

A comprehensive comparison of GPU- and FPGA-based acceleration of reflection image reconstruction for 3D ultrasound computer tomography

Matthias Birk · Michael Zapf · Matthias Balzer ·
Nicole Ruiter · Jürgen Becker

Received: 29 February 2012 / Accepted: 4 August 2012 / Published online: 25 September 2012
© Springer-Verlag 2012

Abstract As today's standard screening methods frequently fail to diagnose breast cancer before metastases have developed, earlier breast cancer diagnosis is still a major challenge. Three-dimensional ultrasound computer tomography promises high-quality images of the breast, but is currently limited by a time-consuming image reconstruction. In this work, we investigate the acceleration of the image reconstruction by GPUs and FPGAs. We compare the obtained performance results with a recent multi-core CPU. We show that both architectures are able to accelerate processing, whereas the GPU reaches the highest performance. Furthermore, we draw conclusions in terms of applicability of the accelerated reconstructions in future clinical application and highlight general principles for speed-up on GPUs and FPGAs.

Keywords Medical imaging · Heterogeneous computing · FPGA · GPU · Performance comparison

1 Introduction

Breast cancer is the most common type of cancer among women in the western world [8]. Unfortunately, it is often initially diagnosed after metastases have already developed. The presence of metastases, however, decreases the patient's probability of survival significantly. A more sensitive imaging method may enable detection in an earlier state and thus enhance the survival probability of affected women.

Three-dimensional ultrasound computer tomography (3D USCT) promises reproducible and high-quality volume images of the female breast and is therefore a good candidate for improvement in early breast cancer diagnosis. During a measurement, the breast is suspended into a water-filled basin (see Fig. 1, top) and surrounded by thousands of ultrasound transducers, i.e. emitters and receivers. The emitters sequentially send an ultrasonic wave front, which interacts with the breast tissue and is recorded by receivers as a pressure variation over time. These signals, also called A-Scans, are sampled and stored for all available emitter–receiver combinations. Thus, the interaction between unfocused ultrasonic waves and the imaged object is recorded from many different angles.

Our 3D USCT prototype system [20] is shown in Fig. 1. It is equipped with a total number of 628 dedicated emitters and 1,413 dedicated receivers. The complete sensor aperture can be rotated and translated in order to create further emitting and receiving positions. For a complete measurement, this results in over 3.5 million A-Scans or approx. 20 GByte of raw data per single breast. For acquisition of the

This work has partly been funded by Deutsche Forschungsgemeinschaft.

M. Birk (✉) · M. Zapf · M. Balzer · N. Ruiter
Institute for Data Processing and Electronics,
Karlsruhe Institute of Technology (KIT),
Karlsruhe, Germany
e-mail: matthias.birk@kit.edu

M. Zapf
e-mail: michael.zapf@kit.edu

M. Balzer
e-mail: matthias.balzer@kit.edu

N. Ruiter
e-mail: nicole.ruiter@kit.edu

J. Becker
Institute for Information Processing Technology,
Karlsruhe Institute of Technology (KIT),
Karlsruhe, Germany
e-mail: becker@kit.edu

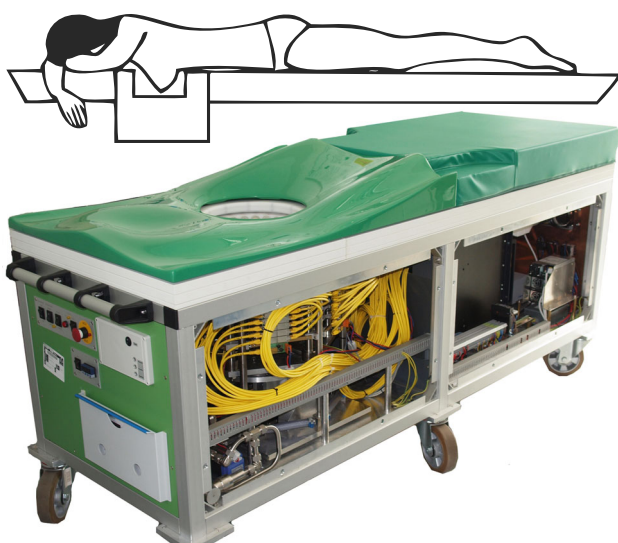


Fig. 1 Schematic drawing of an USCT measurement (*top*) and an image of our prototype system with covers being removed (*bottom*). During a measurement, the breast hangs freely into the water-filled basin. The surface of this basin is equipped with more than 2,000 ultrasound transducers

A-Scans, we use a custom data acquisition (DAQ) system [9]. After completion, the recorded A-Scans are transferred to a computer workstation for time-consuming image reconstruction.

The duration of the image reconstruction currently limits the clinical applicability of the overall imaging method. The core element is a synthetic aperture focusing technique (SAFT) [7], which exploits the pressure over time information in the A-Scans in order to create reflectivity images. A promising approach to accelerate this reconstruction step is parallel processing in application-specific hardware. In this work, we investigate the performance of our SAFT reconstruction kernel on a graphics processing unit (GPU) and on field programmable gate arrays (FPGAs) embedded in our DAQ system. Furthermore, we include a current high-performance FPGA into the comparison. Note that the first FPGA system is given by our current reconstruction scenario and the latter should give insight into future set-ups. We relate the attained performance values with our highly optimized CPU version and draw conclusions in terms of general applicability of the accelerated versions for use in our research as well as in future clinical routine. Furthermore, we aim to highlight general principles for speed-up on GPUs and FPGAs.

The remainder of this paper is structured as follows: Sect. 2 gives a short overview of related work in the fields of parallel medical image reconstruction as well as performance comparisons of GPUs and FPGAs in general. Section 3 introduces our SAFT-based image reconstruction kernel and derives a parallelized algorithm description. The hardware platforms targeted in the comparison are detailed

in Sect. 4 and the respective implementations of the SAFT kernel are described in Sect. 5. In Sect. 6, our performance results are presented and discussed. The paper is concluded in Sect. 7.

2 Related work

The related work for this paper can be grouped into two categories. Firstly, we introduce accelerated implementations of 3D medical image reconstruction methods, which demonstrate that a considerable acceleration can be achieved by parallel processing. Secondly, we review work in the field of performance comparisons between FPGAs and GPUs, showing that a general decision towards one or the other architecture is difficult.

2.1 Parallel medical image reconstruction

Reconstruction algorithms for use in three-dimensional medical imaging commonly feature an enormous compute and data intensity as well as a high degree of parallelism. As a consequence, a lot of research groups investigated the acceleration of these algorithms by means of parallel hardware architectures. In doing so, special interest has recently been put on GPUs as well as FPGAs.

Most groups, however, focus on the acceleration of well-established imaging methods, like X-ray-based computed tomography (CT) or magnetic resonance imaging (MRI). See [10, 16, 21–23] for a representative set of works. Therein, the authors report significant, but varying, acceleration factors of up to two orders of magnitude using either FPGAs or GPUs instead of conventional software-based processing on a CPU. The applied image reconstruction algorithms in CT and MRT differ from USCT but possess similar characteristics. Thus, the obtained results cannot directly be compared, but give a strong motivation for our work.

