Downloaded from orbit.dtu.dk on: Aug 27, 2022

DTU Library

Real-time Volumetric Synthetic Aperture Software Beamforming of Row-Column Probe Data

Stuart, Matthias Bo; Jensen, Patrick Møller; Olsen, Julian Thomas Reckeweg; Kristensen, Alexander Borch; Schou, Mikkel; Dammann, Bernd; Sørensen, Hans Henrik Brandenborg; Jensen, Jørgen Arendt

Published in:
IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control

Link to article, DOI:
10.1109/TUFFC.2021.3071810

Publication date:
2021

Document Version
Peer reviewed version

Link back to DTU Orbit

Citation (APA):
Stuart, M. B., Jensen, P. M., Olsen, J. T. R., Kristensen, A. B., Schou, M., Dammann, B., Sørensen, H. H. B., & Jensen, J. A. (2021). Real-time Volumetric Synthetic Aperture Software Beamforming of Row-Column Probe Data. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, *68*(8), 2608 - 2618. https://doi.org/10.1109/TUFFC.2021.3071810

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TUFFC.2021.3071810, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control

Real-time Volumetric Synthetic Aperture Software Beamforming of Row-Column Probe Data

Matthias Bo Stuart*, Patrick Møller Jensen†, Julian Thomas Reckeweg Olsen†, Alexander Borch Kristensen*, Mikkel Schou*, Bernd Dammann†‡, Hans Henrik Brandenborg Sørensen‡, and Jørgen Arendt Jensen*
*Center for Fast Ultrasound Imaging, DTU Health Technology, Technical University of Denmark
†DTU Compute, Technical University of Denmark
‡DTU Computing Center, Technical University of Denmark

***Abstract*—Two delay-and-sum beamformers for 3-D synthetic aperture imaging with row-column addressed arrays are presented. Both beamformers are software implementations for graphics processing unit (GPU) execution with dynamic apodizations and 3rd order polynomial subsample interpolation. The first beamformer was written in the MATLAB programming language and the second was written in C/C++ with the compute unified device architecture (CUDA) extensions by NVIDIA. Performance was measured as volume rate and sample throughput on three different GPUs: a 1050 Ti, a 1080 Ti, and a TITAN V. The beamformers were evaluated across 112 combinations of output geometry, depth range, transducer array size, number of virtual sources, floating point precision, and Nyquist rate or in-phase/quadrature beamforming using analytic signals. Real-time imaging defined as more than 30 volumes per second was attained by the CUDA beamformer on the three GPUs for 13, 27, and 43 setups, respectively. The MATLAB beamformer did not attain real-time imaging for any setup. The median, single precision sample throughput of the CUDA beamformer was 4.9, 20.8, and 33.5 gigasamples per second on the three GPUs, respectively. The CUDA beamformer's throughput was an order of magnitude higher than that of the MATLAB beamformer.**

I. Introduction

Real-time volumetric imaging requires the ultrasound beam to be steered electronically in both azimuth and elevation, which was first accomplished using a matrix array with individually addressed elements [1], [2]. Such arrays provide the finest level of control of the acoustic beam, but with a highly complicated interconnect. 1D (linear) arrays have around 200 elements to attain high-quality 2D imaging, which translates to 40,000 individually connected elements in a 2D (matrix) array for 3D imaging.

To simplify the interconnect, sparse arrays have been considered, where only a subset of the elements are connected [3]. This increases the element pitch, which results in elevated side- and grating-lobe levels, and reduces the active area and thereby the emitted energy, degrading the signal-to-noise ratio [4]. The sparseness also increases variations in the point-spread function (PSF) – particularly side lobe levels – across the field of view.

Other approaches integrate electronics in the probe handle. Microbeamforming [5] performs delay-and-sum beamforming on groups of elements inside the probe, reducing the number of channels out of the probe. However, programmable electronics are needed in the probe handle, and all element groups have the same relative delays applied to them. More recently, an ASIC for a 240×80 element matrix array was presented [6]. Each row of 80 elements share a transmit and a receive bus without relative delays on the elements' signals before they are summed. Another approach is to use synthetic aperture sequential beamforming (SASB) [7], where low-power electronics apply a fixed focus in receive producing a single output channel from the probe. A second stage beamformer outside the probe then combines a number of these signals to form a dynamically focused volume.

A third approach is to address the rows and columns of the matrix using a crossed electrode design [8]. The interconnect for a matrix array with N^2 elements is thereby reduced to $2N$ channels without any electronics in the probe. Typically, only half the channels are needed in transmit, while the other half are used in receive [9], [10], which enables 3D imaging with the same number of channels as 2D imaging. This paper is based on such row-column (RC) addressed arrays, and Section II gives an overview of and introduction to RC imaging.

The data rate in diagnostic ultrasound imaging systems is typically between one and tens of GB/s of raw radio-frequency (RF) data depending primarily on imaging depth, pulse repetition frequency, and number of receive channels. Traditionally, specialized hardware including field-programmable gate arrays (FPGAs) and digital signal processors (DSPs) have been needed to beamform this data as illustrated by the architectures of research systems [11], [12].

Modern graphics processing units (GPUs) have interfaces that support these data rates, and the first implementations of GPU beamformers were presented in 2011 by Li and Li [13] and by Yiu et al. [14]. At the time, the high-end GPUs' performance was considered acceptable for 2D plane wave and synthetic aperture imaging, but "inadequate" when also considering blood flow imaging and other aspects of a complete ultrasound system [15]. For comparison, today, real-time 2D B-mode and vector flow imaging can be performed on portable consumer devices [16], [17].

Delay-and-sum (DAS) software 3D beamformers have been presented for ring arrays [18] and for individually addressed matrices [19]. For the 64-element ring array, three cross-sectional planes were reconstructed at 45 frames per second (fps) with 24 ksamples per frame, while volumes could be reconstructed at 0.58 volumes per second (vps) with 1.2 Msamples per volume [18]. For a 32×32 individually addressed

0885-3010 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TUFFC.2021.3071810, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control

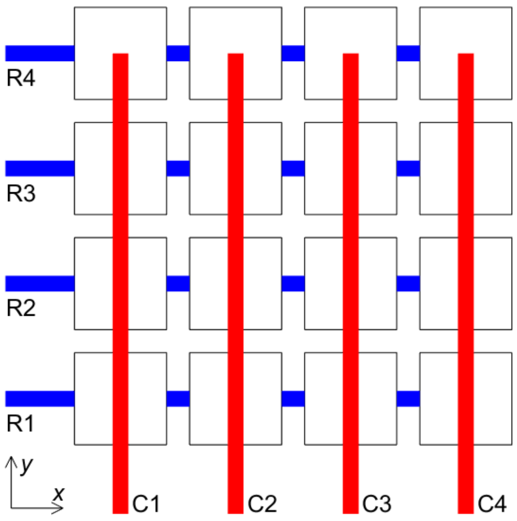


Fig. 1. A RC array is formed by connecting rows of a transducer matrix along the bottom electrode (blue) and columns along the top electrode (red) or vice versa. This conceptually results in two orthogonal linear arrays with tall elements.

matrix array, only 384 channels were used to produce 512 scanlines. Considering only beamforming, a volume rate of 90 vps was attained[1] [19].

