enabling Grids for E-sciencE phase I and II, FP6 European IST project, contract number INFSO-RI-508833" , http://www.eu-egee.org/ [accessed February 26, 2009].

[13] "EELA-2 Project," Mar. 2009 , http://www.eu-eela.eu/ [accessed March 6, 2009].

[14] "AMGA: The gLite Grid Metadata Catalogue" , http://amga.web.cern.ch/amga/ [accessed February 21, 2009].

[15] "GÉANT2 website" , http://www.geant.net/ [accessed May 31, 2009].

[16] K.K. Kakazu, L.W.K. Cheung, and W. Lynne, "The Cancer Biomedical Informatics Grid (caBIG): pioneering an expansive network of information and tools for collaborative cancer research," *Hawaii Medical Journal*, vol. 63, Sep. 2004, pp. 273-5.

[17] M. Russell, P. Dziubecki, P. Grabowski, et. ali, "The Vine Toolkit: A Java Framework for Developing Grid Applications," *Parallel Processing and Applied Mathematics*, 2008, pp. 331-340.

[18] J. Montagnat, Á. Frohner, D. Jouvenot, et. al, "A Secure Grid Medical Data Manager Interfaced to the gLite Middleware," *Journal of Grid Computing*, vol. 6, Mar. 2008, pp. 45-59.

Grid-enabled sentinel network for cancer surveillance

Paul De Vlieger[1,2], Jean-Yves Boire[2], Vincent Breton[1], Yannick Legré[3], David Manset[3], Jérôme Revillard[3], David Sarramia[1] and Lydia Maigne[1]

June 12, 2009

[1]LPC Clermont-Ferrand, Blaise Pascal University, CNRS-IN2P3, 63177 Aubière Cedex, France
[2]ERIM, Faculty of Medicine, P.O. Box 38, 63001 Clermont-Ferrand Cedex, France
[3]Maat-G, 74070 Archamps, France

Abstract

Recent developments of grid services for secured distributed data management open new perspectives for disease surveillance. In this paper, we report on our initiative to develop a surveillance network for cancer in the Auvergne region. The network gathers cytopathology laboratories, structures in charge of cancer screening and institutes in charge of cancer epidemiology. Data stored in laboratories are queried through the grid for the purpose of second diagnosis and to produce statistical indicators. The paper describes the network goal and design and discusses specific issues related to patient identification and security.

Contents

1 Introduction

Cancer is becoming the first cause of mortality in developed countries. In recent years, the number of patients treated for cancer has been constantly growing while mortality has started to decrease, thanks to the progresses accomplished in the treatment of this disease and to the development of cancer screening programs [2]. These programs allow an early detection of the malignant tumours which improves significantly the medical prognosis.

In order to evaluate the public health policies, reliable statistical indicators are needed. In France, several structures have been set up to collect epidemiological data on cancer such as CRISAPs *(Centre de Regroupement Informatique et Statistique en Anatomie et cytologie Pathologiques)* which are like regional data warehouses collecting anonymous data from anatomical pathology laboratories or from the healthcare structures involved in cancer treatment. The extraction of data from laboratories encounters reluctance from the healthcare professionals because of cost and also because they lose some control over the data they have produced.

Several projects in Europe have studied or are currently exploring the grid added value for addressing cancer: the pioneer projects focused on breast cancer, particularly computer-aided diagnosis of mammograms (e-Diamond [7] and MammoGrid [5],[6] projects). These projects have produced most of the middleware bricks being used to build our cancer surveillance network: MDM (Medical Data Manager) [10] and Globus Medicus [11] are some of them; and more recently, the Pandora Gateway designed for the Health-e-Child project [4].

In this paper, we propose a very innovative approach to both cancer screening and epidemiology based on grid technology. We describe how a 'collaboration' grid federating the cytopathology laboratories together with the screening associations and the institutes in charge of cancer epidemiology would manage easily the patient data in a secure and reliable way.

