Energy-aware acceleration on GPUs: Findings on a bioinformatics benchmark

J. Pérez[(1)], A. Rodríguez[(1)], J.F. Chico[(2)], D. López-Rodríguez[(2)], M. Ujaldón[(1)]

[(1)] *Computer Architecture Department, University of Malaga (Spain)*
[(2)] *Brain Dynamics, Technology Park of Andalusia (Spain)*

Abstract

This paper performs a complete study on performance and energy efficiency of biomedical codes when accelerated on GPUs (Graphics Processing Units). We have selected a benchmark composed of three different building blocks which constitute the pillars of four popular biomedical applications: Q-norm, for the quantile normalization of gene expressions, reg_f3d, for the registration of 3D images within the NiftyReg library, bedpostx (from the FSL neuroimaging package) and a multi-tensor tractography for the analysis of diffusion images. We try to identify (1) potential scenarios where performance per watt can be optimal in large-scale biomedical applications, and (2) the ideal GPU platform among a wide range of models, including low power Tegras, popular GeForces and high-end Titans. Experimental results conclude that data locality and arithmetic intensity represent the most rewarding ways on the road to high performance bioinformatics when power is a major concern.

Keywords: GPU, HPC, power-aware, speed-up, Q-norm, NiftyReg, FDT

1. Introduction

Bioinformatics constitutes nowadays one of the most prolific scientific areas where GPU acceleration has been deployed. Aplications of this class are often characterized as large-scale due to its huge computational workload. Most of the times, its methods are memory-intensive and access very large data structures, thus providing representative instances of the new big-data era. Together with the SIMD (Single Instruction Multiple Data) parallelism

led by the GPU over the last decade, the binomial has turned out to be very effective for High Performance Computing (HPC), particularly after the invention of CUDA (Compute Unified Device Architecture) in November, 2006, as paradigm for general-purpose computing using graphics processors [1].

Over this avantgarde era of computing, the HPC community has analyzed a wide range of applications of virtually every computational field, identifying pros and cons to be efficiently mapped onto GPU devices. The ubiquitous presence of GPU-equipped supercomputers within the Top500.org [2] has propelled the interest for scientists to benefit from this technology and now it is considered a very valuable investment when it comes to reduce dramatically the computational time.

The latest generations of Nvidia GPUs, namely Maxwell (2014), Pascal (2016) and Volta (2017), have introduced a new issue as long as GPU performance is evaluated in modern times: Power consumption. Now, we do not only care about acceleration factors, but also about energy consumed to complete the process [3, 4, 5], particularly after GPUs have conquered massively the Green500.org list [6]. The new performance metric is no longer GFLOPS (Giga Floating-Point Operations Per Second), but GFLOPS/w (GFLOPS per watt). This paper emphasizes this metric from a biomedical perspective, analyzing four popular methods in assorted subareas:

- **Q-norm** [7], a quantile-based normalization method for high density oligonucleotide array data based on variance and bias.
- **NiftyReg** [8], a Neuroimaging library to compute the rigid, affine and non-linear registration of magnetic resonance images (MRIs).
- **FDT** [9], part of the Neuroimaging package FSL [10], to fit a probabilistic diffusion model at each voxel of diffusion weighted images.
- **Multi-Tensor Estimation** [11], considered as the deterministic counterpart of FDT, to fit three diffusion tensors at each point of the images.

This collection of methods provides a rich umbrella for discussing data parallelism, arithmetic intensity and power efficiency in high performance computing.

The rest of the paper is organized as follows. Section 2 introduces the biomedical methods involved along this paper. Section 3 briefly outlines the

infrastructure we have deployed for measuring energy on GPUs. Section 4 describe the testbed used in Section 5 along our experimental results. Finally, Section 6 concludes.

2. Methods for our biomedical benchmark

The set of methods chosen for our energy-aware acceleration study have been carefully chosen from different sources with the aim of complementing each other in computational features, assembling a puzzle where we analyze assorted issues related to performance and energy. In the following subsections we provide a short description of this benchmark.

2.1. Quantile normalization: Q-norm

Our first method is a high performance implementation of Q-norm [7], an increasingly popular algorithm for a fast and easy to understand normalization of multiple gene-expression datasets under the assumption of sharing a common distribution. In the era of big data, we often run genetic experiments that involve multiple high density oligonucleotide arrays where sources of variation between samples of non-biological origin have to be cleaned up. The normalization process helps to minimize this variation.

The high computational cost and memory requirements of the Q-norm method are derived from its huge input source, typically a matrix X composed of $p > 6$ millions of gene expression values and $N > 1000$ samples on a regular basis [12]. N spreads over matrix columns and p does it along rows, where a single matrix element, X[i,j], indicates the intensity for the i-th gene expression values into the j-th sample. Values of X are positive integers which are extracted by high density oligonucleotide microarray technology provided by the Affymetrix GeneChip infrastructure [13] widely used in many areas of biomedical research. Those integers are the target numbers to normalize, usually by means of some kind of average for every array element placed in the same quantile [14]. Oligonucleotides of 25 base pairs are used to probe genes, with each probe pair interrogating a different part of the sequence for a gene in what is known as a probeset [15].

In our case study, we consider $N = 470$ samples, each composed of $p = 6.553.600$ gene expression values. The input dataset was taken from the GEO (Gene Expression Omnibus) Web repository [16] as submitted by Affymetrix under the GeneChip Human Mapping 500K Array Set (platform GPL3718).

The ultimate goal of Q-norm is to equalize the distribution of probe intensities for each array in a set of samples [7], under the assumption that there is an underlying common distribution for all those samples. A QQplot tool can be used to compare if two datasets come from the same distribution by checking that the quantiles line up on the diagonal. This suggests that one could give two disparate datasets the same distribution by transforming the quantiles of each dataset to have the same value, which will be their average value. Extending this idea to N dimensions we generate a method for finding a common distribution from multiple data vectors.

The procedure for quantile normalization can be summarized as follows: Let $q_k = (q_{k_1}, ..., q_{k_N})$ for $k = 1, ..., p$ be the vector of the k-th quantiles for all N array samples of length p which compose matrix X of dimensions $p \times N$ where each sample is a column. Then:

1. Sort each column of X to give X_{sort}.
2. Take the means across rows of X_{sort} and assign this mean to each element in the row to get X'_{sort}.
3. Produce X_{norm} as output by rearranging each column of X'_{sort} to have the same ordering as original X.

This method forces the values of quantiles to be equal, which may cause problems in the tails where it is possible for a probe to have the same value across all the array samples. However, this case is unrealistic since probeset expression measures are typically computed using the value of multiple probes.

2.2. Non-rigid registration in Neuroimaging: reg_f3d within NiftyReg

A variety of neuroimaging technologies allow the structure and function of the intact human brain to be studied with minimal invasion, providing a better understanding of healthy states and damage when clinical surgery is applied. Bioinformatics tools are key at all stages of neuroimaging, allowing scientists to control highly sophisticated imaging instruments and to make sense of the vast amounts of complex data generated by them.

The remaining methods in our benchmark share this common background, Neuroimaging, although from different perspectives. The first one to be discussed in this work is NiftyReg, a library to perform 3D image registration, which consists in finding a (non-rigid) transformation that maps a given 3D image (also called volume) to a reference image, usually representing a standard coordinate space.

Over the past few years, members of the neuroimaging research community mainly sponsored by the NIH (National Institutes of Health) have developed a number of reliable, accurate and easy to use tools. Among them, we highlight the Neuroimaging Informatics Technology Initiative (NIfTI) [17].

To facilitate inter-operation of (structural and functional) MRI data analysis software packages, the NIfTI Data Format Working Group proposed the new analyze-style NIfTI data format. Thereafter, a number of NIfTI-aware toolkits were born (e.g. FSL, AFNI, SPM, Freesurfer), with `dicomnifti` allowing the conversion from DICOM images into the NIfTI format for Linux users [18], among others. Later, `niftilib` provided a reference implementation of a C library to read, write and manipulate NIfTI images, and similarly, counterpart versions written in Java and Matlab were created too. The source code for those libraries was put into the public domain, and corresponding projects were hosted at SourceForge [19].

Once all those basic pillars consolidated, scientists started to develop libraries based on NIfTI format for many different purposes. Among them, `NiftyReg` [8] emerged as a a medical image registration library mostly used for brain analysis. It was developed by Marc Modat et al. at UCL (University College London), along with its parent project, NifTK [20].

Its main focus is rigid, affine and non-linear registration (see Fig. 1), and it was fully implemented on CPU using C++, and when CUDA was born, it introduced a GPU-based implementation for Tesla (2008) and Fermi (2010) Nvidia generations [22, 23]. In this paper, we have updated the CUDA implementation to take advantage of Kepler (2012) and Maxwell (2015) GPUs, together with our energy-aware study, for a much richer and newer analysis.

Methods in this library can also be executed from a standalone console application, with a flag to determine whether the CPU or GPU-based version will be used.

Based on this software infrastructure, we take as departure point the execution of the `reg_f3d` method (the library core) to compute the non-rigid registration for a set of 20 volumes obtained from a MRI device, where each volume is composed of $512 \times 512 \times 53$ voxels (a voxel is the 3D analog of a digital pixel). The algorithm produces as output structural images of type 1 in clinical practice, with the corresponding 3D volume of a human brain using MRI images.

