Jon Petter Helgesen Åsen

Accelerating adaptive ultrasound imaging algorithms by means of general-purpose computing on graphics processing units

Thesis for the degree of Philosophiae Doctor

Trondheim, September 2014

Norwegian University of Science and Technology
Faculty of Medicine
Department of Circulation and Medical Imaging

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Medicine
Department of Circulation and Medical Imaging

ISBN 978-82-326-0448-7 (printed ver.)
ISBN 978-82-326-0449-4 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2014:268

Printed by NTNU-trykk

Hurtig beregning av adaptive algoritmer for ultralydavbildning ved hjelp av programmerbare skjermkort

De siste årene har det vært en rivende utvikling innen bruken av skjermkort (GPU-er), som egentlig er laget for å kjøre dataspill, til å gjøre numeriske beregninger for naturvitenskapelige anvendelser. Et sistegenerasjons skjermkort har nå like mye beregningskraft som verdens kraftigste datamaskin hadde rundt år 2000. Utfordringen er at problemet som skal løses må bestå av mange uavhengige deler som kan beregnes i parallell. For ultralydmaskiner har denne utviklingen gjort at algoritmer som tideligere krevde spesialhardware, nå kan programmeres i software og kjøres i sanntid på et rimelig skjermkort. Et ultralydsystem som bruker skjermkort til å gjøre beregninger vil være mye rimeligere, mer fleksibelt og gjøre det lettere å implementere avansert prosessering enn et system hvor alle algoritmer er og må "hugges i stein".

Målet med denne studien har vært å utforske hvordan programmerbare skjermkort kan utnyttes til avansert prosessering i et avbildende ultralydsystem. Blant de problemene vi har sett nærmere på finner man både adaptiv bildedannelse (stråleforming), adaptiv visualisering av 3D ultralyd og ultralydsimuleringer. Felles for disse problemene er at de krever parallellprogrammering for å gå i sann tid.

I første del av avhandlingen utforskes Capons adaptive stråleformingsmetode. Vi starter med å implementere metoden på et skjermkort for sanntidsprosessering av sonardata (**Artikkel I**) og av data fra medisinsk ultralyd (**Artikkel II**). Sanntidsprosessering oppnås for begge modaliteter. **Artikkel II** presenterer også, for første gang, videoer hvor både simulerte og *in vivo* medisinsk ultralyddata er prosessert med Capon-stråleforming. I **Artikkel III** viser vi at Capon-stråleformeren er følsom for små skift, enten ved at ultralydproben eller objektet man avbilder beveger seg. Videre foreslås det en metode som gjør at stråleformeren ikke lenger er like følsom for slike forflytninger. Dette er viktig å få på plass for at Capon-stråleformeren skal kunne tas i bruk i praksis.

I **Artikkel IV** foreslår vi en adaptiv metode for visualisering av 3D ultralydbilder av hjertet. Metoden undertrykker støy som med konvensjonelle metoder hadde skygget for hjertevev. Dette arbeidet viser at det med moderne skjermkort er mulig å utføre avansert visualisering i et ultralydsystem og fortsatt ha sanntidsytelse.

I siste del av avhandlingen utforsker vi hvordan skjermkort kan utnyttes til å akselerere ultralydsimuleringer. Resultatet av dette arbeidet ble et simuleringsprogram hvor geometrien til et gitt ultralydarray kan tegnes interaktivt, og hvor det resulterende trykkfeltet simuleres og visualiseres i sanntid.

Jon Petter Helgesen Åsen
Institutt for sirkulasjon og bildediagnostikk, NTNU
Hovedveileder: Sverre Holm
Biveiledere: Stein-Inge Rabben, Erik Steen and Hans Torp
Finansieringskilde: MI-Lab

Ovennevnte avhandling er funnet verdig til å forsvares offentlig for graden Philosophiae Doctor (Ph.D.) i Medisinsk Teknologi. Disputas finner sted i Auditoriet, Medisinsk teknisk forskningssenter, torsdag 9. oktober 2014 kl. 12.15.

Abstract

A rapid development in computer game technology and accompanying programming languages have recently provided researchers with small personal supercomputers, comprised in a single graphics processing unit (GPU). This immense rise in computational capabilities and improved programmability are currently changing how ultrasound imaging systems are designed. When researchers are exploring new algorithms for ultrasound imaging, it is therefore important to keep the architecture of parallel accelerators like the GPU in mind. If a new complex algorithm is supposed to run in real time, it needs to fit the programmable and parallel pipeline of modern ultrasound scanners.

The aim of this study has been to investigate the possibility of utilizing GPUs for advanced processing in an ultrasound imaging system. Among the investigated problems are both adaptive beamforming, adaptive visualization of ultrasound volumes, and ultrasound simulations. The presented problems have in common that they require parallel programming in order to reach real-time processing.

In the first part of the thesis, the Capon adaptive beamformer is investigated and implemented on a GPU for the application of real-time sonar (**Paper I**) and medical ultrasound imaging (**Paper II**). Real-time frame rates are achieved for both modalities. **Paper II** also presents, for the first time, videos where the Capon beamformer has been applied on loops of simulated and *in vivo* medical ultrasound images. In **Paper III**, we show that Capon beamforming does not provide shift-invariant imaging in a real-time imaging setting. A method is then proposed that improves the shift-invariant property. Shift-invariant imaging is essential if the method is ever to be used in practice.

In paper **Paper IV** we propose an adaptive method for visualization of volumetric cardiac ultrasound images. The method is capable of removing noise that by conventional methods would have occluded cardiac tissue. This work also shows that with modern GPUs it is possible to add advanced visualization methods to an ultrasound imaging system and still have real-time performance.

Finally, we investigate how GPUs can be utilized to accelerate ultrasound simulations (**Paper V**). The result of this work was a simulation program where ultrasound array geometries can be interactively drawn and where the resulting pressure field is simulated and visualized in real time.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* (Ph.D.) at the Faculty of Medicine at the Norwegian University of Science and Technology (NTNU). The research was funded by *Medical Imaging Laboratory* (MI-Lab), and was carried out at the Department of Circulation and Medical Imaging. The main supervisor has been Professor Sverre Holm in the Department of Informatics (IFI), University of Oslo (UiO), and Professor II at MI-Lab, NTNU. Co-supervisors have been Ph.D. Stein Inge Rabben, Ph.D. Erik Steen, both at GE Vingmed Ultrasound AS, and Professor Hans Torp in the Department of Circulation and Medical Imaging, NTNU.

Acknowledgments

First of all I would like to thank my main supervisor Sverre Holm and the rest of my team of supervisors for all the help and guidance through the Ph.D. project. Andreas Austeng also deserves a big thank you for all valuable help and for showing so much interest in my work. Thank you to all my co-authors for their contributions. I also want to thank the Faculty of Medicine, MI-Lab and the Department of Circulation and Medical Imaging, NTNU for believing in me and making it possible to conduct this work.

During these years I have had the great pleasure of working with and getting to know several research communities and companies. I therefore would like to thank all colleagues at the Department of Circulation and Medical Imaging, NTNU, the Department of Informatics, UiO, GE Vingmed Ultrasound AS, and Squarehead Technology AS for providing great working environments. A special thanks goes to all my fellow Ph.D. students at MI-Lab. I will never forget our conference trips. A special thanks also goes to Jo Inge Buskenes in "office" 4470 at IFI, UiO for making every working hour so much fun.

Ines Hafizovic and Carl-Inge Colombo Nilsen have used their spare time to review this thesis, for which I am very grateful.

Finally I would like to thank my family and all my friends for being there, and for providing a world away from work. But most of all, I want to thank Beate for her love and patience. Without your support I would never have managed to complete this thesis.

Dedicated to Sofie Mathilde Åsen

List of publications

This thesis is based on the following five papers, referred to in the text by their Roman numerals (I-V):

I J. I. Buskenes, **J. P. Åsen**, C.-I. C. Nilsen and A. Austeng, "An optimized GPU implementation of the MVDR beamformer for active sonar imaging", *IEEE Transactions on Oceanic Engineering, Jul 2014.*

II **J. P. Åsen**, J. I. Buskenes, C.-I. C. Nilsen, A. Austeng and S. Holm, "Implementing Capon beamforming on a GPU for real-time cardiac ultrasound imaging", *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 61, no. 1, pp. 76-85, Jan 2014.*

III **J. P. Åsen**, A. Austeng and S. Holm, "Capon beamforming and moving objects - An analysis of lateral shift-invariance", *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 61, no. 7, pp. 1152-1160, Jul 2014.*

IV **J. P. Åsen**, E. Steen, G. Kiss, A. Thorstensen and S. I. Rabben, "Adaptive volume rendering of cardiac 3D ultrasound images - utilizing blood pool statistics", *Proc. SPIE Medical Imaging 2012, vol. 8320, pp. 832008.*

V **J. P. Åsen** and S. Holm, "Huygens on speed: Interactive simulation of ultrasound pressure fields", *Proc. IEEE Ultrasonics Symposium 2012, pp. 1643-1646.*

Related publications

The following papers are related to the included papers, and are referred to in the text by their Roman numerals:

VI **J. P. Åsen**, J. I. Buskenes, C.-I. C. Nilsen, A. Austeng and S. Holm, "Implementing Capon Beamforming on the GPU for Real-Time Cardiac Ultrasound Imaging", *Proc. IEEE Ultrasonics Symposium 2012, pp. 2133-2136.*

VII J. I. Buskenes, **J. P. Åsen**, C.-I. C. Nilsen and A. Austeng, "Adapting the Minimum variance beamformer to a graphics processing unit for active sonar imaging systems", *Proc. Meetings on Acoustics 2013, vol. 13. Invited talk.*

VIII G. Kiss, E. Steen, **J. P. Åsen** and H. Torp, "GPU volume rendering in 3D echocardiography: real-time pre-processing and ray-casting", *Proc. IEEE Ultrasonics Symposium 2010, pp. 193-196.*

IX E. S. Brønstad, **J. P. Åsen**, H. Torp and G. Kiss, "Visibility driven visualization of 3D cardiac ultrasound data on the GPU", *Proc. IEEE Ultrasonics Symposium 2012, pp. 2651-2654.*

Contents

Symbols and Abbreviations

$(.)^H$	Complex conjugate transpose
$\delta\theta$	Angular resolution of a two-way imaging system
$\delta\theta_a$	Angular resolution of an aperture
Δ_m	Per-element focusing and steering delays
λ	Wavelength
$\hat{\mathbf{R}}$	Sample covariance matrix (with regularization)
τ	Opacity threshold
$\mathbf{w}$	Array weight vector (apodization)
B_w	Relative bandwidth
c	Speed of sound
D	Aperture size
D_{rx}	Aperture size on receive
D_{tx}	Aperture size on transmit
f	Frequency
f_c	Center frequency of ultrasound pulse
K	Temporal averaging parameter (Capon)
L	Subarray length (Capon)
M	Number of array elements
N	Number of samples in range (or time)
N_θ	Number of lines required for proper sampling of an imaging sector
N_b	Number of selected beams in beamspace (BS-Capon)

N_K Number of samples used for temporal averaging (Capon)

N_L Number of subarrays used for spatial averaging (Capon)

r Imaging range.

