A Parallel Deconvolution Algorithm in Perfusion Imaging

Fan Zhu
Data-Intensive Research Group
School of Informatics
University of Edinburgh
Edinburgh, UK
F.Zhu@ed.ac.uk

David Rodriguez Gonzalez[1,2], Trevor Carpenter[1], Malcolm Atkinson[2], Joanna Wardlaw[1]
[1]*SFC Brain Imaging Research Centre, Division of Clinical Neuroscience*
[2]*Data-Intensive Research Group, School of Informatics*
University of Edinburgh
Edinburgh, UK
David.Rodriguez@ed.ac.uk, Trevor.Carpenter@ed.ac.uk,
MPA@staffmail.ed.ac.uk, Joanna.Wardlaw@ed.ac.uk

Abstract

In this paper, we will present the implementation of a deconvolution algorithm for brain perfusion quantification on GPGPU (General Purpose Graphics Processor Units) using the CUDA programming model. GPUs originated as graphics generation dedicated co-processors, but the modern GPUs have evolved to become a more general processor capable of executing scientific computations. It provides a highly parallel computing environment due to its huge number of computing cores and constitutes an affordable high performance computing method. The objective of brain perfusion quantification is to generate parametric maps of relevant haemodynamic quantities such as Cerebral Blood Flow (CBF), Cerebral Blood Volume (CBV) and Mean Transit Time (MTT) that can be used in diagnosis of conditions such as stroke or brain tumors. These calculations involve deconvolution operations that in the case of using local Arterial Input Functions (AIF) can be very expensive computationally. We present the serial and parallel implementations of such algorithm and the evaluation of the performance gains using GPUs.

***Keywords*-Perfusion Imaging; Deconvolution; Parallelization; GPGPU;**

I. Introduction

With the development of computed tomography (CT) and magnetic resonance (MR) imaging, perfusion imaging becomes a very powerful clinical tool for evaluation of brain anatomy. They can be used to evaluate brain function via assessment of cerebral perfusion parameters.

The main applications of brain perfusion imaging are acute stroke and brain tumors. In the case of acute stroke, the information obtained from brain perfusion imaging can be used to evaluate the appropriateness of administering thrombolytic treatment, which can help to reduce the final volume of dead tissue, but has some risks such as hemorrhages. The results are used to evaluate the possible benefits. In the case of tumors, they are used to distinguish tumor characteristics and follow tumor development, possibly also after treatment to see whether it has been effective.

Evaluating tissue time-concentration curve of a contrast agent intensity after its injection, has become possible on time scales comparable with the mean transit time (MTT). To achieve this, deconvolution is used in perfusion imaging to obtain the Impulse Response Function (IRF) that is then used to create parametric maps of relevant haemodynamic quantities such as Cerebral Blood Flow (CBF), Cerebral Blood Volume (CBV) and Mean Transmit Time[1], [2], [3]. Cerebral blood flow indicates the volume of blood flowing through a given voxel in a given time. Cerebral blood volume refers to the volume of blood in a given voxel of brain tissue. Mean transit time designates the average time blood takes to flow through a given voxel of brain tissue, it is commonly measured in seconds. Time To Peak (TTP) and Time of Arrival (TA) are two other parameters offen be measured [4]. TA refers to the time of arrival of the contrast agent in the voxel after injecting contrast agent. TTP refers to corresponding time of the maximum contrast variation. In previous studies, Singular Value Decomposition (SVD) and its variants were proved to be applicable to perform deconvolution in perfusion imaging [5]. As the raw data obtained from CT or MR scanners is not noise free and as deconvolution is very sensitive to noise, truncated SVD is used to minimize the noise impact [6], [7], [8], [9].

Using voxel based different local Arterial Input Functions (AIF), instead of a single global one, was introduced and showed its advantages to improve the accuracy of parametric maps [10], [11]. However, using local AIFs leads to fairly slow performance as it becomes necessary to decompose many thousands of different AIFs. The running time for the perfusion imaging analysis can be more than half an hour. As time is crucial in clinical diagnosis, there is pressing demand to speed up the analysis.