Figure 2 shows the sequence of functions that `reg_f3d` requires to be completed, and Table 1 summarizes the functionality of those functions, which have been transformed into optimized CUDA kernels as described in [24].

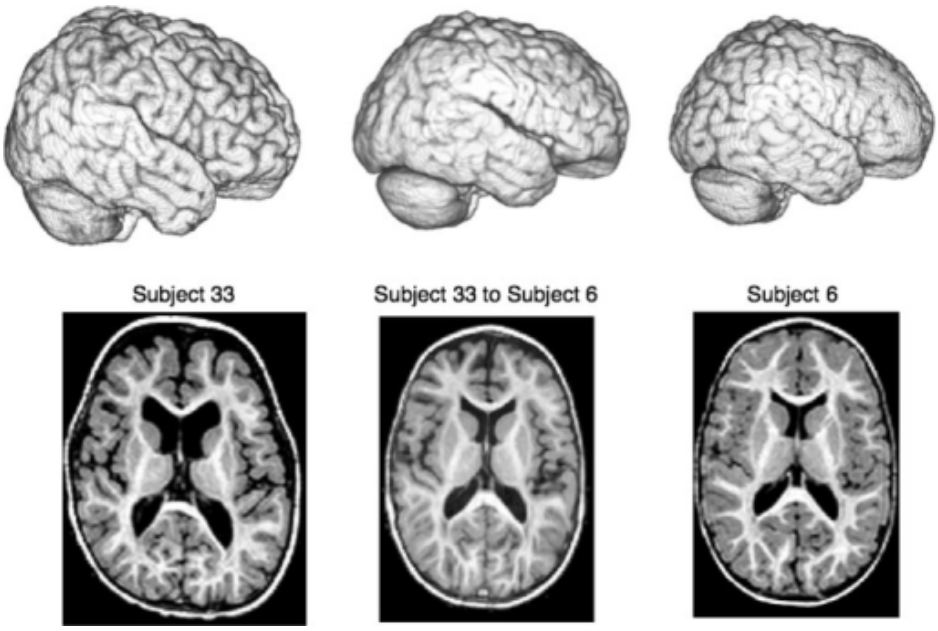


Figure 1: Example of image registration: (Left) Original image of a given subject used as source. (Right) Reference image of another subject. (Center) Non-linear transformation of the source image, overlapping the reference image almost perfectly. Image from [21].

2.3. Diffusion Neuroimaging with Orientation Distribution calculation: bedpostx within FDT

Diffusion weighted imaging (DWI) is the tool used by neuroscientists to measure water diffusion inside brain fibers. In this type of image, the contrast and brightness are highly related to the strength of water diffusion in each point. The aim of DWI processing is to relate the brain image to an underlying mathematical model of fiber orientation [25].

To this end, at each image voxel, diffusion is measured along a set of different possible orientations, also called gradients. These gradients $g_1, \ldots, g_N \in \mathbb{R}^3$ are confined to the sphere $\mathbb{S}^2$, that is, all gradients are of norm equal to 1, being N a constant preset in the MRI acquisition.

Given a specified image voxel, for each $i = 1, \ldots, N$, a voxel value $s_i \in \mathbb{R}$ is obtained by using the corresponding gradient g_i. Also, a baseline (without the application of a gradient) signal intensity s_0 is acquired. There exists a mathematical formulation relating s_i and the baseline s_0, for voxels in which

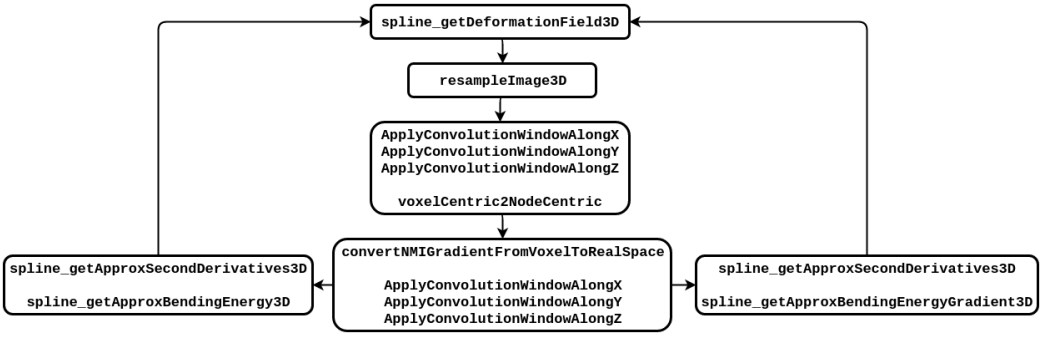


Figure 2: Sequence of CUDA kernels involved during the execution of the `reg_f3d` method within NiftyReg library.

only one diffusion direction is estimated:

$$s_i = s_0 e^{-b g_i^T D g_i} \tag{1}$$

for all i, where b is an acquisition-specific constant and D is the diffusion tensor in that voxel, defined by a symmetric positive-definite 3×3 matrix that can be characterized by 6 parameters (corresponding to the terms in the upper triangular part of the matrix).

The diffusion tensor can be visualized as in Fig. 3 (left), as two juxtaposed ellipsoids pointing in the predominant diffusion direction.

Once determined the diffusion tensor in a voxel, computing its main eigenvector provides us with the predominant direction of water diffusion, aligned with neural fibers, which allows us to reconstruct brain pathways. This process is called tractography [26].

However, there are situations, such as fiber crossing, branching or *kissing*, where there is not a single predominant direction of water diffusion.

There are two main approaches to solve these problems. The first is to estimate the probability function of the possible orientation of the neural fibers within a voxel [27, 28, 29, 30, 31], instead of assuming one deterministic direction. In this sense, tractography is performed by sampling this probability function (called ODF, *Orientation Distribution Function*) to obtain multiple orientations and reconstruct a high number of fibers within each voxel. Examples of ODF with 1 and 3 predominant directions are given in Fig. 3 (center right) and (right).

The second approach, to be discussed in the next subsection, consists in expanding Eq. (1) to allow the contribution of multiple directions of diffusion

Table 1: List of CUDA kernels involved in our implementation of the `reg_f3d` method within NiftyReg library (as outlined in Figure 2). They are listed from more to less weight in the overall execution time of the `reg_f3d` method (see percentage between parenthesis).

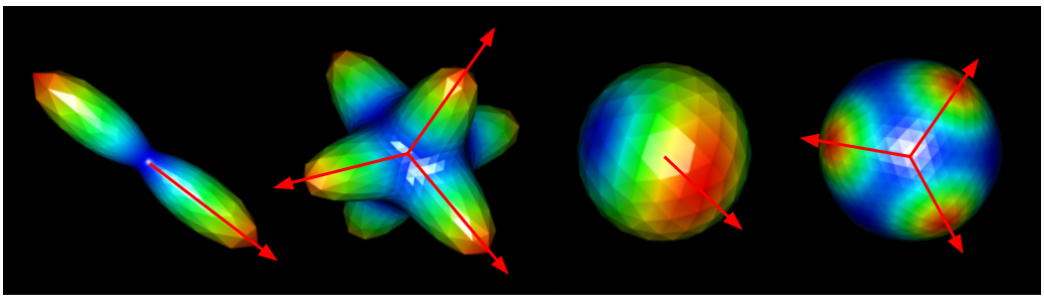
Kernel (computational weight)	**Description**
spline_getDeformationField3D (35,93%)	Cubic B-splines interpolation to deform the image locally.
spline_getApproxSecondDerivs3D (20,00%)	Compute all the second derivatives.
resampleImage3D (15,80%)	Resampling of the floating image.
spline_getApproxBendingEnergy3D (7,97%)	Compute the bending energy from the second derivatives.
ApplyConvolutionWindowAlongZ (6,13%)	NMI gradient field smoothing (Z axis).
ApplyConvolutionWindowAlongY (5,57%)	NMI gradient field smoothing (Y axis).
ApplyConvolutionWindowAlongX (5,26%)	NMI gradient field smoothing (X axis).
spline_getApprox-BendingEnergyGr3D (1,53%)	Compute the gradient from the second derivatives.
convertNMIGr-FromVoxelToRealSpace (1,05%)	The similarity measure gradient is converted from voxel to real space.
voxelCentric2NodeCentric (0,76%)	From the voxel-centric gradient values, it extracts the analytical node-centric derivative of the similarity measure.

in the interior of the brain. A 3-tensor model is visually represented in Fig. 3 (center left).

One of the most popular methods in DWI processing using the first approach is FDT (FMRIB's Diffusion Toolbox) [9]. FDT is a software tool for analysis of DWI which is part of the FSL (FMRIB's Software Library), a complete package of analysis tools for medical images (fMRI, MRI and DTI) developed by FMRIB (University of Oxford).

FDT includes tools for preprocessing DWI data, local modelling of diffusion parameters and tractography. Each stage in FDT is run separately and can be accessible from an easy-to-use graphical user interface. As outlined in Figure 4, FDT provides mainly a set of four methods:

- `eddycorrect`, for correction of eddy current distortion.
- `bedpostx`, for local modelling of diffusion parameters.
- `probtrackx`, for tractography and connectivity-based segmentation.
- `dtifit`, for fitting a diffusion tensor model at each voxel.



