Real-Time Full-Volume Row-Column Imaging

Præsius, Sebastian Kazmarek; Jørgensen, Lasse Thurmann; Jensen, Jørgen Arendt

Published in:
I E E E Transactions on Ultrasonics, Ferroelectrics and Frequency Control

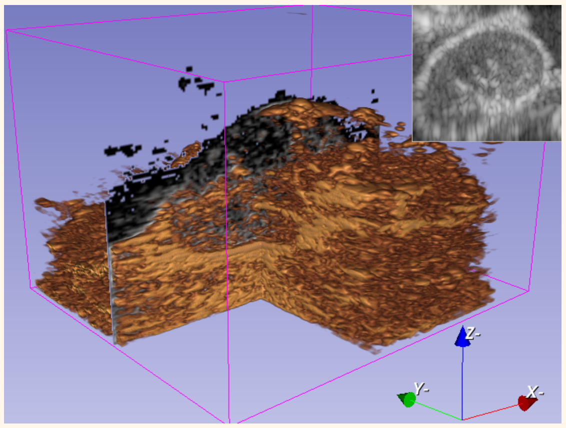
Publication date:
2025

Document Version
Peer reviewed version

Real-Time Full-Volume Row-Column Imaging

Sebastian Kazmarek Præsius, *Graduate Student Member, IEEE*,
Lasse Thurmann Jørgensen, *Member, IEEE*,
and Jørgen Arendt Jensen, *Fellow, IEEE*

***Abstract*— An implementation of volumetric beamforming for row-column addressed arrays (RCAs) is proposed, with optimizations for Graphics Processing Units (GPUs). It is hypothesized that entire volumes can imaged in real time by a consumer-class GPU at an emission rate $\geq$ 12 kHz. A separable beamforming algorithm was used to reduce the number of calculations with a negligible impact on the image quality. Here, a single image was beamformed for each emission and then extrapolated to reproduce the volume, which resulted in 65 times fewer calculations per volume. Reusing computations and samples among adjacent pixels and frames reduced the amount of overhead and load instructions, increasing performance. A GPU beamformer, written in CUDA C++, was modified to implement the dual-stage imaging with optimizations. In-vivo rat kidney data was acquired using a 6 MHz Vermon 128+128 RCA probe and a Verasonics Vantage 256 scanner. The acquisition used 96 defocused emissions at a 12 kHz rate for a volume acquisition rate of 125 Hz. Processing time, including all pre-processing, was measured for an NVIDIA GeForce RTX 4090 GPU, and the resulting beamforming rate was 1440 volumes per second, greatly exceeding the real-time rate. Based on the GPU's floating-point throughput, this corresponds to 22% of the theoretically achievable rate. High efficiency was also shown for an RTX 2080 Ti and RTX 3090, both achieving real-time imaging. This shows that 3D imaging can be performed in real time with a setup similar to 2D imaging: Using a single graphics card, one scanner, and 128 transmit/receive channels.**



***Index Terms*— Synthetic aperture imaging, row-column addressed probes, separable beamforming, array signal processing, parallel processing, volumetric imaging, beamforming**

I. Introduction

ULTRASOUND imaging is widely used due to its safety, affordability, and ability to image at the patient's bedside in real-time. Most examinations use two-dimensional imaging, meaning the sonographer must manually position the probe to find the proper views, which is time-consuming. Another issue is poor work posture; for example, the sonographer may rotate their wrist to rotate the probe, which will cause physical strain, and surveys indicated 80-90% of sonographers scan with pain, most often located in their shoulder, neck, and wrist [1], [2].

Furthermore, standard measurements use assumptions, such as that a kidney's volume can be measured from two (assumed) orthogonal planes using the ellipsoid volume formula [3], [4], or that the image plane is at a specific angle to blood flow [5]. While finding a proper scan plane may be difficult, finding the same plane again is even more challenging, making it hard to replicate measurements in follow-up examinations [3], [6].

Volumetric ultrasound enables entire volumes to be acquired and visualized, alleviating many of the aforementioned issues. Probe motions can then be emulated retrospectively by moving or rotating the reconstructed volume, and by extracting slices, any plane within the field of view may be displayed regardless of the probe's orientation [7]. Measurements can then be made with higher accuracy [8]–[12], velocities in all three directions can be estimated to better quantify blood flow [13]–[15], and the average scan time may be shortened, e.g., from 15 minutes to four minutes in one case [11], improving patient throughput.

To remain competitive with 2D imaging, comparable image quality is needed for all planes within the volume at high frame rates. This may achieved by using 2D arrays, as shown in 1991 [16]–[18]. However, the array's width and height determine the lateral and elevational resolution. Thus, to match an N-element 1D array in both transverse directions, this 2D array must have on the order of N^2 elements, assuming it has a matching pitch. While many scanners support $N = 128$ receiving RF channels, using $N^2 = 16384$ channels would not be practical as it would result in large cables and data rates. Hence, a channel reduction is often necessary in practice for 3D imaging with 2D arrays.

Row-column addressed 2D arrays (RCAs) were proposed as a solution in 2003 [19]. They reduce the number of channels from N^2 to $N + N$ by aggregating the signals along the rows and columns [20]–[22], enabling high-quality 3D imaging with a large 2D aperture [23], [24]. The RCA must be $\approx 36\%$ wider and taller than a corresponding matrix array to have a matching resolution, as measured by the full width at half max (FWHM)

Highlights

- **A novel multi-pixel beamforming optimization improved performance by reducing the number of load instructions.**
- **Real-time beamforming was achieved at a rate of 1440 volumes per second.**
- **The processing time is no longer a limitation for 3D imaging with the 6 MHz Vermon 128+128 row-column array.**

[25], and high image contrast can be attained with apodization on the row and column edges [26]. To attain high volume rates, synthetic aperture or plane wave sequences are often used [26], [27], e.g., for 3D flow estimation [28] or super-resolution [29], [30]. Other methods have been proposed to reduce the channel count, such as sparse 2D arrays [31], [32], micro-beamforming [33], [34], mechanically scanning 1D arrays [35], multiplexing [36], and synthetic aperture sequential beamforming [37]–[39]. These technologies provide trade-offs in imaging rate, contrast, field-of-view, resolution, and hardware complexity. For RCAs, higher image quality in the elevation requires more emissions.

After RF data is acquired, it must be beamformed. However, volumetric beamforming is challenging since each 3D volume essentially contains hundreds of 2D images to be beamformed. Separable beamforming algorithms sacrifice the image quality slightly to reduce this computational complexity considerably. Introduced for frequency-domain beamforming in 1997 [40], and for standard time-domain beamforming in 2011 [41], [42], these algorithms use a two-stage process to focus the acquired RF data in each direction (e.g., x, then y) [38], [43], and they handle the 2D array as a set of 1D arrays. The volume acquired by such a 1D array (with no elevation-focusing acoustic lens) will appear smeared in the elevation direction; then, separable beamformers approximate this volume by computing only one image, which they extrapolate (i.e., smear) in the out-of-plane direction to get the approximated volume. While beamforming a volume containing M planes costs M times more operations than a 2D beamforming, separable beamforming requires only $1+M/N$ (usually less than 2.5) times more operations, where N is the 1D array channel count. Thus, separable beamforming brings the number of computations for 3D imaging to the same order of magnitude as 2D imaging with this 2D array.

In the dual-stage RCA beamformer from 2023 [44], a single plane is beamformed for each emission, and the planes are then extrapolated into volumes and summed to get the final volume. This separable RCA imaging showed a negligible loss in image quality, where the FWHM was increased by 1.19%. However, the number of delay-and-sum operations was $NM/(N+M) =$ 56 times lower than with the conventional method from [26]. (Number of channels $N = 128$, number of planes $M = 100$.)

A high emission rate is used to reduce RCA motion artifacts [45], but this increases the computational throughput required for real-time imaging. High-throughput hardware is therefore needed, and specialized hardware such as field-programmable gate arrays (FPGAs) [46]–[48], digital signal processors [49], and application-specific integrated circuits (ASICs) [50]–[54] have historically been used for beamforming. An ASIC design was proposed in 2020 [54], capable of real-time beamforming for a 32×32 receiver array at a high emission rate of 13 kHz.

Each imaging point (voxel) can be computed independently while beamforming, making it an *embarrassingly parallel* task. Modern Graphics Processing Units (GPUs) are specialized for parallel computations but are also general-purpose processors, like CPUs. They support the data transfer rates commonly used in ultrasound (1-6 GB/s) [55], and their software is written in, e.g., C++ with Compute Unified Device Architecture (CUDA) from NVIDIA to enable efficient use of their GPU hardware. Several open-source GPU-based beamformers exist [56], [57].

Real-time GPU beamforming was presented in 2010 [58], and it has since been demonstrated for the 3D case [59], [60]. However, the number of computations to image the full volume often limits how many emissions can be processed in real-time. Real-time RCA imaging was achieved using GPU in 2021 [61] with 120 emissions beamformed per second in the full-volume case; this beamformer was implemented with CUDA C++, and it was used as the first stage of the dual-stage beamformer [44], with the second stage implemented in MATLAB, to achieve a beamforming of 1805 emissions per second for entire volumes. For this paper, a CUDA implementation of the second beamforming stage was created to further improve the performance.

Finding the real-time beamforming rate should be easy since each GPU model can do a certain number of calculations every second. However, a far lower rate is often achieved in practice. GPUs are not specialized only for beamforming, making them less efficient compared to, e.g., ASICs, as GPUs must use load instructions and address calculations to fetch RF channel data from memory [54]. Our novel contribution is the optimization *multi-pixel beamforming*, where loaded RF samples are reused for neighboring voxels to reduce the load instructions required. Along with other optimizations, it enabled beamforming to be performed more efficiently with GPUs.

The hypothesis is that a GPU-optimized implementation of the dual-stage beamformer will allow for real-time full-volume imaging using a consumer GPU at a pulse-repetition frequency of 12 kHz. This rate is sufficiently fast for flow estimation with RCAs [62], allowing blood flow dynamics to be captured in 3D and displayed at the bedside for diagnostics.

The paper is organized as follows: The beamforming theory is first outlined. Next, the optimizations used to speed up processing are described. Finally, the implementation is evaluated on in-vivo rat kidney data acquired at 125 volumes per second with 96 emissions per volume using a 12 kHz emission rate.

II. Theory

This section outlines the theory behind the dual-stage RCA beamforming algorithm from [44], which reduces the number of calculations used for volumetric beamforming with RCAs. The number of calculations per voxel is still relatively high, so a minimum number of voxels per cubic meter is derived in the final parts of this theory section. To summarize the algorithm: One xz-plane per emission is computed using the conventional synthetic aperture delay-and-sum beamformer from [26], after which it is extrapolated with a second beamforming stage from [44] to get a low-resolution volume. The final high-resolution volume is then computed as a sum of low-resolution volumes.

The delays (time-of-flight), extrapolations and apodizations were well-described in the works [26], [44], [63], respectively. Synthetic aperture imaging in 2D is first outlined since initially xz-planes are beamformed.

A. Synthetic Aperture Imaging

Synthetic aperture imaging [64]–[67] uses multiple converging or diverging emissions to produce a high-resolution image. Each emission produces a low-resolution image (LRI), calculated as a weighted delay-and-sum of received pulse-echo data

$$\mathrm{LRI}(x,\, z) = \sum_{i=1}^{N_{rx}} \alpha_i(x,\, z) \cdot r_i\big(\mathrm{ToF}_i(x,\, z)\big)\,, \tag{1}$$

where $r_i(t)$ is the data from receiver $i = 1, 2, ..., N_{rx}$, and the time-of-flight, ToF_i, is used to look up the sample from the RF channel data corresponding to an echo from the location $(x,\, z)$ for the specific receiver in this emission. The apodization α_i is applied to reduce sidelobes, and it is often varied dynamically. A simple case of dynamic apodization is [63]

$$\alpha_i(x,\, z) = A\left(\mathrm{F\#} \cdot \frac{x_i^{rx} - x}{|z|}\right), \tag{2}$$

where x_i^{rx} is the lateral position of receiver i, and A is some window function, which is zero outside the interval $[-0.5, 0.5]$. (E.g., a von Hann window is $A(u) = \cos^2(\pi u)$ for $|u| \leq 0.5$.) This scheme grows the effective aperture with the depth, z, to maintain an approximately constant F-number of F#. The F-number is the imaging depth divided by aperture width, and it determines the lateral image resolution [68]. Thus, a spatially uniform resolution is approximated with this scheme as long as there are receivers to support the growing apodization window. The final high-resolution image (HRI) is computed as

$$\mathrm{HRI}(x,\, z) = \sum_{j=1}^{N_{tx}} \alpha_j^{tx}(x,\, z) \cdot \mathrm{LRI}_j(x,\, z), \tag{3}$$

where LRI_j is the low-resolution image from emission j, and α_j^{tx} is some transmit apodization. Thus, the 2D imaging entails summing $N_{rx}N_{tx}$ RF channel data values per pixel.

B. Dual-Stage Row-Column Array Imaging

The dual-stage algorithm [44] was used to lower the number of operations per voxel from $N_{rx}N_{tx}$ to less than $N_{rx}+N_{tx}$, where N_{rx} and N_{tx} are the number of receivers and emissions. RCAs usually receive with the columns after emitting with the rows (or vice versa) [26], such that N_{rx} is half of the elements; and, to avoid confusion, the lateral direction x will be defined as the direction where those receiving elements are distributed.

Each emission generates one low-resolution volume (LRV), which is approximately constant in a trajectory along yz [44].

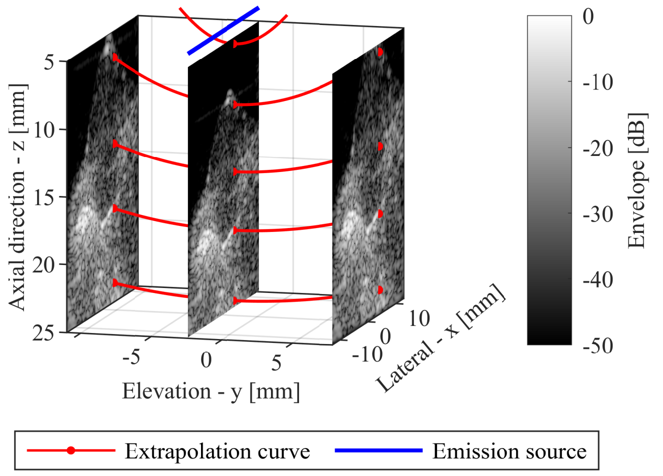


Fig. 1. The volume acquired by a row-column array can be computed by beamforming one plane (per emission) and extrapolating it [44]. Here, the primary $\boldsymbol{xz}$-plane at $\boldsymbol{y_0 = -2.4}$ mm was extrapolated to produce two other planes. The emission is shown at $\boldsymbol{(x, y, z) = (x, y_0, 4)}$, and the red curves give examples of where the volume is constant-valued.

