INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 9(6):768-775, December, 2014.

GLM Analysis for fMRI using Connex Array

A. Ţugui

Andrei Ţugui
Politehnica University of Bucureşti
Romania, 061071 Bucureşti, Splaiul Independenţei, 313
andrei.tugui@yahoo.com

Abstract: In the last decades, magnetic resonance imaging gained lot of popularity, and also functional magnetic resonance imaging (fMRI), due to the fact that MRI is a harmless and efficient technique for human cerebral activity studies; fMRI aims to determine and to locate different brain activities when the subject is doing a predetermined task. In addition, using fMRI analysis, nowadays we can make prediction on several diseases. This paper's purpose is to describe the General Linear Model for fMRI statistical analysis algorithm, for a 64 x 64 x 22 voxels dataset on a revolutionary parallel computing machine, Connex Array. We make a comparison to other computing machines used in the same purpose, in terms of algorithm time execution (statistical analysis speed). We will show that by taking advantage on its specific parallel computation each step in GLM analysis, Connex Array is able to answer successfully to computational challenge launched by fMRI computation: the speed-up.
Keywords: Connex Array, Functional magnetic resonance imaging, Image reconstruction, Parallel algorithms, Parallel processing.

1 Introduction

Nowadays, neurological activity can be studied using several investigation techniques, each of them having its own advantage and disadvantage, by studying human brain from several perspectives. Although other techniques like EEG (ElectroEncephaloGraphy) or MEG (Magneto Encephalography) have a satisfactory temporal resolution (milliseconds), when it comes to spatial resolution, PET (Positron Emission Tomography) and fMRI are much more indicated to use [1] - [4]. Those investigation techniques pick information from the blood flow changes. As for fMRI statistical analysis on which we will focus in this paper, far away from the biological and functional characteristics, from computational point of vue it is very time consuming. Processing dataset is basically MRI images acquired voxel by voxel; once the blood flow changes, MRI signal strength changes also, this way one can analyze these changes using the statistical data analysis.

Typically, input data acquired one time can be in range of 100 000 voxels, but the investigation's main problem remains the fact that this data is repeatedly acquired by 100 up to 2000 times [5] (usually because of head moving, to compensate discrete acquisition in time, slice by slice, or because most of the time we acquire a lot of noise which must be filtered). A typical fMRI computational chart is presented in Fig. 1. Usually, fMRI dataset acquired during one observation consist of 8 volumes 64 x 64 x 22 voxels each, having a 3.75 x 3.75 x 3.75 mm spatial resolution. If we represent this data as simple precision floats, the data requires about 29 MB memory space. Obviously, we deal to a high enough spatial and temporal memory space, in fact the acquisition process itself is not time-consuming (usually 1 volume/s), but the processing data is, which bottlenecks a lot nowaday's computers tasks, especially when large amount of data must be computed in real time [6]- [7]. Estimated, today we can process a volume dataset using a dedicated processor during about 5 s, but if we use a GPU, the processing time decreases to just 0.5 s! So, this is the high importance that we assign to processing speed in fMRI computation.