(a) (b) (c) (d)

Figure 3: (a) One ellipsoid representing a single tensor as described in the text. (b) Extended formulation with three tensors depicting the main diffusion directions inside a voxel. (c) ODF with a single predominant direction estimated. (d) ODF with 3 different fiber orientations computed. In all cases, red arrows represent the assumed main directions given by the different models. In the ODF cases, red areas on the sphere are the most probable fiber directions.

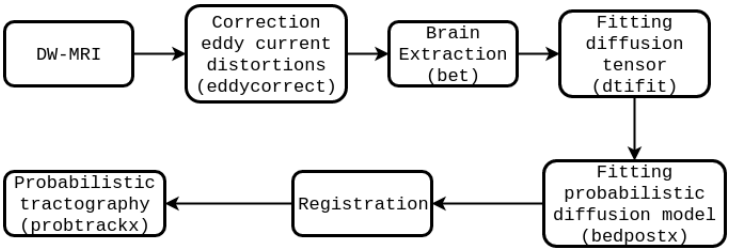


Figure 4: Pipeline of FDT package.

Figure 5: Input and output data for bedpostx and bedpostx_gpu.

The probabilistic tractography tools within FDT [32] are very flexible and allow the user to generate distributions from single or multiple voxels. For an overview of the local diffusion modelling and tractography used within FDT, see [33]. The library is implemented on CPUs using C++, and currently, only `bedpostx` has a CUDA version for GPUs (`bedpostx_gpu`) [34].

`bedpostx` calculates the distribution of diffusion parameters at each voxel, which is the more computational demanding stage for the library. In this technique, each orientation to estimate is coded as (θ_i, ϕ_i) (for all $i \leq M$, being M the maximum number of allowed directions). Thus, only $2 \cdot M$ parameters have to be approximated. In this approximation process, for every voxel, there are two stages: an initial estimation of the parameters with Levenberg-Marquadt and the computation of the distribution of the model parameters through Markov Chain Monte Carlo algorithm.

Additionally, `bedpostx` can estimate crossing fibers. Inputs and outputs for this process are outlined in Fig. 5, whereas Fig. 6 illustrates the set of CUDA kernels involved.

As input data set for `bedpostx`, we have used a female subject, aged 25-29, taken from the MGH HCP Adult Diffusion of the Human Connectome Project [35, 36]. The acquisition matrix was 140x140 with 1.5mm isotropic voxel size and 1.5mm slice thickness (TR=8.8s, TE=57ms, 6/8 Partial Fourier). Overall, 96 slices where processed and the diffusion weighting was applied in $k = 128$ directions with a b-value of 5000 s/mm^2. This original data has been preprocessed using a standard pipeline with FSL's Eddy Current Correction Tool and Brain Extraction Tool to prepare data for `bedpostx`, where power and execution time was averaged over ten launches to collect more stable and unbiased results.

2.4. DWI with multiple tensors calculation

As we anticipated, for voxels containing a mixed diffusion pattern (usually, branching and crossing fibers), more complex formulations can be de-

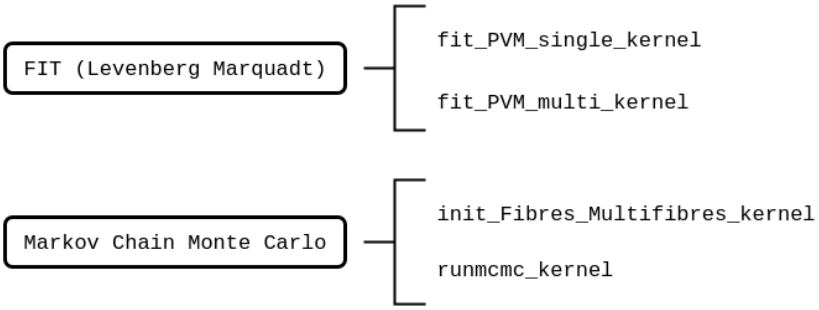


Figure 6: The two main stages executed by bedpostx_gpu, which are splitted into two CUDA kernels each. Each bedpostx_gpu launch comprises 32 iterations of the first three kernels and 64 iterations of the last one (`runmcmc`).

fined as extensions of Eq. (1), using a mixture of deterministic tensors [37, 38, 39, 40, 11, 41].

Our approach follows the line of [11] and [41] and extends it to 3 different tensors estimated inside each image voxel (such as the depicted in Fig. 3 (center left)) as an alternative to the probabilistic computation of the diffusion direction using the FDT library described in the previous section.

Deterministic tractography involves directly following the diffusion pathways (eigenvectors of the diffusion tensors). Using a single tensor model, we just follow the principal diffusion direction [26]. Multi-fiber and multi-tensor models often include techniques for determining the number of fibers present or selecting the most appropriate diffusion direction when pathways branch [40, 42, 43].

These three methods (calculation of one tensor, three tensors and tractography) are implemented in CUDA (see Figure 7) and constitute the core of the algorithm. Figure 8 outlines input and output data and Figure 9 its workflow. The three tensors version is tailored to the required task and extends the single tensor solution to solve its major drawbacks. A similar input dataset is used for running both approaches (FDT and multi-tensor), with the purpose of analyzing acceleration factors, energy spent and correlation between them to quantify payoffs on each side.

3. Monitoring energy

The system used is based on a Beaglebone Black, an open-source hardware computer [44] combined with the Accelpower module [45], which has

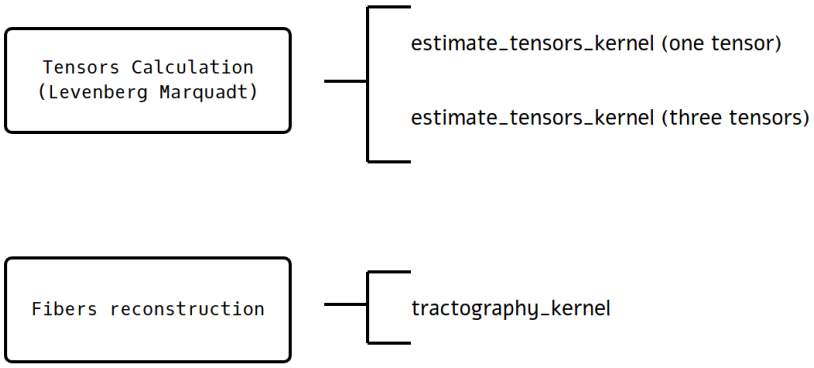


Figure 7: The two main stages executed by the multitensor tractography algorithm and its decomposition into CUDA kernels.

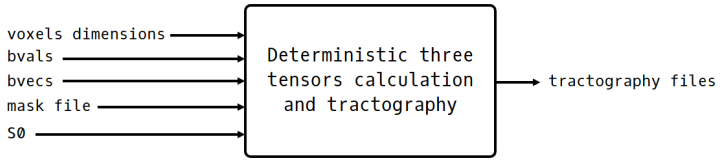


Figure 8: Input and output data for the multitensor tractography algorithm..

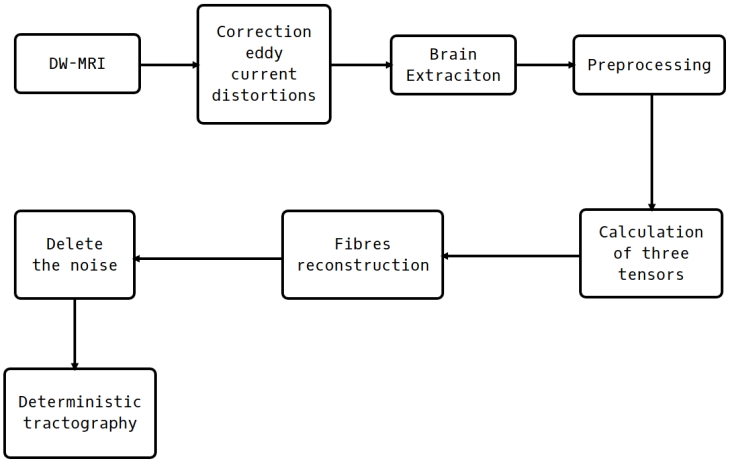


Figure 9: Workflow for the deterministic tractography package.

```
// Declaration of variables to use {\tt pmlib} (server, counter
   and lines).
// Clear all lines and set lines of interest (8 INA219).
// Connect to server (BeagleBone Black).
// Create a counter and start counter

// CODE TO MEASURE

// Stop counter
// Get and save data to a file.
// Finalize counter.
```

Figure 10: Sequence of steps upon kernel launching.

eight INA219 sensors [46] to measure current, voltage and wattage. In order to have a real measure, it can be taken into acccount both power pins in PCI-Express (12 and 3.3 volts) and external 12 volts connectors [47].

Accelpower uses `pmlib` [48], a library for measurement performance. It is formed by two main parts, a server daemon that collects power data from measuring devices and send to the clients; together with a client library for communication and synchronization with server.

The methology for measuring energy begins with server daemon start-up. The next step is modify the code as in Figure 10. Before the kernel launching we need to declare `pmlib` variables, clear and set lines we need (in our case all of them), connect to server (BeagleBone Black with Accelpower cape), create and start a counter. After the code, we stop counter, get and save data (to a .csv file) and at last, finalize counter.