A single slice $\mathrm{LRI}(x,\, z) = \mathrm{LRV}(x,\, y_0,\, z)$, at some elevation, y_0, may therefore be beamformed using the 3D delay-and-sum beamformer from [26] to represent the entire LRV compactly as an extrapolated plane (as illustrated in Fig. 1). This assumes

$$\mathrm{LRV}(x, y, z) = \mathrm{LRI}(x,\, f(y, z)), \tag{4}$$

where $f(y, z)$ is used to look up the LRI-pixel corresponding to the LRV at the voxel position $(x,\, y,\, z)$. The resulting dual-stage beamformer computes the high-resolution volume (HRV) by extrapolating and summing such low-resolution images [44]

$$\mathrm{HRV}(x,\, y,\, z) = \sum_{j=1}^{N_{tx}} \alpha_j^{tx}(y,\, z) \cdot \mathrm{LRI}_j(x,\, f_j(y, z)), \tag{5}$$

where LRI_j is the image from emission $j = 1,\, 2,\, ...,\, N_{tx}$, and α_j^{tx} is a transmit apodization. For any fixed x, (5) is equivalent to (1), where now yz-planes are produced from $\mathrm{LRI}(f(y, z))$, (x omitted) instead of producing xz-planes from $r(\mathrm{ToF}(x, z))$. Thus, one might implement the second beamforming stage (5) as a 2D beamforming for each x, which computes an yz-plane.

Dual-stage RCA imaging of a volume with M_y voxels along the elevation direction requires N_{tx} evaluations of apodization and LRI-extrapolations per voxel. But since the LRIs must also be computed, $N_{tx}+(N_{rx}N_{tx}/M_y)$ operations are used in total, where N_{rx} and N_{tx} are the number of receivers and emissions. Typically, $M_y > N_{tx}$, and 3D imaging then requires less than $N_{rx} + N_{tx}$ summations per voxel, as is the case in this paper. The ToF and f used in this work can be found in Appendix A, along with pseudo-code for the beamforming in Appendix C.

C. Sampling and Interpolation

The RF data, r, is sampled discretely in a digital system, and interpolation is therefore used to find values between samples. Perfect reconstruction is possible using sinc interpolation if the sampling rate is greater than twice the highest frequency in the signal [69] (denoted as its Nyquist rate). If the data is sampled

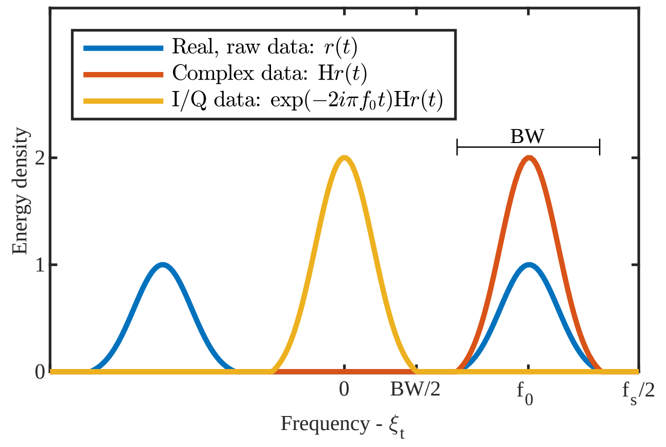


Fig. 2. The spectra of different input data types. The sampling rate, f_s, should be greater than the Nyquist rate (twice the highest frequency). Quadrature sampling [74] reduces the sufficient sampling to $f_s^{\mathrm{IQ}} > \mathrm{BW}$, which is half or less than the Nyquist rate for real-valued data: $2f_0+\mathrm{BW}$, where f_0 is the signal's center frequency, BW is its bandwidth, and $\mathrm{H}r$ is the complex-valued envelope of r, a.k.a. its analytic representation, defined as $\mathrm{H}r(t) = r(t) + is(t)$, where s is the Hilbert transform of r.

at an even higher rate, then inexpensive methods such as linear interpolation can be used with a limited loss in accuracy [70]. Cubic Lagrange interpolation [71] is used in this paper because it needs less oversampling to obtain a sufficient image quality than linear interpolation [72]. This reduces memory footprints, which will be shown to be beneficial for optimizations. Hence, $r(t)$ is calculated as a weighted sum of the four samples closest in time to t.

An image or volume should be beamformed according to its spatial Nyquist rate since interpolation can increase the density of pixels/voxels using far fewer calculations than beamforming additional in-between points. Assuming the RF data is sampled at its Nyquist rate, then a sufficient lateral density for each term $r(\mathrm{ToF}(x,z))$ in the beamforming sum (1) is given by

$$\max_{x,z}\left|\frac{\partial\,\mathrm{ToF}(x,z)}{\partial x}\right| < \frac{\Delta t}{\Delta x}, \qquad (6)$$

where $\Delta t = 1/f_s$ is the time between RF channel data samples and Δx is the lateral spacing between pixels. Approximations were used for (6) – the proof and limitations are found in [73]. For row-column array beamforming with dynamic apodization, the sufficient voxel density from (6) is (derived in Appendix B)

$$\Delta x, \Delta y < \frac{c}{f_s}\sqrt{(2\,\mathrm{F\#})^2+1} \quad \text{and} \quad \Delta z < \frac{c}{2f_s}, \qquad (7)$$

where F# is the F-number in the dynamic apodization, c is the speed of sound, and $f_s = 1/\Delta t$ is the RF data sampling rate.

D. Quadrature Sampling

Quadrature sampling was introduced for ultrasound in 1979 [75], and it reduces the RF Nyquist rate by a factor of (at least) two without loss of information. However, the samples become complex-valued with the real and imaginary parts denoted as the in-phase (I) and quadrature (Q) components. The idea from [75] is summarized here with Fig. 2, which shows how I/Q data may be sampled according to the original RF data's bandwidth. One benefit comes from the volume beamformed with I/Q data inheriting this property because the axial direction corresponds to time under an approximation, meaning the number of voxels can be (at least) halved. For the proof and limitations, see [74]. Thus, quadrature sampling is used to lower the sampling rates for both inputs (RF channel data) and outputs in beamforming, which can reduce memory usage and speed up the processing.

III. Optimizations

This section covers the optimizations applied to the 3D RCA beamformer from [61] to improve its processing speed. They are described for 2D imaging for simplicity, but also because both beamforming stages are essentially 2D (see Section II-B), where the last stage computes every yz-plane in the volume.

Beamforming with cubic interpolation involves the weighted summation of four RF samples for each pixel (x, z), receiver, and emission. Every delay-and-sum term is the 3-step process

1) Calculate the time-of-flight (ToF), interpolation weights, and apodization. A square root must be evaluated for the time of flight and a division for the dynamic apodization. A float-to-integer conversion (i.e., rounding) is also used to convert the time-of-flight to the address to load from.
2) Read the four samples nearest $t = \mathrm{ToF}(x, z)$ using four load operations, as these are needed for the interpolation.
3) Calculate the delay-and-sum (see (1)) as the dot product between the four interpolation weights and four samples, multiplied by the apodization. This requires at least four multiplications and additions.

A model of the number of operations used in the beamforming, per delay-and-sum term, is therefore given by

$$\mathrm{Work} = 4\cdot\mathrm{Load} + 4\cdot\mathrm{Multiply} + 4\cdot\mathrm{Add} + \mathrm{Overhead}, \qquad (8)$$

where ”Overhead” covers all operations that do not depend on the RF data, including but not limited to all operations from the first step in the 3-step process above. Then, $4\cdot\mathrm{Load}$ refers to four load instructions for the second step that fetch RF channel data from the GPU's RAM so it may be used in the third step, where four fused multiply-add instructions carry out the sum.

The model (8) shows the quantity of each type of operation in the initial case to allow comparison with optimizations later. However, performance must be measured to see if there is an improvement in practice, as the time for beamforming is given by the number of operations only when instructions are issued every clock cycle in every multiprocessor (parallel GPU core). Issuing an instruction takes one clock cycle, meaning (8) costs at least eight cycles. Instructions take multiple cycles to finish executing, but GPUs have latency-hiding mechanisms to avoid wasting time waiting for instructions to finish. For example, due to instruction-level parallelism [76], other instructions may be issued after a load instruction and run concurrently with it. Since instructions can run in parallel, their latency is irrelevant to the computation time if no cycles are spent waiting on them, so, e.g., computing a square root is not necessarily very costly; however, each load instruction may run for hundreds of cycles [77], often making them expensive as cycles are spent waiting. Initially, though, most of the time went to overhead operations.

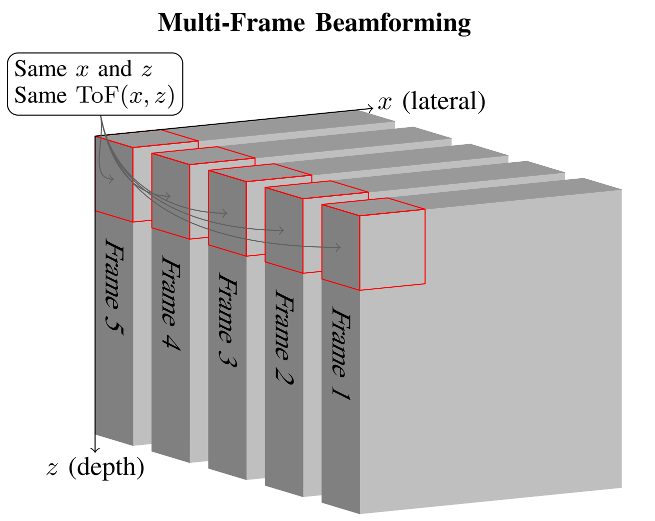


Fig. 3. Illustration of multi-frame beamforming using $B = 5$ frames, where a thread computes the same pixel (red highlights) for five frames. The time-of-flight and apodization values can then be reused five times.

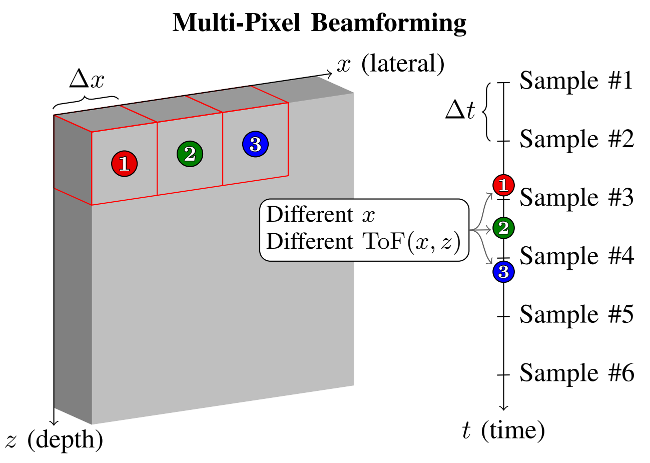


Fig. 4. Illustration of multi-pixel beamforming of laterally adjacent pixels. By computing three pixels (red highlights), the thread only needs to load samples #1-6 once to evaluate the term $r(\mathrm{ToF}(x, z))$ for three pixels. The time-of-flight $t = \mathrm{ToF}(x, z)$ of each pixel is indicated by a circle on the right, and since evaluating $r(t)$ requires the four samples nearest t, one can see that, for example, the second pixel must use samples #2-5.

In the beamformer, without the optimizations of this section, every thread (parallel worker) computed one pixel. By instead making every thread compute three pixels for multiple frames, the amount of overhead and load operations was reduced since redundant computations could be reused. Quadrature sampling then minimized the number of delay-and-sum terms, and zero-weight terms could be skipped using an early exit optimization. Threads get assigned to multiprocessors in blocks. Therefore, letting each block compute a small rectangle of pixels enabled load operations to be accelerated by the multiprocessor's memory cache, as adjacent pixels often required the same samples. Finally, declaring some variables that did not vary as constants in the code allowed for additional optimizations. The following sections describe each of these optimizations in greater detail.

A. Multi-Frame Beamforming

All overhead was reduced by making every thread compute the same pixel for multiple frames in one pass. This is because the time-of-flight and apodization do not vary between frames, and such frame-invariant computations can be reused to reduce overhead by a factor of B if a batch of B frames is processed.

The case of $B = 5$ is illustrated in Fig. 3, and a pseudo-code example is given in Appendix C, Algorithm 3. The operations that are used per frame, pixel, receiver, and emission become

$$\mathrm{Work}_{\mathrm{MF}} = 4 \cdot \mathrm{Load} + 4 \cdot \mathrm{Multiply} + 4 \cdot \mathrm{Add} + \frac{\mathrm{Overhead}}{B}, \quad (9)$$

where RF channel data for B frames must be acquired before beamforming. This method of batching was introduced in [78].

B. Quadrature Sampling

Quadrature sampling halves the number of points that must be beamformed (see Section III-B). This was used to halve the amount of overhead and load operations for the beamforming.

The I/Q samples must be returned to the original frequency, f_0 (see Fig. 2), to get the correct beamforming sum in (1) [74], which was realized by multiplying with $\exp(2i\pi f_0 \mathrm{ToF}(x, z))$ after having interpolated the low-frequency complex RF data. This interpolation used eight multiplications and additions, and the exponential was evaluated as a sine and cosine, increasing overhead slightly. Their complex product was computed using four multiplications and two additions, and the operations per pixel part (I or Q), pixel, receiver, and emission became

$$\mathrm{Work}_{\mathrm{QS}} = 2 \cdot \mathrm{Load} + 6 \cdot \mathrm{Multiply} + 5 \cdot \mathrm{Add} + \frac{\mathrm{Overhead}}{2}. \quad (10)$$

This equation can be directly compared with (8) as each pixel now contains two parts, but the number of pixels is also halved.