Closer to our application is an ultrasound-based imaging system presented by Jensen et al. [11]. They show an FPGA-based data acquisition and processing system for ultrasound synthetic aperture imaging. The overall system is composed of 320 FPGAs and is able to acquire and process 1,024 ultrasound signals in parallel. Also similar to our application, Romero et al. [18] present a GPU-based reconstruction framework. Unfortunately, both do not state an acceleration factor over an equivalent software-based processing approach. They are able to produce SAFT-reconstructed slice images in real time, i.e. over 20 images per second, for their respective system architecture. As both presented systems are composed of planar 2D ultrasound transducer systems, they operate with a much reduced number of recorded A-Scans than our system.

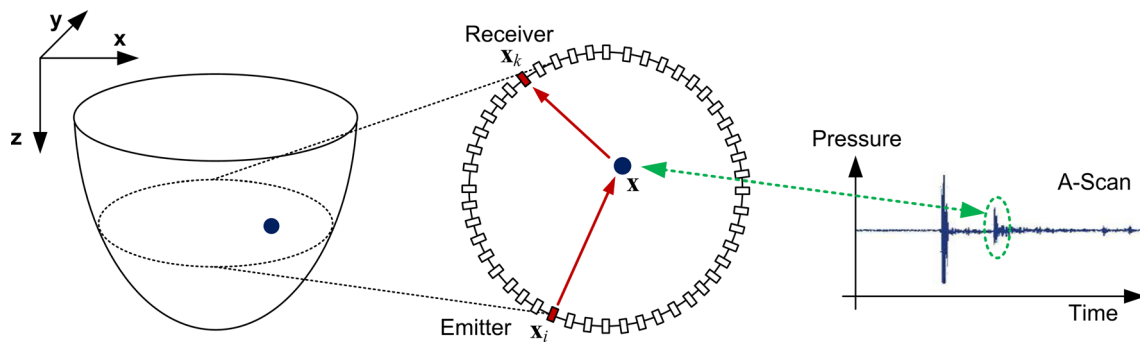


Fig. 2 Simplified principle of the USCT imaging method for an emitter-receiver combination and an arbitrary scattering position projected into an xy -plane: the wave-front emitted at \mathbf{x}_i is scattered at \mathbf{x} and the echo is recorded at \mathbf{x}_k after a certain signal traveling time,

given by the geometric distance of the points \mathbf{x}_i , \mathbf{x} and \mathbf{x}_k and the speed-of-sound. The SAFT algorithm follows the inverse principle to localize scatterers

Thus, the obtained real-time results cannot be expected, but further motivate a direct performance comparison between FPGA- and GPU-based SAFT reconstruction for our system architecture.

classification available, which can be used to determine the best-fitting platform for a given application.

2.2 Performance comparisons: FPGA versus GPU

There exists a body of work presenting performance figures for either FPGAs or GPUs in comparison with a traditional CPU for a wide range of applications, see e.g. [3] for a number of examples. Furthermore, there have been direct comparisons of FPGA-based and GPU-based solutions.

3 SAFT application description

Synthetic aperture focusing technique [7] has been known as a robust imaging algorithm of reflectivity for over 20 years. The early publications date back to the 1970s [17], but have been considered computationally not feasible at that time for medical image reconstruction. The basic idea is to accumulate many low-quality images, recorded by transducers at different geometric positions in order to create one high-resolution image.

For example, Asano et al. [1] compared the processing performance of three different image processing applications on an FPGA, CPU and GPU. With exception of two-dimensional filtering, the FPGA excelled both other applications (stereo-vision and k-means clustering). Similarly, Chase et al. [4] compared performance as well as development time for a real-time capable optical flow algorithm on an FPGA and GPU, achieving a similar performance on both architectures. However, the FPGA implementation took about the tenfold development time.

As discussed in Sect. 1, we therefore record the pressure variation over time for each emitter–receiver combination (A-Scan). Figure 2 illustrates the measurement principle for the A-Scan of emitter i (at position \mathbf{x}_i) and receiver k (\mathbf{x}_k), and an arbitrary scattering position \mathbf{x} in an simplified 2D setup in which all three points are projected into an xy -plane. The emitted ultrasonic wave-front is scattered and the reflection is recorded at a specific point-in-time, given by the distance emitter–scatterer–receiver and the speed-of-sound. The SAFT reconstruction follows the inverse principle. For each assumed scattering position within the USCT device the expected signal arrival times for all recorded emitter–receiver combinations are calculated. The A-Scans’ amplitude values at these specific points-in-time are then loaded and accumulated. By doing so, each A-Scan sample gives a contribution to all voxels on the surface of an ellipsoid, but as a consequence of the conducted superposition of a huge number of A-Scans, the true scattering positions are highlighted most.

Kalarot et al. [13] also compare GPU and FPGA implementations for real-time stereo vision. They found that the custom tailored FPGA implementation is superior in terms of processing performance, while the GPU implementation is much more flexible and still shows the 50-fold performance of a single-core CPU. In contrast, Che et al. [6] undertook a comparison for Gaussian-Elimination, Data Encryption Standard (DES) and the Needleman–Wunsch algorithm. For all of these three compute-intensive applications they conclude that the GPU reaches the highest performance and the FPGA the best computational efficiency, measured in number of execution cycles.

$$I(\mathbf{x}) = \sum_{\forall(i,k)} A \left(t = \frac{\|\mathbf{x}_i - \mathbf{x}\| + \|\mathbf{x}_k - \mathbf{x}\|}{\bar{c}_{(i,k)}} \right) \tag{1}$$

In summary, performance results vary between application domains and particular implementations. None of the platforms seems to be definitely superior. Furthermore, there is currently no comprehensive metric for

$$\|\mathbf{a} - \mathbf{b}\| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2}$$

In practice, the imaged region-of-interest (ROI) within the USCT device has to be divided into a discrete grid of voxels (volume pixels). Assuming an average breast volume of about 10 cm^3 and a maximum geometric resolution of our USCT prototype of 0.2 mm [20], this results in a maximum number of about $1,024^3$ voxels. However, also lower resolution volumes as well as single slice images are of interest. In order to reconstruct the amplitude of voxel $I(\mathbf{x})$, we follow Eq. 1. Beginning with the first A-Scan ($i = 0, k = 0$), we determine the signal traveling time t' for the path emitter–voxel–receiver by dividing the geometric distance of these three points by the speed-of-sound \bar{c} . The preprocessed A-Scans' equivalent pressure value $A(t')$ [19] at this point-in-time is then accumulated. This is then repeatedly done for all further A-Scans (i, k). However, depending on the desired image quality and necessary reconstruction time, also only a subset of all available A-Scans, selected by a maximum angle between emitter–voxel–receiver can be used.

In contrast to the exemplified image-to-signal mapping, the SAFT reconstruction can also be realized as a signal-to-image mapping by determining the voxels on the above-mentioned ellipsoidal surface for each A-Scan sample and then adding these samples' value into the respective voxels. However, the first algorithmic description (image-to-signal mapping) is better suited for a parallelization: The processing can then be parallelized voxel-wise, i.e. different threads process different voxels. As different voxels can be reconstructed independently from each other, each thread only reads once from an A-Scan and accumulates the loaded value into its private voxel. In contrast, when conducting the signal-to-voxel mapping, different threads would process different A-Scan samples and would write non-deterministically into the same output image. In this case, race conditions on memory are inevitable and would have to be circumvented by atomic instruction or a final reduction of separate images, whereas both approaches would degrade the achievable parallel performance.

4 Parallel target hardware

This section briefly introduces the system architectures and parallel processing capabilities of the target hardware.