4. Our testbed

This work analyzes the behaviour of biomedical applications on GPUs from a wide variety of issues, namely:

- How energy is affected by GPU acceleration on every biomedical application and GPU kernel. First, a preliminary analysis is performed in Section 5.1 for different software methods, but limiting the hardware platforms involved to the scope of low-power and low-end devices (see Tables 3 and 5). Second, an illustrative comparison with a popular domestic GeForce model, the GeForce GTX 980 GPU, is included in Section 5.2 for our registration method, `reg_f3d` (see central columns in

Table 3). Third, a breakout of `bedpostx` into kernels will be performed on a variety of mid-end and high-end GPUs in Section 5.4 (see Tables 7, 8, 9 and 10) to later establish a winner GPU from two perspectives: performance and power efficiency (see Tables 11, 12 and 13). Fourth, a similar breakout of the multitensor approach will tell us what the contributions of the new Titan X2 GPU are (final paragraph of Section 5.5 - Table 18).

- How the input/output for the data volume to the GPU influences its execution time and acceleration. Here we will play with the quantile normalization method, `Q-norm`, in Section 5.3 (see Table 6).

- How energy is affected when we reformulate a biomedical problem to provide more accuracy, output data and/or better analysis. Section 5.5 tackles this using our latest GPUs, the two Titan models, and the more sophisticated software application: modelling of diffusion images, either through probabilistic and deterministic approaches (see Tables 14, 15, 16 and 17).

Our testbed is clearly ambitious, but we do not want this completeness goal to lead to an excessive volume of experimental numbers, so we have carefully selected our empirical setup to make it concise but rich at the same time. This balance is promoted by discarding some results and summarizing others. For example, our input data set is simple: We do not vary individuals, number of images, resolutions, formats, etcetera. And even though we have available a wide variety of GPUs (see Table 2), only those better reflecting every analysis were chosen for each experiment.

With the exception of our Tegra model, which is a System-on-Chip (SoC), our set of GPUs were successively plugged on PCI-express 3 slots belonging to an EVGA motherboard endowed with and Intel Sandy Bridge quad-core processor running at 2.13 GHz and 16 Gbytes of DDR3 memory running at 1600 MHz. Note that the features of our CPU platform do not play a significant role on performance and measurements, because we monitor energy, time and power once CUDA kernels are shifted to the graphics card and the GPU starts computing.

Table 2: Characterization of the GPU hardware used along our experimental analysis.

	Low-power GPU	Low-end GPU
GPU family	Tegra	GeForce
GPU model	Jetson TK1	GT 640
(and generation)	(Kepler)	(Kepler)
Number of cores	192	384
Core Speed (MHz)	852	900
GFLOPS (peak)	326	691
Memory Size (GB)	2	2
Memory Speed (MHz)	1848	1962
Memory Width (bits)	128	64
" Bandwidth (GB/s)	28,5	14,8
Thermal Design Power (W)	11	65

	Mid-end to high-end GPUs					
GPU model	GTX 480	GTX 680	GTX 780	GTX 980	Titan X1	Titan X2
(and generation)	(Fermi)	(Kepler)	(Kepler)	(Maxwell)	(Maxwell)	(Pascal)
Number of cores	480	1536	2304	2048	3072	3584
Core Speed (MHz)	1400	1006	954	1216	1000	1405
GFLOPS (peak)	1344	3090	4396	4980	7144	10157
Memory Size (GB)	1,5	2	3	4	12	12
" Speed (MHz)	3696	6000	6000	7000	7000	10000
" Width (bits)	384	256	384	256	384	384
" Bandwidth (GB/s)	177	192	288	224	336	480
Thermal D. Power (W)	250	195	250	165	250	250

5. Experimental results

5.1. Acceleration on energy-efficient GPUs

We start analyzing the execution time and power consumption on low-power devices, to see if they are competitive against low-end GPUs (both on a similar budget around $100) and even with a mid-end GPU ($300-$500), which are much more popular in the marketplace. Low-power GPUs were introduced to maximize power efficiency, as they are oriented to cell phones and PDAs equipped with batteries where autonomy is a major concern for potential customers. Table 3 shows that average power for our low-power GPU, the Jetson TK1 model from Nvidia, is 4-5 watts. For our low-end GPU, the GeForce GT 640, average power is around 20-25 watts. Note that these devices implement a Thermal Design Power (TDP), the heat dissipation solution, ready to fight against peaks of 11 W for the Tegra and 65 W for the GT 640 (see Table 2).

Finally, for the mid-end GPU, the GeForce GTX 980 doubles average power to 41 watts. However, because it is a much faster devices, the kernel is executed on a much shorter execution time, allowing it to save energy in the first two executions, `Q-norm` and `reg_f3d`. `bedpostx` is the only method

Table 3: Power measurement, execution time and energy consumption on a low-power Tegra GPU (Jetson TK1) and a low-end GeForce GPU (GT 640) for three different biomedical methods. Reg_f3d also includes results on a mid-end GeForce GTX 980 GPU.

Biomedical method: →	Q-norm		reg_f3d (NiftyReg)			bedpostx (FDT)	
Running on: →	Jetson	GT 640	Jetson	GT 640	GTX 980	Jetson	GT 640
Average power (W)	5,59	20,13	4,08	19,19	41,06	3,69	24,20
Elapsed time (s)	830,80	246,17	14,10	0,73	0,44	75,37	19,17
Energy spent (Ws)	4644,17	4955,40	57,62	14,03	18,06	278,30	464,09
Time reduction vs. Jetson		70,36%		94,82%	96,87%		74.56%
Energy reduction vs. Jetson		-6.70%		75,64%	68,65%		-66,76%

Table 4: Energy budget on a 28 nm. manufacturing process chip: Fetching operands costs more than computing on them.

	Computational task performed on a 28 nm. GPU (all Kepler and Maxwell models)	Power consumption (energy in picojoules)
Computation	Add operator using integer operands (ALU)	0,4
	Mul operator using fp64 operands (FPU)	25
	Fused multiply-add on fp64 operands (FPU)	40
Data movement	Transition (milimeter traversed per bit)	0,2
	On-chip fp64 communication [1, 10, 20 mm.]	[3, 64, 250]
	Efficient off-chip link	500
Memory access	Local access to a register file	2
	256-bit access to on-chip 8 KB. SRAM cache	50
	DRAM read/write (for an entire cache line)	16000
Instruction scheduling	On an out-of-order 2010 CPU (Intel models)	2000
	On an out-of-order 2015 CPU (Intel models)	560
	On an in-order 2015 GPU (Nvidia models)	3

Source: [49]

where Jetson consumes less power (almost half), despite a slow-down factor of 3.93x in execution time.

Note our selection of assorted workloads requiring hundreds of seconds (`Q-norm`), units (`reg_f3d`) and tens (`bedpostx`) in the GeForce GT 640, where the intermediate workload is the major energy savings winner in Jetson. That way, benefits of low-power devices do not correlate with small or large runs. Instead, they do with data access locality (see Table 4 to realize about the importance of this feature in power consumption). Table 5 summarizes our findings in these respects.

Table 5: Energy savings on a discrete GeForce GT 640 GPU versus a Jetson TK1. Workload has a marginal influence, but data locality highly correlates with a competitive power consumption on discrete GPUs.

Running method	Workload (seconds)	The way data locality is improved	Energy savings in GT 640 vs. Jetson
`Q-norm`	Heavy (hundreds)	Using shared memory (fair) [12]	-6.70%
`reg_f3d`	Low (units)	Using texture memory (fine) [24]	75.64%
`bedpostx`	Average (tens)	No improvements (poor)	-66.76%

5.2. Comparison with a commodity GeForce on reg_f3d

Table 4 reflects the energy budget within a commodity GPU, where the GeForce GTX 980 model represents a good example. We have chosen the `reg_f3d` method to compare this GPU versus the low-power and low-end counterparts, as it is our most representative memory-bound algorithm.

The central part of Table 3 shows the numbers to pay attention here. The GTX 980 contains 2048 processing cores, 5 times more than the GT 640 and 10 times more than the Jetson. But its execution time difference shortens to just 1.66 times faster against GT 640, and widens to 32 times faster versus Jetson, which plays a decisive role on energy spent.

Data access locality in our optimized version of `reg_f3d` using texture memory [24] explains the competitive performance and power for a GPU with that small number of cores and memory bandwidth.

5.3. The input/output influence using Q-norm

It is common to find typical large input/output times on bioinformatics codes. Files containing data structures are usually huge and the computational workload, together with the acceleration effort, is just a fraction of the elapsed time. A good example is our `Q-norm` method, which requires an input data set of 470 probes, 25 Mbytes each, for a total of 11.5 Gbytes to be transfered to the GPU. Unfortunately, this processor is far away from input/output devices, and suffers from its high latency, followed by a small bandwidth of PCI-express.

As Table 6 shows, reading those data from the Jetson requires 1742,63 seconds, for a poor bandwidth of 6,74 Mbytes/s. On a discrete GPU, that time is just 429,93 seconds, for a bandwidth of 27,33 Mbytes/s. And using a RAID 0 setup of two hard disks, the Velociraptor model by Western Digital, the time goes down to 162,32 seconds, for a bandwidth of 72,38 Mbytes/s.

Table 6: Influence of I/O and transfer times in the actual computation time of Q-norm on the GPU (in seconds).