This paper presents 3D software DAS beamforming for RC addressed arrays at real-time volume rates. Two GPU beamformers are compared: one written entirely in the MATLAB programming language using the gpuarray class, and one written using the CUDA extensions to the C/C++ programming languages. Both beamformers have been described in proceedings papers [20], [21], and this paper expands on the evaluation and analysis.

Section II gives a brief introduction to beamforming with RC arrays and presents the two beamformers, Section III describes the setup for evaluating the beamformers' performance, Section IV presents the results, Section V discusses the results, and Section VI offers conclusions.

II. Methods

This section first introduces RC arrays and imaging with a brief description of synthetic aperture imaging with RC arrays. Then the two beamformers that were implemented in this study are described.

A. Row-Column Architecture and Imaging

Considering a matrix array of transducers, a RC array is made by short circuiting the bottom electrodes along rows and the top electrodes along columns or vice versa [8]. This is illustrated in Fig. 1. A RC array has rectangular elements with one side length equal to that of conventional arrays and the other side length equal to the width of the entire array. For flat RC arrays, these long elements produce cylindrical waves [10] as opposed to the spherical waves produced by individually addressed matrix elements.

[1]It is unclear how many samples were in a volume.

A number of approaches to RC imaging have been proposed, including variations on DAS [9], [22], [23], [10], and spatially matched filters [24]. Seo and Yen [9] did line-by-line 3D imaging by sliding 64 element transmit and receive windows across a 256×256 element RC array. Sampaleanu *et al.* [23] excited individual matrix elements to attain synthetic aperture focusing using the methods presented by Daher and Yen [25]. However, on the order of N^2 emissions were needed to construct one volume, where N is the number of elements. Rasmussen *et al.* [10] proposed a line-based delay model for the DAS beamformer to account for the cylindrically shaped waves generated by the long elements, and together with an edge-apodization, previously observed range lobes or ghost echoes [22] were suppressed [10], [26].

A consequence of this line-based delay model is that one of the three spatial dimensions can be disregarded in all delay calculations [10], [20] when the rows and columns are parallel to the x- and y-axes in the imaging coordinate system, which is the typical case. This is seen by considering a line element described by its two end points

$$\mathbf{p_1} = (x_1, y_1, z_1) \text{ and } \mathbf{p_2} = (x_2, y_2, z_2), \tag{1}$$

where $x_1 = x_2$ and $z_1 = z_2$ for an element that is lengthwise parallel to the y-axis. The line element is described by

$$\mathbf{l} = \mathbf{p_1} + u\mathbf{v_y}\ ,\ u \in \begin{cases} [0; y_2 - y_1] & , \quad y_1 < y_2 \\ [y_2 - y_1; 0] & , \quad y_1 > y_2 \end{cases} , \tag{2}$$

where $\mathbf{v_y}$ is the unit vector parallel to the y-axis. The time-of-flight between an imaging point $\mathbf{r_{os}} = (x_{os}, y_{os}, z_{os})$ and this element is [10]

$$T = \frac{||(\mathbf{r_{os}} - \mathbf{p_1}) \times \mathbf{v_y}||}{c}, \tag{3}$$

where $||\cdot||$ is the magnitude of a vector, $\times$ is the cross product, and c is the speed of sound. This evaluates to

$$T = \frac{\sqrt{(z_1 - z_{os})^2 + (x_1 - x_{os})^2}}{c}, \tag{4}$$

from which it is seen that the output sample's y-coordinate has been eliminated.

A similar deduction can be made for elements parallel to the x-axis, and for the calculation of dynamic apodizations [20]. A geometrical illustration is given in Fig. 2, where it is seen that the time-of-flight is independent of the image point's y-coordinate.

B. Synthetic Aperture Imaging

Ultrasonic synthetic aperture (SA) imaging [27] is well described in the literature [28], and a brief description of its adaptation to RC arrays [10] is given here.

In SA imaging, a full image or volume is reconstructed from each transmit/receive event in 2D or 3D, respectively. Each of these volumes is called a low-resolution volume (LRV), and the sum of LRVs yields a high-resolution volume (HRV), where dynamic focusing is attained in both transmit and receive [28]. The emission may use a single element or a set of elements producing a virtual source from a diverging [29], a converging, or a plane wave. Dynamic apodization is

0885-3010 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

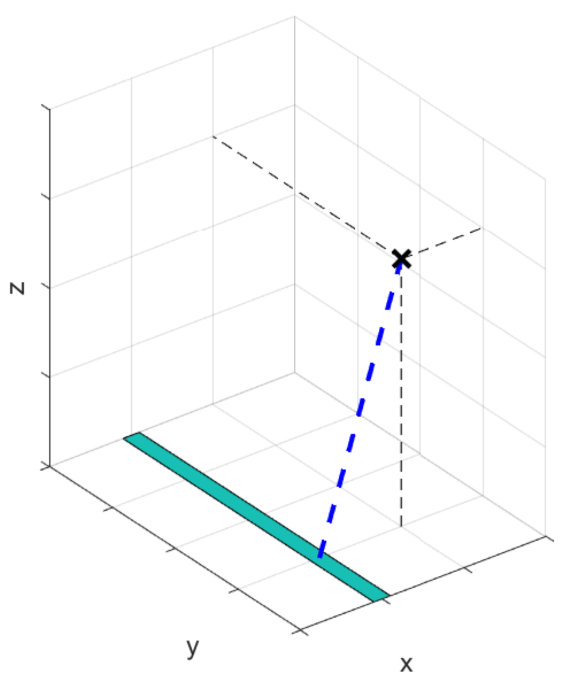


Fig. 2. An illustration that the time-of-flight calculation is independent of the image point's y-coordinate. An element in an RC array is shown in green, and an image point is shown by the black x. The dashed blue line shows the shortest distance (the time-of-flight) between the line element and the image point, and the dashed black lines are guidelines to properly locate the image point in the 3-D coordinate system.

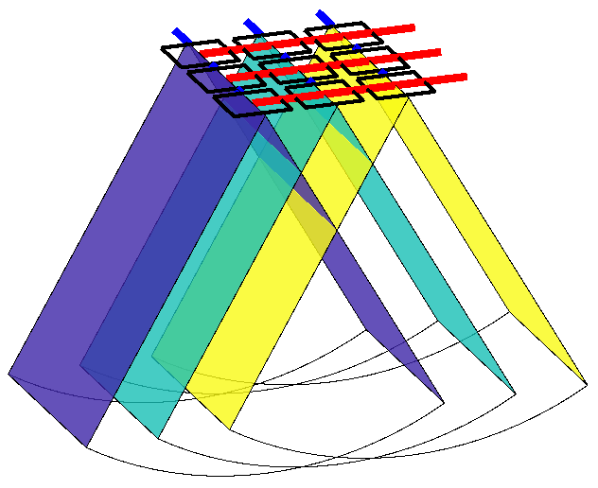


Fig. 3. A SA acquisition with three emissions using the row array and three receiving elements in the column array. The colored planes indicate the opening angle of each emission. The RC array corresponds to the one in Fig. 1, only with three elements in each direction.

often used in both transmit and receive to lower side lobes and maintain uniform resolution with depth.