APU Accelerated processing unit

ARM Designer of mobile chip architectures

ASIC Application-specific integrated circuit

AVX Advanced vector extensions (instruction set)

BS-Capon Beamspace capon beamforming

C++ AMP C++ Accelerated Massive Parallelism (programming model)

CPU Central processing unit

CUDA Compute unified device architecture (GPGPU framework)

DAS Delay-and-sum beamforming

ES-Capon Element space capon beamforming

FMA Fused multiply-add (instruction)

FPGA Field-programmable gate array (processor)

FPS Frames per second

GPGPU General-purpose computing on graphics processing units

GPU Graphics processing unit

IQ In-phase quadrature (ultrasound data format).

LCA Low complexity adaptive (beamformer)

MV Minimum variance beamformer (Synonym for Capon)

MVDR Minimum variance distortionless response beamformer (Synonym for Capon)

O(.) Big O notation (algorithm complexity)

OpenCL Open Compute Language (GPGPU framework)

OTF Opacity transfer function (volume rendering)

rx Receive

SFU Special function unit (sin, exp, sqrt, ect.)

SIMD	Single instruction multiple data (instruction type)
SIMT	Single instruction multiple threads
SM	Streaming multiprocessor (SIMD core on Nvidia GPUs)
SNR	Signal-to-noise ratio
sonar	SOund Navigation And Ranging (usually under water)
SSE	Streaming SIMD extensions (instruction set)
tx	Transmit

Chapter 1

Background

This chapter gives the background information needed in order to understand the topics of this thesis and the associated papers. First, in Section 1.1, the history and the principles of general-purpose computing on graphics processing units (GPGPU) are reviewed, followed by an introduction to medical ultrasound imaging and adaptive beamforming in Section 1.2. Finally, the concepts of volume rendering and ultrasound field simulations are presented in Sections 1.3 and 1.4.

1.1 General-purpose computing on graphics processing units

Herb Sutter, a leading authority on software development, reviews the current state of the software and computer industry in a recent essay [1]. The essay bears the name "Welcome to the Jungle" and is a sequel to the essay "The Free Lunch is Over" from 2004 [2]. The two essays spin around the fact that manufacturers of central processing units (CPUs) hit a frequency wall in the beginning of the 21st century. Until then, all mainstream computer programs were typically running in a single thread on a single CPU *processing core*, and a programmer could expect the software to annually gain performance without touching the code. This was the "free-lunch" era as depicted in Fig. 1.1. The increase in processing power came mostly from an ever-growing clock frequency. However, power consumption and heat generation were also growing until the required cooling finally became too much in 2004. From that point, CPU manufactures had to look at different strategies than just increasing the clock frequency. One strategy was to embed several cores in one CPU. This ensured continuation of Moore's law, since the number of transistors per chip (CPU) could continue to grow. For programmers, multi-core CPUs meant that computer programs now had to be multi-threaded in order to annually gain performance. It was a concurrency revolution, as Herb Sutter named it, and the "free lunch" was over.

Around the same time as CPU manufacturers hit the frequency wall, people had started experimenting with programmable shaders in the field of graphics programming. The graphics processing pipeline had before programmable shaders

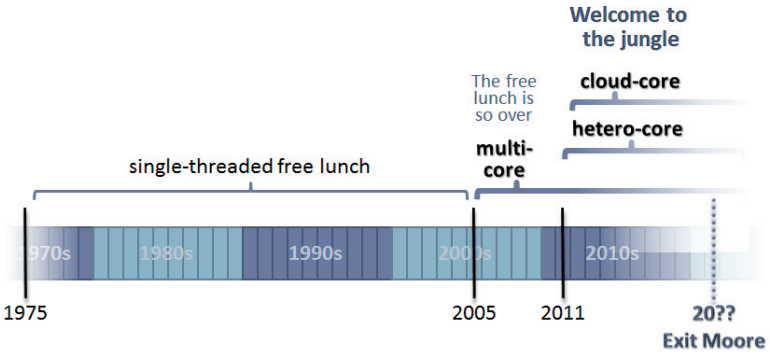


Figure 1.1: Illustration of transitions between different eras in the computer industry. Currently there are three ongoing transitions; multi-core, heterogeneous-core and cloud-core (herbsutter.com).

been a fixed function pipeline, where geometry and colors were converted into shaded and rasterized sceneries by the graphics processing unit (GPU). Programmable shaders introduced the ability to replace some steps in the fixed-function pipeline with custom computations. This made it possible to utilize the GPU for other tasks than just rendering graphics [3]. The rationale behind this development was that GPUs were already available in consumer PCs, and hidden behind its graphics interface were processing capabilities an order of magnitude larger than what the CPU could provide (Fig. 1.2). Actually, at the time when CPU manufacturers hit the frequency wall, the GPU had already gone multi-core, first of all driven by an ever-increasing demand for more realistic computer games.

The original graphics pipeline involved no conditional branching (caused by if-else control flow statements), and data were typically used once [4]. Designers of GPUs could therefore skip advanced features found in most CPUs, like branch prediction and different levels of data caching. These measures reduced the silicon footprint of each core and made it possible to add multiple cores to the GPU before CPU designers where able to do the same. Today, even if more advanced caching has been added to GPUs, this is still the main reason why GPUs have higher peak performance than CPUs (Fig. 1.2). Another reason is the inherent parallel nature of the problem of rendering graphics. When geometry is shaded, projected, and rasterized, the same instruction is needed for a lot of data at the same time. This makes a special kind of instruction, known as Single Instruction Multiple Data (SIMD) [5], especially suited for this problem. How SIMD instructions are utilized in modern CPUs and GPUs is discussed in Section 1.1.1.

For the first adopters of general-purpose computing on GPUs (GPGPU) it soon became evident that offloading all types of computations to the GPU was not always a good idea. In most cases it is important to balance the load equally between the CPU and GPU to get the most out of each processors. Balancing means that highly

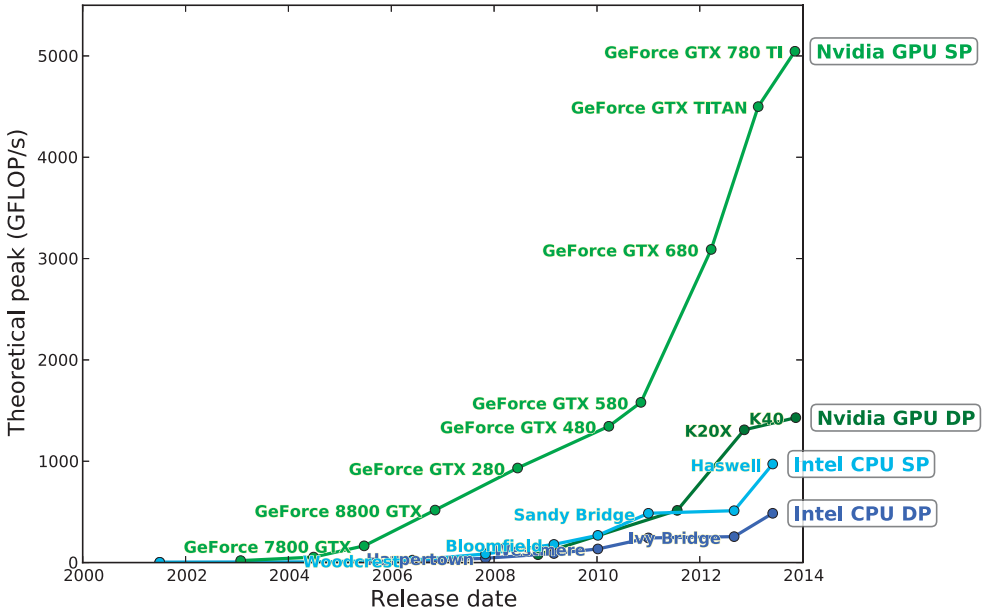


Figure 1.2: Development in theoretical peak throughput, single precision (SP) and double precision (DP), the last decade for Intel CPUs and Nvidia GPUs (github.com/mgalloy/cpu-vs-gpu).

parallel and computationally intensive tasks should be offloaded to the GPU while serialized and memory intensive work should stay on the CPU. This paradigm of utilizing specialized cores for solving specific problems is today known as *heterogeneous computing*. The specialized core can be a GPU or e.g. a field-programmable gate array (FPGA). A recent trend is that specialized accelerator cores are embedded onto the CPU. This heterogeneous processor is often referred to as an accelerated processing unit (APU). The last three generations of CPUs from Intel have all had on-chip integrated graphics, and all of them are therefore APUs. An on-chip GPU is less powerful than a high-performance discrete GPU, but valuable time can be saved by not having to send data across the PCI Express bus, which connects the different parts of a computer to the CPU. This is the reason why one of the first rules of GPGPU (with discrete GPUs) is to minimize (or hide) CPU-to-GPU memory transfers. An integrated GPU can therefore provide increased performance over a discrete GPU if the problem requires a lot of CPU-GPU data transfers.

