cudaBayesreg: Bayesian Computation in CUDA

by Adelino Ferreira da Silva

Abstract Graphical processing units are rapidly gaining maturity as powerful general parallel computing devices. The package **cudaBayesreg** uses GPU–oriented procedures to improve the performance of Bayesian computations. The paper motivates the need for devising high-performance computing strategies in the context of fMRI data analysis. Some features of the package for Bayesian analysis of brain fMRI data are illustrated. Comparative computing performance figures between sequential and parallel implementations are presented as well.

A functional magnetic resonance imaging (fMRI) data set consists of time series of volume data in 4D space. Typically, volumes are collected as slices of 64 x 64 voxels. The most commonly used functional imaging technique relies on the blood oxygenation level dependent (BOLD) phenomenon (Sardy, 2007). By analyzing the information provided by the BOLD signals in 4D space, it is possible to make inferences about activation patterns in the human brain. The statistical analysis of fMRI experiments usually involve the formation and assessment of a statistic image, commonly referred to as a Statistical Parametric Map (SPM). The SPM summarizes a statistic indicating evidence of the underlying neuronal activations for a particular task. The most common approach to SPM computation involves a univariate analysis of the time series associated with each voxel. Univariate analysis techniques can be described within the framework of the general linear model (GLM) (Sardy, 2007). The GLM procedure used in fMRI data analysis is often said to be "massively univariate", since data for each voxel are independently fitted with the same model. Bayesian methodologies provide enhanced estimation accuracy (Friston et al., 2002). However, since (non-variational) Bayesian models draw on Markov Chain Monte Carlo (MCMC) simulations, Bayesian estimates involve a heavy computational burden.

The programmable Graphic Processor Unit (GPU) has evolved into a highly parallel processor with tremendous computational power and very high memory bandwidth (NVIDIA Corporation, 2010b). Modern GPUs are built around a scalable array of multithreaded streaming multiprocessors (SMs). Current GPU implementations enable scheduling thousands of concurrently executing threads. The *Compute Unified Device Architecture* (CUDA) (NVIDIA Corporation, 2010b) is a software platform for massively parallel high-performance computing on NVIDIA manycore GPUs. The CUDA programming model follows the standard single-program multiple-data (SPMD) model. CUDA greatly simplifies the task of parallel programming by providing thread management tools that work as extensions of conventional C/C++ constructions. Automatic thread management removes the burden of handling the scheduling of thousands of lightweight threads, and enables straightforward programming of the GPU cores.

The package **cudaBayesreg** (Ferreira da Silva, 2010a) implements a Bayesian multilevel model for the analysis of brain fMRI data in the CUDA environment. The statistical framework in **cudaBayesreg** is built around a Gibbs sampler for multilevel/hierarchical linear models with a normal prior (Ferreira da Silva, 2010c). Multilevel modeling may be regarded as a generalization of regression methods in which regression coefficients are themselves given a model with parameters estimated from data (Gelman, 2006). As in SPM, the Bayesian model fits a linear regression model at each voxel, but uses uses multivariate statistics for parameter estimation at each iteration of the MCMC simulation. The Bayesian model used in **cudaBayesreg** follows a two–stage Bayes prior approach to relate voxel regression equations through correlations between the regression coefficient vectors (Ferreira da Silva, 2010c). This model closely follows the Bayesian multilevel model proposed by Rossi, Allenby and McCulloch (Rossi et al., 2005), and implemented in **bayesm** (Rossi and McCulloch., 2008). This approach overcomes several limitations of the classical SPM methodology. The SPM methodology traditionally used in fMRI has several important limitations, mainly because it relies on classical hypothesis tests and p–values to make statistical inferences in neuroimaging (Friston et al., 2002; Berger and Sellke, 1987; Vul et al., 2009). However, as is often the case with MCMC simulations, the implementation of this Bayesian model in a sequential computer entails significant time complexity. The CUDA implementation of the Bayesian model proposed here has been able to reduce significantly the runtime processing of the MCMC simulations. The main contribution for the increased performance comes from the use of separate threads for fitting the linear regression model at each voxel in parallel.

