Fast GPU-beamforming of Row-Column Addressed Probe Data

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

Published in:
Proceedings of 2019 IEEE International Ultrasonics Symposium

Link to article, DOI:
10.1109/ultsym.2019.8925802

Publication date:
2019

Document Version
Peer reviewed version

Link back to DTU Orbit

Citation (APA):

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

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

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Fast GPU-beamforming of Row-Column Addressed Probe Data

Matthias Bo Stuart*, Patrick Møller Jensen†, Julian Thomas Reckeweg Olsen‡, Alexander Borch Kristensen*, Mikkel Schou*, Bernd Dammann†‡, Hans Henrik Brandenburg Sørensen‡, and Jørgen Arendt Jensen*

*Center for Fast Ultrasound Imaging, DTU Health Technology, Technical University of Denmark
†DTU Compute, Technical University of Denmark
‡DTU Computing Center, Technical University of Denmark

Abstract— A delay-and-sum beamformer for 3D imaging using row-column arrays and written in CUDA is presented and compared to an existing similar GPU-based beamformer written in the MATLAB programming language. Data from a 192×192 row-column array single element emission sequence is simulated and beamformed. The two beamformers’ performance is evaluated in two synthetic aperture setups comprised of 1) two orthogonal planes and 2) a full volume on three different NVIDIA GPUs: a 1050 Ti, a 1080 Ti, and a TITAN V. The execution time and the sample throughput (samples beamformed per second) are reported. The CUDA beamformer performs consistently better than the MATLAB beamformer with speed-ups ranging from 1.9 to 64.6 times, and the worst-case throughput of the CUDA beamformer exceeds the best-case of the MATLAB beamformer. High-resolution images of crossing planes can be beamformed at up to 13 Hz, while a 50-by-50-by-20 cubic-millimeter high-resolution volume sampled at one quarter of a millimeter is beamformed in 3 seconds.

I. INTRODUCTION

3D ultrasound imaging requires a 2D distribution of transducer elements to steer the beam in both azimuth and elevation. Fully populated arrays can be full matrices [1] or crossed electrodes that are also known as row-column (RC) arrays [2]. While the interconnect scales quadratically with the number of transducer elements for matrix arrays, it scales linearly for RC arrays. The number of elements translates to resolution, contrast, and better signal-to-noise ratio and thus penetration, where a RC array performs better than a matrix array with the same number of elements [3]. Flat RC arrays are limited to a forward-facing field-of-view, which can be remedied by applying a defocusing lens [4], [5]. The imaging performance of RC arrays has been verified by a number of groups [6]–[9], and volumetric 3D vector flow imaging has been demonstrated [10], [11].

Another advantage of RC arrays’ linear scaling is that volumetric images are made from an equivalent amount of data to that used for 2D imaging. This greatly alleviates the issue of the memory wall [12] for 3D ultrasound imaging making frame rates depend primarily on processing speed instead of memory bandwidth. Beamforming is inherently parallel since no computational dependency exists between output samples, and parallel processors such as GPUs can deliver real-time 2D imaging [13], [14]. This work investigates and compares the performance of two RC beamformers on three different GPUs.

Section II presents the two beamformers being investigated, Section III describes the experiments conducted to evaluate their performance and the performance metrics used, Section IV presents the results, Section V discusses the results, and Section VI offers conclusions.

II. BEAMFORMERS

Two delay-and-sum (DAS) beamformers are written for execution on GPUs. Both beamformers take as input the geometry (transmit focus, receive element coordinates, and image geometry), apodization parameters, speed of sound, and sampling frequency of the data. Dynamic apodization is supported by both beamformers parametrized on the F-number and window function.

The RC array’s long elements result in cylindrically shaped waves – rather than the spherical waves from small elements – which requires a different delay model for DAS beamforming [15]. A side effect of this is that the delay calculation can be reduced to a 2D problem through appropriate choice of the coordinate system [15], [16]. This reduction is used in both beamformers.