The overhead is halved since there are half as many pixels, and the number of load operations is halved because a complex number can be loaded with a single vectorized load instruction. (A pair of 32-bit numbers can be loaded as one 64-bit number.) The increased number of multiplications and additions was not an issue in practice since most of the time was spent on loading samples after multi-frame beamforming reduced the overhead.

C. Multi-Pixel Beamforming

The number of load operations was halved by making every thread compute three laterally adjacent pixels. This allowed RF channel data to be reused because three of the samples loaded to evaluate the term $r(\mathrm{ToF}(x, z))$ in the beamforming (1) were the same samples needed to beamform the two adjacent pixels.

The cubic interpolation of $r(t)$ uses the four samples nearest t. Consequently, a neighboring pixel requires at least three of the same samples if its time-of-flight $\mathrm{ToF}(x + \Delta x, z)$ satisfies

$$\left|\mathrm{ToF}(x + \Delta x, z) - \mathrm{ToF}(x, z)\right| < \Delta t, \quad (11)$$

where $\Delta t = 1/f_s$ is the time between consecutive RF samples, and Δx is the lateral pixel spacing. If N adjacent pixels satisfy (11), then $N+3$ samples are sufficient for their interpolations. Three pixels were used in this instance, as illustrated in Fig. 4, reducing the operations per pixel, receiver, and emission to

$$\mathrm{Work}_{\mathrm{MP}} = 2 \cdot \mathrm{Load} + 4 \cdot \mathrm{Multiply} + 4 \cdot \mathrm{Add} + \mathrm{Overhead}. \quad (12)$$

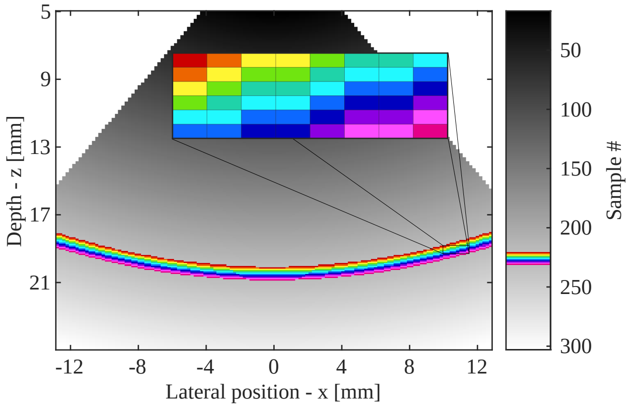


Fig. 5. The initial index of the four samples that were loaded to complete the evaluation of $\boldsymbol{r(\mathrm{ToF}(x,z))}$ for one term/receiver in (1) is shown. All pixels of a matching color loaded exactly the same four samples for their interpolation, and high-contrast colors are used in the zoomed-in patch to highlight this case. Note that only 14 unique RF samples are used by the pixels in the rainbow arc, meaning there is great overlap in accesses.

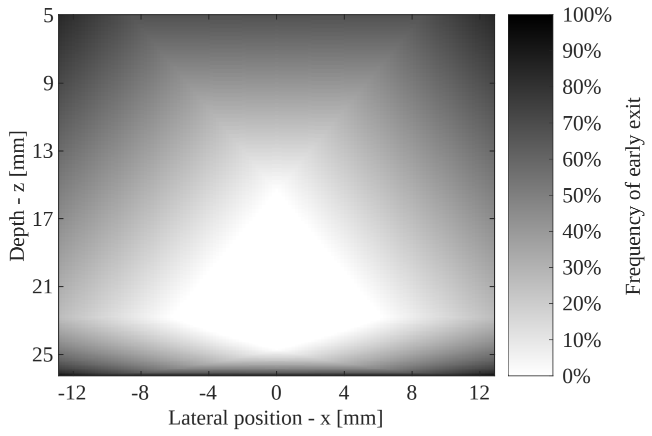


Fig. 6. The number of beamforming terms that are skipped due to early exit. Not all $\boldsymbol{N_{rx} = 128}$ terms have to be summed when beamforming the image (see (1)), as zero-weight terms may be skipped. This resulted in 32% fewer delay-and-sum operations per image.

Threads were made to first load the four RF samples for the evaluation/interpolation of $r(\mathrm{ToF}(x,z))$ for their second pixel. It was *assumed* three of these already-loaded samples could be reused for the other two pixels, and the delay-and-sum term for the first, second, and third pixel was computed (in this order), while only loading at most two additional samples if necessary. The pseudo-code for this is found in Appendix C, Algorithm 4.

An error would be raised if the assumption was not satisfied, meaning it could be verified by simply beamforming one frame and checking for errors. There will be no error in the common case, where the imaging is sampled densely, and this is because when (6) is satisfied, it implies that (11) holds for every pixel. To prove this claim, let $f(c) = \mathrm{ToF}(c,z)$ for any lateral c, then by the mean value theorem there exists some $c_0 \in [x, x+\Delta x]$ (open interval), where

$$f'(c_0) = \frac{f(x+\Delta x) - f(x)}{\Delta x}. \tag{13}$$

The exact place c_0 is unimportant because assuming (6) holds, $\lvert f'(c)\rvert < \Delta t/\Delta x$ for any c, and also for c_0. The result is that

$$\lvert f'(c_0)\rvert\,\Delta x = \lvert f(x+\Delta x, z) - f(x,z)\rvert < \Delta t, \tag{14}$$

which is equivalent to $\lvert \mathrm{ToF}(x+\Delta x, z) - \mathrm{ToF}(x,z)\rvert < \Delta t$. This shows that multi-pixel beamforming can be utilized when the image is oversampled to the same degree (or more) as the RF channel data is oversampled. Note that Δt is doubled when quadrature sampling is used, so is Δz, but Δx stays the same (assuming all three are maximized). This eases the criteria (11) and doubles the rate of overlapping sample accesses laterally.

Overlap in loaded samples is very common in beamforming. This is illustrated in Fig. 5, which shows the index (sample #) of the first RF channel data sample that was loaded to compute the cubic interpolation for some arbitrary delay-and-sum term. Figure 5 essentially shows $\mathrm{ToF}(x,z)/\Delta t$ after a rounding: All pixels that needed samples #1-4 were assigned the value 1, and pixels that needed samples #2-5 were assigned the value 2, and so on. All pixels whose apodization is zero are shown as blank because they are irrelevant (this is explained in Section III-D). One might interpret Fig. 5 as a zoomed-out version of Fig. 4.

Laterally adjacent pixels used the same or adjacent samples in Fig. 5, meaning multi-pixel beamforming may be used here. Pixels with the same time-of-flight need the same RF samples, as highlighted by the rainbow arc in Fig. 5, in which the pixels only need 14 different RF data samples. Essentially, this curve is the solution to $\mathrm{ToF}(x,z) = k$ for some constant k; however, since the time-of-flight is usually different for each term in (1), *the set of pixels that use the same samples vary for each delay-and-sum term*, making it difficult to reuse samples among more than a few pixels. When the time-of-flight is nearly constant: $\partial\,\mathrm{ToF}/\partial x \approx 0$ (as in the bottom of the rainbow arc in Fig. 5), the number of load operations in (12) is further reduced from $(N+3)/N = 2$ sample loads per pixel to just $4/N = 1.33$. Thus, using laterally adjacent pixels promotes this special case, which would be rare if axially adjacent pixels were used since typically $\partial\,\mathrm{ToF}/\partial z > 0$ in pulse-echo ultrasound – this is also evident from Fig. 5, where the sample # increases with depth. Only $N = 3$ pixels were used because the per-thread memory is limited, and each thread must store NB sums if multi-frame beamforming is also used with the batch size B. This means increasing N quickly has diminishing returns for performance. Note that multi-pixel beamforming does not change the output, meaning it does not affect image quality.

In summary, multi-pixel beamforming of three neighboring pixels reduces the load operations per delay-and-sum term (1) from four to at most two, and this can be used if (6) is satisfied.

D. Early Exit

Early exit was used to effectively skip delay-and-sum terms if their apodization was zero. It is zero when the argument to the window $A(u)$ in (2) does not satisfy $\lvert u\rvert \le 0.5$. In this case, there is no need to load any samples for the cubic interpolation.

Figure 6 shows the frequency of zero-weight terms for the sum in (1). This delay-and-sum was implemented using a loop, and early exit was performed by jumping to the next iteration (next RF channel) if the current term has zero weight, as shown in Appendix C, Algorithm 1, at Line 7. Terms with zero weight are overhead since the RF data samples do not influence them. This optimization was also used in [78].

E. Rectangular Thread Blocks

Load operations could be accelerated by effectively reusing samples in a larger area than through multi-pixel beamforming. Threads are grouped in blocks of typically 128 to 512 threads, and threads in the block share the same first-level (L1) memory cache. Thus, RF channel data can be loaded quicker if another thread in the block recently loaded it and it's still in the cache, for instance, in 32 cycles instead of 188 on one GPU [77].

The zoomed-in 8×6 patch in Fig. 5 has an access footprint of just $14/(8{\cdot}6) = 0.29$ RF samples per pixel, which illustrates how the number of samples used in a large area can be small. A block of threads can beamform such a rectangular area, and its size can be tuned to maximize the cache hit rate, reducing the time spent on load operations without changing the number of operations. This was also done in [79], and the thread block size is a common consideration for GPU programs.

For example, a 8×16 thread block would manage a 8×16 patch of pixels. But with multi-pixel beamforming, each thread covers three pixels, and the block then manages 24×16 pixels. If multi-frame beamforming is used with a batch size B, then the memory footprint is B times greater as data for B frames is accessed. This makes the block size crucial for performance at large batch sizes. One can, therefore, measure the performance of different block sizes using a large batch size to optimize it.

F. Compile-Time Constants

Turning some inputs into compile-time constants improved performance as the compiler could apply more optimizations when translating the source code into machine instructions.

The memory hardware should primarily be used to fetch RF channel data. Therefore, the receiver positions were computed from a parametrization instead of loading them from an array. Likewise, apodizations were evaluated directly instead of using linear interpolation. For Hann windowing, the direct evaluation of $A(u) = \cos(\pi u)^2$ used only four instructions instead of 16 (which included two load instructions for linear interpolation).

When applying multi-pixel or multi-frame beamforming, the batch size, B, was specified at compile time. The main reason is that the partial sums were stored in an array with B elements ($3B$ in the multi-pixel case). If such a thread-local array is not indexed using only compile-time constants (e.g., if B varies), it gets stored in RAM and accessed with load instructions [76].

Lastly, giving the total number of samples per frame allowed address calculations to be reused in multi-frame beamforming. Every load instruction for RF data uses an initial sample index (see Fig. 5), plus a relative offset of 0, 1, 2, or 3, to load each of the four samples for cubic interpolation. But for multi-frame beamforming, B sets of four samples were loaded, which used the offsets $c + bS$ for $c = 0 \ldots 3$ and $b = 0 \ldots B-1$ (relative to the initial sample in the first frame), where $S = N_s N_{rx} N_{tx}$ was the number of RF samples between frames. If this access stride S is not known at compile-time (or it is too large[1]), then the values bS must be computed at run-time, and (9) should be corrected to include $B - 1$ integer multiplications. However, with this optimization, the compiler just computes every offset $c + bS$ during compilation and stores it in the load instruction.

[1]Only 24-bit values were supported, allowing a stride up to 16-megabytes.

TABLE I
OVERVIEW OF DATA ACQUISITION PARAMETERS

Subject	Rat kidney (in-vivo)
Scanner	Verasonics Vantage 256
Transducer	Vermon 128+128 RCA
Center frequency, f_0	6 MHz ($\lambda = 257\,\mu$m)
Element pitch	200 μm
Transmit voltage	96 V
Sampling rate of RF channel data, f_s	31.25 MHz
Number of emissions per volume, N_{tx}	96
Number of receiving channels, N_{rx}	128
Number of RF channel data samples, N_s	894
Pulse-repetition frequency, f_{prf}	12 kHz
Volume acquisition rate, f_{prf}/N_{tx}	125 Hz
RF data rate, 16 bits $\cdot\, N_s N_{rx} f_{prf}$	2.75 GB/s

TABLE II
OVERVIEW OF BEAMFORMING PARAMETERS

Voxels per volume, $M_x \times M_y \times M_z$	$133 \times 115 \times 250$
Volume dimensions	$25.8 \times 25.8 \times 20$ mm^3
Volume start depth	5 mm
Matched filter size, N_f	23 coefficients
Apodization window	Hann in all cases
F-number for dynamic apodization	0.6 (40° acceptance angle)
Decimation factor for I/Q RF data	3 (two of three discarded)
Measured center frequency, $\hat{f}_0$	5.12 MHz ($\hat{\lambda} = 301\,\mu$m)
Measured bandwidth, BW	5.36 MHz at -36 dB

The different versions of the beamformer were pre-compiled and dispatched as needed. This demonstrates how flexibility is not necessarily lost with compile-time constants. Alternatively, any version can be just-in-time compiled at run-time and used.

IV. METHOD

A. Data Acquisition

In-vivo data from a Sprague Dawley rat kidney was used. The acquisition parameters are shown in Table I, and they are similar to those used in [15]. The synthetic aperture sequence contained 96 evenly-spaced virtual sources at an F-number of -1 ($z^{tx} = -6.4$ mm) to 32 Hann-apodized emitting elements. The role of rows and columns as transmitter and receiver was swapped with each frame, resulting in a volume rate of 62.5 Hz if both configurations are desired for the 3D imaging.

B. Beamforming Parameters

The beamforming parameters are shown in Table II. These were chosen to minimize the sampling rates while still obeying the Nyquist limits (see Fig. 7). To compare the data types, real data, and I/Q data, two setups were created:

1) Real-valued input and output throughout the processing, with twice as high axial voxel density as the other setup.
2) Real-valued input, converted I/Q samples before beamforming. Here, both beamforming stages use I/Q inputs.