SA imaging with RC arrays follows the same principle. However, the cylindrical waves prohibit focusing in the length direction of the line elements, and transmit and receive must therefore be performed with orthogonal arrays, e.g., transmit with rows and receive with columns [10]. Fig. 3 illustrates a SA acquisition with three emissions with row elements and three receiving column elements. Each emission insonifies the region between the colored planes, and the corresponding LRVs therefore only contain signal (as opposed to noise) within these regions. Focusing in the y-direction is attained by summing the LRVs from each of the three emissions, and focusing is only attained in the overlapping regions between LRVs. Focusing in the x-direction is attained by dynamically focusing the receive elements in each LRV. This potentially produces different PSFs in the x- and y-directions, if e.g. the x-direction is focused by 64 receive elements, but the y-direction is focused by only 16 emissions. An isotropic PSF may be attained by combining LRVs created with row as well as column emissions [10]. In this work, emissions were made with rows only, while columns were used in receive.

C. Beamformers

This section describes the two beamformer implementations that were evaluated. The first beamformer was implemented in the MATLAB programming language and uses the `gpuarray` class provided by the programming language to executed on the GPU. The other beamformer was implemented in C/C++ with CUDA extensions and a MATLAB MEX interface.

Both beamformers calculated delays and apodizations in 2D as described in Section II-A. They both accepted single or double precision, real or complex valued inputs, and the output values were of the same type as the input. Subsample delays were attained through interpolation.

Preliminary results [20], [21] indicate that the CUDA beamformer clearly outperforms the MATLAB beamformer. They are, however, both included here to quantify the difference in performance and thereby provide an indication of the expected performance improvement when moving from MATLAB to CUDA.

1) MATLAB Beamformer: The MATLAB programming language provides a class named `gpuarray` that stores numeric variables and performs operations on them on the GPU. The class supports both single and double precision floating point data types, and many of the programming language's built-in functions have GPU implementations.

The beamformer was implemented with matrix operations, e.g., delay and apodization values of all output samples were calculated in one matrix before being applied. This minimized the use of loops to take advantage of the optimized matrix operations provided by the programming language. The disadvantage was the need to store large matrices with intermediate results in main memory.

Output sample coordinates were provided as a matrix of coordinates, and dynamic and user-specified apodizations could be enabled in both transmit and receive. In the evaluation, dynamic apodizations were enabled, while the user-specified apodizations were set to matrices of all ones. Subsample delay interpolation was performed using cubic convolution provided by the programming language. Matrix dimensions were chosen to optimize parallel access, e.g., output sample coordinates were stored in memory with all the x-coordinates first followed

0885-3010 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

by all the y-coordinates and finally the z-coordinates.

2) CUDA Beamformer: The CUDA extensions to the C/C++ programming language provide an application programming interface to nvidia GPUs [30]. CUDA is a single-instruction multiple-thread (SIMT) architecture, where many threads execute the same instructions, but on different data. A program must specify the division of its data on a set of thread blocks, and threads should have equal (or similar) workload and a minimum of synchronization for efficient execution. The general beamforming problem has two immediate approaches for distribution across threads: 1) for a window of input samples, calculate the output values these input samples contribute to, and 2) for a given output sample, load the corresponding input samples and calculate the output value. The first approach is likely to result in uneven workloads between threads and requires synchronization for summing the different contributions to each output value. The second approach has equal workload for all threads and requires no synchronization. The CUDA beamformer therefore calculates one output sample per thread, i.e., each thread calculates the sum across all receive channels.

The threads in a block share hardware including memory caches. It is therefore beneficial to group threads in blocks that operate on the same data. The degree to which this can be attained depends on the output geometry. In 2D, an input sample contributes to output samples along an ellipse, so a regular output grid could be partitioned in subgrids that minimize the difference in time-of-flight within the subgrid, thereby minimizing the input data size to the subgrid. This concept can readily be extended to 3D. However, the beamformer presented here was designed for more general use including directional beamforming for synthetic aperture vector flow imaging [31], where the output samples do not lie on a regular grid. Instead, the output geometry was specified as a set of lines with an origin and a step vector containing both direction and step size. To minimize overhead in distributing output samples on threads, the number of samples was the same for all lines. Each thread block corresponded to an output line, and output lines with more than the maximum number of threads per block (1,024) were split in multiple blocks. This is illustrated in Fig. 4.

Each thread computed the sum across receive channels to form a voxel in a low-resolution volume. This voxel needed to be summed with those from the other low-resolution voxels. To support concurrent acquisition and processing this running sum was stored in main memory: once a thread calculated its low-resolution voxel, the corresponding high-resolution voxel was read from memory, added to the low-resolution voxel that was just calculated, and the result written back to main memory. The overall algorithm and its memory accesses are illustrated in Fig. 5. Alternatively, the sum could be computed locally, but that would require data from all emissions to be available before beamforming was started.

Subsample interpolation is needed to attain low side-lobes [32] and was attained by third order Lagrange polynomial

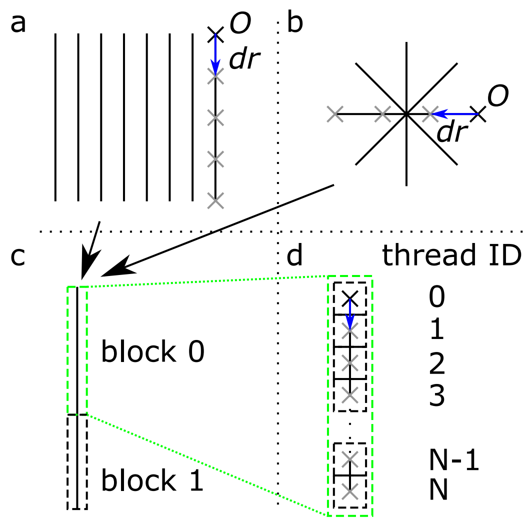


Fig. 4. An illustration of beamformed geometries and mapping to CUDA blocks and threads. (a) shows the typical geometry for a 2-D B-mode image, and (b) shows the typical geometry for synthetic aperture velocity estimation. Lines are defined by an origin, O, shown by a black marker and a step vector dr. Samples along the lines are shown with grey markers. (c) shows the mapping of a line to CUDA blocks, where several blocks are needed only for lines with more than 1,024 samples. (d) shows that each sample in a line is mapped to a CUDA thread.

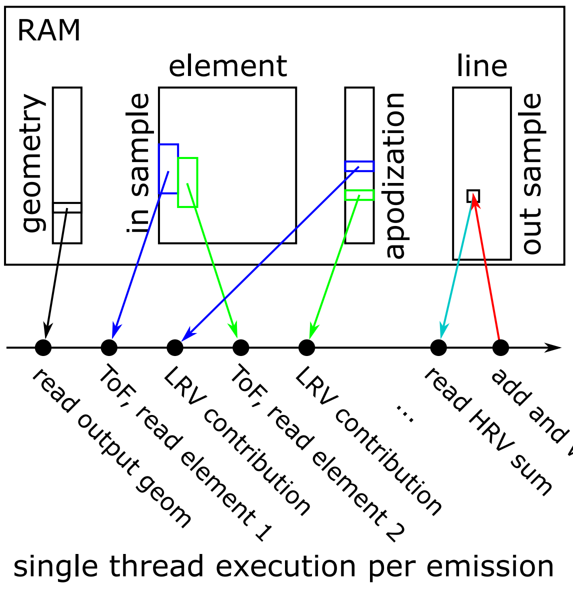


Fig. 5. An illustration of the algorithm and memory accesses by a single thread for each emission. The RAM contains separate arrays with image geometry, input data, sampled apodization windows, and beamformed data. Each thread calculates the coordinates of its output sample, calculates low-resolution voxel contributions iteratively across receive elements, and finally reads the high-resolution voxel from memory, adds the contribution, and writes the updated value back. Each thread only writes to RAM once in its lifetime. The transmit apodization array and a single read from it are omitted from the illustration.

interpolation [33][2]. Dynamic apodization required apodization values to be computed according to a user-specified window function. Here, linear interpolation in a sampled window was used to avoid a number of GPU-related issues including

[2]This interpolation method was originally published by Waring in 1779, but it is named after Lagrange, who published it in 1795.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

thread-divergence and limited computational units for transcendental functions often used in windowing functions.