One beamformer is written entirely in the MATLAB programming language using the gparray type for GPU processing and has been presented elsewhere [16]. The other beamformer is written in C++ using NVIDIA’s CUDA extensions and called from MATLAB using the MEX interface. These two beamformers perform the same calculations with some differences in implementation outlined below. They are referred to as the MATLAB and the CUDA beamformer, respectively. An overall description is given of the MATLAB beamformer, while the CUDA beamformer is described in more detail.

A. MATLAB beamformer

The MATLAB beamformer has been presented and evaluated on a single GPU previously [16] and is briefly summarized here. All intermediate results (transmit, receive, and total delays and apodization values) are stored in memory on the GPU for all output samples to minimize recomputations at the cost of increased memory transfers. This essentially creates memory and apodization tables that are only recalculated if the related input parameters change. Apodization values are calculated by evaluating the window function with the
appropriate parameters. Sub-sample precision is attained using cubic interpolation, and the image geometry is specified as a list of the output samples’ coordinates.

B. CUDA beamformer

CUDA uses a single instruction, multiple threads (SIMT) execution model, where all program threads execute the same set of instructions. For efficient execution, threads should have equal (or similar) workload and have a minimum of synchronization among other considerations. The general beamforming problem has two immediate approaches for distribution across threads: 1) for a window of input samples, calculate the output values these input samples contribute to, and 2) for a given output sample, load the corresponding input samples and calculate the output value. The first approach is likely to result in uneven workloads between threads and requires synchronization for summing the different contributions to each output value. The second approach has equal workload for all threads and requires no synchronization. The CUDA beamformer therefore calculates one output sample per thread, i.e., each thread calculates the sum across all receive channels.

The intermediate results (delays and apodization values) are recalculated every time they are needed. The calculation is performed based on a minimal set of parameters (tens of bytes) as opposed to the delay and apodization tables read from memory by the MATLAB beamformer. The image geometry is specified as a set of lines with an origin and a step vector, where all lines must have the same number of samples, and 3rd order Lagrange interpolating polynomials are used for subsample interpolation.

A cumulative sum of images is maintained in GPU memory for synthetic aperture (SA) imaging [17], [18] to avoid transferring each low-resolution image out of the GPU.

III. EXPERIMENTS

The two beamformers are evaluated on three different NVIDIA GPUs: a GeForce 1050 Ti, a GeForce 1080 Ti, and a TITAN V. CUDA toolkit version 9.0 and MATLAB R2018a are used.

A single-element SA imaging sequence with a 192+192 \( \lambda/2 \) RC array is used for two use-cases: 1) beamforming two orthogonal planes and 2) beamforming a volume. The output is sampled at \( \lambda/2 \) in all directions. In setup 1, the XZ and YZ planes contain 192 \( \times \) 76 samples each. In setup 2, the volume is 192 \( \times \) 192 \( \times \) 76 samples \((x \times y \times z)\). For the MATLAB beamformer, the volume in setup 2 needs to be split in sub-volumes due to the memory used for intermediate results. For the 1050 Ti with 4 GB memory, 21 sub-volumes are needed to cover the full volume, the 1080 Ti with 11 GB memory needs 8 sub-volumes, and the TITAN V with 12 GB needs 7 sub-volumes. The inputs to the beamformers are analytic signals from a Field II simulation [19], [20] represented using double precision floating point numbers.

For performance measurements, execution times are measured using the wall-clock time, while the respective systems are otherwise unused. The same beamforming operations and read-out of the results to main memory were repeated 10 times in a loop, and each loop iteration was timed. The first three iterations were discarded to eliminate initialization effects, while the mean and standard deviation were calculated from the remaining seven iterations. Reads and writes from and to disk are not included in the timing, and a routine to flush the CPU caches was called between each iteration to avoid a false speed-up from beamforming the same data repeatedly.

