Improving the execution performance of FreeSurfer

A new scheduled pipeline scheme for optimizing the use of CPU and GPU resources

J. Delgado[1,2], J.C. Moure[2], Y. Vives-Gilabert[3,4], M. Delfino[3,5], A. Espinosa[2], B. Gómez-Ansón[6]

[1] *Asociación para la Innovación en Análisis, Gestión y Procesamiento de Datos Científicos y Tecnológicos (INNDACYT), E-08980 Sant Feliu de Llobregat (Barcelona), Spain.*

[2] *Computer Architecture and Operating Systems (CAOS), Universitat Autònoma de Barcelona, Edifici Q, E-08193 Bellaterra (Barcelona), Spain.*

[3] *Port d'Informació Científica (PIC), Universitat Autònoma de Barcelona, Edifici D, E-08193, Bellaterra (Barcelona), Spain.*

[4] *Institut de Física d'Altes Energies (IFAE), Universitat Autònoma de Barcelona, Edifici Cn, E-08193 Bellaterra (Barcelona), Spain.*

[5] *Department of Physics, Universitat Autònoma de Barcelona, E-08193 Bellaterra (Barcelona), Spain.*

[6] *Hospital de la Santa Creu i Sant Pau Barcelona, Sant Antoni Maria Claret 167, E-08025, Barcelona, Spain.*

Abstract

A scheme to significantly speed up the processing of MRI with FreeSurfer (FS) is presented. The scheme is aimed at maximizing the productivity (number of subjects processed per unit time) for the use case of research projects with datasets involving many acquisitions. The scheme combines the already existing GPU-accelerated version of the FS workflow with a task-level parallel scheme supervised by a resource scheduler. This allows for an optimum utilization of the computational power of a given hardware platform while avoiding problems with shortages of platform resources.

The scheme can be executed on a wide variety of platforms, as its implementation only involves the script that orchestrates the execution of the workflow components and the FS code itself requires no modifications. The scheme has been implemented and tested on a commodity platform within the reach of most research groups (a personal computer with four cores and an NVIDIA GeForce 480 GTX graphics card). Using the scheduled task-level parallel scheme, a productivity above 0.6 subjects per hour is achieved on the test platform, corresponding to a speedup of over six times compared to the default CPU-only serial FS workflow.

Keywords *FreeSurfer, MRI, Medical Imaging, GPU, CUDA, Resource Scheduler.*

1 Introduction

FreeSurfer (FS) is a set of software tools for analysis and visualization of structural and functional brain imaging data developed at the Athinoula A. Martinos Center for Biomedical Imaging. FS is documented and freely available to download online [1]. The FS workflow for cortical reconstruction and volumetric segmentation of Magnetic Resonance Images (MRI) is a very commonly used tool in medical research of the brain. We will refer to this workflow as FS-MRI. It comprises two streams: the surface-based and the volume-based stream. The surface-based stream calculates, for each point on the cortex, the cortical thickness, volume, surface area, and curvature [2, 3]. The volume-based stream quantifies the volumes of the subcortical structures [4, 5].

FS-MRI is structured as a pipelined stream of computation processes, whose execution is controlled by the *recon-all* shell script. The execution time of the workflow for a single MRI acquisition ranges from 10 to 30 hours, depending on the hardware platform. Shortening this execution time is important for research projects which involve many MRI acquisitions or many subjects, as well as for potential future clinical applications.

One method to accelerate the FS code is to execute some of the components of the pipeline on Graphical Processing Units (GPUs) which are present on many computers. This has been implemented by the authors of FS [6] starting with the version released in March 2010, using the Compute Unified Device Architecture (CUDA) language from NVIDIA Corporation [7,10]. The execution on GPU can be activated by an input parameter, providing a hybrid execution using both the normal Central Processing Unit (CPU) of the computer and the GPU. This works well when executing a single copy of FS-MRI, but can fail when executing multiple instances due to lack of memory in the GPU.

In this paper, we describe additional ways to accelerate FS-MRI through two complementary approaches. The first introduces intra-workflow parallelismto accelerate the execution of each single FS-MRI instance. The second shortens the total execution time for processing a group of acquisitions, by exploting inter-workflow parallelism. In practice, a scheduler is needed to manage the concurrent execution of several FS-MRI instances, and to limit the number of concurrent GPU tasks. The scheduler will also optimize the utilization of the available computing resources, both CPUs a GPUs.The scheduler presented is a general method that can be applied to other workflow-based applications.

There exist many open-source job schedulers addressed to computer clusters and grids, like SLURM [11], PSCHED [12] or PBS [13]. They have recently included support for simple management and scheduling of GPU resources, by separatelly allocating each job into individual GPU cards on a computational node. However, they do not provide simple ways to allow several jobs or tasks sharing a single GPU, which is required to get higher performance when executing FS workflow as we will show later.