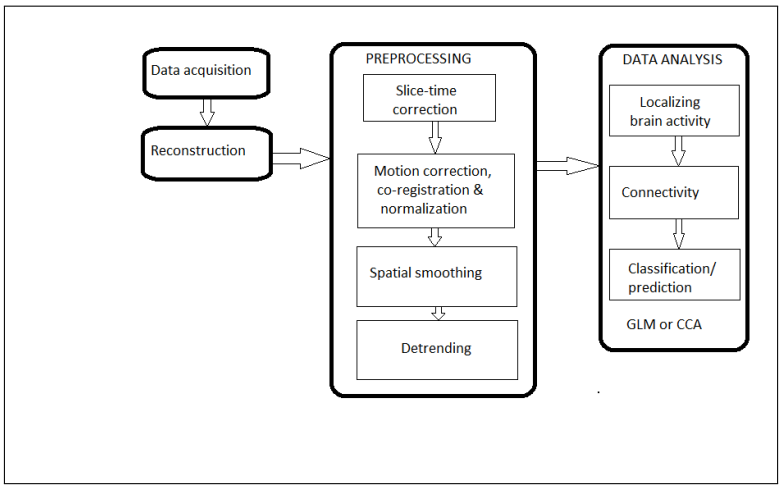

Figure 1: fMRI pipline

In addition, many times fMRI data reconstruction and analysis are made in real time, like BCI (Brain Computer Interface) —a cooperation method between the PC and the subject aiming to solve a given task.
As one can see in Fig.1, fMRI data analysis requires two steps:

- Preprocessing input dataset
- Statistical dataset analysis

In the following, we will present a typically 3D fMRI volume computation using input data from [8], meaning first the preprocessing step and then the statistical data analysis using the GLM (General Linear Model), on a revolutionary parallel computing machine, Connex Array. A full description of this parallel machine's architecture and processing manner is given in papers [9]-[10]. We introduce here only the data vector definition, being in fact the cell that Connex operates, a new N-length data type containing fMRI samples, modeling the vectors from Connex Array [10]. Once we understand the vectorial architecture, we will describe much better this machine's parallelism. Thereby, if we have a C language instruction to be executed as the following one:

```
for( ind = 1;ind <= SIZEOF_VECTOR;ind ++ )
    vect3[ind] = vect1[ind] + vect2[ind]
```

then Connex will sum in one step vect1 and vect2 in the sum vector vect3, whereas a sequential processor would normally do the addition element by element in SIZEOF_VECTOR steps. Here, SIZEOF_VECTOR is a constant holding the vector's length we operate with, by default. Thereby, by typical operators' overloading, using a special C++ library named CVector [11], Connex can sum, multiply, decrease or shift vectors of data. All cycling instructions like *for* are executed using a new CVector instruction like the following:

```
    WHERE (vect1 %2 == 0) {vect1 = 0}
ELSEWHERE {vect1 = 1} ;
```

2 fMRI data preprocessing

Preprocessing usually requires signal filtering. The acquired signal is exposed to a lot of perturbation as head moving (physiological noise), which can be eliminated using Motion Cor-

rection filtering step, or even the acquisition process itself, which is done in discrete steps (once a section) can generate noise: this noise is filtered using the Slice Time Correction filtering step. We show this discrete time slice acquisition of one voxel in Figure 2.

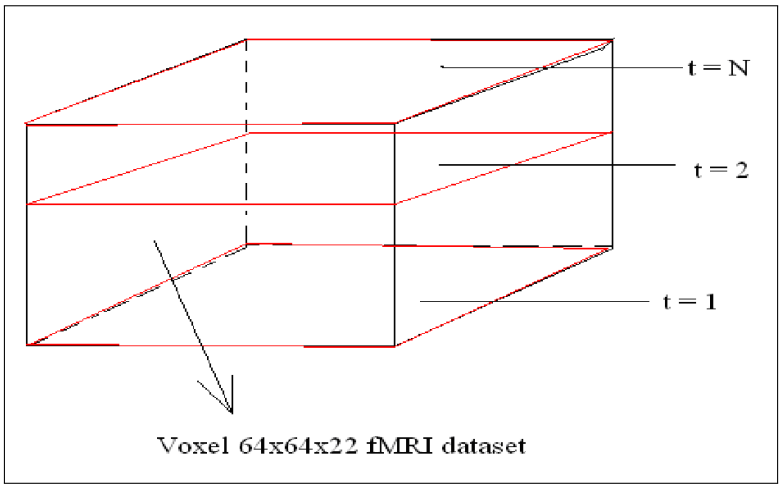

Figure 2: fMRI slice time acquisition

3 Slice time correction

During this preprocessing step, slice time correction is made using *sinc* interpolation. From computational point of view, this step involves:

- one 1D FFT (one Fast Fourier Transform)
- one point-wise multiplication
- one 1D IFFT (Inverse Fast Fourier Transform)