Efficiency should increase if the image/volume is oversampled since more overlap in accesses leads to more L1/L2 cache hits. However, this would also result in a lower volume rate because

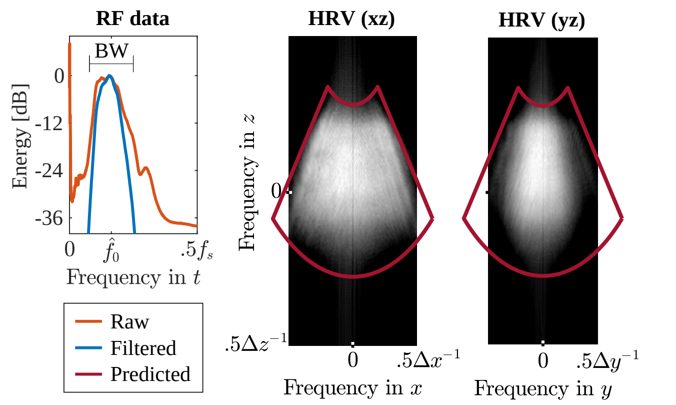


Fig. 7. The spectra of the input and outputs are shown in a 36 dB range. The axis limits of each spectrum correspond to the sampling rate used. The red fans encircling the volume's spectra show the sampling criteria (7) for I/Q outputs, with width $\frac{c}{\hat{f}_0+\mathrm{BW}/2}\sqrt{(2\,\mathrm{F\#})^2+1}$ and height $\frac{c}{2\,\mathrm{BW}}$.

more interpolations would be needed. Thus, the I/Q data setup was created with a minimized (but sufficient) spatial sampling, while the real-valued data setup was made to be comparable.

The spectrum of RF channel data is shown in Fig. 7, where the bandwidth, BW, was measured as the width of the band of frequencies above a -36 dB threshold after a matched filtering (while the center frequency was the center of this band). Fig. 7 shows the following spectra from the first acquired volume:

$$\mathcal{S}_{\mathrm{RF}}(\xi_t) = \sum_{j=1}^{N_{tx}} \sum_{i=1}^{N_{rx}} \left| \mathcal{F}_t r_{ij}(\xi_t) \right| , \tag{15}$$

$$\mathcal{S}_{\mathrm{HRV}}(\xi_x, \xi_z) = \int_{y\in\mathbb{R}} \left| \mathcal{F}_x \mathcal{F}_z \mathrm{HRV}(\xi_x, y, \xi_z) \right| , \tag{16}$$

$$\mathcal{S}_{\mathrm{HRV}}(\xi_y, \xi_z) = \int_{x\in\mathbb{R}} \left| \mathcal{F}_y \mathcal{F}_z \mathrm{HRV}(x, \xi_y, \xi_z) \right| , \tag{17}$$

where $\mathcal{F}_t$ is the fast Fourier transform in t, r_{ij} is the RF data of receiver i for emission j, and HRV is the beamformed volume. For Fig. 7, the rectangular window apodization $A(u) = 1$ was used, showing that in this case, (7) gives a conservative bound for sufficient sampling of volumes based on the F-number, F#. The large black margins above and below the volume's spectra in Fig. 7 show that the volume was oversampled by a factor of around 1.6-2 axially. There was almost no oversampling for x and y in order to minimize the number of voxels to beamform. The LRIs from the first stage must be axially oversampled to allow cubic interpolation in the second stage. However, HRVs do not always need to be oversampled, depending on use [44]. Here, the same sampling was used for the HRVs to find out if complex-valued dense volumes can be computed in real-time, as this is the most computationally intense case that is relevant in practice. Note that LRIs from the first stage were taller than the output from the second stage to cover all observed look-up positions given by $z' = f(y, z)$ (see Table IV and (5)).

C. Processing Setup

An overview of the setup is given in Table III. The GPU was locked to a 2.7 GHz clock rate and a power limit of 600 W for a throughput of 88.47 TeraFLOPS (trillion 32-bit floating-point arithmetic operations per second). The processing kernels were written in CUDA C++ and used in MATLAB R2021b with the MEX interface from [61]. The host had 768 GB DDR4 RAM, which held the RF data before moving it to the GPU's RAM. The host processor was an Intel© Xeon© W-3223 CPU with a 3.5 GHz clock rate, but processing was offloaded to the GPU. The host-to-GPU connection matched the host-to-scanner connection of the Verasonics Vantage 256 scanner and can transfer approximately 6 GB/s in both directions simultaneously.

TABLE III
OVERVIEW OF PROCESSING SETUP

GPU	NVIDIA GeForce RTX 4090 @ 2.70 GHz
GPU driver	Version 550.54.14
GPU connection	PCI Express 3.0, 8 lanes (Same as the scanner.)
Software	CUDA 11.8, MATLAB R2021b, Ubuntu 20.04.1

Measurements were repeated using GPUs from two previous generations, a GeForce RTX 3090 and GeForce RTX 2080 Ti, to validate that the performance gains were not device-specific. Both of their peak clock rates were measured to be 1.95 GHz.

D. Processing

The imaging consisted of transferring RF data to the GPU, applying pre-processing, and finally, beamforming the volume. Two GPU kernels carried out the processing:

1) Matched filter: This kernel pre-processed the RF data. The raw RF channel data was converted to 32-bit floating point numbers in this kernel because it was stored as 16-bit integers. Matched filtration was then performed as the convolution

$$(r * f)[n] = \sum_{j=1}^{N_f} r[n - j + 1] \cdot f[j], \tag{18}$$

where the output spans $n = 1$ to $n = N_s + N_f - 1$. Here, $r[k]$ denotes the k'th acquired channel data sample, with $r[1]$ being the first acquired RF sample and the last being $r[N_s] = r[894]$. The number of coefficients for the FIR-filter f was $N_f = 23$. The definition in (18) would lead to out-of-bounds accesses in the sample data, such as $r[0]$ or $r[N_s + 1]$, which was handled by zero-padding the data, i.e. defining $r[k] = 0$ for $k \notin [1, N_s]$. Only a convolution was applied to produce real-valued outputs. For complex outputs (see Section III-B), the Hilbert transform, H, was applied to the filter, because $\mathrm{H}(r*f) = \mathrm{H}r*f = r*\mathrm{H}f$. Thus, the only difference from the real-valued case is that the kernel argument $f \in \mathbb{R}^{N_f}$ was replaced by some pre-computed $\mathrm{H}f \in \mathbb{C}^{N_f}$ to get $(r * \mathrm{H}f)[n]$. Finally, to produce I/Q outputs, these complex samples were multiplied by $\exp(-2i\pi \hat{f}_0 \Delta t\, n)$ (see Fig. 2) before writing them to the GPU RAM. Here, $\hat{f}_0$ is the estimated center frequency of $r*f$ (see Fig. 7 and Table II), and $\Delta t = 1/f_s$ is the time between samples, where $f_s = 31.25$ MHz is the RF data sampling rate. For I/Q data, a decimation was also applied to reduce the output-rate by a factor of three, i.e., getting $f_s^{\mathrm{IQ}} = 31.25/3 = 10.4$ MHz. This was performed by only computing every third output, resulting in a total of $(N_f + N_s - 1)/3$ (rounded up if non-integer) I/Q samples being computed and written to memory for the case of I/Q outputs.

Each thread computed one output for four batched channels, with 128 threads in a block. The block began by cooperatively converting input samples and moving them to shared memory, which is also where zero-padding was applied. Shared memory is an L1 cache partition shared within the block for fast access. Then, the block moved the filter to shared memory, after which a block-level synchronization was performed to ensure threads were finished writing to shared memory. The convolution (18) was finally computed using a loop of N_f iterations, where the samples and filter coefficients were read from shared memory, and the output was written to the GPU's RAM in the end.

The filter and RF samples were transferred from GPU RAM to shared memory by distributing the work among threads. For example, the j'th thread in the block moved $f[j]$, and it would write the j'th RF sample used in the block to shared memory. Note that the first sample used in the block might be padding. For instance, the first block needed $r[k]$ for $k \in [1 - N_f, 128]$ to compute outputs $n = 1 \ldots 128$ (assuming no decimation, for the I/Q case, $k \in [1-N_f, 128{\cdot}3]$ and $n \in \{1, 3, \ldots, 385\}$); here, the first thread would write $r[1-N_f] = 0$ (zero-padding), while the N_f'th thread would be the one to load $r[1]$, convert it to 32-bit floating-point format, and store it in shared memory.

Since four channels were batched, all non-padding samples needed four load instructions to retrieve their value from RAM. By writing these four samples to shared memory as a vector, storing and loading could be done using single fully vectorized (128-bit) shared memory instructions. Thus, batching reduced the number of load operations required as well as the overhead.

2) Beamformer: The beamformer from [61] was used, with the optimizations described in Section III. For 3D imaging, the beamformer was first used to calculate each LRI (xz-planes at an elevation position, y_0, that matches the emission source). Then, the beamformer was applied again for each x (yz-plane) in the volume to extrapolate and sum the LRIs by (5) and thus produce the final high-resolution volume.

The beamforming kernel's register usage (a fast thread-local memory) was tuned manually because the compiler's heuristic was not always optimal. There are a finite number of registers, so if each thread uses too many registers, it reduces the number of threads that can be active on the GPU, lowering *occupancy* [76]. A high occupancy is usually preferable for performance, but the multi-frame beamforming optimization benefits from more registers because this enables even more load instructions to be issued earlier to better mitigate latencies (see Section III). CUDA launch bounds were used to specify the register usage, which allows one to set the minimum number of thread blocks that should fit each multiprocessor (allowing the register usage to be increased till no less than this number of blocks can fit).

In the experiment, where the RF channel data was streamed from the host to the GPU while beamforming, this data transfer was overlapped with computations to avoid having to wait for transfers to complete; this was done by processing each LRI in a separate CUDA stream, enabling the LRIs to be computed concurrently. For example, then the second LRI could have its RF data transferred while the first LRI was being beamformed. In the experiment without compile-time constants, the receiver position array was moved to shared memory for faster loading. For further technical details, see Appendix D.

TABLE IV
THE THEORETICAL THROUGHPUT FOR THE GEFORCE RTX 4090.

	Real Data	I/Q Data
xz-plane size (1st stage)	133×544	133×271
yz-plane size (2nd stage)	115×500	115×250
Interpolations (1st stage)	$N_{tx} \cdot 5\,804\,364$	$N_{tx} \cdot 2\,888\,478$
Interpolations (2nd stage)	$M_x \cdot 4\,733\,065$	$M_x \cdot 2\,363\,733$
FLOP per interpolation	8	22
Total FLOP per volume	9 493 732 712	13 016 748 294
Theoretical rate [volumes/s]	9319	6797

The kernels were compiled with the `use_fast_math` flag for faster calculation of square root, division, sine, and cosine. The flag reduced each of these calculations to 1-2 instructions, with a minor reduction in accuracy (see section 13.2 of [76]).

E. Measuring Performance

The processing time was measured from RF channel data to the final volumes with the MATLAB function "`gputimeit`", which was called 25 times. The volume rate is the rate at which volumes are beamformed, and it was calculated as the average

$$R = \frac{1}{25} \sum_{i=1}^{25} \frac{F}{T_i}, \tag{19}$$

where T_i was the 25 reported timings, and $F \geq 68$ was the number of volumes beamformed each time. Real-time imaging is achieved when R exceeds the acquisition rate of 125 Hz.

Three other metrics may aid in comparing performance with other beamformers: 1) The voxel rate, which makes it easier to compare cases with different volume sampling densities. 2) The beamforming operation rate, which shows the throughput. 3) The efficiency, which shows how well the GPU is utilized. Here, 100% efficiency means every multiprocessor uses every clock cycle to issue arithmetic instructions for delay-and-sum. The voxel rate is

$$R_{voxel} = CR \cdot M_x M_y M_z \tag{20}$$

where R is the volume rate, $M_x M_y M_z$ is the number of voxels per volume (see Table II), and $C = 1$ for real-valued voxels. For complex or I/Q voxels, $C = 2$, to allow a fair comparison with the real-valued case, which requires twice as many voxels (Section III-B), but uses half as many calculations per voxel.

The beamforming operation rate, Ω, is the number of delays and summations per second that were attained. In the simplest case, this would be $\Omega = CR(M_x M_z N_{rx} N_{tx} + M_x M_y M_z N_{tx})$ (see Tables I and II). However, zero-weight terms are skipped due to early exit, and the LRIs in the first stage were taller than the volume. The exact number of non-zero beamforming terms was therefore instead counted empirically (see "interpolations" in Table IV) and multiplied with CR to get Ω. This means Ω is not improved by early exit, even though R is. But Ω allows one to compare different beamformers, even GPUs vs. ASICs.

Beamforming efficiency is defined as the measured volume rate divided by a theoretical volume rate. Each GPU device can perform some number of floating-point operations per second (FLOPS), and assuming these are only used for interpolations, a theoretical volume rate can be found. Other operations inside

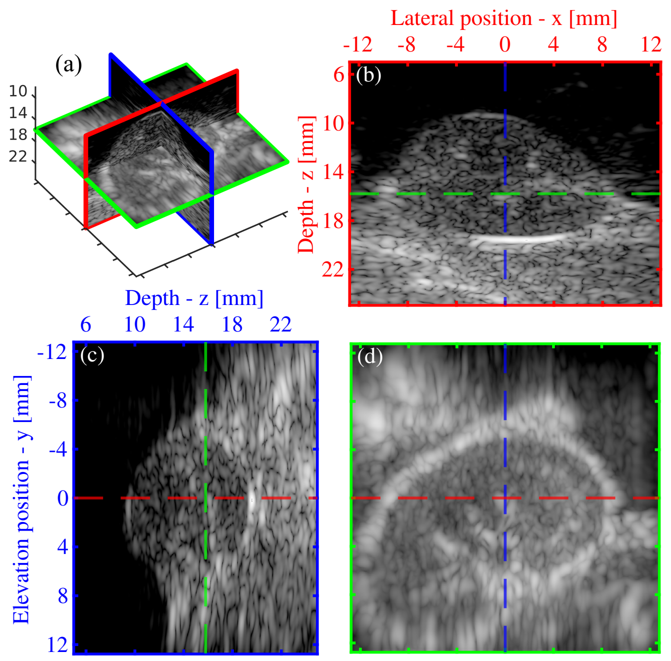


Fig. 8. Three views of the volume are shown in the same 60 dB dynamic range. (a) The images in space. (b) A xz-slice at $y = 0$. (c) A yz-slice at $x = 0$. (d) A maximum intensity projection at $z = 15.8 \pm 0.5$ mm, highlighting specular reflections from anatomical structures within the rat kidney; this oblique viewing angle is possible since the entire volume is reconstructed, meaning any plane in the field of view can be visualized.