There are studies on GPU scheduling at the *kernel* and *warp* level [8,9]. The first approach improves the scheduling of kernels and warps, by allowing interrupting or prioritizing different kernels. The second schedules *CUDA threads* to achieve a more efficient (Single Instruction Multiple Data) SIMD branch execution.
We have designed and implemented our own task scheduler addressed to a single computational node, with a single address space and single file system. It manages concurrent tasks from several FS-MRI instances, controlling task dependencies and assuring resource usage limits, specially regarding GPUs. The scheduler built-in policy has provide us the means to prove that a careful management of the GPU and CPU cores yields significant performance improvements. Additionally, by modifying the internal description tables, the presented scheduler can be applied to other workflow-based applications. The implementation is publicly available and can be used in any server platform with multi-core CPUs and multiple GPUs.

Both acceleration methods have been implemented and tested with a commodity desktop computer equipped with a single GPU. Results indicate a very substantial improvement in execution time. The implementation can be directly used in more powerful platforms, such as servers with multi-core CPUs and multiple GPUs.

This paper is organized in six sections. The next section presents a brief analysis of the FS-MRI workflow scheme and the opportunities for introducing parallelism in order to shorten execution times. Section 3 explains the two complementary approaches used to accelerate FS-MRI execution. Section 4 briefly describes the test setup used to validate the implementation. Section 5 presents the results of test runs. Finally, we summarize and conclude our results, and present some options for future work.

2. Analysis of the FS-MRI workflow scheme

The FS-MRI workflow scheme is composed of 31 stages organized serially as a strictly sequential pipeline (Fig. 1a). A dependency analysis of the stages has been performed searching for opportunities to introduce parallelism. The conclusions are as follows:

Within a single instance of the workflow, many of the consecutive stages in the FS-MRI present input/output data dependencies. However, the stages processing each brain hemisphere (left and right) are independent. Therefore, it is possible to modify the pipeline in order to execute simultaneously the stages for the two brain hemispheres (see LH and RH in Fig. 1b).

There are no dependencies between any pipeline components which execute in different workflow instances. This is an example of Data Parallelism. Therefore, it is possible to execute K instances of the workflow simultaneously, as long as the underlying operating system and hardware can cope with the demand for resources. This is equally applicable to the original non-parallel FS-MRI (Fig. 1a) as well as to the modified (left-right) parallel scheme described above (Fig. 1b). The result of combining both intra-workflow (left-right) parallelism with inter-workflow parallelism (executing K instances concurrently) is shown in Fig. 1c.

It should be noticed that both of the modifications described above are, in principle, compatible with the introduction of GPU acceleration. When more than one module of the pipeline is executed simultaneously, the operating system will manage the execution by scheduling the underlying hardware, either directly to the CPU cores (Fig 1d and 1e) or to the GPU units (Fig. 1d and 1f). In practice, however, the execution of many simultaneous instances can overwhelm the computing system resources. In particular, the GPU lacks sophisticated management of its resources and the available memory of the GPU can be a major limitation on the total number of instances which can be executed simultaneously. Whereas this may be a minor problem for intra-workflow (left-right) parallelism, it is a severe problem for inter-workflow parallelism with large values of K. This problem is solved by the introduction of a scheduler to manage the concurrent execution which optimizes access to the CPU and GPU resources.