Loading dataset into Connex's memory was made thereby in 64 vectors, 64 floating point elements each (one slice at the time), this way:

```
    vector<float>X[64];
X[0] = [-0.00416487...0.09823550 ];
X[63] = [-0.00687657...0.09823550];
```

1DFFT computation took 6 steps using Cooley-Tuckey algorithm [12], each step the output vector is loaded with the old sample and summed with the new resulted sample, like this:

$$X[0] += (c_i * X[0] + x_{12}) \tag{1}$$

where c_i is the coefficient vector at current step i and x_{12} is a temporary vector obtained from successive input vector shifts, like:

```
    i = 0;
temp1 = shiftLeft(X[0],32);
WHERE (INDEX <32)
{
```

$$x_{12} = temp1;$$

```
}
temp1 = shiftRight(X[0],32);
i += 32;
where (INDEX >= i && INDEX <(i+32))
{
```

$$x_{12} = temp1;$$

} [13]

For *sinc* function computation we used this formula to multiply each pixel:

$$sinc[(\pi/TR)(r - iTR)] \tag{2}$$

where TR = repetition time for each pulse sequence, resulting a coefficient vector c[INDEX] like this:

```
    WHERE (X[INDEX] == 0) {c[INDEX] = 1;}
ELSEWHERE { c[INDEX]=(sin(3.14*(1-INDEX*2)/2))/ (3.14*(1- INDEX*2)/2);}
```

For the reverse Fourier transform computation, the algorithm is similar to 1DFFT algorithm. It worth nothing to see that for this preprocessing step, the total computation time used by SPM (Statistical Parametric Mapping Software) is about 32 s [14], but the same algorithm implementation on Connex Array took only about 100 ms. It worth nothing to say that the same algorithm implementation using fixed-point representation is much faster [15].

4 Image registration

This preprocessing step filters the noise generated by head movement. It involves the alignment of all voxels to a reference voxel (still state). For fMRI, it is sufficient one resolution range with 3 iterations per volume. In each iteration, three quadrature filters are applied on the x, y and z directions, the filters having each 7 x 7 x 7 voxels, complex elements and not Cartesian separable. From the filter response we compute three phase shifts, three gradient shifts and the statistical certainty. Then it results an equation system by adding all voxels and all filters to find the optimization parameter vector. From the optimization parameter vector we compute then a movement vector for each voxel and we apply trilinear interpolation using a 3D texture to rotate and translate the volume. Paper [16] describes the complete image registration algorithm. Head movement field is modeled using a 12—length parameter vector p, like:

$$p = [p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}]^T \tag{3}$$

p vector is computed by solving the 12–length linear equation system with:

$$p = A^{-1}h \tag{4}$$

In this paper we used FFT convolution to compute head movement, although there is also the possibility to use spatial convolution in this purpose. It is worth nothing to follow the inverse matrix computation with Connex, which reveals the parallelism used once the data is loaded into memory. One may consult papers [10] and [13] for further information on Connex architecture and its vectorial computation. We mention that image registration is the most time consuming preprocessing step (about 95% from total time computation time used for a single volume), but using Connex's computational power, we showed once again (see Table 1)

that time bottleneck can be eliminated using Connex Array. If we compare image registration algorithm implementation on Connex Array to the same algorithm implemented on an usual PC using Matlab testing environment, in Matlab we solve this problem during about 14 s, whereas Connex uses only half of this time (Table 1). A closer look on matrix computation with Connex, intense used in this paper, can be found in reference [17].

