Computer Methods and Programs in Biomedicine 207 (2021) 106143

Contents lists available at ScienceDirect

Computer Methods and Programs in Biomedicine

journal homepage: www.elsevier.com/locate/cmpb

A clinically viable vendor-independent and device-agnostic solution for accelerated cardiac MRI reconstruction

Elena Martín-González[a], Elisa Moya-Sáez[a], Rosa-María Menchón-Lara[a], Javier Royuela-del-Val[c], César Palencia-de-Lara[b], Manuel Rodríguez-Cayetano[a], Federico Simmross-Wattenberg[a,*], Carlos Alberola-López[a]

[a] Laboratorio de Procesado de Imagen (Image Processing Laboratory), Universidad de Valladolid, Valladolid 47011, Spain
[b] Departamento de Matemática Aplicada, Universidad de Valladolid, Valladolid, Spain
[c] Health Time Group, Córdoba, Spain

ARTICLE INFO

Article history:
Received 18 December 2020
Accepted 25 April 2021

Keywords:
Magnetic resonance imaging
GPU
OpenCLIPER
OpenCL

ABSTRACT

Background and objective: Recent research has reported methods that reconstruct cardiac MR images acquired with acceleration factors as high as 15 in Cartesian coordinates. However, the computational cost of these techniques is quite high, taking about 40 min of CPU time in a typical current machine. This delay between acquisition and final result can completely rule out the use of MRI in clinical environments in favor of other techniques, such as CT. In spite of this, reconstruction methods reported elsewhere can be parallelized to a high degree, a fact that makes them suitable for GPU-type computing devices. This paper contributes a vendor-independent, device-agnostic implementation of such a method to reconstruct 2D motion-compensated, compressed-sensing MRI sequences in clinically viable times.

Methods: By leveraging our OpenCLIPER framework, the proposed system works in any computing device (CPU, GPU, DSP, FPGA, etc.), as long as an OpenCL implementation is available, and development is significantly simplified versus a pure OpenCL implementation. In OpenCLIPER, the problem is partitioned in independent black boxes which may be connected as needed, while device initialization and maintenance is handled automatically. Parallel implementations of both a groupwise FFD-based registration method, as well as a multicoil extension of the NESTA algorithm have been carried out as processes of OpenCLIPER. Our platform also includes significant development and debugging aids. HIP code and precompiled libraries can be integrated seamlessly as well since OpenCLIPER makes data objects shareable between OpenCL and HIP. This also opens an opportunity to include CUDA source code (via HIP) in prospective developments.

Results: The proposed solution can reconstruct a whole 12–14 slice CINE volume acquired in 19–32 coils and 20 phases, with an acceleration factor of ranging 4–8, in a few seconds, with results comparable to another popular platform (BART). If motion compensation is included, reconstruction time is in the order of one minute.

Conclusions: We have obtained clinically-viable times in GPUs from different vendors, with delays in some platforms that do not have correspondence with its price in the market. We also contribute a parallel groupwise registration subsystem for motion estimation/compensation and a parallel multicoil NESTA subsystem for $l1 - l2$-norm problem solving.

* Corresponding author.
E-mail addresses: emargon@lpi.tel.uva.es, marcma@lpi.tel.uva.es (E. Martín-González), emoysae@lpi.tel.uva.es (E. Moya-Sáez), rmenchon@lpi.tel.uva.es (R.-M. Menchón-Lara), j.royuela.v@htime.org (J. Royuela-del-Val), palencia@mac.uva.es (C. Palencia-de-Lara), manrod@lpi.tel.uva.es (M. Rodríguez-Cayetano), fedsim@tel.uva.es, fedesim@lpi.tel.uva.es (F. Simmross-Wattenberg), carlos@lpi.tel.uva.es (C. Alberola-López).

https://doi.org/10.1016/j.cmpb.2021.106143

1. Introduction

Magnetic resonance (MR) is a high-quality and highly versatile medical imaging modality. Not only anatomical images with different contrasts can be acquired, but also a number of properly-designed pulse sequences provide additional information, such as diffusion, perfusion, fMRI and a few others. Dynamic imaging is also possible and MR currently constitutes the golden standard for functional studies of the heart [1]. However, its major drawback is its inherent slowness relative to other alternatives such as CT and US. A complete study in MRI may take 30–45 min, whereas CT takes a few minutes and US is observed in real time. This has the consequence of patient discomfort, which leads to motion artifacts in the images, the need of synchronization for dynamic studies and poses additional difficulties for imaging non-cooperative patients, such as children or the elderly.

This being the case, the MR community is doing a considerable effort to increase the velocity of MR acquisition and to diminish the need of additional hardware (such as navigators, ECG or PPG synchronization, etc.). Apart from improvements in the hardware of the scanners —that provide higher and more homogeneous fields, more intense and faster gradients, etc.—, many different software solutions have been described, which include parallel imaging (PI) and compressed sensing (CS). In terms of reconstruction [2], different algorithms are capable of reconstructing images from undersampled k-space information; leaving aside modern deep learning-based approaches, for which their bottleneck is the training stage, classical algorithms are optimization-based and they may be computationally demanding, depending on their complexity. Hence fast implementations and appropriate computing hardware make a difference so as to obtain solutions in clinically viable times. Here, clinical viable time is interpreted as the time taken to reconstruct an image that is compatible with repeating an acquisition —whenever the reconstruction is considered of insufficient quality— without causing an unacceptable overload in clinical routine[1]. The MR reconstruction workload turns out to be highly parallelizable so proper implementations on natively-parallel devices, such as (but not limited to) GPUs, are currently playing an outstanding role.

Many different attempts have been reported to speed up reconstruction algorithms by means of parallel devices. Section 2 provides an overview of the field. As of today, the field of parallel computing has seen a tremendous growth of tailored solutions that work exclusively on a particular vendor GPU, namely, nVidia. Nevertheless, other alternatives exist. One of them is OpenCL [4]; OpenCL is an open standard for parallel computing, similar in its objectives to the nVidia-exclusive language CUDA, but covers a much wider set of computing devices (including nVidia GPUs) while ensuring that source code and data are unique, thus avoiding the need for code and data replication among prospectively supported devices. However, it is generally regarded as more complicated and less mature than CUDA. As a matter of fact, OpenCL programmers must deal on their own with device selection and initialization, memory management, kernel loading and compilation, host-device interaction, and administration overload.

Another example is HIP [5]; this initiative helps developers in converting CUDA source code into a more portable form, so that it can run on both nVidia and AMD GPUs. However, as of today, the HIP API is not compatible with OpenCL, so data objects defined in OpenCL cannot be used as parameters to HIP kernels and vice-versa, even though OpenCL data is undistinguishable from HIP (or CUDA) data in the GPU memory.

The framework OpenCLIPER [6] was developed to address all these shortcomings of OpenCL. In this paper, OpenCLIPER has been used to provide a parallel implementation of our previous proposal [7] on 2D cardiac cine imaging. This algorithm combines CS with Groupwise Motion Compensation (MC) to achieve CINE sequences with acceleration factors (AFs) of 8–15x, which show higher quality than other methods with pairwise approaches for motion compensation. Nevertheless, in its non-parallel version, the post-acquisition reconstruction takes a significant computation time (about 40 min in a Core i7-4790 CPU for the full heart coverage), which is far from being clinically viable. In this paper, we show how OpenCLIPER eases dramatically the process of parallelizing a complex algorithm previously programmed in a scripting language.[2] As a byproduct, we have enlarged the OpenCLIPER library with additional parallel functionality, such as a groupwise version of a registration algorithm based on free form deformations (FFD) [8] or a multicoil parallel adaptation of the NESTA optimization algorithm [9]. None of these algorithms, to the best of our knowledge, are currently available on GPUs. A comparison with another popular framework (BART, see Section 2) is included. In addition, we show how to seamlessly incorporate HIP code to OpenCLIPER by solving the integration difficulty referred to above; this opens up the possibility of using CUDA code in our framework —as long as this code can be converted to HIP— at programmer will.

The rest of this paper is organized as follows: Section 2 reviews the subject of current parallel implementations of undersampled MRI reconstruction algorithms. Section 3 describes the innards of our proposed implementation. Section 4 evaluates our proposal performance in various computing devices and compares it, as previously mentioned, with another popular framework. The discussion can be found in Sections 5 and 6 concludes the paper. A number of appendices of both mathematical and algorithmic details have been included for the sake of self-completeness.

2. Related work

The field of MR reconstruction is very active and quite an activity has been focused on algorithm implementation on GPUs. Two recent survey papers [10,11] give a general overview of the field; the former is dedicated to GPU-based medical image reconstruction while the latter is specifically targeted to MR reconstruction. Both survey papers, however, have a similar structure as for their common topic; they review FFT-based methods, either for Cartesian and non-Cartesian data, PI and CS-based applications. The 2018 survey also includes a section on reconstruction based on deep-learning, probably spurred by the enormous activity that has been reported in the last two years. Both papers also quote some recent work on multi-GPU solutions. The reader is kindly invited to consult the references therein.

Apart from particular implementations of reconstruction algorithms, the field has been enriched with a number of libraries, some examples of which are AGILE [12], BART [13], Gadgetron[14] and Impatient [15], to explicitly mention four of them. Additional libraries are mentioned in our former publication on OpenCLIPER [6]. These libraries are conceived to ease the process of algorithm prototyping and testing; Gadgetron shows additional client-server characteristics, where the server takes care of the reconstruction process while the client focuses on data handling, i.e., Gadgetron could be considered both as a library and as a network service.

The common ground of these references, both in terms of specific algorithms and of reconstruction toolkits, is that they are based on the CUDA language and, consequently, they are vendor

[1] Issues related to DICOM interoperability are a step further and well-known solutions are available [3]. We concentrate on operations on the reconstruction problem once all the necessary data is available.

[2] OpenCLIPER code available at http://opencliper.lpi.tel.uva.es/ .

dependent. nVidia, since its onset in 2007 with the release of its CUDA application programming interface [16], has become the main supplier in the field of parallel computing with GPUs, both in academia and in industry. Its mainstream condition is accompanied, however, with some unavoidable consequences which can be summarized in two, namely, the need of code duplication should the application be run in GPU and in CPU (or in any other device), and the need of data replication in the platforms in which the application is deployed.