Bayesian multilevel modeling

We are interested in the following Bayesian multilevel model, which has been analyzed by Rossi

et al. (2005), and has been implemented as `rhierLinearModel` in **bayesm**. Start out with a general linear model, and fit a set of m voxels as,

$$y_i = X_i\beta_i + \epsilon_i, \quad \epsilon_i \overset{iid}{\sim} N\left(0, \sigma_i^2 I_{n_i}\right), \quad i = 1, \ldots, m. \tag{1}$$

In order to tie together the voxels' regression equations, assume that the $\{\beta_i\}$ have a common prior distribution. To build the Bayesian regression model we need to specify a prior on the $\{\beta_i\}$ coefficients, and a prior on the regression error variances $\{\sigma_i^2\}$. Following Ferreira da Silva (2010c), specify a normal regression prior with mean $\Delta' z_i$ for each β,

$$\beta_i = \Delta' z_i + \nu_i, \quad \nu_i \overset{iid}{\sim} N\left(0, V_\beta\right), \tag{2}$$

where z is a vector of n_z elements, representing characteristics of each of the m regression equations.

The prior (2) can be written using the matrix form of the multivariate regression model for k regression coefficients,

$$B = Z\Delta + V \tag{3}$$

where B and V are $m \times k$ matrices, Z is a $m \times n_z$ matrix, Δ is a $n_z \times k$ matrix. Interestingly, the prior (3) assumes the form of a second–stage regression, where each column of Δ has coefficients which describes how the mean of the k regression coefficients varies as a function of the variables in z. In (3), Z assumes the role of a prior design matrix.

The proposed Bayesian model can be written down as a sequence of conditional distributions (Ferreira da Silva, 2010c),

$$\begin{aligned} y_i &\mid X_i, \beta_i, \sigma_i^2 \\ \beta_i &\mid z_i, \Delta, V_\beta \\ \sigma_i^2 &\mid \nu_i, s_i^2 \\ V_\beta &\mid \nu, V \\ \Delta &\mid V_\beta, \bar{\Delta}, A. \end{aligned} \tag{4}$$

Running MCMC simulations on the set of full conditional posterior distributions (4), the full posterior for all the parameters of interest may then be derived.

GPU computation

In this section, we describe some of the main design considerations underlying the code implementation in **cudaBayesreg**, and the options taken for processing fMRI data in parallel. Ideally, the GPU is best suited for computations that can be run on numerous data elements simultaneously in parallel (NVIDIA Corporation, 2010b). Typical textbook applications for the GPU involve arithmetic on large matrices, where the same operation is performed across thousands of elements at the same time. Since the Bayesian model of computation outlined in the previous Section does not fit well in this framework, some design options had to be assumed in order to properly balance optimization, memory constraints, and implementation complexity, while maintaining numerical correctness. Some design requirements for good performance on CUDA are as follows (NVIDIA Corporation, 2010a): (i) the software should use a large number of threads; (ii) different execution paths within the same thread block (warp) should be avoided; (iii) inter-thread communication should be minimized; (iv) data should be kept on the device as long as possible; (v) global memory accesses should be coalesced whenever possible; (vi) the use of shared memory should be preferred to the use of global memory. We detail below how well these requirements have been met in the code implementation. The first requirement is easily met by **cudaBayesreg**. On the one hand, fMRI applications typically deal with thousands of voxels. On the other hand, the package uses three constants which can be modified to suit the available device memory, and the computational power of the GPU. Specifically, `REGDIM` specifies the maximum number of regressions (voxels), `OBSDIM` specifies the maximum length of the time series observations, and `XDIM` specifies the maximum number of regression coefficients. Requirements (ii) and (iii) are satisfied by **cudaBayesreg** as well. Each thread executes the same code independently, and no inter-thread communication is required. Requirement (iv) is optimized in **cudaBayesreg** by using as much constant memory as permitted by the GPU. Data that do not change between MCMC iterations are kept in constant memory. Thus, we reduce expensive memory data transfers between host and device. For instance, the matrix of voxel predictors X (see (1)) is kept in constant memory. Requirement (v) is insufficiently met in **cudaBayesreg**. For maximum performance, memory accesses to global memory must be coalesced. However, different fMRI data sets and parameterizations may generate data structures with highly variable dimensions, thus rendering coalescence difficult to implement in a robust manner. Moreover, GPU devices of *Compute Capability* 1.x impose hard memory coalescing restrictions. Fortunately, GPU devices of *Compute Capability* 2.x are expected to lift some of the most taxing memory coalescing constraints. Finally requirement (vi) is not met by the available code. The current kernel implementation does not use shared memory. The relative complexity of the Bayesian computation performed by each thread compared to the conventional arithmetic load assigned to the thread has prevented us from exploiting shared memory operations. The task assigned to the kernel may be subdivided to reduce kernel complexity. However, this option may easily compromise other optimization requirements, namely thread independence. As detailed in the next paragraph, our option has been to keep the computational design simple, by assigning the whole of the

univariate regression to the kernel.

