Efficient Online 4D Magnetic Resonance Imaging

Marco Barbone [*1], Andreas Wetscherek [†2], Thomas Yung[1], Uwe Oelfke[2], Wayne Luk [‡1], and Georgi Gaydadjiev [§3]

[1]Department of Computing, Imperial College London, London, UK
[2]Joint Department of Physics at The Institute of Cancer Research and The Royal Marsden NHS Foundation Trust,London, UK
[3]Bernoulli Institute, University of Groningen, Groningen, Netherlands

February 15, 2022

Abstract

Magnetic Resonance (MR)-guided online Adaptive RadioTherapy (MRgoART) utilises the excellent soft-tissue contrast of MR images taken just before the patient's treatment to quickly update and personalise radiotherapy treatment plans. Four-dimensional (4D) MR Imaging (MRI) can resolve variations in respiratory motion patterns. 4D MRI data can be used to adapt the radiation beams to maximally target the tumour while sparing as much healthy tissue as possible. 4D MRI reconstruction, however, is computationally challenging and current state-of-the-art implementations are unable to meet MRgoART time requirements. This study bridges the gap between high-performance computing and medical applications by developing and implementing a parallel, heterogeneous architecture for the XD-GRASP algorithm capable of meeting the MRgoART time requirements. Our architecture exploits long-vector instructions and utilises all available resources, while minimising and hiding the communication overhead when external GPUs are used. As a result, the reconstruction time was reduced from 994 seconds to just 90 seconds with a speed-up of more than 11x. In addition, we evaluated the impact of the emerging Processing-in-Memory (PIM) technology. Our simulation results show that 16 low power, in-order PIM cores with no SIMD unit are 2.7x faster than an Intel Core™ i7-9700 8-core CPU equipped with AVX512 SIMD units. Additionally, 40 PIM cores match the performance of two AMD EPYC 7551 CPUs, with 32 cores each and just 87 PIM cores

*m.barbone19@imperial.ac.uk
†andreas.wetscherek@icr.ac.uk
‡w.luk@imperial.ac.uk
§g.gaydadjiev@rug.nl

will match the performance of an NVIDIA Tesla V100 GPU equipped with 5,120 CUDA cores.

Heterogeneous computing; Parallel computing; Processing-in-Memory; Magnetic Resonance Imaging; Radiotherapy; XD-GRASP;

1 Introduction

In recent years, the wide availability of multi-core CPUs and the evolution of GPUs, from pure graphics processors to massively parallel many-core multiprocessors, have dramatically changed high-performance computing. To achieve high-performance, in addition to reducing computational complexity and taking into account memory latency, modern algorithms require parallelisation across all cores available in the system. Further, to push performance, vector instructions and, where possible, external accelerators like GPUs, TPUs and FPGAs are used. Present-era computational challenges cannot be solved by simply using faster hardware: algorithms which are not designed to harness the full potential of hardware architectures may not lead to a significantly lower execution time on faster hardware, due to their low system utilisation. Additionally, many modern and emerging applications require the processing of very large datasets. For example, many artificial intelligence and machine learning applications require scans over large quantities of data, to extract the key properties of the dataset. Most of the data used by these applications are not reused by the CPU, rendering the caching either inefficient or unnecessary while overwhelming the data storage and routing resources of modern computers by continuously fetching new data from the main memory [1]. Given the high cost of data movement and the performance penalty of non-sequential memory data accesses, innovations such as three-dimensional (3D)-stacked memory dies, that combine a logic layer with DRAM layers, and the ability to perform logic operations using memory cells themselves, provide the possibility to perform general-purpose computation directly within the memory. This new trend in computer architecture is known as processing-in-memory (PIM). While PIM can allow many applications to overcome data-movement bottlenecks, the programmer has to identify the suitable memory-bound portions of a program that could be offloaded [2].

Magnetic resonance imaging (MRI) scans have been in routine clinical use in diagnostic radiology for a long time, due to its excellent soft-tissue demarcation and the variety of available imaging contrasts in MRI. More recently, the use of MRI in the context of radiotherapy has increased, due to its better differentiation of soft tissue which can, for example, resolve the infiltration of healthy tissue by tumours. This information can be used to design more appropriate radiotherapy treatment plans [3]. The recent availability of hybrid MR-Linacs, combining MR imaging with radiotherapy treatment delivery in one device, led to new requirements in terms of the acquisition and reconstruction time of MR images [4]. Both need to be completed within minutes to be used for treatment planning, as the patient is already positioned on the MR-Linac for treatment. This study aims to bridge the gap between high-performance computing and medical applications by applying a heterogeneous computing approach to accelerate four-dimensional (4D) MR image reconstruction, in the context of MR-guided online adaptive radiotherapy (MRgoART). 4D MRI can resolve the extent of respiratory motion of tumours and risk structures, which can be incorporated to improve the treatment plan or be used as training data for real-time imaging solutions [5, 6]. The results show that, by fully exploiting the available computing resources, orders of magnitude speed-up can be achieved which facilitates clinical adoption of the technique speed-up can band improves MR-Linac treatmentse achieved which facilitates clinical adoption of the technique.

4D MR image reconstructions, however, are often based on iterative compressed-sensing based approaches, as regularisation is required to remove undersampling artefacts. This becomes, in particular, a challenge if large imaging fields of views are required to resolve the motion of tumours and risk organs and if a high spatial resolution of the images is necessary. A variety of 4D MR image reconstruction techniques have been in consideration for MRgoART [7]. These techniques can be divided into direct methods and iterative conjugate gradient-based algorithms. While direct methods are faster, they are particularly sensitive to streaking artefacts, hence producing lower-quality images. Iterative algorithms produce higher-quality images but are characterised by higher computational complexity that results in much larger execution times [7]. While GPU acceleration has already been applied in the context of MRI [8], additional performance can be harnessed by simultaneously using CPU and GPU compute capabilities. Hence, a heterogeneous computing approach, in which the SIMD computations can be executed on GPUs while the CPUs perform the remaining algorithmic parts, can further reduce the reconstruction time compared to homogeneous implementations based only on either a CPU or GPU. Although the heterogeneous computing approach is promising, it creates new computational challenges as it requires the transfer of data to and from the GPU over the PCIe bus. While CUDA offers different communication APIs to transfer data between CPU and GPU, the PCIe bandwidth is limited, hence data transfers may dominate the total execution time, leading to a low overall system utilisation and limited performance.

In this study, the MRI data are acquired in k-space, following a stack-of-stars acquisition pattern [9] with golden angle spacing [10]. Given this acquisition scheme, one major component of MRI reconstruction algorithms is the use of the Non-Uniform Fast Fourier Transform (NUFFT) [11]. Despite this algorithm being widely researched, with many optimised implementations existing, these implementations are not thread-safe. Further, they are often internally parallelised and optimised for 3D MRI data. As such, they are not efficient, or even usable, in the context of 4D MRI. Additionally, the NUFFT algorithm is memory bound because nearby points in k-space may not be stored contiguously in memory, even though gridding strategies, optimised data structures and memory access patterns could significantly speed-up the naïve implementations [12]. The performance achieved is not optimal and there is still room for improvement. Thus, the performance achieved by PIM is investigated, which allows overcoming data movement bottlenecks by directly processing the data within the memory.