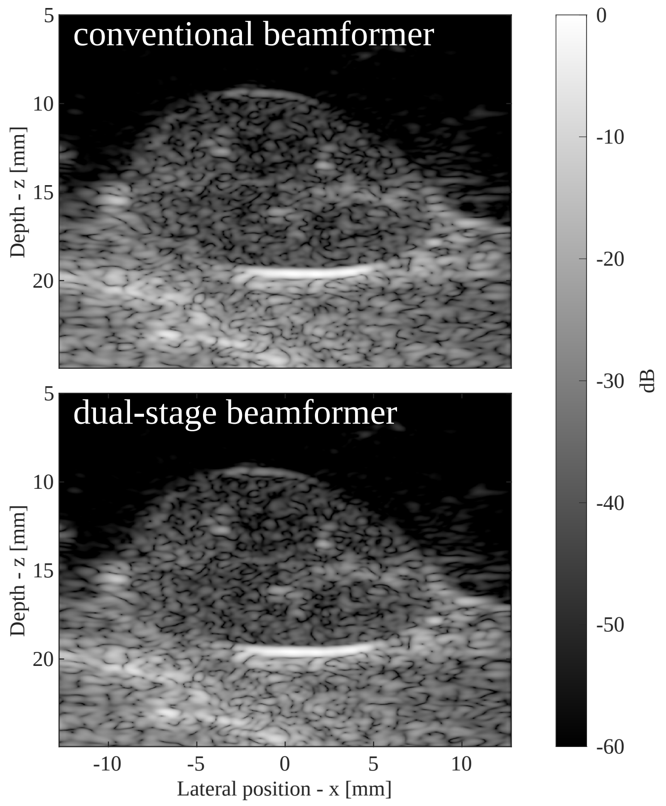


Fig. 9. A slice from the volume computed with the conventional method [26] (without any optimizations), and one computed using the dual-stage RCA beamforming algorithm, with optimizations for GPU processing.

or outside the kernel are considered reducible as optimizations showed, e.g., load operations and overhead could be reduced. Interpolations are irreducible since all ToF(x, z)-values could be unique and without redundant structure. Thus, the efficiency measures how much can be gained from further optimizations. The theoretical volume rate is shown in Table IV.

V. Results

Examples from a volume are shown in Fig. 8. Volumes were resized ×4, to remove any pixelated appearance in the display, with an FIR-based interpolation (see [70]) with 49 coefficients given by "`resample(HRV, 4, 1, 6, 8)`" in MATLAB.

A. Quality

The differences from using the `use_fast_math` compiler flag were in a similar range as floating-point rounding errors. Subtracting one volume created with this flag from one without it resulted in deviations below -75 dB, where 0 dB is the peak value from the volume created without the flag, showing that this flag does not significantly alter the beamforming accuracy.

The dual-stage beamformer with all optimizations enabled can be compared with the conventional beamforming method without any optimizations in Fig. 9. The number of interpolations is ≈ 49 times greater without the dual-stage algorithm, requiring over 90% efficiency to attain real-time 3D imaging.

B. Tuning

The beamformer's block size and batch size were optimized sequentially to find the best overall beamformer configuration.

To find the optimal block size in the first beamforming stage, the rate of matched filtering and first-stage beamforming was measured while the block size was varied. After this, the first beamforming stage block size was locked to its optimal value, and the matched filtering and dual-stage beamforming rate was measured while varying the second-stage block size to find the optimal size for the second stage. The candidate block sizes in this process were 1×128, 2×64, 4×32, $\ldots$, 64×2, 128×1, and a batch size of $B = 13$ was used. It was found that blocks of 128 threads spread laterally or axially would result in a 24-54% lower performance than the optimal size in the first stage. Thus, block size tuning significantly improves performance.

The optimal batch size was found while using the previously optimized block sizes for each stage. The same batch size was used in both stages, with the results shown in Fig. 10. One can see how the beamforming rate scales better with the batch size when I/Q data is used (compared to real-valued data). Further improved scaling comes with multi-pixel beamforming, which decreases the number of load instructions. However, the multi-pixel setup performs poorly for $B \geq 16$ because it reaches the per-thread register memory limit of 255 registers – additional local memory is instead stored in RAM using load instructions, which is detrimental to performance, but this may be detected at compile-time with the `warn-on-spills` compiler flag.

There is a dip in performance after $B = 7$ for real data, and the other curves would have had similar dips if register counts were not manually tuned. However, a better register count was not found for this specific case.

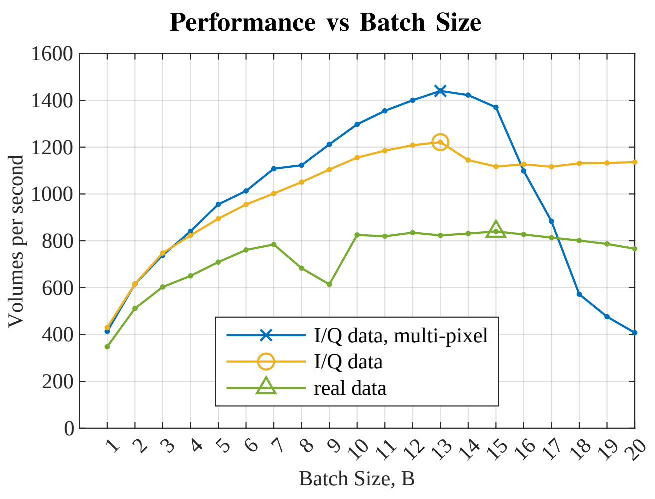


Fig. 10. The beamforming rate with different batch sizes and input data types. The markers show the best configuration for each case (Table V).

TABLE V
PERFORMANCE FOR THE NVIDIA GEFORCE RTX 4090

	Input RF channel data type		
	Real	I/Q	I/Q*
Volume rate [volumes/s]	840	1221	**1440**
Voxel rate [Mvoxels/s]	6424	9339	**11012**
Beamforming efficiency	10%	19%	**22%**
(1st stage) Block size	16×8	16×8	16×8
(2nd stage) Block size	16×8	32×4	8×16
Batch size	15	13	13

*with multi-pixel beamforming.

C. Overall Performance

Performance was measured with the RF data already on the GPU using the tuning process of the previous section. Metrics for the RTX 4090 device are shown in Tables V and VI, and it can be seen that the best-performing configuration was multi-pixel beamforming using I/Q RF data, with 13 frame batches, achieving a real-time rate of 1440 full 3D volumes per second.

D. Performance of Alternatives

Without using early exit (which occurred 22% of the time), the volume rate was 16% lower. The rate became 13% lower if compile-time constants were not used. For 2D imaging, 3018 HRI/s (xz-planes) could be computed, which was also the rate for producing the 96 LRIs in the first stage; this corresponds to 47% of the total processing time. Thus, *beamforming a volume took approximately twice as long as beamforming one image*. The efficiency for 2D imaging was also 22% (214 Mpixels/s).

When transferring RF data while beamforming, the rate was 209 volumes/s; at most, 273 volumes/s can be achieved in this setup since, at most, 6 GB/s can be transferred to the GPU.

The beamforming rate for the RTX 3090 was 638 volumes/s, and for the RTX 2080 Ti, it was 388 volumes/s, corresponding to an efficiency of 20% and 32% for these devices; this shows that the high efficiency is carried over to previous generations. The 2080 Ti has higher efficiency because Turing generation GPU devices have half as many FP32 cores per multiprocessor when compared to later generations [80], halving the achievable FLOPS and the theoretical beamforming rate; this means that the high efficiency here is a result of hardware limitations.

VI. DISCUSSION

A. Volume Rate

The real-time volumetric imaging rate was greatly exceeded. However, *in a live acquisition scenario, the volumes cannot be shown before they are acquired*, so in this setting, the imaging would be limited to 125 Hz, which was exceeded in all cases. Higher processing rates, in this case, means that

1) The pulse-repetition frequency might be set to 138 kHz.
2) Real-time 3D imaging may be achieved with a lower-end GPU or with a lower clock rate and energy consumption, reducing the carbon footprint and other costs.
3) There is time left over for display and further processing as only 9% of the time per frame is spent beamforming.

The emission rate will be limited by other factors in practice, but processing time is no longer a problem. The additional time may be spent on tasks such as resizing the volume, extracting slices, 3D velocity estimation [62], and visualizing the volume.

The upper limit for the emission rate is $f_s/N_s \approx 35$ kHz if emissions must not overlap in time because this corresponds to continuous sampling and emitting every N_s sample. Lowering the sample count, N_s, will raise this limit but reduce M_z since the depth of the volume corresponds to the number of samples. Thus, the emission rate is essentially physics-limited, and the GPU-optimized RCA beamformer supports any realizable rate.

The optimizations are also applicable for 2D beamforming. In fact, the first beamforming stage is equivalent to imaging an un-steered plane-wave emission in 2D, and the same efficiency was achieved for this stage (Section V-D). One may also apply the optimizations to separable beamforming for matrix arrays, where similar efficiency gains would be expected.

Finally, the results may be compared to other beamformers in Table VI, where it can be seen that delays-and-summations per second, Ω, is in the same order of magnitude as for ASICs, with the benefit that the GPU can be used for additional tasks such as 3D flow estimation, image analysis, and visualization. Table VI also shows the efficiency, and the beamformer in [61] had a $32/2.2 = 14.5$ times lower efficiency than the RTX 2080 Ti (a device from the same generation as the Titan V). Note, [54] used nearest-neighbor interpolation (after ×4 resampling), which is less efficient on the GPU than cubic interpolation.

B. Voxel Density

Before applying any optimizations, the most important step is to minimize the number of pixels/voxels since each point is expensive to beamform. To further reduce the LRI axial pixel density, the cubic Lagrange interpolation might be replaced by Lanczos interpolation. [81], which requires less oversampling; the cost of evaluating interpolation weights may be higher, but multi-frame beamforming significantly reduces such overhead.

One can also apply FIR-based interpolation (image resizing) to the LRIs to increase their degree of axial oversampling sufficiently for the second stage, even if they are only beamformed at an axial pixel density very close to their spatial Nyquist rate.

TABLE VI
COMPARISON OF VOLUMETRIC BEAMFORMERS

Source	Main Processor	Array Type	N_{rx}	Volume [voxels]	Emis. rate* [Hz]	BOP/emis.	Efficiency	Ω [BOPS]
2013 [50]	ASIC: Sonic Millip3De	128×96 Matrix	1024	50 × 50 × 4096	192	10 486 M		2 013 G
2020 [54]	ASIC: Tetris ×2	32×32 Matrix	1024	32 × 32 × 1679	13 020	123 M		2 621 G
2020 [54]	GPU: Titan Xp	32×32 Matrix	1024	32 × 32 × 1679	81	123 M	0.2%	10 G
2021 [61]	GPU: Titan V	64+64 RCA	64	63 × 63 × 557	120	283 M	2.2%	34 G
2023 [44]	GPU: GeForce RTX 3090	128+128 RCA	128	100 × 100 × 200	1 805	14 M	0.7%	26 G
This work	GPU: GeForce RTX 4090	128+128 RCA	128	133 × 115 × 250	138 240	12 M	22%	**1 650 G**

*The number of emissions processed per second, not necessarily the pulse-repetition frequency that was used during acquisition.

C. Multi-Frame Beamforming

Multi-frame beamforming gave the largest improvement of all the optimizations applied, increasing the imaging rate by a factor of three when combined with multi-pixel beamforming. A drawback is that if $B = 13$ volumes must be beamformed in one pass, this would introduce a noticeable delay of 104 ms to wait for 13 frames. The solution is to batch other dimensions.

The only difference between LRIs was the emission source y-position, but each was beamformed at an elevation matching their respective emission source, canceling out this difference. (See the time-of-flight function that was used in Appendix A.) Thus, all LRIs may be beamformed in a batch in the first stage. All yz-planes in the volume can always be batch-beamformed in the second stage because this is uniform for each x (see (5)). Therefore, the dual-stage beamformer can use large batch sizes (e.g., 96 in this case) without introducing additional buffering.

D. Transfer Rates

Volumes can be displayed without a data transfer to the host [76] as the computer screen is connected directly to the GPU. Each volume used $M_xM_yM_zC \cdot 32\,\text{bits} = 30.6$ MB of space, meaning $(6\ \text{GB/s})/(30.6\ \text{MB}) = 196$ volumes per second can be transferred to the host over the 8-lane PCIe 3.0 connection. However, the RTX 4090 can transfer 24 GB/s (784 volumes/s) using a 16-lane PCIe 4.0 connection, meaning an emission rate of 35 kHz is supported since this is 11 GB/s (364 volumes/s).

An 11 GB/s data rate might seem extreme (this corresponds to 1000 GB after 1.5 minutes), but if the sampling rate, f_s, and the number of receivers, N_{rx}, are doubled to support a higher center frequency, the resulting RF and volume data could be in this order of magnitude. The main limitation is the scanner, and there is a future need for research scanners for high-quality 3D imaging to support RF data transfer rates beyond 6 GB/s. It is possible to transfer this RF data directly from the scanner to the GPU, but the rate will still be limited to 6 GB/s.

E. Performance Limitations

The profiler NVIDIA Nsight™ Compute indicated that the *latency* of load operations was limiting the performance, with over 50% of cycles spent on waiting for load instructions. This might be reduced by further improving the L1 cache hit rates. For example, each block manages a rectangle of pixels, which ensures they are computed using the same L1 cache. However, there is no guarantee that adjacent rectangles share caches, but one may use locality-aware thread block scheduling [82], [83] to control which blocks (and pixel computations) share caches.

F. Limitations

An efficiency of 22% was achieved. Thus, the beamforming can only be made around four times faster without using fewer summations or switching to 16-bit numbers (half-precision).