The final transition currently taking place in the computer industry is the introduction of cloud computing, where compute power is provided as a web service and automatically scales on demand. Together, multi-cores, heterogeneous cores, and cloud cores form the "jungle" that today's software engineers have to handle. In this thesis we focus on heterogeneous computing with a multi-core CPU and a discrete

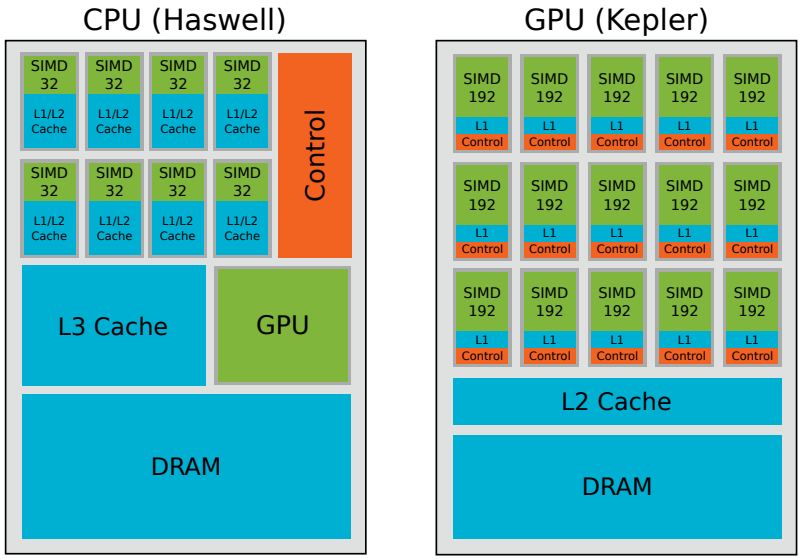


Figure 1.3: Schematic overview of CPU (Intel Haswell) and GPU (Nvidia Kepler) architectures. Note how more space is dedicated to compute cores on a GPU, and how more space is dedicated to control and caching on a CPU.

high performance GPU. This is a cheap, high-performance system which is likely to be the specification of current and future ultrasound imaging systems.

1.1.1 Comparing CPU and GPU performance

The CPU is designed to be the main control unit of a personal computer, and is therefore good at general processing. The GPU, on the other hand, has a history of processing graphics which involves heavy vector arithmetic. Although they started out as very different processors, modern CPUs are now including more and more GPU-like features, and GPUs are adding CPU features. In this section we will see what these features are and why there is still a big difference in theoretical peak arithmetic performance between a CPU and a GPU.

A special kind of instructions called SIMD was mentioned in Section 1.1. These are instructions that are designed to execute a single instruction on multiple data elements in one clock cycle. In the late 1990s, Intel introduced a SIMD instruction set for their Pentium III processor called the Streaming SIMD Extensions (SSE). The first SSE instruction set had registers of 128 bit which meant that four single-precision (32 bit) floating point values could be processed per clock cycle. Updates of the SSE instruction set (SSE2, SSE3, SSE4, AVX, AVX2, and AVX-512) have added new instructions and longer registers. The recently introduced extensions, AVX and AVX2, add registers of 256 bit and fused multiply-add (FMA) instructions which means that sixteen single-precision floating point operations can be performed per

clock cycle per core. The new extension AVX-512, that extends the registers to 512 bit, is found in the Intel accelerator board Xeon Phi, and might show up in the next generation of CPUs from Intel. The latest CPU architecture from Intel, the Haswell architecture, has two processing ports per core supporting FMA-AVX2 instructions, and is therefore capable of performing 32 single-precision floating point operations per clock cycle per core (Fig. 1.3). For an eight-core[1] CPU with a clock frequency of 3.8 GHz this sums to a theoretical peak arithmetic performance of

$$32 \times 8 \times 3.8 = 972.8\,\text{GFLOPS}, \tag{1.1}$$

where FLOPS stands for floating point operations per second. This is the same number as visualized in Fig. 1.2 for the Haswell architecture (single precision). In addition, the most powerful integrated GPU found on Haswell CPUs, the HD 5200, has 40 execution units capable of processing sixteen single-precision floating point values at 1.3 GHz. This adds

$$16 \times 40 \times 1.3 = 832\,\text{GFLOPS} \tag{1.2}$$

in extra processing capabilities to the APU.

Modern GPUs can also be interpreted as units consisting of multiple SIMD cores (Fig. 1.3). The recent Kepler architecture from Nvidia provides up to 15 SIMD cores[2] capable of processing 192 single-precision floating point values per clock cycle per core. These GPUs are also capable of performing FMA instructions, effectively doubling the number of theoretical FLOPS. The maximum theoretical peak single precision FLOPS for this architecture with a core frequency of 745 MHz is then found to be

$$15 \times 192 \times 2 \times 745 = 4291\,\text{GFLOPS}. \tag{1.3}$$

Each SIMD core also includes 16 special function units (SFU) with close to 1 TFLOPS in computational capabilities. One Kepler GPU therefore provides more than 5 TFLOPS as shown in Fig. 1.2 for the GeForce GTX 780 TI GPU. From Fig. 1.3 we see how the high number of SIMD cores found on GPUs and their larger width, compared to the SIMD cores on CPUs, results in five times the computational performance of a high-end CPU (without the integrated GPU).

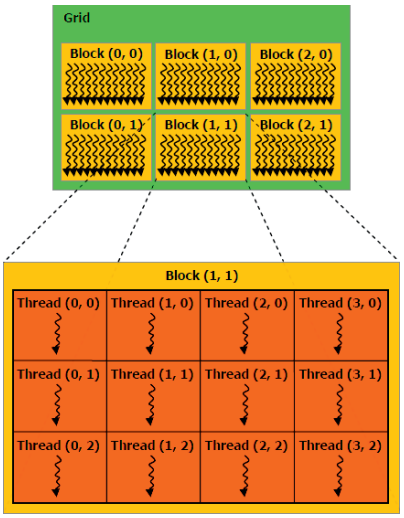
Note that the derived numbers are strictly theoretical and that the actual throughput is highly algorithm dependent. Nevertheless, in theory, the computational throughput provided by modern GPUs is five times higher than that provided by CPUs. A claimed speed-up of several orders of magnitude is therefore typically a result of comparing a non-optimized CPU implementation with an optimized GPU implementation or by comparing incomparable hardware [6, 7]. However, in the next section we will see why it is often easier to obtain an optimized implementation for the GPU than for the CPU.

[1] An 8-core Haswell CPU is scheduled to launch during 2014. The numbers in Fig. 1.2 for the last three generations of CPUs are all based on 8-core CPUs.

[2] Nvidia likes to refer to the unit capable of processing one single precision floating point value as a core. In Nvidia's world, Kepler therefore has $15 \times 192 = 2880$ cores. A SIMD core on a Nvidia GPU is called a Streaming Multiprocessor (SM or SMX).

```
__global__ void square(float *a, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col  = blockIdx.x * blockDim.x + threadIdx.x;
    if (row<N && col<N) {
        int idx = N*row + col;
        a[idx] = a[idx] * a[idx];
    }
}
```

(a) Kernel



(b) Threads, blocks and grid

Figure 1.4: GPU threads are grouped into blocks and arranged in a grid. One thread runs a copy of a kernel function. In this example the kernel function performs an in-place element-wise matrix square ((b) is from http://docs.nvidia.com/cuda).

In this section we have focused on CPUs from Intel and GPUs from Nvidia, however, CPUs and GPUs from AMD have similar specifications and differences in theoretical peak performance.

1.1.2 Programming a GPU

Early GPGPU programming required expert knowledge of computer graphics and the graphics pipeline. When GPGPU gained a broader interest and the GPU started to be used to compute problems that did not involve graphics, it quickly became evident that new programing languages were needed. One of the first GPGPU oriented languages was the Brook language by Buck *et al.* [8]. Buck later joined Nvidia where he led the work on CUDA (Compute unified device architecture), a GPGPU language and framework by Nvidia for Nvidia GPUs only. The first version of CUDA was released in 2007. The following year, Apple launched a GPGPU framework known as OpenCL, a standard which is maintained by the Khronos Group. These two frameworks are still today the main programming frameworks for GPGPU. While CUDA only runs on Nvidia GPUs, OpenCL can now run on GPUs from both Nvidia and AMD, CPUs from both Intel and AMD, FPGAs from Altera, and mobile processors from ARM.

Both CUDA and OpenCL are centered around a kernel function which is executed by a large group of threads. The kernel function, for both CUDA and OpenCL, has a C-like syntax with some additional specifiers. An example of a CUDA kernel is depicted in Fig. 1.4a. The `__global__` specifier tells the CUDA compiler that this

function is a kernel function, and in CUDA this kernel function is launched by using a CUDA-specific syntax:

```
square<<<grid, block>>>(a, N),
```

where `grid` and `block` are 3-component vectors specifying grid and block size in "xyz" as depicted in Fig. 1.4b, `a` is a pointer to a memory buffer located on the GPU, and `N*N` is the size of the buffer `a`.

When the kernel is launched, a copy of it is scheduled for each thread in each block, and each kernel instance gets as input its location in the grid (`blockIdx` and `threadIdx`) together with the grid size (`gridDim` and `blockDim`). This information is then typically used to calculate which values to gather from the global memory, in this case an element of `a`. When a kernel is executed, multiple arbitrary thread blocks are scheduled to each SIMD core where they stay until all instructions in the kernel are finished for all threads in all the blocks. This step is then repeated until all blocks have been processed. Thus, in order to formulate a given problem as a kernel function, it must be possible for the GPU to schedule the resulting thread blocks in any order. As an example, the global memory input to each thread can therefore not depend on the output from other blocks.

As we see, a thread has a different conceptual meaning on a GPU than on a CPU. While a CPU is said to execute threads which can perform SIMD instructions, a thread on a GPU can be interpreted as a unit capable of processing one data element in the SIMD instruction. Because of this the GPU is sometimes referred to as a single instruction multiple threads (SIMT) processor instead of a single instruction multiple data (SIMD) processor. This also explain why GPU threads are said to be light-weighted. Such a conceptual difference means that there is an inherent focus on breaking algorithms down into SIMD instructions when programing for the GPU. This often leads to a highly optimized parallel implementation on the GPU and not on the CPU. Manually typed SIMT, or SIMD, instructions is the only way of programming the GPU, while a single thread with automatic vectorization of for-loops is the standard way of programming the CPU.

It is important to understand that not all algorithms will benefit from wider SIMD registers. As the width increases, the only algorithms that will benefit are those which can be divided into a sufficiently large number of independent calculations. Also, to achieve peak performance, the implementation must perform one FMA operation on each core per clock cycle, and it must be possible for the GPU to hide all memory latency by scheduling threads which are not stalled by a memory fetch. All of this is something that is not likely to happen in practical applications, and therefore one will usually end up with a throughput which is a fraction of the peak throughput.

