Performance gains with Compute Unified Device Architecture-enabled eddy current correction for diffusion MRI

Jerome J. Maller[a,b,c], Stuart M. Grieve[b,d], Simon J. Vogrin[e] and Thomas Welton[b]

Correcting for eddy currents, movement-induced distortion and gradient inhomogeneities is imperative when processing diffusion MRI (dMRI) data, but is highly computing resource-intensive. Recently, Compute Unified Device Architecture (CUDA) was implemented for the widely-used eddy-correction software, 'eddy', which reduces processing time and allows more comprehensive correction. We investigated processing speed, performance and compatibility of CUDA-enabled eddy-current correction processing compared to commonly-used non-CUDA implementations. Four representative dMRI datasets from the Human Connectome Project, Alzheimer's Disease Neuroimaging Initiative and Chronic Diseases Connectome Project were processed on high-specification and regular workstations through three different configurations of 'eddy'. Processing times and graphics processing unit (GPU) resources used were monitored and compared. Using CUDA reduced the 'eddy' processing time by a factor of up to five. The CUDA slice-to-volume correction method was also faster than non-CUDA eddy except when datasets were large. We make a series of recommendations for eddy configuration and hardware. We suggest that users of eddy-correction software for dMRI processing utilise CUDA and take advantage of the slice-to-volume correction option. We recommend that users run eddy on computers with at least 32GB motherboard random access memory (RAM), and a graphics card with at least 4.5GB RAM and 3750 cores to optimise processing time. *NeuroReport* **31: 746–753**

NeuroReport 2020, **31**:746–753

Keywords: Compute Unified Device Architecture, diffusion, eddy currents, MRI

[a]General Electric Healthcare, Victoria, [b]Imaging and Phenotyping Laboratory, Charles Perkins Centre, Faculty of Medicine and Health, The University of Sydney, NSW, [c]Department of Psychiatry, Monash Alfred Psychiatry Research Centre, Victoria, [d]Department of Radiology, Royal Prince Alfred Hospital, Camperdown, Sydney, NSW and [e]Department of Neurology, Centre for Clinical Neuroscience and Neurological Research, St Vincent's Hospital, Fitzroy, Victoria, Australia

Correspondence to Jerome J. Maller, PhD, Sydney Translational Imaging Laboratory, Heart Research Institute, Charles Perkins Centre, University of Sydney, Sydney, NSW 2050, Australia
Tel: +(02) 8627 1616; e-mail: jerome.maller@ge.com

Received 5 April 2020 **Accepted** 18 April 2020

Introduction

Performing correction for eddy currents, patient movement and off-field resonance/gradient field inhomogeneity is vital for proper preprocessing of diffusion MRI (dMRI) data. Eddy currents are strong when acquiring dMRI data especially when b-values (diffusion strengths) are high. Movement induced by subjects is ubiquitous in MRI, but during dMRI, its effect becomes greater due to the relatively long acquisition time, and pulsing of the diffusion-sensitising gradients creating noise and vibration [1]. Whilst the latest generation of scanners have reduced vibration and increased magnet and gradient homogeneity, subject movement and inhomogeneity is inevitable. Even small movements (1–2 mm) can produce inter-voxel distortion, creating a partial volume effect. Off-field resonance is also problematic in MRI scanning and leads to image distortion, particularly in brain regions at the tissue-bone interface, such as the inferior temporal and frontal lobes.

In recent years, preprocessing software has become available which aims to estimate these distortions and apply corrections to the dMRI data; for example, using the popular software, 'topup' [2]. By acquiring the dMRI data in one phase-encoding direction and repeating the acquisition in the opposite phase-encoding direction, field maps can be generated. The result is fed into eddy-, motion- and off-field resonance-correction software, referred to in the FSL package as 'eddy' [3]. This performs various stages of correction for susceptibility- and eddy current-induced distortions as well as between-volume movement signal loss caused by subject motion. The time required to process the data depends on several factors, including the amount and speed of computer memory [random access memory (RAM)], speed and cache size of the CPU core, the read/write speed of the storage medium and size of the dataset. Eddy is also available as multi-threaded ('eddy_openmp', multi-processor); however, even then, processing is time consuming, and may take several hours for a standard multishell dMRI dataset on conventional hardware. Although some computers have the option to activate hyperthreading (a process where a CPU splits each of its physical cores into virtual cores, which are known as threads), it has not been documented as to whether this makes a substantial difference to processing time.