There are three key challenges that a successful MRgoART platform would need to address. First, it must overcome the limitation of current state-of-the-art NUFFT implementations which are optimised for 3D and 2D data. Second, it must minimise multi-threading overheads, hide data transfers and avoid inter-process communication. Third, a load-balancing strategy based on the 4D MRI data characteristics suited for multi-processing parallelism is required.

The main contributions of this 4D MRI centric study are:

- A parallelisation scheme optimised for 4D MRI data, to reduce multi-threading overhead, hide data transfer latency and minimise inter-process communication;
- An efficient architecture capable of meeting online time requirements of MRgoART to facilitate clinical use;
- Evaluation of the impact of future PIM technologies.

Considering an image of size 256x256x48 pixels and 10 respiratory phases as a high-quality 4D image [5] and a reconstruction time of at most two minutes suitable for MRgoART use, the throughput that a reconstruction algorithm should achieve is 262,144 pixels/second. Additionally, the reconstructed image should be indistinguishable to the naked eye when compared to an artefact-free reference image. Hence, the image should score at least 97% structural similarity, for the difference to be invisible to the human eye [13,14]. The result of this study is a heterogeneous implementation of the XD-GRASP algorithm which reduces reconstruction time by more than 95%, achieving five times the minimum throughput necessary to meet online MRgoART time requirements, producing high-quality images with similarity score over 97% when compared to the reference implementation, allowing early-adoption of 4D MRI in MR-guided online adaptive radiotherapy (MRgoART).

The remainder of this paper is organised as follows. In Section 2, there are introductions to MRI, radiotherapy and the XD-GRASP algorithm. Section 3 illustrates the architecture developed to accelerate the 4D MRI. Section 4 explains the testing methodology and the performance achieved by the architecture in three configurations. Finally, Section 5 concludes the paper and lists future directions.

2 Background

This section will outline some fundamental concepts.

2.1 Radiotherapy and MRI

Radiotherapy is part of approximately 50% of all cancer treatments [15]. It aims to treat the targets, such as tumours and affected lymph nodes, by delivering the prescribed radiation dose, while avoiding damaging healthy tissue to reduce toxicity and side effects. To optimise this therapeutic ratio, radiotherapy is typically delivered over the course of several treatment days (fractions). In conventional radiotherapy, the patient needs to be set up in exactly the same way for each fraction, to accurately deliver the previously calculated treatment plan. For this, radiation therapists have to rely on external markings and laser alignment, because only low contrast imaging is available on a conventional linac (linear accelerator) to verify the exact internal anatomy.

In this context, MR-guided online adaptive radiotherapy (MRgoART) is a game changer. It uses the excellent soft-tissue contrast of magnetic resonance imaging (MRI) [3] to adapt the radiation on each fraction according to day-to-day variations in anatomy, such as the filling of the bladder, bowel, stomach and rectum, and anatomical changes over the course of treatment, such as tumour shrinkage and weight-loss. The resulting increased certainty, with respect to the internal anatomy, could be translated into smaller treatment margins to account for setup errors, leading to lower radiation doses to healthy tissues. MRgoART became possible with the integration of MRI scanners with linacs to deliver radiation therapy [16–19]. Current clinical MRgoART workflows involve re-planning of the treatment based on a freshly acquired image, while the patient is on the table inside the hybrid MR-Linac (online). Treatment is then delivered, while organ and tumour motion is monitored on two-dimensional (2D) cine-MRI images. However, 2D cine-MRI cannot be used to perform treatment adaption as it does not contain enough information about the patient anatomy.

4D MRI resolves the three-dimensional patient anatomy during different phases of the respiratory cycle, adding the respiratory phase as the fourth (temporal) dimension. MRI data are assigned into different respiratory bins based on a respiratory surrogate signal, representing the different phases of the respiratory cycle. Including respiratory motion information on both target structures and healthy radiation-sensitive organs would be valuable for the treatment planning process. It could be used to identify favourable anatomical configurations, which allow delivery of higher doses to the tumour while maintaining safe radiation levels for the healthy organs. Another application of 4D-MRI is the retrospective calculation of delivered doses, which can then inform the radiation prescription for the following treatment days. For the treatment of moving targets, such as abdominal or thoracic tumours, 4D MRI could be used to inform mid-position planning [20], based on image registration to calculate a sharp image of the patient anatomy corresponding to the average position during the respiratory cycle.

2.2 XD-GRASP

XD-GRASP uses the conjugate gradient algorithm to iteratively minimise the following cost function [21]:

$$d = \underset{d}{\operatorname{argmin}}\{\|F \cdot S \cdot d - m\|_2^2 + \lambda \|T \cdot d\|_1\} \tag{1}$$

where:

- $\mathbf{d}$ are image series to be reconstructed in the x-y-t space
- $\mathbf{T}$ it the temporal total-variation (TV) operator (sparsifying transform)
- $\boldsymbol{m} = \left[m_1 \ldots m_c\right]^T$ are the acquired multi-coil radial k-space data with c coils
- $\mathbf{F}$ is the NUFFT operator defined on the radial acquisition pattern
- $\boldsymbol{S} = \left[S_1 \ldots S_c\right]^T$ are coil sensitivity maps in x-y space
- $\boldsymbol{\lambda}$ is the regularisation weight that controls the trade-off between data consistency and the regularisation enforcing sparsity

Intuitively, this optimisation problem, solved using the conjugate-gradient method, aims to minimise the difference between the k-space representation of the image and the acquired MRI signal, while avoiding the formation of streaking artefacts through regularisation.

2.3 Related Work

Mickevicius and Paulson [7] tested different reconstruction algorithms and compared the overall image quality, reconstruction time, artefact prevalence and motion estimates. They tested four of the most promising algorithms: a direct non-uniform fast Fourier transform (NUFFT) [22], iterative self-consistent parallel imaging reconstruction (SPIRiT) [23], sensitivity encoding solved with a conjugate gradient algorithm (CG-SENSE) [24], and multi-dimensional golden-angle radial sparse parallel MRI (XD-GRASP) [25]. Their results show that, while NUFFT was the fastest, it exhibited the largest amount of streaking artefacts. On the other hand, XD-GRASP was not fast but was least sensitive to undersampling artefacts, hence leading to short scan durations. This, combined with a longer processing time of the XD-GRASP algorithm, resulted in both high-quality images and a reduction of the overall time required to acquire and process the image data. For this work, XD-GRASP [25], an open-source reconstruction algorithm for 4D MRI, was chosen; it provides a good compromise between its ability to reconstruct images from undersampled radial MR acquisitions and its computational complexity.

XD-GRASP was previously explored for use in MR-guided radiotherapy, where 6 respiratory phases were reconstructed for 40 slices with an in-plane resolution of 1.77mm and a slice thickness of 6mm [7]. Recently, its use for MR signature matching was proposed, where 10 respiratory phases were reconstructed for 48 slices with an in-plane resolution of 1.25mm and a slice thickness of 5mm in 74 minutes [5]. Using 10 respiratory phases is common in clinical 4D CT imaging. Incorporating motion estimation into image reconstruction can achieve artefact-free reconstruction from more highly undersampled data. Using the joint MoCo-HDTV algorithm, 20 respiratory phases were reconstructed from only 240 radial spokes for 60 slices with an in-plane resolution of 1.5mm and a slice thickness of 5mm, with an average reconstruction time of 6.5 hours [26].