4.1 CPU: Intel Core-i7 920

We used the Intel Core i7-920 “Bloomfield” CPU as state-of-the-art reference platform. The quad-core CPU has a Nehalem micro architecture [12] and is clocked at a frequency of 2.67 GHz . Simultaneous multi-threading (SMT) allows each core to have two threads in flight in order to avoid memory stalls by interleaving the processing of these

two threads. Besides the task-parallel thread processing, each core has a single-instruction-multiple-data (SIMD) capable processing unit and 128bit wide registers.

Furthermore, each core has its separate Level-1 data cache of 32 kByte size and a 256 kByte large Level-2 data and instruction cache. The overall processor shares the unified Level-3 cache of 8 MByte as well as the off-chip DDR3 system memory. Note that from a programmer's point-of-view, the described memory subsystem can be used transparently, i.e. data elements are automatically loaded and stored throughout the hierarchy. However, this may also lead to performance penalties if the automatic caching mechanisms are not used appropriately.

4.2 GPU: Nvidia Geforce GTX 580

For this comparison, we chose an Nvidia Geforce GTX 580, which is based on the current Fermi (GF110) micro-architecture [15] and is, at the time of writing this paper, Nvidia's highest-performance consumer class single chip GPU. Figure 3 gives an overview of the GPU architecture.

The PCI-Express expansion card has 1.5 GByte of global memory (GDDR5) and the GPU chip a total number of 512 CUDA Cores (CCs) running at 1.54 GHz , each consisting of a floating-point and integer unit. Every 32 CCs are grouped into one streaming multiprocessor (SM). Besides the CCs each SM contains 16 Load/Store-units, a $32\text{k} \times 32\text{bit}$ Register File and four special function units (SFUs) for computation of transcendental functions. Furthermore, each SM has a Texture- and Constant Cache as

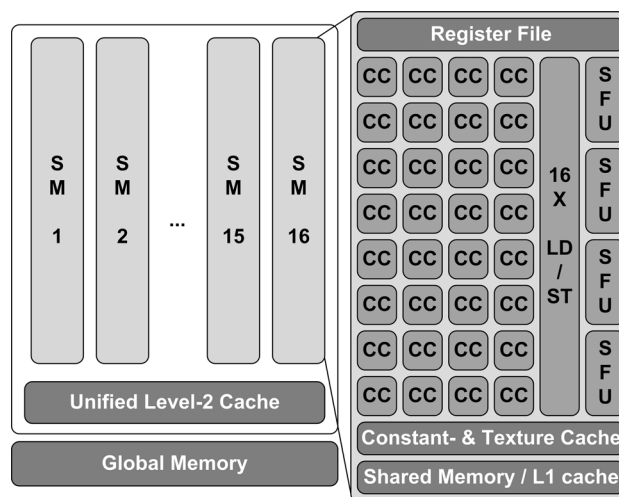


Fig. 3 Block diagram of the Nvidia Geforce GTX 580 GPU consisting of 16 Streaming Multiprocessors (SM). Each SM has 32 CUDA Cores (CC), 16 Load/Store Units, four special function units (SFUs), a large register file and a partly configurable shared memory for usage as L1 cache or user-managed scratch pad memory as well as a Constant- and Texture-Cache. The 768 kByte large L2 cache is shared by all SMs and accesses the external GDDR5 memory

well as a 64 kByte local memory common to all its CCs. This local memory is partly configurable for the use as scratch pad memory or as L1 cache. There is also a unified 768 kByte L2 cache present, which is shared by all SMs and transparently accelerates all accesses of the external memory.

Nvidia’s CUDA programming framework matches the underlying hardware architecture. In order to hide memory access latencies, it uses a single-instruction-multiple-threads (SIMT) strategy, which makes use of extensive hardware multi-threading. For processing, a scalar kernel is launched on a specified number of threads, which are further organized into thread-blocks. Every thread is processed in one CC and a block of threads is local to a particular SM. Except for the caches, data elements have to be moved explicitly between the different types of memory, i.e. registers, shared and global memory. Only the global memory, including its constant and texture memory spaces, is accessible from the host CPU. Within the GPU device, registers are private to a single thread and data in shared memory can be accessed from all threads within the same block. In contrast, the global memory is accessible by all threads, whereas constant and texture memory can only be read from the GPU.

4.3 FPGA: custom data acquisition (DAQ) system

Our custom DAQ system consists of 21 expansion boards: one second level card (SLC) and 20 identical first level cards (FLC) [9]. The overall system holds 81 Altera Cyclone II FPGAs [5], see Table 1 for an overview of the FPGAs’ device features. In the current configuration, the DAQ system is only used to control the measurement procedure and to acquire the ultrasound receiver signals. As the FLCs hold the vast majority of FPGAs within the complete system, only these are considered in this comparison.

An FLC consists of an analogue and a digital part. As we focus on processing capabilities, only the digital part is relevant for this work. A block diagram of the digital part is given in Fig. 4. Each FLC is equipped with four Cyclone II FPGAs; conventionally, we use one of these as a local master (CntrlFPGA) in order to manage communication to the other FPGAs on board and to the SLC. We employ the other three FPGAs (CompFPGA) for signal acquisition. There are two different types of memory on-board available. Each CompFPGA is connected to a distinct static RAM module (2 MB each) and the CntrlFPGA is attached to a large dynamic RAM module (DDR II) of 2 GB. Communication on-board takes either place via a Local Bus with a data rate of 80 MB/s or by direct links between each CompFPGA and the CntrlFPGA (Fast Links, 3×80 MB/s).

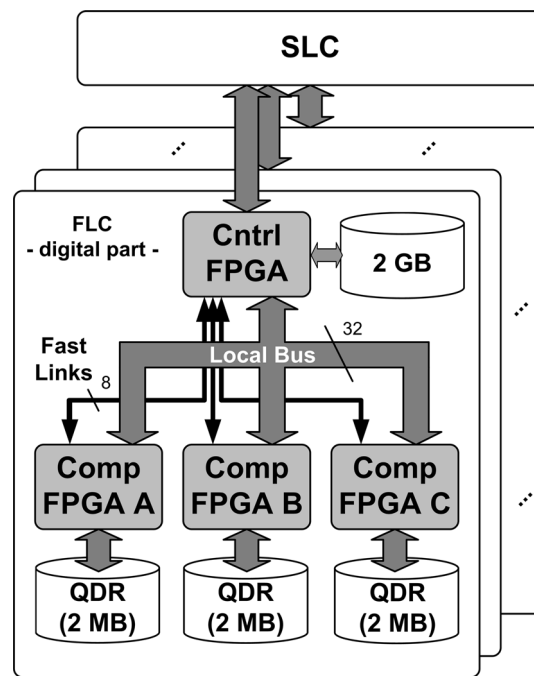


Fig. 4 Block diagram of the digital part of an FLC in the DAQ system. It is equipped with four Altera Cyclone II FPGAs (1 × CntrlFPGA, 3 × CompFPGA). Each CompFPGA is attached to a 2 MB QDR II static RAM module and the Cntrl FPGA to a DDR II dynamic RAM (2 GB). The FPGAs are connected by a Local Bus (80 MB/s) and by direct Fast Links (3 × 80 MB/s)

The DAQ system is currently used only for acquisition and intermediate storage of the A-Scans. This functionality includes signal decimation and pre-filtering, which occupies about 17 % of the available logic elements, 97 % of the embedded multipliers and about 1 % of the memory bits on each CompFPGA. Especially, the extensive use of embedded multipliers makes a dynamic reconfiguration of the overall system necessary, when switching from DAQ to processing mode and vice versa. The reconfiguration of the complete DAQ system takes currently up to 2 min, but could be done in an optimized manner in less than 6 s by a concurrent reconfiguration of all FLCs [2].