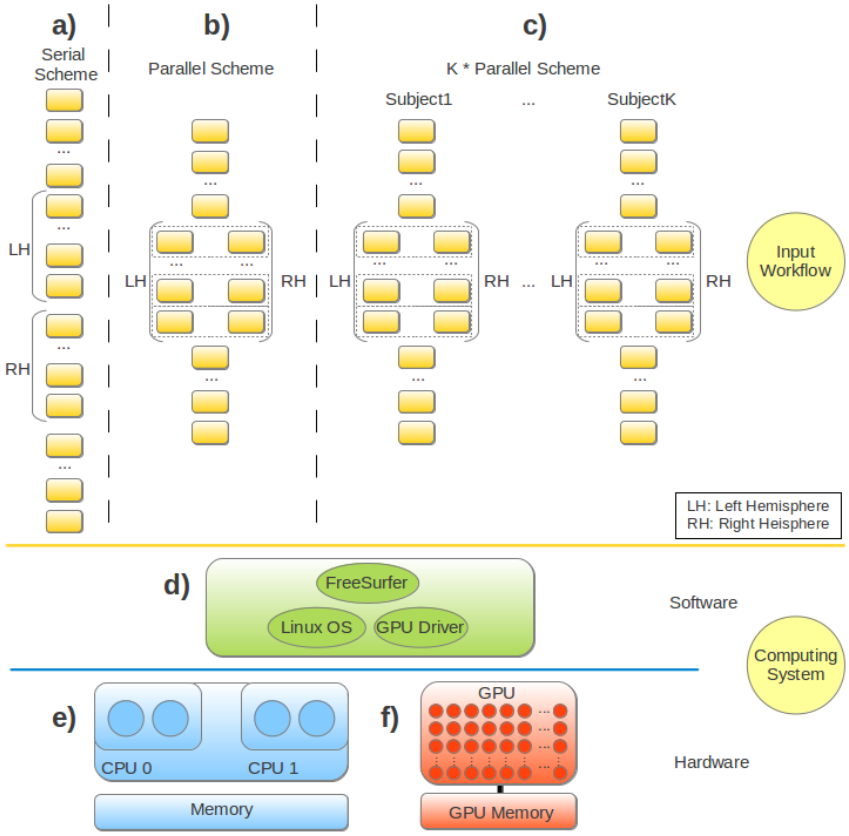

Fig. 1 - Input workflows and computing system configuration for executing FS.

3. FS-MRI pipeline parallelization and scheduling

In this section we describe the implementation of the two modifications of the pipeline scheme that introduce intra-workflow and inter-workflow parallelism, and a workflow execution scheduler which avoids overflow of the computing system resources, particularly the GPU memory.

3.1 Intra-workflow parallelization

We modified the FS-MRI *recon-all* script in order to execute in parallel two instances of the subsets of the pipeline which process the left and right brain hemispheres, following the scheme in Fig. 1b. We forced a synchronization barrier at the end of these two execution streams in order to guarantee the data dependencies for the next stages of the workflow. This modification can be used for executing FS-MRI in CPU-only or hybrid CPU+GPU platforms.

Fig. 2 shows a Gantt diagram of the execution of the first nine blocks of a single instance of a FS-MRI workflow with the modification described above, executing on a hybrid platform with a four CPU cores and a single GPU with 15 stream multi-processors (smp). Blocks b1, b3, b5, b6 and b8 execute on the CPU cores, whereas blocks b2, b4, b7 and b9 execute on the stream multi-processors of the GPU, supervised by a control task executing on one of the CPU cores. The inter-workflow parallelism results in two copies of blocks b6, b7, b8 and b9 executing simultaneously for the left and right brain hemispheres (labelled lh and rh). This places additional demands on the hardware, occupying a second CPU core and doubling the computation requirements on the GPU smps.

Total execution time is improved in comparison to the default, strictly sequential, CPU-only case by executing some blocks faster in the GPU and by executing the left and right hemisphere blocks concurrently.

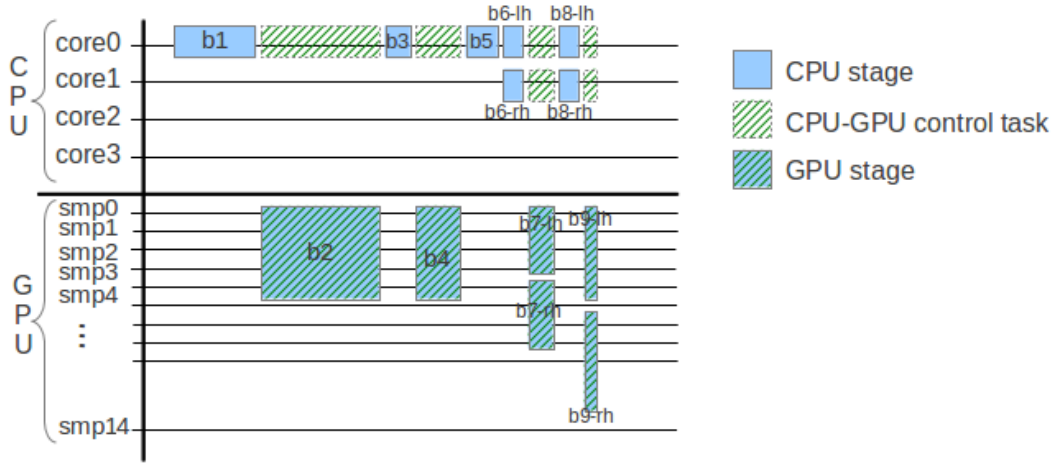

Fig 2 - Approximate resource usage as a function of time in the intra-workflow parallel scheme.

3.2 Inter-workflow parallelization

Implementing intra-workflow parallelism, as explained in the previous section, will double CPU and GPU resource usage during part of the workflow execution. There are, however, additional resources available to run tasks, and thus we propose the strategy of executing multiple FS-MRI instances in order to use more of the available resources.

The introduction of inter-workflow parallelism is of interest given that a realistic use case for running FS-MRI is that of processing multiple subjects belonging to a given research study. A number K of identical instances of the workflow are executed, each using different input data, as shown in Fig 1c. In absence of computing system limitations, there would be perfect scalability and the total execution time for the K subjects would be shortened by approximately a factor of 1/K compared to single workflow execution.