5 GLM smoothing algorithm

Smoothing dataset is an additional filter step applied before statistical fMRI analysis. Typically, this filter is done for each voxel, except for those voxels near the edge of dataset, and it involves one 3D convolution. Basically, this is done by using a Gaussian lowpass Cartesian separable filter, so the smoothing algorithm becomes in fact a 1D convolution applied in all the three directions (x, y, z). For fMRI smoothing, we use typically 9 x 9 x 9 filters. Thus, for an entire 1D slice we define:

- the input vector:
  - vector<float >X[64];
- the output vector:
  - vector<float >Y[64];
- the impulse response vector:
  - vector<float >H;

By applying smoothing to each line (finally to the entire slice and then to the entire volume), the output vector is processed by multiplying the input vector with the impulse response vector:

$$Y = XH \tag{5}$$

6 Detrending

The final step in preprocessing fMRI data is a special filtering step called detrending. Thus, papers [18] and [19] show how exactly detrending is eliminating the drifts caused by physiological noise and scanner imperfections. Typically, detrending is applied to each slice of the voxel and it searches for the best linearity fit between the slices on one hand, and a particular polynomial on the other hand, fit which is eliminated by the filtering process (polynomial detrending). This filtering procedure uses in fact linear regression, an algorithm similar to statistical GLM analysis, which we will describe in the following.

7 Statistical fMRI analysis using GLM model

After preprocessing, fMRI dataset is subjected to a statistical analysis which can be made using two approaches [20]:

- General Linear Model (GLM)
- Canonical correlation Analysis (CCA)

In this paper we approached General Linear Model (GLM) analysis on each slice (the whole volume), where observation matrix is computed like this:

$$Y = X\beta + \epsilon \tag{6}$$

where:

- Y = the observations (all samples from the slice)
- β = optimization parameters
- X = design matrix given by stimulus paradigm
- ϵ = the errors can not be explained by the model

By minimizing $\| \epsilon \|^2$ it can be shown that the best parameters [14] are given by:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \tag{7}$$

The term $(X^T X)^{-1} X^T$ (we note it with C) is slice independent and can be precomputed and loaded into memory. The only thing we must compute for each voxel in order to find the estimated parameters is the product between the constant C and the current slice Y. If we define the contrast as a column vector (e.g. $[1\ 0]^T$), the test value t [14] is given by:

$$t = \frac{c^T \hat{\beta}}{\sqrt{var(\hat{\epsilon})} c^T (X^T X)^{-1} c} \tag{8}$$

The matrix product is computed fast, as the error and its variance. Then we load again the slice time-series into memory to compute the mean error:

$$\bar{\epsilon} = \frac{1}{N} \sum_{t=1}^{N} (Y(t) - X(t)\hat{\beta}) \tag{9}$$

N = number of time-points and X(t) = design matrix values corresponding to time-point t. The third memory load is made this time to compute error variance an then the test value t. The term $c^T (X^T X)^{-1} c$ is a scalar in fact, which can be precomputed. On Connex, parameter matrix is loaded into 64 vectors 64-length pixels. After the data load into the memory and the matrix computation, we compute matrix product $\hat{\beta} = c^T Y$ for each pixel this way:

$B_{ij} = addAll(Y[i] * C^T[j])$,

`WHERE (INDEX == i) {B[i] =` B_{ij}`; }`

addAll adds all products from elements located at the same positions:

$M_i = addAll(e[i])$

`WHERE (INDEX == i) { M =`M_i`; }`

Finally, the test value t is computed with:

$T[0] = B[0] * (C^T[0] * \sqrt{(variance)})$

where *variance* is the statistical variance (the difference between the mean's square and the mean, computed prior).

8 Conclusions

If on a sequential processor like SPM, GLM analysis would take about 33 s, GLM algorithm implementation on Connex Array requires only 1.7 ms (see Table 1). We demonstrate in this paper that the speed-up in fMRI reconstruction is a challenge that Connex Array can easily overcome.

Table 1: fMRI GLM data analysis on different processors

GLM step	SPM	Matlab	OpenMP	Matlab CUDA	Connex Array
Slice time correction	32 s	280 ms	235 ms	7.8 ms	100.4 ms
Motion compensation	28 s	13.7 s	4.6 s	650 ms	7.857 s
Smoothing	32 s	1.5 s	195 ms	10.4 ms	0.05 ms
Detrending	-	8.6 ms	4.6 ms	0.37 ms	1.73 ms
GLM	33 s	16.6 ms	5.8 ms	0.38 ms	1.7 ms
Total time for GLM	125 s	15.51 s	5.04 s	0.43 s	7.96 s

Bibliography

[1] Tong, S.; Alessio, A.M. (2010); Noise properties in PSF—based fully—3D PET image reconstruction: an experimental evaluation, *Physics in Medicine and Biology*, 55: 1453—1473.