4.4 FPGA: Xilinx Virtex-6

As a high-performance counterpart on FPGA side we use a current Xilinx Virtex-6 VLX240T FPGA [24]. The overall FPGA communication system architecture, see Fig. 5, consists of a Peripheral Component Interconnect Express (PCIe) subsystem and an Advanced Microcontroller Bus Architecture (AMBA), Advanced eXtensible Interface (AXI) bus subsystem, whereas the latter also includes a controller for the attached 2 GByte DDR3 memory module. This communication infrastructure occupies 20 % of the available Logic Slices and 5 % of the BlockRAMs.

Table 1 Cyclone II FPGA device elements [5]

Element	Number per FPGA
Logic elements	33,216
Embedded multipliers (9 bit)	70
Total memory bits	483,840

The PCIe subsystem is based on the design¹ of Marcus et al. [14] and has been optimized to meet the 8-lane, generation 1 timing. Furthermore, it has been adapted to serve as an AXI-Lite (control) and AXI-Full (data) bus master. Thus, the host PC is able to access the connected AXI-Lite slaves via memory-mapped Programmable-Input-Output (PIO) reads and writes. The AXI-Full slaves can be read from and written to via autonomous direct memory accesses (DMA). For both, PIO and DMA functionality, the PCIe endpoint implements separate channels so that simultaneous transactions are possible. In order to achieve the necessary PCIe throughput, the endpoint runs at 250 MHz and interfaces to the AXI subsystem running 125 MHz, which doubles the width from 64 to 128 bit.

As indicated above, the AXI subsystem consists of an AXI-Lite and an AXI-Full bus infrastructure and also includes the DDR3 memory controller connected as a slave interface of the AXI bus. All necessary AXI cores have been created using the Xilinx EDK environment. Both bus infrastructures run at 125 MHz and have an address width of 32 bit. The multi-master and burst-capable AXI-Full bus is implemented via a crossbar scheme and has a data width of 128 bit, whereas the single-master AXI-Lite bus uses a shared bus architecture of 32 bit data-width. Due to the separate read write channels of the AXI interconnect, the bus can be operated in full duplex mode. As the native DDR3 memory bandwidth is not saturated by either AXI read or write transactions, the same is true for the DDR3 slave module.

5 Application integration

The following subsections describe our implementations of the SAFT algorithm for the 3D USCT image reconstruction on the CPU, the GPU as well as the Virtex-6 FPGA and the FPGAs in our DAQ system. A simplified data flow diagram of the SAFT processing scheme derived from Eq. 1 can be found in Fig. 6.

5.1 CPU implementation

We parallelized the processing on different processor cores by eight Pthreads and further by usage of SSE instructions in order to exploit the SIMD processing capabilities within

¹ Available online at http://opencores.com/project/pcie_sg_dma.

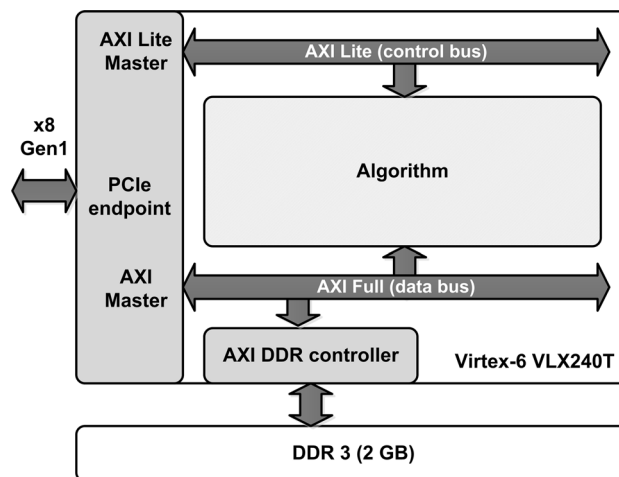


Fig. 5 Block diagram of the Xilinx Virtex-6 device architecture. The communication systems consists of an PCIe endpoint and an AXI subsystem, including an AXI-Lite and AXI-Full bus infrastructures as well a DDR3 controller connected as a slave interface. The PCIe endpoint serves as bus master

each core. Therefore, we partly programmed in plain C and partly in Assembler, and compiled with gcc 4.5.1 and nasm 2.9.3, respectively. The assembler optimized reconstruction kernel originates from our previous work in [25].

As discussed in Sect. 3, we parallelized the processing voxel-wise, so that each thread reconstructs a subset of the voxel grid and uses all selected A-Scans. Within the thread, we perform the actual assembler-optimized SAFT reconstruction. Depending on the dimension of the chosen voxel grid, we divide the overall image in y -direction between the threads, if less xy slices are reconstructed than threads are available, or otherwise in z direction (see Fig. 2 for a definition of the coordinate system). We then loop over the A-Scans and within the reconstruction kernel we loop over all assigned voxels. Two neighboring voxels are processed concurrently and interleaved with the reconstruction of two further voxels in order to achieve a full SIMD utilization in each processing step. All geometric calculations are done in single precision floating point arithmetic. Firstly, the distances emitter-voxel and voxel-receiver are sequentially calculated for the first and the second voxel pair. Then, we add the partial distances and multiply the sum with the precomputed inverse speed-of-sound value for all four voxels in parallel. Finally, the four A-Scan samples are loaded and two-way parallel added into the respective voxels in double precision.

5.2 GPU implementation

All GPU routines, in this context called kernels, were written in CUDA C and use the toolkit version 4.0. We voxel-wise parallelized the reconstruction similar to the

CPU description above. However, as threads are much more lightweight and a huge number of threads are needed to reach a good performance on the GPU, each thread reconstructs only a single voxel. In order to further achieve high data-transfer efficiency, we batch-wise transfer A-Scans between GPU and host memory and then use this batch for image reconstruction. By usage of asynchronous CUDA API calls, both types of operations (data transfers over PCIe and kernel executions) are performed simultaneously. This enables us to apply a double-buffering scheme. After an initial synchronous transfer all further batches are transferred during the previous batch is being processed, and thus data transfer latencies over PCIe are hidden completely. We experimentally determined a batch-size of 11'304 A-Scans, which equals all recorded A-Scans for eight emitters, as a good trade-off between reconstruction flexibility, global memory usage and the efficiency of large data sets. In this case, computation time dominates the transfer via PCIe for all resolutions, larger than an image slice of 192×192 pixels.

In the SAFT reconstruction kernel, each thread reconstructs one voxel. Each thread-block consists of 16×16 voxels in xy -direction (see Fig. 2). In each thread, the voxel's value resulting from potential previous kernel executions is loaded. We then loop over the A-Scans of this batch and within the loop body we calculate the distances emitter-voxel and voxel-receiver, sum up and multiply with the precomputed inverse speed-of-sound. As in the CPU implementation, all geometric calculations are done in single-precision. Then, the determined sample's value is loaded and locally accumulated in double precision. When exiting the loop, the resulting voxel value is stored back to

global memory. After the reconstruction is completed, the voxel field is transferred via PCI-Express from global GPU to host memory.

