Realtime plane-wave software beamforming with an iPhone

Cameron Lowell Palmer[1]
Department of Circulation
and Medical Imaging
Norwegian University of Science
and Technology
7491 Trondheim, Norway
Email: cameron.palmer@ntnu.no

Ole Marius Hoel Rindal[1],
Sverre Holm,
and Andreas Austeng
University of Oslo
Oslo, Norway
Email: omrindal@ifi.uio.no

***Abstract*—This work describes the implementation of software plane-wave beamforming performed on the GPU of an Apple iPhone 6s and 6s Plus. The code can run on any current iOS device that supports the Metal API. The implementation is largely written in Swift, with some Objective-C, while the core processing component was written in Metal, Apple's new GPU programming language which provides low-overhead compute shaders for exploiting the device's GPU. We have demonstrated that ultrasound channel data recorded on a Verasonics Vantage system can be wirelessly transmitted to the iPhone using a simple networking implementation obtaining a frame rate up to 5 FPS including serialization and transmission, and easily 60+ FPS for on device processing depending on number of samples and output image size.**

I. Introduction

Handheld devices and ultrasound scanners with wireless probes have in recent years received increased attention. The main bottleneck in a system with a wireless probe is to transfer the channel data to the device for processing. The traditional way of dealing with this bottleneck is to do some preprocessing, usually the first step in beamforming, on the probe to reduce the amount of data being transmitted. We have chosen to avoid adding this complexity in the probe, moving all work to the device, and leaving the probe to capture, serialize and transmit the raw channel data. When performing plane-wave beamforming; one full image can be created from a single transmission. These images are of a lower quality, but multiple images can be coherently compounded into images of high quality [1]. This allows a tradeoff between image quality and frame rate.

This work describes the implementation of plane-wave beamforming on an Apple iPhone 6s, managing realtime processing of PW images streaming channel data wirelessly from a Verasonics Vantage research scanner mimicking the probe. The plane-wave beamforming is implemented using Swift, Objective-C and the Metal GPU programming language. Thus we have shown that the processing power of a commercially available smartphone is enough to facilitate software beamforming of US images. This is a proof of concept that the probe in a handheld system may only need to contain hardware to acquire the channel data, while the beamforming can be done in software on commercially available devices.

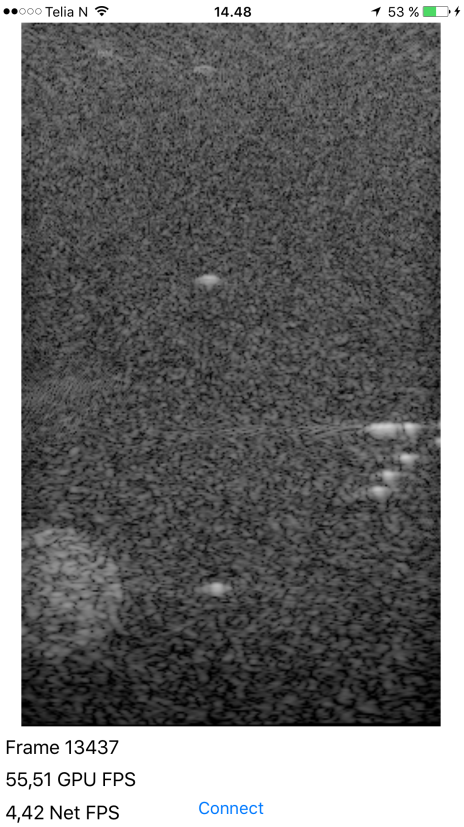

Fig. 1: Screen capture of an ultrasound image reconstructed on an iPhone 6s plus from channel data transmitted wireless to the phone from a Verasonics Vantage ultrasound scanner.

II. Materials and Methods

A. The iPhone 6s

The development target was an Apple iPhone 6s. Apple does not release detailed technical specifications of its products,

[1]The two first authors are equal contributors.