3 Architecture

The use of 4D MRI in the context MRgoART requires minimising the combined acquisition and reconstruction time while still producing high-quality images. To this end, the XD-GRASP algorithm was accelerated. This iterative algorithm is one of the least sensitive to undersampling artefacts and hence supports relatively short acquisition times while leading to clinically acceptable images [7]. Besides, this reconstruction algorithm can reconstruct different slices (2D images which, when stacked, build a 3D image) independently, making it one of the best candidates for acceleration. Further, the proposed architecture can exploit the slice parallelism offered by the 4D data, minimising the overhead related to multi-threading and multi-process communications. In the case of GPU acceleration, the architecture overlaps data transfers to and from the GPU with useful computation avoiding idling times. In addition, the impact of future PIM technology in MRgoART workflows was tested. With NUFFT being a memory bound algorithm, the proposed architecture fully exploits the bandwidth offered by PIM architectures, fully loading every PIM core to outperform non-PIM architectures with the same core count.

3.1 NUFFT

The XD-GRASP algorithm relies on the NUFFT, a known and extensively studied algorithm with several optimised open-source implementations available. As such, the *FINUFFT* [27] and the *gpuNUFFT* [28] libraries are used, as they are already optimised and implement all the features required by the XD-GRASP algorithm. *FINUFFT* depends on the *FFTW* library [29] and, since they all are open-source, we recompiled them to take advantage of CPU vector instructions, including the recent AVX512.

These libraries are optimised for the 2D and 3D cases but they do not natively support 4D data. Given the characteristics of the XD-GRASP algorithm and the slice independence, it was possible to execute different calls to the NUFFT, one for each slice. This resulted in a severe performance loss due to the design assumptions made by the *FINUFFT* library. In case the slice parallelism is not exploited properly, the threads are created every time the NUFFT executed instead of being created once at the beginning of the program. This greatly increases the multi-threading overhead dominating the NUFFT execution time. On the other hand, the *gpuNUFFT* library, used in this context to offload the intensive SIMD computations of the XD-GRASP algorithm, spent most of the time transferring data between the CPU and GPU. Besides, the GPU was not fully utilised: the parallelism potential offered by respiratory phase and the number of coils is not enough to saturate a modern GPU. We decided not to leverage the internal parallelism but focus on an optimal parallelisation scheme for the 4D MRI to overcome the above limitations.

3.2 Processing in Memory

In non-Cartesian MRI reconstruction, the acquired non-uniformly spaced data are usually interpolated onto a uniform Cartesian grid before performing a fast Fourier transform (FFT) [22, 30]. However, this method has many different bottlenecks: the uniform grid needed to retain the k-space coordinates greatly increases the memory requirement; close points in k-space might not be stored in contiguous memory locations, causing cache misses; and neighbourhood search requires scanning all the non-uniform space coordinates for each grid location [12]. As a consequence, the NUFFT is a memory-intensive workload. Ghose et al. [2] present a methodology to identify applications, or portion of applications, that can take advantage of PIM. By applying this methodology, the NUFFT algorithm, which is heavily memory intensive, was identified as suitable for PIM.

The PIM programming model is still being researched and, as such, many different solutions have been proposed [2]. In this study, the PIM targets are offloaded to the PIM cores by means of compiler directives, so that the compiler takes care of generating the threads for PIM execution. For the sake of generality, concurrent execution of CPU and PIM cores was not enabled, removing the need for coordination between the CPU and PIM logic. This allows PIM to be used even in the absence of a cache coherence mechanism between the CPU and PIM. Non-concurrent execution limits parallelism causing performance losses, hence PIM performance results underestimate the full potential.

Lastly, similarly to previous studies [31], PIM kernels were integrated into the NUFFT library itself making the approach invisible to the user and more general, since it can be integrated into algorithms other than XD-GRASP.

3.3 Inter-slice and Intra-slice Parallelism

The output of the MR scanner is a complex signal measured in each of the receiver coils, a radio frequency receiver that produces the output signal of an MRI scanner. Data are acquired in k-space, following a stack-of-stars acquisition pattern [9] with golden angle spacing [10]. For each angle, nx samples are acquired across all slices. The samples taken at a particular angle are referred to as a spoke. These data are then pre-processed and sorted, following a self-gating signal [32], into respiratory phases. Lastly, Figure 1 shows how slices are processed independently and how the XD-GRASP algorithm is used to reconstruct a 3D volumetric image for each respiratory phase. Each input slice is used to compute only one output slice. The set of 3D images per respiratory phase produces a 4D reconstruction.

Given the input and the logic of the algorithm, there are different dimensions which can be considered for parallelisation. An example is the intra-slice parallelism across respiratory phases or coil channels. While relatively simple to implement and natively supported by the NUFFT libraries used, this parallelisation scheme has four major problems:

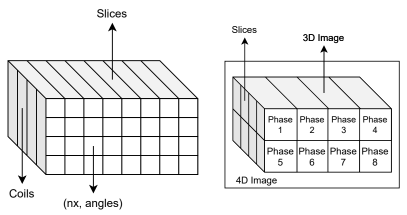

Figure 1: The input and output of the XD-GRASP algorithm.

- the computation of the gradient is a synchronisation point and it cannot be entirely parallelised;
- it does not allow early termination of the computation of slices that have already converged;
- in case of the utilisation of external accelerators, it prevents overlapping communications and computation;
- threads are created and destroyed multiple times for each iteration.

In our experiments, the intra-parallelism scheme limits the scalability of the XD-GRASP algorithm resulting in poor performance due to the overhead introduced by the thread creation/destruction and the synchronisation mechanism. Also, this form of parallelism does not allow overlapping data transfers and computations, so when external accelerators are used to perform the NUFFT, the transfer time dominates the compute time causing major performance degradation.

Another possible way to parallelise the algorithm is to exploit inter-slice parallelism. This parallelisation scheme consists of executing an instance of the XD-GRASP algorithm for each slice – thus splitting the 4D problem into separate and independent 3D problems along the z-axis. The XD-GRASP problem is solved for each slice separately since both the gradient and the objective function are local for each slice, so no communication, no synchronisation and no output recombination are needed. When the inter-slice parallelism is exploited, it is possible to overlap communications and computations since different slices can request their NUFFT computations concurrently. Hence, while processing a set of slices, another set of slices can be transferred to the accelerator. When adopting the inter-slice parallelism, the intra-slice parallelism is disabled, as it decreases the performance due to overhead caused by thread-creation, communication, and thread destruction.