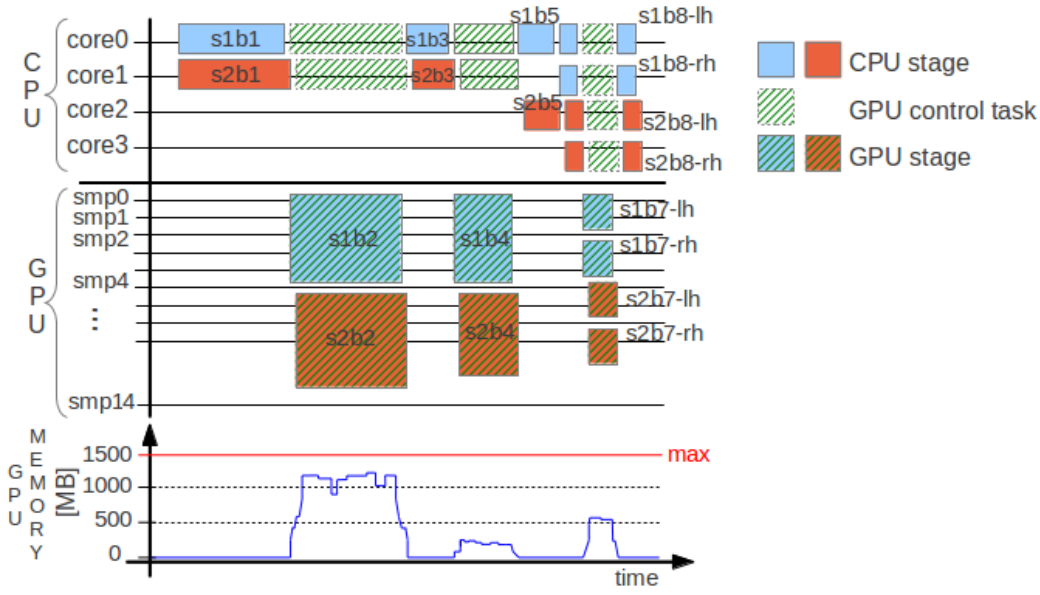

Fig. 3 Approximate resource usage in the simultaneous executions of K=2 FS-MRI instances.

A Gantt diagram for the case K = 2 is shown. It is based on results that will be presented in section 5.2. The notation is the same as in Fig. 2, but using the s1 and s2 to label the two workflow instances. The operating system schedules the execution of the available tasks into the CPU cores. Some of the executing tasks generate requests to the GPU driver for executing both computation and data movement tasks on the GPU. The GPU driver schedules the execution of GPU tasks from different applications in a way that several tasks (computation kernels or data movement) can be performed concurrently.

The execution time for K = 2 workflow instances is almost the same as for K = 1, which provides a near perfect throughput speedup. The four CPU cores in the system assure that during execution with K = 2 all CPU tasks will be assigned a separate CPU core. However, tasks b2 of both workflow instances have to be executed concurrently on the same GPU, and the execution time barely grows compared to a single execution. This fact indicates that the GPU internal resources are underused by a single workflow instance.

The bottom part of Fig. 3 illustrates the amount of GPU memory in use during the execution. For K > 2, the GPU hardware memory capacity will be overwhelmed when all tasks b2 are concurrently executed. The Linux operating system and the GPU driver are unable to detect and handle this condition, and the execution of one or several of the instances fails.

3.3 Parallelism with resource scheduling

In order to overcome the problem described in the previous section, we developed a centralized scheduler for multi-subject execution, which is able to manage shortages of resources such as GPU memory.

The scheduler centralizes the management of the input subjects and the processing stages defined in the workflow scheme using a greedy policy that always assigns available resources when there are ready tasks, trying to yield the maximum possible performance.

The scheduler scheme is presented in Fig. 4. In order to minimize management overheads, the 31 steps of the FreeSurfer workflow were grouped into 13 blocks of steps according to two criteria: dependence between steps and CPU/GPU implementation. These blocks represent the minimum independent execution unit and will be labeled b1 through b13. Blocks will execute in either CPU or GPU resources.