978-1-4673-9897-8/16/$31.00 ©2016 IEEE

however third-party sources have identified the following components;

- Dual Core 1.85GHz 64-bit ARMv8 processor
- ARM VFP and NEON SIMD extensions
- 4 MB (victim) L3 cache
- Dual-channel 64-bit LP-DDR4 SDRAM interface (25.6 GB/s)
- Samsung 2 GB LPDDR4 RAM
- 6-core PowerVR GT7600 GPU
- Broadcom BCM4350 802.11ac WiFi.

In benchmarks performed by Ars Technica [2] the iOS device family and Intel-based laptops have been shown to have similar performance. In some tests the iOS device can even outperform a laptop. Arguably the phone is a performance competitor with more traditional computing platforms and therefore it is reasonable to expect plane-wave beamforming to be possible on a wide range of mobile computing devices. The Apple 6s has a unified memory architecture, meaning that the GPU and the CPU access the same memory. This simplifies and speeds up the GPU processing significantly since there is no bandwidth bottleneck between the two blocks of memory.

B. The channel data

The data is recorded on a Verasonics Vantage research scanner [3] using the Verasonics L12-3v 192-element 0.2 mm pitch linear array. However, we are only using 128 elements since the connector is limited to 128 channels. We are transmitting one plane-wave at a 0° angle using a center frequency of 7.8 MHz and recording a 45.8 mm deep image. The data is IQ sampled using Verasonics' built in processing. This means that we are only transmitting one real and one imaginary sample for every wavelength received. Thus, when we are imaging at 45.8 mm depth we need to transmit a total of 128×512 IQ samples.

1) IQ data: The concept of In-phase Quadrature (IQ) data sampling is worth a brief explanation, for a more thorough description see [4]. When we have a bandlimited RF-signal centered around a center frequency, the IQ-signal can be obtained by down-mixing the signal. Down-mixing means multiplying the signal with a complex sinusoid signal given by a demodulation frequency f_{demod} "moving down" the signal in the frequency spectrum by using a negative demodulation frequency. This gives an asymmetrical and thus complex signal. This signal can then be low-pass filtered removing the negative frequency spectrum and noise outside the desired bandwidth. This lowpass-filtered signal can then be decimated reducing the number of samples by a integer factor, in our case 4 compared to the default Verasonics RF-sampling frequency. However, we need to keep in mind that the IQ-sample is complex, thus having both a real and an imaginary part.

C. Networking

For the wireless communcation we connected the phone and the Verasonics scanner through a dedicaded WiFi network. In order to pass structured IQ data frames from MATLAB across the network without implementing custom supporting code in MATLAB we used a Python script running Twisted on a PC which acted as an HTTP POST-to-WebSocket [5] bridge. On MATLAB we serialize the Verasonics IQ-data struct to JSON, optionally compress it, POST it to the Python script listener that in turn broadcasts the frame to all connected WebSocket clients.

```
{
    "identifier"   : x,
    "channel_data" : [[channel_1_samples],
                      [channel_2_samples],
                      [...],
                      [channel_n_samples]]
}
```

Fig. 2: JSON-serialized Channel Data consists of an arbitrary numeric identifier x and the sample values for each of the n channels. Message framing is handled by the WebSocket. Compression greatly reduces the size of the payload.

The native networking options for streaming structured data in MATLAB leave the developer wanting so our first-pass methodology [6] has the advantages of being easy to implement and easy to work with, but is inefficient in byte size and more importantly MATLAB's internal JSON serialization is slow. We also hit a bug in MATLAB 2015a for Windows that serialized arrays of integers to doubles which caused the size of serialized frame data to grow significantly (5MB per frame). Uncompressed JSON serialized channel data takes around 300kB per frame so a simple solution we used was to compress the data, but in our limited testing calling out to Python for this purpose slowed things down by a couple of milliseconds per frame.

D. Pixel-based Beamforming

The flexibility of software beamforming allows us to do so-called pixel based beamforming. This means that we can define a set of pixels, our image, and beamform the ultrasound data directly to the pixels. The first step in achieving this is to calculate the time delay for every signal received on every element to every pixel in the image.