Recently, Compute Unified Device Architecture (CUDA) has been implemented for eddy, which is an extension of the C programming language (Nvidia, 2007,

0959-4965 DOI: 10.1097/WNR.0000000000001475

CUDA Technology; http://www.nvidia.com/CUDA). Using CUDA, the user can utilise parallel computing power on an Nvidia graphics card. The number of CUDA cores, their speed, the amount and speed of memory on the graphics card and the memory bandwidth are contributing factors to this. The most recent version of the CUDA-enabled eddy correction software also offers an option for intra-volume (slice-to-volume) correction. That is, it aims to realign slices within a single volume affected by motion artefacts. It has been shown that this approach can estimate motion with greater accuracy, and parameters derived from the data (e.g. fractional anisotropy) are estimated more accurately when data have been corrected in this way [4]. Andersson *et al.* [4] also demonstrated that, when corrected with the slice-to-volume approach, datasets with various amounts of motion have a smaller disparity in fidelity, and this was recently validated using an automated quality control framework [5]. However, it is not currently known the extent to which CUDA reduces eddy correction processing time and under what conditions.

We therefore aimed to investigate the speed, performance and compatibility of CUDA-enabled eddy current correction. Specifically, we aimed to understand the impact of the type of input dataset, the type of hardware on which it is run and the configuration of the software. We test four different widely-used datasets which were selected to represent common acquisition schemes. We implemented our methods on both a computer at the high-end of performance for a single machine, and on a conventional computer. Finally, we tested multi-processor, CUDA and slice-to-volume configurations of eddy to gauge the performance gains and value of each relative to the processing time.

Materials and methods

Subjects

For two datasets, a 42-year-old male subject who had no history of a psychiatric, neurological or cardiac disorder and no contraindication to MRI scanning underwent MR imaging at Macquarie Medical Imaging, Sydney, NSW, Australia, as part of the Chronic Diseases Connectome Project (CDCP). The study has Institutional Ethics approval (from the Macquarie University Human Research Ethics Committee for Medical Sciences, ID number 5201500943) and the subject provided informed consent. All methods were performed in accordance with the relevant guidelines and regulations. Two other datasets were drawn from the ADNI project (ID003.S.4900, 59-year-old female, control subject), and from the HCP project (ID100307, 27-year-old female, control subject).

Image acquisition

Data from the CDCP study were acquired using a General Electric MR750w scanner (GE Healthcare, Milwaukee, Wisconsin, USA) with a Nova 32-channel brain coil (Nova Medical). Anterior commissure-posterior commissure aligned scans were acquired: spin-echo DTI EPI pulse sequence comprising three shells (b = 700, 20 directions, b = 1000, 45 directions, b = 2800, 75 directions; TR = 4323 ms, TE = 91.8 ms, FOV = 256 mm, matrix = 128 × 128, voxel dimensions = 2 mm isotropic) with eight interleaved b = 0 volumes, referred to as dataset 1. We then extracted the outer shell (b = 2800) for a second dataset (with five b = 0 volumes at the beginning and three interleaved, i.e. one after every 20 diffusion volumes), referred to as dataset 2. Acceleration factor of two and multiband factor of three (resulting in 66 slices per volume) was used for the dMRI session. A further diffusion-weighted sequence was also acquired (six directions) with a reverse phase-encoded (blipped) non-diffusion weighted (b = 0) volume and a phase offset applied to each multiband component.

Data from the ADNI2 project were acquired on a General Electric Signa HDxt 3T scanner (GE Healthcare) with an eight-channel head coil (TR = 13 000 ms, TE = 68.4 ms, matrix = 128 × 128 interpolated to 256 × 256, 60 slices, slice thickness = 2.7 mm, in-plane = 1.37 × 1.37 mm, 41 unique diffusion directions, b-value = 1000, in-plane acceleration = 2) and five bzero volumes at the beginning. This will be referred to as dataset 3.

Data from the HCP project were acquired on a Siemens 3T Connectom scanner with a 32-channel head coil. Three separate acquisitions were merged into a single dataset to yield 273 unique diffusion directions comprising three interleaved b-values (1000, 2000, 3000) with 16 bzero interleaved volumes, 111 slices per volume, slice thickness = 1.25 mm, in-plane = 1.25 × 1.25 mm. FA = 78, TR = 5520 ms, TE = 89.5 ms, FOV = 210 mm, matrix = 144 × 168, multiband factor = 3, GRAPPA = 2. Additionally, a reverse phase-encoded (blipped) non-diffusion weighted (bzero) volume was acquired. This HCP dataset will be referred to as dataset 4.