The computational model has been specified as a grid of thread blocks of dimension 64 in which a separate thread is used for fitting a linear regression model at each voxel in parallel. Maximum efficiency is expected to be achieved when the total number of required threads to execute in parallel equals the number of voxels in the fMRI data set, after appropriate masking has been done. However, this approach typically calls for the parallel execution of several thousands of threads. To keep computational resources low, while maintaining significant high efficiency it is generally preferable to process fMRI data slice-by-slice. In this approach, slices are processed in sequence. Voxels in slices are processed in parallel. Thus, for slices of dimension 64×64, the required number of parallel executing threads does not exceed 4096 at a time. The main computational bottleneck in sequential code comes from the necessity of performing Gibbs sampling using a (temporal) univariate regression model for all voxel time series. We coded this part of the MCMC computation as device code, i.e. a kernel to be executed by the CUDA threads. CUDA threads execute on the GPU device that operates as a coprocessor to the host running the MCMC simulation. Following the proposed Bayesian model (4), each thread implements a Gibbs sampler to draw from the posterior of a univariate regression with a conditionally conjugate prior. The host code is responsible for controlling the MCMC simulation. At each iteration, the threads perform one Gibbs iteration for all voxels in parallel, to draw the threads' estimators for the regression coefficients β_i as specified in (4). In turn, the host, based on the simulated β_i values, draws from the posterior of a multivariate regression model to estimate V_β and Δ. These values are then used to drive the next iteration.

The bulk of the MCMC simulations for Bayesian data analysis is implemented in the kernel (device code). Most currently available RNGs for the GPU tend to be have too high time- and space-complexity for our purposes. Therefore, we implemented and tested three different random number generators (RNGs) in device code, by porting three well-known RNGs to device code. Marsaglia's multicarry RNG (Marsaglia, 2003) follows the R implementation, is the fastest one, and is used by default; Brent's RNG (Brent, 2007) has higher quality but is not-so-fast; Matsumoto's Mersenne Twister (Matsumoto and Nishimura, 1998) is slower than the others. In addition, we had to ensure that different threads receive different random seeds. We generated random seeds for the threads by combining random seeds generated by the host with the threads' unique identification numbers. Random deviates from the normal (Gaussian) distribution and chi-squared distribution had to be implemented in device code as well. Random deviates from the normal distribution were generated using the Box-Muller method. In a similar vein, random deviates from the chi-squared distribution with ν number of degrees of freedom, $\chi^2(\nu)$, were generated from gamma deviates, $\Gamma(\nu/2, 1/2)$, following the method of Marsaglia and Tsang specified in (Press et al., 2007).

The next Sections provide details on how to use **cudaBayesreg** (Ferreira da Silva, 2010b) for fMRI data analysis. Two data sets, which are included and documented in the complementary package **cudaBayesregData** (Ferreira da Silva, 2010b), have been used in testing: the 'fmri' and the 'swrfM' data sets. We performed MCMC simulations on these data sets using three types of code implementations for the Bayesian multilevel model specified before: a (sequential) R-language version, a (sequential) C-language version, and a CUDA implementation. Comparative runtimes for 3000 iterations in these three situations, for the data sets 'fmri' and 'swrfM', are as follows.

```
Runtimes in  seconds for 3000 iterations:


      slice R-code C-code CUDA
fmri      3   1327    224   22
swrfM    21   2534    309   41
```

Speed-up factors between the sequential versions and the parallel CUDA implementation are summarized next.

```
Comparative speedup factors:


      C-vs-R CUDA-vs-C CUDA-vs-R
fmri     6.0      10.0      60.0
swrfM    8.2       7.5      61.8
```