[2] Chen, C.M.; Lee, S.Y. (1990); A parallel implementation of 3—D CT Image reconstruction on hypercube multiprocessor, *IEEE Transactions on Nuclear Science*, 37(3): 1333-1346, DOI: 10.1109/23.57385.

[3] Nishimoto, S.; Vu, A.T.; Naselaris, Th.; Benjamini, Y.; Yu, B.; Gallant, J.L. (2011); Reconstructing Visual Experiences from Brain Activity Evoked by Natural Movies, *Current Biology 21*, 1641—1646.

[4] Holland, D.; Liu, J.; Song, C.; Mazerolle, X. et al. (2013); Compressed sensing reconstruction improves sensitivity of variable density spiral fMRI, *Magnetic Resonance in Medicine*, 70.

[5] Lindquist, M.A. (2008); The Statistical Analysis of fMRI Data, *Statistical Science*, 23(4): 439—464.

[6] Cohen, M.S. (2001); Real—Time Functional Magnetic Resonance Imaging, *Methods*, 25(2): 201—220.

[7] Bernstein, M.A.; King, K.F.; Zhou, X.J. (2004); Handbook of MRI Pulse Sequences, *Elsevier Academic Press.*

[8] http://v04.pymvpa.org/examples.html

[9] Maliţa, M.; Ştefan, Gh. M. (2010); Many-processors & KLEENE's model, *UPB Scientific Bulletin Series C*, 72.

[10] Ştefan, Gh. M. (2010); Integral Parallel Architecture In System—On—Chip Designs, *The 6th International Workshop on Unique Chips and Systems*, Atlanta, USA, 23—26. http://www.arh.pub.ro/gstefan/2010ucas.pdf

[11] Mîţu, B. (2008); C Language Extension for Parallel Processing. http://arh.pub.ro/gstefan/VectorC.ppt

[12] Cooley, J.W.; Tuckey, J.W. (1965); An Algorithm for the Machine Calculation of Complex Fourier Series. Math. Computation, *JSTOR Mathematic of Computation*, 19(90):297-309.

[13] Ţugui, A. (2012); FFT Parallel Implementation for MRI Image Reconstruction, *U.P.B. Scientific Bulletin Series C*, 74: 229-244.

[14] Eklund, A.; Anderson, M.; Knutsson, H. (2012); fMRI Analysis on the GPU Possibilities and Challenges, *Computer Methods and Programs in Biomedicine*, 145—161.

[15] Ţugui, A. (2013); Fixed—point real time MRI reconstruction using Connex Array, *Proceedings of the Romanian Academy Series A*, 14(3): 255—258. http://www.acad.ro/sectii2002/proceedings/doc2013-3/11-Tugui.pdf

[16] Eklund, A.; Andresson, M.; Knutsson, H. (2010); Phase based volume registration using CUDA, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Dallas, USA, 658—661.

[17] Calfa, A.M.; Ştefan, Gh. M. (2010); Matrix Computation on Connex Parallel Architecture, *ICES 2010 —The International Conference on Signals and Electronic Systems*, Gliwice, Poland, 375—378.

[18] Friman, O.; Borga, M.; Lundberg, P.; Knutsson, H. (2004); Detection and detrending in fMRI data analysis, *NeuroImage*, 22(2): 645—655.

[19] Tanabe, J.; Miller, D.; Tregellas, J.; Freedman, R.; Meyer, F.G. (2002); Comparison of Detrending Methods for Optimal fMRI Preprocessing, *NeuroImage*, 15(4): 902—907.

[20] Poldrack, R.H.; Mumford, J.A. (2011); Handbook of Functional MRI Data Analysis, *Cambridge University Press*, New York, USA.