Figure 2 is a timing diagram that shows the overlap strategy when three CPU processes, P1, P2 and P3, are forked. The GPU is called multiple times during each iteration of the conjugate gradient. The lines represent the computation time, while the rectangles represent the waiting time for the data to be processed, and the arrows show the data-transfers. When only one process is used, the communications between CPU and GPU are not overlapped meaning there are times when the CPU is idle waiting for the GPU to finish and vice versa. When more than one process is forked, communications can be overlapped and, in cases where a sufficient number of processes are forked, the full overlap is achieved. While the CPU performs other steps of the XD-GRASP algorithm, the GPU performs NUFFT, ideally achieving full utilisation without idling time.

3.4 Maximising resource utilisation

After we derived the optimal parallelisation scheme, a major challenge arose. State-of-the-art implementations of the NUFFT are not thread-safe, therefore unless they handle the parallelism internally, they cannot be used in a multi-threaded programming environment. To overcome this challenge and avoid manual editing of these libraries, a multi-processing approach was used instead of multi-threading. While this is thread-safe by definition and enables changing of the parallelisation scheme, it has major drawbacks:

- process creation and scheduling introduce more overhead than thread creation and scheduling;
- implementing dynamic load-balancing is complex;
- inter-process communications and synchronisations are costly due to the lack of shared memory;
- multi-process GPU scheduling is more expensive than multi-thread scheduling as the multi-process service has to guarantee memory isolation [33].

Given these constraints, the focus was to develop and implement an architecture that does not require any inter-process communication and leverages the copy-on-write features of the modern implementation of the *fork* system-call [34], as well as minimising multi-process scheduling on the GPU using the hardware process scheduler introduced with the NVIDIA *VOLTA* GPU architecture [33].

Due to the lack of communication between the processes, as they read the input and then process the necessary partition, linear scaling is expected in both the number of cores and the number of GPUs utilised, as the total number of slices to be processed increases. Since process creation overhead is negligible, it is better to create more processes than needed and achieve full overlap between CPU and GPU computation. Creating fewer processes does not give any benefit and with low number of processes, the performance degrades when communications are not fully overlapped.

In the case of PIM systems, the parallelization strategy used is the same. The only difference is that we create one process per PIM core and we allocate the slices that the PIM core computes in its memory vault. This way the PIM cores can compute independently without the need to: (a) access global memory and (b) communicate among themselves.

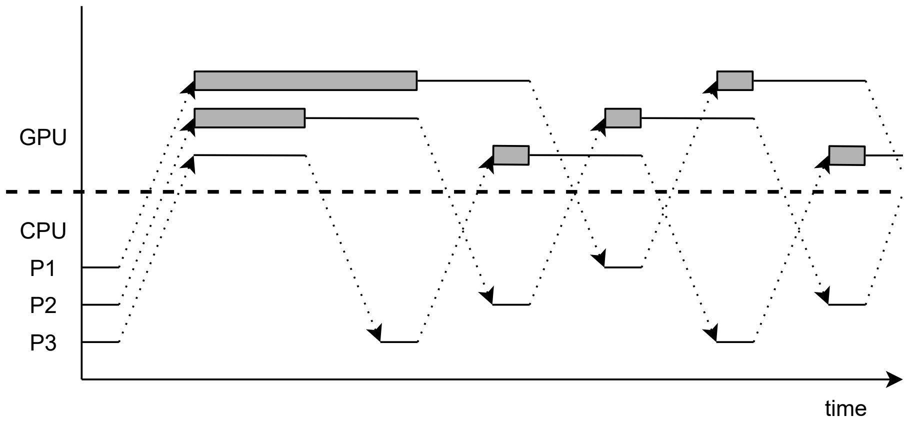

Figure 2: XD-GRASP architecture timing diagram. Dimensions are not representative of real-time duration.

3.5 Load-balancing

The adoption of multi-processing creates additional challenges since dynamic load-balancing techniques, like work-stealing [35] or a simple shared queue, can lead to huge overhead due to the lack of shared-memory amongst the worker-processes. Hence to maximise resource utilisation, a static load-balancing technique is required, based on the MRI data characteristics. Without regularisation, the stack-of-stars acquisition scheme leads to undersampling artefacts in the same locations across different slices, although the intensities of which can vary. As a result, the number of iterations performed by the XD-GRASP algorithm is about the same for each slice. To test this hypothesis, the time required for each slice was measured. Our experiments showed that the standard deviation is less than 1% of the total execution time. To minimise the overhead, we deployed a strategy that divides the slices into equal-sized partitions and then assign the first element of each partition to the first process, the second element to the second process, iterating over the processes until all work-units are assigned.

4 Results

The algorithm was evaluated on two systems, one physical and one simulated. The physical system incorporates two AMD EPYC™ 7551 32-core CPUs and a NVIDIA Tesla V100 PCIe 32GB GPU. Two different configurations were used: with and without GPU acceleration. The simulated system consists of an Intel Core™ i7-9700 CPU and PIM accelerators. There is no system with both GPU and PIM accelerators as PIM and GPUs would be performing the same portion of the algorithm since the only memory intensive workload of the XD-GRASP is the NUFFT.

The input data reading times are not included in the evaluation since in a real case scenario data will be directly loaded in RAM during the acquisition. The processing data transfers are completely hidden and overlapped with the computation. Only, the latency to send the first batch of data to the GPU and to read the last batch from the GPU would matter. It is worth mentioning that these two latencies are completely negligible as they account for much less than 0.1% of the total time. In the PIM case, there are no transfers as the data are already in the vault connected to the PIM core.

4.1 System Configuration

The AMD EPYC™ 7551 is a 64-bit 32-core 2 contexts x86 enterprise server microprocessor based on the 14*nm* Zen micro-architecture. It is equipped with 64MB of L3 cache with a Thermal Design Power (TDP) of 180W, running at 2.0GHz with an all cores boost clock up to 2.55 GHz and up to 3.0GHz for 12 active cores. The Tesla V100 PCIe 32GB is an enterprise GPU based on the 12*nm* Volta micro-architecture. It is equipped with 32GB GDDR5X with a TDP of 250W. It has 5,120 CUDA cores running at 1,455 MHz. The PIM system was simulated using *ZSim+Ramulator*, an open-source PIM simulation framework [36, 37]. This framework is based on two widely-known simulators: ZSim [38] and Ramulator [39]. A 3D-stacked DRAM, similar to the Hybrid Memory Cube [40], was modelled, where the memory contained 16 vaults. One low-power single-issue core, similar in design to the ARM Cortex-R8 [41], was added per vault ensuring that the area of the PIM core did not exceed the total available area for logic inside each vault (3.5-4.4mm^2) [42–44]. There is communication mechanism between PIM cores, the PIM cores can access the contents of their memory vault but not the contents of the other memory vaults. There is cache coherence mechanism among PIM cores and between PIM cores and CPU cores. The following system configuration was used:

- CPU: Intel Core™ i7-9700 CPU @ 3.00GHz – 8 out-of-order cores, L1 instruction/data caches 64KB, private, 8-way associative; private L2 caches 256KB, 8-way assoc.; shared L3 cache 12MB, 16-way assoc., 16-wide SIMD unit per core (AVX512 compatible);
- PIM core: 1 in-order core per vault, 1-wide issue, 1 floating-point unit, L1 instruction/data caches: 32KB private, 4-way associative;
- 3D-Stacked Memory: 4GB cube, 16 vaults/cube; Internal (256GB/s) and Off-Chip (32GB/s) bandwidths;
- Baseline Memory: DDR4 2400, 32GB, 4 channels.