General-purpose computing on graphics processing units (GPGPU) [12] are state-of-the-art approaches to many computing applications. They provide a highly parallel computing environment due to their huge number of computing cores and constitute an affordable, high-performance computing platform.

In this paper, we present a GPGPU-based brain perfusion imaging analysis implementation using the CUDA programming model. We also compared the performance of the serial and parallel perfusion imaging analysis methods.

II. Algorithm for Perfusion Imaging

From a CT or MR scanner, we get a series of brain images at different sampling times. For each voxel, we collect data at specific time intervals to build a tissue time-concentration curve of contrast agent intensity, which is also called volume of fluid (VOF) curve. This curve will be referred to as C_t.

The other input is the local AIF matrix, which is referred to as C_a. First of all, a local AIF vector is generated by measuring a small set of vessels in a specified area near the voxel of interest. Then a local AIF matrix is created from the local AIF vector as follows:

$$C_a = \Delta t \begin{pmatrix} C_a(t_1) & 0 & \cdots & 0 \\ C_a(t_2) & C_a(t_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ C_a(t_N) & C_a(t_{N-1}) & \cdots & C_a(t_1) \end{pmatrix} \quad (1)$$

where $(t_1, t_2, \cdots, t_N)$ is the sampling time, $(Ca(t_1), Ca(t_2), \cdots, Ca(t_N))$ is an arterial input function given as an input and Δt is time scale.

In perfusion imaging, the output we want to obtain is Impulse Response Function (IRF), which is referred to as h.

The volume of fluid, C_t, the C_a, and IRF h satisfies the following equation:

$$C_t = C_a \otimes h + \epsilon \quad (2)$$

where $\otimes$ denotes convolution and ϵ is the noise.

Finally, the CBF, CBV and MTT for each voxel are calculated as follows:

$$CBF = Max(h) \quad (3)$$

$$CBV = \int_0^\infty h(t)\,dt \quad (4)$$

$$MTT = CBF/CBV \quad (5)$$

Singular Value Decomposition (SVD) is one of the most popular techniques to solve deconvolution problems in perfusion imaging. Suppose C_a from Equation (1) is an m-by-m matrix, there exists a factorization such that:

$$C_a = U \cdot W \cdot V^T \quad (6)$$

where U is an $m \times m$ unitary matrix, W is $m \times n$ diagonal matrix and V^* is the transpose of an $n \times n$ unitary matrix V. A common convention is to order the diagonal matrix W in a decreasing order and this diagonal entries of W are known as the singular values of original matrix C_a.

The C_a^{-1} can then be written as:

$$C_a^{-1} = V \cdot W^{-1} \cdot (U^T) \quad (7)$$

To solve the deconvolution problem in Equation (2), The solution can be simply delivered after applying SVD:

$$h = V \cdot W^{-1} \cdot (U^T \cdot C_t) \quad (8)$$

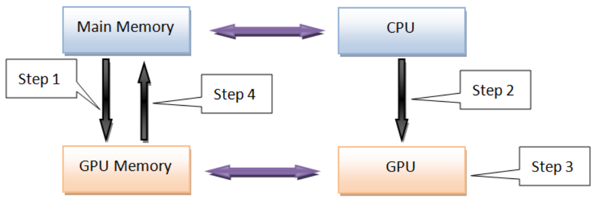

Figure 1. CUDA Data and Control Flow

Furthermore, as rows in C_a in Equation (2) are close to linear combinations, the deconvolution is an ill-posed problem, hence, it is very sensitive to noise. Truncated SVD is introduced to minimize the noise. In truncated SVD, a threshold is added and elements of the diagonal matrix W whose value is smaller than this threshold will be set to zero [7], [8].