The code in Fig. 1.4b is an incomplete and minimal example of a GPU program. A full example will involve several lines of C++ code transferring data to and from the GPU, and the kernel could include manual administration of the L1 cache (shared memory), and intra-block thread barriers. This is part of the reasons why writing a high performance GPU kernel is often found to be a complex and time consuming task. Therefore, we will probably see a lot of work aimed at reducing this overhead in the future. A recent development is the introduction of C++ AMP which enables

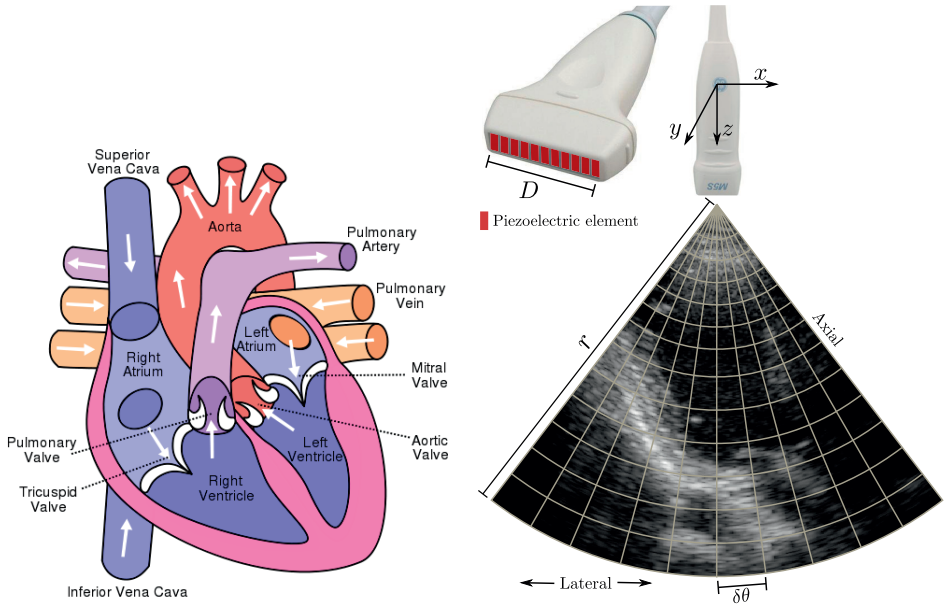
developers to write accelerated code (e.g. by a GPU) in plain C++. Hopefully, in near future, programming the GPU will be a job for compilers, and direct fiddling with GPU kernels should only be needed if maximum throughput is absolutely needed. This is similar to how SIMD instructions are handled in CPU-code today. Most of the time developers will rely on the CPU compilers to analyze the code and auto-apply SIMD instruction whenever it is possible. However, when maximum throughput is needed, developers with deep knowledge of the algorithm are often able to make faster code by manually writing SIMD instruction. The relevance of auto-applied GPU instructions will also increase due to the recent embedding of GPUs on the same chip as the CPU. Soon, GPU programming will hopefully be a matter of coding a heterogeneous collection of cores where operations are automatically offloaded to the right core (e.g. a GPU) by the compiler.

1.2 Medical ultrasound imaging

Medical ultrasound imaging is considered to be a low-risk tool for non-invasive investigation of the human body. Neither gamma rays nor high intensity magnetic fields are involved, just sound pressure below well-controlled limits. It is also far less costly than other medical imaging modalities and offers real-time imaging. Ultrasound imaging is well known for its use in fetal imaging during pregnancy, however, in this thesis we will focus on a different modality, namely cardiac ultrasound imaging. An illustration of the human heart is therefore included in Fig. 1.5a. Throughout the thesis we will refer to some of the parts shown in Fig. 1.5a. Especially note the four chambers of the heart, the left and right atria, and the left and right ventricles. The tissue separating the left and right ventricle is known as the inter-ventricular septum. The left side is responsible for pumping blood out to the whole body through the aortic valve. It is therefore the strongest and hardest working part of the heart. When blood flows out of the aorta, it is important that the mitral valve, which separates the left atrium from the left ventricle, is closed and does not leak. The left side, and especially the left ventricle and the mitral valve, therefore receives the most attention during cardiac ultrasound examinations. When visualizing 3D ultrasound images in **Paper IV** we will have a strong focus on visibility of cardiac tissue, thus note for later that the endocardium refers to the innermost layer of the cardiac muscle.

To generate the ultrasonic frequencies used for imaging, a piezoelectric ceramic is used. A piezoelectric material has a special property where mechanical vibrations are generated when voltage is applied and *vice versa*. It can therefore be used to both send and receive ultrasound signals, like a combined microphone and speaker. A set of piezoelectric elements organized in an array is referred to as an ultrasound *probe*, and special probes are typically designed for different medical imaging modalities. Examples of probes for imaging of the carotid artery in the neck and for cardiac imaging are shown in Fig. 1.5b, to the left and right respectively.

If the width of a piezoelectric element is less than the wavelength of the transmitted signal the element approaches *omnidirectional* characteristics, hence, energy will be distributed or sensed equally in all directions. When multiple omnidirectional



(a) (b)

Figure 1.5: a) Overview of the human heart. Illustration from wikipedia.org. b) Illustration of probe geometry and layout for phased array imaging.

elements are arranged in an array, the transmitted pulse will get shaped into a beam[3] of ultrasound by diffraction as the width (the *aperture*) of the array grows. The angular width of this beam, and also the angular resolution in radians of the array, is approximately given by [9]:

$$\delta\theta_a = \frac{\lambda}{D} = \frac{c}{fD}, \tag{1.4}$$

where λ and f are the ultrasound wavelength and frequency respectively, c is the speed of sound in human tissue, and D is the aperture width. With an average speed of sound in human tissue of $1540\frac{\text{m}}{\text{s}}$, the angular resolution or opening angle of a 2 cm cardiac probe operating at 3 MHz is around 1.5°. The axial resolution is inversely proportional to the pulse length, and is typically better than the lateral resolution. An overall view of the cardiac ultrasound imaging geometry is depicted in Fig. 1.5b. A line in the depicted ultrasound image is generated by firing an ultrasound pulse or beam into the body and sample the received echo. We will refer to this amplitude trace as an *image line*.

[3]A beam refers to the accumulated magnitude across time of a pulse traveling in space.

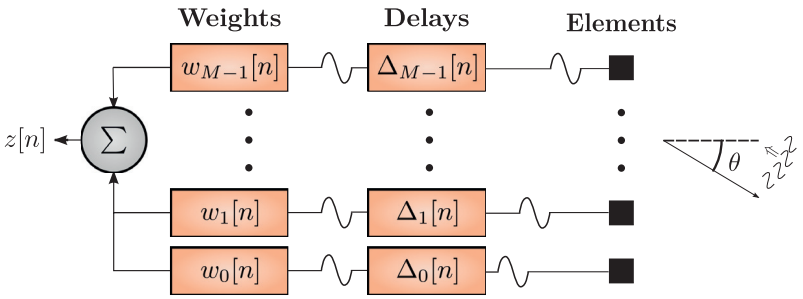


Figure 1.6: Delay-and-sum array beamforming. Image courtesy of Jo Inge Buskenes.

1.2.1 Beamforming

If time delays are applied to the signal emitted by the array elements it is possible to steer and focus the ultrasound beam in space (Fig. 1.6). This technique is known as *beamforming*, and for ultrasound imaging it is applied both when transmitting and receiving ultrasound pulses. A cardiac probe is a phased probe, which means that the image is constructed from image lines which are gradually steered from $-\theta/2$ to $\theta/2$. As depicted in Fig. 1.5b, this results in an imaging sector which is θ wide. This is known as a 2D B-mode image (B for brightness). If image lines are acquired for multiple B-mode images that are tilted in the yz-plane we get what is known as a volumetric B-mode or a 3D ultrasound image. A probe for 3D imaging has several thousand elements arranged as a matrix in order to facilitate steering in both the xz and yz planes. In this thesis we will focus on 2D imaging when the topic is adaptive beamforming, and 3D imaging when the topic is adaptive visualization.

The standard form of beamforming used in ultrasound imaging is known as *delay-and-sum* (DAS):

$$z[n] = \sum_{m=0}^{M-1} w_m^* x_m[n - \Delta_m[n]] = \mathbf{w}^H \mathbf{x}[n]. \tag{1.5}$$

Here, M is the number of elements in the array, x_m is the per-element data, Δ_m is the per-element focusing and steering delay, $\mathbf{w}$ is the array *weight vector* or *apodization*, $\mathbf{x}$ is a vector of pre-delayed element data, and $z[n]$ is the output image line in the direction given by $\{\Delta_m[n]\}$. From now on we will omit the dependency on $\{\Delta_m[n]\}$ and x_m will refer to delayed element data. A schematic overview of delay-and-sum array beamforming is depicted in Fig. 1.6. If p_o is the phase center of our array and p is the point to which we want to steer and focus the ultrasound beam, we have to use the following delays:

$$\Delta_m[n] = \frac{|p_o + p| - |p_m + p|}{c}, \tag{1.6}$$

where c is the speed of sound in human tissue, and p_m is the position of the m'th element. When transmitting, the pipeline depicted in Fig. 1.6, without the sum

operator, is reversed and the negated set of delays has to be used. Also note that on transmit we have to decide where to steer and focus prior to transmitting a pulse. Transmit delays therefore do not vary within an image line.

1.2.2 Sampling and processing complexity

When generating a B-mode image, we have to ensure that the spatial and temporal sampling is sufficient. We have already seen that the lateral resolution of an ultrasound probe is given by (1.4). Hergum *et al.* [10] have further shown that the lateral frequency support for a system that both emits and receives energy is governed by the sum of the transmit (D_{tx}) and receive (D_{rx}) aperture. The lateral resolution of what is known as an active imaging system is found to be

$$\delta\theta = \frac{\lambda}{D_{tx} + D_{rx}}. \tag{1.7}$$

Then, in order to properly sample an imaging sector of θ radians we have to acquire

$$N_\theta = \frac{\theta}{\delta\theta} \tag{1.8}$$

image lines distributed across the imaging sector. If r is the maximum range in our image, as shown in Fig. 1.5b, we can acquire

$$f_{\text{FPS}} = \frac{c}{2rN_\theta} \tag{1.9}$$

images or frames per second (FPS). With our 2 cm cardiac probe we get a frame rate of around 55 FPS when imaging a 70° sector with $r = 15$ cm.