4.2 Evaluation Methodology

To evaluate the architecture, the original MATLAB implementation supplied by Feng et al. [25] was extended to the 4D case by adding support to the z-axis, allowing the reconstruction of multiple slices. To perform a fair comparison, a MATLAB *parfor* (parallel for) over the *slices* was used, which replicates the same inter-slice parallelism strategy implemented in *C++* instead of serialising the different slices. The CPU benchmarks were executed on the above-described system. The MATLAB code was executed with MATLAB R2019a Update 6. The CPU code was compiled with GCC 9.3.1 and the GPU code was compiled with CUDA compilation tools 10.2. The NVCC flags were the default set by *gpuNUFFT* which only specify the architectural and code-generation flags to support different GPU micro-architectures. The GCC flags were **-O3 -march=native** which are the default flags set by *FFTW* [29]. Moreover, these flags were set so that the generated binary fully exploits the vector instruction offered by the CPU since most of the code can be vectorised. Since the implementation will be released as open-source, anyone can compile it with the best optimisation flags.

4.3 Dataset

The dataset was supplied by The Institute of Cancer Research, London (ICR) and consists of a female patient (48y) with pancreatic cancer who consented to participation in the PRIMER trial (NCT02973828) for the development of daily online MR imaging for MR image-guided radiotherapy. Images were acquired on ICR's Elekta Unity MR-Linac system (Elekta AB, Stockholm, Sweden), which operates at a field strength of 1.5 Tesla and has 8 receive coil channels (4 anterior and 4 posterior). A volumetric radial stack-of-stars gradient echo sequence with golden angle spacing [45] and balanced MRI contrast was acquired for 287 seconds, resulting in the acquisition of 831 spokes for each k-space partition. SPAIR fat-suppression was used in combination with a partial Fourier factor of 0.7. Further MRI parameters were: echo time 1.34 ms, repetition time 3.5 ms, field of view 500 x 500 x 200 mm^3, flip angle: 40°, bandwidth 866 Hz/px, acquired voxel size: 1.5x1.5x3.0mm^3. After a gradient delay correction [46] was performed, coil sensitivities were estimated using the adaptive combine algorithm [47] before applying the XD-GRASP algorithm. The tests were repeated 5 times and the results were averaged across the runs. The standard error is shown in Table 2. Input pre-processing was excluded as it is not part of the algorithm.

4.4 CPU Benchmarks

Figure 3 shows the scalability. The difference between the theoretical and normalised speed-up is that the theoretical speed-up does not account for the CPU boosting behaviour, whereas the normalised does. In these tests, the CPU can boost up to 3.0 GHz if 24 cores or less are utilised. All of the tests were run with the default CPU settings, not limiting the boosting behaviour, and with SMT enabled, which resulted in the CPU boosting up to 3.0 GHz until a maximum of 24 cores was loaded, then dropping down to 2.55 GHz. Then the CPU boosting frequency was limited to be the same in all cases, 2.55 GHz single-core and

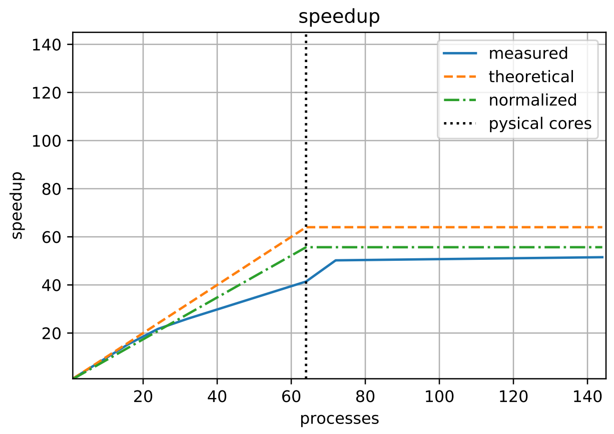

Figure 3: Measured, theoretical and normalised speed-up for the single-process implementation.

multi-core boost, and the single-process implementation was tested again. In this test, the implementation was 13% slower. The ideal speed-up was plotted according to this value, which corresponds to the normalised line in Figure 3. It can be observed that the boosting behaviour of the CPU measurably affects the results with the algorithm exhibiting almost perfect linear scaling when the difference in clock frequency is taken into account. In the case where the dataset contained a multiple of 64 slices, instead of a total of 144, the linear scaling could be observed up to 64 processes. The highest speed-up was measured with 72 processes as, for this number of processes, the workload is balanced with each process involving two work units. After 72 processes, the performance is constant within margins of error. Due to the number of slices and the number of cores not being multiples of one another and the coarse-grained parallelism adopted, the ideal speed-up is not reached on this dataset. When 72 processes are forked, the first 64 slices are processed in parallel and then the program must wait for the remaining 8 to be processed. Due to the SMT technology, the 8 remaining processes can already begin their computation, exploiting idle resources left from the other 64 processes.

4.5 GPU Benchmarks

Figure 4 shows the time measured for the GPU implementation with different numbers of processes. When the number of processes increases from 1 to 2, the partial communication overlap increases the performance by more than 60%. With only two processes, according to the NVIDIA *nvprof* tool, the GPU was not 100% utilised. By increasing the number of processes, a higher overall GPU utilisation, close to 100% for the entire duration of the program, was measured.

Since the *gpuNUFFT* library invokes different CUDA-kernels, some of them

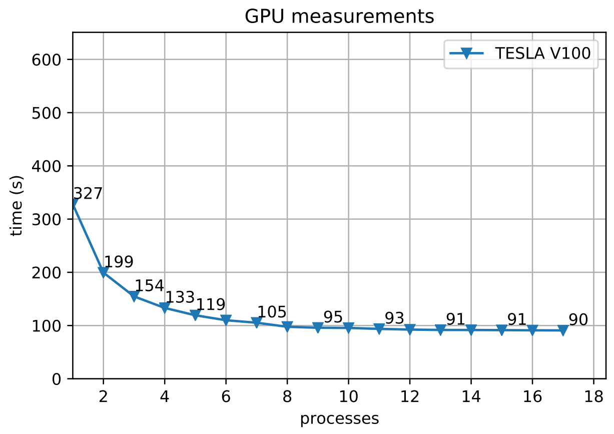

Figure 4: Measured GPU performance.

requiring more constant and shared memory and others requiring less, the increased number of processes likely leads to higher performance due to better resource utilisation – kernels requiring different resources can be executed simultaneously. On the TESLA V100, the time goes down to 90s and remains constant when 16 processes are forked. These results show that the multi-process overhead is negligible and it does not impact the performance even if tens of processes compete for the same GPU resources. This can be attributed to the multi-process scheduler implemented in hardware on GPUs based on the VOLTA micro-architecture [48].