Alternatives to this technology are also available, mainly, Direct Compute issued by Microsoft and, as previously stated, OpenCL, an open source project led by the Khronos Group [11]. The latter has the benefits of being both open-source and device agnostic; the second feature makes it comply with the write-once run-anywhere (WORA) paradigm. An additional solution has been proposed, which is a domain specific computational image reconstruction language, referred to as *Indigo* [17]. It provides a front-end with a number of image reconstruction facilities, as well as a back-end that is responsible to evaluate the operators provided by the front-end on a particular platform. As long as different back-ends are available, the language will follow the WORA paradigm, although as of today it seems limited to CUDA as for GPU implementation.

As we mentioned in the Introduction, in this paper we illustrate how OpenCLIPER can be used to find a solution to a 2D dynamic cardiac image reconstruction algorithm, that makes whole-heart single breath-hold reconstruction possible [18] in clinically feasible times. Our algorithm makes use of an optimizer, for which we have used NESTA [9] as well as FFD [8] for motion compensation. Further algorithmic details of the method are described elsewhere [7,19]. As for the NESTA algorithm, we are only aware of a GPU version, which was recently described in Dinh et al. [20]; however, in that implementation, a single coil is apparently used. In our implementation, multiple coils are employed and the algorithm is fully parallel in the coils as well. As for the FFDs, some GPU-based implementations were described in Modat et al. [21], Ruijters et al. [22], Du et al. [23], Ellingwood et al. [24], Punithakumar et al. [25]; however, these proposals are pairwise and device-specific. We show in Royuela-del Val et al. [7] that a groupwise registration has added benefits in terms of reconstruction quality; consequently, the implementation we have carried out is groupwise and specifically targeted for cardiac cine MRI.

3. The proposed system

3.1. The algorithm

CINE cardiac MR images can be reconstructed from highly undersampled data while keeping a very high level of detail from the fully-sampled images. A state-of-the-art algorithm on this topic was published in Royuela-del Val et al. [7] and later enhanced in Royuela-del Val et al. [19]. Briefly described, the algorithm departs from the multi-coil k-space subsampled information b and solves a CS reconstruction problem which gives the resulting image sequence m_i, at iteration step i, as

$$m_i = \arg\min_m ||b - Em||_{l_2}^2 + \lambda ||\Phi \mathcal{T}_\Theta m||_{l_1} \quad (1)$$

m_i is a vectorized stack of image frames, one frame per cardiac phase. E is the encoding operator that includes the multiplication by the coil sensitivities S, the intra-frame spatial Fourier transform $\mathcal{F}$ and the application of the undersampling mask A. Φ is the temporal total variation (tTV) operator. $\mathcal{T}_\Theta$ is the groupwise (GW) transformation for motion compensation (MC) that registers all the frames in the sequence to a common reference and λ is a regularization parameter. Appendix A describes a matrix formalization of these data structures and operators. As for m_0, the transformation $\mathcal{T}_\Theta$ is the identity, since there is no data from which motion information can be estimated. Then, the iterative refinement process finds m_i with $\mathcal{T}_\Theta$ obtained by the heart motion estimation (ME) on m_{i-1}. The whole process continues until a predefined number of iterations is met. In this manuscript, the number of ME/MC reconstruction iterations was set to two.

The ME/MC is carried out by means of elastic registration [26]. As previously stated, we use FFDs [8]; Appendix B provides details on the ME/MC process. B-splines serve the purpose of interpolating the deformation field (see Eq. (B.3)) from a given control point configuration to give rise to a dense field. Specifically, we apply a GW paradigm in which no particular frame is selected as a reference, but the reference is built along the optimization process as the average of the transformed images. As for the metric to be optimized, we have used the sum of the intensity square differences (SSD) with respect to the reference image (see Eq. (B.1)); this metric is enlarged with smoothness terms so as to force a realistic motion field solution (see Eq. (B.2)).

Eq. (1) is minimized with the well-known NESTA algorithm based on Nesterov's method [9]. A detailed pseudocode of the MRI reconstruction algorithm can be found in Appendix C. Specifically, Nesterov's method iteratively minimizes a function f by estimating three sequences x_k, y_k and z_k. The x_k sequence corresponds to the sequence we want to estimate (m_i in the Eq. (1)), and it is obtained from a weighted sum of the other two sequences. Nesterov's method can be used for the minimization of both smooth and nonsmooth convex functions if using the appropriate smoothing techniques. In [9], the $l1$-norm in Eq. (1) is component-wise approximated by the well-known Huber function $f_\mu(x)$, which depends on a smoothing parameter μ; this parameter is iteratively decreased during the optimization process. The starting guess x_0 is the result of applying the adjoint encoding operator (E^H) to the subsampled k-space ($x_0 = E^H b$).

3.2. Parallel implementation

Our parallel implementation is coded in OpenCL [4], and all parallel operations are programmed as OpenCL kernels. In the next two subsections we provide details on the ME/MC stage and the NESTA algorithm. Pseudocode for the latter is included in Appendix C.

3.2.1. Groupwise registration

The registration procedure is determined by the values of the parameters θ_u in Eq. (B.3). These parameters are obtained by minimizing the metric defined in Eq. (B.2). In this section we will concentrate on the implementation of the most resource-demanding operation, which turns out to be finding the gradient of Eq. (B.1). The gradient is, in essence, calculated as follows:

$$\frac{\partial V(x)}{\partial \theta} = \frac{\partial V(x)}{\partial m} \cdot \frac{\partial m}{\partial \mathcal{T}(x)} \cdot \frac{\partial \mathcal{T}(x)}{\partial \theta} \quad (2)$$

where $V(x)$ is the metric, m represents the image sequence and $\mathcal{T}(x)$ the transformation. Notice that x will be a grid point (with coordinates given as row and column numbers r, $1 \le r \le N_1$, and c, $1 \le c \le N_2$, respectively). In addition, each parameter θ is associated to each of the frames (with index n, $1 \le n \le N_t$) as well as to each of the control points (with coordinates, say (r_u, c_u)) and each of the directions of variation, namely, horizontal (index $l = 1$) and vertical ($l = 2$); these indices will be borne in mind for the derivatives.

The first factor in equation Eq. (2) can be written:

$$\frac{\partial V(x)}{\partial m}_{r,c,n} = \frac{2}{N_t}\left(m^{(n)}\left(\mathcal{T}^{(n)}(x)\right)_{r,c} - \frac{1}{N_t}\sum_{n'=1}^{N_t} m^{(n')}\left(\mathcal{T}^{(n')}(x)\right)_{r,c}\right)$$

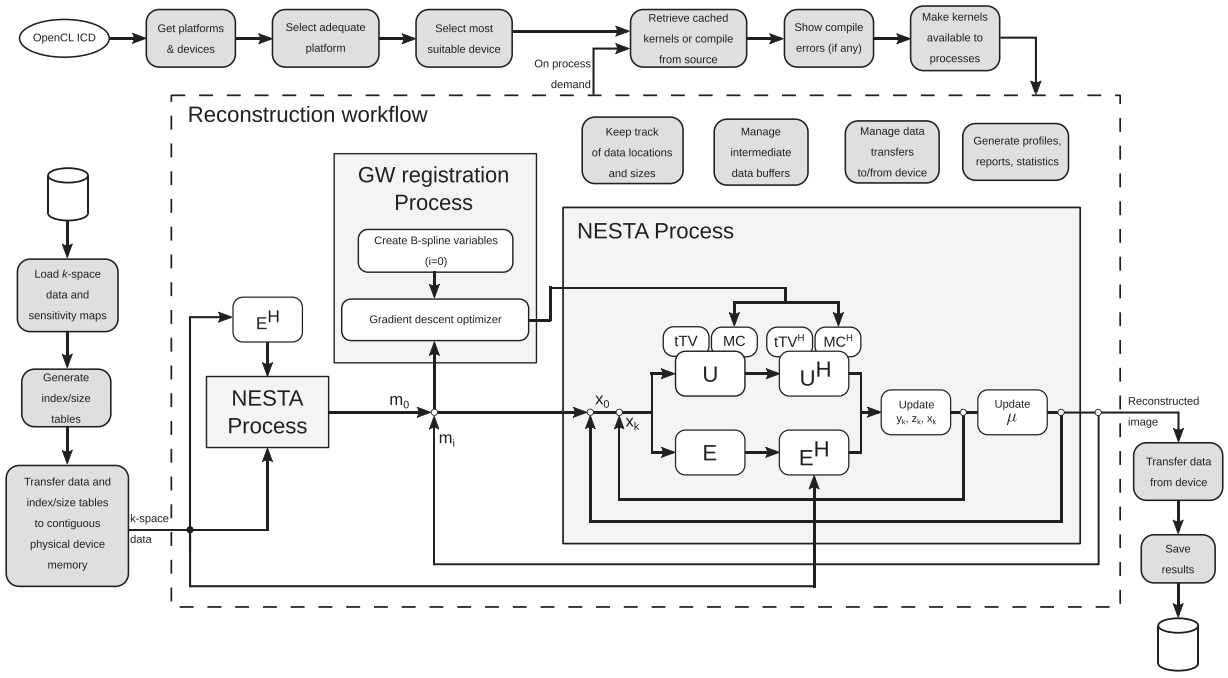


Fig. 1. Functional units of the proposed reconstruction system. Gray boxes represent necessary albeit unproductive work, which is taken care of by OpenCLIPER. White boxes represent actual, productive work. Each white box is abstracted as a *process*, which may be composed of several other processes. U, U^H, E, E^H are the internal NESTA operators. tTV: temporal Total Variation; GW: Groupwise Registration.

with $m^{(n)}(x)$ the nth frame in sequence m, evaluated at pixel x, and $\mathcal{T}^{(n)}(x)$ it the transformed position of pixel x in that phase. $m^{(n)}$ will be also referred to as the nth frame. This equation is computed in parallel for the $N_1 \times N_2 \times N_t$ pixels.