Image analysis

We processed the data using FSL version 6.0 (FMRIB Software Library, Oxford, UK). Data were first converted from DICOM to NIFTI format using dcm2nii (https://people.cas.sc.edu/rorden/mricron/dcm2nii.html), a b = 0 image was extracted from the main dMRI dataset and from the blipped dataset, and merged into a single file, a brain mask and the acquisition parameters file were created and processed in 'topup'. After this, the data were processed through MRtrix3 using dwidenoise [6], mrdegibbs [7], and dwibiascorrect [8], Dataset 1 was 640.2MB, dataset 2 was 367.7MB, dataset 3 was 723.5MB, and dataset 4 was 3.1GB. These were then processed using eddy_openmp, eddy_CUDA without and with slice-to-volume correction. For each run of eddy_openmp and eddy_cuda, the command 'time' was included to record the processing time. For dataset 1:

openmp

time eddy_openmp --imain=dwi.nii --mask=nodif_brain_mask --index=index148.txt --acqp=acqparams_148.txt --bvecs=bvec.bvec --bvals=bval.bval --fwhm=0 --topup=topup_b0_blips --flm=quadratic --out=eddy_unwarped_images_openmp -v

CUDA volume-to-volume

time eddy_cuda9.1 --imain=dwi.nii --mask=nodif_brain_mask --index=index148.txt --acqp=acqparams_148.txt --bvecs=bvec.bvec --bvals=bval.bval --fwhm=0 --topup=topup_b0_blips --flm=quadratic --out=eddy_unwarped_images_cuda -v

CUDA slice-to-volume

time eddy_cuda9.1 --imain=dwi.nii --mask=nodif_brain_mask --index=index148.txt --acqp=acqparams_148.txt --bvecs=bvec.bvec --bvals=bval.bval --fwhm=0 --topup=topup_b0_blips --flm=quadratic -- out=eddy_unwarped_images_cuda_mporder6 --mporder=6 -v

For dataset 2, the commands above were modified (imain=dwi_85.nii, acqp=acqparams_85.txt, bvecs=bvec85.bvec, bvals=bval85.bval), and for the ADNI and HCP data, these files were generated to end in 46 and 288, respectively. On computer 2, the above commands were changed so that eddy_cuda9.1 was replaced with eddy_cuda8.0.

To gauge GPU usage to determine the amount of GPU RAM and number of cores used, we used the following command to produce a running log every second:

nvidia-smi dmon -i 0 -s mu -d 1 -o TD >> cuda_resources.txt

To determine the extent to which hyperthreading could reduce processing time we ran every command twice: once with CPU hyperthreading and once without.

We used two computers for image processing: one had Ubuntu Linux 18.04 LTS (computer 1) and the other with 16.04 LTS (computer 2), to assess three implementations of eddy (eddy_openmp, eddy_CUDA without intra-volume correction, eddy_CUDA with intra-volume correction). Computer specifications are listed in Table 1.

Results

Processing time was highly dependent on the input dataset size, the available computing resources (Table 1) and the chosen configuration of eddy (Table 2). For example, among all tests performed, processing time ranged from 5 minutes (in dataset 3, using eddy CUDA on computer 1) to >7 hours (in dataset 4, using eddy CUDA slice-to-volume on computer 2).

We first compared processing times between the smallest and largest publicly available datasets (ADNI and HCP, datasets 3 and 4, being 723 and 3100 MB, respectively), to investigate the impact of file size (Figs. 1 and 2). As expected, this was large: using the same eddy CUDA volume-to-volume processing pipeline, the processing time for dataset 4 was 441% longer on computer 1 (9:38 in dataset 3 compared to 52:08 in dataset 4) and 900% longer on computer 2 (33:38 in dataset 3 compared to 201:05 in dataset 4). This corresponds to the difference in input file size, with dataset 4 being 328% larger than dataset 3, but implies greater gains in processing time are achieved on higher-specification hardware relative to size.

We then investigated the effect of slice-to-volume correction compared to the standard volume-to-volume correction. We found a consistent increase in processing time by a factor of 1.45–2.74. For example, using computer 1, slice-to-volume correction increased processing times relative to using volume-to-volume correction by 203% (13:04–39:24), 274% (5:13–19:31), 145% (9:38–23:34), and 160% (52:08–135:45), for datasets 1–4, respectively. The same tests on computer 2 yielded 200, 225, 205, and 234% longer processing times.