2 Objectives of a grid-enabled surveillance network

Context

Most EU countries have launched a national program for breast cancer screening [2]. In France, breast cancer screening is achieved through inviting women above 50 to have mammograms every 3 years. When a woman is positively diagnosed with a risk of tumour, cancer structures are in charge of providing a second diagnosis on the mammograms and have to follow-up on the anatomical pathology (or cytopathological) data about the tumour which are stored by the laboratories. Presently, the patient data are faxed on request or carried physically by the patient to the associations where they are recorded again. This process is costly and errors prone as data have to be typed and reinterpreted twice.

The cytopathological data are also extremely important for epidemiological analysis. The INVS (Institut National de Veille Sanitaire = Sanitary Surveillance Institute), French equivalent to (E)CDC[1] for the (EU)USA, is in charge of publishing indicators about global health and particularly about cancer. To produce its indicators, INVS relies on regional cancer registries (CRISAPs) set up to collect relevant information to support statistical and epidemiological studies about cancer incidence, mortality, prevalence and screening.

However, regional cancer registries have several drawbacks:

1. In any healthcare system, physicians are responsible of patient information. The pathologists refuse to trust these systems as data is exported outside their databases, so patient identification

[1] (E)CDC: (European) Centre for Disease Prevention and Control

criterias cannot be attached to the file in order to disengage responsibility. In this way, neither disambiguation nor patient linkage is possible, so statistics are biased. Effectively, only medical information is carried out without patient identification. As a patient can undergo several biopsies in different laboratories, the information about his cancer would be present twice in the register without linkage.

2. Some pathologists refuses to export since this method requires losing control of the data produced in their own laboratories which is like the fruits of their labor.

3. The current data gathering system needs physicians to export manually data from their software, format it according to the Crisap specification, connect to the central repository and send their file. This method is costly in time without any compensation.

Now, their usage is being questioned as the global quality of these repositories decreased and less and less laboratories contribute to the registers.

Solution proposed

Our alternative is for the clients to query anatomical pathology laboratories databases directly on site. A collaborative data grid, federating the laboratories, (see **Figure 1**) would provide a secured framework enabling the screening associations to query databases and fill their local patient file. No action is required by physicians to push their data on the network. Thanks to the Grid Security Infrastructure (GSI) [8], the pathologists are able to define and modify the access rights of the users querying their data.

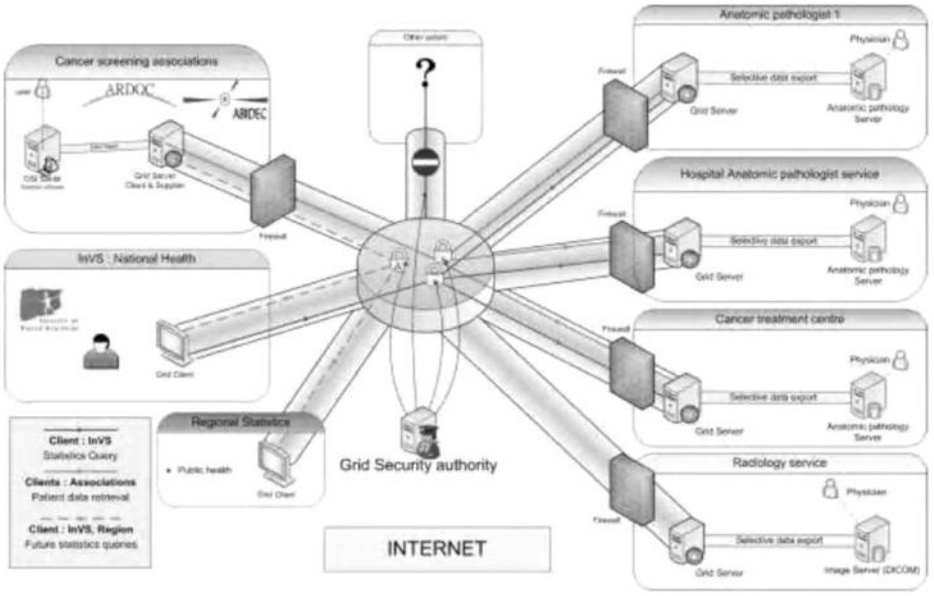


Figure 1 Cancer surveillance network using Grid technologies.