4.6 PIM benchmarks

The simulation results show that 16 PIM cores are already faster than an 8-core Intel CPU, while 40 PIM cores match the performance of two 32-core AMD EPYC 7551 CPUs and 87 PIM cores match the performance of an NVIDIA Tesla V100 GPU equipped with 5,120 CUDA cores. The worst-case scenario was simulated, in which the CPU supports AVX instructions and the compiled CPU code fully leverages them. It is worth noticing that, while the total energy consumption has not been estimated, due to the thermal constraints of PIM and the lack of the most energy consuming CPU features such as out-of-order execution and SIMD units on the PIM cores, the proposed PIM architectures consume much less power than the non-PIM system used for testing. *Zsim+Ramulator* supports the simulation of PIM architectures in which the number of PIM cores (HMC vaults) is a power of 2. Given this limitation, the results of configurations that cannot be simulated were obtained by combining simulated results and the speed-up shown in Figure 3. The speed-up obtained by comparing the PIM simulation results was higher than the one shown in Figure 3, hence PIM results in Figure 5 can be considered a conservative estimate.

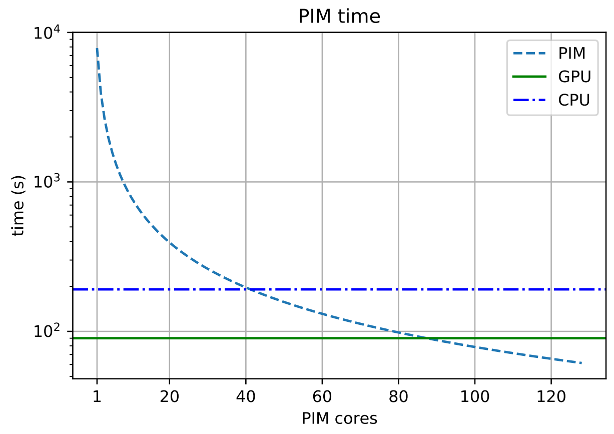

Figure 5: PIM time prediction (log scale).

4.7 Validation

In order to assess the validity of reconstructed images, they are compared to a set of reference images produced by the baseline MATLAB implementation of XD-GRASP supplied alongside [25]. The comparisons are evaluated quantitatively using a pair of measures: mean squared error (MSE) and structural similarity (SSIM). The MSE and the SSIM of two identical images are 0% and 100% respectively. In our experiments, the image similarity score was above 99% with a normalised MSE (nMSE) of 10^{-5} for the CPU case and 97% with an nMSE of $3 * 10^{-5}$ when the GPU was used. The lower accuracy of the GPU results arises from the oversampling factor and kernel width chosen for the GPU NUFFT calculation, parameters which could be tuned to increase image accuracy. Table 1 shows how different combinations of kernel widths (KW), sector widths (SW) and oversampling factors (OSF) affect gpuNUFFT performance and accuracy. The speedup is obtained by normalising the execution times against the chosen configuration (in bold). For our tests, we chose the fastest configuration capable of achieving at least 97% SSIM; achieving 99% SSIM requires 7% more time. As mentioned in Section 1, a minimum of 97% SSIM is required for the difference between two MRIs not to be visible to the human eye [13, 14] hence considered equivalent in this study. Figure 6 shows an example of the reconstruction performed by the different implementations. It is possible to notice that these images are indistinguishable to the naked eye.

5 Conclusions and future work

In this paper, we proposed a method to parallelise 4D MRI while minimising and hiding the specific data transfers. This method is generally applicable to any MRI processing algorithm as long as the stack-of-stars acquisition pattern [9]

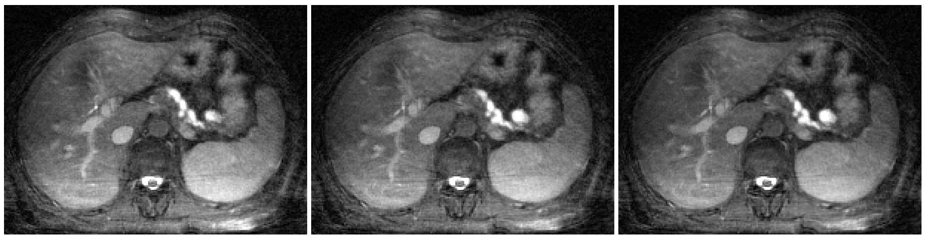
(a) Reference (b) CPU (c) GPU

Figure 6: Example of one respiratory phase of one slice.

Table 1: The NUFFT benchmarks ordered by speed-up.

KW	**OSF**	**SW**	**SSIM**	**Speed-up**
1	1,25	16	93%	1.14
1	2,00	16	91%	1.10
1	1,25	8	93%	1.09
1	2,00	8	91%	1.04
3	**1,25**	**16**	**97%**	**1.00**
3	2,00	16	97%	0.99
3	1,25	8	97%	0.94
3	2,00	8	99%	0.93

with golden angle spacing [8] is used. The results shown in Table 2 indicate, by harnessing the full potential of modern CPUs and GPUs, the proposed architecture can reduce the reconstruction time by more than 80% and by 91%, resulting in speed-up of 5.2x and 11x respectively. Table 3 shows the throughput in terms of pixel/second of the state-of-the-art implementations present in the literature. V100 and EPYC both use two AMD EPYC™ 7551, Mickevicius et al. and Feng et al. do not report the CPU used in their study but only the number of cores and the frequency. Rank et al. used an Intel Xeon E5-2687. From Section 1, we consider a throughput of 262,144 pixels/second as the minimum requirement for MRgoART. Higher throughput allows higher resolution images to be reconstructed within the time requirements of MRgoART, which provides more detailed image features potentially leading to improved target delineation and treatment. Our implementation, achieving a throughput five

Table 2: Results of the different implementations

Implementation	Time (s)	Standard Error (s)	speed-up
Reference	994	5.46	1.00
EPYC	191	1.76	5.20
V100	90	0.45	11.04

times higher than the minimum requirements, enables the reconstruction of images with 5 times higher resolution measured in the total number of pixels. This reduces the combined acquisition and reconstruction time from 21 minutes to 6 minutes, while maintaining very high similarity scores. This time reduction shows the benefits of heterogeneous computing applied to the medical context by increasing the clinical utility of 4D MRI in an MRgoART setting, making it possible to integrate 4D MRI into the treatment workflow on an MR-Linac. Because of the fast reconstruction time, volumetric real-time imaging solutions, such as MRSIMGA [5] or image-driven techniques [6] become possible. The scaling characteristics show that this architecture achieves high efficiency in all the tested systems, suggesting that it can exploit the full potential of higher core-count CPU and more powerful GPUs. Since the reported results are computed using different CPUs, Table 4 shows the throughput normalised per core-count. The GPU results are reported only for completeness in this table since this comparison is not fair towards the GPU implementation, because CUDA cores are much less powerful than CPU cores. However, we notice that in this case our implementation is 5.2 times faster than the reference executed on the same machine and 10.6 times faster than the fastest results reported in the literature.