1) Delay calculations: Given the transducer geometry, the element width and pitch, we can get the coordinates for our elements, x_n, and then define the coordinate for every pixel in the image (z, x). If we know the speed of sound in human tissue we can calculate the two-way time delay [1] by

$$\tau(z, x, x_n) = (z + \sqrt{z^2 + (x - x_n)^2})/c. \quad (1)$$

Graphically the delay calculation can be viewed as in Figure 3. We calculate each transducer element's contribution to each pixel in the image assuming a fixed aperture.

2) Linear Interpolation: The delay calculated in (1) can be a non-integer sample number. To get the correct value for every pixel we linear interpolate between the samples. Eg. if a certain pixel has the delay 15.5 for a sample from a certain element, we simply linear interpolate between sample 14 and 15 to get the correct value for that pixel.

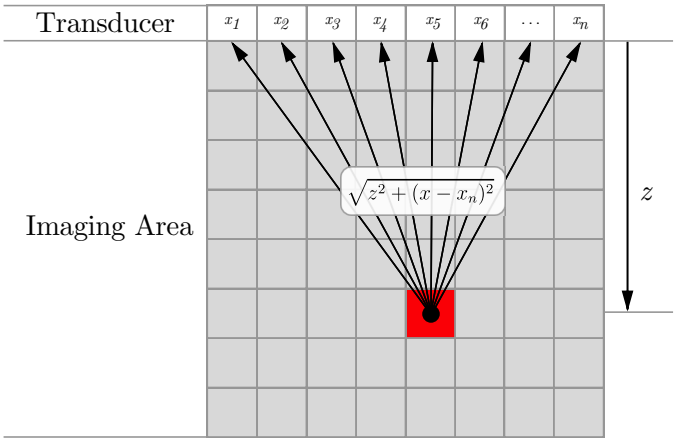

Fig. 3: The figure shows how each element in the transducer contributes to each pixel in the simple plane-wave case.

3) Reconstruction from IQ-data: In Section II-B1 we were introduced to IQ-data. When reconstructing from IQ-data we need to keep in mind that the IQ-data is the down-mixed version of the original signal, thus we have to up-mix the signal by doing the reverse of down-mixing and multiply the signal with a complex sinusoid using the positive demodulation frequency f_{demod}

$$x_{\text{RF}}(t) = x(t)_{\text{IQ}} e^{i2\pi f_{\text{demod}} t}. \tag{2}$$

4) Beamforming: For every pixel in the image, we now have a vector, $\boldsymbol{y}$, containing the correctly delayed signal from every element. The next step is then to sum the correctly delayed signal,

$$I_{DAS}[z,x] = \sum_{m=0}^{M-1} w_m y_m[z,x] = \boldsymbol{w}^H \boldsymbol{Y}[z,x]. \tag{3}$$

Here M is the number of elements, y_m is the correctly delayed signal from element m, and w_m is a predefined weight. When this is done for every pixel position x and z we get the full image.

5) Envelope detection: The up-mixed IQ-data is the one sided analytic signal, so the envelope can easily be detected by taking the absolute value of the complex value after the summation of each channel contribution.

6) Logarithmic Compression and Dynamic Range: The last step in our ultrasound image processing is to convert the raw image amplitudes to decibel values. This is achieved by applying

$$I_{\text{dB}} = 20 \cdot \log_{10}(|I_{\text{DAS}}|) \tag{4}$$

to each value, cutting off values that fall outside our desired dynamic range, for example 60 dB, and normalizing the values to the range of range of 0-255 graylevels to finally be displayed on the screen.

E. iPhone implementation

Our beamforming algorithm was originally prototyped in MATLAB and was subsequently ported several times as we worked to find the simplest implementation for iOS that achieved the desired performance. After an initially naive port to C and then in Swift, Apple's new systems programming language, we needed to move on to parallelized implementations. Several iterations of the parallelized code were worked through, adopting the Accelerate framework (a collection of hand tweaked high-performance libraries), and then finally settling on the Metal shader language. An advantage of using Metal is that since it is derived from C++ we can write, debug, and then copy-and-paste working code into the Metal shader file.