In these tests, the C-implementation provided, approximately, a $7.6\times$ mean speedup factor relative to the equivalent R implementation. The CUDA implementation provided a $8.8\times$ mean speedup factor relative to the equivalent C implementation. Overall, the CUDA implementation yielded a significant $60\times$ speedup factor. The tests were performed on a notebook equipped with a (low–end) graphics card: a 'GeForce 8400M GS' NVIDIA device. This GPU device has just 2 multiprocessors, *Compute Capability* 1.1, and delivers single–precision performance. The compiler flags used in compiling the source code are detailed in the package's `Makefile`. In particular, the optimization flag `-O3` is set there. It is worth noting that the CUDA implementation in **cudaBayesreg** affords much higher speedups. First, the CUDA implementation may easily be modified to process all voxels of a fMRI volume in parallel, instead of processing data in a slice-by-slice basis. Second, GPUs with 16 multiprocessors and 512 CUDA cores and *Compute Capability* 2.0 are now available.

Experiments using the fmri test dataset

The data set 'fmri.nii.gz' is available from the FMRIB/FSL site (www.fmrib.ox.ac.uk/fsl). This data set is from an auditory–visual experiment. Auditory stimulation was applied as an alternating "boxcar" with 45*s*-on-45*s*-off and visual stimulation was applied as an alternating "boxcar" with 30*s*-on-30*s*-off. The data set includes just 45 time points and 5 slices from the original 4D data. The file 'fmri_filtered_func_data.nii' included in **cudaBayesregData** was obtained from 'fmri.nii.gz' by applying FSL/FEAT pre-preprocessing tools. For input/output of NIFTI formatted fMRI data sets **cudaBayesreg** depends on the **R** package **oro.nifti** (Whitcher et al., 2010). The following code runs the MCMC simulation for slice 3 of the `fmri` dataset, and saves the result.

```
> require("cudaBayesreg")
> slicedata <- read.fmrislice(fbase = "fmri",
+     slice = 3, swap = TRUE)
> ymaskdata <- premask(slicedata)
> fsave <- "/tmp/simultest1.sav"
> out <- cudaMultireg.slice(slicedata,
+     ymaskdata, R = 3000, keep = 5,
+     nu.e = 3, fsave = fsave, zprior = FALSE)
```

We may extract the posterior probability (PPM) images for the visual (`vreg=2`) and auditory (`vreg=4`) stimulation as follows (see Figures 1 and 2).

```
> post.ppm(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 2,
+     col = heat.colors(256))
```

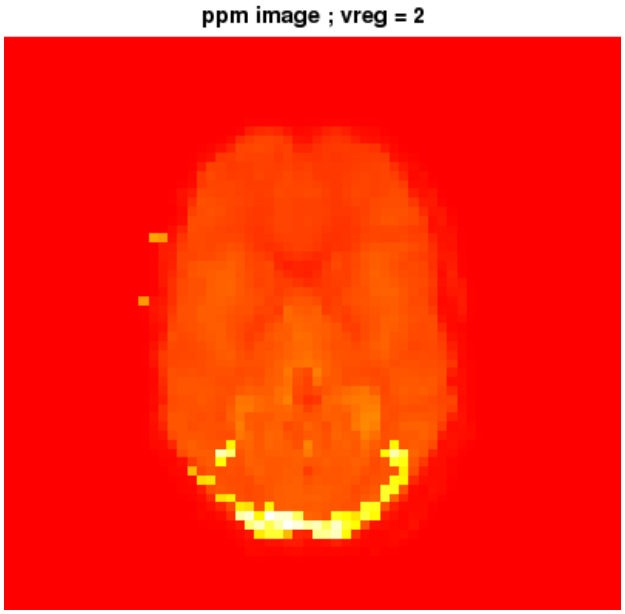

Figure 1: PPM images for the visual stimulation

```
> post.ppm(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 4,
+     col = heat.colors(256))
```

