Author's response to the reviewer's comments

Manuscript number:	2013JOE001295
Title:	An Optimized GPU Implementation of the MVDR Beamformer for Active Sonar Imaging
Authors:	Buskenes, J.I., Åsen, J.P., Nilsen, C.-I.C, Austeng, A.

We would like to thank the associate editor for his efforts and for his speedy replies to queries from the authors, and reviewers for their thorough and highly relevant comments. A summary of our answers are provided below, and subsequent sections will target each reviewer's comments in detail. We quote in **bold face** statements from the reviewers, and provide our replies in ordinary print,

1 Summary of response to reviewer's comments

2 Answer to reviewer 1

General

1. *Mpx/s is not a widely known abbreviation. This needs to be defined upon first use.*

   This is a good point. "Mpx" is now corrected to "MP", in accordance with the second reviewer's suggestion and the Oxford dictionary.

2. *"Matlab" should be all caps: "MATLAB" throughout the paper.*

   Thank you, we agree. Corrections applied.

Abstract

3. *A key attribute of the MVDR studied in this paper is the use of a subarray approximation. This should be stated up front in the abstract, i.e. "subarray MVDR" instead of MVDR.*

   The reviewer is right that subarray averaging is important in our study. But we believe that "subarray MVDR" would be misleading, as to the best of our knowledge this term is usually used to describe the use of subarrays to reduce complexity, while we average over subarrays to deal with the coherent echoes present in *active* systems. We still apply the weights to the full array. To avoid confusion we have now stated this better up front in the article (section I paragraph 5).

4. *This paper advertises an impressive ("more than two orders of magnitude) improvement over a C implementation of MVDR, but this is really misleading. In reality, an improvement of up to an order of magnitude is achieved over an optimized C implementation, as later stated in the caption for figure 8. A comparison of an optimized GPU implementation to an unoptimized C implementation is not very meaningful — both should be optimized. The abstract should be revised to indicate up to an order of magnitude over an optimized C implementation.*

   We agree with the reviewer that this should be improved. We still state the two order of magnitude improvement over our reference implementation as a "total payoff" for our efforts, but now also state that this figure would likely be 5-10 had the CPU side received similar treatment.

Section I

5. *Typo in second paragraph: "litterature" should be "literature".*

   Thank you. Fixed.

6. *Subarray processing is a common approximation and should be included in this list (reference D J Chapman, "Partial Adaptivity for the Large Array," IEEE Trans. Antennas & Prop., Vol. Ap-24, No. 5, September 1976, pp. 685-696).*

   We thank the reviewer for providing this reference. However, unlike the subarray

technique described by Chapman, which seems focused on reducing array complexity but in a way that causes grating lobe trouble, we use overlapping subarrays to (1) deal with the coherency problem in active sonar, and (2) to improve the rank of covariance matrix. The latter is particularly important in our case since we have hardly any temporal sample support. The Chapman reference, distinction and a reference to [9] Kailath and Shan, "Adaptive beamforming for coherent signals and interference" is now included in the introduction to hopefully avoid future confusion.

Section II

7. *Should reference the literature (from above comment 6) on the subarray approximation in paragraph 3.*

   We agree that that this paragraph lacks a reference, but feel that [9] Kailath and Shan "Adaptive beamforming for coherent signals and interference" better describes this method in the context of active sonar. The Chapman reference is added to the introduction instead.

Section III (A-C)

8. *Typo in first paragraph after figure: "got" should be "get".*

9. *Move "a" in "It is a Gauss Jordan based".*

10. *"which are a mere scaled version" should be "which are merely scaled versions".*

    We thank the reviewer for these corrections (8.-10.). They are all applied.

Section IV

11. *The sub-plots on the top break the processing out into 3 pieces, yet the lower sub-plots reflect the entire processing string for MATLAB and C. It would be useful to show the entire processing time for the GPU for better comparison. Perhaps breaking the two rows of sub-plots into two figures would work better.*

    The reviewer has a point. But since there are 3 occurrences of the covariance processing time, including the total time GPU processing time would introduce three new lines. We feel this would clutter the plot unnecessarily as it is fairly easy derive the total time by looking at the current plots. Putting the total processing time in the MATLAB/C total time figure is an option, but then a logarithmic y-scale would necessary and detail would be lost.

12. *Comment about real processing improvement over optimized C should be related in the text instead of a caption. This is an important point about the comparisons made in the paper.*

    It is indeed. We dedicated the second paragraph in the discussion (IV) to this

topic, but will reword the relevant result paragraph slightly to hopefully cause less confusion.