Due to the necessary indirection in the sample determination, i.e. the calculated point in time has to be used as memory index, the A-Scans' samples are loaded in a non-deterministic manner. Thus, we compare here three different implementations:

- Direct global access: In this implementation the determined A-Scans sample are directly loaded from global memory. This results in cached reads via the L1- and L2-Cache hierarchy and therefore we use the L1 cache configuration of the shared memory.
- Direct texture access: Here, the samples are also directly loaded from global memory, but in contrast to the implementation above, the A-Scans are addressed via the GPUs' texture fetching units, resulting in cached texture reads.
- Shared memory preload: In each iteration of the above-mentioned A-Scan loop, the samples of the complete A-Scans are preloaded into the shared memory. By doing so, the read commands are perfectly aligned and can be coalesced. However, this preload scheme requires an synchronization of all threads within the thread-block before the actual calculation can be begin.

5.3 FPGA implementation: DAQ system

The FPGA implementation is custom-tailored to our DAQ system architecture. As 1/20 of all A-Scans reside in the DDR2 memory of each FLC after the measurement took place, we parallelized here signal-wise between different FLCs and then again voxel-wise on every FLC. Due to the distributed A-Scan storage, this concept is more efficient than broadcasting the A-Scans over the backplane to all other FLCs. As a consequence, after the local reconstruction is finished, each FLC has to send its partial voxel field, which is only reconstructed with a subset of A-Scans, to the SLC for a final reduction. Table 2 gives an overview of the used real estate per FPGA type.

The CntrlFPGA is responsible for reconstruction control on each FLC. The A-Scans are then one after the other loaded from DDR2 memory and sent via Fast Links to the CompFPGAs. Besides the pressure samples, each A-Scan transfer includes a header, containing the emitter and receiver coordinates. After the local reconstruction has finished, the CntrlFPGA receives a subset of voxels from each CompFPGA via Fast Links and sends the partial voxel field to the SLC.

On each CompFPGA, 1/3 of the voxel field is reconstructed tenfold parallel. See Fig. 7 for a block diagram of the processing modules. All modules on the CompFPGA

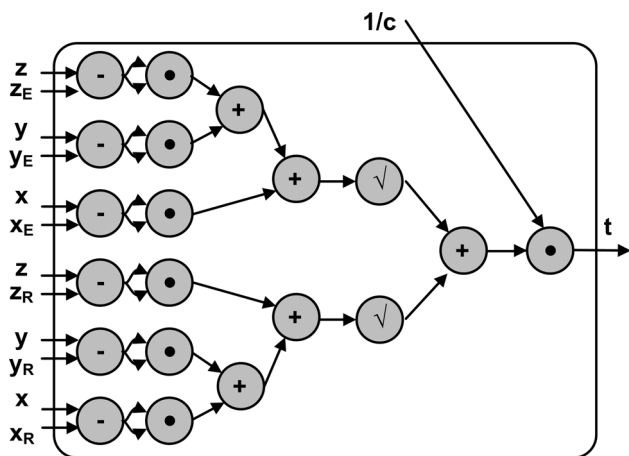


Fig. 6 Data flow diagram of SAFT processing for one voxel $\mathbf{x} = (x, y, z)$, an emitter $\mathbf{x}_E = (x_E, y_E, z_E)$ and a receiver $\mathbf{x}_R = (x_R, y_R, z_R)$ and the precalculated inverse of the related speed of sound value c . The determined index t is used to load the A-Scan sample. This flow has to be repeated for all voxel-emitter-receiver combinations

Table 2 FPGA occupation

Element	CntrFPGA (%)	CompFPGA (%)
Logic elements	39	54
Embedded multipliers (9 bit)	49	100
Total memory bits	3	27

Table 3 Virtex-6 VLX240T device occupation

Element	Total	Occupied (%)	Comm. system (%)	SAFT module (%)
Logic slices	37,680	91	20	71
DSP48E1 slices	768	75	0	75
BlockRam (36kBit)	416	80	5	75

are clocked at a frequency of 60 MHz. After the first A-Scan is transferred, the reconstruction is started by calculating the geometric coordinates for the first 10 voxels in the *PixelGen* unit and afterwards feeding them into the *SAFT units*. In each *SAFT unit*, the respective emitter–voxel and voxel–receiver distances are calculated, summed up and divided by the speed of sound in order to determine the sample indices. These indices are then stored in the *SampleRegister*. The *MemoryManagement* unit loads the ten determined samples one after the other from an A-Scan buffer and feeds them to the *PixelRegister*. There, the needed voxels are already pre-loaded from QDRII memory, then accumulated and afterwards stored back. This procedure is repeated for all voxels before moving to the next A-Scan. In order to hide transfer times over the Fast Links, an A-Scan double-buffering scheme is applied.

5.4 FPGA implementation: Virtex-6

Similar to the GPU, we operate the Virtex-6 evaluation board as a PCIe accelerator, i.e. the reconstruction flow is

initiated and controlled by a host PC. We also batch-wise reconstruct and copy the A-Scans into the DDR3 memory (see Fig. 5). In order to hide transfer latencies, double-buffering is applied on various levels: First, the transfer via PCIe is overlapped with the processing of the previous batch and second, during the actual processing of an A-Scan the following scan is already preloaded from DDR3 memory into an internal buffer.

The SAFT processing modules run at 125 MHz. The overall structure can be divided into a control and a computational part, see also Fig. 8. Besides the AXI-Lite slave module, which implements the host accessible registers, and the AXI-Full master module for accessing the DDR3 memory, the *saftController* consists of four further modules: firstly, the *controlFSM*, which is responsible for the overall processing flow, and the *geoCoordMem* providing the computational units with the emitter and receiver coordinates of the currently processing A-Scan. Data-wise the *saftController* inhibits the *ascanLoad* and *pixelRdWr* modules, which translate the respective transfer requests into AXI bus transaction for either reading a complete A-Scan into the *scanBuffer* or reading and writing the internally buffered pixel set from or to the *pixelBuffer*.

The computational part consists of 32 parallel *saftUnits*, whereas each contains a *pixelCoordGen*, two *sampleUnits*, a *scanBuffer* and a *pixelBuffer*. Beginning with the supplied start pixel x_p , the *pixelCoordGen* generates two successive pixels in each clock cycle, which are then fed into the separate *sampleUnits*. There, the sample indices are calculated according to Fig. 6 and Eq. 1. These indices are then used to address a true dual-ported BlockRAM holding the current A-Scan in the *scanBuffer*. Finally, the loaded sample values are accumulated in the *pixelBuffer*, which contains two 64 bit adders as well as a deep Fifo buffering a image slice of 1,024 pixels. The complete processing is deeply pipelined and has a throughput of 2 pixels per *saftUnit* and 64 pixels in total. Table 3 gives an overview

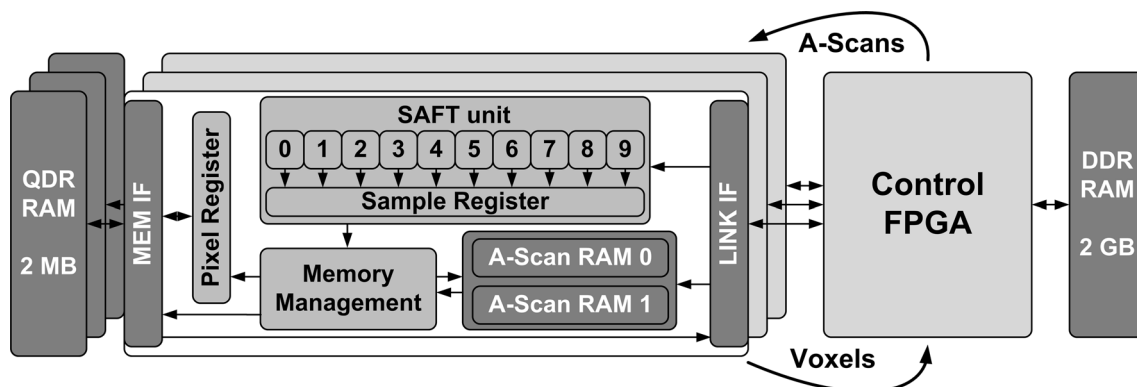


Fig. 7 Block diagram of CompFPGA processing modules. Each FPGA processes 1/3 of the voxel field for all local A-Scans in ten parallel SAFT units. The overall processing, i.e. determining the