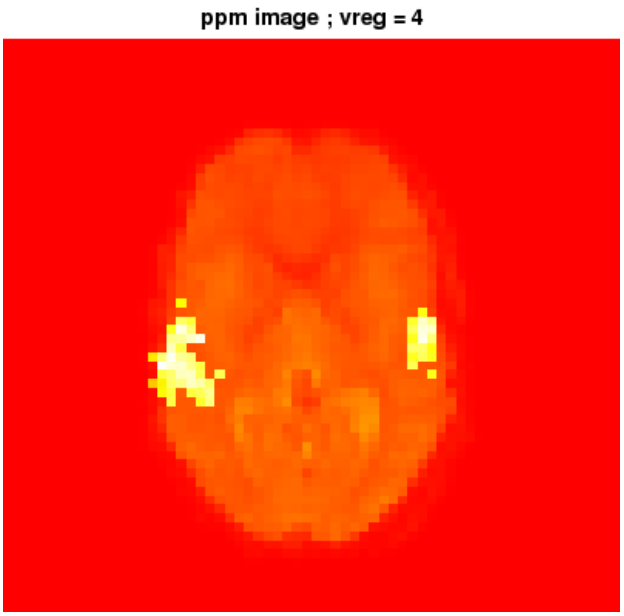

Figure 2: PPM images for the auditory stimulation

To show the fitted time series for a (random) active voxel, as depicted in Figure 3, we use the code:

```
> post.tseries(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 2)

range pm2: -1.409497 1.661774
```

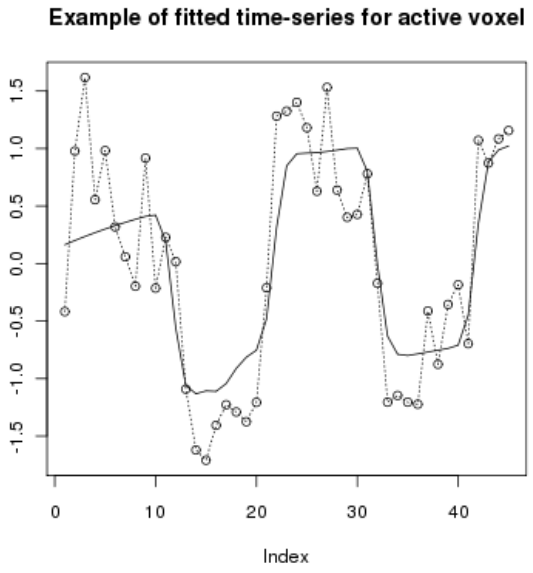

Figure 3: Fitted time–series for an active voxel

Summary statistics for the posterior mean values of regression coefficient `vreg=2`, are presented next. The same function plots the histogram of the posterior distribution for `vreg=2`, as represented in Figure 4.

```
> post.simul.hist(out = out, vreg = 2)
```

```
Call:
        density.default(x = pm2)

Data: pm2 (1525 obs.);        Bandwidth 'bw' = 0.07947

       x                 y
 Min.   :-1.6479   Min.   :0.0000372
 1st Qu.:-0.7609   1st Qu.:0.0324416
 Median : 0.1261   Median :0.1057555
 Mean   : 0.1261   Mean   :0.2815673
 3rd Qu.: 1.0132   3rd Qu.:0.4666134
 Max.   : 1.9002   Max.   :1.0588599
[1] "active range:"
[1] 0.9286707 1.6617739
[1] "non-active range:"
[1] -1.4094965  0.9218208
hpd (95%)=               -0.9300451 0.9286707
```

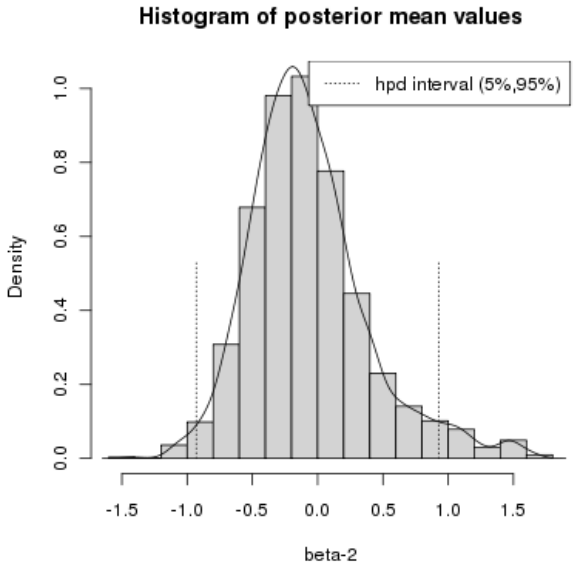

Figure 4: Histogram of the posterior distribution of the regression coefficient β_2 (slice 3).