The second factor in Eq. (2) is the image gradient with respect to the transformation. The actual operations we perform are the interpolation of the gradient of each frame $m^{(n)}$ at position $\mathcal{T}^{(n)}(x)$. The interpolation is carried out within a region of interest, say χ_m, so the number of operations launched in parallel is $N_{1\chi_m} \times N_{2\chi_m} \times N_t$, where the two first factors are the dimensions of region χ_m.

The last factor in Eq. (2) is the gradient of the transformation with respect to the variables θ_u to be optimized; since these variables enter Eq. (B.3) as factors, their derivative is straightforward and can be precomputed.

Hence, the calculation of the gradient of Eq. (2) will be performed parallelizing along the rows and columns affected by the B-splines (r_s and c_s), the frame dimension (n), the rows and columns of the control points (r_u and c_u) and the spatial dimension (l), as can be seen below:

$$\frac{\partial V(x)}{\partial \theta}_{r_s,c_s,n,r_u,c_u,l} = \frac{\partial V(x)}{\partial m}_{r_s,c_s,n,l} \cdot \frac{\partial m}{\partial T(x)}_{r_s,c_s,n,l} \cdot \frac{\partial T(x)}{\partial \theta}_{r_s,c_s,r_u,c_u}.$$

3.2.2. NESTA

A matrix formalism of data structures and operators (the encoding operator E and the sparsifying operator U) used in NESTA is presented in Appendix A; this formalism makes it simple to derive the adjoint operators. However, in practice, high data dimensionality implies that storage of matrices associated with operators is not feasible due to memory requirements. Consequently, operators are implemented by means of functions and the latter have been the focus of our parallelization effort. Specifically, for the encoding operator E the following parallelizations were made: 1) Multiplication by the coil sensitivities S is performed parallelizing along the spatial dimensions (i.e., $N_1 \times N_2$ operations launched in parallel). 2) The by-frame spatial Fourier transform $\mathcal{F}$ is performed parallelizing along the coils and frames dimensions (i.e., $C \times N_t$ parallel operations) using the clFFT [27] library. 3) Application of the undersampling mask A is performed parallelizing along the spatial dimensions (i.e., $N_1 \times N_2$ parallel operations). Note that in S, the coils and frames dimensions could have also been parallelized; nevertheless, we empirically verified that this did not constitute a performance gain, presumably caused by the limited number of cores in the hardware. As for the adjoint encoding operator E^H, the following parallelizations were made: 1) Multiplication by coil conjugate sensitivities S^H and, 2) by-frame spatial inverse Fourier transform $\mathcal{F}^{\mathcal{H}}$, Both of them were parallelized analogously to their counterparts S and $\mathcal{F}$, respectively. 3) Summation of the resulting image sequence in each coil is parallelized along the spatial and temporal dimensions (i.e., $N_1 \times N_2 \times N_t$ operations launched in parallel).

Regarding the sparsifying operator $U = \Phi\mathcal{T}_\Theta$, the temporal total cyclic variation Φ is performed parallelizing along the spatial dimensions (i.e., $N_1 \times N_2$ operations launched in parallel) and the $\mathcal{T}_\Theta$ groupwise transformation for MC is performed parallelizing along both the spatial and the frame dimensions (i.e. $N_{1\chi_m} \times N_{2\chi_m} \times N_t$ parallel operations). Similarly, the adjoint sparsifying operator U^H is parallelized along the same dimensions as U.

Finally, matrix operations needed in NESTA, such as scaling a vector by a constant or addition of two vectors, are performed efficiently by the clBLAST library [28]. Thus, all the algorithm steps (see Appendix C for details) can be straightforwardly computed by combining operators E and U, their adjoints E^H and U^H, and a few matrix operations.

3.3. Algorithm implementation on a generic parallel device

We have translated the original method into an actual device-agnostic software solution which (a) finishes in clinically viable times, comparable to other popular reconstruction frameworks, (b) is suitable for execution in parallel devices, and (c) complies with the WORA paradigm (see Section 2) so neither code nor data

need be duplicated for CPU/GPU or any other device. As stated in Section 1, we make extensive use of our framework OpenCLIPER to provide this solution.

In this section we provide details on this implementation; Fig. 1 shows a diagram of our system functional units that will guide us through the description that follows.

3.3.1. Generalities

Together with the core algorithm —enclosed within the dashed rectangle in Fig. 1 and its operations represented by rectangles with white background—, several other tasks must be carried out for the system to be functional by itself. These tasks are not only part of the initialization process but they are also active while the actual processing is taking place. All these administrative tasks (shaded boxes in Fig. 1) are dealt with by OpenCLIPER with minimal programmer intervention. While the system is at work, data passes through several transformations (called *processes* in OpenCLIPER) that need to be connected appropriately. Each process may be seen as a black box with a single input, a single output and an arbitrary set of parameters. This specification allows prospective programmers to interconnect and compose processes arbitrarily, as long as outputs from one are compatible with inputs to the next, since their interfaces must all comply with this specification.

An appropriate computing device must be selected (top row in Fig. 1). Since OpenCL supports a wide range of them, an additional workload is the need to detect the available platforms and devices (which may be from different vendors), and to choose the most suitable among them. Moreover, once a device has been selected, kernels must be loaded and compiled for the chosen device. OpenCLIPER simplifies these tasks significantly: kernel loading and compiling is done without user intervention when a process demands it, caching them as necessary so time is dedicated to compilation only once per device (and driver version). Platform/device detection and selection may be either specified by the programmer (via hints such as device type, model, supported CL version, etc.) or, alternatively, left to the framework by just a single line of code. In the latter case —or, in the former, whenever the specified hints match several candidate devices—, the *a priori* fastest device is automatically chosen.

All compute-intensive transformations are performed in the computing device, so the system takes full advantage of its parallel capabilities and hence clinically viable execution times may be achieved (see quantification in Section 4). The algorithms implemented in OpenCLIPER, which are themselves processes, are made up from several other processes which may be reused at programmer discretion. In this sense, OpenCLIPER provides a pool of frequently used processes as a ready-to-use toolbox; for example, Hadamard matrix multiplication, fast Fourier transform, channel integration, image sum, etc. These tools make the implementation of more involved processes —such as the two described in Section 3.2— easy.

With respect to the four blocks on top of the NESTA Process in Fig. 1, after the selection of the most appropriate device (either manually or automatically), kernel loading is delayed until a process requires it. At that time, a previously compiled (i.e. cached) version of the kernel is sought. If it does not exist yet, kernel compilation is automatically triggered and the kernel cache is updated (loading of cached kernels is typically two orders of magnitude faster than on-the-fly compilation). Compilation logs (if any) are shown to the user for debugging purposes.

3.3.2. Input and output

MRI data loading is done in a single code line with OpenCLIPER by a front-end function (see leftmost side of Fig. 1). As of today, k-space lines are input by means of a.clf file (and its accompanying.hdr file); Matlab format file is also allowed. Output formats include the two formats just mentioned as well as other popular image formats (JPEG, PNG, etc).

To maximize utilization of the computing device, data objects may be loaded and saved concurrently while the device is busy processing other objects. Since MRI data files are often large, this can save a noticeable amount of processing time per patient.

3.3.3. Data structures

A common burden in GPU computing lies in the need to keep track of pointers to data objects and their properties: number of dimensions, sizes and strides along each of them, data types and sizes (complex, float, integer, etc.), and passing them to the computing device. This is typically done by adding arguments to the kernels, but kernel argument space is usually limited by the computing hardware (apart form adding more burden for the user to call their kernels). OpenCLIPER simplifies this burden by encapsulating all data properties within the buffer in the computing device in a way that is transparent to the kernel programmer (i.e. there is no offset between the object pointer and the real data). Thus, users just have to pass a single pointer to the data object to have all its properties readily available for the kernel code. OpenCLIPER provides support for data with an arbitrary number of frames, coils, sensitivity maps, sampling masks and data dimensionality for the special case of MRI data, and arbitrarily complex data for the general case. Our framework also provides methods to traverse data buffers along any given dimension. This functionality is provided by generating index and size tables for each data object, as shown in Fig. 1.

An additional feature of OpenCLIPER is that all sub-objects in a data object (e.g. sensitivity maps for each coil) are mapped contiguously in device memory and properly adjusted to hardware alignment, so kernel programmers can assume a linear layout when processing compound data objects. Data transfers between host and device are driven by the DMA controller to maximize speed.

Listing 1 shows a simple OpenCL kernel which sums a set of MR images along the coil dimension. It can be seen how a kernel can access attributes from input and output buffers (such as coil or frame strides, sizes, and so on) just by passing their natural pointer as arguments with no need to worry about header offsets or the like. All attribute access functions are $O(1)$ as long as all ND

```
#include <OpenCLIPER/OpenCLIPERDataModelCommonDefs.hpp>
#include <OpenCLIPER/kernels/hostKernelFunctions.h>

kernel void xImageSum_kernel(global realType* pInBuffer,
     global realType* pOutBuffer) {
  uint inOffset = get_global_id(0);
  uint outOffset = get_global_id(0);
  uint inCoilStride = getCoilStride(pInBuffer,0);
  uint outFrameStride = getTemporalDimStride(pOutBuffer,0,0);
  uint nInFrames = getTemporalDimSize(pInBuffer, 0);
  uint nCoils = getNumCoils(pInBuffer);

  realType accum;
  for(uint frame=0; frame<nInFrames; frame++) {
    accum=0;
    for (uint coil = 0; coil < nCoils; coil++) {
      accum+= pInBuffer[inOffset];
      inOffset+= inCoilStride;
    }

    pOutBuffer[outOffset] = accum;
    outOffset+= outFrameStride;
  }
}
```

Listing 1. Accessing attributes of complex data structures. Note how data properties (sizes, strides and number of coils in this particular case) are available from kernel code without passing them explicitly as kernel arguments.

arrays in the given object have the same number of dimensions ($O(n)$ otherwise).