III. Performance Evaluation Setup

This section first presents the different acquisition sequences and corresponding beamforming used in the evaluation. Next, the different GPUs used in the evaluation and their key features are described. Lastly, the performance metrics used are explained.

A. Scan Sequences and Beamforming Setups

The beamformers were evaluated on all combinations of two simulated phantoms, two RC arrays, two beamformed geometries, single or double precision calculations, real or complex input data, four different scan sequences, and three GPUs. These seven categories of variations result in a matrix of performance metrics for each beamformer with seven dimensions, most of which are binary.

The first phantom was a single point target at (0,0,50) mm, and the second phantom were point targets at $x = y = 0$ and z varying from 10 to 150 mm in steps of 10 mm. The simulated RC arrays were a 64+64 and a 192+192 element, 3 MHz array with 270 μm pitch and integrated edge-apodization to suppress range lobes [10].

The two geometries were 1) two cross planes with $x = 0$ and $y = 0$ respectively, and 2) a full volume. The z range depended on the phantom. For phantom 1, beamforming was performed in the range 40 to 60 mm, for phantom 2, it was done from 0 to 160 mm. The sampling density in x and y was equal the array pitch with line origins located between the element centers, i.e., 63 lines were made in each direction for the 64+64 array. The sampling density in z depended on whether real or analytic signals were beamformed. For real signals, the Nyquist limit must be observed in the beamformed output, and the sampling density was therefore $\lambda/8$. For analytic signals, the sampling density was $\lambda/2$.

The four scan sequences used 4, 16, 64, and 192 single element emissions for both arrays, except the 64+64 element array was not evaluated for 192 emissions. The transmitting elements were uniformly distributed over the array. The number of calculations are independent of whether a single element is used or a virtual line source is created by using multiple elements in transmit. Short sequences are desirable for, e.g., blood flow imaging, while longer sequences improve contrast for higher B-mode image quality.

For the MATLAB beamformer, the storage of intermediate results in GPU RAM necessitated that the output volume be split in two or more subvolumes in many evaluations. The number of subvolumes needed for a given setup was not readily predictable since it depended on any additional memory allocations within MATLAB functions. Initially, the beamforming was run on one subvolume equal to the full volume. If an out-of-memory error occurred, the number of subvolumes were doubled. This process was iterated until the beamforming succeeded or until more than 64 subvolumes were needed. The amount of GPU RAM was the limiting factor, when the processing needed to be split in any number of subvolumes. The threshold of 64 subvolumes was chosen to include performance evaluations for some of these cases while avoiding execution times in excess of 1 hour in extreme cases.

Simulations were made with Field II [34], [35], and the simulated data was matched filtered and decimated to a sampling frequency of 12 MHz, i.e., four times the transducers' center frequency.

While single precision floating point numbers provide sufficient precision for most real-time ultrasound imaging applications, double precision was included in the investigation, since it may be required in some applications. It can also be useful in the early stages of development to not need to consider the potential effects caused by loss of precision. Ultimately, the trade-off in performance (execution time), precision (including floating and fixed point), and development and verification time must be considered for the system being developed.

B. GPUs

The beamformers were evaluated on three different nvidia GPUs: a 1050 Ti, a 1080 Ti, and a TITAN V. Here, the main architectural aspects which impact the GPU beamformers are highlighted. Other implementations may be made to use other aspects of the architectures.

The TITAN V is of the Volta architecture [36] and made for scientific computing, while the 1050 Ti and 1080 Ti are of the Pascal architecture [37] and made for general purpose computing. The main difference is that the TITAN V has twice the number of single precision floating point computational cores as double precision cores, while the ratio is 32 to 1 for the other two GPUs.

Each GPU contains a number of streaming multiprocessors (SM) that are assigned blocks of threads to execute. Several architectural and implementation aspects combine to limit the number of threads per SM, and the relevant aspects will be discussed in Section V. The 1050 Ti has 6 SMs, the 1080 TI has 28, and the TITAN V has 80.

The GPUs also differ with respect to memory. The 1050 Ti has 4 GB memory with a theoretical bandwidth of 112 GB/s, the 1080 Ti has 11 GB with 484 GB/s bandwidth, and the TITAN V has 12 GB with 652.8 GB/s bandwidth.

C. Performance Metrics

The beamformers were evaluated on both the number of samples beamformed per second and on the corresponding volumetric imaging rate for the different setups. The number of lines, N_l, were $2{\times}63 = 126$ and $63{\times}63 = 3,969$ for the cross-plane and volume geometries respectively for the $64 + 64$-element probe and $2 \times 191 = 382$ and $191 \times 191 = 36,481$ for the $192 + 192$-element probe. The number of samples per line, N_{sl}, were 298 and 76 for phantom 1 with real and analytic signals, respectively, and 2,224 and 557 for phantom 2.

The performance was measured as the average execution time, t_m, of up to seven repetitions of the same setup. A total of ten repetitions were run with the first three ignored in the timing measurements to eliminate effects of, e.g., GPU- and cache-initialization. Input data was preloaded in system

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

(not GPU) memory to avoid any effect of disk accesses. Transfers in to and out of GPU memory were included in the timing. Outliers may occur from interrupts and other operating system events in the test setups. This is a general problem of predictable software execution times and may be handled by using a real-time operating system that provides guaranteed services. Outliers were defined as an execution time greater than two standard deviations above the mean and were removed. No more than one outlier was found in any test case.

The beamformer throughput in samples per second was calculated as

$$B = \frac{N_l N_{sl} N_{LRV} N_{rx}}{t_m}, \tag{5}$$

where N_{LRV} is the number of LRVs and N_{rx} is the number of receive elements. The frame or volume rates (collective denoted volume rates) were calculated as

$$V = \frac{1}{t_m} \tag{6}$$

The performance across different setups followed a non-Gaussian distribution, and aggregated results are presented on the form (25-pct, med, 75-pct), where med is the median and 25-pct and 75-pct are the quartiles.

The GPUs were compared by normalizing the performance metrics to those of the 1050 Ti. The beamformers were compared by normalizing to the MATLAB beamformer.

IV. Results

Fig. 6 shows B-mode images of the cross-planes for the phantom with large depth range imaged with 64 emissions from the 192-element array and contour plots of the point target at 50 mm depth. The grating lobes in the YZ plane are caused by spatial undersampling of the transmit array, since 64 uniformly distributed single-element emissions were used giving an effective pitch of 1.5λ or three times the array pitch.

It was verified that all combinations of GPUs, numerical precision, and beamformer produced identical PSFs within acceptable tolerances.

Below, the MATLAB beamformer's memory use is first presented, and then the volume rates and throughput are presented for each GPU and beamformer. Finally, the GPUs and beamformers are compared.

A. MATLAB Memory Use

The MATLAB beamformer required the output to be partitioned in more than 64 subvolumes for the deep phantom with full volumes and the large array when using real-valued signals (1080 Ti and TITAN V) and either real- or complex-valued signals (1050 Ti). No results are thus reported for these setups.