Multi-frame beamforming assumes all values stay the same between frames except for the RF channel data. For example, it cannot be used if motion correction is performed by varying voxel positions between frames, as in [45]. Motion correction could help reduce motion artifacts from the (1/125 Hz) = 8 ms acquisition duration, which is a limitation of the RCA imaging. Alternatively, fewer emissions could be used for faster imaging but with lower image quality elevationally, or all 96 emissions could be fired (at a 35 kHz rate) in a burst every 8 ms to attain the same 125 Hz frame rate with fewer motion artifacts.

The time for post-processing of volumes was not measured since processing after beamforming depends on the use case. Any display-related processing should be performed 60 times per second if that is the screen's refresh rate, and then the GPU is free 95% of the time for this post-processing since it finishes beamforming in $1/(1440\ \text{Hz}) \approx 0.7$ ms. An example of such post-processing is the volume resizing before display in Fig. 8, which used 98 FLOP/voxel. This resizing of three slices costs $98 \cdot (M_xM_y + M_xM_z + M_yM_z) \cdot 4 + 98 \cdot (M_xM_y + M_xM_z + M_yM_z) \cdot 4 \cdot 4 = 148\,406\,400$ FLOP, which is approximately 3% of the cost of the beamforming, which used 1702 FLOP/voxel. However, one does not need to beamform the entire volume if only a few slices are displayed, but this enables more advanced 3D visualization, and arbitrary slices can be extracted from it.

VII. CONCLUSION

An imaging rate of 1440 volumes per second was achieved, greatly exceeding the 125 Hz acquisition rate. The processing time is no longer a limitation for volumetric beamforming with the 128+128 Vermon row-column array (RCA), even when the volumes have to be computed in real-time at high volume rates. Specifically, dense Nyquist-sampled volumes of the entire field of view could be processed, meaning any plane can be shown.

The proposed beamformer achieved 22% of the volume rate that is achievable using the NVIDIA GeForce RTX 4090 GPU, and a similar efficiency was also demonstrated for GPUs from two previous generations (along with real-time beamforming).

Volumetric imaging can thus be performed with row-column arrays in real-time at the patient's bedside with a setup similar to 2D imaging: one GPU, scanner, and 128 receiving channels, making this method of 3D imaging translatable to the clinic.

VIII. Acknowledgements

This work was financially supported by European Research Council's (ERC) Synergy Grant #854796. The authors thank Nathalie Sarup Panduro, Sofie Bech Andersen, Karin Larsen, and Charlotte Mehlin Sørensen (Dept. of Biomedical Sciences, University of Copenhagen, DK-2200, Copenhagen, Denmark) for setting up and performing the rat kidney acquisition.

References

[1] K. Evans, S. Roll, and J. Baker, "Work-related musculoskeletal disorders (WRMSD) among registered diagnostic medical sonographers and vascular technologists: A representative sample," *Journal of Diagnostic Medical Sonography*, vol. 25, no. 6, pp. 287–299, 2009.

[2] G. Harrison and A. Harris, "Work-related musculoskeletal disorders in ultrasound: Can you reduce risk?," *Ultrasound*, vol. 23, no. 4, pp. 224–230, 2015.

[3] J. Bakker, M. Olree, R. Kaatee, E. E. D. Lange, K. G. M. Moons, J. J. Beutler, and F. J. A. Beek, "Renal volume measurements: Accuracy and repeatability of US compared with that of MR imaging," *Radiology*, vol. 211, no. 3, pp. 623–628, 1999.

[4] S. H. van Vuuren, H. A. M. Damen-Elias, R. H. Stigter, R. van der Doef, R. Goldschmeding, T. P. V. M. de Jong, P. Westers, G. H. A. Visser, and L. R. Pistorius, "Size and volume charts of fetal kidney, renal pelvis and adrenal gland," *Ultrasound Obstet. Gyn.*, vol. 40, no. 6, pp. 659–664, 2012.

[5] H. Baumgartner, J. Hung, J. Bermejo, J. B. Chambers, A. Evangelista, B. P. Griffin, B. Iung, C. M. Otto, P. A. Pellikka, and M. Quiñones, "Echocardiographic assessment of valve stenosis: EAE/ASE recommendations for clinical practice," *J. Am. Soc. Echocardiog.*, vol. 22, no. 1, pp. 1–23, 2009.

[6] B. Cheong, R. Muthupillai, M. F. Rubin, and S. D. Flamm, "Normal values for renal length and volume as measured by magnetic resonance imaging," *Clin. J. Am. Soc. Nephrol.*, vol. 2, no. 1, pp. 38–45, 2007.

[7] D. D. Johnson, D. H. Pretorius, N. E. Budorick, M. C. Jones, K. V. Lou, G. M. James, and T. R. Nelson, "Fetal lip and primary palate: Three-dimensional versus two-dimensional US," *Radiology*, vol. 217, no. 1, pp. 236–239, 2000.

[8] B. L. Partik, A. Stadler, S. Schamp, A. Koller, M. Voracek, G. Heinz, and T. H. Helbich, "3D versus 2D ultrasound: Accuracy of volume measurement in human cadaver kidneys," *Investigative Radiology*, vol. 37, no. 9, pp. 489–495, 2002.

[9] J. Zamorano, P. Cordeiro, L. Sugeng, L. Perez de Isla, L. Weinert, C. Macaya, E. Rodríguez, and R. M. Lang, "Real-time three-dimensional echocardiography for rheumatic mitral valve stenosis evaluation: An accurate and novel approach," *J. Am. Coll. Cardiol.*, vol. 43, no. 11, pp. 2091–2096, 2004.

[10] I. A. Sebag, J. G. Morgan, M. D. Handschumacher, J. E. Marshall, F. Nesta, J. Hung, M. H. Picard, and R. A. Levine, "Usefulness of three-dimensionally guided assessment of mitral stenosis using matrix-array ultrasound," *Am. J. Cardiol.*, vol. 96, no. 8, pp. 1151–1156, 2005.

[11] B. R. Benacerraf, C. B. Benson, A. Z. Abuhamad, J. A. Copel, J. S. Abramowicz, G. R. Devore, P. M. Doubilet, W. Lee, A. S. Lev-Toaff, E. Merz, T. R. Nelson, L. D. Platt, D. H. Pretori, and I. E. Timor-Tritsch, "Three- and 4-dimensional ultrasound in obstetrics and gynecology: Proceedings of the american institute of ultrasound in medicine consensus conference," *J. Ultrasound Med.*, vol. 24, no. 12, pp. 1587–1597, 2005.

[12] E. Altiok, M. Frick, C. G. Meyer, G. Al. Ateah, A. Napp, A. Kirschfink, M. Almalla, S. Lotfi, M. Becker, L. Herich, W. Lehmacher, and R. Hoffmann, "Comparison of two- and three-dimensional transthoracic echocardiography to cardiac magnetic resonance imaging for assessment of paravalvular regurgitation after transcatheter aortic valve implantation," *Am. J. Cardiol.*, vol. 113, no. 11, pp. 1859–1866, 2014.

[13] M. Correia, J. Provost, M. Tanter, and M. Pernot, "4D ultrafast ultrasound flow imaging: in vivo quantification of arterial volumetric flow rate in a single heartbeat," *Phys. Med. Biol.*, vol. 61, no. 23, pp. L48–L61, 2016.

[14] M. S. Wigen, S. Fadnes, A. Rodriguez-Molares, T. Bjåstad, M. Eriksen, K. H. Stensæth, A. Støylen, and L. Løvstakken, "4-D intracardiac ultrasound vector flow imaging-feasibility and comparison to phase-contrast MRI," *IEEE Trans. Med. Imag.*, vol. 37, no. 12, pp. 2619–2629, 2018.

[15] L. T. Jørgensen, M. B. Stuart, and J. A. Jensen, "Transverse oscillation tensor velocity imaging using a row-column addressed array: Experimental validation," *Ultrasonics*, vol. 132, p. 106962, 2023.

[16] 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.

[17] 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.

[18] D. H. Turnbull and F. S. Foster, "Beam steering with pulsed two-dimensional transducer arrays," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 38, pp. 320–333, July 1991.

[19] 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.*, pp. 968–971, 2003.

[20] 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, pp. 837–847, April 2009.

[21] 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.

[22] 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.

[23] J. A. Jensen, M. Schou, L. T. Jørgensen, B. G. Tomov, M. B. Stuart, M. S. Traberg, I. Taghavi, S. H. Øygaard, M. L. Ommen, K. Steenberg, and et. al., "Anatomic and functional imaging using row-column arrays," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 69, pp. 2722–2738, October 2022.

[24] M. R. Sobhani, M. Ghavami, A. K. Ilkhechi, J. Brown, and R. Zemp, "Ultrafast orthogonal row-column electronic scanning (uFORCES) with bias-switchable top-orthogonal-to-bottom electrode 2-D arrays," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 69, no. 10, pp. 2823–2836, 2022.

[25] H. Bouzari, M. Engholm, S. I. Nikolov, M. B. Stuart, E. V. Thomsen, and J. A. Jensen, "Imaging performance for two row-column arrays," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 66, no. 7, pp. 1209–1221, 2019.

[26] 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.

[27] M. Flesch, M. Pernot, J. Provost, G. Ferin, A. Nguyen-Dinh, M. Tanter, and T. Deffieux, "4D in vivo ultrafast ultrasound imaging using a row-column addressed matrix and coherently-compounded orthogonal plane waves," *Phys. Med. Biol.*, vol. 62, no. 11, pp. 4571–4588, 2017.

[28] S. Holbek, T. L. Christiansen, M. B. Stuart, C. Beers, E. V. Thomsen, and J. A. Jensen, "3-D vector flow estimation with row-column addressed arrays," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 63, no. 11, pp. 1799–1814, 2016.

[29] J. Sauvage, J. Porée, C. Rabut, G. Férin, M. Flesch, B. Rosinski, A. Nguyen-Dinh, M. Tanter, M. Pernot, and T. Deffieux, "4D functional imaging of the rat brain using a large aperture row-column array," *IEEE Trans. Med. Imag.*, vol. 39, pp. 1884–1893, June 2020.

[30] J. A. Jensen, M. L. Ommen, S. H. Øygard, M. Schou, T. Sams, M. B. Stuart, C. Beers, E. V. Thomsen, N. B. Larsen, and B. G. Tomov, "Three-dimensional super resolution imaging using a row-column array," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 67, no. 3, pp. 538–546, 2020.

[31] A. Austeng and S. Holm, "Sparse 2-D arrays for 3-D phased array imaging - design methods," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 49, pp. 1073–1086, August 2002.

[32] A. Ramalli, E. Boni, E. Roux, H. Liebgott, and P. Tortoli, "Design, implementation, and medical applications of 2-D ultrasound sparse arrays," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, pp. 1–1, 2022.

[33] J. D. Larson, III, "2-D phased array ultrasound imaging system with distributed phasing." Patent US 5229933, July 1993.

[34] P. Santos, G. U. Haugen, L. Løvstakken, E. Samset, and J. D'hooge, "Diverging wave volumetric imaging using subaperture beamforming," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 63, no. 12, pp. 2114–2124, 2016.

[35] A. Fenster, D. B. Downey, and H. N. Cardinal, "Three-dimensional ultrasound imaging," *Phys. Med. Biol.*, vol. 46, pp. R67–R99, 2001.

[36] J. Provost, C. Papadacci, J. E. Arango, M. Imbault, M. Fink, J. L. Gennisson, M. Tanter, and M. Pernot, "3-D ultrafast ultrasound imaging in vivo," *Phys. Med. Biol.*, vol. 59, no. 19, pp. L1–L13, 2014.

[37] J. Kortbek, J. A. Jensen, and K. L. Gammelmark, "Sequential beamforming for synthetic aperture imaging," *Ultrasonics*, vol. 53, no. 1, pp. 1–16, 2013.

[38] J. Zhou, S. Wei, R. Jintamethasawat, R. Sampson, O. D. Kripfgans, J. B. Fowlkes, T. F. Wenisch, and C. Chakrabarti, "High-volume-rate 3-D ultrasound imaging based on synthetic aperture sequential beamforming with chirp-coded excitation," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 65, no. 8, pp. 1346–1358, 2018.

[39] J. Zhou, S. K. Mandal, B. L. West, S. Wei, U. Y. Ogras, O. D. Kripfgans, J. B. Fowlkes, T. F. Wenisch, and C. Chakrabarti, "Front-end architecture design for low-complexity 3-D ultrasound imaging based on synthetic aperture sequential beamforming," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 2, pp. 333–346, 2021.

[40] A. Tawfik, S. Stergiopoulos, and A. Dhanantwari, "A generic beamforming structure allowing implementation of adaptive processing schemes for 2-D & 3-D arrays of sensors," in *Proc. MTS/IEEE OCEAN*, vol. 1, pp. 369–373, Oct 1997.

[41] K. Owen, M. I. Fuller, and J. A. Hossack, "Application of X-Y separable 2-D array beamforming for increased frame rate and energy efficiency in handheld devices," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 59, no. 7, pp. 1332–1343, 2012.

[42] K. Owen, "Separable beamforming for ultrasound array." Patent US 9433398B2, 2016.

[43] M. Yang, R. Sampson, S. Wei, T. F. Wenisch, and C. Chakrabarti, "Separable beamforming for 3-D medical ultrasound imaging," *IEEE Trans. Sig. Proc.*, vol. 63, no. 2, pp. 279–290, 2015.

[44] L. T. Jørgensen, S. K. Præsius, M. B. Stuart, and J. A. Jensen, "Row-column beamformer for fast volumetric imaging," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 70, no. 7, pp. 668–680, 2023.

[45] L. T. Jørgensen, M. Schou, M. B. Stuart, and J. A. Jensen, "Tensor velocity imaging with motion correction," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 68, no. 5, pp. 1676–1686, 2020.

[46] 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.

[47] J. Amaro, B. Y. S. Yiu, G. Falcão, M. A. Gomes, and A. C. H. Yu, "Software-based high-level synthesis design of FPGA beamformers for synthetic aperture imaging," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 62, no. 5, pp. 862–869, 2015.

[48] 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.

[49] C. R. Hazard and G. R. Lockwood, "Theoretical assessment of a synthetic aperture beamformer for real-time 3-D imaging," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 46, pp. 972–980, 1999.

[50] R. Sampson, M. Yang, S. Wei, C. Chakrabarti, and T. F. Wenisch, "Sonic Millip3De: A massively parallel 3D-stacked accelerator for 3D ultrasound," *IEEE 19th International Symposium on High Performance Computer Architecture*, pp. 318–329, 2013.