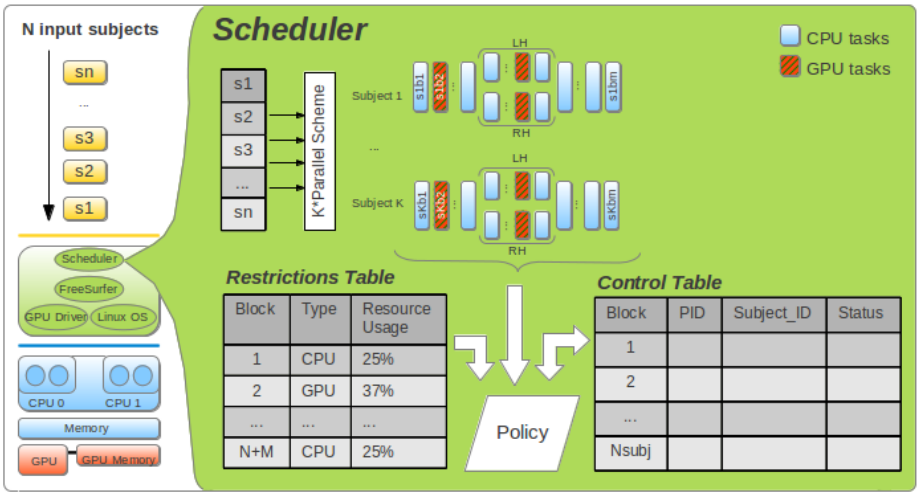

Fig. 4 - Multi-workflow scheduler scheme and components.

The scheduler receives three input parameters: a list with all the subjects to be processed (N subjects), the number of CPU tasks that can be executed simultaneously (K), and a *restriction table* which stores information about the limiting resource (CPU or GPU) and the estimated percentage of use of this resource per instance.

The scheduler tracks resource usage by counting accumulated usage percentage, both on CPU and GPU, ensuring that it never reaches above 100%. The usage percentage for CPU blocks is set to 100%/K, meaning that the scheduler allows the simultaneous execution of K CPU blocks. The usage percentage of each GPU blocks is set to the usage percentage of GPU memory used by the block, obtained empirically from test runs.

The scheduler uses a *control table* to store information about the execution status of the different blocks. A block can be in one of these four states:

"not started"	Initial status for a block.
"running"	Set when a block can be allowed to execute.
"waiting"	Set when a block requires more resources than those currently available.
"finished"	Set when a block has finished execution.

A block is set to *waiting* state when it requires more resources than those available. The list of all waiting blocks is periodically checked by the scheduler, which consults the *restriction table* and the resources available at that moment and selects blocks which can be released to the *running* state. In this way, the scheduler dynamically maximizes the total use of resources.

4. Development and testing setup

The setup that was used to develop and test the implementations of the parallel schemes is summarized in this section.

We used a desktop computer with four cores (two dual-core Intel Core 2 Extreme at 3.67 GHz processors), 12MB of last-level cache memory, 8 GB of RAM memory and a NVIDIA GeForce 480 GTX GPU (15 Stream Multi-Processors, 1401 MHz, 1536 MB GDDR5 memory, CUDA 2.0).

The software used in this study was:

- CentOS 5.6 (64 bits) + gcc (4.1.2) and g++ (4.1.2) compilers.
- NVIDIA drivers 4.0 for Linux with CUDA support.
- CUDA Toolkit 3.2 and CUDA Software Development Kit for Linux.
- FreeSurfer 5.0, compiled to CUDA 2.0.

The tests were performed with a dataset of 100 T1 MPRAGE MRIs acquired with a 3-Tesla Philips Achieva MR scanner at Hospital de la Santa Creu i Sant Pau in Barcelona. The MRIs belonged to a Parkinson's Disease (PD) study divided in 19 healthy controls, 37 Cognitive Intact PD, 24 Mild Cognitive Impairment PD, and 20 PD with Dementia.

5. Results

In this section we present the results of a number of test runs with the different scenarios presented in section 3 and with various values of simultaneous executions (K).
The use of the scheduler solves the problem of GPU memory capacity limitations, allowing executions involving more than two subjects simultaneously (K>2) and a full exploitation of intra- and inter-workflow parallelisms.

5.1 Behaviour of the resource scheduler

Fig. 5 shows a Gantt trace of the initial steps in the execution of FS-MRI with resource scheduling, as described in section 3.3, with four simultaneous subjects (K=4). The notation is an abbreviated version of that of figures 2 and 3, where the GPU control tasks and the details of cores and stream multi-processors have been omitted.