3.3.4. Additional support for researchers

OpenCLIPER has also been designed as a development tool for researchers. Hence, it includes a number of additional utilities, which we now enumerate:

1. 1D, 2D and 3D objects in device memory, either standalone or as a movie, can be graphically presented to the user at any time by a simple call to `pointerToObject->show()` method; scaling and video velocity facilities are provided at programmer convenience. OpenCV [29] is currently used to display windows on the desktop.
2. Similarly, any object in device memory can be saved for further inspection in Matlab or any other supported format at any time.
3. When compiling in debug mode, device and host are automatically synchronized in every process so that run-time errors trigger exactly at the responsible kernel, so fault location detection should be immediate.
4. When OpenCL kernels are compiled on the fly (as opposed to loading a previously cached version), errors may show up when the host program is run. If this is the case, kernel compilation errors are automatically shown to the user.
5. In the case of program abortion, the full stack backtrace is shown to the programmer.
6. OpenCLIPER includes facilities to gather profiling data and generate statistical reports as well.
7. A memory map of data objects in device memory may be obtained at any time.

3.3.5. Incorporation of HIP and CUDA functionality

As stated in the introduction, HIP [5] is an interesting effort to bridge the gap between GPUs vendors by providing facilities to migrate CUDA code into a reusable form in other platforms. However, handling exchange is still an issue; specifically, data objects defined in one language (HIP/OpenCL) cannot be used as parameters to kernels written in the other language (OpenCL/HIP). We have solved this problem by providing a HIP handle to every OpenCL data object defined through the OpenCLIPER API. This way, developers may invoke calls to HIP libraries on pure OpenCL data objects and, consequently, OpenCLIPER may benefit from libraries written in CUDA for which a HIP implementation is also available.

As an example, Listing 2 shows how an OpenCLIPER data object may be passed seamlessly to the rocFFT library [30] and use its result as an OpenCLIPER object again. After initialization (lines 11–15), a vector of complex numbers is created as an OpenCL memory object (lines 17–20) and its corresponding HIP handle is obtained (line 23). The next code section is pure HIP code in which the obtained handle is used to feed the rocFFT library (lines 25–31) and, finally, the FFT result is saved as a Matlab file (line 34).

```
#include <OpenCLIPER/XData.hpp>
#include <OpenCLIPER/ProgramConfig.hpp>
#include <rocfft.h>
#include <string>

using namespace OpenCLIPER;
int main(int argc, char *argv[]) {
  std::shared_ptr<CLapp> pCLapp;

  try {
    // Create an OpenCLIPER app with HIP enabled
    PlatformTraits platformTraits;
    DeviceTraits deviceTraits;
    deviceTraits.useHIP = true;
    pCLapp = CLapp::create(platformTraits, deviceTraits);

    // Create input data using OpenCLIPER
    auto pData = new std::vector<complexType>({1,2,3,4,5});
    auto pNDArray = NDArray::createNDArray(pData);
    auto pIn = std::make_shared<XData>(pCLapp, pNDArray,
         TYPEID_COMPLEX);

    // Get HIP handle to created XData Object
    void* hipBuffer = pIn->getHIPDeviceBuffer();

    // Prepare and run the FFT using HIP
    rocfft_plan plan = NULL;
    size_t fftLength = pIn->getNDArray(0)->size();
    rocfft_plan_create(&plan, rocfft_placement_inplace,
         rocfft_transform_type_complex_forward,
         rocfft_precision_single, 1, &fftLength, 1, NULL);
    rocfft_execute(plan, &hipBuffer, NULL, NULL);
    hipDeviceSynchronize();
    rocfft_plan_destroy(plan);

    // Get back and save output data using OpenCLIPER
    pIn->matlabSave("output.mat");
  }
  catch (std::exception& e) {
    std::cerr << "Generic exception: " << e.what() << std::
         endl;
  }
}
```

Listing 2. Passing OpenCLIPER data objects to HIP libraries.

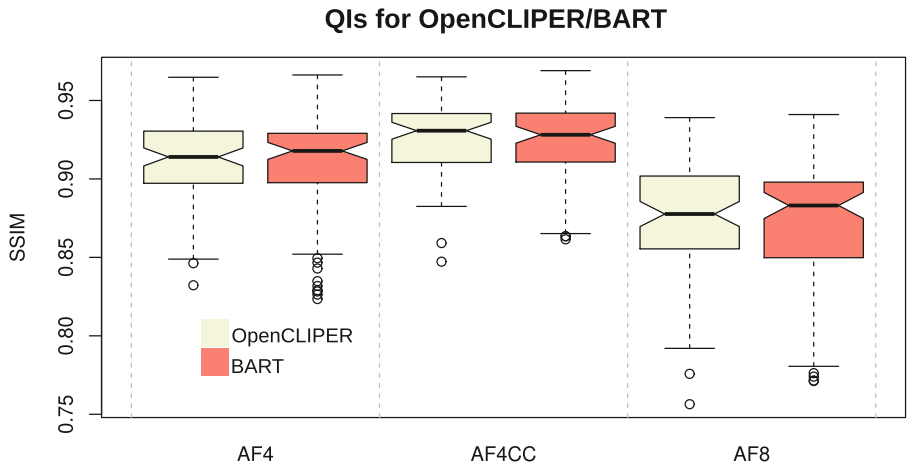
4. Evaluation

We have executed several reconstructions to test the behaviour of our platform. Reconstructions have been carried out on 7 healthy volunteers courtesy of King's College London. These data are 2D Cartesian, fully sampled dynamic short axis cine breath-hold ECG-triggered acquisitions in a 1.5 T Philips scanner with a bSSFP sequence. Some relevant parameters of the acquisitions include flip-angle 60°, TR/TE = 3/1.5 ms, spatial resolution 2 × 2 mm^2, slice thickness 8 mm, 20 cardiac phases, FOV 320 × 320 mm^2, 12–14 slices and the number of channels is between 19 and 32, depending on the subject. Both sensitivity maps and k-space data from all coils were provided to us. These datasets were retrospectively subsampled with a Gaussian variable-density random undersampling pattern along the phase encoding direction described in Asif et al. [31] for different values of acceleration factor (AF). Hence, for the experiments carried out we have fully sampled images which have been used as a reference for measuring some quality indices (QIs).

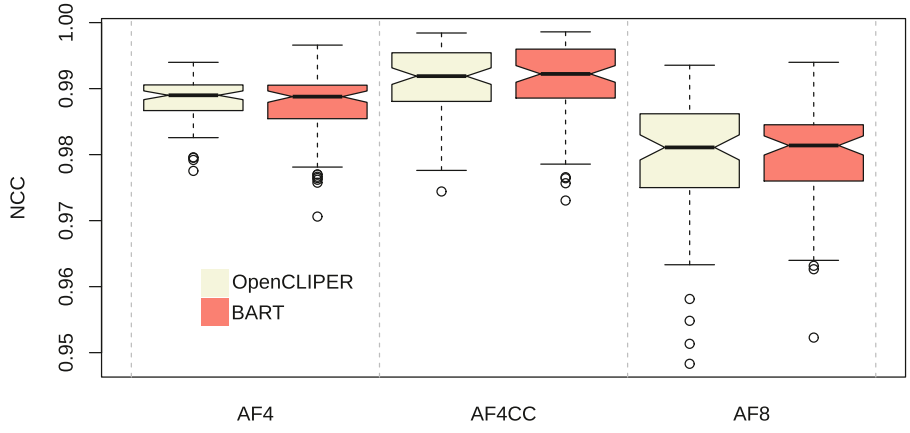
First, we have compared our platform with BART [13] to solve Eq. (1). As for OpenCLIPER we have used NESTA. As for BART,[3] since NESTA is not available in that platform, we have used the ADMM method [32]. As for these comparisons, $\mathcal{T}_\Theta$ in Eq. (1) is the identity due to the fact that BART does not incorporate —to the best of our knowledge— a groupwise ME/MC implementation.

Three experiments have been conducted, namely (a) AF = 4 (AF4), (b) AF = 4 plus coil compression (AF4CC) from 19–32 to half of the channels (9–16), and (c) AF = 8 (AF8). The parameter λ in Eq. (1) has been chosen to maximize the structural similarity index (SSIM) [33] in the BART reconstruction. Table 1 shows SSIM values for the three experiments. Thus, $\lambda = 0.01$ has been used for BART and OpenCLIPER reconstructions. For the sake of fairness, since the two optimization algorithms are different, we have chosen the internal NESTA parameters (namely, both μ and the stopping criterion, see lines 5 and 12 in Appendix C) to guarantee that SSIM values are comparable for ADMM and NESTA reconstructions; this is graphically shown in Fig. 2a, with boxplots of SSIM along all the slices of all the patients. Boxplots have been grouped in pairs, i.e., AF4 for OpenCLIPER and BART, and the same structure goes for

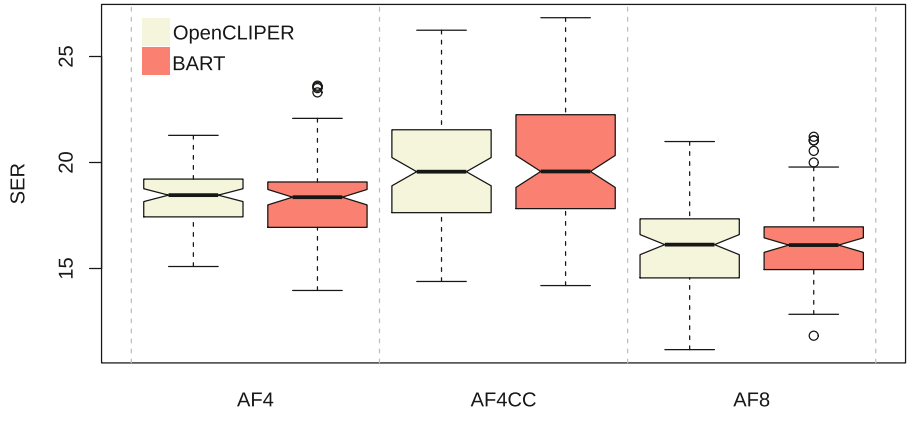
[3] Release downloaded on February 25, 2020, from https://github.com/mrirecon/bart/releases/tag/v0.5.00



(a)



(b)



(c)

Fig. 2. Boxplots of SSIM (a), NCC (b), and SER (c) for experiments AF4, AF4CC, and AF8 on both OpenCLIPER which makes use of NESTA, and BART (which uses ADMM). No significant differences have been found.