If a sentinel network is able to federate anatomical pathology databases, it can be used by the epidemiological services of the INVS and the regional epidemiological observatory to build epidemiologic studies.

3 Material and methods

List of requirements

As shown in **Figure 1**, the main medical data providers are the cytopathology laboratories. These different laboratories host different software systems and local databases for medical data management. Radiology services are additional data providers for mammograms in a step futher. The data requesters are the cancer screening associations and the different epidemiological structures in charge of producing statistics on specific cancers at regional and national level. Contrary to cancer screening associations who need to obtain the entire medical patient sheet, epidemiological structures need only anonymous medical data to produce statistics but with disambiguation to avoid double counting of cancer patients.

In a near future, the network should be able to grant access to medical images like mammograms to the cancer screening associations in order to ease the second diagnosis. The infrastructure design should offer a good flexibility to ease the entrance of new actors in the network.

The security infrastructure of the network needs to comply with French regulation on medical data transfer and exchange. The pathologists need to control the access rights to their own data. The network users must authenticate themselves using recognized accreditation tools like healthcare professionals cards. The individual certificates used have to be delivered by a certification authority (CA) recognized officially by the ministry of health and compatible with the grid infrastructure deployed.

Sentinel network infrastructure

The proposed dedicated grid architecture is built upon a central set of servers hosting security features and core grid services as illustrated in **Figure 2**:

- VOMS is an authorization manager, which implements a PKI-based authentication with certificates delivered by trusted authorities (CA) [3]. The usage of VOMS in this project is almost mandatory as VOMS is part of the gLite [13] middleware and guarantees a robust access control to the grid. Each user must own a certificate in order to log in. During the network development phase, a local certificate authority will be created, but once the network is operational, only official certificates for duly authorized healthcare professionals will be used to log in.

- The Pandora GateWay is a set of software designed as a Service Oriented Architecture (SOA). It was developed by the Maat-G society for the Health-e-Child project [4]. The Pandora Gateway is used to address medical data accessibility, exchange and processing while guaranteeing a high level of security for sensitive data. The main added value of the GateWay, compared to a classic SOA platform, is the high-level security. The GateWay Authentication Service is based on several security checkpoints required for log in. The access point is a Two Factor Authentication with user certificates and pin code followed by an authentication process using a VOMS Grid Proxy creation.

- The AMGA (ARDA Metadata Grid Application) [1] server, which provides a way to access and store metadata. Beside its high performance, the main advantages of AMGA are the full implementation of the grid security infrastructure (GSI) as well as the integration VOMS. When dealing with medical images, the use of metadata is mandatory and MDM (Medical Data Manager) allows bridging DICOM servers and the gLite middleware through AMGA [10]. AMGA is a very attractive software to fulfill the strong security requirements and access right management of medical data in a grid infrastructure.
- GridFtp server for data transfer [9].
- LFC server, for data management, included in the gLite middleware [13]

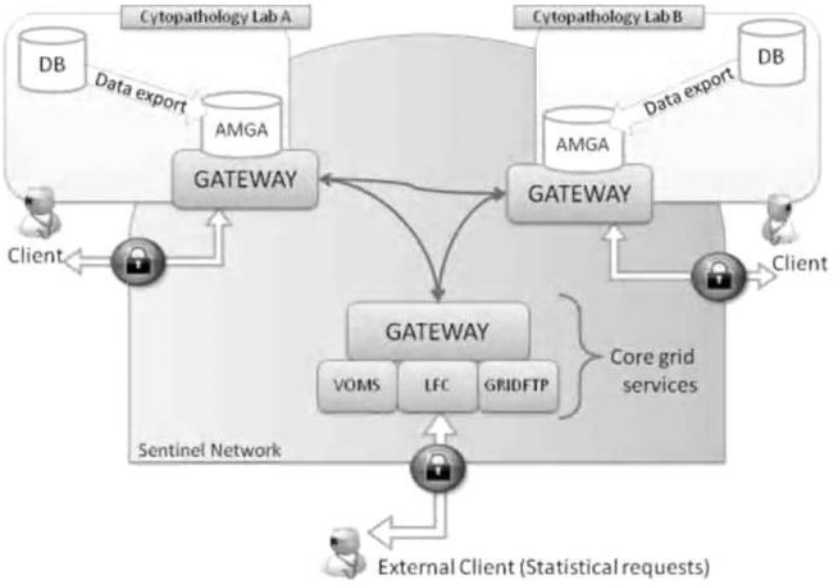


Figure 2 Service Oriented Architecture of the sentinel network.