The sample throughput, \( N_{sps} \) (number of samples beamformed per second) is calculated as

\[
N_{sps} = \frac{N_l N_s N_{rx} N_{tx}}{\bar{t}}
\]

where \( N_l \) is the number of image lines, \( N_s \) is the number of samples per line, \( N_{rx} \) is the number of receive elements, \( N_{tx} \) is the number of transmit events (the number of low-resolution images used to make a high-resolution image), and \( \bar{t} \) is the mean execution time.

Denoting the mean execution time of the MATLAB beamformer \( \bar{t}_M \) and of the CUDA beamformer \( \bar{t}_C \), the speed-up, \( s \), is calculated as

\[
s = \frac{\bar{t}_M}{\bar{t}_C}.
\]

IV. RESULTS

Fig. 1 shows the YZ planes of setup 1 for both beamformers. The point spread functions (PSFs) have only small deviations with a mean difference of -79 dB relative to the peak of the envelopes. The two beamformers have slightly different implementations of the calculations as described in Section II, which may be the cause of these deviations. Both PSFs have the same resolution measured as the full-width at half-maximum (FWHM) of 2.4\( \lambda \) in the y-direction and 0.66\( \lambda \) in the z-direction, where \( \lambda \) is the wavelength at the transducer’s center frequency. The 20 dB cystic resolution [21] is 1.9\( \lambda \) in the YZ plane for both beamformers.

Fig. 2 shows the performance of the beamformers on both setups. The vertical axis shows the sample throughput for each setup and GPU. Tables I and II show the mean and standard deviation of the execution time for each GPU for setups 1 and 2, respectively.

The standard deviations on the execution times are very low, showing consistent running times, which is to be expected of...
Fig. 1. PSF from the MATLAB (left) and CUDA (right) beamformers for the YZ plane of setup 1. Contour lines are shown at 6 dB intervals down to -42 dB.

Fig. 2. Performance of the beamformers measured as the number of beamformed samples per second.
required in SA imaging with the same \( \lambda/2 \)-pitch probe showed minimal differences between using 256 and 32 emissions [23]. This translates directly to an increase in frame rate by a factor of 3 to 8. This corresponds to frame rates of 52 Hz to 112 Hz for setup 1, or 1 to 2.5 full volumes per second for setup 2 with the CUDA beamformer on the TITAN V. This, of course, assumes the beamformer throughput is independent of the number of emissions, which is reasonable since the only added overhead is the read-out of the high-resolution image or volume. This is 45 MBytes for setup 2 using double precision complex values. This should be contrasted to the 15.75 GB/s maximum bandwidth of 16-lane PCI Express 3 interfaces used by many contemporary GPUs.

Further speed-ups can be attained by using single-precision floating point numbers or fixed point integer calculations. Experiments using single-precision floating point numbers indicate that the frame rate of setup 1 can be improved from 1.5 Hz to 13 Hz for the 1080 Ti, and from 14 Hz to 20 Hz for the TITAN V. The 1080 Ti has 32 times as high performance for single precision compared to double precision operations due to differences in hardware support for the two [24]. The TITAN V has the same theoretical performance for both precisions. The frame rate increase for the TITAN V is likely caused by the halved memory bandwidth needed in the beamforming process, while the larger improvement for the 1080 Ti is explained by the improved hardware throughput.

VI. CONCLUSION

Two row-column beamformers written in the MATLAB programming language and CUDA have been compared. The worst-case performance of the CUDA beamformer exceeds the best-case of the MATLAB beamformer across all GPUs. 3D synthetic aperture imaging is attained at 14 Hz for two cross-planes with a high-end scientific GPU (TITAN V) for a 192 emission sequence with a 192+192 array. This corresponds to a pulse-repetition frequency of 2.6 kHz rapidly approaching real-time 3D synthetic aperture imaging. Volumetric synthetic aperture imaging of a 50 \( \times \) 50 \( \times \) 20 mm\(^3\) box at one high-resolution volume every three seconds is attained. This volume easily covers e.g. the mitral valve that can be imaged throughout a single heart beat with no need for ECG gating or similar techniques in less than one minute of processing time with only 192 channels of data.

REFERENCES