B. Volume Rates

Fig. 7 shows the volume rates of the CUDA beamformer with single precision calculations. The horizontal line is at 30 Hz, which is commonly used as the real-time imaging limit. The volume rates spanned seven orders of magnitude from mHz to kHz depending on imaging depth, geometry, number of probe elements, number of LRVs, numeric precision, real- or complex-valued samples, and GPU.

In general, beamforming complex samples yielded higher volume rates than real samples by a factor (2.0, 2.4, 2.9) for the 1050 Ti, (1.5, 2.1, 2.7) for the 1080 Ti, and (1.2, 2.0, 2.4) for the TITAN V. Even though each complex sample had two components, the axial sampling rate was lower by a factor four compared to real samples. When using real samples, the Nyquist limit must be satisfied for the output RF signals, while complex samples provided the instantaneous envelope in each sample and only needed to satisfy the Nyquist limit for the envelope, which was of much lower frequency.

Out of the 56 single precision setups investigated, the 1050 Ti attained real-time synthetic aperture beamforming for 11 setups, the 1080 Ti for 18 setups, and the TITAN V for 22 setups. All GPUs satisfied the real-time limit for cross-planes at shallow depths using the 64 element probe with 64 emissions for high-quality B-mode imaging. The TITAN V additionally satisfied this limit for full volumetric imaging and for shallow depth cross-planes with the 192 element probe. This depth-range is suitable for a range of applications including various rodent organs (kidneys, brain) and human thyroids that are of interest in current super-resolution research as well as major blood vessels including a range-limited view of the aorta.

At large depth ranges, the 1080 Ti and TITAN V provided real-time beamforming of cross-planes with 16 emissions on the 64-element probe or four emissions on the 192-element probe. Only the TITAN V attained real-time beamforming of full volumes at large depth ranges and then only with four emissions on the 64-element probe.

Similar trends were found for double precision calculations, albeit at lower volume rates for the 1050 Ti and 1080 Ti: Real-time volume rates were attained in two setups for the 1050 Ti, 9 setups for the 1080 Ti, and 21 setups for the TITAN V. The difference to single precision is partially explained by the different number of single and double precision computational units as described in Section III-B.

The MATLAB beamformer did not attain real-time beamforming for any setup on any GPU. The maximum volume rates were 8, 7, and 11 Hz for the 1050 Ti, 1080 Ti, and TITAN V, respectively. The MATLAB beamformer's performance is discussed in terms of throughput below.

C. Throughput

Throughputs are reported in 10^9 samples per second (Gs). The highest throughputs were found for the CUDA beamformer on the TITAN V GPU, which had the highest computational power and memory bandwidth of the tested GPUs. Fig. 8 shows the TITAN V single precision throughput of all setups. A small dependence on the number of low-resolution images was seen: as more emissions contributed to a volume, a smaller portion of the total time was spent on reading out results. This translated to higher throughputs. For the same reason, larger numbers of output samples (Fig. 8b and d compared to a and c) resulted in higher throughputs.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

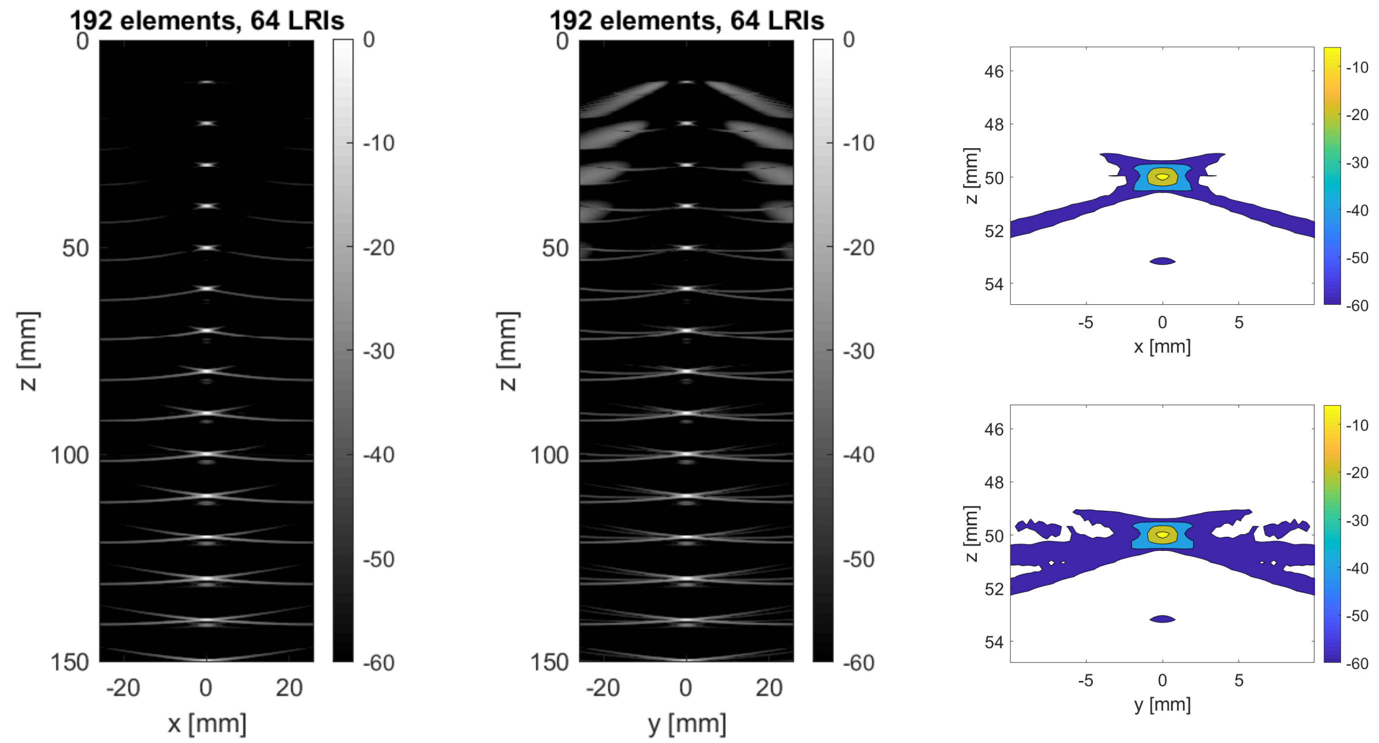


Fig. 6. B-mode images of the XZ (left) and YZ (center) cross-planes of the phantom with large depth range for the 192-element array with 64 emissions. Contour plots (right) of the point target at 50 mm depth with -6, -20, and -40 dB contours.

Using real-valued samples, the TITAN V throughputs were $(26.6, 33.8, 39.4)$ Gs (double) and $(29.6, 33.5, 45.9)$ Gs (single). The single precision samples required half as much data to be transferred, but in the same amount of transfers. Since the throughput was independent of numeric precision, memory bandwidth did not limit the CUDA beamformer on the TITAN V. The throughput was lower for complex-valued samples at $(10.0, 13.5, 20.6)$ Gs (double) and $(9.5, 18.0, 28.9)$ Gs (single) even though the volume rate was higher as described above.