	Jetson TK1			GeForce GT 640		
Operation performed	File on hard disk	PCI express	GPU processing	File on hard disk	PCI express	GPU processing
Read from CPU	1742,63			429,93		
CPU to GPU transfer		14,86			1,87	
GPU computation			830,80			246,17
GPU to CPU transfer		27,13			4,10	
Write from CPU	625,37			59,46		
TOTAL	2368,00	41,99	830,80	489,39	5,97	246,17
Relative weight	73,06%	1,29%	25,63%	66,00%	0,008%	33,19%

Table 7: Power measurements and execution times for the `Fit_PVM_multi` kernel (alias `PVM_m`) within FDT.

CUDA kernel: `Fit_PVM_multi` (alias `PVM_m`)					
	GTX 480	GTX 680	GTX 780	GTX 980	Titan X1
PCIe 12v power line	46,66 W	29,96 W	31,88 W	30,61 W	33,27 W
PCIe 3,3v power line	1,26 W	1,63 W	2,19 W	1,86 W	2,54 W
Ext. 12v 6-pin back	71,42 W	41,32 W	55,89 W	35,15 W	81,71 W
Ext. 12v 6-pin front	37,07 W	27,55 W	35,22 W	30,75 W	37,23 W
Average power	165,22 W	100,48 W	130,45 W	102,48 W	161,51 W
Elapsed time (secs.)	**20,94 s**	**26,47 s**	**19,16 s**	**8,47 s**	**5,89 s**
Energy spent (Jules)	**3459,96 J**	**2660,14 J**	**2500,42 J**	**868,18 J**	**952,61 J**
Time reduction vs. GTX 480		-26,42%	8,48%	59,55%	71,84%
Energy reduction vs. GTX 480		23,12%	27,74%	74,91%	72,47%

For `Q-norm`, we have found the input/output time to be as much as 3-4 times the execution time. Fortunately, this excessive time will soon be decreased using the new Solid State Disks (SSD) and the NVlink bus developed by Nvidia to replace PCI-express on newer GPUs.

5.4. Performance per watt analysis on bedpostx

We decompose the GPU contribution for `bedpost` into four kernels: `PVM_m` (responsible for around 2-5% of the execution time - see results in Table 7), `PVM_s` (1-2%, Table 8), `fibres` (negligible, Table 9), and `runmcmc`(the bulk of the GPU computation time with 20% - see results in Table 10).

5.4.1. Average power

The first thing we discuss is the average power, which keeps consistent across kernels, with a little increment in the last two GPUs, close to 130 W

Table 8: Power measurements and execution times for the `Fit_PVM_single` kernel (alias `PVM_s`) within FDT.

CUDA kernel: `Fit_PVM_single` (alias `PVM_s`)					
	GTX 480	GTX 680	GTX 780	GTX 980	Titan X1
PCIe 12v power line	46,66 W	29,96 W	31,94 W	28,88 W	27,96 W
PCIe 3,3v power line	1,26 W	1,63 W	2,19 W	1,86 W	2,54 W
Ext. 12v 6-pin back	69,72 W	41,32 W	56,11 W	32,87 W	72,33 W
Ext. 12v 6-pin front	35,97 W	27,55 W	35,47 W	29,82 W	36,23 W
Total average power	162,13 W	100,48 W	131,03 W	99,90 W	147,25 W
Elapsed time	**11,06 s**	**12,42 s**	**9,51 s**	**5,51 s**	**4,40 s**
Energy spent	**1794,65 J**	**1252,37 J**	**1247,19 J**	**550,78 J**	**648,52 J**
Time reduction vs. GTX 480		-12,22%	14,02%	50,20%	60,22%
Energy reduction vs. GTX 480		30,22%	30,51%	69,31%	63,87%

Table 9: Power measurements and execution times for the `Init_fibres_multifibres` kernel (alias `fibres`) within FDT.

CUDA kernel: `Init_fibres_multifibres` (alias `Fibres`)					
	GTX 480	GTX 680	GTX 780	GTX 980	Titan X1
PCIe 12v power line	51,18 W	33,65 W	30,16 W	24,83 W	28,07 W
PCIe 3,3v power line	1,27 W	1,63 W	2,19 W	1,86 W	2,53 W
Ext. 12v 6-pin back	42,25 W	36,11 W	51,63 W	45,38 W	88,54 W
Ext. 12v 6-pin front	77,28 W	24,09 W	31,68 W	36,07 W	48,91 W
Total average power	181,80 W	95,50 W	120,54 W	112,87 W	177,20 W
Elapsed time	**0,0138 s**	**0,0200**	**0,0158**	**0,0136**	**0,0092**
Energy spent	**2,50 J**	**1,91**	**1,90**	**1,53**	**1,63**
Time reduction vs. GTX 480		-44,93%	-14,49%	1,45%	33,34%
Energy reduction vs. GTX 480		23,87%	24,09%	38,82%	35,03%

Table 10: Power measurements and execution times for the `Runmcmc` kernel within FDT.

CUDA kernel: `Runmcmc`					
	GTX 480	GTX 680	GTX 780	GTX 980	Titan X1
PCIe 12v power line	53,46 W	38,10 W	34,48 W	38,92 W	40,88 W
PCIe 3,3v power line	1,26 W	1,62 W	2,18 W	1,85 W	2,53 W
Ext. 12v 6-pin back	44,25 W	52,85 W	62,50 W	44,28 W	99,42 W
Ext. 12v 6-pin front	83,87 W	34,92 W	42,29 W	38,68 W	45,58 W
Total average power	193,29 W	127,51 W	148,20 W	128,80 W	197,11 W
Elapsed time	**46,77 s**	**66,37 s**	**46,62 s**	**40,57 s**	**28,25 s**
Energy spent	**9041,90 J**	**8463,63 J**	**6909,31 J**	**5225,85 J**	**5570,11 J**
Time reduction vs. GTX 480		-41,90%	0,40%	13,27%	39,59%
Energy reduction vs. GTX 480		6,40%	23,59%	42,21%	38,40%

and 200 W in the last kernel. Note, however, that this is far from dangerous values, established by the manufacturer as the Thermal Design Power (165 W and 250 W, respectively - see Table 2). The PCI slot has a power limit of 70W, which is never surpassed, and, in fact, newer models use less extensively, overloading more the external lines (6-pin connector directly from PSU).

Average power remains stable in the first two kernels, which have similar arithmetic intensity, and goes up for the last two, particularly `runmcmc`, that computes a Markow Chain Monte Carlo process and is the only one doubling the number of iterations (64 versus 32 in the other three kernels). That way, `runmcmc` improves data locality and doubles arithmetic intensity. Even though operations are not demanding in energy budget (see Table 4), they are extremely fast (particularly those required by Monte Carlo), and therefore, energy compresses in short periods of time, increasing the average. When GPU takes most of the time communicating, the energy budget increases, but because data takes longer to traverse the chip, watts per second are not that demanding and average power may eventually relax.

5.4.2. Power by generations

Average power is quite unstable across GPU generations. We start with the highest values in the older GPU, the GTX 480, as a consequence of its impressive frequency, 1.40 GHz, and 40 nm. transistors. All remaining GPUs analized in this section are manufactured with 28 nm. transistors, which cuts the energy budget by as much as 30% on every similar operation. Average power values in Titan X1 are disappointing, and can be justified only from the optimistic perspective of its high speed-up factors: This GPU manages to reduce both time and energy around 40% on the heavier kernel, which is not bad at all for a five year period. We now enter into more details about time and energy.

5.4.3. Speed-up factors

We compile acceleration factors for FDT and all GPU models in Table 11. The only disappointing result stands for the GTX 680, which is hard to explain. Versus GTX 480, the frequency relaxes 30%, but the number of cores increase by a 3.2x factor, and cores occupancy ratios keep high for the most demanding kernel, `runmcmc`. Honestly, we do not find an explanation for this. On the optimistic side, we have the Titan X1, responsible for an acceleration of 2,04x versus GTX 480 and 2,72x versus GTX 680, with a peak acceleration factor of 4,48x for the `PVM_m` kernel (see Table 12). The

Table 11: Average power, execution time and kernel percentage for every GPU and FDT kernel analyzed.

		PVM_m	PVM_s	Fibres	Runmcmc	**Total**
GTX 480	Power	165,22 W	162,13 W	181,80 W	193,29 W	**186,81 W**
	Time	20,94 s	11,06 s	0,01 s	46,77 s	**78,78 s**
	% kernel	3,87 %	1,70 %	0,00 %	19,40 %	**24,97 %**
GTX 680	Power	100,48 W	100,48 W	95,50 W	127,51 W	**120,31 W**
	Time	26,47 s	12,42 s	0,02 s	66,37 s	**105,29 s**
	% kernel	4,62 %	2,00 %	0,00 %	18,25 %	**24,87 %**
GTX 780	Power	130,45 W	131,03 W	120,54 W	148,20 W	**144,59 W**
	Time	19,16 s	9,51 s	0,01 s	46,62 s	**75,33 s**
	% kernel	3,65 %	1,47 %	0,00 %	19,87 %	**24,99 %**
GTX 980	Power	102,48 W	99,90 W	112,87 W	128,80 W	**124,32 W**
	Time	8,47 s	5,51 s	0,01 s	40,57 s	**54,57 s**
	% kernel	2,07 %	0,90 %	0,25 %	21,97 %	**25,19 %**
Titan X1	Power	161,51 W	147,25 W	177,20 W	197,11 W	**185,37 W**
	Time	5,89 s	4,40 s	0,00 s	28,25 s	**38,57 s**
	% kernel	2,10 %	0,92 %	0,00 %	21,97 %	**24,99 %**

maximum speed-up for remaining kernels is also attained on the Titan X1 vs. GTX 680 comparison: 2,82x for PVM_m, 2,17x for Fibres, 2,34x Runmcmc and 2,72x in total (on average).