If a client wants to use the system, he has to login through his local GateWay which authenticates himself on the sentinel network. When he launches a request, his local GateWay calls the different available AMGA servers on the network aggregate the responses and finally delivers the result to the client. In this way, no information is exchanged out of the different GateWays, guarantying a high level of security and easing data tracking. . Thanks to the Grid Security Infrastructure, authentication and credentials are available in the whole Grid, through GateWays and AMGA server.

4 Specific issues for prototype implementation

Data specification and retrieval system

For a better readability of cytopathological data, data sheets standardization is used to simplify the addition of medical records in the database without interfering with other data. Care must be taken at this step not to lose data coherence and the ownership of medical diagnosis. The customer part of the application hosts a grid server (under a firewall) to link patients' data in the network. For public health centers, a computer with an Internet access is just needed to launch epidemiologic requests.

Patient identification and data linkage

Patient identification is one of the major issues of modern healthcare systems. How can a healthcare system provide a way to identify surely his citizens while respecting their privacy?

As disambiguation and patient linkage is one of the central part of this project, patient identification is at the core of the data linkage problem. Currently, due to the lack of global identifier, each patient is linked to two different identification numbers (medical folder numbers) which are used inside each medical structure (see **Figure 3**).

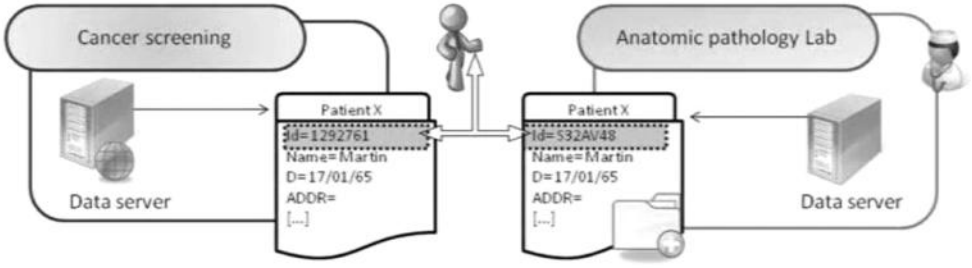


Figure 3 Patient identification problem.

Our solution requires an additional identifier for the sentinel network. This identifier consists in a random number generated (uuid type as defined in the RFC 4122 [12]) for each patient. This identifier would be created only for data linkage and would always be encrypted using different keys in each database to protect patient privacy.

When a data provider downloads some new data from his local data server to the local grid server, the Pandora Gateway is in charge of searching the patient in all the local databases respect to information on the patient. It will produce a unique identification number corresponding to the medical data if two identifiers are correlated to the same patient. With this procedure, we build a consistent network with unambiguous usage as we provide statistical requests free of doubles.

Security issues: strategy

One important step in the analysis is to clarify all security aspects in the sentinel network. The usage of SSL (Secure Socket Layer) and GSI will create a trusted network where data exchange and authorization procedures will fit the security requirements. However, and in order to protect patient privacy, the network access has to be restricted to authorized physicians or related staff only. The usage of Electronic Health Cards seems to be the best solution to settle this issue. These cards exist in France[2] and also in other European countries thanks to the EU framework[3]. Basically, these cards are smartcards containing certificates on a chip and supports strong authentication, electronic signature and data encryption. As bundled certificates are X509 formatted, they are intrinsically compatible with authentication on a gLite-powered Grid. These cards are also a response for data holders (i.e. pathologists) to accept the sharing of their data. As the sentinel network will be reachable only by physicians, the different queries will be launched under the responsibility of the physician who launches the query. The pathologists are now free to make available safely their data without any risk in case of wrong use.