When comparing the effect of the computer hardware used, we found that, for volume-to-volume correction, computer 1 was 341, 401, 182, and 386% faster than computer 2 for datasets 1–4, respectively. When performing slice-to-volume correction, a similar magnitude of difference was observed (Table 2). When measuring the number of GPU cores recruited during processing, we found that, on computer 1 with the HCP dataset (dataset 4), up to 3750 cores were used most of the time with slice-to-volume eddy and only 1590 with volume-to-volume processing (Figs. 3 and 4). Computer 2 consistently used a greater proportion of its available cores. When processing

Table 1 Computer components

Computer	CPU	RAM	GPU	HDD
1 (Custom) MB = GA-X299 FSB = 2066 Chipset = Intel X299	Intel 7820x (3.6 GHz, 8 cores = 28.8 GHz), max turbo frequency = 4.30 GHz (x8 = 34.4 GHz), 11 MB L3 cache, memory bandwidth = 85 GB/s	128GB, PC4-3200	GV100 (32 GB HBM2 RAM, 5120 cores, 4096-bits memory interface width, 870 GB/s memory bandwidth, PCI-express 3.0)	1TB M.2 SSD (read/write speed of 3500/2500 MB/s)
2 (HP Z800) MB = Z800 FSB = 1333 MHz Chipset = Intel 5520	Intel x5650 x2 (2.66 GHz, 12 cores = 31.92 GHz), max turbo frequency = 3.07 GHz (x12 = 36.84), 12MB L3 cache, memory bandwidth = 32 GB/s	48GB, PC3-10600U/DDR1333)	Quadro K1200 (4GB GDDR5 RAM, 512 cores, 128-bit memory interface, 80 GB/s memory bandwidth, PCI-express 2.0)	4TB SAS (read/write speeds of approximately 500 MB/s)

CPU, central processing unit; GPU, graphics processing unit; HDD, hard disk drive; RAM, random access memory.

Table 2 Processing time

		eddy_openmp		eddy_cuda				
PC	Data	Time (m:s)	Max MD RAM Minus 2.7GB for PC 1, Minus 2.2GB for just eddy_openmp on PC 2	Time (m:s)	Max GPU RAM[a]	Max MD RAM	Mean (SD) % GPU cores used	Times faster compared to openmp
1 (with hyperthreading)	1	52:11	5.5GB (2.8GB openmp)	13:04	1608MB	1.8GB	31.6 (25.3)	3131s/784s = 3.99
sv				39:24	1623MB	1.8GB	39.7 (19.1)	3131/2364 = 1.32
	2	24:22	4.2GB (1.5GB openmp)	5:13	1988MB	1.1GB	26.9 (22.2)	1462/313 = 4.67
sv				19:31	1982MB	1.1GB	35.0 (20.2)	1462/1171 = 1.25
	3	45:39	15.6GB (13.9GB openmp)	9:38	1884MB	1.9GB	27.2 (25.7)	2739/578 = 4.74
sv				23:34	2441MB	1.9GB	40.7 (25.4)	2739/1414 = 1.94
	4	251:36	23.0GB (20.3GB openmp)	52:08	4409MB	11.3GB	30.5 (26.1)	15 096/3128 = 4.83
sv				135:45	4443MB	11.3GB	47.4 (25.8)	15 096/8145 = 1.85
			Minus 2.2GB for PC 1, Minus 1.3 GB for just eddy_openmp on PC 2					
(without hyperthreading)	1	63:19 (3799/3131 = 1.21; HT was 21% faster)	6.2GB (4.0 openmp)	13:04	1608MB	1.8GB	31.6 (25.3)	3799/784s = 4.85
sv				39:24	1623MB	1.8GB	39.7 (19.1)	3799/2364 = 1.61
	2	27:33 (1653/1462 = 1.13)	5.2GB (3.0GB openmp)	5:13	1988MB	1.1GB	26.9 (22.2)	1653/313 = 5.28
sv				19:31	1982MB	1.1GB	35.0 (20.2)	1653/1171 = 1.41
	3	47:03 (2823/2739 = 1.03)	10.0 (7.8GB openmp)	9:38	1884MB	1.9GB	27.2 (25.7)	2823/578 = 4.88
sv				23:34	2441MB	1.9GB	40.7 (25.4)	2823/1414 = 1.99
	4	323:15 (19 395/15 096 = 1.28)	17.3GB (15.1GB openmp)	52:08	4409MB	11.3GB	30.5 (26.1)	19 395/3128 = 6.20
sv				135:45	4443MB	11.3GB	47.4 (25.8)	19 395/8145 = 2.38
2 (with hyperthreading)	1	83:38 5018/3131 = 1.60 (PC1 is 60% faster)	8.0GB (6.7GB openmp)	44:31 (PC1 was faster by a factor of 3.41)	1296MB	1.8GB	66.9 (28.3)	5018/2671 = 1.88 (2671/784 = 3.41; PC1 was faster than PC2 by a factor of 3.41)
sv				133:43 (3.39)	1296MB	1.8GB	73.1 (23.8)	5018/8023 = 0.63 (8023/2364 = 3.39)
	2	38:34 (2314/1462 = 1.58; PC1 is 58% faster)	6.2GB (4.9GB openmp)	20:55 (4.01)	1038MB	1.1GB	77.2 (20.6)	2314/1255 = 1.84 (1255/313 = 4.0)
sv				68:04 (3.49)	1107MB	1.1GB	77.8 (19.2)	2314/4084 = 0.57 (4084/1171 = 3.49)
	3	77:40 (4660/2739 = 1.70; PC1 is 70% faster)	17.0 (15.7GB openmp)	33:38 (1.82)	1364MB	4.3GB	74.3 (27.4)	4660/2018 = 2.31 (2018/578 = 3.49)
sv				102:26 (1.96)	1895MB	4.3GB	80.5 (22.2)	4660/6146 = 0.76 (6146/1414 = 4.34)
	4	481:23 (25 103/15 096 = 1.66; PC1 is 66% faster)	24.4GB (23.1GB openmp)	201:05 (3.86)	2576MB	10.8GB	74.9 (28.7)	25 103/12 065 = 2.08 (12 065/3128 = 3.86)
sv				671:02 (4.94)	2733MB	10.9GB	83.3 (22.6)	25 103/40 262 = 0.62 (40 262/8145 = 4.94)
(without hyperthreading)	1	96:18 (5778/5018 = 1.15; HT was 15% faster)	5.4GB (3.2GB openmp)	As above	As above	As above	As above	5778/2671 = 2.16
sv				As above	As above	As above	As above	5778/8023 = 0.72
	2	42:32 (2552/2314 = 1.10)	4.2GB (2.0GB openmp)	As above	As above	As above	As above	2552/1255 = 2.03
sv				As above	As above	As above	As above	2552/4084 = 0.62
	3	83:04 (4984/4660 = 1.07)	11.6GB (9.4GB openmp)	As above	As above	As above	As above	4984/2018 = 2.47
sv				As above	As above	As above	As above	4984/6146 = 0.81
	4	568:00 (34 080/25 103 = 1.36)	20.8GB (18.6GB openmp)	As above	As above	As above	As above	34 080/12 065 = 2.82
sv				As above	As above	As above	As above	34 080/40 262 = 0.85