A-Scan sample and accumulating the sample value, is divided into four pipeline stages. In order to hide transfer latencies, A-Scans are double-buffered. The voxels reside in QDRII memory

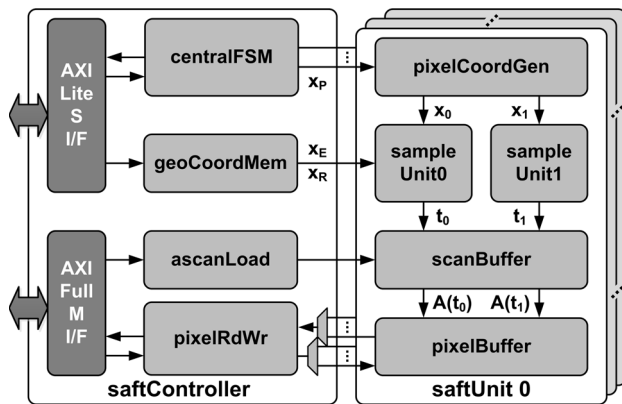


Fig. 8 Block diagram of the SAFT processing module clocked at 125 MHz and consisting of the *saftController* and 32 parallel *saftUnits*. The *saftController* connects to the outer system, see Fig. 5, via an AXI Lite slave and an AXI Full master interface. The *saftController* is responsible for controlling and monitoring the processing flow, whereas the fully pipelined *saftUnits* reconstruct two pixels per unit or 64 pixels in total per clock cycle

of the total number of device elements as well as their percentile occupation by the total system, the communication and the SAFT processing part.

During processing, all A-Scans of the current batch are used one after the other to reconstruct an image slice of $32 \times 1,024 = 32,768$ voxels, which are internally buffered in the *pixelBuffer* modules. At larger image dimensions, the buffered pixels are then exchanged with the next slice, i.e. the current pixel set is written to DDR3 memory and optionally the successive set is read in. This is repeatedly done until the current batch has been used for all pixels and the procedure starts over with the next batch. Due to this necessary pixel exchange at volume dimensions larger than 32k pixels, the batch-size has been maximized to 169,560 A-Scans, optimally using the available 2 GB memory module. In this case, the pixel exchange time accounts only for approx. 0.023 % of the total reconstruction time and thus has only a marginal effect on performance.

6 Results and discussion

In this section, we present our performance results for the SAFT kernel on the CPU and the GPU as well as the Virtex-6 FPGA and the FPGAs embedded in our DAQ system. Firstly, we analyze the attained performance on each platform separately and then directly compare these performance values between different platforms. As a performance metric we use the throughput of reconstructed voxels per second normalized to a single A-Scan. This metric allows for a simple projection of the total SAFT reconstruction time for a given voxel grid and number of used A-Scans.

6.1 Performance analysis and discussion

6.1.1 CPU

The best performance on the CPU is reached at optimal load balancing, i.e. when the parallelized image dimension (y or z) is a multiple of the number of threads. For this case, Fig. 9 shows the detailed voxel throughput over the number of reconstructed voxels for different thread configurations. The single-threaded version saturates at a voxel grid dimension of about 500k voxels to a performance value of 0.2 GVoxel/s. After reaching its peak values, the four and eight threaded configurations start to level off at 1 M voxels, i.e. when the reconstructed voxel grid drops out of the L3 cache of 8 MB, and saturate at about 1.5 M voxels (4 threads) and 2 M voxels (8 threads). From there on the four-threaded configuration using all four physical cores of the processor performs at 0.74 GVoxel/s and a performance of 1.0 GVoxel/s is reached using hyper-threading. When moving from one to four threads, processing accelerates by of factor 3.7. An additional factor of 1.35 is gained with hyper-threading, i.e. using idle cycles of each processor core by processing a further thread.

As neighboring voxel pairs are consecutively processed and tend to have similar signal traveling times for an A-Scan, the loaded samples possess a good spatial and temporal locality, which both is optimally suited for caching. Furthermore, as the reconstructed voxels per core reside side by side in memory, the pre-fetching features of the processor are elegantly used and limit the cache impact to only 37 MVoxel/s for the 8 and 14 MVoxel/s for the four-thread configuration. With a marginal performance drop of 3.7 % for the eight threaded and 1.9 % for the four threaded configuration, the CPU SAFT kernel can be considered greatly cache-independent.

6.1.2 GPU

On the GPU, the x and y resolution has to be a multiple of 16, which is the dimension of a basic thread-block. Figure 10 shows the obtained performance for the GTX 580 again in terms of voxel throughput versus the number of processed voxels. For each three implementation variants (see Sect. 5.2), the performance saturates between 500k and 1 M voxels to its peak value. The *Shared Memory* version shows the lowest performance of 2.8 GVoxel/s, followed by the direct *Global Memory* access with a performance of 14.3 GVoxel/s and by usage of *Texture Memory* and its dedicated fetching units the best performance of 17.2 GVoxel/s is reached.

Apparently, the synchronization within each loop iteration as well as pre-loading a complete A-Scan of 3,000 samples in each tread-block severely limits the performance

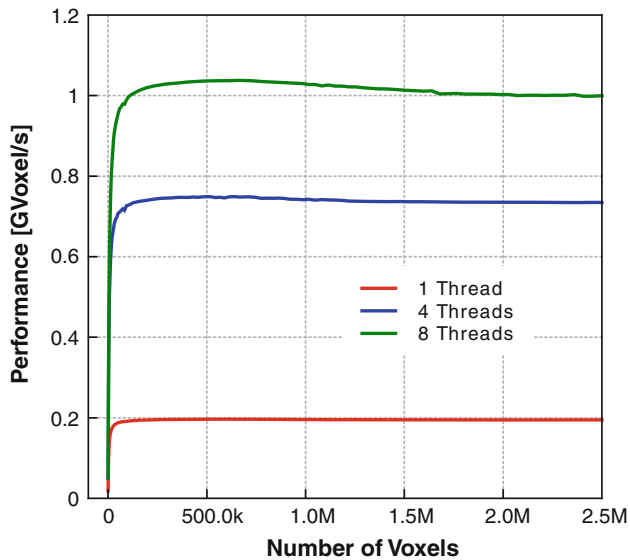


Fig. 9 CPU performance, measured in voxel throughput over the number of voxels for one, four and eight threads

of the *Shared Memory* version and results in a performance degradation of over factor six in comparison to the *Texture Memory* implementation. Furthermore, the usage of texture memory accelerates processing by 20 % in comparison to the direct global memory access. For small voxel grids (smaller than 500k voxels) all three performance curves in Fig. 10 exhibit a complex behavior, caused by a superposition of different effects like load balancing between different streaming multiprocessors. However, as these are only identifiable at very low resolutions and have hardly any influence on the performance of actual 3D USCT reconstructions, a detailed analysis and discussion is left for future work.