An important feature of the Bayesian model used in **cudaBayesreg** is the shrinkage induced by the hyperprior ν on the estimated parameters. We may assess the adaptive shrinkage properties of the Bayesian multilevel model for two different values of ν as follows.

```
> nu2 <- 45
> fsave2 <- "/tmp/simultest2.sav"
> out2 <- cudaMultireg.slice(slicedata,
+     ymaskdata, R = 3000, keep = 5,
+     nu.e = nu2, fsave = fsave2,
+     zprior = F)
> vreg <- 2
> x1 <- post.shrinkage.mean(out = out,
+     slicedata$X, vreg = vreg,
+     plot = F)
> x2 <- post.shrinkage.mean(out = out2,
+     slicedata$X, vreg = vreg,
+     plot = F)
> par(mfrow = c(1, 2), mar = c(4,
+     4, 1, 1) + 0.1)
> xlim = range(c(x1$beta, x2$beta))
> ylim = range(c(x1$yrecmean, x2$yrecmean))
> plot(x1$beta, x1$yrecmean, type = "p",
+     pch = "+", col = "violet",
+     ylim = ylim, xlim = xlim,
+     xlab = expression(beta), ylab = "y")
> legend("topright", expression(paste(nu,
+     "=3")), bg = "seashell")
> plot(x2$beta, x2$yrecmean, type = "p",
+     pch = "+", col = "blue", ylim = ylim,
+     xlim = xlim, xlab = expression(beta),
+     ylab = "y")
> legend("topright", expression(paste(nu,
+     "=45")), bg = "seashell")
> par(mfrow = c(1, 1))
```

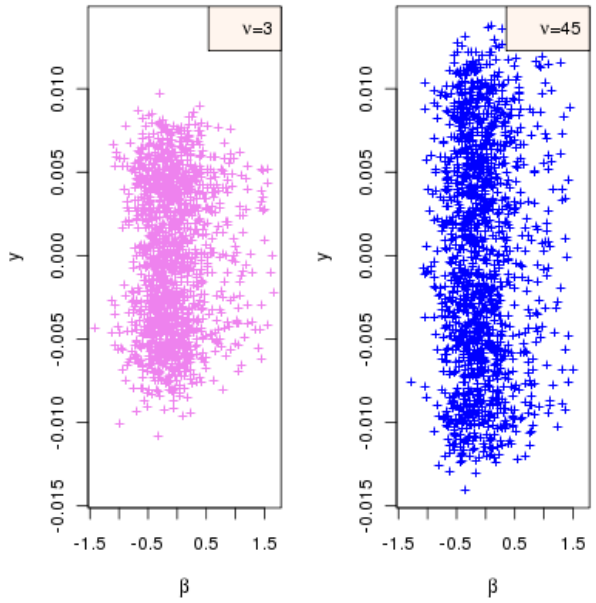

Figure 5: Shrinkage assessment: variability of mean predictive values for $\nu = 3$ and $\nu = 45$.

Experiments using the SPM auditory dataset

In this Section, we exemplify the analysis of the random effects distribution Δ, following the specification of cross-sectional units (group information) in the Z matrix of the statistical model. The Bayesian multilevel statistical model allows for the analysis of random effects through the specification of the Z matrix for the prior in (2). The dataset with prefix `swrfM` (argument `fbase="swrfM"`) in the package's data directory, include mask files associated with the partition of the fMRI dataset 'swrfM' in 3 classes: cerebrospinal fluid (CSF), gray matter (GRY) and white matter (WHT). As before, we begin by loading the data and running the simulation. This time, however, we call `cudaMultireg.slice` with the argument `zprior=TRUE`. This argument will launch `read.Zsegslice`, that reads the segmented images (CSF/GRY/WHT) to build the Z matrix.

```
> fbase <- "swrfM"
> slice <- 21
> slicedata <- read.fmrislice(fbase = fbase,
+     slice = slice, swap = TRUE)
> ymaskdata <- premask(slicedata)
> fsave3 <- "/tmp/simultest3.sav"
> out <- cudaMultireg.slice(slicedata,
+     ymaskdata, R = 3000, keep = 5,
+     nu.e = 3, fsave = fsave3,
+     zprior = TRUE)
```