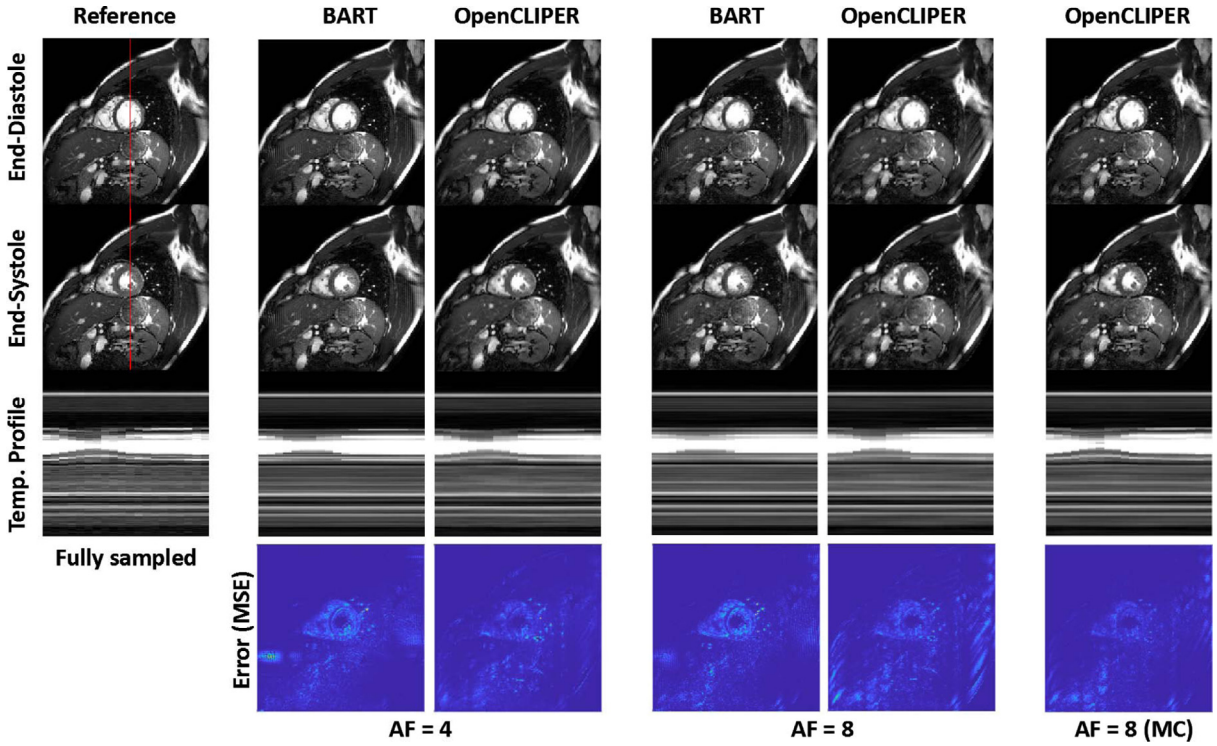


Fig. 3. Example of reconstructions using BART and OpenCLIPER for the experiments AF4, AF8, and AF8 with ME/MC; the latest only for OpenCLIPER. Fully sampled images are shown as reference. End-systole and end-diastole frames are shown in each case, as well as the intensities along time of the vertical profile marked with red line in reference images. Last row shows the error images (mean squared error, MSE, between reference and reconstructed image). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Mean SSIM values evaluated on undersampled BART reconstructions for a single slice of all patients and the three experiments (AF4, AF4CC, and AF8). Maximal values (bold highlighted) indicate the optimal regularization weight (λ in Eq. (1)) for the experiments.

Reg. Term	AF4	AF4CC	AF8
$\lambda = 10^{-5}$	0.8776	0.8923	0.8200
$\lambda = 10^{-4}$	0.8832	0.8979	0.8293
$\lambda = 10^{-3}$	0.9037	0.9201	0.8642
$\boldsymbol{\lambda = 0.01}$	**0.9056**	**0.9229**	**0.8719**
$\lambda = 0.1$	0.8955	0.9133	0.8592
$\lambda = 1$	0.8951	0.9128	0.8589

AF4CC as well as for AF8, as indicated in the horizontal labelling and in the legends on the figures. For completeness, different QIs have been analyzed. Fig. 2 also includes the corresponding boxplots for the normalized cross correlation (NCC, Fig. 2b) and the signal to error ratio (SER, Fig. 2c). Mann Whitney tests,[4] show no significant differences in any of these parameters between the compared frameworks. Additionally, Figs. 3 and 4 show two reconstruction examples for two different patients. In both cases, the fully sampled reconstructed images are shown as reference with the images reconstructed using BART and OpenCLIPER for the experiments AF4 and AF8; for the latter, ME/MC reconstruction is also shown for OpenCLIPERẆith these parameter setting, execution times in identical computer load situations have been compared. To deal with variability, each experiment has been run one hundred times for each patient.

Four platforms with four different GPUs have been employed. All of them are standard PC-class workstations based on Intel Core or AMD Ryzen processors. The exact GPU models are GeForce 2080Ti and Quadro RTX 6000 from nVidia, and Radeon RX 480 and Radeon RX 5700XT from AMD. For reference, some tests have also been carried out using a CPU as the computing device in a 70-thread Intel Xeon server. Notice that BART can only be run on CPU or nVidia GPUs, so no further comparison could be done between AMD and nVidia as for BART performance.

Fig. 5 shows boxplots of execution times on the GeForce 2080Ti (a) and RTX6000 (b) for the three experiments, with the same ordering as in Fig. 2. As for a comparison betweeen nVidia and Radeon, Fig. 6 shows boxplots of the AMD Radeon RX 5700XT (GFX1010) and the nVidia Quadro RTX 6000 in the same conditions as in the previous experiment. In terms of performance, the FFT implementation used by the reconstruction algorithm plays a prominent role. BART uses the nVidia's well-known proprietary implementation cuFFT, whereas OpenCLIPER uses the AMD's open-source implementation clFFT. While the former is highly optimized for nVidia GPUs, the latter is conceived to be run on every possible computing device and hence lacks of any specific optimization. This results in clFFT being slower than cuFFT, as shown in Table 2. OpenCLIPER compensates this with performance enhancements in other areas, such as parallel loading/saving of data and kernel caching (see Section 3.3).[5]

Fig. 7 shows boxplots of SSIM for AF8 on OpenCLIPER with and without ME/MC; whereas boxplots for NCC and SER are shown in Fig. 8. Results of the unilateral Mann-Whitney test for SSIM are significant ($p = 0.008$) for ME/MC. If a unilateral signed rank test is run per patient, differences favor ME/MC in six out of the seven patients tested. Mann-Whitney tests are also significant for NCC and SER ($p = 0.006$ and $p = 0.007$, respectively). The price

[4] Function `wilcox.test` in RStudio.

[5] In our previous work [34], FFT times reported were overestimated since they incorporated an additional synchronization of the device command queue.

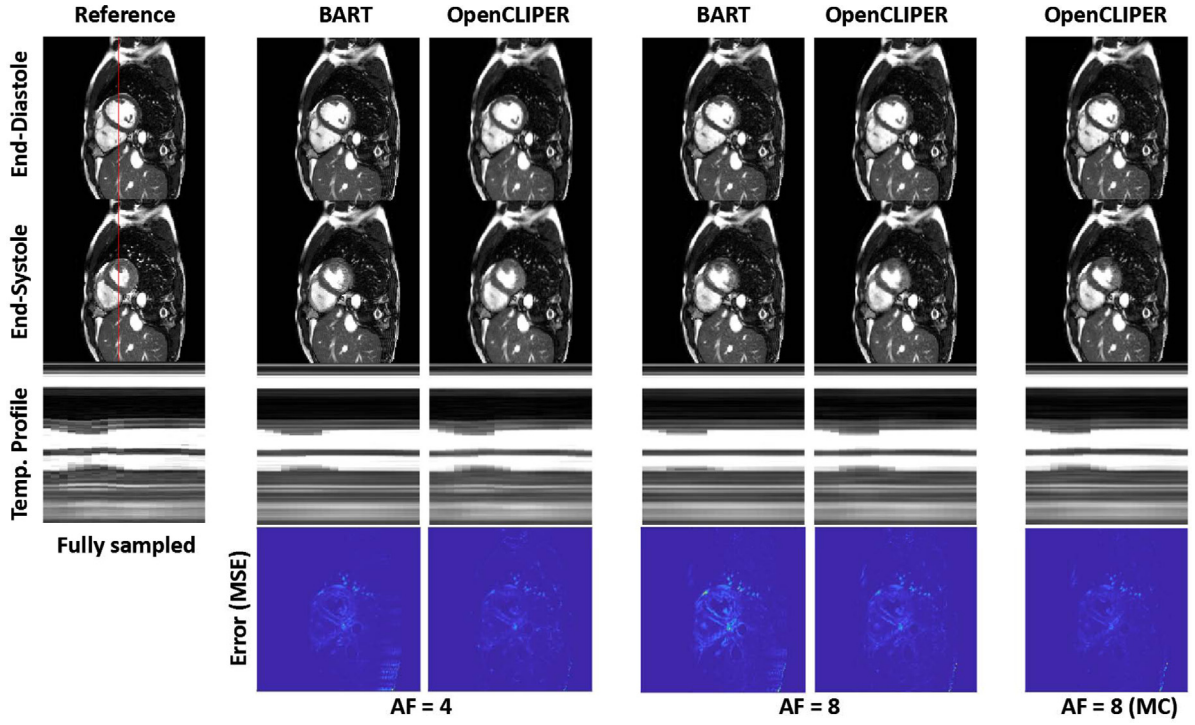


Fig. 4. Example of reconstructions using BART and OpenCLIPER for the experiments AF4, AF8, and AF8 with ME/MC; the latest only for OpenCLIPER. Fully sampled images are shown as reference. End-systole and end-diastole frames are shown in each case, as well as the intensities along time of the vertical profile marked with red line in reference images. Last row shows the error images (mean squared error, MSE, between reference and reconstructed image). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