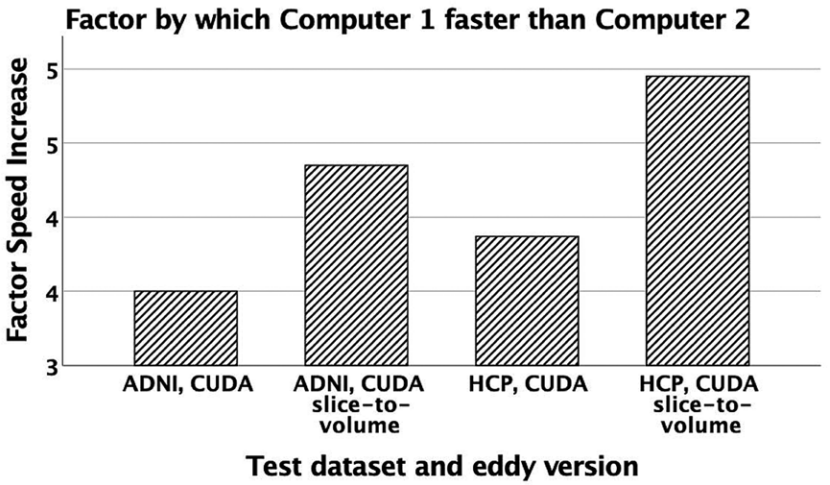
GB, gigabytes; GPU, graphics processing unit; HDD, hard disk drive; HT, hyperthreading; MB, megabytes; MD, motherboard; RAM, random access memory; sv, slice-to-volume.

[a]Including Xorg (~400MB) and compiz (~200MB). Numbers in parentheses represent time factor difference when computer 2 compared to computer 1, unless otherwise specified.

the ADNI dataset with volume-to-volume correction, up to 2700 GPU cores were used most of the time for computer 1 (~27%).