We confirm that areas of auditory activation have been effectively selected by displaying the PPM image for regression variable `vreg=2`.

```
> post.ppm(out = out, slicedata = slicedata,
+     ymaskdata = ymaskdata, vreg = 2,
+     col = heat.colors(256))
```

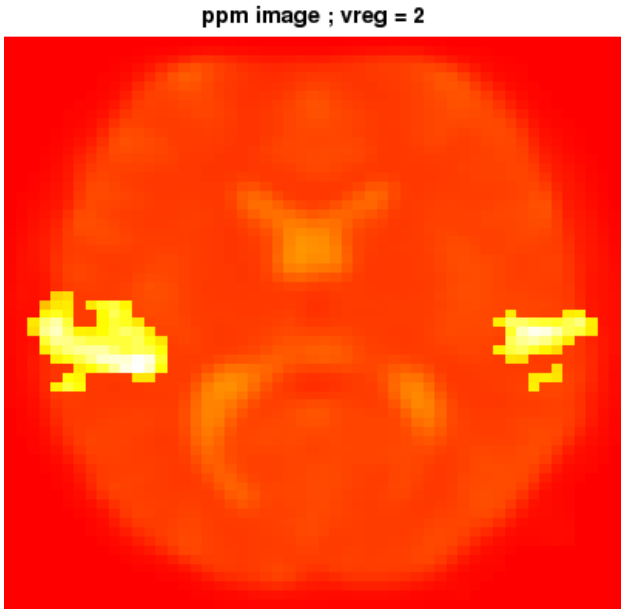

Figure 6: PPM images for the auditory stimulation

Plots of the draws of the mean of the random effects distribution are presented in Figure 7.

```
> post.randeff(out)
```

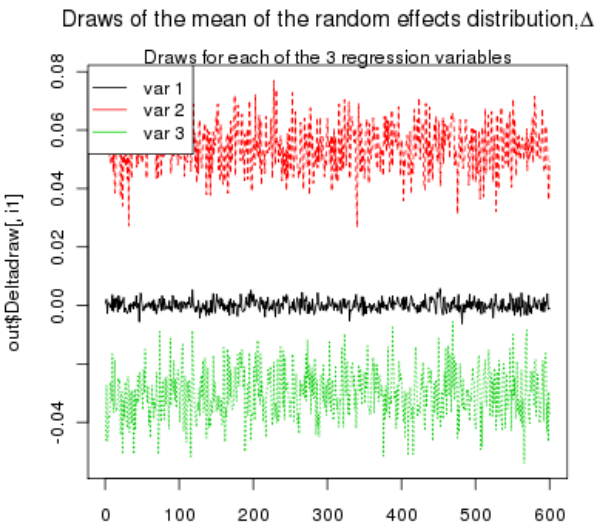

Figure 7: Draws of the mean of the random effects distribution

Random effects plots for each of the 3 classes are obtained by calling,

```
> post.randeff(out, classnames = c("CSF",
+     "GRY", "WHT"))
```

Plots of the random effects associated with the 3 classes are depicted in Figures 8–10.

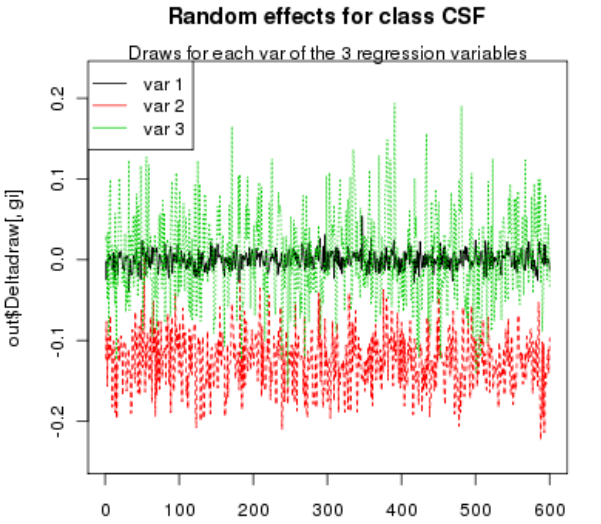

Figure 8: Draws of the random effects distribution for class CSF

Figure 9: Draws of the random effects distribution for class GRY

```
> post.randeff(out, classnames = c("CSF",
+     "GRY", "WHT"))
```