The required axial sampling is dictated by the transmit frequency and the frequency bandwidth of the probe. From Fourier theory we know that a short pulse consists of many frequency components. A short ultrasound pulse, which would yield a high axial resolution, requires therefore a large frequency bandwidth. Medical ultrasound systems typically have a bandwidth between 33 % and 100 % of the pulse frequency. This corresponds to transmitting a pulse which is between 1 and 3 oscillations long. In the analog-to-digital converter of an ultrasound imaging system a high sampling frequency is typically used (e.g. 40 MHz). However, the ultrasound signal is band limited, and the data rate can therefore be reduced in a process known as *in-phase quadrature (IQ) demodulation*. In this process the axial sampling rate is reduced to the bandwidth of the imaging system in the following three steps: demodulation with a complex exponential, low-pass filtering, and decimation. The number of samples in one image line is after this process given by:

$$N = \frac{2r}{c} B_w f_c, \tag{1.10}$$

where f_c is the center frequency of the transmitted pulse, and where B_w is the relative bandwidth in percent. Note that after this process the ultrasound data will be complex valued. The total number of samples in a B-mode image from a phased array is then

$$N_{\text{B-mode}} = N_\theta N. \tag{1.11}$$

If our cardiac probe, which is used to image a 70° sector, has a relative bandwidth of 80 % we have to process almost 50000 complex data vectors of length M every 1/55 second. The formula for the number of FLOPS required for real-time DAS beamforming of cardiac ultrasound is then roughly given as:

$$\text{FLOPS}_{\text{DAS}} = MN_{\text{B-mode}} f_{\text{FPS}} = MB_w f_c. \tag{1.12}$$

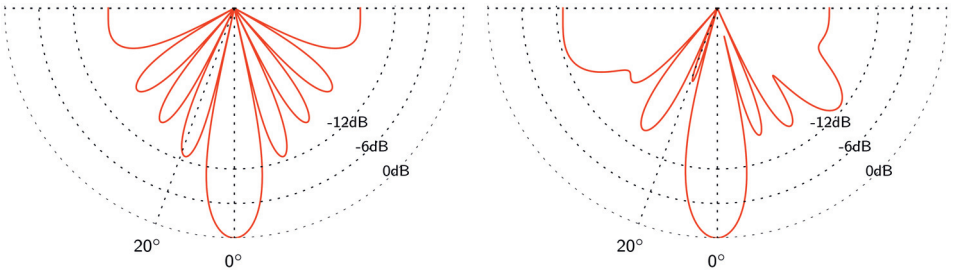
If our cardiac probe has $M = 128$ elements, DAS beamforming requires a processing throughput of around 300 MFLOPS. This is far below the peak processing throughput of modern GPUs and CPUs. The main challenge with DAS beamforming is the amount of data that has to be moved. In our example, for a sample size of four bytes, around 1.2 GB has to be moved to the GPU every second. This is however less than the maximum bandwidth of PCI Express 2.0 (8 GB/s), even after the relative bandwidth, transmit frequency and number of array elements are increased with a total factor of 5. Newer versions of PCI Express, which support even higher transfer rates, have also recently been introduced.

1.2.3 GPUs in medical ultrasound imaging

Ultrasound scanners generate an incredible amount of data. A modern system capable of 3D imaging will generate several tera-bytes of data inside the probe handle during a clinical scan. Fortunately the data rate is reduced to the level of 2D imaging before the data reaches the scanner. The data rate of a 2D scan is "only" in order of giga-bytes per second. To handle this amount of data, ultrasound imaging systems have, for the last twenty years, relied heavily on application-specific integrated circuits (ASIC) for all steps in the processing pipeline [11]. However, as the computational power of generic processors have increased, the expensive ASICs have gradually been replaced by software [12]. One of the last remaining ASICs in ultrasound scanners is used for beamforming [11]. As seen from (1.5), beamforming involves a reduction of the data rate with a factor equal to the number of elements in the ultrasound probe. The input data rate is therefore very high. Nevertheless, real-time beamforming has recently been achieved on a GPU [13], and a French company (SuperSonic Imagine) are utilizing GPUs in their plane wave beamforming algorithm [14]. This is the same technique as reported implemented on a GPU by Yiu *et al.* [15]. Software beamforming is also available in the Vantage research ultrasound system by Verasonics[4]. We can therefore conclude that beamforming is finally moving into software too. The largest players in the medical ultrasound business, General Electric and Philips, are also well aware of where the trend is going [16] [17].

Software beamforming opens up for cost-efficient and rapid prototyping of advanced processing of ultrasound data. One example is Capon beamforming which is explored in this thesis (**Papers I** and **II**) and introduced in the next section. The increased computational power introduced by GPUs makes it also possible to include more advanced 3D visualization as presented in [18], **Papers IV** and **IX**.

[4] http://www.verasonics.com/25



(a) Uniform weights (b) Capon weights

Figure 1.7: Examples of array beampatterns where the array weights are either uniform (a) or derived using Capon's method (b). An interfering echo is coming from 20°. Notice the reduced sidelobe level for the Capon beampattern in the direction of the interferer. Image courtesy of Jo Inge Buskenes.

1.2.4 Adaptive beamforming

Adaptive beamforming has several meanings in the medical ultrasound imaging literature. It can for instance refer to aberration correction [19] which is the task of applying time delays to compensate for pulse distortion introduced by fatty layers in the human body. However, in this thesis adaptive beamforming will refer to data-dependent weighting derived from a given optimization criterion. The optimization criterion that we focus on is the minimum variance distortionless response criterion, which is also known as Capon's method [20]. A beamformer that uses weights derived with Capon's method will in this thesis mainly be referred to as the *Capon beamformer* [21]. However, it is also known as the minimum variance (MV) beamformer [22], the minimum variance distortionless response beamformer (MVDR) [**Paper I**], or the constrained adaptive beamformer [23].

The Capon beamformer produces a weight vector, as introduced in (1.5), based on the impinging signal and a constrained optimization problem. The problem is as follows [20]:

$$\begin{aligned} &\min_{\mathbf{w}} E\{|z[n]|^2\} \approx \min_{\mathbf{w}} \mathbf{w}^H \hat{\mathbf{R}} \mathbf{w} \\ &\text{subject to } \mathbf{w}^H \mathbf{a} = 1, \end{aligned} \tag{1.13}$$

where $\hat{\mathbf{R}}$ is a sample covariance matrix, and $(.)^H$ is the complex conjugate transpose. Hence, the resulting weight vector minimizes the output power (the criterion) while maintaining unit gain in the steering direction $\mathbf{a}$ (distortionless response). The benefit of Capon beamforming is seen from Fig 1.7, which presents a sensitivity map (an *array beampattern*) for DAS beamforming with uniform weights and the Capon beamformer. In this simulated case, an interfering echo is coming from 20°, which for the DAS beamformer with uniform weights (1.7a) will leak into the beamformer output with its power attenuated by around 13 dB. Since the Capon weights are data-dependent, the method "knows" where the interfering echo is coming from and is able

to position a zero in its direction (Fig. 1.7b). Hence, the interfering echo will be highly attenuated independent of the direction from which it arrives. Another interesting case is when the interfering echo has a direction-of-arrival within the mainlobe of the array beampattern. In theory, an echo will be regarded by the Capon beamformer as an interferer as long as the signal propagation vector of the echo does not match the steering vector ($\mathbf{a}$) exactly. Positioning of a zero in the direction of an interferer within the mainlobe will lead to a shifted and scaled mainlobe if this is what is needed in order to achieve minimum output power. Adaptive positioning of zeros and mainlobe shifting are the sources of the super-resolution capabilities of the Capon beamformer [22], and the positive implications of this effect [24] is the main reason why this beamformer has been highly studied in the field of medical ultrasound imaging over the last decade.

To "know" the direction of interfering echoes the Capon beamformer has to estimate the statistics of the impinging wave field. This is done by estimating a sample covariance matrix, which for an active broadband system can be done in the following way [24]:

$$\breve{\mathbf{R}}[n] = \frac{1}{N_L N_K} \sum_{n'=n-K}^{n+K} \sum_{l=0}^{N_L-1} \mathbf{x}_l[n'] \mathbf{x}_l[n']^H \tag{1.14}$$

$$\mathbf{x}_l[n'] = [x_l[n'], \ldots, x_{l+L}[n']] \tag{1.15}$$

where $N_K = 2K + 1$ is the number of samples used for temporal smoothing, $\mathbf{x}_l$ is a subarray of length L, and $N_L = M - L + 1$ is the number of subarrays used for spatial averaging. Temporal averaging is used to obtain DAS-like speckle statistics in combination with large subarrays [25], and subarray averaging is applied to reduce signal cancellation caused by coherent signals [26]. The number of temporal samples is selected relative to the pulse length. To ensure robustness against model errors and numerical stability the matrix is further loaded with a diagonal factor,

$$\epsilon = d \times \text{trace}\{\breve{\mathbf{R}}\}/L \tag{1.16}$$

$$\hat{\mathbf{R}} = \breve{\mathbf{R}} + \epsilon \mathbf{I}. \tag{1.17}$$

This adds a white noise component proportional to the in-coherent signal power to the estimate [27].

The solution to the constrained minimization problem in (1.13) is, by the method of Lagrange multipliers [28], found to be

$$\mathbf{w}[n] = \frac{\hat{\mathbf{R}}[n]^{-1}\mathbf{a}}{\mathbf{a}^H \hat{\mathbf{R}}[n]^{-1}\mathbf{a}} \in \mathbb{C}^L. \tag{1.18}$$

Since $\mathbf{w}$ is time (or range) dependent, a matrix has to be constructed and inverted for each data vector $\mathbf{x}[n]$ received by the system. This is indeed computationally demanding, something which becomes clear when each step of the beamformer is examined closer. It is then found that constructing the covariance matrix has a computational complexity of $O(L^2 N_L N_K)$, inverting it has a computational complexity of $O(L^3)$, and the final normalization and calculation of $\mathbf{w}$ is $O(L)$. For

$L = M/2$, a subarray size often used in the literature, the first two steps both have a complexity of $O(L^3)$. These steps should therefore receive the same amount of attention when a real-time implementation of Capon beamforming is attempted.

A rough estimate of the number of FLOPS required by a Capon beamformer for real-time cardiac ultrasound imaging is then given by:

$$\text{FLOPS}_{\text{Capon}} = L^3 B_w f_c. \tag{1.19}$$

For our example, with $L = M/2 = 64$, this sums up to a required throughput of more than 1/2 TFLOPS, which is more than thousand times that needed for real-time DAS beamforming. On the other hand, 1/2 TFLOPS is just 10% of the peak throughput of modern GPUs, but remember that the latter is only a theoretical number, and that the actual throughput is highly algorithm dependent. The required throughput for Capon beamforming might therefore still be hard to achieve in practice [29, **Paper II**]. As for DAS beamforming, this number also scales with higher relative bandwidth, pulse frequency, and more array elements since the subarray size is typically proportional to the number of array elements.