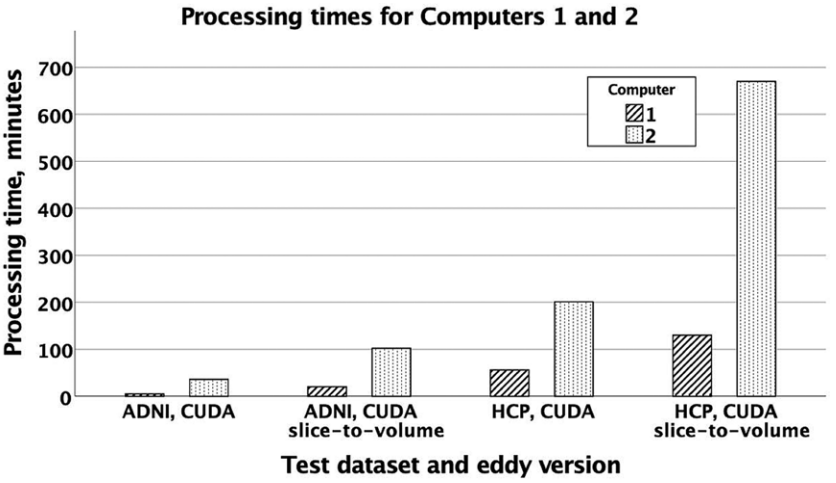
Finally, we tested the effect of hyperthreading on processing times. For eddy_openmp, using hyperthreading reduced processing times by 3–26% on computer 1, and 7–36% on computer 2. Additionally, eddy_openmp (with hyperthreading) was 58–70% faster on computer 1 when compared to computer 2. For eddy CUDA (both slice-to-volume and volume-to-volume iterations), hyperthreading made no difference to processing time. Using the eddy CUDA volume-to-volume command, processing times were greatly reduced compared to using eddy_openmp, depending on the number and speed of GPU cores available. As shown in Table 2, eddy CUDA (volume-to-volume) was 3.99–4.83 times faster than eddy_openmp (with hyperthreading) on computer 1, and

Fig. 1



Relative speed of the two computers when processing a small (ADNI) and large (HCP) dataset. s2v, slice-to-volume.

Fig. 2



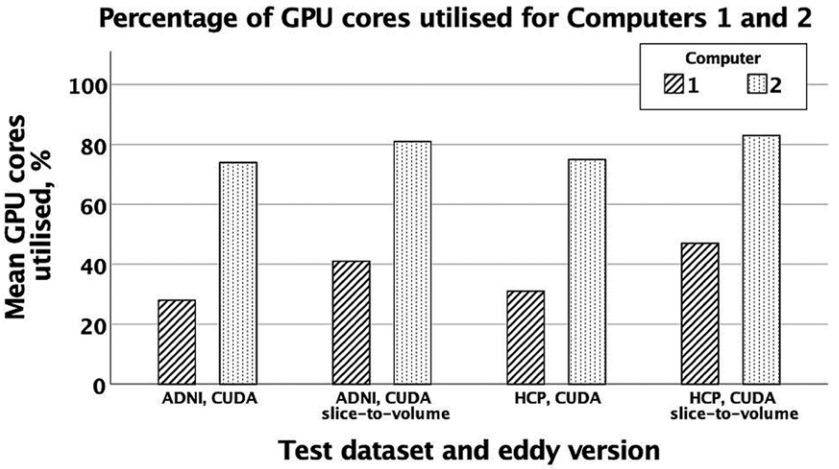
Time (minutes) for the two computers to process a small (ADNI) and large (HCP) dataset.

1.88–2.13 times faster on computer 2. For eddy_cuda (slice-to-volume), processing time was 1.25–1.94 times faster than eddy_openmp on computer 1. On computer 2, eddy_cuda (volume-to-volume) processing time was 1.84–2.31 times faster than eddy_openmp, and eddy_cuda (slice-to-volume) was between 0.57 and 0.76 of the speed – in other words, it was approximately 25–50% the speed of eddy_openmp.