III. CUDA for GPGPU

A. What is CUDA

Compute Unified Device Architecture (CUDA) is a parallel computing architecture developed by NVIDIA in 2006 [13] with an associated software toolkit. It is the entry point for developers who prefer high-level computer programming, compared with Open Computing Language (OpenCL), which is the entry point for developers who want low-level Application Programming Interfaces (APIs).

C for CUDA offers programmers a simple way to write C-like programs for GPGPUs. It consists of a set of extensions to the C language for code running on CPUs and a runtime library for code running on GPUs. In addition, it allows programmers to access GPUs via low level APIs to avoid the overhead graphics APIs common with. It significantly reduces the runtime overhead of GPGPU applications. As a result, CUDA has become one of the most popular programming languages for GPU programming.

B. CUDA Data and Control Flow

Figure 1 is a typical example of CUDA processing:
1. Copy data from main memory to GPU memory.
2. CPU instructs the GPU to start processing.
3. GPU executes in parallel on each core.
4. Wait for completion.
5. Copy the result from GPU memory to main memory.
6. CPU acts on result, and return to step 1.

IV. Algorithm

Truncated Singular Value Decomposition mentioned above is used to calculate the IRF. The following defines the variables in the pseudo code.
Input: 4D MR or CT image data stored in Nifti format [14] file.
Output: A set of CBF, CBV and MTT colored maps.
Time: the number of time intervals.

ALGORITHM 1 - SERIAL PERFUSION IMAGING ANALYSIS

```
 1  $A(1:Time, 1:Size) \leftarrow$ 4D MR or CT image data
 2  **if** DoImageDenoising = $true$
 3      **then** $A'(1:Size, 1:Time) \leftarrow$ reorganise $A(1:Time, 1:Size)$
 4      **else**  $A''(1:Size, 1:Time) \leftarrow$ Denoise and reorganise $A'(1:Size, 1:Time)$
 5  **for** $i \leftarrow 1$ **to** $dim$
 6        **do** Generate $localAIF(1:Time)$
 7           $IRF(1:Time) \leftarrow$ Deconvolution result (A"(i,1:Time) & localAIF(1:Time))
 8           $CBF(i) \leftarrow Max(IRF(1:Time))$
 9           $CBV(i) \leftarrow Sum(IRF(1:Time))$
10           $MTT(i) \leftarrow CBV(i)/CBF(i)$
11  CBF colored map $\leftarrow$ CBF(1:Size)
12  CBV colored map $\leftarrow$ CBV(1:Size)
13  MTT colored map $\leftarrow$ MTT(1:Size)
```

ALGORITHM 2 - PARALLEL PERFUSION IMAGING ANALYSIS

```
 1  $CPU.A(1:Time, 1:Size) \leftarrow$ 4D MR or CT image data
 2  $GPU.A(1:Time, 1:Size) \leftarrow CPU.A(1:Time, 1:Size)$
 3  GPU: Parallel do, shared(A, A', A")
 4  **if** DoImageDenoising = $true$
 5      **then** $GPU.A'(1:Size, 1:Time) \leftarrow$ reorganise $GPU.A(1:Time, 1:Size)$
 6           $GPU.A''(1:Size, 1:Time) = GPU.A'(1:Size, 1:Time)$
 7      **else**  $GPU.A''(1:Size, 1:Time) \leftarrow$ Denoise and reorganise $GPU.A(1:Time, 1:Size)$
 8  GPU: Parallel do, $private(localAIF, i, IRF), shared(A, CBF, CBV, MTT)$
 9  **for** $n \leftarrow 1$ **to** $Dim3$
10        **do for** $i \leftarrow 1$ **to** $Dim1 \times Dim2$
11               **do** Generate $localAIF(1:Time)$
12                  $IRF \leftarrow$ Deconvolution result (GPU.A"(i+n $\times$ Dim1 $\times$ Dim2,1:Time) & localAIF(1:Time))
13                  $GPU.CBF(i + n \times Dim1 \times Dim2) \leftarrow Max(IRF)$
14                  $GPU.CBV(i + n \times Dim1 \times Dim2) \leftarrow Sum(IRF)$
15                  $GPU.MTT(i + n \times Dim1 \times Dim2) \leftarrow GPU.CBV/GPU.CBF$
16           $CPU.CBF(slice\ n) \leftarrow GPU.CBF(slice\ n)$
17           $CPU.CBV(slice\ n) \leftarrow GPU.CBV(slice\ n)$
18           $CPU.MTT(slice\ n) \leftarrow GPU.MTT(slice\ n)$
19           CBF colored map $\leftarrow CPU.CBF(slice\ n)$
20           CBV colored map $\leftarrow CPU.CBV(slice\ n)$
21           MTT colored map $\leftarrow CPU.MTT(slice\ n)$
```