In the medical ultrasound imaging literature we find few attempts of implementing a Capon beamformer for real-time imaging. In addition to the papers in this thesis only Chen *et al.* [30–32] are known to have explored GPU and FPGA implementations of the Capon beamformer for medical ultrasound imaging. The differences between our work and Chen *et al.* are discussed in **Papers I**, **II**, and **VI**.

To reduce the processing requirement of the Capon beamformer for medical ultrasound imaging, several simplified versions have been proposed in the literature [33, 34]. Among the most interesting proposals we find the low complexity adaptive (LCA) beamformer by Synnevåg *et al.* [35], and the beamspace (BS) Capon beamformer by Nilsen and Hafizovic [36]. The LCA beamformer uses a predefined set of weights, and the BS-Capon beamformer takes advantage of the narrow transmit and receive beams used in medical ultrasound imaging. Beamspace typically refers to the polar grid that cardiac ultrasound data are located in before it is scan converted into a Cartesian grid. In combination with the Capon beamformer, beamspace refers to the wavenumber-space representation of the recorded signals, i.e., the spatial Fourier transform of the element data:

$$\mathbf{x}_{\text{BS},l} = \mathbf{B}[x_l[n], \ldots, x_{l+L}[n]]^T \tag{1.20}$$

$$[\mathbf{B}]_{p,q} = \frac{1}{\sqrt{L}} e^{-j2\pi pq/L}. \tag{1.21}$$

The idea behind BS-Capon beamforming is that spatial beams with little information can be removed from $\hat{\mathbf{R}}$ by removing rows in B. When investigating the complexity of the BS-Capon beamformer [**Paper II**] we find that it is dominated by the transformation of data into beamspace, $O(N_L L N_b)$, where N_b is the number of selected beams.

The number of FLOPS required for BS-Capon beamforming of real-time cardiac ultrasound is then found to be around:

$$\text{FLOPS}_{\text{BS-Capon}} = N_L L N_b B_w f_c \tag{1.22}$$

For our example, BS-Capon beamforming with $L = M/2 = 64$ and $N_b = 3$ will require a throughput of around 30 GFLOPS. This is a level of processing which both modern CPUs and GPUs should be able to handle. A GPU implementation of the BS-Capon beamformer for real-time cardiac ultrasound imaging is presented in **Paper II**. The rationale behind using the GPU is that in the heterogeneous computing paradigm a highly parallel task like the Capon and BS-Capon beamformer should be offloaded to the processor made for parallel processing. The CPU is then left to do managing and all the tasks it traditionally performs. It should also be possible to achieve higher processing throughput on a GPU than on a CPU for this kind of problem.

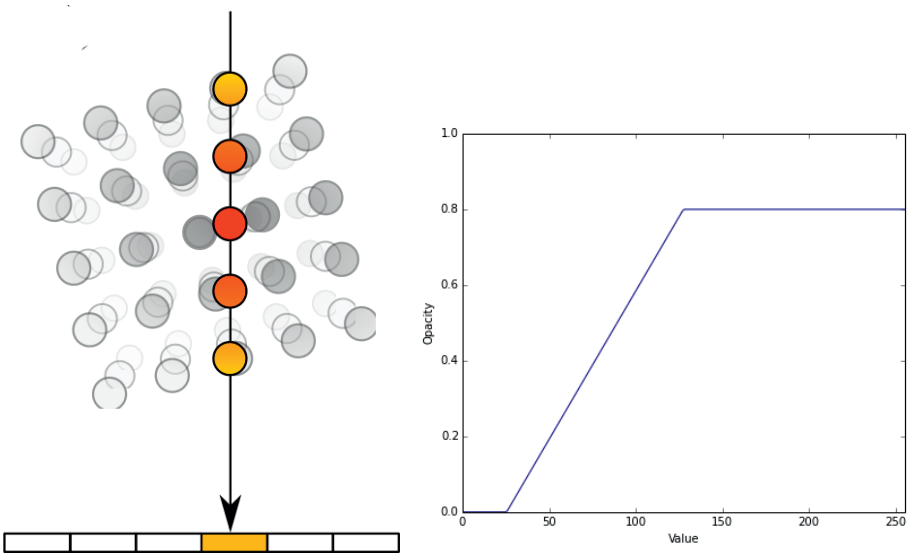
1.2.5 Capon beamforming of focused broadband beams

Both the LCA and BS-Capon beamformers are based upon a detailed investigation of how Capon beamfoming behaves when applied to medical ultrasound images. For the LCA beamformer, the windows selected by the Capon beamformer were investigated and a special set of predefined windows were defined. If one looks closer at the windows selected for the LCA beamformer we see that half the set performs micro-steering of the mainlobe and the other half provides different mainlobe widths. The results from applying the LCA beamformer [35, Fig. 9] and the success of the BS-Capon beamformer shows that the main benefit of applying Capon beamforming to medical ultrasound imaging is micro-steering of the mainlobe, and not attenuation of far off-axis backscatter. This is mainly due to the focused beams used in medical ultrasound which highly limits the amount of energy arriving off axis. This focusing of energy also explains why Capon beamforming works on broadband ultrasound data.

The Capon beamformer is derived for narrowband applications and has traditionally been applied to medical ultrasound imaging "as is". Medical ultrasound imaging is, however, a broadband imaging modality and one could think that a broadband Capon beamformer would therefore be needed [37]. The investigation performed for the BS-Capon beamformer [36] showed, that with pre-delayed data, energy will be centered on-axis and as little as three near-axis beams in beamspace is therefore adequate in order to achieve similar performance for BS-Capon and Capon beamforming. The two-way beam will be wider for unfocused (plane wave) transmit beams [37, 38], but steering and focusing on receive will still make the main amount of energy arrive on axis. As shown in **Paper III** it is also possible to micro-steer with phase delays (narrowband) instead of time delays (broadband) without introducing large errors. Since Capon beamforming mainly performs main-lobe shifts smaller than the beam width [35, 36], the difference between a broadband and narrowband Capon beamformer will be minimal [39]. Broadband Capon beamforming is also significantly more computationally demanding than narrowband Capon beamforming.

1.3 Volume rendering

A 3D ultrasound dataset is a cloud of scalar values that somehow needs to be presented to the user. One approach used for cardiac ultrasound is to extract slices corresponding



(a) Ray-casting (b) Opacity transfer function

Figure 1.8: The principle behind volume ray casting. a) A ray is casted out from every pixel in a 2D image. Along the ray, samples are gathered from the volume. The samples are assigned a color and an opacity before they are combined into a single pixel value (wikipedia.org/wiki/Volume_ray_casting). b) The opacity transfer function used to determine how opaque a sample should be.

to different long-axis and short-axis views of the cardiac chambers. Another way is to project the whole volume into a two-dimensional image. The latter technique is known as volume rendering, and it presents to the user the three dimensional structure of the heart in a single two-dimensional image. Many different methods for volume rendering exist [40], but in this thesis we will focus on ray casting [41].

The principle behind ray casting is depicted in Fig. 1.8. The projection of volumetric data is performed by casting a ray out from every pixel in the 2D image (Fig. 1.8a). Along each ray, samples are gathered from the volume and assigned a color and opacity. For ultrasound visualization, the color intensity is dictated by the ultrasound backscatter intensity, and the colorfulness is often encoded with depth information in order to improve depth perception. An example of a volume rendering of cardiac ultrasound data is presented in Fig. IV.1. The opacity, however, is governed by a user-controlled *opacity transfer function* (OTF) as shown in Fig. 1.8b. For ultrasound visualization a monotonically increasing function is typically used, which means that below a certain threshold all samples will be rendered transparent (opacity = 0). We will refer to this as the *opacity threshold* (τ), and for the example in Fig. 1.8b this threshold is 25.

The iterative composition scheme for front-to-back ray casting of one ray, from s_0 to s_{N_s-1} where s is the sample position and N_s is the number of samples along the

ray, is [42]

$$I(s_0, s_{i+1}) = I(s_0, s_i) + (1 - \alpha_{\text{acc},i})c_{i+1} \tag{1.23}$$

$$\alpha_{\text{acc},i+1} = \alpha_{\text{acc},i} + (1 - \alpha_{\text{acc},i})\alpha(c_i + 1). \tag{1.24}$$

Here, $\alpha(v)$ is the opacity transfer function, c_k is the color of sample s_k, α_{acc} is the accumulated opacity, and $I(s_0, s_{N_s-1})$ is the intensity value of all the projected samples combined. If $\alpha(v)$ is close to one for one or two consecutive samples, α_{acc} will quickly approach a value of one. An $\alpha_{acc,k}$-value close to one means that the contribution from $c_{k+1}, \cdots, c_{N_s-1}$ to the final value, $I(s_0, s_{N_s-1})$, will be minimal. Hence, the iteration can stop at $i = k$ without loosing visible samples. This optimization technique is known as *early-ray-termination*, and will significantly speed up the rendering algorithm if an opaque material is rendered.

For visualization of cardiac ultrasound we want α_{acc} to be close to one only when a tissue interface has been encountered. If a sample of clutter noise is assigned a high opacity value, cardiac tissue further behind will not be visible. It is said to be occluded by clutter noise. The general goal in cardiac ultrasound visualization is therefore to assign high opacity to cardiac tissue and low opacity to noise samples. This is a big challenge for cardiac ultrasound since the backscattered signal level from tissue varies a lot throughout the volume. The sample value distributions for tissue and noise therefore always overlap to some degree. With a global OTF it is therefore impossible to find a threshold that separates tissue from noise.

Volume rendering of a single volume has a complexity of $O(N_x N_y N_s)$, where $N_x N_y$ is the number of pixels in the output image. With $N_x = N_y = N_s = 512$ and a required frame rate for 3D cardiac ultrasound between ten and twenty, we find that the required number of FLOPS for real-time volume rendering of 3D cardiac ultrasound is in the order of several GFLOPS. When calculation of gradients for shading and more advanced processing is added per sample this number continues to grow with a constant factor.

1.3.1 Adaptive volume rendering

As for adaptive beamforming, adaptive volume rendering means to derive and use information about the data being visualized from the data itself. In the literature on volume rendering this is known as feature enhancement or visibility driven visualization [43–45]. The common feature of these techniques is how the sample opacity is modulated in order to achieve visibility of relevant structures. If we somehow are able to classify samples as cardiac tissue we can use this information as input to an adaptive volume rendering algorithm, and by that achieve improved visibility of cardiac tissue.