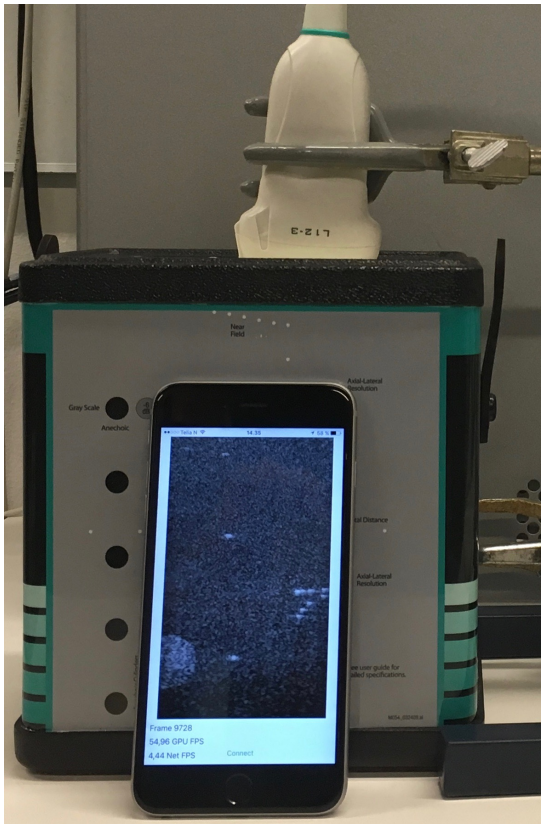

Fig. 4: Wireless Ultrasound Test.

For this version we are only beamforming individual plane-waves transmitted parallel to the linear probe's surface so we only need to calculate one set of delays for the defined grid of image pixels. This is calculated in the initialization of the streaming session and stored in memory, so the only processing occurring between two image frames is performed on the channel data captured from the ultrasound probe.

1) Channel Data Processing: The channel data is processed in the Metal compute shader in parallel across the GPU's cores. Each thread of execution calculates the amplitude value of a single pixel by summing the contribution of each channel to the pixel. This means for an $Z \times X$ number of pixels in an ultrasound image there are $Z \times X$ threads each summing the contribution of the M number of channels. If we were to add multiple angles for plane-wave compounding we would need to perform this calculation for each angle.

After summing the M channel contributions to the pixel we take the decibel value and store it.

III. Results

The software beamformer implemented is highly flexible but we settled for a setup with fixed aperture imaging a 25.4 mm wide and 45.8 mm deep image, giving us 512 IQ samples for the 128 elements to reconstruct the image from. The results are presented as frame rates obtained running the implementation described in the previous sections on an iPhone 6s Plus. We are presenting two different frame rates. One is the full cycle frame rate including the time for acquisition on the Verasonics Vantage scanner, serializing the data, transferring the data wirelessly to the phone, deserializing the data and finally processing it and displaying the final image, we will call this the Net FPS. The other is the frame rate obtained by only processing the data when it is stored on the phone, we will call this the GPU FPS. We have experimentally benchmarked this for three different numbers of pixels, using λ, $\lambda/2$ and $\approx \lambda/3$ as the pixel spacing. The pixel spacing $\approx \lambda/3$ was chosen since it matches the screen width of the iPhone 6s (375 units). The mean and standard deviation are reported in Table I.

	Net FPS	**GPU FPS**
129x217px λ	4.5 (σ=0.5)	142.0 (σ=5.9)
258x434px $\lambda/2$	4.5 (σ=0.2)	55.5 (σ=0.7)
375x632px $\lambda/3$	4.9 (σ=3.3)	29.4 (σ=1.1)

Table I: Experimental benchmarks for three different numbers of pixels, using λ, $\lambda/2$ and $\approx \lambda/3$ as the pixel spacing for the Net FPS and GPU FPS.