13. *(caption of figure 10) "cruel" should be "crude"*

    Thank you, crude grammar correction applied.

14. *The point made in the first paragraph about the data transfer from GPU to CPU being ignored further muddies the water in comparing the two approaches. This point should also be stated up front. I would suggest making a very clear itemization of the assumptions and caveats about the two approaches.*

    We agree that this should hould be clarified, and have We disagree that this

Section V

15. *The reason the C implementation is linear in L is because it is dominated by the calculation of the covariance matrix and data movement. The inversion is negligible here. On the other hand, the GPU optimizes the covariance calculation so much that now the inversion is significant, hence the non-linear computation time as a function of L. This is really the key point to be made in the comparison between the two approaches, in my opinion.*

    Yes! This is likely the reason. Thank you for pointing this out. This is now added to the discussion.

16. *Bottom of page, "hold-ups" has an extra space after the hyphen.*

    Fixed.

Section V

17. *Two spaces before "1 Mpx/s" in 4th paragraph.*

    Thank you, corrected.

3 Answer to reviewer 2

The authors describe their work on implementing an MVDR adaptive beamformer for a high-frequency imaging sonar on a particular graphical processing unit (GPU), the prevous-generation NVIDIA Quadro 6000, using NVIDIA's CUDA C environment. The authors report 1-2 orders of magnitude faster processing with their GPU implementation, compared to their MATLAB and single-thread, unoptimized CPU implementations, without taking the input/output of data in and out of the GPU.

Because the paper is a valuable result to imaging sonar users, who will (to some extent) know that they can speed up their processing by 1-2 orders of magnitude if they use a GPU, I think the paper should be published.

However, I think the authors could greatly improve the appeal of the paper to a wider audience by adding helpful context and background on their implementation. I will suggest some questions that I think the authors could easily address toward this end. This is a report from the "front lines" of migrating scientific computing to GPUs, a topic which seems to be of great interest to everybody in our field that does numerical computing. In my review, I will comment on the information it provides to a reader interested in starting to use GPUs (and the authors for some selective elaboration).

Suggestions for improving the exposition of the processing

I suspect many readers of the paper will not have the same assumptions as the authors, if they are not processing high-frequency narrowband signals with undersampled arrays. I see this disconnect between low-frequency passive processors and mid-frequency active processors, but the gap to imaging sonars is even greater. I venture that the authors can greatly broaden the appeal of their paper by addressing the questions I pose below.

We thank the reviewer for these remarks, it seems we should state this better upfront. In the 5th paragraph of the introduction we now explain that our system is a well sampled wideband sonar.

1. *Here are my calculations of how the 32 elements are spaced in the HISAS 1030 system:*

$$fc = 100 \text{ kHz} \tag{1}$$
$$c = 1500 \text{ m/s} \tag{2}$$
$$\lambda = c/f_c = 15 \text{ mm} \tag{3}$$
$$d = \lambda/2 \tag{4}$$
$$D = 1200 \text{ mm} \tag{5}$$
$$M = D/d = 160 \tag{6}$$

The reviewer made us realize that this was not clear enough in the article. The 32

elements are evenly spaced along the 1.2m array, so:

$$M = 32 \tag{7}$$

$$d = D/M = 3.75 \text{ cm} = 2.5\lambda \tag{8}$$

Further details are provided in 2. A sentence elaborating on the element spacing has been added in section IV first paragraph; where the HISAS1030 is introduced.

2. *What is impact of using only 32 elements? How are they spaced? It would seem that spatial aliasing would be an issue. Authors should explain what their assumptions are.*

   Appreciations for pointing this out. We agree that this should be clarified. The array configuration produce the first grating lobes at $\theta = \arcsin(\lambda M/D) = 23.5°$, but this is not a problem since the element opening angle is $25°$ ($\theta \in [12.5°, 12.5°]$) at $f_c = 100$ kHz. In the paper we addition we perform sidescan imaging where only very narrow beams need to be beamformed. These details are now included in the paper, some in the 5th paragraph of the introduction and some in the first paragraph of section IV.

3. *What is bandwidth of transmitted waveform? At 100 kHz, perhaps the authors are comfortably in a narrowband regime, and they do not need to worry about array response changing over their bandwidth (which would suggest that MVDR would need to be performed at different frequencies independently, unless some sort of pre-steering was used, which may be what the authors are doing anyway). Authors should explain what their assumptions are.*

   We mention in the beginning of section IIA that we pre-steer to every angle and range bin in the image, but now repeat this in IIA paragraph 3, and state that we operate in wideband conditions in the introduction.