In **Paper IV** we extend the work by Honigmann *et al.* [46] on a global adaptive OTF by formulating a regional adaptive OTF for visualization of cardiac ultrasound volumes. This regional adaptive OTF is based on estimated statistics of the clutter noise inside the left ventricle (the blood pool). The added complexity of this technique is caused by the statistical estimation, which is basically a second volume rendering

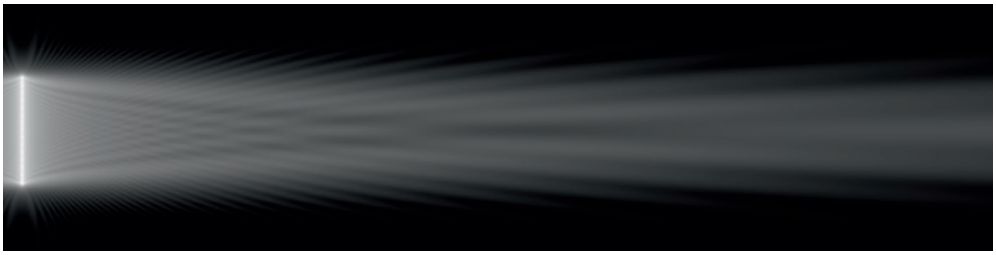


Figure 1.9: An example of a simulated ultrasound pressure field from a linear array without focusing. The field was simulated using the simulator in **Paper V**.

(mean and variance projection) that has to be performed before the rendered image is produced.

1.4 Ultrasound field simulations

Simulation of ultrasound probes are heavily used by researchers in order to test out new algorithms and probe designs. A simulator for ultrasound probes typically works by approximating the probe surface with discrete points or patches. In software it is then possible to transmit a pulse from a given probe and measure its response in space and time. Figure 1.9 shows a snapshot of a simulated pressure field from an unfocused linear array emitting a continuous wave. The image was simulated with the simulator from **Paper V**.

The most popular simulator for medical ultrasound field simulations is Field II [47]. All simulations in both **Paper II** and **III** are produced with this simulator. The Field II simulator applies a far-field approximation where each element is divided into mathematical elements. All field points then needs to be in the far-field of these small elements[5]. It is then possible to approximate the spatial impulse response for a given probe by summing a trapezoid impulse response function originating from each mathematical element in all field points [47]. A rough estimate of the number of required FLOPS for calculating the spatial impulse response as it is done in Field II is:

$$\text{FLOPS}_{\text{Field II}} = f_s \frac{w}{c} N_{\text{elements}} N_{\text{field points}}, \tag{1.25}$$

where f_s is the sampling frequency, w is the width of the mathematical elements, $f_s \frac{w}{c}$ is the maximal number of samples in each trapezoid function, N_{elements} is the total number of mathematical elements, and $N_{\text{field points}}$ is the number of field points. The required number of FLOPS for a simulation at 100 MHz with 500, $\lambda/2$ wide, mathematical elements across a field of 0.5 million points is 6 GFLOPS in order for it to complete after one second. If we want to simulate an ultrasound image the spatial

[5] $l \gg \frac{w^2}{4\lambda}$, where w is the mathematical element width, l is the distance to the field point, and λ is the wavelength of the transmitted pulse. For $w = \lambda/2$ the approximation is valid for $l > \lambda/16$.

impulse response on both transmit and receive have to be calculated and convolved in time. The response will also be different for each image line. Hence, more than 800 GFLOPS is required in order to complete a Field II simulation in one second with the given example values and an image consisting of 70 image lines. Other simulators have also been introduced which are faster but less exact than Field II [10].

For the simulator presented in **Paper V** the required number of FLOPS in order to achieve a certain frame rate f_{FPS} is:

$$\text{FLOPS}_{\text{HOS}} = N_{\text{source points}} N_{\text{field points}} f_{\text{FPS}}. \tag{1.26}$$

To simulate a field of one million points with several thousand source points at 50 FPS, more than 100 GFLOPS is required. Note that this simulator evaluates a cosine and sine function for each calculation, something which makes it even harder to achieve the required throughput, since the GPU has less silicon devoted to such tasks (SFUs).

The above examples show that realistic and dense ultrasound simulations are computationally demanding. On the other hand we find that the required amount of processing is within the theoretical limits of modern CPUs and GPUs. Since ultrasound field simulations are quite easy to accelerate in parallel it should be possible to add increased interactivity to ultrasound simulations tools by utilizing parallel programing and SIMD instructions. In addition to **Paper V**, several simulators have recently accomplished this by leveraging the powers of GPUs [48–50].

1.5 Concluding remarks

All the presented problems in this chapter, except DAS beamforming, have in common that they require a processing throughput of several GFLOPS. Comparing this to the typical clock frequency of modern CPUs at 3 GHz we see that GFLOPS problems quickly require parallel execution in order to achieve real-time performance. With CPU threads the theoretical peak performance on an eight-core CPU increases to 24 GFLOPS. However, to reach a peak performance above 100 GFLOPS, SIMD instructions on a CPU or SIMT instructions on a GPU have to be used. However, to apply threads and SIMD instructions, it must be possible to divide the problem into a large number of independent calculations.

Chapter 2

Introduction to Papers

2.1 Motivation

The rapid development of computer game technology and accompanying programming languages, as shown in Chapter 1, have recently provided researchers with small personal supercomputers, comprised in a single graphics processing unit (GPU). This immense rise in computational capabilities and improved programmability are currently changing how ultrasound imaging systems are designed. Algorithms that previously had to be implemented in hardware, for performance reasons, can now be implemented in software. It is clear that high-performance programmable processors, like the GPU, already have and will continue to make future ultrasound imaging systems more flexible, cheaper to produce, and equipped with even more cutting-edge processing.

The Capon beamformer in ultrasound imaging is a good example of an algorithm that has been widely studied over the last decade where the number of calculations was thought to be too high for real-time processing to happen in near future. As shown in Section 1.2.4, an effective processing rate of hundreds of GFLOPS is required to achieve real-time processing for cardiac ultrasound imaging. This number, as explained in Section 1.2.4, also grows with larger subarrays and an increasing number of samples per second. Nevertheless, with modern GPUs, this amount of processing is finally available within a single GPU card. When researchers are exploring new algorithms for ultrasound imaging, it is important to keep the architecture of parallel accelerators in mind. If a new complex algorithm is supposed to run in real time, it needs to fit the programmable and parallel pipeline of modern ultrasound scanners.

2.2 Aims of study

The overall aim of this study has been to investigate the possibility of utilizing GPUs for advanced processing in an ultrasound imaging system.

The main focus has been the Capon beamformer, and the problem of making this computationally intense algorithm available for real-time ultrasound imaging

[**Papers I** and **II**]. However, two additional methods have also been explored. The first one being adaptive visualization of cardiac ultrasound volumes [**Paper IV**], and the second one being the simulation of dense ultrasound pressure fields [**Paper V**]. Both share the property of being computationally complex, but on the other hand they consist of many independent computations, which makes them perfectly suited for parallel GPU acceleration.

Well into the project, when a real-time Capon beamformer was realized, and loops of images for the first time were processed at real-time frame rates, new issues where discovered and had to be solved. This led to a more theoretical study of the Capon beamformer in **Paper III**, with special attention on how to obtain high lateral resolution while preserving the important shift-invariant property of ultrasound imaging. Shift-invariant behavior is crucial if the method is ever to be applied for live scanning.

2.3 Summary of papers

2.3.1 Paper I

An Optimized GPU Implementation of the MVDR Beamformer for Active Sonar Imaging
J. I. Buskenes, **J. P. Åsen**, C.-I. C. Nilsen and A. Austeng
IEEE Transactions on Oceanic Engineering, Jul 2014.

The first paper describes in detail how the Capon beamformer is mapped to the GPU architecture. Even though the paper is written within the field of active sonar imaging, the presented implementation is applicable to a range of active imaging systems. A similar discussion for cardiac ultrasound imaging can be found in **Paper VI**. Active sonar imaging typically differs from medical ultrasound imaging by a less strict real-time requirement and fewer array elements. This makes it somewhat easier to reach the goal of real-time processing. The transmit beams used for sonar imaging are also much wider than the once used for ultrasound imaging. This makes techniques like the BS-Capon beamformer less attractive for sonar imaging. However, as shown in this paper, low-complexity methods are not needed to reach real time Capon beamforming for sonar imaging.

The estimation of the spatial covariance matrix receives special attention in this study. In previous literature on Capon beamforming the matrix inversion has always been regarded as the most computationally complex step. In this paper we show that estimation of the sample covariance matrix is the most complex part when spatial and temporal smoothing is performed with common parameters. We then show how the computational complexity of this estimation procedure can be reduced from cubic to square. Finally, an in-depth analysis of the arithmetic throughput on multiple platforms is given. Despite our efforts, the number of effective FLOPS is only 4 % and 14 % of the theoretical throughput of the target GPU for the matrix equation solver and covariance matrix estimator respectively. Still, the reported throughput

of 1 Mpx/s on a high-end GPU is enough to provide real-time processing for sonar imaging.

2.3.2 Paper II

Implementing Capon Beamforming on a GPU for Real-Time Cardiac Ultrasound Imaging
J. P. Åsen, J. I. Buskenes, C.-I. C Nilsen, A. Austeng and S. Holm
IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 61, no. 1, pp. 76-85, Jan. 2014.

The second paper aims at implementing real-time Capon beamforming for cardiac ultrasound imaging. Achieving this will facilitate further study of the method's *in vivo* performance in the future. This is something that has been suggested as further work in many publications on Capon beamforming for medical ultrasound imaging over the past decade. In **Paper I**, arrays with no more than 32 elements were investigated. A linear array for cardiac ultrasound imaging typically has 64 or more elements. In **Paper VI**, which is summarized in **Paper II**, it is shown that our implementation from **Paper I** does not reach the level of throughput required for real-time Capon beamforming in cardiac ultrasound imaging. The matrices that have to be inverted become to large, and the frames that need to be processed per second are too many.

In this paper, we implement the beamspace version of the Capon beamformer (BS-Capon) on the GPU to reduce the covariance matrix size. For parameters that give similar performance for BS-Capon and element-space Capon (ES-Capon), we show that real-time BS-Capon beamforming is feasible for cardiac ultrasound imaging. The reported processing throughput is able to keep up with the acquisition frame rate in a typical cardiac ultrasound imaging system equipped with 64- and 96-element linear arrays. **Papers II** and **VI** also present, for the first time, videos where the ES-Capon and BS-Capon beamformer have been applied on loops of simulated and *in vivo* medical ultrasound images.