IV. Discussion

From the values in Table I we can see that the bottleneck in our current system is the transmission of data from the probe, the Verasonics Vantage computer, to the phone. We can draw this conclusion since the network FPS is stable as the number of pixels are increased and the GPU FPS is lowered. Using the MATLAB debugging tools we can show that most of our time is spent serializing and compressing the data before it is actually transmitted on the network. This tells us that the choice of using JSON will need to be revisited and switching to a binary protocol like Apache Thrift or Google Protocol Buffers with less computing overhead might significantly improve the performance of the system while maintaining simplicity of implementation.

Regarding the image quality, see Figure 1, it is as expected from a single plane-wave image with some visible noise from sidelobes and especially a noisy near field. Switching to dynamic aperture, constant $f\#$, should clear up some of the noise seen in the near field of the image. To significantly improve the image quality we need to compound plane-wave images from multiple angles. To be able to perform plane-wave compounded imaging and maintain realtime performance we will need to reduce or eliminate our MATLAB networking bottlenecks. The current performance of 55 FPS with $\lambda/2$ pixel spacing should be sufficient to allow us to trade some of our frame rate for increased image quality.

Another interesting approach could be to use a minimum variance approach to compound plane-wave images. It has earlier been shown that by using minimum variance compounding higher image quality is achieved using fewer plane-waves than for conventional coherent compounding [7]. This allows us to utilize the fact that the bottleneck is to transmit the data and not processing it on the phone.

V. Conclusion

We have shown that realtime plane-wave ultrasound imaging with an iPhone is feasible when processing wirelessly streamed channel data using a Verasonics Vantage ultrasound scanner and iPhone 6s at ≈ 5 FPS. The GPU processing frame rate is much higher measured at 55 FPS for a 25.5 mm wide and 45.5 mm deep image using $\lambda/2$ pixel spacing and can be viewed as the upper-bound of this implementation. The primary bottleneck in this implementation was identified as the serialization of the channel data into the JSON format.

Acknowledgment

The authors would like to thank Thomas Hansen, senior engineer at UiO for assisting in providing and setting up Apple products making this research possible.

After submitting this paper we discovered a paper [8] that demonstrated a commercial product at 2015 ICU Metz that obtained similar results on an iPad.

References

[1] G. Montaldo, M. Tanter, J. Bercoff, N. Benech, and M. Fink, "Coherent plane-wave compounding for very high frame rate ultrasonography and transient elastography," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 56, no. 3, pp. 489–506, 2009.

[2] A. Cunningham, "iPad Pro review: Mac-like speed with all the virtues and restrictions of ios," http://arstechnica.com/apple/2015/11/ipad-pro-review-mac-like-speed-with-all-the-virtues-and-limitations-of-ios/.

[3] Verasonics, Inc.,12016 115th Ave NE, Kirkland, WA 98034,USA, "Verasonics Vantage," http://www.verasonics.com.

[4] J. Proakis and D. Manolakis, *Digital Signal Processing*. Pearson Prentice Hall, 2007.

[5] "The WebSocket protocol," https://tools.ietf.org/html/rfc6455.

[6] "Sending JSON data over WebSocket from Matlab using python Twisted and Autobahn," http://stackoverflow.com/questions/34357973/sending-json-data-over-websocket-from-matlab-using-python-twisted-and-autobahn.

[7] A. Austeng, C.-I. C. Nilsen, A. C. Jensen, S. P. Nasholm, and S. Holm, "Coherent Plane-Wave Compounding and Minimum Variance Beamforming," in *2011 IEEE International Ultrasonics Symposium*. IEEE, oct 2011.

[8] H. J. Hewener and S. H. Tretbar, "Mobile Ultrasound Plane Wave Beamforming on iPhone or iPad using Metal- based GPU Processing," *Physics Procedia*, vol. 70, pp. 880–883, 2015.