Overall, the evolution of GPUs is satisfactory along these years, because we benefit from all these accelerations for free. That is, without being required to tune our kernels to exploit any of the new CUDA Compute Capabilities (CCC - we start with a CCC 2.0 model and end with a 5.2 model). Making use of GPU Boost (introduced in CCC 3.0), Dynamic Parallelism and/or Hyper-Q (CCC 3.5) or Unified Memory (CCC 5.0) we would have pushed forward speed-up factors (but it remains to be seen the cost in programming effort).

5.4.4. Energy efficiency

Table 13 summarizes power, time and energy on its main diagonal (higher values are bad news), and leaves remaining cells to establish a final comparison across GPUs (with higher values being good news this time). Values below the diagonal are expected to be better than those above, because they compare a newer/better GPU versus and older/worse counterpart.

Reading the table vertically on the lower triangle, we are witnesses of

Table 12: Acceleration factors for every FDT kernel and overall.

CUDA kernel	PVM_m	PVM_s	Fibres	Runmcmc	Total
Kernel weight (approx.)	3%	1,75%	0,25%	20,00%	**25,00** %
GTX 680 vs. GTX 480	0,79x	0,89x	0,69x	0,70x	**0,74x**
GTX 780 vs. GTX 480	1,09x	1,16x	0,87x	1,00x	**1,04x**
GTX 980 vs. GTX 480	2,47x	2,00x	1,01x	1,15x	**1,44x**
Titan X1 vs. GTX 480	3,55x	2,51x	1,50x	1,65x	**2,04x**
GTX 780 vs. GTX 680	1,38x	1,30x	1,26x	1,42x	**1,39x**
GTX 980 vs. GTX 680	3,12x	2,25x	1,47x	1,63x	**1,92x**
Titan X1 vs. GTX 680	4,48x	2,82x	2,17x	2,34x	**2,72x**
GXT 980 vs. GTX 780	2,26x	1,72x	1,16x	1,14x	**1,38x**
Titan X1 vs. GTX 780	3,24x	2,16x	1,71x	1,64x	**1,95x**
Titan X1 vs. GTX 980	1,43x	1,25x	1,47x	1,43x	**1,41x**

scalability on GPU technology, a feature GPUs are famous for (higher speed-ups on lower rows, with a peak of 2,73x). However, the power for Titan X1 (185,37 W) is not that competitive, particularly when compared to the GTX 980 (124,32 W), which becomes the most energy efficient GPU with a peak of 2,30x versus GTX 480. That is the toll to pay for having 50% more cores activated (3072 versus 2048 for the same generation and architectural model).

With a total of 6784,14 jules spent to complete our entire FDT application, and on an average fare of 0,13€/kWh, the GTX 980 spends 0,245€ versus 0,563€ required by the GTX 480. Remember that GTX 480 is the only GPU manufactured on a 40 nm. process (remaining four are at 28 nm. process), and also abuses of frequency (1.4 GHz), which has dramatic consequences on the performance per watt ratio.

Now reading the table horizontally, we can conclude that acceleration attained and energy spent correlate quite well, particularly within all 28 nm. GPUs (Kepler and Maxwell generations). (higher speed-up factors are more energy demanding). For example, when Titan X1 accelerates 2,04x on the lower left corner, consumes 2,19x less energy. But if we reduce the speed-up to 1,41x (lower right corner), there is an energy deficit around 5%.

5.5. Performance per watt analysis on multi-tensor tractography

We now analyze the modelling of diffusion images through our deterministic approach. Tensors extract enhanced information about the diffusion process inside a voxel, which allow us to compute a wide range of popular

Table 13: Main diagonal: Power, time and energy spent for every GPU running FDT (note that in all cases, higher values are worse). Remaining cells are comparison factors in the same order (but this time, higher values are good news). For example, Titan X1 is 1,01x, 2,04x and 2,19x better in power, time and energy versus GTX 480, respectively (see last triplet of rows, first column).

	GTX 480	GTX 680	GTX 780	GTX 980	Titan X1
GTX 480	186,81 W	0,65x	0,76x	0,67x	0,99x
	78,78 s	1,37x	0,95x	0,70x	0,49x
	15662,25 J	0,80x	0,69x	0,43x	0,46x
GTX 680	1,55x	120,31 W	1,20x	1,03x	1,54x
	0,73x	105,29 s	0,71x	0,52x	0,37
	1,24x	12667,43 J	0,87x	0,54x	0,56x
GTX 780	1,29x	0,83	144,59 W	0,86x	1,28x
	1,05x	1,40x	75,33 s	0,72x	0,51x
	1,44x	1,16x	10891,96 J	0,62x	0,66x
GTX 980	1,50x	0,97x	1,16x	124,32 W	1,49x
	1,44x	1,93x	1,38x	54,57 s	0,71x
	2,30x	1,86x	1,61x	6784,14 J	1,05x
Titan X1	1,01x	0,65x	0,78x	0,67x	185,37 W
	2,04x	2,73x	1,95x	1,41x	38,57 s
	2,19x	1,77x	1,52x	0,95x	7149,72 J

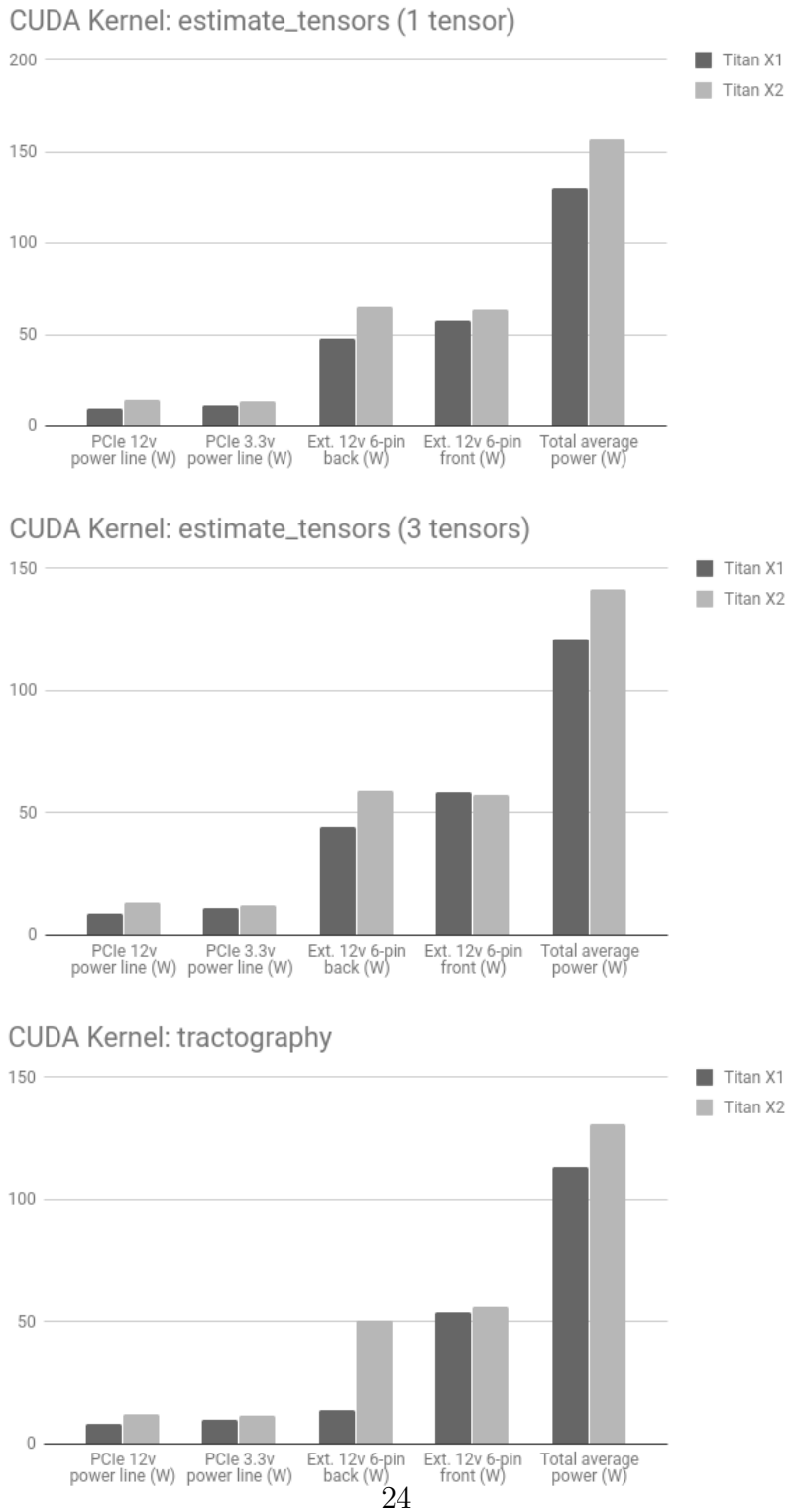


Figure 11: Watts spent for each power line and kernel involved in our multitensor tractography.