Table 3: Throughput of the state-of-the-art implementations

Implementation	CPU/GPU Cores	Speed (GHz)	Algorithm	$\frac{\textbf{Pixels}}{\textbf{Second}}$
V100	5,120	1.455	GRASP	**1,310,720**
EPYC	64	2.0	GRASP	**617,616**
Reference	64	2.0	GRASP	**118,676**
Mickevicius et al. [7]	40	2.3	GRASP	**14,269**
Feng et al. [5]	8	NR	GRASP	**7,281**
Rank et al. [26]	16	3.10	MoCo HDTV	**2,564**

The PIM results show that the proposed architecture can greatly benefit from the adoption of PIM. To the best of our knowledge, most PIM architectures in the literature are based on HMC memory with 16 vaults and one PIM core per vault. While architectures equipped with more vaults are not frequently proposed, high core count PIM architectures might be available in the future. The 16 PIM cores architecture significantly outperform an Intel Core™ i7-9700 8-core CPU equipped with AVX512 SIMD units by a factor of 2.7x. In the case where systems equipped with more than 16 PIM cores becoming available in the future, the proposed architecture could even be faster than a 64-core system or an enterprise GPU, while consuming significantly less power. Moreover, the benchmarks show that the PIM cores are compute-bound and cannot fully exploit the large bandwidth offered by the HMC vaults. Introducing SIMD units or more powerful PIM cores will increase performance by an

Table 4: Throughput of the state-of-the-art implementations normalised per core-count.

Implementation	CPU/GPU Speed (GHz)	Algorithm	$\frac{\textbf{Pixels}}{\textbf{Second*Cores}}$
EPYC	2.0	GRASP	**9,650**
Reference	2.0	GRASP	**1,854**
Feng et al. [5]	NR	GRASP	**910**
Mickevicius et al. [7]	2.3	GRASP	**356**
V100	1.455	GRASP	**256**
Rank et al. [26]	3.10	MoCo HDTV	**160**

order of magnitude. It is worth noticing that the NUFFT implementation was optimised to best-exploit CPU compute resources such as caching and SIMD vector units. Since the PIM cores used in this study have small caches and are not equipped with SIMD units, the performance achieved underestimates the real PIM potential.

In the future, we plan to implement the complete pre-processing pipeline, interfacing our XD-GRASP 4D MRI reconstruction directly to the MR-Linac system. After completing the integration, the library will be released open-source and likely with a MATLAB interface to simplify its clinical application. We will explore applying our architecture to real-time volumetric imaging, where low reconstruction time is mandatory. We also plan to test the proposed architecture in a distributed scenario with multiple GPUs, to evaluate the scalability and the efficiency of such systems. Finally, Zhang et al. [49] showed that the performance of throughput-oriented programmable processing in memory can slightly outperform the performance of mainstream GPU. If this solution becomes available, we would like to benchmark our solution against it.

It is worth noting that the authors of FINUFFT [27] are working on a GPU implementation of the API which, at the time of writing, is still incomplete. According to their benchmarks [50], their implementation is much faster than gpuNUFFT [51] that we used in this paper. When cuFINUFFT is completed, we will integrate it in our implementation in order to produce the same output on both CPU and GPU, thus making the evaluation even fairer and, additionally, improving the performance.

Acknowledgements

This research project was supported by the CRUK Convergence Science Centre at The Institute of Cancer Research, London, and Imperial College London (A26234). We acknowledge funding from the Cancer Research UK programme grant C33589/A19727. The Institute of Cancer Research and The Royal Marsden NHS Foundation Trust are members of the Elekta MR-Linac Research Consortium.

References

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings - International Symposium on Computer Architecture*, vol. 13-17-June-2015. Institute of Electrical and Electronics Engineers Inc., 6 2015, pp. 105–117.

[2] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-memory: A workload-driven perspective," *IBM Journal of Research and Development*, vol. 63, no. 6, 11 2019.

[3] M. A. Schmidt and G. S. Payne, "Radiotherapy planning using MRI," *Physics in Medicine and Biology*, vol. 60, no. 22, pp. R323–R361, 2015.

[4] J. J. Lagendijk, B. W. Raaymakers, A. J. Raaijmakers, J. Overweg, K. J. Brown, E. M. Kerkhof, R. W. van der Put, B. Hårdemark, M. van Vulpen, and U. A. van der Heide, "MRI/linac integration," *Radiotherapy and Oncology*, vol. 86, no. 1, pp. 25–29, 1 2008.

[5] L. Feng, N. Tyagi, and R. Otazo, "MRSIGMA: Magnetic Resonance SIGnature MAtching for real-time volumetric imaging," *Magnetic Resonance in Medicine*, 2 2020.

[6] B. Stemkens, R. H. N. Tijssen, B. D. de Senneville, J. J. W. Lagendijk, and C. A. T. van den Berg, "Image-driven, model-based 3D abdominal motion estimation for MR-guided radiotherapy," *Physics in Medicine and Biology*, vol. 61, no. 14, pp. 5335–5355, 2016.

[7] N. J. Mickevicius and E. S. Paulson, "Investigation of undersampling and reconstruction algorithm dependence on respiratory correlated 4D-MRI for online MR-guided radiation therapy," *Physics in Medicine and Biology*, vol. 62, no. 8, pp. 2910–2921, 2017.

[8] H. Wang, H. Peng, Y. Chang, and D. Liang, "A survey of GPU-based acceleration techniques in MRI reconstructions," *Quantitative Imaging in Medicine and Surgery*, vol. 8, no. 2, pp. 196–208, 3 2018.

[9] K. T. Block, H. Chandarana, S. Milla, M. Bruno, T. Mulholland, G. Fatterpekar, M. Hagiwara, R. Grimm, C. Geppert, B. Kiefer, and D. K. Sodickson, "Towards routine clinical use of radial stack-of-stars 3d gradient-echo sequences for reducing motion sensitivity," *J Korean Soc Magn Reson Med*, vol. 18, no. 2, pp. 87–106, Jun 2014.

[10] S. Winkelmann, T. Schaeffter, T. Koehler, H. Eggers, and O. Doessel, "An optimal radial profile order based on the golden ratio for time-resolved MRI," *IEEE Transactions on Medical Imaging*, vol. 26, no. 1, pp. 68–76, Jan 2007.

[11] A. Dutt and V. Rokhlin, "Fast Fourier Transforms for Nonequispaced Data," *SIAM Journal on Scientific Computing*, vol. 14, no. 6, pp. 1368–1393, 11 1993.

[12] D. S. Smith, S. Sengupta, S. A. Smith, and E. Brian Welch, "Trajectory optimized NUFFT: Faster non-Cartesian MRI reconstruction through prior knowledge and parallel architectures," *Magnetic Resonance in Medicine*, vol. 81, no. 3, pp. 2064–2071, 3 2019.

[13] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 4 2004.

[14] "Results of the first fastMRI image reconstruction challenge." [Online]. Available: https://ai.facebook.com/blog/results-of-the-first-fastmri-image-reconstruction-challenge/

[15] Cancer Researh UK, "What is radiotherapy? — Cancer treatment — Cancer Research UK." [Online]. Available: https://www.cancerresearchuk.org/about-cancer/cancer-in-general/treatment/radiotherapy/about

[16] B. W. Raaymakers *et al.*, "Integrating a 1.5 t mri scanner with a 6 mv accelerator: proof of concept," *Physics in Medicine and Biology*, vol. 54, no. 12, pp. N229–N237, 2009.