Discussion

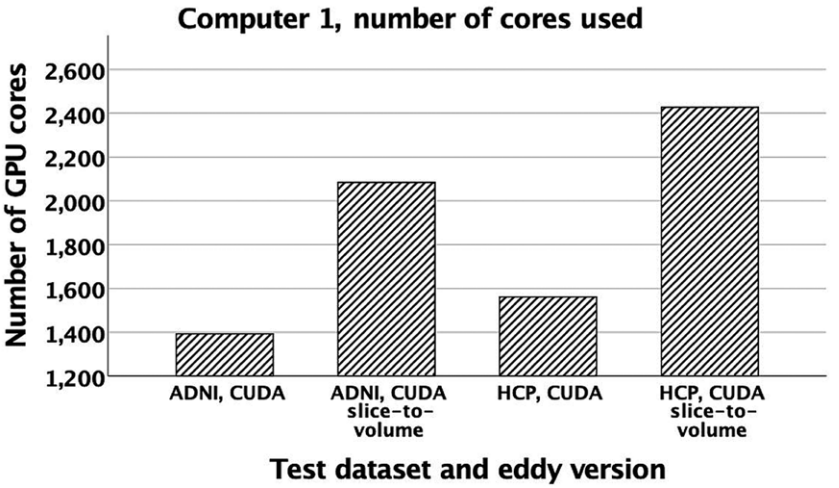
The capability of neuroimaging researchers to gather and store large quantities of high-resolution image data has recently undergone rapid growth. The advent of 'big data', high-performance computing and large-scale replication analyses means that it is increasingly important that the tools we use are efficient and make the best possible use of available resources. We present data comparing three different configurations of CUDA-enabled eddy correction software across four datasets and on two hardware configurations with different levels of performance. We make four key observations: (1) CUDA-enabled eddy correction is much faster than the multiprocessor configuration, even accounting for the improvement provided by hyperthreading, (2) processing time is directly proportional to the input file size regardless of hardware or software configuration, (3) slice-to-volume processing

Fig. 3



Percentage of GPU cores utilised during processing on both computers.

Fig. 4



Number of GPU cores used during processing on computer 1.

approximately doubles processing time, and (4) greater hardware investment provides greater relative value for larger datasets.

CUDA has been implemented for an array of MRI applications, including image reconstruction [9], image filtering [10], creation of fibre orientation distributions [11], and dMRI tractography [12]. Those studies demonstrated that, depending on GPU specifications and whether they are individual or as part of a cluster, gains in speed can be in the range of a 100-fold. Our study used a single GPU in each computer and showed substantial gains in processing time. We find that eddy correction processing using CUDA and the standard volume-to-volume approach was approximately 4–5 times faster than the multiprocessor version of eddy on a high-specification computer. On a more conventional computer, this difference was approximately two times. Since this mostly utilised the GPU (as only one CPU core was used), we expect that the largest explanatory factor is related to the graphic card specification. While almost 100% of the GPU cores on both computer graphic cards were used for brief moments, the mean was approximately 70% of the GPU cores used on computer 1 (which equates to 3584 cores) and around 90% of the GPU cores on computer 2 (which equates to 461 cores).

A second factor relates to the speed of the CPU, RAM, and hard drive. Computer 1 was up to 70% faster than computer 2 when using multiprocessor eddy; therefore, while the total CPU speed (GHz) appeared to be similar, the architectures (and therefore specifications) of the CPUs were very different (Table 1), the motherboard RAM ran at different speeds, and the read-write speeds of the hard drives were different. Collectively, this led to 70% greater processing speed for multiprocessor eddy correction. Based on our finding that hyperthreading improved processing time on both computers, we suggest that hyperthreading be activated when using eddy_openmp.

Based on these results and considering that high-resolution multi-shell dMRI data (e.g. from the HCP) are becoming more widely used, we suggest that a graphics card with more than 4.5GB RAM and 3750 CUDA cores and a motherboard with at least 32GB RAM should be used in order to optimise processing speed.

This study has several limitations. First, from each data source, we only tested an individual subject, which may limit the generalisability of our findings; however, in terms of processing times, our results were consistent between across datasets in the context of the various computer configurations. We felt that testing the procedures with larger samples was not required as we did not intend to generate exact differences between the different analysis techniques, but rather, to provide an indication of where the differences lie and how these will effect processing time. These differences were quite clear cut using even three datasets, hence using more datasets was unlikely to further highlight these differences. Specifically, using the CUDA version of eddy is much faster than using the non-CUDA versions, and even when using the optional flag to correct for slice-to-slice variation was still very fast. We also tested only three different hardware and software configurations which cannot necessarily be extended to others (e.g. comparing a more extensive collection of Nvidia graphics cards in terms of CUDA core number, speed, GPU RAM, and bandwidth). While we did not test more than one graphics card in a single computer, it is possible to install two identical Nvidia cards and link them via a 'Scalable Link Interface'. This would provide twice the number of GPUs available for CUDA-enabled eddy processing, and therefore potentially reduce processing time further. We also did not investigate the impact of optional flags in eddy; for example, for quality checking and outlier replacement (--cnr_maps to generate CNR maps, and --repol to replace outliers). These, and other optional flags, would require further processing time.