Table 14: Power measurements and execution times for the estimate_tensors_kernel using one tensor.

CUDA kernel: `estimate_tensors` (1 tensor)				
	Titan X1	Titan X2	Difference	
PCIe 12v power line	9,26 W	14,73 W	+5,47 W	(+59,07%)
PCIe 3.3v power line	11,55 W	13,71 W	+2,16 W	(+18,70%)
Ext. 12v 6-pin back	47,70 W	64,99 W	+17,29 W	(+36,24%)
Ext. 12v 6-pin front	57,47 W	63,67 W	+6,20 W	(+10,78%)
Total average power	129,59 W	157,30 W	+27,71 W	(+21,38%)
Elapsed time	**106,68 s**	**88,98 s**	-17,70 s	(-16,59%)
Energy spent	**13823,62 J**	**13996,35 J**	**+172,73 J**	**(+1,25%)**

Table 15: Power, time and energy elapsed for the estimate_tensors_kernel using three tensors.

CUDA kernel: `estimate_tensors` (3 tensors)				
	Titan X1	Titan X2	Difference	
PCIe 12v power line	8,74 W	13,03 W	+4,29 W	(+49,08%)
PCIe 3.3v power line	10,92 W	12,14 W	+1,22 W	(+11,17%)
Ext. 12v 6-pin back	44,23 W	58,80 W	+14,57 W	(+32,94%)
Ext. 12v 6-pin front	58,26 W	57,51 W	-0,75 W	(-1,29%)
Total average power	121,09 W	141,54 W	+20,45 W	(+20,45%)
Elapsed time	**562,79 s**	**308,58 s**	**-254,21 s**	**(-45,16%)**
Energy spent	**68150,68 J**	**43677,33 J**	**-24473,35 J**	**(-35,91%)**

neuroimaging biomarkers.

Tables 14, 15 and 16 gather power, time and energy spent on each of the three kernels involved for this method, and Figure 11 compares kernels for each power line. The first thing we appreciate is a different way of distributing power through PCIe and external lines. The probabilistic method, `bedpostx`, takes around 40 W from the PCIe 12v line on the Titan X1 GPU, whereas the deterministic formulation, let us call it (`multitensor`, keeps this value consistently below the 10 W threshold. With the PCIe 3,3v line, we see the opposite effect: around 2.5 W for `bedpostx` and 10 W for `multitensor`. For the pair of external lines, `bedpostx` takes much more power from the back line (100 W vs. 45 W), whereas `multitensor` relies slightly more on the front line (45 W vs. 57 W).

Table 17 summarizes the comparison between `bedpostx` and `multitensor`

Table 16: Power measurements and execution times for the tractography_kernel.

CUDA kernel: `tractography`				
GPU	Titan X1	Titan X2	Difference	
PCIe 12v power line	7,95 W	12,22 W	+4,27 W	(+53,71%)
PCIe 3.3v power line	9,95 W	11,39 W	+1,44 W	(+14,47%)
Ext. 12V 6-pin back	41,08 W	50,36 W	+9,28 W	(+22,59%)
Ext. 12V 6-pin front	54,12 W	56,27 W	+2,15 W	(+3,97%)
Total average power	113,33 W	130,96 W	+17,63 W	(+15,55%)
Elapsed time	**203,00 s**	**155,36 s**	**-47,64 s**	**(-23,46%)**
Energy spent	**23006,87 J**	**20347,15 J**	**-2659,72 J**	**(-11,56%)**

Table 17: Comparing power, execution time and energy spent on the Titan X1 GPU for the probabilistic and deterministic approaches within modelling of diffusion images.

	Probabilistic	Deterministic	Difference
Total power	185,37 W	120,32 W	0,65x
Total time	38,57 s	872,46 s	22,62x
Total energy	7149,42 J	104976,38 J	14,68x
Energy cost	0,00026 €	0,0038 €	14,68x
% GPU was used	24,99%	5,31%	0,21x

for power, time and energy spent on the Titan X1 GPU. Average power is much lower on the latter, but time elapsed and energy spent are 22 and 14 times larger, respectively. It is logical to find more relaxed values for power on a process which takes much longer. Moreover, our deterministic approach perform more calculations on a similar data set, and arithmetic intensity goes up. GPUs are processors that consume much more power when fetching data than computing on them (see Table 4), so we see how power goes down on average and the energy factor is not as high as the speed-up. We also consider that chip temperature is not that dramatic on the `multitensor`side to end up on a better aging process for the hardware over time, but at the expense of a higher energy cost.

Finally, Table 18 shows the contribution of the new Pascal GPU to speed-up `multitensor`, our slowest method, and the energy costs associated. Acceleration factors vary from 1,2x to 1,8x depending on the kernel, and energy costs correlate well with speed-up factors. That way, the best scenario is offered by the 3 tensors kernel (maximum speed-up and energy savings - 1,8x

Table 18: Average power, execution time and energy spent on every kernel within the multitensor tractography algorithm. Values are shown for the Titan X1 GPU, Titan X2, and a comparison of both.

		1 tensor	3 tensor	tractography	TOTAL
Titan X1	Power	129,59 W	121,09 W	113,33 W	120,32 W
	Time	106,67 s	562,78 s	203,00 s	872,46 s
	Energy	13823,36 J	68147,03 J	23005,99 J	104976,38 J
	% kernel	0,65%	3,43%	1,24%	5,31%
Titan X2	Power	157,30 W	141,54 W	130,96 W	141,10 W
	Time	88,98 s	308,57 s	155,36 s	552,92 s
	Energy	13996,55 J	43674,99 J	20345,94 J	78017,48 J
	% kernel	0,55%	1,92%	0,96%	3,43%
Titan X2 gain	Power	0,82x	0,85x	0,86x	**0,85x**
	Time	1,20x	1,80x	1,31x	**1,58x**
	Energy	0,98x	1,63x	1,13x	**1,34x**
Titan X2 scoring vs. X1:		Worst	Best	Intermediate	

and 1,63x), followed by the tractography kernel (1,31x and 1,13x) and the one tensor kernel (1,2x and 0,98x) The final row in the table summarizes this analysis. Overall, the Titan X Pascal GPU is able to reduce average power by 15%, time by 58% and energy by 34% versus the Titan X Maxwell, which is the similar GPU model of the previous GPU generation.

6. Conclusions

This paper explores a wide set of biomedical methods from a HPC and power-efficiency perspective, providing a variety of scenarios where GPUs are able to accelerate applications at different ranges without hurting power consumption. Even more, we have demonstrated that speep-up factors and energy savings correlate quite well, mainly due to the following reasons:

1. Average power usually goes up whenever time shortens, but the time reduction uses to be wider than the average increase.
2. Higher speed-up factors correspond to newer technology, where manufacturing process (transistor gate width) shrinks, thus providing us more power-efficient hardware over time.
3. GPUs do not abuse of frequency in latest generations. They try to keep it relaxed, thus influencing power consumption in a very positive way.

4. Horsepower in GPUs relies much more on number of cores provided by more recent GPU models, where the risk for energy penalties come only from larger communications within the chip. However, Nvidia architectures managed to increase the core count through a greater number of multiprocessors endowed with fewer cores. In CUDA, most of the communications are internal to each multiprocessor, and that way, paths to communicate data are shorter on every newer generation. With that recipe in mind, you can increase GPU throughput and save energy at the same time, as our work has extensively proven.

For the set of GPUs involved in our study, Kepler designs include multiprocessors of 192 cores, Maxwell ones reduce this amount to 128, and Pascal shorten to 64. If we keep frequency on similar ranges and benefit from transistor shrinks, speed-up factors and energy savings can be attained both at a time. For example, the Titan X Pascal GPU was able to reduce time by 58% and energy by 34% versus the counterpart Maxwell model when running one of our biomedical methods. We expect those benefits to increase on the already announced Volta generation by Nvidia, where number of cores increase from 3584 to 5120, frequency relaxes from 1480 to 1455 MHz and transistors shrink from 16 to 12 nm.

Additionally, the set of experimental results we have gathered along this paper suggest that data locality and arithmetic intensity represent the most rewarding ways to accelerate applications when energy is a major concern.

Acknowledgements

This work was supported by the Ministry of Education of Spain under Project TIN2013-42253-P and by the Junta de Andalucia under Project of Excellence P12-TIC-1741. We thank Nvidia for hardware donations within GPU Education Center 2011-2016 and GPU Research Center 2012-2016 awards at the University of Malaga (Spain). We also thank Francisco D. Igual and Luis Piñuel from the Computer Architecture and Automated Department at the Complutense University of Madrid (Spain) for providing us Accelpower modules to measure power during our experimental survey. Our measuring system is based on a tool being continuously upgraded as reported in http://accelpowercape.dacya.ucm.es.

References

[1] GPGPU, General-Purpose Computation Using Graphics Hardware, http://www.gpgpu.org (2009) [cited Nov 2009].
URL `http://www.gpgpu.org`

[2] The Top 500 Supercomputers List, `http://www.top500.org`.

[3] G. Karthikeyan, P. Jayachandran, N. Venkataraman, Energy Aware Network Scheduling for a Data Centre, International Journal of Big Data Intelligence 2 (1).