[51] P. A. Hager, A. Bartolini, and L. Benini, "Ekho: A 30.3W, 10k-channel fully digital integrated 3-D beamformer for medical ultrasound imaging achieving 298M focal points per second," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 5, pp. 1936–1949, 2016.

[52] 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.

[53] D. Wildes, W. Lee, B. Haider, S. Cogan, K. Sundaresan, D. M. Mills, C. Yetter, P. H. Hart, C. R. Haun, M. Concepcion, J. Kirkhorn, and M. Bitoun, "4-D ICE: A 2-D array transducer with integrated ASIC in a 10-Fr catheter for real-time 3-D intracardiac echocardiography," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 63, no. 12, pp. 2159–2173, 2016.

[54] B. L. West, J. Zhou, R. G. Dreslinksi, O. D. Kripfgans, J. B. Fowlkes, C. Chakrabarti, and T. F. Wenisch, "Tetris: Using software/hardware co-design to enable handheld, physics-limited 3D plane-wave ultrasound imaging," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1209–1220, 2020.

[55] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical image processing on the GPU - past, present and future," *Med. Image Anal.*, vol. 17, no. 8, pp. 1073–1094, 2013.

[56] 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," *Int. J. Comput. Assist. Radiol. Surg.*, vol. 13, no. 6, pp. 759–767, 2018.

[57] 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, pp. 20–23, 2019.

[58] B. Y. S. Yiu, I. K. H. Tsang, and A. C. H. Yu, "Real-time GPU-based software beamformer designed for advanced imaging methods research," in *Proc. IEEE Ultrason. Symp.*, pp. 1920–1923, 2010.

[59] 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.

[60] C. Risser, H. Hewener, M. Fournelle, H. Fonfara, S. Barry-Hummel, S. Weber, D. Speicher, and S. Tretbar, "Real-time volumetric ultrasound research platform with 1024 parallel transmit and receive channels," *Applied Sciences*, vol. 11, no. 13, p. 5795, 2021.

[61] 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, "Real-time volumetric synthetic aperture software beamforming of row-column probe data," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 68, no. 8, pp. 2608–2618, 2021.

[62] L. T. Jørgensen, M. S. Traberg, M. B. Stuart, and J. A. Jensen, "Performance assessment of row-column transverse oscillation tensor velocity imaging using computational fluid dynamics simulation of carotid bifurcation flow," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 69, no. 4, pp. 1230–1242, 2022.

[63] M. B. Stuart, M. Schou, and J. A. Jensen, "Row-column beamforming with dynamic apodizations on a GPU," in *Proc. SPIE Med. Imag.*, p. 109550Q, 2019. Paper number 10955-20.

[64] J. J. Flaherty, K. R. Erikson, and V. M. Lund, "Synthetic aperture ultrasound imaging systems." Patent US 3,548,642, 1967.

[65] 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.

[66] C. H. Frazier and W. D. O'Brien, "Synthetic aperture techniques with a virtual source element," *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 45, no. 1, pp. 196–207, 1998.

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

[68] J. Y. Lu, H. Zou, and J. F. Greenleaf, "Biomedical ultrasound beam forming," *Ultrasound Med. Biol.*, vol. 20, pp. 403–428, 1994.

[69] C. Shannon, "Communication in the presence of noise," in *IEEE*, vol. 37, pp. 10–21, 1949.

[70] R. Schafer and L. Rabiner, "A digital signal processing approach to interpolation," *Proc. IEEE*, vol. 61, no. 6, pp. 692–702, 1973.

[71] E. Waring, "Problems concerning interpolations," *Philos. Trans. R. Soc.*, vol. 69, pp. 59–67, 1779.

[72] J. Kortbek, H. Andresen, S. Nikolov, and J. A. Jensen., "Comparing interpolation schemes in dynamic receive ultrasound beamforming.," in *Proc. IEEE Ultrason. Symp.*, pp. 1972–1975, 2005.

[73] S. Bergner, T. Moller, D. Weiskopf, and D. J. Muraki, "A spectral analysis of function composition and its implications for sampling in direct volume visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1353–1360, 2006.

[74] S. M. Gehlbach and R. E. Alvarez, "Digital ultrasound imaging techniques using vector sampling and raster line reconstruction," *Ultrason. Imaging*, vol. 3, no. 1, pp. 83–107, 1981.

[75] R. G. Pridham and R. A. Mucci, "Digital interpolation beamforming for low-pass and bandpass signals," *Proc. IEEE*, vol. 67, no. 6, pp. 904–919, 1979.

[76] NVIDIA, "CUDA C++ Programming Guide (Version 12.3)," tech. rep., NVIDIA Corporation, 2024. Accessed March 2024.

[77] Z. Jia, M. Maggioni, J. Smith, and D. P. Scarpazza, "Dissecting the NVidia Turing T4 GPU via microbenchmarking," tech. rep., Citadel, 2019.

[78] S. K. Præsius, M. B. Stuart, M. Schou, B. Dammann, H. H. B. Sørensen, and J. A. Jensen, "Real-time super-resolution ultrasound imaging using GPU acceleration," in *Proc. IEEE Ultrason. Symp.*, pp. 1–4, 2022.

[79] 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.

[80] NVIDIA, “Nvidia Ampere GA102 GPU architecture,” tech. rep., NVIDIA Corporation, 2020. Accessed February 2024.
[81] C. E. Duchon, “Lanczos filtering in one and two dimensions,” *J. Appl. Meteorol. Climatol.*, vol. 18, no. 8, pp. 1016–1022, 1979.
[82] A. Li, S. Song, W. Liu, X. Liu, A. Kumar, and H. Corporaal, “Locality-aware CTA clustering for modern GPUs,” *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 297–311, 04 2017.
[83] A. Ukarande, S. Patidar, and R. Rangan, “Locality-aware CTA scheduling for gaming applications,” *ACM Trans. Archit. Code Optim.*, vol. 19, no. 1, 2021.

Appendix

A. Time-of-Flight

For row-column array beamforming of the plane at the same elevation as the emission source: $\mathrm{LRI}(x,z) = \mathrm{LRV}(x, y^{tx}, z)$, the time-of-flight from [26] can be simplified to:

$$\mathrm{ToF}(x,z) = \frac{z + \sqrt{\Delta x^2 + z^2}}{c} \tag{21}$$

where (x, y^{tx}, z^{tx}) is the emission position, $(x^{rx}, y, 0)$ is the receiver position, c is the speed of sound, and $\Delta x = x - x^{rx}$. Note that the time-of-flight in (21) is the same for every LRI.

The extrapolation mapping [44] for the second beamforming stage (see (4)) was simplified to

$$f(y,z) = z + 0.5\left(\sqrt{\Delta y^2 + \Delta z^2} - \Delta z\right), \tag{22}$$

where $\Delta y = y - y^{tx}$ and $\Delta z = z - z^{tx}$. The mapping in (22) is valid for defocused emissions ($z^{tx} \le 0$), assuming that (21) was the time-of-flight used in the first beamforming stage.

B. Spatial Sampling Criteria

The sampling criteria for $\mathrm{LRI}(x,z) = \alpha(x,z) r(\mathrm{ToF}(x,z))$ is given by the rate-of-change of time-of-flight, as stated in (6). With the ToF from (21), the lateral rate-of-change becomes

$$\frac{\partial\,\mathrm{ToF}(x,z)}{\partial x} = \frac{1}{c} \cdot \frac{\Delta x}{\sqrt{\Delta x^2 + z^2}}. \tag{23}$$

Only points with a non-zero apodization need to be considered. In case of dynamic apodization (2), it may be non-zero when $\left|\mathrm{F\#} \cdot \Delta x / z\right| \le 0.5$, giving the following left-hand side for (6):

$$\max_{\alpha(x,z)\neq 0}\left|\frac{\partial\,\mathrm{ToF}(x,z)}{\partial x}\right| = c^{-1} \max_{\left|\mathrm{F\#}\cdot\Delta x/z\right|\le 0.5}\left|\frac{\Delta x}{\sqrt{\Delta x^2 + z^2}}\right| \tag{24}$$

$$= c^{-1} \max_{\left|x/z\right|\le 0.5/\mathrm{F\#}}\left|\frac{x}{\sqrt{x^2+z^2}}\right| \tag{25}$$

$$= c^{-1} \max_{|x|/\sqrt{1-x^2}\le 0.5/\mathrm{F\#}} |x| \tag{26}$$

$$= c^{-1}/\sqrt{(2\mathrm{F\#})^2 + 1} \tag{27}$$

The coordinate system was first shifted to be centered on the receiver position (without loss of generality) to arrive at (25). These coordinates were then scaled such that $x^2 + z^2 = 1$ to get to (26) (also without any loss of generality because $z > 0$). Finally, (26) is maximized by maximizing $|x|$, and the solution becomes $|x|/\sqrt{1-x^2} = 0.5/\mathrm{F\#} \Leftrightarrow |x| = 1/\sqrt{(2\mathrm{F\#})^2 + 1}$. Inserting the result (27) in (6) gives the lateral sampling criteria shown in (7). For the axial sampling criteria, one can see that

$$\left|\frac{\partial\,\mathrm{ToF}(x,z)}{\partial z}\right| = c^{-1}\left|1 + \frac{z}{\sqrt{\Delta x^2 + z^2}}\right| \tag{28}$$

is maximized for $\Delta x = 0$, giving the global maximum of $2/c$.

C. Pseudo-code examples

Pseudo-code for the beamforming is shown in Algorithm 1, where the early exit (Section III-D) in Line 7 should have been ”**if** $|\mathrm{apoidx}| \le 0.5$ **and** $0 \le \mathrm{FractionalAddress(tof)} \le N_s - 1$”, to prevent cubic extrapolations, but this detail is omitted from Algorithm 1 and other examples for brevity. For completeness, pseudo-code for auxiliary functions is given in Algorithm 5.

Algorithm 1 Beamforming of a pixel $(x,\, z)$ for the j’th LRI.

```
 1: function BEAMFORM($x, z$)
 2:     sum ← 0                                   ▷ Initialize the pixel to zero.
 3:     for $i \leftarrow 1$ to $N_{rx}$ do                     ▷ This loop implements (1).
 4:         $\Delta x \leftarrow x - x_i^{rx}$
 5:         apoidx ← F# · $\Delta x / z$                            ▷ See (2).
 6:         tof ← $(z + \sqrt{\Delta x^2 + z^2})/c$                   ▷ See (21).
 7:         if |apoidx| ≤ 0.5 then                         ▷ Early exit.
 8:             apo ← $\cos^2(\pi \cdot \mathrm{apoidx})$                  ▷ Hann apod.
            ▷ Now evaluate $r_{ij}(\mathrm{ToF}_i(x,z))$ using cubic interpolation
 9:             $\boldsymbol{s}$ ← FourNearestSamples($r_{ij}$, tof)
10:             $\boldsymbol{w}$ ← CubicWeights(tof)          ▷ Interp. weights.
11:             delayed ← $w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4$
12:             sum ← sum + apo · delayed
13:         end if
14:     end for
15:     $\mathrm{LRI}_j(x,z)$ ← sum                      ▷ Write pixel to memory.
16: end function
```

Pseudo-code for the extrapolation is shown in Algorithm 2, where $\mathrm{LRI}_j(x, \circ)$ in Line 10 is $\mathrm{LRI}_j(x,z)$ as a function of z. To compute the high-resolution volume (HRV), Algorithm 1 is executed for each pixel in each LRI, after which Algorithm 2 is executed for each voxel in the high-resolution volume.

Algorithm 2 Beamforming of the voxel at $(x,\, y,\, z)$.

```
 1: function BEAMFORMSECONDSTAGE($x, y, z$)
 2:     sum ← 0                                   ▷ Initialize the voxel to zero.
 3:     for $j \leftarrow 1$ to $N_{tx}$ do                     ▷ This loop implements (5).
 4:         $\Delta y \leftarrow y - y_j^{tx}$
 5:         $\Delta z \leftarrow z - z_j^{tx}$
 6:         apoidx ← F# · $\Delta y / \Delta z$                           ▷ See (2).
 7:         fval ← $z + 0.5(\sqrt{\Delta y^2 + \Delta z^2} - \Delta z)$    ▷ See (22).
 8:         if |apoidx| ≤ 0.5 then
 9:             apo ← $\cos^2(\pi \cdot \mathrm{apoidx})$
10:             $\boldsymbol{s}$ ← FourNearestSamples($\mathrm{LRI}_j(x, \circ)$, fval)
11:             $\boldsymbol{w}$ ← CubicWeights(fval)
12:             delayed ← $w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4$
13:             sum ← sum + apo · delayed
14:         end if
15:     end for
16:     $\mathrm{HRV}(x,y,z)$ ← sum                    ▷ Write voxel to memory.
17: end function
    ▷ Note how $x$ plays an identical role to $j$ in Algorithm 1.
```

A multi-frame case (Section III-A) is shown in Algorithm 3 for the first stage. Here, LRIs can be multi-frame-beamformed because tof and apoidx are the same for every LRI (every j). A multi-pixel (Section III-C) case is shown in Algorithm 4.