The ratio between single and double precision was greater on the 1050 Ti and 1080 Ti GPUs. Using real-valued samples, the 1080 Ti throughputs were $(2.7, 3.0, 3.1)$ (double) and $(19.0, 20.8, 26.8)$ (single), and the corresponding 1050 Ti throughputs were $(0.6, 0.6, 0.6)$ (double) and $(4.1, 4.9, 6.0)$ (single). The ratios of single to double precision throughput were $(6.7, 8.3, 10.4)$ (1050 Ti) and $(6.3, 6.8, 9.5)$ (1080 Ti). These results indicate that the double precision throughput was limited by arithmetic units, while the single precision throughput was limited by other factors as discussed in section V.

Fig. 9 shows the MATLAB beamformer's throughputs on the TITAN V. A larger dependence on number of low-resolution images was seen. This corresponds well with the greater reuse of precalculated receive delays and apodizations. A larger dependence was also seen on problem size with low performance for small problems (Fig. 9a) and a limit was quickly reached as seen by the near identical graphs in 9b and d.

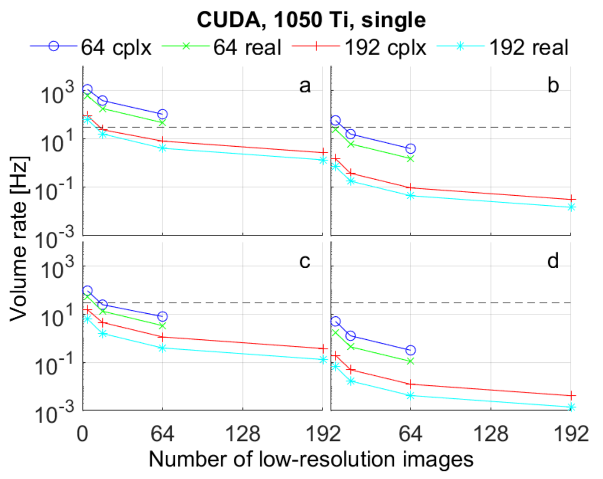
The TITAN V single precision throughput for the MATLAB beamformer was $(0.2, 1.0, 1.6)$ (complex) and $(0.7, 1.8, 3.4)$ (real). The ratio of single to double precision throughput was $(0.9, 1.0, 1.3)$ (complex) and $(0.9, 1.0, 1.1)$ (real) showing that performance was independent of precision. The single to double ratio was similar on the 1080 Ti and only slightly higher on the 1050 Ti with $(1.0, 1.3, 1.4)$ (complex) and $(1.2, 1.3, 1.4)$ (real). This indicates that the MATLAB beamformer's performance was not limited by the number of computational units, but rather by the need to move intermediate results of each basic operation in and out of RAM.

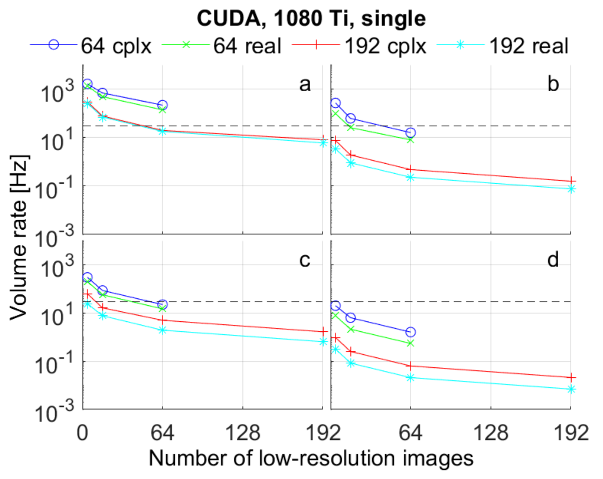
D. GPU Comparison

The median speed-up of the 1080 Ti over the 1050 Ti for the CUDA beamformer was 4 to 5 across both precision and real or complex-valued samples with quartiles within ± 1 of the median. A larger speed-up was seen for the TITAN V at $(45.0, 55.2, 65.1)$ (double real) and $(5.7, 8.0, 9.0)$ (single real). The large double precision speed-up primarily stemmed from micro-architectural differences as described above. At the extremes, the TITAN V's single precision speed-up was no less than 1.6 and up to 10 times.

The speed-ups were smaller for the MATLAB beamformer. The 1080 Ti median speed-up ranged from 1.9 to 3.6, while the TITAN V speed-ups were $(2.2, 6.5, 7.9)$ (double real) and $(1.6, 5.6, 6.2)$ (single real). The similar speed-ups between double and single precision indicate that the MATLAB beamformer was unable to utilize the greater theoretical double precision performance of the TITAN V.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.





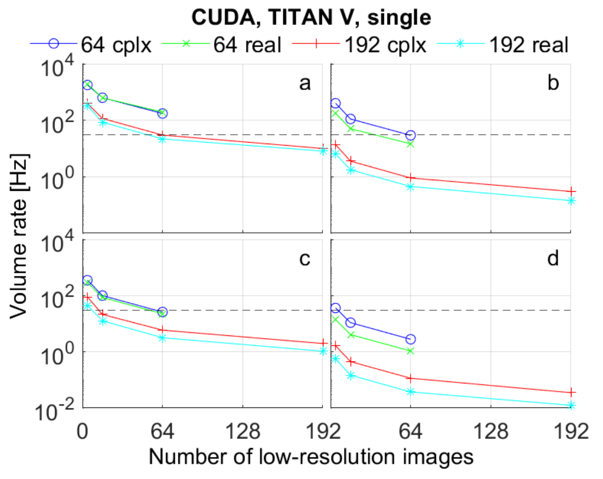


Fig. 7. Volume rate of the CUDA beamformer with single precision calculations on the 1050 Ti (top), 1080 Ti (middle), and TITAN V (bottom). (a) and (b) show throughputs for the shallow phantom, and (c) and (d) show for the deep phantom. (a) and (c) show for cross planes, and (b) and (d) show for full volumes. The horizontal line shows a volume rate of 30 Hz, often considered the limit for real-time imaging.

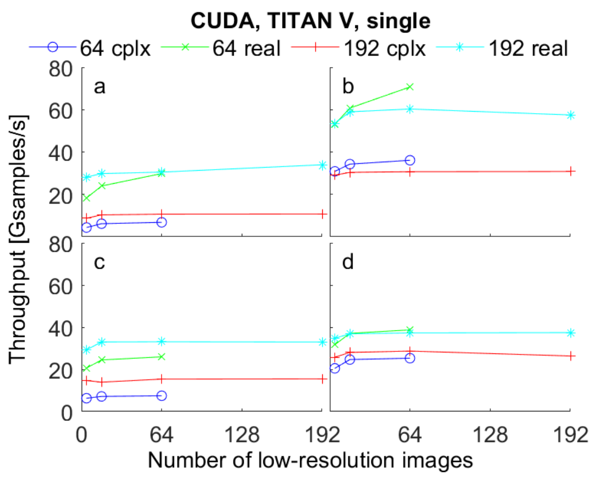


Fig. 8. Throughput as function of number of low resolution images for the CUDA beamformer on the TITAN V GPU with single precision calculations. (a) and (b) show throughputs for the shallow phantom, and (c) and (d) show for the deep phantom. (a) and (c) show for cross planes, and (b) and (d) show for full volumes.