[4] Y. Ngoko, C. Cerin, P. Gianessi, C. Jiang, Energy-aware Service Provisioning in Volunteers Clouds, International Journal of Big Data Intelligence 2 (4).

[5] P. Sarwesh, N. Shekar, V. Shet, K. Chandrasekaran, Effective Integration of Reliable Routing Mechanism and Energy Efficient Node Placement Technique for Low Power IoT Networks, International Journal of Grid and High Performance Computing 9 (4).

[6] The Green 500 Supercomputers List, `http://www.green500.org`.

[7] B. Bolstad, R. Irizarry, M. Astrand, T. Speed, A Comparison of Normalization Methods for High Density Oligonucleotide Array Data based on Variance and Bias, Bioinformatics 19 (2) (2003) 185–193.

[8] M. Modat, NIFTYREG - A library to perform rigid, affine and non-linear registration of NIfTI images, http://sourceforge.net/projects/niftyreg/.

[9] FMRIB's Group. FDT: FMRIB's Diffusion Toolbox [online, cited October, 2017].

[10] FMRIB's Group. FSL: FMRIB Software Library [online, cited October, 2017].

[11] S. Peled, O. Friman, F. Jolesz, C.-F. Westin, Geometrically constrained two-tensor model for crossing tracts in dwi, Magnetic resonance imaging 24 (9) (2006) 1263–1270.

[12] A. Rodríguez, O. Trelles, M. Ujaldón, Using Graphics Processors for a High Performance Normalization of Gene Expressions, in: 13^{th} IEEE International Conference on High Performance Computing and Communications, Banff (Canada), 2011, pp. 599–604.

[13] J. Warrington, S. Dee, M. Trulson, Large-Scale Genomic Analysis Using Affymetrix GeneChip, Microarray Biochip Technologies, BioTechniques Books, New York, USA, 2000, Ch. 6, pp. 119–148.

[14] Affymetrix, Statistical Algorithms Referende Guide, Technical report, Affymetrix (2001).

[15] R. Irizarry, B. Hobbs, F. Colin, Y. Beazer-Barclay, K. Antonellis, U. Scherf, T. Speed, Exploration, Normalization and Summaries of High Density Oligonucleotide Array Probe Level Data, Biostatistics 4 (2) (2003) 249–264.

[16] The GeneChip Human Mapping 500K Array Dataset Submitted to GEO by Affymetrix [online, cited Jan, 2010].

[17] NIfTI, The NIfTI Format Home Page, http://nifti.nimh.nih.gov.

[18] DICOMNIFTI, A tool for converting DICOM files into the NIfTI format, http://cbi.nyu.edu/software/dinifti.php (October 2013).

[19] NIFTILIB, I/O libraries for NIfTI-1 neuroimaging data format, http://niftilib.sourceforge.net.

[20] CMIC, The NifTK Software Platform, http://www.niftk.org.

[21] B. B. Avants, N. J. Tustison, M. Stauffer, G. Song, B. Wu, J. C. Gee, The insight toolkit image registration framework, Frontiers in neuroinformatics 8.

[22] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, S. Ourselin, Fast free-form deformation using Graphics Processing Units, Computer Methods in Programs and Biomedicine 98 (3) (2010) 278–284.

[23] W. B. Langdon, M. Modat, J. Petke, M. Harman, Improving 3D Medical Image Registration CUDA Software with Genetic Programming, in:

ACM (Ed.), Proceedings Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 951–958.

[24] F. Álvarez, J. Cabrera, J. Chico, J. Pérez, M. Ujaldón, Neuroimaging Registration on GPU: Energy-aware Acceleration, in: 4th International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO'16), Granada (Spain), 2016.

[25] P. J. Basser, D. LeBihan, Fiber orientation mapping in an anisotropic medium with nmr diffusion spectroscopy, in: 11th Annual Meeting of the SMRM, Berlin, Vol. 1221, 1992.

[26] P. J. Basser, S. Pajevic, C. Pierpaoli, J. Duda, A. Aldroubi, In vivo fiber tractography using dt-mri data, Magnetic resonance in medicine 44 (4) (2000) 625–632.

[27] D. S. Tuch, Q-ball imaging, Magnetic resonance in medicine 52 (6) (2004) 1358–1372.

[28] C. P. Hess, P. Mukherjee, E. T. Han, D. Xu, D. B. Vigneron, Q-ball reconstruction of multimodal fiber orientations using the spherical harmonic basis, Magnetic Resonance in Medicine 56 (1) (2006) 104–117.

[29] B. Jian, B. C. Vemuri, E. Özarslan, P. R. Carney, T. H. Mareci, A novel tensor distribution model for the diffusion-weighted mr signal, NeuroImage 37 (1) (2007) 164–176.

[30] A. D. Leow, S. Zhu, L. Zhan, K. McMahon, G. I. de Zubicaray, M. Meredith, M. Wright, A. Toga, P. Thompson, The tensor distribution function, Magnetic Resonance in Medicine 61 (1) (2009) 205–214.

[31] I. Aganj, C. Lenglet, G. Sapiro, E. Yacoub, K. Ugurbil, N. Harel, Reconstruction of the orientation distribution function in single-and multiple-shell q-ball imaging within constant solid angle, Magnetic Resonance in Medicine 64 (2) (2010) 554–566.

[32] M. Xu, X. Zhang, Y. Wang, L. Ren, Z. Wen, Y. Xu, G. Gong, N. Xu, H. Yang, Probabilistic Brain Fiber Tractography on GPUs, in: IEEE (Ed.), Proceedings 26th Intl. Parallel and Distributed Processing Symposium. Workshop, 2012.

[33] T. Behrens, M. Woolrich, M. Jenkinson, H. Johansen-Berg, R. Nunes, S. Clare, P. Matthews, J. Brady, S. Smith, Characterization and Propagation of uncertainty in Diffusion-Weighted MR imaging, Magn Reson Med 50 (5) (2003) 1077–1088.

[34] M. Hernández, G. Guerrero, J. Cecilia, J. García, A. Inuggi, S. Jbabdi, T. Behrens, S. Sotiropoulos, Accelerating Fibre Orientation Estimation from Diffusion Weighted Magnetic Resonance Imaging Using GPUs, PLoS ONE 8 (4).

[35] Human connectome project, http://www.humanconnectome.org.

[36] Human connectome database, http://db.humanconnectome.org/data-/projects/MGH_DIFF.

[37] A. L. Alexander, K. M. Hasan, M. Lazar, J. S. Tsuruda, D. L. Parker, Analysis of partial volume effects in diffusion-tensor mri, Magnetic Resonance in Medicine 45 (5) (2001) 770–780.

[38] D. S. Tuch, T. G. Reese, M. R. Wiegell, N. Makris, J. W. Belliveau, V. J. Wedeen, High angular resolution diffusion imaging reveals intravoxel white matter fiber heterogeneity, Magnetic resonance in medicine 48 (4) (2002) 577–582.

[39] G. J. Parker, D. C. Alexander, Probabilistic anatomical connectivity derived from the microscopic persistent angular structure of cerebral tissue, Philosophical Transactions of the Royal Society of London B: Biological Sciences 360 (1457) (2005) 893–902.

[40] B. Kreher, J. Schneider, I. Mader, E. Martin, J. Hennig, K. Il'Yasov, Multitensor approach for analysis and tracking of complex fiber configurations, Magnetic resonance in medicine 54 (5) (2005) 1216–1225.

[41] J. G. Malcolm, M. E. Shenton, Y. Rathi, Filtered multitensor tractography, IEEE transactions on medical imaging 29 (9) (2010) 1664–1675.

[42] P. Hagmann, T. Reese, W. Tseng, R. Meuli, J. Thiran, V. Wedeen, Diffusion spectrum imaging tractography in complex cerebral white matter: an investigation of the centrum semiovale, in: International Society for Magnetic Resonance in Medicine, ISMRM TWELFTH SCIENTIFIC

MEETING, Kyoto, Japan, 15-21 May 2004, Vol. 12, International Society for Magnetic Resonance in Medicine, 2004, p. 623.

[43] W. Guo, Q. Zeng, Y. Chen, Y. Liu, Using multiple tensor deflection to reconstruct white matter fiber traces with branching, in: Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on, IEEE, 2006, pp. 69–72.

[44] BeagleBone, Beaglebone black, http://beagleboard.org/BLACK.

[45] J. González-Rincón, Sistema basado en open source hardware para la monitorización del consumo de un computador, Master Thesis Project. Universidad Complutense de Madrid.

[46] L. Ada, Adafruit INA219 Current Sensor Breakout, `https://learn.adafruit.com/adafruit-ina219\--current-sensor-breakout`.

[47] F. Igual, L. Jara, J. Gómez, L. Piñuel, M. Prieto, A Power Measurement Environment for PCIe Accelerators, Computer Science - Research and Development 30 (2) (2015) 115–124.

[48] P. Alonso, R. Badía, J. Labarta, M. Barreda, M. Dolz, R. Mayo, E. Quintana-Ortí, R. Reyes, Tools for power-energy modelling and analysis of parallel scientific applications, in: Proceedings 41st Intl. Conference on Parallel Processing (ICPP'12), IEEE Computer Society, 2012, pp. 420–429.

[49] W. J. Dally, From here to exascale: Challenges and potential solutions, in: Keynote Talk, Supercomputing 2012, 2012.