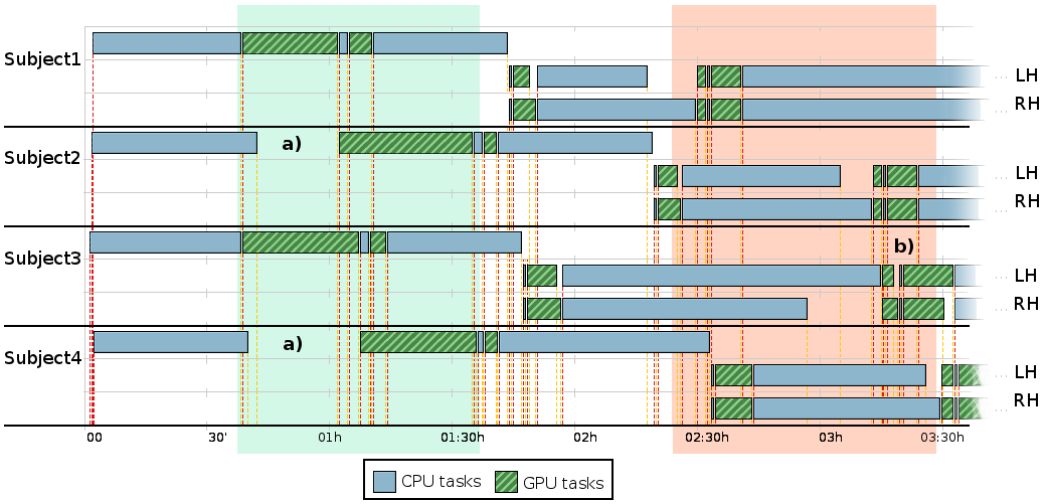

Fig. 5 Gantt trace of the execution of four simultaneous instances of the FS-MRI workflow

Initially, the four subjects start executing in parallel, each on one of the four CPU cores. The second processing block is a GPU task that requires a large amount of memory. The resource scheduler allows only two GPU tasks to run simultaneously in order to handle the GPU memory restrictions defined in the *restrictions table*. The other two GPU tasks are in *waiting* state until enough resources are available (Fig. 5 label a). For example, when the first GPU task corresponding to subject 1 ends, the GPU task of subject 2 is started by the resource scheduler.

The scheduler can also manage more complex situations with multiple simultaneous block executions (Fig. 5 label b). In these cases some of the blocks consume a smaller amount of resources and are allowed to start execution as soon as possible.

5.2 Speedup of the execution of multiple subjects

We ran several experiments with different number of subjects (N), and different number of simultaneous executions (K). In order to account for variability of the results deriving from the order in which the subjects are submitted for processing, we performed experiments with several combinations of the N subjects, and averaged the total execution times. We then calculate the average productivity, measured in subjects per hour, and the relative productivity, or speedup, using the K=1 CPU-only serial version of FS as a normalization.

Table 1 summarizes the data obtained in the tests. The tested scenarios are serial executions (with or without GPUs), the introduction of intra-workflow parallelization of the Left and Right hemispheres (section 3.1), the introduction of inter-workflow parallelization (section 3.2, limited to K=2 by the GPU memory of our test setup), and the introduction of both intra- and inter-workflow parallelism using the resource scheduler (section 3.3) to overcome GPU memory limitations for K>2.

Scenario	N	K	Time to completion experiment (seconds)	Time per subject (hours)	Productivity	Prod Norm
Serial CPU	6	1	220866	10,23	0,098	1
Inter parallel CPU-only	6	2	140685	6,51	0,154	1,57
Inter parallel CPU-only	6	3	105314	4,88	0,205	2,10
Inter parallel CPU-only	6	4	84255	3,90	0,256	2,62
Inter parallel CPU-only	6	6	85431	3,96	0,253	2,59
Inter parallel CPU-only	8	8	115428	4,01	0,250	2,55
Inter parallel CPU-only	10	10	134342	3,73	0,268	2,74
Intra parallel CPU-only	6	1	196272	9,09	0,110	1,13
Intra+Inter parallel CPU-only	2	2	37113	5,15	0,194	1,98
Serial CPU+GPU	4	1	110596	7,68	0,130	1,33
Inter parallel CPU+GPU	6	2	51462	2,38	0,420	4,29
Intra parallel CPU+GPU	4	1	65128	4,52	0,221	2,26
Intra+Inter parallel CPU+GPU	2	2	34308	2,38	0,420	4,29
Resource scheduling	6	3	38984	1,80	0,554	5,67
Resource scheduling	6	4	36035	1,67	0,599	6,13
Resource scheduling	6	6	33050	1,53	0,654	6,68
Resource scheduling	8	8	43361	1,51	0,664	6,79
Resource scheduling	10	10	57968	1,61	0,621	6,35

Table 1. Results of FS tests under different execution scenarios.

Fig. 6 shows the evolution of the performance speedup as a function of K. In the CPU-only scenario, using the Linux Operating System as the default scheduler, productivity flattens for $K \geq 4$. This result indicates that the four CPU computation cores become a performance bottleneck We have measured memory and I/O requirements and they do not represent a problem. The performance variability for K>4 is due to the differences in the input subjects, the irregular contention between concurrent tasks sharing memory and computation resources, and the uneven amount of available work at the final phase of the execution..

The CPU+GPU scenario coupled with our scheduling policy can effectively exploit more inter-workflow parallelism (up to K=6) to achieve increasing performance. Given that the computation workload is divided into CPU and GPU tasks, an in-depth analysis is needed to identify the performance bottleneck in the system. Different hardware resources will determine a different value of K where performance increase flattens. In this example, the decrease of productivity for K=10 is just an effect of the variability described previously.