[17] S. Mutic and J. F. Dempsey, "The viewray system: Magnetic resonance–guided and controlled radiotherapy," *Seminars in Radiation Oncology*, vol. 24, no. 3, pp. 196–199, 2014.

[18] B. W. Raaymakers *et al.*, "First patients treated with a 1.5 T MRI-Linac: clinical proof of concept of a high-precision, high-field MRI guided radiotherapy treatment," *Physics in Medicine and Biology*, vol. 62, no. 23, p. L41, 2017.

[19] B. G. Fallone, "The rotating biplanar linac–magnetic resonance imaging system," *Seminars in Radiation Oncology*, vol. 24, no. 3, pp. 200–202, 2014.

[20] J. W. H. Wolthaus, J. J. Sonke, M. Van Herk, and E. M. F. Damen, "Reconstruction of a time-averaged midposition ct scan for radiotherapy planning of lung cancer patients using deformable registrationa," *Medical Physics*, vol. 35, no. 9, pp. 3998–4011, 2008.

[21] L. Feng, R. Grimm, K. T. Block, H. Chandarana, S. Kim, J. Xu, L. Axel, D. K. Sodickson, and R. Otazo, "Golden-angle radial sparse parallel MRI: combination of compressed sensing, parallel imaging, and golden-angle radial sampling for fast and flexible dynamic volumetric MRI," *Magnetic Resonance in Medicine*, vol. 72, no. 3, pp. 707–717, 9 2014.

[22] J. A. Fessler and B. P. Sutton, "Nonuniform fast fourier transforms using min-max interpolation," *IEEE Transactions on Signal Processing*, vol. 51, no. 2, pp. 560–574, Feb 2003.

[23] M. Lustig and J. M. Pauly, "SPIRiT: Iterative self-consistent parallel imaging reconstruction from arbitrary k-space," *Magnetic Resonance in Medicine*, vol. 64, no. 2, pp. 457–471, 8 2010.

[24] K. L. Wright, J. I. Hamilton, M. A. Griswold, V. Gulani, and N. Seiberlich, "Non-Cartesian parallel imaging reconstruction," *Journal of Magnetic Resonance Imaging*, vol. 40, no. 5, pp. 1022–1040, 11 2014.

[25] L. Feng, L. Axel, H. Chandarana, K. T. Block, D. K. Sodickson, and R. Otazo, "XD-GRASP: Golden-angle radial MRI with reconstruction of extra motion-state dimensions using compressed sensing," *Magnetic Resonance in Medicine*, vol. 75, no. 2, pp. 775–788, 2 2016.

[26] C. M. Rank, T. Heußer, M. T. A. Buzan, A. Wetscherek, M. T. Freitag, J. Dinkel, and M. Kachelrieß, "4d respiratory motion-compensated image reconstruction of free-breathing radial mr data with very high undersampling," *Magnetic Resonance in Medicine*, vol. 77, no. 3, pp. 1170–1183, 2017.

[27] A. H. Barnett, J. Magland, and L. af Klinteberg, "A Parallel Nonuniform Fast Fourier Transform Library Based on an "Exponential of Semicircle" Kernel," *SIAM Journal on Scientific Computing*, vol. 41, no. 5, p. C479–C504, 2019.

[28] F. Knoll, A. Schwarzl, C. Diwoky, and D. K. Sodickson, "gpuNUFFT-an open source GPU library for 3D regridding with direct Matlab interface," in *Proceedings of the 22nd annual meeting of ISMRM, Milan, Italy*, 2014, p. 4297.

[29] M. Frigo and S. G. Johnson, "FFTW: An adaptive software architecture for the FFT," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 3, 1998, pp. 1381–1384.

[30] J. I. Jackson, C. H. Meyer, D. G. Nishimura, and A. Macovski, "Selection of a Convolution Function for Fourier Inversion Using Gridding," *IEEE Transactions on Medical Imaging*, vol. 10, no. 3, pp. 473–478, 1991.

[31] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 316–331, 11 2018.

[32] J. Paul *et al.*, "High-resolution respiratory self-gated golden angle cardiac MRI: Comparison of self-gating methods in combination with k-t SPARSE SENSE," *Magnetic Resonance in Medicine*, vol. 73, no. 1, pp. 292–298, 2015.

[33] NVIDIA, "Volta Tuning Guide :: CUDA Toolkit Documentation." [Online]. Available: https://docs.nvidia.com/cuda/volta-tuning-guide/index.html#multi-process-service

[34] L. Nyman and M. Laakso, "Notes on the History of Fork and Join," *IEEE Annals of the History of Computing*, vol. 38, no. 3, pp. 84–87, 7 2016.

[35] R. D. Blumofe, C. E. Leiserson, R. D. Blumofe, and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM*, vol. 46, no. 5, pp. 720–748, 9 1999.

[36] "CMU-SAFARI/ramulator-pim: A fast and flexible simulation infrastructure for exploring general-purpose processing-in-memory (PIM) architectures." [Online]. Available: https://github.com/CMU-SAFARI/ramulator-pim

[37] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "NAPEL: Near-memory computing application performance prediction via ensemble learning," in *Proceedings - Design Automation Conference*. Institute of Electrical and Electronics Engineers Inc., 6 2019.

[38] D. Sanchez and C. Kozyrakis, "ZSim: fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 475–486, 6 2013.

[39] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 1 2016.

[40] Hybrid Memory Cube Consortium, "HMC Specification 2.0," 2014.

[41] "Cortex-R8 – Arm Developer." [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-r/cortex-r8

[42] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 751–764, 5 2017.

[43] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The Mondrian Data Engine," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 639–651, 9 2017.

[44] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Digest of Technical Papers - Symposium on VLSI Technology*, 2012, pp. 87–88.

[45] S. Winkelmann, T. Schaeffter, T. Koehler, H. Eggers, and O. Doessel, "An optimal radial profile order based on the golden ratio for time-resolved mri," *IEEE Transactions on Medical Imaging*, vol. 26, no. 1, pp. 68–76, 2007.

[46] J. D. Ianni and W. A. Grissom, "Trajectory Auto-Corrected image reconstruction," *Magnetic Resonance in Medicine*, vol. 76, no. 3, pp. 757–768, 9 2016.

[47] D. O. Walsh, A. F. Gmitro, and M. W. Marcellin, "Adaptive reconstruction of phased array mr imagery," *Magnetic Resonance in Medicine*, vol. 43, no. 5, p. 682, 2000.

[48] NVIDIA, "MULTI-PROCESS SERVICE vR450," Tech. Rep., 2020. [Online]. Available: https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[49] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented programmable processing in memory," in *HPDC 2014 - Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing.* Association for Computing Machinery, 2014, pp. 85–97.

[50] Y.-h. Shih, G. Wright, J. Andén, J. Blaschke, and A. H. Barnett, "cuFINUFFT: a load-balanced GPU library for general-purpose nonuniform FFTs," *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, vol. 1, pp. 688–697, 2 2021.

[51] "gpuNUFFT - An Open-Source GPU Library for 3D Gridding with Direct Matlab and Python Interface." [Online]. Available: https://github.com/andyschwarzl/gpuNUFFT