In summary, the SAFT algorithm naturally fits to the GPU architecture, as it offers a huge number of independent voxels, no divergent branches, many floating point operations as well as a beneficial communication-to-computation ratio. As each thread-block processes a neighboring region of voxels, the A-Scan caching principles discussed for the CPU also apply on a larger scale to the GPU.

6.1.3 FPGA: DAQ System & Virtex-6

In case of the DAQ System, only power-of-two resolutions have been considered in order to optimize the hardware design. Out of these, only the two medium resolutions of 256^2 and 512^2 pixels have been realized and due to the deterministic and cycle-wise processing both show the same performance of 1.6 GVoxel/s. Besides the parallelization on different FPGAs, the performance is obtained by

pipelined processing, which is, in this custom-tailored form, not possible on both CPU and GPU. The overall processing on a CompFPGA is divided into multiple pipeline stages, whereas the iterative square-root operations constrain a further increase in stages and thus, limit the overall performance. Similarly, the number of SAFT units per FPGA is limited by the available embedded multipliers. A further increase of units using logic elements instead of hardware multipliers reduces the maximum FPGA operation frequency below our margin of 60 MHz.

Note that if only a single CompFPGA is compared, the performance is degraded almost by a factor of 50 in comparison to the CPU version. However, the Altera Cyclone II FPGAs in our DAQ system are already quite dated and not targeted to high performance computing. Although this result is important for our current reconstruction setup as will be shown in the following section, the next paragraph gives insight into the results on a more competitive FPGA platform.

Due to design optimizations also for the Virtex-6 implementation only power-of-two resolutions between 128^2 and $1,024^2$ voxels have been realized, whereas a full utilization of the 32 implemented SAFT units need at least a resolution of 256^2 . From this point on, the Virtex-6 implementation constantly performs at a peak value of 7.8 GVoxel/s. Also here, this performance is reached by extensive pipelining with approx. 100 stages besides the 32 parallel SAFT units.

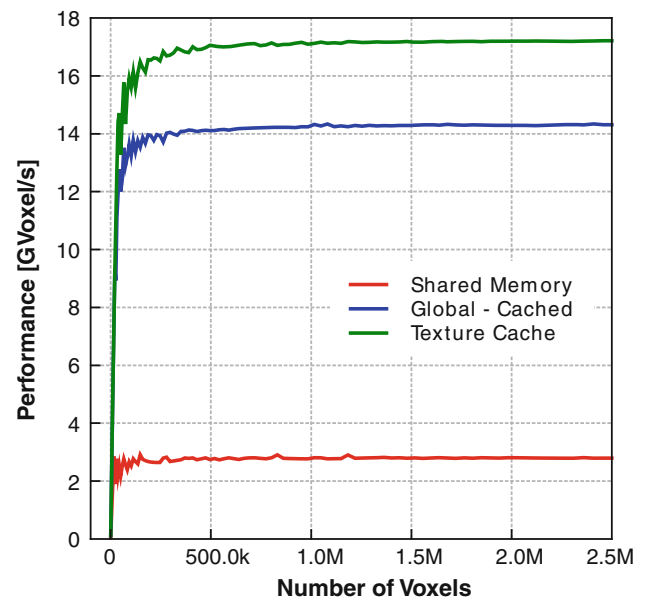


Fig. 10 Performance for global memory, texture memory and shared memory implementation variants on the GTX580 GPU, measured in voxel throughput over number of voxels

Table 4 Performance comparison in GVoxel/s

Platform	Type	Performance
CPU	Intel Core i7 920	1.0
GPU	Nvidia GTX 580	17.2
FPGA	Custom DAQ System	1.6
FPGA	Xilinx Virtex-6	7.8

6.2 Performance comparison

For this comparison, all implementations have been optimized with respect to the target architecture. This leads to individual restrictions in terms of image resolution and A-Scan batch sizes, which have been explained in the last sections. However, none of these limitations constricts the needs of 3D USCT volume reconstructions. As the performance values also mostly depend on the number of reconstructed voxels, we used values taken from the saturation region of all platforms for the direct comparison. Table 4 summarizes the obtained performance values for all investigated architectures.

Our CPU implementation, serving as a state-of-the-art reference, reaches a performance of 1.0 GVoxel/s. The DAQ system, combining all 60 CompFPGAs, shows a performance of 1.6 GVoxel/s and thus accelerates processing approx. 1.6-fold and the Virtex-6 FPGA implementation performs at 7.8 GVoxel/s (acceleration factor 7.9). Finally, the GTX580 GPU reaches the highest performance of about 17.2 GVoxel/s, which is over an 17-fold increase in comparison to the CPU, an 11-fold increase to the DAQ system and an 2.2-fold increase to the Virtex-6 FPGA.

7 Conclusion and outlook

In this work, we compared the processing performance of our SAFT-based image reconstruction kernel on a GPU, a single a high-performance Virtex-6 FPGA and our custom FPGA-based DAQ system with a recent multicore CPU. Our results show that all system architectures accelerate processing and the GPU reaches by far the highest performance.

Finally, we show use-cases for the different accelerated versions within the 3D USCT image reconstruction sequence. One interesting aspect is the reconstruction of medium-quality slice images providing an estimation of the overall data quality and a localization of the breast within the USCT device for the following high-resolution volume reconstructions. As the A-Scans reside in the DAQ system after a measurement and have to be transferred to the PC in order to be processed by either CPU or GPU, this time can be profitably used by DAQ system reconstructions. This

transfer currently needs about 30 min for the full data-set using a 100 Mbit/s Ethernet connection, which allows up to 11 slice reconstructions of 512×512 pixels with 1/4 of all A-Scans, including the non-optimized reconfiguration time. As a further advantage, the patient can be still present at the time the first slice reconstructions are finished (less than 5 min), giving the opportunity to repeat the examination in case of parameter maladjustments or measurement errors, which can be caused, for example, by movements of the patient. Once the measurement data are transferred to the PC, the processing performance of the GPU is accessible. Considering a reconstruction setup of 64 slices of $1,024 \times 1,024$ slices using 1/2 of all A-Scans, the SAFT processing time is reduced from over 1 day on the CPU to about 2 h on the GTX 580 GPU. As detailed in Sect. 3, the reconstruction of different voxels is independent and thus, by dividing the overall voxel grid and using multiple GPUs, this time can almost be linearly scaled down with the number of used GPUs.

Furthermore, as already a single current FPGA has almost half the performance of the GPU and a substantial number of FPGAs is and will be needed for data acquisition in 3D USCT, FPGA-based image reconstruction is nevertheless a viable solution for the next generation of our data acquisition system, which will be even more targeted towards data processing. Also here, reconstruction time could be scaled down by distributing independent voxels between multiple FPGAs.

Our plans for future work include the steady improvement of the accelerated versions of our imaging algorithms in terms of quality and processing speed. For example, in the above implementations, each A-Scan is assumed with an average speed-of-sound. However, a correction for variations within the breast is mandatory for high-resolution and high-quality volumes. Furthermore, we want to investigate the performance differences between GPUs and FPGAs as well as the fundamental application characteristics, which are suitable for either GPU-based or FPGA-based computing in more detail.