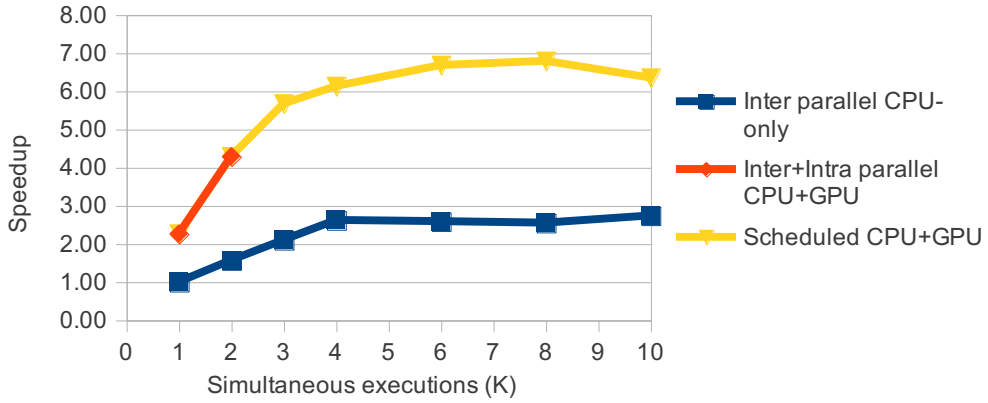

Fig. 6 - Speedup for serial, parallelised and scheduled scenarios.

Scheme	Productivity CPU-only	Productivity CPU+GPU	CPU-only speedup	CPU+GPU speedup
Serial	0.098	0.130	1	1.33
Intra parallel.	0.110	0.221	113	2.26
Inter+Intra parallel. (K=2)	0.194	0.420	1.98	4.29
Resource scheduling (K=8)	-	0.664	-	6.79

Table 2. Maximum productivities (subjects/hour) and speedup with different scenarios.

Table 2 summarizes the best results obtained for each scheme. Using our relatively modest test platform and with enough MRIs to be processed, the best scenario achieves a productivity of 0.664 subjects per hour. The GPU improves performance by 1.33 on the serial scenario, but provides above 2x when using intra-and inter-workflow parallelism. The scheduler further improves the overall speedup from 4.29 to 6.79.

Finally, we performed an execution of a realistic case of a research project with an input subject list of N=100. The scheduled execution was performed using GPU acceleration with intra-workflow parallelism using the scheduler to manage eight simultaneous workflow instances (K=8). The mean average execution with K=1 was 16282 seconds. The execution of the 100 subjects using the scheduler and K=8 gave an average execution time per subject of 6291 seconds, an improvement of a factor of 2.59x, in reasonable agreement with the expectation of 3.00x from the shorter test runs (6.79/2.26, ratio of fourth and second row of Table 2).

7. Conclusions and future work

In this paper we present a new scheme to accelerate the execution of the FS-MRI workflow. The solution is based on using intra- and inter-workflow parallelism with resource scheduling to fully exploit the available hardware. This strategy provides an improvement of the throughput of MRI processing of more than six times compared to the standard serial CPU scheme, and more than 2.5 times compared to the standard serial GPU accelerated scheme. Performance improvement will depend on the hardware platform. The setup used to obtain the results, however, is an inexpensive commodity platform within reach of most research groups.

We have identified a problem of GPU memory overflow when executing more than two simultaneous workflows. We have designed and implemented a scheduler that controls this problem, which requires setting a restrictions table with the memory usage per task obtained empirically. The scheduler uses a greedy policy to exploit the available application parallelism in order to maintain the CPU and GPU computation resources busy.

The proposed scheduler is general enough to be used in other workflow-like applications. One of the main advantages is that it works with high-level task entities, scheduling the execution of the workflow stages, avoiding the need to modify the application source code. To adapt the scheduler to a specific workflow we only need to consider the structure of the workflow, the dependencies between different stages and the resources needed by each stage.

Fig. 6 summarizes the speedups obtained with the CPU execution, the CPU+GPU execution, and the CPU+GPU scheduled execution. The highlights of the results can be seen in table 2. There is a 2.7x improvement between CPU execution and CPU+GPU scheduled execution when using intra-workflow and inter-workflow parallelism. The major improvement of productivity was obtained executing between six and eight FS-MRI workflow instances simultaneously using the scheduler. The tests indicate that running more than eight FS-MRI workflow instances provides no benefit in the particular hardware used.

A separate test run of a realistic research project scenario with 100 subjects compares well with the results presented in table 2.

A future line of work line would be to explore the use of other CPU+GPU platforms (with different architectures, processors with hyperthreading, GPUs with more internal memory, multiple GPUs, etc.) to evaluate the performance of our proposals. A different proportion of CPU and GPU computation power may promote using load balancing strategies to select either the CPU or the GPU version of given task. Also using hyperthreading can be useful for some combinations of tasks, and can be detrimental in other cases. We want to explore how to use the restriction table to get the maximum performance of each platform for our FS-MRI application.