Dim1, Dim2, Dim3: the size of each dimension.
Size: the size of each 3D brain image which equals to $Dim1 \times Dim2 \times Dim3$.
A(): a 4D array used to store data directly read from brain images.
A'(): a 4D array used to store data after reorganization.
A"(): a 4D array used to store data after denoising.
IRF: a 1D array used to temporary store the result of deconvolution.
CBF(),CBV(),MTT(): 3D arrays used to store the analyzed result.
CPU.A: Parameter A is stored on the CPU.
GPU.A: Parameter A is stored on the GPU.
GPU $A \leftarrow B$: Operation $A \leftarrow B$ is executed on the GPU.
GPU.A $\leftarrow$ *CPU.A*: Copy data from CPU to GPU.
CPU.A $\leftarrow$ *GPU.A*: Copy data from GPU to CPU.

A. Serial Perfusion Imaging Analysis

The algorithm for perfusion imaging analysis without parallelization can then be written as Algorithm 1.

The first step (Line 1) is to load MR or CT imaging data stored in NIfTI format file. The computational complexity of step one is $O(time * Dim1 * Dim2 * Dim3)$.

As in deconvoluting step (Line 3), we always read the intensity value of one voxel along all of the time intervals together, we can optimise the data localization and reduce the cache swap cost if data is organized in the form of $Dim1 * Dim2 * Dim3 * time$. So the second step is to reorganize data into this form. The computational complexity of this step is $O(time * Dim1 * Dim2 * Dim3)$.

As blood always flows from one cell to its neighbors, the intensity values should be continuous. This allow us to use an image level denoising method (Line 4) such as applying 2D, 3D and 4D weighted mean filters. The computational complexity of this step is also $O(time * Dim1 * Dim2 * Dim3)$.

Line 6 to 10 is the deconvolution. This operation runs voxel by voxel. The most expensive part in the deconvolution is to decompose local AIF matrices using singular value decomposition whose computational complexity is $O(time^2)$ for each $time * time$ matrix. The computational complexity of deconvolution can be roughly considered as the same as decomposition: $O(time^2)$. Using SVD, three $time^2$ local arrays and one $time$ array is required for each voxel. So the space complexity of each deconvolution is $(3 * time^2)$. Note that all of these arrays are local data and will be freed immediately after deconvolution calculation completes.

Furthermore, as voxel based deconvolution in Line 5 to Line 10 needs to be repeated $Dim1 * Dim2 * Dim3$ times, the overall computational complexity is $O(Dim1 * Dim2 * Dim3 * time^2)$. This is the most expensive part of the whole workflow, more details can also be found in Section V-B.

B. Parallel Perfusion Imaging Analysis

As GPGPUs is an ideal solution for matrix operations; it can be expected to improve the performance of *Data reorganization* and *Denoising* steps. In the deconvolution step, the deconvolution of different voxels are pleasingly parallel tasks, so that there is little effort required to separate the problem into parallel tasks and there is no dependency or communication between those parallel tasks, parallel implementation can be easily achieved. The parallel algorithm for the whole workflow can then be written as Algorithm 2.

For the serial algorithm, the first step in the parallel implementation is to load images into CPU memory (Line 1). The implementation of this step is exactly as before, so its computational complexity remains $O(time * Dim1 * Dim2 * Dim3)$.