Algorithm 3 Multi-frame beamforming for LRI j to $j+B$.

```
function MultiFrameBeamform($x, z$)
    sum$[0 \dots B-1] \leftarrow 0$    ▷ Initialize array of $B$ pixels.
    for $i \leftarrow 1$ to $N_{rx}$ do    ▷ Same as in Algorithm 1.
        $\Delta x \leftarrow x - x_i^{rx}$    ▷ Also the same.
        apoidx $\leftarrow$ F# $\cdot \Delta x / z$    ▷ Also the same.
        tof $\leftarrow (z + \sqrt{\Delta x^2 + z^2})/c$    ▷ Also the same.
        if $|$apoidx$| \leq 0.5$ then    ▷ Also the same.
            $\boldsymbol{w} \leftarrow \cos^2(\pi \cdot$ apoidx$) \cdot$ CubicWeights(tof)
    ▷ Compute the $B$ delay-and-sum terms while reusing $\boldsymbol{w}$.
            for $b \leftarrow 0$ to $B-1$ do
                $\boldsymbol{s} \leftarrow$ FourNearestSamples($r_{i,j+b}$, tof)
                delayed $\leftarrow w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4$
                sum$[b] \leftarrow$ sum$[b]$ + delayed
            end for
        end if
    end for
    LRI$_{j \dots j+B}(x, z) \leftarrow$ sum$[0 \dots B-1]$    ▷ Write pixels.
end function
```

Algorithm 4 Multi-pixel beamforming for the j'th LRI.

```
function MultiPixelBeamform($x_1, x_2, x_3, z$)
    sum$[1 \dots 3] \leftarrow 0$    ▷ Initialize array of three pixels.
    for $i \leftarrow 1$ to $N_{rx}$ do
        for $p \leftarrow 1$ to $3$ do    ▷ First check for early exit.
            $\Delta x \leftarrow x_p - x_i^{rx}$    ▷ Note $\Delta x$ varies with $p$.
            apoidx$[p] \leftarrow$ F# $\cdot \Delta x / z$
            tof$[p] \leftarrow (z + \sqrt{\Delta x^2 + z^2})/c$
            good$[p] \leftarrow \big|$apoidx$[p]\big| \leq 0.5$    ▷ Early exit.
        end for
    ▷ Load four samples only if any pixel does not exit early.
        if good[1] or good[2] or good[3] then
            $\boldsymbol{s} \leftarrow$ FourNearestSamples($r_{ij}$, tof[2])
            baseaddr $\leftarrow$ Address(tof[2])    ▷ Address of $s_1$.
        end if
    ▷ At most two additional samples will be loaded below.
        for $p \leftarrow 1$ to $3$ do
            if good$[p]$ then    ▷ Early exit.
                apo $\leftarrow \cos^2(\pi \cdot$ apoidx$[p])$
                $\boldsymbol{w} \leftarrow$ CubicWeights(tof$[p]$)
                addrdiff $\leftarrow$ Address(tof$[p]$) $-$ baseaddr
                if addrdiff $= 0$ then    ▷ Same samples.
                    delayed $\leftarrow w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4$
                else if addrdiff $= 1$ then    ▷ Adjacent.
                    $s_5 \leftarrow$ LoadSample($r_{ij}$, baseaddr $+ 4$)
                    delayed $\leftarrow w_1 s_2 + w_2 s_3 + w_3 s_4 + w_4 s_5$
                else if addrdiff $= -1$ then    ▷ Adjacent.
                    $s_0 \leftarrow$ LoadSample($r_{ij}$, baseaddr $- 1$)
                    delayed $\leftarrow w_1 s_0 + w_2 s_1 + w_3 s_2 + w_4 s_3$
                else    ▷ This should not happen.
                    delayed $\leftarrow$ NaN
                end if
                sum$[p] \leftarrow$ sum$[p]$ + apo $\cdot$ delayed
            end if
        end for
    end for
    LRI$_j([x_1, x_2, x_3], z) \leftarrow$ sum$[1 \dots 3]$    ▷ Write pixels.
end function
```

Algorithm 5 Auxiliary functions.

```
function FourNearestSamples($r_{ij}$, tof)
    ▷ Loads the four samples nearest "tof" for an interpolation.
    baseaddr $\leftarrow$ Address(tof)    ▷ Address of $s_1$.
    $s_1 \leftarrow$ LoadSample($r_{ij}$, baseaddr $+ 0$)
    $s_2 \leftarrow$ LoadSample($r_{ij}$, baseaddr $+ 1$)
    $s_3 \leftarrow$ LoadSample($r_{ij}$, baseaddr $+ 2$)
    $s_4 \leftarrow$ LoadSample($r_{ij}$, baseaddr $+ 3$)
    return $[s_1, s_2, s_3, s_4]$    ▷ Return the vector $\boldsymbol{s}$.
end function

function CubicWeights(tof)
    ▷ Weights for Lagrange interpolation [71] on $\tau \in [0, 3]$.
    $\tau \leftarrow$ FractionalAddress(tof) $-$ Address(tof)
    $w_1 \leftarrow -(\tau - 1)(\tau - 2)(\tau - 3)/6$
    $w_2 \leftarrow \tau(\tau - 2)(\tau - 3)/2$
    $w_3 \leftarrow -\tau(\tau - 1)(\tau - 3)/2$
    $w_4 \leftarrow \tau(\tau - 1)(\tau - 2)/6$
    return $[w_1, w_2, w_3, w_4]$    ▷ Return the vector $\boldsymbol{w}$.
end function

function FractionalAddress(tof)
    ▷ Convert a time-of-flight to the corresponding RF sample
    index using the sampling rate $f_s$ and first-sample-time $t_0$.
    return (tof $- t_0) \cdot f_s$
end function

function Address(tof)
    ▷ Convert a time-of-flight to the address offset of $s_1$.
    addr $\leftarrow$ FractionalAddress(tof) $- 1$ ▷ F.Address of $s_1$.
    ▷ Then, to protect against out-of-bounds array accesses:
    addr $\leftarrow \min(\max(\text{addr}, 0), N_s - 4)$
    return $\lfloor$addr$\rfloor$    ▷ Round down to integer.
end function

function LoadSample($r_{ij}$, addr)
    ▷ C-style array access for RF data of size $N_s \times N_{rx} \times N_{tx}$.
    return $r[\text{addr} + i \cdot N_s + j \cdot N_s N_{rx}]$
    ▷ = $r[(jS + iN_s + \text{baseaddr}) + (c + bS)]$, where $S = N_s N_{rx}$.
end function

function LoadSample(LRI$_j(x, \circ)$, addr)
    ▷ C-style array access for LRI data of size $M_z \times N_{tx} \times M_x$.
    ▷ Essentially, LRI$_j(x, \circ)$ is equivalent to a $r_{jx}(\circ)$.
    return LRI$[\text{addr} + j \cdot M_z + x \cdot M_z N_{tx}]$
end function

function PhaseRotate(delayed, tof)
    ▷ Apply this after interpolation (Line 11 in Algorithm 1)
    to use quadrature sampled RF inputs (see Section III-B).
    return delayed $\cdot$ $(\cos(2\pi \hat{f}_0 \text{tof}) + \sqrt{-1}\sin(2\pi \hat{f}_0 \text{tof}))$.
end function

function PhaseRotate(delayed, fval)
    ▷ Apply after Line 12 in Algorithm 2 to use I/Q LRIs.
    return delayed $\cdot \exp(2\pi\sqrt{-1}(2\hat{f}_0/c)\text{fval})$
end function
```

D. Further Technical Details

The main result of an imaging rate of 1440 volumes/s was achieved with the RF data already in the GPU's memory, and thus, no memory transfer was necessary for this result. In this case, the RF data was stored in an int16 MATLAB gpuArray, which was passed to the MEX file and accessed as a C-style array using mxGPUGetDataReadOnly. A gpuArray to hold the beamformed result was created with mxGPUCreateGPUArray and accessed with mxGPUGetData; thus, both the inputs and outputs were stored in GPU memory managed by MATLAB.

The first kernel performed matched filtering of the RF data, and its output was stored inside an internal buffer in the MEX file (created using cudaMalloc). Likewise, instead of returning LRIs from the first beamforming stage to MATLAB, they were simply written to an internal buffer, which was then read from the second beamforming stage kernel to compute the volume. These two intermediate result buffers were reused every frame, meaning there was no "memory management" in the MEX file beyond the initial creation of internal buffers for intermediates.

A volume-beamforming consisted of N_{tx} calls to the MEX file to launch the matched filtering and first-stage beamforming kernels and compute N_{tx} LRIs, followed by one call to create a volume gpuArray and launch the second beamforming stage. Each thread executing the first-stage beamformer kernel would start off by identifying the three pixels it should compute by:

```
z_idx = blockIdx.x*blockDim.x+threadIdx.x
x_idx[0]=(blockIdx.y*blockDim.y+threadIdx.y)*3
x_idx[1]=x_idx[0]+1
x_idx[2]=x_idx[0]+2
```

For a thread block size of 16×8 in the first stage and an LRI output of 133×271 (see Table V and Table IV), the necessary launch grid size was $\lceil \frac{271}{8} \rceil \times \lceil \frac{133}{16 \cdot 3} \rceil = 34 \times 3$ blocks to ensure there were threads computing every pixel. Each kernel launch thus resulted in 102 blocks or 13056 threads; essentially, 48×272 threads were used to cover an output size up to 144×272. This is an excessive amount of threads, so the following guard is placed at the beginning of the kernel to prevent threads from writing outside the LRI image bounds (crashing the program)

```
if not (x_idx[0]<133 and z_idx <271) return
```

Note that x_idx[1] and x_idx[2] may still go outside the bounds, but similar a guard was placed at the end of the kernel, where these two pixels are written to RAM only if they are inside the bounds. Thus, more threads are created than strictly necessary, but most excess threads finish executing almost immediately. This is a standard pattern for distributing work among threads for GPU parallelization in CUDA; see, e.g., section 2.2 in [76].

The second beamforming stage kernel used a similar scheme

```
z_idx = blockIdx.x*blockDim.x+threadIdx.x
y_idx[0]=(blockIdx.y*blockDim.y+threadIdx.y)*3
y_idx[1]=y_idx[0]+1
y_idx[2]=y_idx[0]+2
x_idx = blockIdx.z
if not (y_idx[0]<115 and z_idx <250) return
```

Instead of launching this kernel repeatedly $M_x = 133$ times to compute every yz-plane in the volume, it was simply launched once using a three-dimensional grid. The result was a $\lceil \frac{250}{16} \rceil \times \lceil \frac{115}{8 \cdot 3} \rceil \times M_x = 16 \times 5 \times 133$ grid with 10640 thread blocks.

The beamforming was essentially the following algorithm, where loops over pixels and voxels were parallelized on GPU.

```
// First beamform LRIs.
for j = 1 to Ntx do
  for x in xpositions
    for z in zpositions
      LRI_j(x, z) = Beamform(x, z)

// Then beamform the volume using these LRIs.
for x in xpositions
  for y in ypositions
    for z in zpositions
      HRV(x,y,z) = BeamformSecondStage(x,y,z)
```

Here, "Beamform" is from Algorithm 1, and "BeamformSecondStage" is from Algorithm 2. The initial j-loop to produce LRIs might also be distributed over threads using a 3D grid. However, it was split into N_{tx} kernel launches to enable each emission to be processed on a separate CUDA stream to speed up processing when the RF data was not already on the GPU (see Section IV-D.2.) The result of parallelizing computations by launching the same kernel in separate streams is essentially the same as using a 3D grid, but it allows for more fine-grained control of the order of operations, enabling kernel execution to overlap with memory transfers; see 3.2.8.5 in [76] for details.

All thread-local memory must be held in registers to get the optimal performance in the beamforming kernels. It is recommended to use the flags "`-Xptxas -warn-lmem-usage`", "`-Xptxas -warn-spills`" when compiling to verify this.

As a final detail, when multi-frame beamforming was used, each yz-plane in the volume was essentially treated as a frame. When processing F volumes in the second beamforming stage, the grid dimensions would then be $16 \times 5 \times \lfloor 133F/B \rfloor$, with

```
x_idx = blockIdx.z*B
```

Note that $\lfloor 133F/B \rfloor$ is rounded down. If $133F$ is not divisible by B, there would be a second kernel launch that used a batch size of $133F \bmod B$ (the remainder of $133F$ divided by B) to compute the few remaining planes. This design allows multi-frame beamforming to be used without having to store multiple volumes in memory and without having to wait for B frames since it can be used even for $F = 1$, as outlined in Section VI-C. To avoid bias in the performance measurements depending on whether $133F$ is divisible by B, it was the case that $F = B$, which is why at least 68 volumes were beamformed to measure the rate in (19). For example, with $B = 13$, 78 volumes would be processed in batches of $F = B = 13$ frames.

Sebastian Kazmarek Præsius received the M.Sc. degree in Mathematical Modelling and Computation from the Technical University of Denmark (DTU) in 2022. He is currently a Ph.D. student at the Center for Fast Ultrasound imaging, working on real-time 2-D and 3-D beamforming, motion estimation, and super-resolution. His research interests include GPU acceleration, Deep Learning, and computer graphics.

Lasse Thurmann Jørgensen received the M.Sc. degree in 2019 and the Ph.D. Degree in 2022 from the Technical University of Denmark (DTU), Lyngby, Denmark. He is currently a postdoctoral researcher at the Center for Fast Ultrasound imaging, Department of health technologies, DTU, where his research topics include imaging and motion estimation with 3-D ultrasound systems.

Jørgen Arendt Jensen (M'93-SM'02-F'12) received the M.Sc. degree in 1985, the Ph.D. degree in 1989, and the Dr.Techn. degree in 1996, all from the Technical University of Denmark. Since 1993, he has been a Full Professor of Biomedical Signal Processing with the Department of Health Technology, Technical University of Denmark. He has been the founder and head of the Center for Fast Ultrasound Imaging since its inauguration in 1998. CFU has contributed with innovations in transverse oscillation vector flow imaging, synthetic aperture flow imaging in 2-D and 3-D, ultrasound simulation, research scanners, and row-column probes and beamforming. He has published more than 500 journal and conference papers on signal processing and medical ultrasound and the book Estimation of Blood Velocities Using Ultrasound (Cambridge Univ. Press), 1996. He is also the developer and maintainer of the Field II simulation program. He has been a visiting scientist at Duke University, Stanford University, and the University of Illinois at Urbana-Champaign. He was the founder and head of the Biomedical Engineering group from 2007 to 2010. In 2003, he was one of the founders of the biomedical engineering program in Medicine and Technology, which is a joint degree program between the Technical University of Denmark and the Faculty of Health and Medical Sciences at the University of Copenhagen. The degree is one of the most sought-after engineering degrees in Denmark. He was chairperson of the study board from 2003 to 2010 and Adjunct Professor with the University of Copenhagen from 2005 to 2010. He has given several short courses on simulation, synthetic aperture imaging, and flow estimation at international scientific conferences and teaches biomedical signal processing and medical imaging at the Technical University of Denmark. His research is centered around the simulation of ultrasound imaging, synthetic aperture imaging, vector blood flow estimation, 3-D and super-resolution imaging, row-column probes, and the construction of ultrasound research systems.