4. *Authors show solution of MVDR optimization as* $\boldsymbol{w} = \frac{\boldsymbol{R}^{-1}\mathbf{1}}{\mathbf{1}^T\boldsymbol{R}^{-1}\mathbf{1}}$ *and state this is the broadside solution. Are the authors pre-steering the array for every angle? Or are they using* $\boldsymbol{w} = \frac{\boldsymbol{R}^{-1}\boldsymbol{s}}{\boldsymbol{s}^T\boldsymbol{R}^{-1}\boldsymbol{s}}$ *at off-broadside angles, with* $\boldsymbol{s}$ *the conventional steering vector (which would be a frequency-domain phase-shift vector, to be applied to a narrowband signal)? Pre-steering would enable disparate frequencies over a wide band to be coherently summed, to provide coherent gain in the look direction (over the band) by providing additional sample support for the sample covariance estimation.*

   Yes, we are pre-steering the array for every angle. We have clarified this further (see 3.), and added a remark saying that while not necessary here we could have phase steered the array by introducing the steering vector $\boldsymbol{s}$.

5. *Authors time average* $\boldsymbol{x}\boldsymbol{x}^H$ *over* $K \in \{0, 1, 2\}$ *time samples. Without doing some sort of reduced-rank processing, why is as little as 2 samples enough to estimate the covariance? Perhaps this is due to high SNR? At high SNR, though, why are*

the authors doing MVDR at all? What interference is being canceled, and what is its spatial distribution and stationarity?

Again, a helpful remark, we will clarify this as well. After pulse compressing a chirp using the full transceiver bandwidth (30 kHz) range resolution is roughly $\Delta r \approx \frac{1}{B} = 2.5$ cm, which corresponds to a sample shift of just 1.2[1]. Hence we cover the entire pulse compressed signal with only a few samples. $K = 2$ corresponds to a window length of $2K + 1 = 5$ samples, as is shown by (4) in the article. Some edits have been made to make this clearer.

6. *I read the first paragraph of Section II, "..principle of adjusting the array's focus in range and bearing is commonly referred to as beamforming", and was confused - were the authors doing wavefront curvature beamforming, where every combination of range bin and beam angle required a distinct set of steering weights? After reading further, I would venture that the authors are steering only in angle, and that the index n is over time of arrival, at the resolution of the width in time of the matched filter output (after pulse compression... the authors do not say anything about the waveform, so I have no idea whether there WAS any pulse compression). What is the resolution in time or range?*

   We understand how the confusion might arise. We operate under near field conditions and are indeed performing wavefront curvature beamforming. Hence, the index n just an index over time at a resolution given by the sampling frequency. This has now been clarified.

3.1 Suggestions for additional GPU details so that paper can also serve as "field report" on CPU work in general

1. *It is unfortunate that so little effort was made to speed up the CPU versions of the code. For example, there are public-domain versions of optimized LAPACK libraries that provide the same sort of linear algebra codes as the authors used in their CPU implementation (for example, GoToBLAS, or Intel's MKL libraries).*

   We agree, and have now included our reasons for not comparing with an optimized CPU implementation. One reason is that we have not been able to find any CPU batch solvers out there and lacked the time to make our own. In our tests our custom inline Cholesky solver beat alternatives from e.g. MKL as our matrices are too small to justify the extra function call overhead. Another reason is that to make the comparison fair we feel an equal effort would have to put into writing a covariance builder for the CPU, and to minimize data movement in each step and in between them. The details may be found in section V paragraph 2.

2. *The extra input/output of data to the GPU is a burden only on the GPU implementation, so would penalize the GPU implementation only, unless the authors*

[1] $\Delta n = \frac{\Delta r f_s}{c} = 1.2$

were to "hide" the i/o by setting it up to run in parallel with the CPU processing, which is easier and easier with the latest GPU architectures. This should perhaps be pointed out.

Yes this is a good point. Included in last paragraph of section IV.

3. *The authors do not provide much information about the GPU hardware they used, the NVIDIA Quadro 6000. How does this compare to the desktop video cards that reside in most desktop, workstation, server and even high-end laptop systems? What is the vintage of the architecture of this GPU? How has the architecture evolved? For example, the authors tell us the Quadro 6000 ($1900, 6GB VRAM, 448 cores) is a Fermi architecture - this is the previous architecture, having been superceded by the Kepler architecture, with a new Quadro K6000 board ($3600, 12 GB VRAM, 2880 cores). The Quadro boards are high-end boards with video output targeting the CADAM market - there are many cheaper alternatives targeted at gamers and an even more expensive compute-only board called TESLA targeted at high performance computing. It's difficult to say how much of this irrelevant.*

   There is certainly a potential for comparing with many other cards here, but we decided to be thorough with one architecture than to compare with everything. To the paragraph introducing the Quadro 6000 we have explained what its equivalent consumer grade Geforce card would be. We have also included remarks that we expect Kepler to perform better had the design been optimized for it.