Line 2 is an extra step, as mentioned in section III-B, we need to copy data from CPU memory to GPU memory first. Furthermore, we also need to copy results from GPU memory back to CPU memory before we can write them into parametric maps (Lines 16 to 18).

The *Data reorganization* (Line 5) and *Denoising* (Line 7) steps are matrix operations. As GPGPUs are good at matrix operations, these two steps can be simply optimized by assigning one GPU thread to each matrix element.

Lines 7 to 15 are the most expensive part of the whole workflow. Decomposition of each local AIF matrix (the dominant part of deconvolution), whose size is 80×80, is not large enough to be parallelised (Section IV-C). Consequently, we assign each decomposition to different GPU threads. Synchronization and result colletcion is performed as a sequence of operation on slice. The reason of doing this is that it reduces the total local memory requirement and enables users to access some results before the whole workflow finishes.

The last step, *Drawing parametric maps*, is also the same as in the serial version. The computational complexity is also $O(time \times Dim1 \times Dim2 \times Dim3)$.

Table I
COMPUTATION TIME FOR SVD (IN SECONDS)

Matrix Size	MATLAB	MKL	GPU
64 x 64	0.01	0.003	0.054
128 x 128	0.03	0.014	0.077
256 x 256	0.210	0.082	0.265
1K x 1K	72	11.255	3.725
2K x 2K	758.6	114.625	19.6
4K x 4K	6780	898.23	133.68

C. Using GPGPU in SVD

Lahabar [15] compared the performance in terms of speed of SVD in MATLAB, SVD in Intel Math Kernel Library (MKL) 10.0.4 LAPACK and his implementation on GPU using CUDA. The test environment is an *Intel Dual Core 2.66GHz* PC and *NVIDIA GTX 280* graphics processor. Their study focuses on evaluating the performance of parallel and serial versions of the SVD algorithms rather than some specified application of SVD.

As the largest data set in our case was a 80×80 matrix, using GPGPU for individual matrix decomposition is not suitable according to the results in Table I from [15]. From this table, SVD using GPU will improve the performance only if the matrices are larger than $1K \times 1K$ but will impair the performance for small matrices. In our case, the matrices we want to decompose range from 44×44 to 80×80 which are too small to obtain improvement. As a result, using GPGPU for individual matrix decomposition will not gain performance improvement.

Table II
PERFORMANCE OF EACH STEP

Step	Serial Running Time (s)	Parallel Running Time (s)	Speedup Factor
Brain data load	0.10	0.10	Not Applied
Data copying (CPU to GPU)	Not Applied	0.17	Not Applied
Data reorganization	1.1	0.01	110
Reorganization & denoising	4.3	0.01	430
Deconvolution	2.1×10^3	8.2×10^2	2.5
Data copying (GPU to CPU)	Not Applied	0.01	Not Applied
Draw parametric maps	0.20	0.20	Not Applied
Overall	2.1×10^3	8.2×10^2	2.5

Table III
OVERALL PERFORMANCE

Data Size ($Dim1 \times Dim2 \times Dim3 \times time$)	Serial Running Time (min)	Parallel Running Time (min)	Speedup Factor
128*128*11*44	6.0	1.25	4.8
128*128*22*80	35	13.5	2.6

V. PERFORMANCE

A. Experimental Environment

The performance tests run on ECDF (The Edinburgh Compute and Data Facility) [1]. In our experiment, the worker node we use contains 4 $Intel(R)Xeon(R)$ CPU cores and connects to two Tesla C1060 GPUs which provide 480 GPU cores in total. The frequency of each CPU core is 3.0 GHz and the frequency of GPU core is 1.44 GHz. Its single precision floating point performance (peak) is 2.073 TFLOPS. It has 2.0 GB of global memory and 8 KB of shared memory. The parametric maps produced by serial and parallel implementations are identical. In the other word, the quality of the results is not changed. The test data we used are simulated images contains $128 \times 128 \times 22$ voxels and the number of time intervals is 80, which is the same as MR images. The results showed below are the arithmetic mean of ten repeated tests.