References

- Asano, S., Maruyama, T., Yamaguchi, Y.: Performance comparison of FPGA, GPU and CPU in image processing. In: International Conference on Field Programmable Logic and Applications, FPL, pp. 126–131 (2009)
- Birk, M., Hagner, C., Balzer, M., Ruiter, N.V., Huebner, M., Becker, J.: Evaluation of the reconfiguration of the data acquisition system for 3D USCT. *Int. J. Reconfigurable Comput.* **2011**, 9 (2011)
- Brodtkorb, A.R., Dyken, C., Hagen, T.R., Hjelmervik, J.M., Storaasli, O.O.: State-of-the-art in heterogeneous computing. *Sci. Progr.* **18**, 1–33 (2010)

4. Chase, J., Nelson, B., Bodily, J., Wei, Z., Lee, D.J.: Real-time optical flow calculations on FPGA and GPU architectures: a comparison study. In: 16th International Symposium on field-programmable custom computing machines, FCCM '08., pp. 173–182 (2008)
5. Cyclone II Device Handbook. Tech. rep., Altera Corporation (2007). http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf (online, accessed 28-Feb-2012)
6. Che, S., Li, J., Sheaffer, J., Skadron, K., Lach, J.: Accelerating compute-intensive applications with GPUs and FPGAs. In: Symposium on Application Specific Processors, SASP, pp. 101–107 (2008)
7. Doctor, S., Hall, T., Reid, L.: SAFT the evolution of a signal processing technology for ultrasonic testing. *NDT Int.* **19**(3), 163–167 (1986)
8. Ferlay, J., Shin, H., Bray, F., Forman, D., Mathers, C., Parkin, D.: GLOBOCAN 2008 v1.2, Cancer Incidence and Mortality Worldwide: IARC CancerBase No. 10". Tech. rep., International Agency for Research on Cancer, Lyon. <http://globocan.iarc.fr> (Internet, accessed 20-Feb-2012)
9. Gemmeke, H., Berger, L., Birk, M., Goebel, G., Menshikov, A., Tcherniakhovski, D., Zapf, M., Rüter, N.: Hardware setup for the next generation of 3D ultrasound computer tomography. In: IEEE Nuclear Science Symposium Conference Record (NSS/MIC), pp. 2449–2454 (2010)
10. Heigl, B., Kowarschik, M.: High-speed reconstruction for C-arm computed tomography. In: Proceedings of the 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and, Nuclear Medicine, pp. 25–28 (2007)
11. Jensen, J., Hansen, M., Tomov, B., Nikolov, S., Holten-Lund, H.: 8A–3 System Architecture of an Experimental Synthetic Aperture Real-Time Ultrasound System. In: Ultrasonics Symposium, IEEE pp. 636–640 (2007)
12. Intel Microarchitecture Codename Nehalem. Tech. rep., Intel Corporation. <http://www.intel.com/technology/architecture-silicon/next-gen/index.htm> (online, accessed 20-Feb-2012)
13. Kalarot, R., Morris, J.: Comparison of FPGA and GPU implementations of real-time stereo vision. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 9–15 (2010)
14. Marcus, G., Gao, W., Kugel, A., Manner, R.: The MPRACE framework: an open source stack for communication with custom FPGA-based accelerators. In: VII Southern Conference on Programmable Logic (SPL), pp. 155–160 (2011)
15. Nickolls, J., Dally, W.: The GPU computing era. *Micro.* **IEEE** **30**(2), 56–69 (2010)
16. Nol, P.B., Walczak, A.M., Xu, J., Corso, J.J., Hoffmann, K.R., Schafer, S.: GPU-based cone beam computed tomography. *Comput. Methods Progr. Biomed.* **98**(3), 271–277 (2010)
17. Norton, S.J., Linzer, M.: Ultrasonic reflectivity imaging in three dimensions: reconstruction with spherical transducer arrays. *Ultrason. Imag.* **1**(3), 210–231 (1979)
18. Romero, D., Martinez-Graullera, O., Martin, C., Higuti, R., Octavio, A.: Using GPUs for beamforming acceleration on SAFT imaging. In: IEEE International Ultrasonics Symposium (IUS), pp. 1334–1337 (2009)
19. Rüter, N., Schwarzenberg, G., Zapf, M., Gemmeke, H.: Improvement of 3D ultrasound computer tomography images by signal pre-processing. In: IEEE Ultrasonics Symposium, IUS 2008, pp. 852–855 (2008)
20. Rüter, N.V., Goebel, G., Berger, L., Zapf, M., Gemmeke, H.: Realization of an optimized 3D. USCT **7968**(1), 796–805 (2011)
21. Sorokin, N.: Parallel backprojector for cone-beam computer tomography. In: International Conference on Reconfigurable Computing and FPGAs, ReConFig '08, pp. 175–180 (2008)
22. Stone, S., Haldar, J., Tsao, S., Hwu, WmW, Sutton, B., Liang, Z.P.: Accelerating advanced MRI reconstructions on GPUs. *J. Parallel Distributed Comput.* **68**(10), 1307–1318 (2008)
23. Wu, X.L., Zhuo, Y., Gai, J., Lam, F., Fu, M., Haldar, J.P., Hwu, W.M., Liang, Z.P., Sutton, B.P.: Advanced MRI reconstruction toolbox with accelerating on GPU. *Proc SPIE* **7872**(1), 78720Q (2011)
24. Xilinx Virtex-6 Documentation. Tech. rep., Xilinx Corporation. <http://www.xilinx.com/support/documentation/virtex-6.htm> (online, accessed 20-Feb-2012)
25. Zapf, M., Schwarzenberg, G.F., Rüter, N.V.: High throughput SAFT for an experimental USCT system as MATLAB implementation with use of SIMD CPU instructions. *Proc SPIE* **6920**, 692010–692011 (2008).

Author Biographies

Matthias Birk is a Ph.D. student and research assistant at the Institute for Information Processing Technology and the Institute for Data Processing and Electronics at Karlsruhe Institute of Technology, Germany. His research interests include high-performance heterogeneous embedded computing. He has a diploma in electrical engineering and information technology from the University of Karlsruhe.

Michael Zapf is a researcher at the Institute for Data Processing and Electronics at Karlsruhe Institute of Technology. His research interests include embedded and parallel systems, image reconstruction and data acquisition systems. He has a Master of Science in Computer Science from the University of Applied Science Karlsruhe.

Matthias Balzer is a senior researcher and group leader at the Institute for Data Processing and Electronics at Karlsruhe Institute of Technology. His research interests include digital data processing and embedded systems. He has a diploma in Physics from the University of Karlsruhe.

Nicole Rüter is a senior researcher at the Institute for Data Processing and Electronics at Karlsruhe Institute of Technology. Her research interests include 3D ultrasound imaging, medical image reconstruction, digital image and signal processing and biomechanical models. She has a Ph.D. in computer science from the University of Mannheim.

Jürgen Becker received the Diploma degree in 1992, and his Ph.D. (Dr.-Ing.) degree in 1997, both at Kaiserslautern University, Germany. Since 2001, he has been professor for embedded electronic systems at the Institute for Information Processing Technologies (ITIV) at the Karlsruhe Institute of Technology (KIT). His current research is focused on industrial-driven SoCs with emphasis on adaptive embedded systems, e.g. dynamically reconfigurable hardware architectures. He is author and co-author of more than 250 scientific papers, published in peer-reviewed international journals and conferences and active in several technical programs and steering committees of international conferences and workshops. Prof. Becker started in October 2004 as Vice President of Universität Karlsruhe (TH), responsible for the area Studies and Teaching. Since October 2009 to March 2012, he was the Chief Higher Education Officer (CHEO) of the KIT.