4. *The authors state that OpenCL is an equally attractive alternative to CUDA. This would be the case if the authors had been able to find something in OpenCL akin to the recently released batch-optimized linear solver code for CUDA. Unfortunately, this is a pervasive story - the support for scientific programming in CUDA is years ahead of OpenCL. The benefit of OpenCL is that it is supposed to be "cross-platform", which is true in theory, but it is hard to imagine being true in practice considering the wide range of heterogeneous computing elements OpenCL is supposed to cover; from phones to DSP chips to FPGAs.*

   These are important remarks, thank you. We have written OpenCL programs too, and run them on Intel CPUs and GPUs from AMD and Nvidia. It was no walk in the park, but with a satisfactory end result. But like you say, these are just a subset of devices that OpenCL support, and we have now changed the wording to be more specific here. Your remark on the lack a batched linear solver is spot in, the sentence is now rewritten.

3.2 Not clear how R is being calculated in CPU

Page 5, last paragraph in 1st column is puzzling. Authors say that "when a thread has finished up a diagonal, we have them wrap around to compute one of the diagonals in the lower triangular of R." The diagonals are all of different lengths, and, presumably the code to do a copy and conjugation from the upper triangular part to the lower triangular

part of R requires different instructions than the moving average being calculated along a diagonal in the upper triangular part. The cores MUST execute the exact same code; or "diverge" (with one set executing one set of code while the complementary set sitting idle, then vice versa, as you would get with two if-then-else code sections), since this is a SIMD architecture. What is going on here?

This requires a better explanation.

The reviewer is absolutely right. The copy will be serialized due to divergent branching. It is unfortunate that no Cholesky based batch linear equation solver exist, which would have eliminated the need for the lower triangular... The paragraph has been edited to explain this better.

3.3 Opportunities for additional GPU tutorial content?

The authors describe the methodology they used to get things to run fast on the GPU. The analysis of the workload was informative and I think it is very informative for readers to see how the analysis progresses from less hardware specific (operations counts), to algorithm restructuring to reduce operations; to more hardware specific concerns (relative throughput of operations and memory transfers).

We thank the reviewer for the credit.

Let me suggest some concepts that could be discussed a little bit more to give readers more of a sense of what programming GPUs is like.

1. *What does somebody need to program in CUDA? Authors can mention that CUDA C is just C with a few extra language features to specify how to distribute the work across the cores. Authors can mention that there is a nvcc compiler, a profiler, a debugger, math libraries (CUFFT, CUBLAS), and a large number of examples of varying complexity, all from NVIDIA. Authors can mention that CUDA can be used on Windows, MacOSX, or Linux, with an NVIDIA graphics card. Authors can mention that there are also third party CUDA interfaces in Fortran and Python. When discussing the batch-optimized linear solver code from NVIDIA, the authors should cite the NVIDIA site, and also links to CULA and MAGMA, other CUDA versions of LAPACK or subsets thereof.*

   These are good suggestions. We have added some of these suggestions to the paragraph where we introduce CUDA, but regrettably had to leave some of it out due to space considerations. Citation to the "Nvidia Registered Developers Program" is added where we speak of the batch solver.

2. *Problems need to have high "arithmetic intensity" (the authors use "computational intensity") so that the cores stay busy. If there is not enough arithmetic, the cores sit idle waiting for data to be moved around. The authors do a nice job describing this situation on page 4 and the start of page 5.*

   We thank the reviewer for those kind words. In the mentioned context we agree

that "arithmetic" is likely a better alternative to "computational", and will make the relevant edits.

3. *The authors use the term "arithmetic reduction". Unfortunately, reduction is an important word/topic in GPU work. Perhaps "minimizing arithmetic operations" would be better. Reduction is any N-to-1 mapping, which poses a conundrum for N cores having to combine their individual results at a single location all at the same time... you don't want the cores to all line up and wait their turn.*

   Yet again, very true! The reviewer's suggestion is much better, we have used it.