B. Performance for Each Step

Table II indicates our measurement of the performance for each step in the whole workflow.

The steps *Brain data load* and *Draw parametric maps* are not suitable for parallelization and their running time in parallel version can be considered as the same as in the serial version.

In parallel deconvolution, the first step of parallel workflow is to copy data from CPU memory to GPU memory. The input data is about 55 MB, which is mainly an array with $128 \times 128 \times 22 \times 80$ $short$ elements. The copying takes 0.17 seconds. The result size to be moved back from GPU memory to CPU memory is much smaller and only takes 0.01 second to perform the copy back operation.

In serial deconvolution, the *Reorganization & Denoising* step, which performs the reorganization and denoising prior to deconvolution, takes 4.3 seconds compared to the 1.1 seconds for reorganization only. After applying parallelization to these steps, the performance dramatically reduced to 0.01 seconds. The speedup factors are 430 and 110, respectively.

The running time of the *Deconvolution* step, the most expensive one, reduced from 2100 seconds to 820 seconds after applying parallelization. The speedup factor is 2.6.

This result supports the computational complexity analysis mentioned in section IV-A and section IV-B.

C. Overall Performance

As the running time is dominated by *Deconvolution* step, the overall running time can be roughly considered as the same as the running time of deconvolution step which is also showed in Table II. In other words, the final performance depends on *Deconvolution* step and the overall speedup factor is also 2.6.

D. Comparison with Previous Approach

C. Lorenz [10] did experiments on deconvolution using local AIFs. They did performance experiments on a small data set size level, which was 128×128 voxels per slice, 11 slices and the number of time intervals was 44. The overall running time (Table III) to finish their deconvolution is six minutes with speedup factor of 4.8. This is reduced to one and a quarter minutes after applying GPGPU approach. However, in our research, the data size has increased to 128×128 voxels per slice, 22 slices and the number of time intervals is now 80, approximately four times as much data. The serial running time for such a data set would expand to around 35 minutes (estimated) using their approach. This is close to the results from our experiments. After using GPGPUs, the running time reduced to 12 minutes, with a speedup factor of 2.6.

[1]ECDF: https://www.wiki.ed.ac.uk/display/ecdfwiki/Home

VI. Conclusion

In this paper, we introduced an implementation of perfusion imaging analysis which provides considerable speed improvement and equivalent quality of results than current serial implementations. The *Deconvolution* step is the bottleneck for perfusion imaging analysis, although the speedup factor is more than a hundred for both the *Data reorganization* and *Denoising* steps, the overall performance speedup is 2.6. The estimated improvement over previous methods on similar data is a facter of 4.8. The speedup depends on data size and as resolution and number of time steps used for brain imaging increases, this approach will show great benefits. In clinical diagnosis, time is vitally important especially for acute stroke cases, the earlier we deliver the result for diagnosis, the higher the possibility that patients will be cured. Therefore, performance is as important as accuracy in perfusion imaging and our implementation can be used to help clinical diagnosis. In conclusion, using GPGPU is a desirable approach in perfusion imaging analysis.

Acknowledgment

The authors wish to express their gratitude to Dr. Jano van Hemert for his guidance, assistance and support.

The authors also would like to thank the Scottish Imaging Network - A Platform for Scientific Excellence (SINAPSE) for providing the financial funds and trainings and to the school of informatics, University of Edinburgh for their laboratory facilities.

The work was carried out at the SFC Brain Imaging Research Centre (www.sbirc.ed.ac.uk), Division of Clinical Neurosciences, University of Edinburgh, a core area of the Wellcome Trust Clinical Research Facility and part of the SINAPSE collaboration funded by the Scottish Funding Council and the Chief Scientist Office.

This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF). (http://www.ecdf.ed.ac.uk/). The ECDF is partially supported by the eDIKT initiative (http://www.edikt.org.uk).