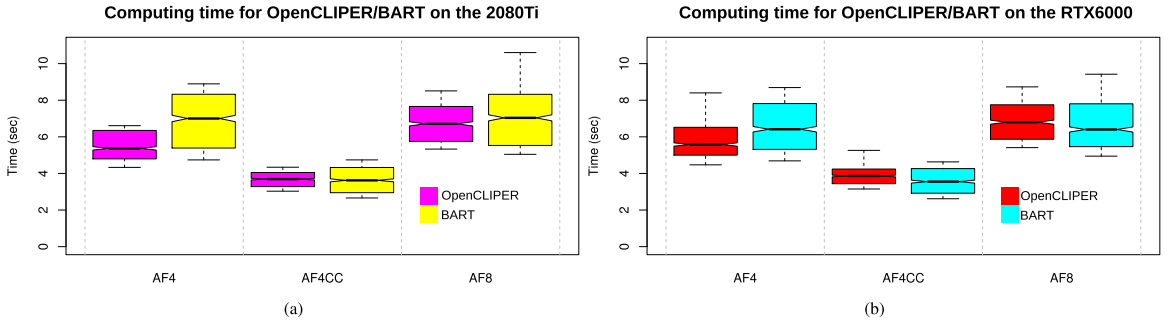


Fig. 5. Boxplots for experiments AF4, AF4CC, and AF8 on (a) GeForce 2080Ti and (b) RTX6000. Times reported are per whole slice stack.

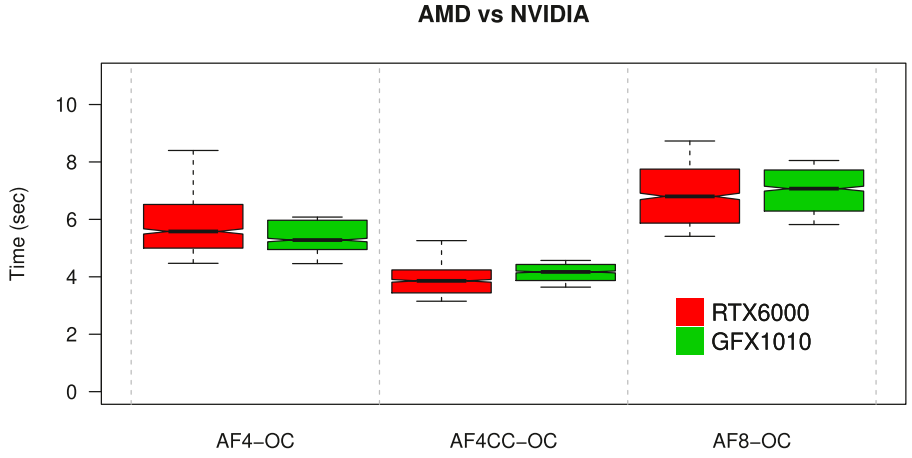


Fig. 6. Boxplots of computing time for AMD Radeon RX 5700XT (GFX1010) and nVidia Quadro RTX 6000 (RTX6000) for the three experiments. Notice that red-shaded boxes coincide with the red-shaded boxes in Fig. 5. Times reported are per whole slice stack. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Performance comparison between cuFFT and clFFT libraries on various devices. A single experiment consists in transforming a K-space dataset (160×160 images, 19 coils, 20 frames, complex floats) 1000 times. Each experiment is run 100 times on each library and device. For each combination we show mean execution times (mean) and their standard deviation (std), both in seconds.

	cuFFT		clFFT	
Device	mean	std	mean	std
GeForce 2080Ti	0.6299	0.001	0.6518	0.0227
Quadro RTX 6000	0.5880	0.0016	0.6413	0.0020
Radeon RX 5700XT	N/A	N/A	0.8276	0.0015

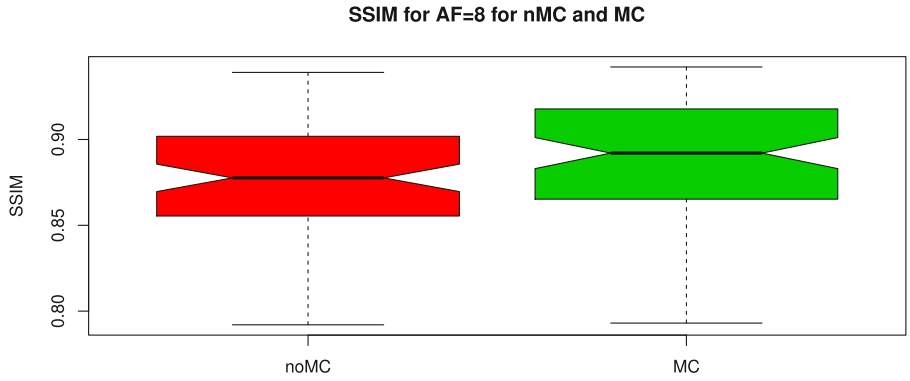


Fig. 7. Boxplots of SSIM for AF8 on OpenCLIPER with (right, green) and without (left, red) ME/MC. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

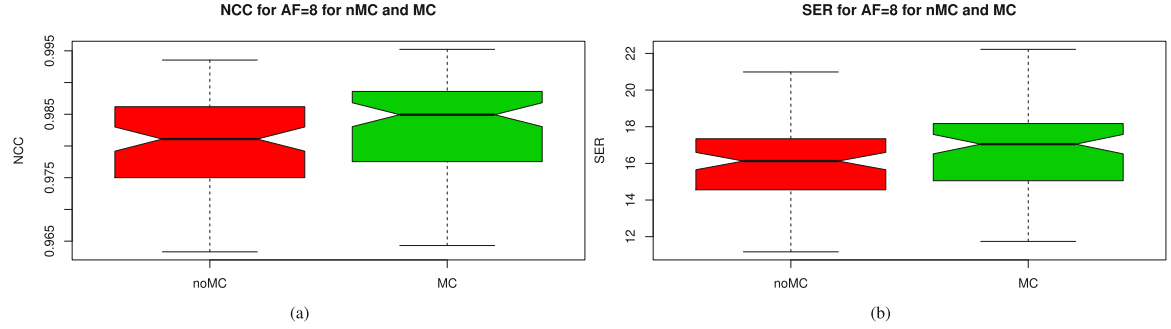


Fig. 8. Boxplots of NCC (a) and SER (b) for AF8 on OpenCLIPER with (right, green) and without (left, red) ME/MC. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

to pay is the increase in computation time, which reaches a median equal to 62.12 s and $(q_1, q_3) = (53, 72.13)$ for 2080Ti, with q_i the ith quartile. For RTX6000 the median is 60.16 and $(q_1, q_3) = (50.74, 69.24)$ (times reported are per whole slice stack).

Finally, we have also run an experiment on CPU with both BART and OpenCLIPER; the experiment is AF4CC with a randomly selected patient, ten repetitions. As for the former, the median execution time is 24.58 s ($(q_1, q_3) = (24.84, 24, 45)$) while for OpenCLIPER median is 16.33 and (q_1, q_3)= (16.09,16.59).

5. Discussion

OpenCLIPER reveals itself as a device agnostic platform for reconstruction of MR dynamic images. We have shown results for 2D although the code is prepared for higher dimensionality provided that sufficient memory is available. Fig. 5 shows that our computing times are comparable to those needed by BART; this has been tested in two nVidia devices with different memory capacities. Results are not point-by-point comparable since optimization methods used by both approaches are different. However, we meant to be fair by assuring that both platforms gave rise to images with similar qualities; optimization parameters were selected to this end, a goal that seems accomplished according to the evidence shown in Fig. 2. As for computing times themselves, we observe that our times are fairly similar to those from BART; hence, no obvious losses are appraised by using OpenCLIPER despite some administrative tasks need to be handled, as pointed out in Section 3.3.1, due to its device agnostic character; this overload is represented in Fig. 1 by the shaded blocks located outside the dashed box that contains the core of the reconstruction algorithm.

BART cannot be tested in AMD devices; hence only OpenCLIPER enters the comparisons in Fig. 6. The figure shows that a device one order of magnitude more economical can do a remarkable job. Therefore, OpenCLIPER makes it possible that an affordable device is used for image reconstruction in viable clinical times. When ME/MC enters the algorithm the computing time needed reaches a quantity of about one minute for a multislice reconstruction, a delay that seems also realistic in a clinical setting. Whether ME/MC is worth taking depends on the acceleration factor; for the AF = 8 case we have employed in our experiments, statistical differences were found.

The execution on CPU revealed that BART needed extra time for image reconstruction with respect to OpenCLIPER. Since no obvious differences were found on the nVidia devices, chances are that this is due to the fact that BART, when executed on the CPU, par-

allelizes only the FFT part of the reconstruction (making use of the threading support in the FFTW library), whereas in OpenCLIPER, every process' kernels are executed in parallel. Additionally, clFFT (the OpenCL version of FFT used by OpenCLIPER), has been used in both CPU and GPU experiments since kernel code is unique.

The FFT algorithm is exhaustively used in each reconstruction step. Hence, optimized implementations of this algorithm should have an appreciable impact in the overall reconstruction computing time. Our HIP interface could be an alternative to use other FFT libraries written in CUDA (cuFFT, which is known to be highly optimized for nVidia devices, might be one of such alternatives if its source code were available). Nevertheless, interfacing overload is non-negligible and this requires further investigation.