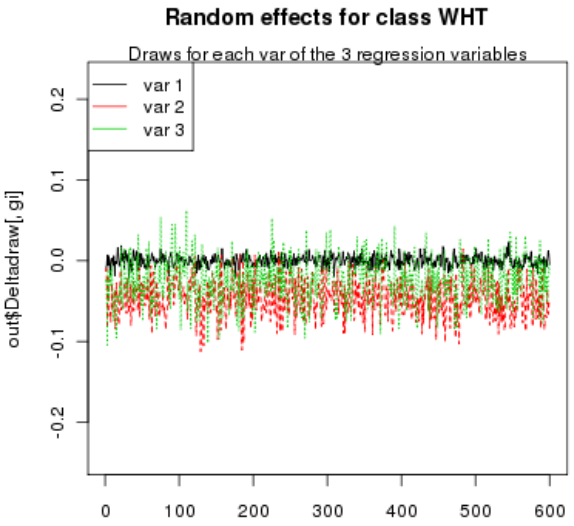

Figure 10: Draws of the random effects distribution for class WHT

Conclusion

The CUDA implementation of the Bayesian model in **cudaBayesreg** has been able to reduce significantly the runtime processing of MCMC simulations. For the applications described in this paper, we have obtained speedup factors of 60× compared to the equivalent sequential R code. The results point out the enormous potential of parallel high-performance computing strategies on manycore GPUs.

Bibliography

J. Berger and T. Sellke. Testing a point null hypothesis: the irreconciability of p values and evidence. *Journal of the American Statistical Association*, 82: 112–122, 1987.

R. P. Brent. Some long-period random number generators using shifts and xors. *ANZIAM Journal*, 48 (CTAC2006):C188–C202, 2007. URL http://wwwmaths.anu.edu.au/~brent.

A. R. Ferreira da Silva. *cudaBayesreg: CUDA Parallel Implementation of a Bayesian Multilevel Model for fMRI Data Analysis*, 2010a. URL http://CRAN.R-project.org/package=cudaBayesreg. R package version 0.3-8.

A. R. Ferreira da Silva. *cudaBayesregData: Data sets for the examples used in the package cudaBayesreg*, 2010b. URL http://CRAN.R-project.org/package=cudaBayesreg. R package version 0.3-8.

A. R. Ferreira da Silva. A Bayesian multilevel model for fMRI data analysis. *Comput. Methods Programs Biomed.*, in press, 2010c.

K. J. Friston, W. Penny, C. Phillips, S. Kiebel, G. Hinton, and J. Ashburner. Classical and Bayesian Inference in Neuroimaging: Theory. *NeuroImage*, 16: 465– 483, 2002.

A. Gelman. Multilevel (hierarchical) modeling: What it can and cannot do. *Technometrics*, 48(3):432–435, Aug. 2006.

G. Marsaglia. Random number generators. *Journal of Modern Applied Statistical Methods*, 2, May 2003.

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, Jan. 1998.

NVIDIA Corporation. *CUDA C Best Practices Guide Version 3.2*, Aug. 2010a. URL http://www.nvidia.com/CUDA.

NVIDIA Corporation. *NVIDIA CUDA Programming Guide, Version 3.2*, Aug. 2010b. URL http://www.nvidia.com/CUDA.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes, The Art of Scientific Computing*. CUP, third edition, 2007.

P. Rossi and R. McCulloch. *bayesm: Bayesian Inference for Marketing/Micro-econometrics*, 2008. URL http://faculty.chicagogsb.edu/peter.rossi/research/bsm.html. R package version 2.2-2.

P. E. Rossi, G. Allenby, and R. McCulloch. *Bayesian Statistics and Marketing*. John Wiley and Sons, 2005.

G. E. Sardy. *Computing Brain Activity Maps from fMRI Time-Series Images*. Cambridge University Press, 2007.

E. Vul, C. Harris, P. Winkielman, and H. Pashler. Puzzlingly high correlations in fMRI studies of emotion, personality, and social cognition. *Perspectives on Psychological Science*, 4(3):274–290, 2009.

B. Whitcher, V. Schmid, and A. Thornton. *oro.nifti: Rigorous - NIfTI Input / Output*, 2010. URL http://CRAN.R-project.org/package=oro.nifti. R Package Version 0.1.4.

Adelino Ferreira da Silva
Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Portugal
afs@fct.unl.pt