References

[1] P. Meier and K. Zierler, "On the theory of the indicator-dilution method for measurement of blood flow and volume," *J Appl Physiol*, vol. 6, no. 12, pp. 731–44, 1954.

[2] J. Gore and S. Majumdar, "Measurement of tissue blood flow using intravascular relaxation agents and magnetic resonance imaging," *Magn Reson Med*, vol. 14, no. 2, pp. 242–8, 1990.

[3] G. Gobbel and J. Fike, "A deconvolution method for evaluating indicator-dilution curves," *Phys Med Biol*, vol. 39, no. 11, pp. 1833–54, 1994.

[4] O. Wu, L. Ostergaard, R. Weisskoff, T. Benner, B. Rosen, and A. Sorensen, "Tracer arrival timing-insensitive technique for estimating flow in mr perfusion-weighted imaging using singular value decomposition with a block-circulant deconvolution matrix," *Magn Reson Med*, vol. 50, no. 1, pp. 164–74, 2003.

[5] H. Liu, Y. Pu, Y. Liu, L. Nickerson, T. Andrews, P. Fox, and J. Gao, "Cerebral blood flow measurement by dynamic contrast mri using singular value decomposition with an adaptive threshold," *Magn Reson Med*, vol. 42, no. 1, pp. 167–72, 1999.

[6] T. O'Haver, "An introduction to signal processing in chemical analysis," *University of Maryland at College Park*, 2009.

[7] L. Ostergaard, R. Weisskoff, D. Chesler, C. Gyldensted, and B. Rosen, "High resolution measurement of cerebral blood flow using intravascular tracer bolus passages. part i: Mathematical approach and statistical analysis," *Magn Reson Med*, vol. 36, no. 5, pp. 715–25, 1996.

[8] L. Ostergaard, A. Sorensen, K. Kwong, R. Weisskoff, C. Gyldensted, and B. Rosen, "High resolution measurement of cerebral blood flow using intravascular tracer bolus passages. part ii: Experimental comparison and preliminary results," *Magn Reson Med*, vol. 36, no. 5, pp. 726–36, 1996.

[9] R. Wirestam, L. Andersson, L. Ostergaard, M. Bolling, J. Aunola, A. Lindgren, B. Geijer, S. Holtås, and F. Ståhlberg, "Assessment of regional cerebral blood flow by dynamic susceptibility contrast mri using different deconvolution techniques," *Magn Reson Med*, vol. 43, no. 5, pp. 691–700, 2000.

[10] C. Lorenz, T. Benner, P. J. Chen, C. J. Lopez, H. Ay, M. W. Zhu, N. M. Menezes, H. Aronen, J. Karonen, Y. Liu, J. Nuutinen, and A. G. Sorensen, "Automated perfusion-weighted mri using localized arterial input functions," *J Magn Reson Imaging*, vol. 24, pp. 1133–1139, Nov 2006.

[11] G. Duhamel, G. Schlaug, and D. Alsop, "Measurement of arterial input functions for dynamic susceptibility contrast magnetic resonance imaging using echoplanar images: comparison of physical simulations with in vivo results," *Magn Reson Med*, vol. 55, no. 3, pp. 514–23, 2006.

[12] M. Harris, "Gpgpu: General-purpose computation on gpus," in *Game Developpers Conference*, 2005.

[13] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *Micro, IEEE*, vol. 28, no. 2, pp. 39–55, 2008.

[14] R. Cox, J. Ashburner, H. Breman, K. Fissell, C. Haselgrove, C. Holmes, J. Lancaster, D. Rex, S. Smith, J. Woodward, *et al.*, "A (sort of) new image data format standard: Nifti-1," *Human Brain Mapping*, vol. 25, 2004.

[15] S. Lahabar and P. J. Narayanan, "Singular value decomposition on gpu using cuda," *2009 IEEE International Symposium on Parallel and Distributed Processing*, 2009.