4. *On page 4, the authors talk about avoiding global memory, without getting into "coalesced" memory access patterns. This might merit a mention - for example, is at best only able to move one float for every 30 floats processed, and potentially much worse if care is not taken to use "coalesced memory access patterns (that avoid memory bank conflicts)". On page 5, the authors talk about "a collective access pattern that maximizes global memory bandwidth". Here too, the authors should perhaps use the "coalesced" term, since this is the term used in all of the GPGPU documentation.*

   The reviewer is right. It is better to use the "official" terms. We have applied the suggestions.

3.4 Localized comments

1. *Equation (2) using a "broadside steering vector" - this is confusing... are the authors only steering at broadside? clearly not... do the authors pre-steer all of the elements prior to forming* $\boldsymbol{R}$ *so that can "focus" in a wideband sense?*

   We indeed presteer all the elements. To make this clearer we have now stated it again after equation (2), and explained that a generalization is straight forward by substituting $\mathbf{1}$ with a steering vector.

2. *Paragraph after equation (2) mentions "spatial averaging to avoid signal cancellation" - this problem might be better identified for readers by citing the Kailath reference and using "signal cancellation due to coherent multipath".*

   Good point, both suggestions are now included.

3. *Equation 4 has second summation using index* n'*, but this index does not appear in any of the terms inside the summation.*

   Good catch! Fixed.

4. *Page 2 -* $\boldsymbol{R}$ *is averaged over K time samples... at what rate is beamformer output produced? is* $\boldsymbol{R}$ *averaged using a moving window process, so that* $\boldsymbol{R}[n]$ *is just a rank-1 update to* $\boldsymbol{R}[n - 1]$ *and there is one beamformer output for every input time sample?*

Actually, $\boldsymbol{R}$ is averaged over $2K + 1$ time samples, this can be seen from (4) and in the text that follows. There is indeed one beamformer output for every input time sample. We are not sure where the article is indicating otherwise...

5. *Page 4, in list of three optimizations - item 3 is not self-evident, but if I understand what the authors are doing, it is described by the 1985 reference by Kailath.*

   We must have conveyed this in a flawed way. We are simply implementing a text-book sliding window summation, where a start sum is computed and then this is updated with a new term for each element in $\hat{\boldsymbol{R}}$. Item 3 has now been rewritten to hopefully express this better.

6. *Page 4, last paragraph of 1) Arithmetic reduction - authors should identify that the "Minimized memory" version skips step 2) from list of three optimization, and that the "Minimized instructions" version uses all three steps... I had a hard time understanding how the three "arithmetic reduction steps" were related to the plots in Figure 5, and was puzzled about where the "minimized memory" and "minimized instruction" variants of the processing had come from. Perhaps the authors can introduce these variants more clearly and more prominently.*

   The reviewer is right. This is now rewritten to be more explicit.

7. *On page 6, authors say "The bottleneck in the final design is now the inversion step, which is typically 5 times slower than the build step". Surely, this merits an explanation. What is the final design being compared with? What was the expectation? What changed? Figure 3 shows FLOPs before optimization - this shows build and inversion steps of roughly the same order of magnitude in the right half of the figure. Does Figure 3 set the expectation? Figure 5 shows "relative complexity reduction" of entire pro-cess, so there is no information on relative speed of build and inversion steps. Figure 10 shows estimated FLOPs for the two steps, but since both steps have different theoretical complexities (operations counts), and the two steps do not optimize the same way, this does not provide an expectation of the relative times.*

   TODO

8. *On page 6, authors say their implementation was improved by factor of 1.5 - 2 when run on a K20. Authors could say a few words about what a K20 is, how it differs from the Quadro 6000, that the Kepler architecture actually has features that are dramatically better for scientific computing if the code is adapted to use them (kernels can call kernels, more registers, more shared memory, ability to eliminate some host-to-GPU i/o).*

   Good suggestions. They are now added to the last paragraph of section V.

9. *Section I says 1-2 orders of magnitude. Section IV, page 6, says 2-3 orders of magnitude.*

Well seen. This is now corrected.

10. *Reference 9 has two authors, Shan and Kailath. 11. Reference 17 seems to be the reviewed manuscript?*

    Reference 9 has been corrected, and reference 17 (now 18) has been updated to point to the correct article.

4 Changes to figures

1. Fig. 3 - Slightly increased spacing between subplots.
2. Fig. 8 - Abbreviating megapixels as "MP" instead of "MPx".
3. Fig. 9 - Abbreviating megapixels as "MP" instead of "MPx".
4. Fig. 11 - Abbreviating megapixels as "MP" instead of "MPx".