6. Conclusion and further work

In this paper, a 2D MRI cardiac reconstruction system has been presented. This system is a) clinically viable in terms of execution times, and b) suitable for any computing device which has an OpenCL implementation, including CPUs, GPUs, FPGAs and DSPs from main vendors. The use of our framework OpenCLIPER allowed us to partition the problem in independent black boxes (processes) which are then connected as needed and executed in parallel on any capable device, while the source code remains unique for all prospective computing devices. Device initialization and maintenance is reduced to a minimum as well, while providing relevant development and debugging aids. Administrative but time-consuming tasks such as data loading/saving and kernel compilation are parallelized or cached so as to minimally affect overall performance. HIP code (and prospectively CUDA code) can be integrated in developments whenever needed as OpenCLIPER makes data objects shareable between OpenCL and HIP. We also contribute a parallel groupwise registration subsystem for motion estimation/compensation and a parallel multicoil NESTA subsystem for $l1-l2$ problem solving.

Further work includes extending the 2D reconstruction to the free-breathing 3D problem, which poses additional issues due to typical data volumes that extend far beyond the capacity of a single computing device memory, along with much higher processing times.

Declaration of Competing Interest

The entire research has been funded by the Government of Spain; the authors declare that they have no conflict of interest.

Acknowledgements

This work is supported by MINECO under grant TEC2017-82408-R. In addition, the authors acknowledge the Asociación Española Contra el Cáncer (AECC) and its Scientific Foundation for its grant PRDVL19001MOYA. We also acknowledge European Social Fund, Operational Programme of Castilla y León, and the Junta de Castilla y León, through the Ministerio de Educación. We express our gratitude to Dr. Claudia Prieto, King's College London, for kindly sharing with us the data in the experiments.

Appendix A. Notation and main operators

We aim to reconstruct a sequence of 2D images of size $N_1 \times N_2$ and N_t temporal frames. Data has been acquired using a coil array of C elements from M samples of a discretized k-space grid of size $K = K_1K_2$. The different terms included in Eq. (1) can be represented by the following matrices:

1. Multi-coil k-space subsampled data b is a vector of size $M \times 1$.
2. Reconstructed image m is a vector of size $N \times 1$, where $N = N_1N_2N_t$.
3. $E = AFS$ is the encoding operator, which can be represented as a matrix of size $M \times N$.
   (a) S is a matrix of size $NC \times N$ given by:
   $$S = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_C \end{pmatrix}$$
   where S_c, $1 \le c \le C$, is a diagonal matrix of size $N \times N$ whose diagonal elements represent the sensitivity maps of the coil c at a spacial location.
   (b) $\mathcal{F}$ is a matrix of size $M \times NC$ given by:
   $$\mathcal{F} = \begin{pmatrix} \mathcal{F}_1 & 0 & \dots & 0 \\ 0 & \mathcal{F}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathcal{F}_C \end{pmatrix}$$
   where $\mathcal{F}_c$, $1 \le c \le C$, is matrix of size $KN_t \times N$ representing the coefficients of Fourier transform.
   (c) A is a matrix of size $M \times M$ given by:
   $$A = \begin{pmatrix} \mathcal{A}_1 & 0 & \dots & 0 \\ 0 & \mathcal{A}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathcal{A}_C \end{pmatrix}$$
   where A_c, $1 \le c \le C$, is a diagonal matrix of size $KN_t \times KN_t$ whose diagonal elements take the value 1 if the entry corresponds to a sensed k-space location and 0 otherwise. Note that $A_1 = A_2 = ... = A_C$, since the sampling mask is the same for all coils.
4. $U = \Phi\mathcal{T}_\Theta$ is the sparsifying operator, which can be represented as a matrix of size $N \times N$. In this case, the operator is composed of two matrices:
   (a) Φ is a matrix of size $N \times N$ given by:
   $$\Phi = \begin{pmatrix} I & 0 & 0 & \dots & -I \\ -I & I & 0 & \dots & 0 \\ 0 & -I & I & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -I & I \end{pmatrix}$$
   where I is a identity matrix of size $N_1N_2 \times N_1N_2$. This matrix computes the temporal finite differences, i.e., tTV operator. We have added a cyclical extension with respect to [26] based on the cardiac cycle periodicity.
   (b) $\mathcal{T}_\Theta$ is a matrix of size $N \times N$. A detailed description of this matrix can be found in Appendix B. Recall that for m_0, the transformation $\mathcal{T}_\Theta$ is the identity.

Note that with this formalism, the adjoint operator representation is straight forward as the Hermitian of these matrices (i.e., E^H and U^H).

Appendix B. Groupwise registration and motion compensation operators

The goal of the groupwise registration procedure is to jointly obtain a set of spatial transformations, one for each frame contained in the temporal sequence, so that transformed images ideally coincide. No image is taken as a reference to avoid any sort of bias. In practice, registered images will not be exactly coincident but they will constitute a sparse sequence in the temporal dimension, i.e., registration promotes sparsity. The target is the minimization of a cost function H (see Eq. (B.2)), which consists of a data

fidelity term —the sum of squares of the intensity differences with respect to the reference that is built on the fly (see Eq. (B.1))—and a smoothness term, which prevents the onset of unrealistic transformations. Minimization is achieved by gradient descent. Gradients are calculated following standard procedures [35].

$$V(x) = \frac{1}{N_t}\sum_{n=1}^{N_t}\left(m^{(n)}\left(\mathcal{T}^{(n)}(x)\right) - \frac{1}{N_t}\sum_{n'=1}^{N_t} m^{(n')}\left(\mathcal{T}^{(n')}(x)\right)\right)^2 \quad \text{(B.1)}$$

$$H(\tau) = \int_{\chi}\left[V(x) + \int_0^{T_c}\sum_{p=1}^{4}\lambda_p R_p\, dt\right] dx \quad \text{(B.2)}$$

Eq. (B.2) shows that the metric is calculated within region χ, which may be the whole image or only an estimated area where most of the motion takes place. In these equations T_c is the cardiac cycle and λ_p and $1 \le p \le 4$ are the weights of the regularization terms R_p. These terms account for derivatives of the motion vector field; specifically R_1 and R_2 are respectively first and second order spatial derivatives while R_3 and R_4 are first and second order temporal derivatives.

The motion vector field is approximated by means of B-Splines functions, the coefficients are which of the parameters that enter the optimization. This is the ME stage, where the dense (forward) transformation is calculated as follows:

$$\mathcal{T}(x) = x + \sum_{u_1=\mathcal{C}_{11}}^{\mathcal{C}_{12}}\sum_{u_2=\mathcal{C}_{21}}^{\mathcal{C}_{22}}\left(\prod_{l=1}^{2} B_3\left(\frac{x_l - p_{u_l}}{\Delta_l}\right)\right)\cdot\theta_u \quad \text{(B.3)}$$

where $\mathcal{C}$ is the grid of control points, B_3 represents the uniform B-spline function of grade 3, p_{u_l} the control point coordinate along dimension l, Δ_l is the pixel resolution of the grid of control points in each dimension and θ_u the free parameter that drives the deformation. The mesh boundaries given by $\mathcal{C}$ are set so that the region of interest is completely covered with a certain margin for approximation via B-splines to avoid large displacement values producing inconsistencies at the edges of the region of interest leading to errors in interpolation. The registered image set is obtained by means of interpolation, which may be represented in matrix form as [26]:

$$\mathcal{T}_\Theta = \begin{pmatrix} \mathcal{T}_{\Theta 1} & 0 & \dots & 0 \\ 0 & \mathcal{T}_{\Theta 2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathcal{T}_{\Theta t} \end{pmatrix}$$

where $\mathcal{T}_{\Theta t}$ is a matrix of size $N_1 \times N_2$ associated to the transformation of the frame t in the sequence; it contains the interpolation coefficients that result as a consequence of the transformation. Its generic shape for will be:

$$\mathcal{T}_{\Theta 1} = \begin{pmatrix} & & & & & \dots & & & & & \\ 0 & \dots & w_1 & w_2 & 0 & \dots & w_3 & w_4 & 0 & \dots & 0 \\ 0 & \dots & 0 & w_1' & w_2' & \dots & 0 & w_3' & w_4' & \dots & 0 \\ & & & & & \dots & & & & & \end{pmatrix}$$

i.e., the matrix will be sparse, the sparsity degree of which depends on the interpolation order. For a bilineal interpolation, for instance, only four coefficients will be non-null in each row, as indicated in the equation. This is the MC part of the algorithm. These matrices and their adjoints are not explicitly built, but they are applied as operators for efficiency. For more information about B-Spline free-form deformations see [36].

Appendix C. CS reconstruction using NESTA

Algorithm 1 MRI reconstruction.

Step 0: **NESTA multicoil reconstruction**
Inputs: multi-coil k-space subsampled data b, encoding operator $E = A\mathcal{F}S$, sparsifying operator $U = \Phi$, their adjoints and parameters $\lambda, \gamma, L_a \in \mathbb{R}^+$
Initialization: $\mu^{(0)}$, $x_0 = E^H b$, the number of steps *maxIter* and parameter L_μ
for $t = 0$ to *maxIter* **do**
Step 0.1: Apply Nesterov's algorithm with $\mu = \mu^{(t)}$
for $k \ge 0$ **do**
a. Compute the gradient of the l1-norm:

$$\nabla f_\mu(Ux_k) = \begin{cases} \frac{1}{\mu}U^H U x_k, & \text{if } |Ux_k| \le \mu, \\ U^H sgn(Ux_k), & \text{otherwise,} \end{cases}$$

where function $sgn()$ applied to vector v means the stack of $sign(v_i/|v_i|)$, with v_i the ith vector component.
b. Compute the gradient of the cost function:

$$\nabla f(x_k) = E^H(Ex_k - b) + \lambda\nabla f_\mu(Ux_k)$$

c. Compute y_k:

$$y_k = x_k - \frac{1}{\lambda L_\mu + L_a}\nabla f(x_k)$$

d. compute z_k:

$$z_k = x_0 - \frac{1}{\lambda L_\mu + L_a}\sum_{j\le k}\alpha_j \nabla f(x_k)$$

where $\alpha_k = 1/2(k+1)$.
d. Compute x_{k+1}:

$$x_{k+1} = \tau_k z_k + (1-\tau_k)y_k.$$

where $\tau_k = 2/(k+3)$.
Stop when a given criterion is met
end for
Step 0.2: Decrease the value of $\mu : \mu^{(t+1)} = \gamma\mu^{(t)}$
end for
Outputs: reconstructed image $m_0 = x_{k+1}$
for $i = 1$ to *MotionIters* **do**
Step 1: **ME-GW**
Inputs: reconstructed image m_{i-1}
Initialization: fix the region of interest, create the control points mesh, compute B-splines products and coefficients, as in [7]
for $j \ge 0$ **do**
Step 1.1: Calculate the pixel-wise displacement fields
Step 1.2: Transform images using linear interpolation
Step 1.3: Calculate the metric and smoothing terms
Step 1.4: Calculate gradients
Step 1.5: Update the movements of the control points
Stop $\frac{1}{KN}||\theta_{n-1} - \theta_n|| < \epsilon_{\mathcal{T}}$ and $\frac{1}{|\chi|}(H_{n-1} - H_n) < \epsilon_H$
end for