As a second future line we will implement the scheduling policy described in this paper into the SLURM open source scheduler [11]. This supposes adding a mechanism to allow multiple jobs or tasks sharing the same GPU. This will allow extending our scheduling policies to distributed-memory clusters or grid environments, where provisioning resources is mandatory.

A final research line would be the automatic tuning of the scheduler operation. This involves measuring resource usage, detecting execution faults and modifying the scheduler's restriction table at runtime. Such a scheme would simplify the deployment of our proposals for other application workflows.

In the era of multi-core computers with powerful graphics hardware, the available parallelism has to be identified and exploited to make an efficient usage of the computer. This will save time as well as energy and lead to improved research.

Information Sharing Statement

The source code of the scheduler is available at http://sourceforge.net/projects/freesurferscheduler/ and it is provided without maintenance guarantee.

The dataset used for the test runs presented in this article is property of the Institut de Recerca de l'Hospital de la Santa Creu i Sant Pau in Barcelona and is not publicly available.

Acknowledgements

This research has been supported by INNDACYT Association, partially founded by the Talent Empresa program of the AGAUR Agency, Knowledge and Economy Department of Generalitat de Catalunya and also MICINN-Spain under contract TIN2011-28689-C02-01.

Our work was possible thanks to the cooperation of INNDACYT Association with Port d'Informació Científica (PIC), and Hospital de la Santa Creu i Sant Pau (Barcelona). The Port d'Informació Científica (PIC) is maintained through a collaboration agreement between the Generalitat de Catalunya, CIEMAT, IFAE and the Universitat Autònoma de Barcelona.

We thank specially Richard Edgar, Bruce Fischl et al from Athinoula A. Martinos Center for Biomedical Imaging (Harvard-MIT) for the offered support.

References

1. http://surfer.nmr.mgh.harvard.edu/fswiki.

2. Dale A.M., Fischl B., Sereno M.I., Cortical Surface-Based Analysis I: Segmentation and Surface Reconstruction, Neuroimage 9(2), 179-194 (1999).

3. Fischl B., Sereno M.I., Dale A.M., Cortical Surface-Based Analysis II: Ination, Flattering and a Surface-Based Coordinate System, Neuroimage 9(2), 175-207 (1999).

4. Fischl, B., D.H. Salat, E. Busa, M. Albert, M. Dieterich, C. Haselgrove, A. van der Kouwe, R. Killiany, D. Kennedy, S. Klaveness, A. Montillo, N. Makris, B. Rosen, and A.M. Dale, Whole Brain Segmentation: Automated Labeling of Neuroanatomical Structures in the Human Brain, Neuron, 33:341-355(2002).

5. Fischl, B., A. van der Kouwe, C. Destrieux, E. Halgren, F. Segonne, D. Salat, E. Busa, L. Seidman, J. Goldstein, D. Kennedy, V. Caviness, N. Makris, B. Rosen, and A.M. Dale, Automatically Parcellating the Human Cerebral Cortex, Cerebral Cortex, 14:11-22 (2004).

6. Richard Edgar, Acceleration of the Freesurfer Suite for Neuroimaging Analysis, GPU Technology Conference, (2011).

7. NVIDIAs Next Generation CUDA Compute Architecture: Fermi. White Paper (Oct 2009) http://www.nvidia.com/content/PDF/fermi white papers/NVIDIA Fermi Compute Architecture Whitepaper.pdf.

8. Antonio Ruiz et al., "The GPU on biomedical image processing for color and phenotype analysis", Proceedings of the 7th IEEE International Conference on Bionformatics and Bioengineering, 1124-1128 (2007).

9. Daren Lee, Ivo Dinov, Bin Dong, Boris Gutman, Igor Yanovsky, Arthur W. Toga, CUDA optimization strategies for compute and memory bound neuroimaging algorithms, Preprint submitted to Computer methods and programs in biomedicine, (2010).

10. NVIDIA CUDA Programming Guide - pp. 86, 136-139 - http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_ Programming Guide.pdf

11. Yoo, A., Jette, M. & Grondona, M. (2003). Simple Linux Utility for Resource Management. Job Scheduling Strategies for Parallel Processing, volume 2862 of Lecture Notes in Computer Science, (pp. 44–60). Springer-Verlag.

12. PSCHED: An API for Parallel Job/Resource Management, The PSCHED Working Group, November (1996).

13. PBS: Portable Batch System, External Reference Specification, Bayucan, A., Henderson, R. L., Lesiak, C., Mann, B., Proett, T., Tweten, D., (1999). MRJ Technology Solutions, November.