Using CUDA allows neuroscience questions to be answered more rapidly. In large samples, for example, a few hundred, using the CUDA version of eddy can complete the pre-processing in far less time, for example, days, or even weeks, than using the non-CUDA versions. Hence, data would be ready to be analysed earlier which would allow the neuroscience questions to be addressed more quickly. Furthermore, using the extra CUDA flag to correct for slice-to-slice motion results in more accurate motion correction which ultimately would translate to more accurate analyses to address the neuroscience questions.

We conclude that the CUDA implementation of eddy substantially reduces processing time for correction of dMRI data, even when performing slice-to-volume correction. The impact of GPU resources is large, and we have outlined recommended specifications for graphics cards and motherboard RAM necessary to efficiently reduce processing time. Given the large improvements in processing time and performance, it may be pertinent to create CUDA implementations of other common computing-intensive tasks in image processing.

Acknowledgements

We thank Jeff Macintosh and Trong Dang and the other staff at the MRI facility at Macquarie Medical Imaging (New South Wales, Australia). S.M.G. acknowledges the support of the Parker-Hughes Bequest and the Frecker Family Bequest.

J.J.M., S.M.G., S.J.V. and T.W. designed the experiment and collected the data. All authors processed the data and conducted the data analysis. All authors were involved in the design of the imaging protocol and data acquisition. All authors contributed to each stage of manuscript drafting.

The data that support the findings of this study are available from the Human Connectome Project and the Alzheimer's Disease Neuroimaging Initiative but restrictions apply to the availability of these data, which were used under license for the current study, and so are not publicly available. Results are however available from the authors upon reasonable request.

Conflicts of interest

There are no conflicts of interest.

References

1 Gallichan D, Scholz J, Bartsch A, Behrens TE, Robson MD, Miller KL. Addressing a systematic vibration artifact in diffusion-weighted MRI. *Hum Brain Mapp* 2010; **31**:193–202.

2 Andersson JL, Skare S, Ashburner J. How to correct susceptibility distortions in spin-echo echo-planar images: application to diffusion tensor imaging. *Neuroimage* 2003; **20**:870–888.

3 Andersson JLR, Sotiropoulos SN. An integrated approach to correction for off-resonance effects and subject movement in diffusion MR imaging. *Neuroimage* 2016; **125**:1063–1078.

4 Andersson JLR, Graham MS, Drobnjak I, Zhang H, Filippini N, Bastiani M. Towards a comprehensive framework for movement and distortion correction of diffusion MR images: within volume movement. *Neuroimage* 2017; **152**:450–466.

5 Bastiani M, Cottaar M, Fitzgibbon SP, Suri S, Alfaro-Almagro F, Sotiropoulos SN, *et al*. Automated quality control for within and between studies diffusion MRI data using a non-parametric framework for movement and distortion correction. *Neuroimage* 2019; **184**:801–812.
6 Veraart J, Novikov DS, Christiaens D, Ades-Aron B, Sijbers J, Fieremans E. Denoising of diffusion MRI using random matrix theory. *Neuroimage* 2016; **142**:394–406.
7 Kellner E, Dhital B, Kiselev VG, Reisert M. Gibbs-ringing artifact removal based on local subvoxel-shifts. *Magn Reson Med* 2016; **76**:1574–1581.
8 Smith SM, Jenkinson M, Woolrich MW, Beckmann CF, Behrens TE, Johansen-Berg H, *et al*. Advances in functional and structural MR image analysis and implementation as FSL. *Neuroimage* 2004; **23** (Suppl 1):S208–S219.
9 Wang H, Peng H, Chang Y, Liang D. A survey of GPU-based acceleration techniques in MRI reconstructions. *Quant Imaging Med Surg* 2018; **8**:196–208.
10 Chang HH, Li CY. An automatic restoration framework based on GPU-accelerated collateral filtering in brain MR images. *BMC Med Imaging* 2019; **19**:8.
11 Hernández M, Guerrero GD, Cecilia JM, García JM, Inuggi A, Jbabdi S, *et al*. Accelerating fibre orientation estimation from diffusion weighted magnetic resonance imaging using GPUs. *PLoS One* 2013; **8**:e61892.
12 Hernandez-Fernandez M, Reguly I, Jbabdi S, Giles M, Smith S, Sotiropoulos SN. Using GPUs to accelerate computational diffusion MRI: From microstructure estimation to tractography and connectomes. *Neuroimage* 2019; **188**:598–615.