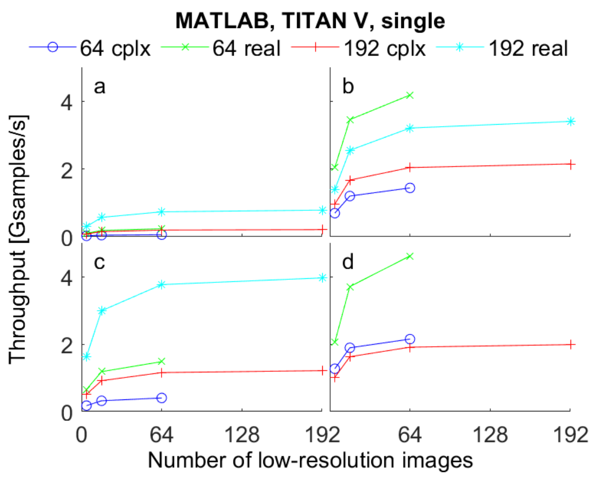


Fig. 9. Throughput of the MATLAB beamformer on the TITAN V GPU with single precision calculations. Subfigures labeled as in Fig. 8.

E. Beamformer Comparison

The CUDA beamformer attained a much greater throughput than the MATLAB beamformer. The single precision median speed-up ranged from 12 for the 1050 Ti to 24 times for the 1080 Ti and TITAN V. The minimum speed-up was 8 times for the TITAN V across all setups, and the maximum speed-up across all setups and GPUs was in excess of 200 times. The CUDA beamformer, thus, attained one to two orders of magnitude higher throughput and volume rates than the MATLAB beamformer.

V. Discussion

The MATLAB beamformer's performance was an order-of-magnitude lower than the CUDA beamformer. The MATLAB `gpuarray` type provides a simple entry point to GPU execution with implementations of individual operations. However, this does not consider the algorithm consituted by sequences

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TUFFC.2021.3071810, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control

of operations, and intermediate results are written to and read back from memory. Delay-and-sum beamforming mainly consists of basic arithmetic operations, and the ratio of non-memory to memory instructions is very low. Since the latency of memory accesses is very high compared to that of simple arithmetic operations, the memory latency dominates the execution time. This is also known as the “memory wall” [38].

GPUs address this problem by sharing the arithmetic units between groups of threads. Two grouping levels exist: ”blocks” are specified by the programmer, and ”warps” are subdivisions of blocks to better map varying block sizes to the SM microarchitecture. The idea is that, while one or more warps are waiting for memory, other warps may use the computational units. If sufficiently many warps are allocated to the same SM, the memory access latency experienced by one warp is masked by other warps using the arithmetic units during this time. The number of warps allocated to one SM is affected by – among others – the number and sizes of thread blocks and the number of registers (input and intermediate results) used by each thread block.

The CUDA beamformer is limited by the number of registers per thread, which is high in part due to the third order interpolation. This requires four samples and coefficients that take up between 8 and 24 32-bit registers depending on precision and complexity. On top of this, element and image coordinates, pointers to input and output memory, and various bookkeeping counters must also be stored. The end result is that each thread uses many registers, which limits the number of warps that may be assigned to each SM, which again limits the possibility of latency masking. The main bottleneck of the CUDA beamformer is thus the latency of individual memory accesses.

Several possibilities for performance optimization may be explored. If each thread computes more than one consecutive output values, the input samples may be reused depending on in- and output sampling frequencies. Another option could be to design thread blocks to maximize overlap in the input sample range and thereby minimize the input size and thereby minimize the number of memory reads and maximize spatial cache locality. This may be further combined with the use of local shared memory as a fast-access software managed cache to minimize the number of registers used to store samples without introducing the need to request each sample individually from the memory system as was also done by Yiu *et al.* [14]. It is also possible to exploit the GPU’s specialized texture memory that has special caching structures and incorporates linear interpolation [14], [39]. This is better suited for demodulated input data, however, since linear interpolation of signals sampled close to the Nyquist limit increases side-lobe levels reducing contrast in the image. [32]

Best volume rates were attained for single precision, complex valued data, which corresponds to the lowest number of output samples with the lowest number of bits per sample. A prerequisite for this is the use of either quadrature sampling [40], which requires an additional phase rotation in the beamforming calculations for broadband signals, or analytic signals, which requires the Hilbert transform to be calculated for the input signals. This has previously been demonstrated in GPUs by Yiu et al [14].

VI. Conclusion

Two delay-and-sum beamformers for row-column addressed arrays were evaluated on three GPUs. The CUDA beamformer attained an order-of-magnitude higher performance than the MATLAB beamformer. The beamformers appeared to be primarily limited by memory access latencies. The median sample throughputs were 4.9, 20.8, and 33.5 gigasamples per second for a 1050 Ti, 1080 Ti, and TITAN V GPU, respectively. Out of 112 setups, real-time, volumetric imaging was attained in 13, 27, and 43 setups for the three GPUs, showing that even the low-end GPU (1050 Ti) can provide real-time 3D imaging. The use of single precision, analytic input signals yielded the highest frame rates albeit the cost of calculating analytic signals was not included in this investigation.

References

[1] S. W. Smith, H. G. Pavy, and O. T. von Ramm, “High speed ultrasound volumetric imaging system – Part I: Transducer design and beam steering,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 38, pp. 100–108, 1991.

[2] O. T. von Ramm, S. W. Smith, and H. G. Pavy, “High speed ultrasound volumetric imaging system – Part II: Parallel processing and image display,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 38, pp. 109–115, 1991.

[3] D. H. Turnbull and F. S. Foster, “Beam steering with pulsed two-dimensional transducer arrays,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 38, no. 4, pp. 320–333, July 1991.

[4] E. Roux, F. Varray, L. Petrusca, C. Cachard, P. Tortoli, and H. Liebgott, “Experimental 3-d ultrasound imaging with 2-d sparse arrays using focused and diverging waves,” *Scientific Reports*, vol. 8, no. 9108, 2018.

[5] S. Blaak, Z. Yu, G. C. M. Meijer, C. Prins, C. T. Lancee, J. G. Bosch, and N. de Jong, “Design of a micro-beamformer for a 2D piezoelectric ultrasound transducer,” in *Proc. IEEE Ultrason. Symp.*, sep 2009, pp. 1338–1341.

[6] E. Kang, Q. Ding, M. Shabanimotlagh, P. Kruizinga, Z. Y. Chang, E. Noothout, H. J. Vos, J. G. Bosch, M. D. Verweij, N. D. Jong, and M. A. Pertijs, “A reconfigurable ultrasound transceiver ASIC with 24x40 elements for 3D carotid artery imaging,” *IEEE J. Solid-State Circuits*, vol. 53, no. 7, pp. 2065–2075, 2018.

[7] M. C. Hemmsen, M. F. Rasmussen, M. B. Stuart, and J. A. Jensen, “Simulation study of real time 3D synthetic aperture sequential beamforming for ultrasound imaging,” in *Proc. SPIE Med. Imag.*, vol. 9040, 2014, pp. 90 401K–1–90 401K–9.

[8] C. E. Morton and G. R. Lockwood, “Theoretical assessment of a crossed electrode 2-D array for 3-D imaging,” in *Proc. IEEE Ultrason. Symp.*, 2003, pp. 968–971.

[9] C. H. Seo and J. T. Yen, “A 256 x 256 2-D array transducer with row-column addressing for 3-D rectilinear imaging,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 56, no. 4, pp. 837–847, April 2009.

[10] M. F. Rasmussen, T. L. Christiansen, E. V. Thomsen, and J. A. Jensen, “3-D imaging using row–column-addressed arrays with integrated apodization — Part I: Apodization design and line element beamforming,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 62, no. 5, pp. 947–958, 2015.