Outputs: $\mathcal{T}_\Theta$
Step 2: **NESTA reconstruction with MC**
Inputs: multi-coil k-space subsampled data b, encoding operator $E = K\mathcal{F}S$, sparsifying operator $U = \Phi\mathcal{T}_\Theta$, their adjoints, and parameters $\lambda, \gamma L_a \in \mathbb{R}^+$
Initialization: $\mu^{(0)}$, $x_0 = m_{i-1}$, the number of steps *maxIter* and parameter L_μ
See above (step 0) for further details
Outputs: reconstructed image $m_i = x_{k+1}$
end for

References

[1] R.G. Assomull, D.J. Pennell, S.K. Prasad, Cardiovascular magnetic resonance in the evaluation of heart failure, Heart 93 (8) (2007) 985–992.

[2] R.-M. Menchón-Lara, F. Simmross-Wattenberg, P. Casaseca-de-la Higuera, M. Martín-Fernández, C. Alberola-López, Reconstruction techniques for cardiac cine MRI, Insights Imaging 10 (1) (2019) 100, doi:10.1186/s13244-019-0754-2.

[3] S. Jodogne, Orthanc: Open-source, lightweight DICOM server, 2012(accessed December 7, 2020).

[4] The Khronos Group, Inc., OpenCL Overview, (https://www.khronos.org/opencl/).

[5] A.M.D. Inc., HIP: Convert CUDA to portable C++ code, (https://github.com/ROCm-Developer-Tools/HIP).

[6] F. Simmross-Wattenberg, M. Rodríguez-Cayetano, J. Royuela-del Val, E. Martín-González, E. Moya-Sáez, M. Martín-Fernández, C. Alberola-López, OpenCLIPER: an OpenCL-based C++ framework for overhead-reduced medical image processing and reconstruction on heterogeneous devices, IEEE J. Biomed. Health Inform. 23 (2019) 1702–1709, doi:10.1109/JBHI.2018.2869421. http://opencliper.lpi.tel.uva.es

[7] J. Royuela-del Val, L. Cordero-Grande, F. Simmross-Wattenberg, M. Martín-Fernández, C. Alberola-López, Nonrigid groupwise registration for motion estimation and compensation in compressed sensing reconstruction of breath-hold cardiac cine MRI, Magn. Reson. Med. 75 (4) (2016) 1525–1536.

[8] D. Rueckert, L.I. Sonoda, C. Hayes, D.L. Hill, M.O. Leach, D.J. Hawkes, Nonrigid registration using free-form deformations: application to breast mr images, IEEE Trans. Med. Imaging 18 (8) (1999) 712–721.

[9] S. Becker, J. Bobin, E. Candés, Nesta: a fast and accurate first-order method for sparse recovery, SIAM J. Imaging Sci. 4 (1) (2011) 1–39.

[10] P. Després, X. Jia, A review of GPU-based medical image reconstruction, Phys. Med. 42 (2017) 76–92, doi:10.1016/j.ejmp.2017.07.024.

[11] H. Wang, H. Peng, Y. Chang, D. Liang, A survey of GPU-based acceleration techniques in MRI reconstructions, Quant. Imaging Med. Surg. 8 (2) (2018) 196.

[12] K. Bredies, F. Knoll, M. Freiberger, H. Scharfetter, R. Stollberger, The agile library for biomedical image reconstruction using GPU acceleration, Comput. Sci. Eng 15 (2013) 34–44.

[13] M. Uecker, F. Ong, J.I. Tamir, D. Bahri, P. Virtue, J.Y. Cheng, T. Zhang, M. Lustig, Berkeley advanced reconstruction toolbox, in: Annual Meeting of the International Society for Magnetic Resonance in Medicine, no. 23, Toronto, Canada, 2015, p. 2486. Available at https://mrirecon.github.io/bart/

[14] M. Hansen, T. Sorensen, Gadgetron: an open source framework for medical image reconstruction, Magn. Reson. Med. 69 (6) (2013) 1768–1776. Available at http://gadgetron.github.io/

[15] J. Gai, N. Obeid, J.L. Holtrop, X.-L. Wu, F. Lam, M. Fu, J.P. Haldar, W.H. Wen-mei, Z.-P. Liang, B.P. Sutton, More impatient: a gridding-accelerated toeplitz-based strategy for non-cartesian high-resolution 3D MRI on GPUs, J. Parallel Distrib. Comput. 73 (5) (2013) 686–697.

[16] A.R. Brodtkorb, T.R. Hagen, M.L. Sætra, Graphics processing unit (GPU) programming strategies and trends in GPU computing, J. Parallel Distrib. Comput. 73 (1) (2013) 4–13.

[17] M. Driscoll, Domain-Specific Techniques for High-Performance Computational Image Reconstruction, UC Berkeley, 2018 Ph.D. thesis.

[18] J. Royuela-del Val, M. Usman, L. Cordero-Grande, M. Martin-Fernandez, F. Simmross-Wattenberg, C. Prieto, C. Alberola-López, Whole-heart single breath-hold cardiac cine: a robust motion-compensated compressed sensing reconstruction method, in: Reconstruction, Segmentation, and Analysis of Medical Images, Springer, 2016, pp. 58–69.

[19] J. Royuela-del Val, L. Cordero-Grande, F. Simmross-Wattenberg, M. Martín-Fernández, C. Alberola-López, Jacobian weighted temporal total variation for motion compensated compressed sensing reconstruction of dynamic MRI, Magn. Reson. Med. 77 (3) (2017) 1208–1215.

[20] S.B. Dinh, N.H. Le, H.M. Nguyen, Accelerating genset reconstruction for sparsely sampled DCE-MRI with GPU, in: 2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), IEEE, 2018, pp. 29–34.

[21] 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, Comput. Methods Prog. Biomed. 98 (3) (2010) 278–284.

[22] D. Ruijters, B.M. ter Haar Romeny, P. Suetens, GPU-accelerated elastic 3D image registration for intra-surgical applications, Comput. Methods Prog. Biomed. 103 (2) (2011) 104–112, doi:10.1016/j.cmpb.2010.08.014.

[23] X. Du, J. Dang, Y. Wang, S. Wang, T. Lei, A parallel nonrigid registration algorithm based on B-spline for medical images, Comput. Math. Methods Med. 2016 (Article ID 7419307) (2016) 14, doi:10.1155/2016/7419307.

[24] N.D. Ellingwood, Y. Yin, M. Smith, C.-L. Lin, Efficient methods for implementation of multi-level nonrigid mass-preserving image registration on GPUs and multi-threaded CPUs, Comput. Methods Prog. Biomed. 127 (2016) 290–300, doi:10.1016/j.cmpb.2015.12.018.

[25] K. Punithakumar, P. Boulanger, M. Noga, A GPU-accelerated deformable image registration algorithm with applications to right ventricular segmentation, IEEE Access 5 (2017) 20374–20382, doi:10.1109/ACCESS.2017.2755863.

[26] C. Bilen, Y. Wang, I. Selesnick, Compressed sensing for moving imagery in medical imaging, arXiv preprint arXiv:1203.5772(2012).

[27] Advanced Micro Devices, Inc., clFFT, (https://github.com/clMathLibraries/clFFT).

[28] C. Nugteren, CLBlast: a tuned OpenCL BLAS library, IWOCL'18: International Workshop on OpenCL, ACM, New York, NY, USA, 2018, doi:10.1145/3204919.3204924. Available at https://github.com/CNugteren/CLBlast

[29] OpenCV Team, Open source computer vision library (OpenCV), (https://opencv.org).

[30] Advanced Micro Devices, Inc., rocFFT, (https://github.com/ROCmSoftwarePlatform/rocFFT).

[31] M.S. Asif, L. Hamilton, M. Brummer, J. Romberg, Motion-adaptive spatio-temporal regularization for accelerated dynamic MRI, Magn. Reson. Med. 70 (3) (2013) 800–812, doi:10.1002/mrm.24524.

[32] S. Boyd, N. Parikh, E. Chu, Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers, Now Publishers Inc, 2011.

[33] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612.

[34] F. Simmross-Wattenberg, M. Rodríguez-Cayetano, J. Royuela-del Val, E. Martin–Gonzalez, E. Moya-Sáez, M. Martín-Fernández, C. Alberola-López, Opencliper: an opencl-based C++ framework for overhead-reduced medical image processing and reconstruction on heterogeneous devices, IEEE J. Biomed. Health Inform. 23 (4) (2019) 1702–1709.

[35] L. Cordero-Grande, S. Merino-Caviedes, S. Aja-Fernández, C. Alberola-López, Groupwise elastic registration by a new sparsity-promoting metric: application to the alignment of cardiac magnetic resonance perfusion images, IEEE Trans. Pattern Anal. Mach. Intell. 35 (11) (2013) 2638–2650, doi:10.1109/TPAMI.2013.74.

[36] R.-M. Menchón-Lara, J. Royuela-del Val, F. Simmross-Wattenberg, P. Casaseca-de-la Higuera, M. Martín-Fernández, C. Alberola-López, Fast 4D elastic group-wise image registration. convolutional interpolation revisited, Comput. Methods Prog. Biomed. (2021) 105812.