Validation

Once the prototype is operational and upstream of the real exploitation phase, the next step will be validation of epidemiologic queries in order to certify the coherence and consistency of the results obtained. A comparison will then be possible with the earlier conclusions of epidemiological surveys locally obtained with manual methods by the regional public health services.

5 Conclusion

The article describes the goals and design of a surveillance network for breast cancer in the Auvergne region. The network will allow federating, in a fully secured way, cytopathology databases with cancer associations. The cancer network will be used to improve cancer screening programs and to produce reliable and theorically exhaustive cancer epidemiological indicators. Implementation of the sentinel network has started. The aim is to deploy a first prototype by the summer 2009 between one of the 2 cancer screening associations in Auvergne and a cytopathology laboratory. The beta version of the network will be operational by the end of 2009.

The integration of additional laboratories as well as the development of query interfaces for epidemiological structures will be addressed in 2010. Extension of the network to other cancer types and medical images sharing are foreseen within the framework of a new project to be funded by French Research Funding Agency.

Acknowlegements

The authors wish to acknowledge numerous discussions with P.Bouchet, A.Gaillot, L.Gerbaud, M-A.Grondin, A.Lautier, P.Lonchambon and C.Mestre.

[2] www.gip-cps.fr
[3] www.hprocard.eu

The work described in this article was partly supported by grants from the European Commission (EGEE, Embrace), the French Ministry of Research (GWENDIA) and the regional authorities (Conseil Régional d'Auvergne, Conseil Général du Puy-de-Dôme, Conseil Général de l'Allier). The Enabling Grids for E-sciencE (EGEE) project is co-funded by the European Commission under contract INFSO-RI-031688. The EMBRACE project is co-funded by the European Commission under the thematic area "Life sciences, genomics and biotechnology for health", contract number LHSG-CT-2004-512092. Auvergrid is a project funded by the Conseil Regional d'Auvergne. The GWENDIA project is supported by the French ministry of Research.

Reference

[1] B. Koblitz et al, The AMGA Metadata Service, *Journal of Grid Computing* **6** (2008), 61-76.

[2] Cancer Screening in the European Union; *Report on the implementation of the Council Recommendation on cancer screening* (2007).

[3] R. Alfieri, R. Cecchini, et al, From gridmap-file to VOMS: managing authorization in a Grid environment, *Future Generation Computer Systems* **21** (4) (2005): 549-558.

[4] The Health-e-Child project: http://www.health-e-child.org/

[5] R Warren et al, A Prototype Distributed Mammographic Database for Europe, Clinical Radiology 62.11 pp 1044-51 (Elsevier) (2007)

[6] R Warren et al, A Comparison of Some Anthropometric Parameters between an Italian and a UK Population: "proof of principle" of a European project using MammoGrid, Clinical Radiology Vol 62.11 pp 1052-60 (Elsevier) (2007)

[7] M. Brady et al, eDiamond: a grid-enabled federated database of annotated mammograms, *Grid Computing: Making the Global Infrastructure a Reality F Berman*, G Fox and T Hey (eds), Wiley, (2003).

[8] V. Welch et al, Security for Grid services, *High Performance Distributed Computing,* 12th IEEE International Symposium on (2003), 48-57.

[9] V. Allcock et al, *The Globus Striped gridFTP Framework and Server,* ACM/IEEE conference on Supercomputing (2005), 54-64.

[10] J. Montagnat et al, Bridging Clinical information systems and grid middleware: a Medical Data Manager, *Studies in health technologies and informatics* **120** (2006), 14-24.

[11] SG. Erberich et al, Globus MEDICUS - federation of DICOM medical imaging devices into healthcare Grids. *Studies in health technologies and informatics* ***126*** (2007), 269-78.

[12] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. IETF RFC 4122 (2005), http://www.ietf.org/rfc/rfc4122.txt

[13] gLite Middleware : http://glite.web.cern.ch/glite