[11] J. A. Jensen, H. Holten-Lund, R. T. Nilsson, M. Hansen, U. D. Larsen, R. P. Domsten, B. G. Tomov, M. B. Stuart, S. I. Nikolov, M. J. Pihl, Y. Du, J. H. Rasmussen, and M. F. Rasmussen, “SARUS: A synthetic aperture real-time ultrasound system,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 60, no. 9, pp. 1838–1852, 2013.

[12] E. Boni, L. Bassi, A. Dallai, F. Guidi, V. Meacci, A. Ramalli, S. Ricci, and P. Tortoli, “ULA-OP 256: A 256-channel open scanner for development and real-time implementation of new ultrasound methods,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 63, no. 10, pp. 1488–1495, 2016.

[13] Y. F. Li and P. C. Li, “Software beamforming: Comparison between a phased array and synthetic transmit aperture,” *Ultrason. Imaging*, vol. 33, no. 2, pp. 109–118, 2011.

0885-3010 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TUFFC.2021.3071810, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control

[14] B. Y. S. Yiu, I. K. H. Tsang, and A. C. H. Yu, “GPU-based beamformer: Fast realization of plane wave compounding and synthetic aperture imaging,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 58, no. 7, pp. 1698–1705, 2011.

[15] H. K. H. So, J. Chen, B. Y. S. Yiu, and A. C. H. Yu, “Medical ultrasound imaging: to GPU or not to GPU?” *IEEE Micro*, vol. 31, no. 5, pp. 54–65, 2011.

[16] T. Di Ianni, C. Villagomez-Hoyos, C. Ewertsen, T. Kjeldsen, J. Mosegaard, and J. A. Jensen, “A vector flow imaging method for portable ultrasound using synthetic aperture sequential beamforming,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 64, no. 11, pp. 1655–1665, 2017.

[17] C. L. Palmer and O. M. H. Rindal, “Wireless, real-time plane-wave coherent compounding on an iPhone: A feasibility study,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 66, no. 7, pp. 1222–1231, 2019.

[18] J. W. Choe, A. Nikoozadeh, O. Oralkan, and B. T. Khuri-Yakub, “GPU-based real-time volumetric ultrasound image reconstruction for a ring array,” *IEEE Trans. Med. Imag.*, vol. 32, no. 7, pp. 1258–1264, 2013.

[19] R. Göbl, N. Navab, and C. Hennersperger, “SUPRA: open-source software-defined ultrasound processing for real-time applications: A 2D and 3D pipeline from beamforming to B-mode,” *International Journal of Computer Assisted Radiology and Surgery*, vol. 13, no. 6, pp. 759–767, 2018.

[20] M. B. Stuart, M. Schou, and J. A. Jensen, “Row-column beamforming with dynamic apodizations on a GPU,” in *Proc. SPIE Med. Imag.*, 2019, pp. 1–7, paper number 10955-20.

[21] M. B. Stuart, P. M. Jensen, J. T. R. Olsen, A. B. Kristensen, M. Schou, B. Dammann, H. H. B. Sørensen, and J. A. Jensen, “Fast GPU-beamforming of row-column addressed probe data,” in *Proc. IEEE Ultrason. Symp.*, 2019, pp. 1–4.

[22] C. E. M. Démoré, A. W. Joyce, K. Wall, and G. R. Lockwood, “Real-time volume imaging using a crossed electrode array,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 56, no. 6, pp. 1252–1261, 2009.

[23] A. Sampaleanu, P. Zhang, A. Kshirsagar, W. Moussa, and R. Zemp, “Top-orthogonal-to-bottom-electrode (TOBE) CMUT arrays for 3-D ultrasound imaging.” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 61, no. 2, pp. 266–276, 2014.

[24] J. T. Yen, “Beamforming of sound from two-dimensional arrays using spatial matched filters,” *J. Acoust. Soc. Am.*, vol. 134, no. 5, pp. 3697–3704, 2013.

[25] N. M. Daher and J. T. Yen, “2-D array for 3-D ultrasound imaging using synthetic aperture techniques,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 53, no. 5, pp. 912–924, 2006.

[26] T. L. Christiansen, M. F. Rasmussen, J. P. Bagge, L. N. Moesner, J. A. Jensen, and E. V. Thomsen, “3-D imaging using row–column-addressed arrays with integrated apodization — part II: Transducer fabrication and experimental results,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 62, no. 5, pp. 959–971, 2015.

[27] J. J. Flaherty, K. R. Erikson, and V. M. Lund, “Synthetic aperture ultrasound imaging systems,” United States Patent, US 3,548,642, 1967, united States Patent, US 3,548,642, 1967, Published 22 Dec 1970.

[28] J. A. Jensen, S. Nikolov, K. L. Gammelmark, and M. H. Pedersen, “Synthetic aperture ultrasound imaging,” *Ultrasonics*, vol. 44, pp. e5–e15, 2006.

[29] M. Karaman, P. C. Li, and M. O’Donnell, “Synthetic aperture imaging for small scale systems,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 42, pp. 429–442, 1995.

[30] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA,” *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[31] J. A. Jensen and N. Oddershede, “Estimation of velocity vectors in synthetic aperture ultrasound imaging,” *IEEE Trans. Med. Imag.*, vol. 25, pp. 1637–1644, 2006.

[32] H. Andresen, S. I. Nikolov, and J. A. Jensen, “Precise time-of-flight calculation for 3D synthetic aperture focusing,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 56, no. 9, pp. 1880–1887, 2009.

[33] E. Waring, “Problems concerning interpolations. by Edward Waring, M. D. F. R. S. and of the Institute of Bononia, Lucasian Professor of mathematics in the University of Cambridge,” *Philosophical Transactions of the Royal Society of London*, vol. 69, pp. 59–67, 1779. [Online]. Available: http://www.jstor.org/stable/106408

[34] J. A. Jensen and N. B. Svendsen, “Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers,” *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 39, no. 2, pp. 262–267, 1992.

[35] J. A. Jensen, “Field: A program for simulating ultrasound systems,” *Med. Biol. Eng. Comp.*, vol. 10th Nordic-Baltic Conference on Biomedical Imaging, Vol. 4, Supplement 1, Part 1, pp. 351–353, 1996.

[36] NVIDIA, “Nvidia Tesla V100 GPU architecture,” Whitepaper, 2017, wP-08608-001_v1.1.

[37] ——, “NVIDIA Tesla P100,” Whitepaper, 2017, wP-08019-001_v01.2.

[38] W. A. Wulf and S. A. McKee, “Hitting the memory wall: Implications of the obvious,” *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, mar 1995.

[39] D. Hyun, Y. Li, I. Steinberg, M. Jakovljevic, T. Klap, and J. J. Dahl, “An open source gpu-based beamformer for real-time ultrasound imaging and applications,” in *Proc. IEEE Ultrason. Symp.*, vol. 2019-October, 2019, pp. 20–23.

[40] J. E. Powers, D. J. Phillips, M. A. Brandestini, and R. A. Sigelmann, “Ultrasound phased array delay lines based on quadrature sampling techniques,” *IEEE Trans. Son. Ultrason.*, vol. SU-27, pp. 287–294, 1980.

0885-3010 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on May 03,2021 at 10:46:54 UTC from IEEE Xplore. Restrictions apply.