2.3.3 Paper III

Capon Beamforming and Moving Objects - An Analysis of Lateral Shift-Invariance
J. P. Åsen, A. Austeng and S. Holm
IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, , vol. 61, no. 7, pp. 1152-1160, Jul 2014.

In the third paper we study the shift-invariant property of an imaging system based on the Capon beamformer. As mentioned, shift-invariant imaging is essential if the method is ever to be used in practice. The paper is based on an observation of aliasing artifacts when the Capon beamformer was used to image bright point scatterers *in vitro* and in simulations. The intensity of the point scatterers were observed to oscillate as either they or the probe moved. In earlier work on Capon beamforming for medical

ultrasound imaging the real-time nature of ultrasound imaging has often been ignored, and a large degree of oversampling on transmit might have been applied without this being clearly stated. Oversampling on transmit will reduce the acquisition frame rate, and is therefore not an option if the imaging system is used to image rapidly varying objects. Smooth presentation of probe motion is equally important since real-time user interaction is one of the key selling points of ultrasound imaging.

Further, it has been investigated how the Capon beamforming achieves its super-resolution, and how this makes it highly sensitive to differences between the assumed steering vector and the signal propagation vector. Earlier work, focusing on single images, has typically referred to this effect as speed-of-sound errors, however, it also comes naturally into play when objects move across an imaging sector which has insufficient lateral sampling. With a narrowband and farfield model it is shown that the Capon beamformer suffers from lateral gain variations as large as 27 dB for parameters previously used to obtain high resolution medical ultrasound images. To obtain the same lateral gain variations as for DAS beamforming, 25 times oversampling is shown to be required. The same gain variations are observed for broadband nearfield simulations. Finally, we show that the gain variation can be reduced by oversampling on receive using phase rotation steering, hence improving shift-invariance. This is achieved without affecting the acquisition frame rate and with only a minor increase in computational complexity. The method is successfully applied on simulations and *in vitro* phantom data.

2.3.4 Paper IV

Adaptive Volume Rendering of Cardiac 3D Ultrasound Images - Utilizing Blood Pool Statistics
J. P. Åsen, E. Steen, G. Kiss, A. Thorstensen and S. I. Rabben
Proc. SPIE Medical Imaging 2012, vol. 8320, pp. 832008.

The fourth paper proposes an adaptive volume rendering method based on statistics derived from the blood pool of the left ventricle. Noise located in the blood pool tends to occlude cardiac tissue when rendered in 3D, and is often impossible to remove by adjusting global rendering parameters. This is because ultrasound signals have high variability within blood, as well as high variability within tissue, even when heavily smoothed. Delineation of the blood pool is done by a state estimation algorithm, capable of tracking the endocardium in real time. The final visualizations are compared, with respect to location of tissue, by using a state-of-the-art endocardium segmentation tool. The paper presents both quantitative and qualitative results supporting the method's improved capabilities of reducing tissue dropouts and spurious structures inside the blood pool, leading to a maximized amount of visible endocardium tissue. The proposed method is implemented on a GPU with real-time frame rates as the result. This results shows that with modern GPUs it is possible to add more advanced visualization methods to an ultrasound imaging system and still have real-time performance.

2.3.5 Paper V

Huygens on Speed: Interactive Simulation of Ultrasound Pressure Fields
J. P. Åsen and S. Holm
Proc. IEEE Ultrasonics Symposium 2012, pp. 1643-1646.

The fifth paper presents an interactive simulation tool capable of simulating dense pressure fields from ultrasound arrays in real time. Simulation tools are heavily used by researchers to test new algorithms and the performance of new array designs prior to manufacturing. However, dense simulations involve extensive calculations and computational time. With the presented simulator we aim at reducing the time needed for calculating pressure fields by means of GPU acceleration. The tool is based on Huygens' principle, which describes diffraction caused by a slit as the superposition of several point sources located inside the opening. Thus, the simulator works by accumulating the contribution from a collection of point sources in a set of observation points. How the source and observation points are laid out is left for the user to decide. Point sources positioned along a line will for instance simulate an ultrasound array. The simulator is linked with both a painting interface for interactive drawing of arrays, and a Matlab interface for precise scripting of array configurations. The main contribution in this paper is the increased performance of the GPU implementation compared with CPU and Matlab[1] versions. Second, the painting interface provides a neat way of demonstrating array beamforming principles in real time. We believe this to be of great value both for array designers and when teaching array processing.

2.4 Main contributions

The main contributions of this thesis are:

1. a GPU implementation of the Capon beamformer.
2. a GPU implementation of the beamspace Capon beamformer.
3. achieving real-time beamspace Capon beamforming for cardiac ultrasound imaging.
4. a first-time investigation of Capon beamforming applied on multiple frames in medical ultrasound imaging (simulated, *in vitro*, and *in vivo*).
5. an analysis of shift-invariance of the Capon beamformer when applied to medical ultrasound imaging.
6. a method for real-time reduction of blood-pool noise in volume renderings of cardiac ultrasound volumes.
7. a GPU implementation of a painting tool for rapid simulation and visualization of ultrasound pressure fields.

[1] The Ultrasim toolbox.

In addition, the thesis provides several more general discussions on how to utilize the GPU to accelerate advanced ultrasound imaging algorithms.

2.5 Discussion and future work

2.5.1 Paper I

As mentioned, we were only able to achieve a fraction of the peak theoretic throughput of the target GPU in **Paper I**. This shows how hard it is to reach the theoretical level of throughput, and that how close one can get is highly algorithm dependent. There might be ways to improve on these numbers, but both the matrix equation solver and the covariance matrix estimation step consist of several points where fine granular synchronization and serialized instructions are needed. These facts will restrict even the most finely tuned implementation from reaching the maximum throughput of the GPU. The work in **Paper I** also showed that the covariance estimation kernel was memory bound, hence there are not enough instructions to hide all the memory latency. Further improvements through improved memory management could therefore be possible.

An interesting observation in **Paper I** can be found in Fig. I.10. Here we see how the achieved throughput of the covariance estimation kernel is more than three times higher than the theoretical throughput of a high-end CPU (The target CPU is from 2010). Note that the target high-end GPU, the Nvidia Quadro 6000, is also from 2010. This clearly shows the benefit of modern GPU computing.

The matrix equation found in the Capon beamformer is solved using a batched Gauss Jordan solver implemented by Nvidia. In our work we focused on the covariance matrix estimation step. Nevertheless, **Paper I** showed that the solver had significantly lower throughput than the covariance matrix estimation kernel. A natural next step would therefore be to analyze the current solver, and figure out if any further optimization is possible. Since the sample covariance matrix is hermitian, a solver based on Cholesky decomposition should also help to improve the throughput. However, a triangular matrix can make memory management and finding the best thread layout even more difficult.

2.5.2 Paper II

In **Paper II** we focused on achieving real time Capon beamforming for cardiac ultrasound imaging by implementing the BS-Capon beamformer on a GPU. However, we also present the first medical ultrasound loops processed with ES- and BS-Capon beamforming. The paper does not provide any detailed evaluation of the method's *in vivo* performance. Yet, initially it has been proven hard to transfer all results obtained in simulations to *in vivo* images. Improvements are observed, but they are not large enough to justify the large number of calculations involved with Capon Beamforming. The beamformers are clearly able to decrease the lateral width of point scatterers and thin structures *in vivo*, and to sharpen edges. On the other hand, the observed contrast is not improved, and sometimes it is even worse (Fig. 6b of

Paper II). Since the Capon beamformer only affects directional noise, it is clear that contrast will get worse in low SNR situations when signal cancellation occurs. The oversampling approach presented in **Paper III** might improve on this, but it will only reduce signal cancellation if the phase shifts across the aperture are close to linear and small. Another issue is that bright points with a wide lateral profile exhibit better visual contrast when surrounded by speckle noise than points with narrow lateral profiles. Future work should address a detailed investigation of why some results, especially the contrast obtained in simulations, are so hard to achieve *in vivo*. It will also be crucial to show larger improvements *in vivo* than what is presented in **Paper II**, both to justify all our computations, and to show an image with both highly improved resolution and contrast.

As pointed out at the end of **Paper II**, cardiac ultrasound imaging might not be the best modality for Capon beamforming. Cardiac ultrasound imaging was mainly selected because it is the medical ultrasound modality that requires the most calculations per second. A better suited adaptive beamformer for cardiac ultrasound imaging might be the low-complexity adaptive beamformer (LCA beamformer), where the Capon optimization problem is applied on a set of predefined windows. This method has linear computational complexity and has similar super-resolution properties as the Capon beamformer. If only minor improvements are obtained, it would be easier to ignore this fact if the algorithm is less computationally costly.

The work in **Paper II** made it clear that interesting findings are done when a real-time implementation makes it possible to watch the temporal behavior of an algorithm. An observation of temporal artifacts was the fact that stimulated for the work of **Paper III**.

2.5.3 Paper III

In **Paper III** we investigate the inherent self nulling of Capon beamforming, and how this effect comes naturally into play when the beamformer is subjected to linear steering vector errors caused by lateral undersampling. This observation is something which has been missing in the literature on Capon beamforming for medical ultrasound imaging. The reason could be that researchers have focused more on the theoretical aspects of Capon beamforming than on the practical aspects of ultrasound imaging. A lack of studies where Capon beamforming is performed on consecutive frames might also explain the lack of discussions on this theme. Fortunately, as shown in **Paper III**, it is possible to maintain the same acquisition frame rate as for DAS beamforming by applying oversampling on receive using a set of steering vectors (phase delays). It also turns out that more matrix inversions are not needed either. Oversampling on receive will, however, increase the penalty of applying post-processing, like filtering, on polar-grid data.

As just mentioned in the discussion of **Paper II**, reduced signal cancellation should prevent contrast from degrading to less than the contrast obtained with DAS beamforming. With improved shift invariance it will also be interesting to see if high-resolution beamforming could act as a post-processing step for other algorithms, for example to increase the precision of speckle and edge tracking. In the case where

the image is generated for post processing it does not need to be visually pleasant anymore, as long as signal cancellation is avoided and the shift-invariance property is maintained. DAS-like speckle has been the driving force behind many of the robustification techniques and the default parameters seen in the literature. Hence, more aggressive parameters might be used if the image produced is intended as input to another algorithm and not for the human eye.

2.5.4 Paper IV

One limitation of the algorithm described in **Paper IV** is its dependence on a successful tracking. If the tracking fails, then the proposed method will fail. Another issue is the view-dependent visibility, where an object could become visible or disappear while rotating or if the local statistics change a lot during the cardiac cycle. Even if the method successfully removes noise, this means that the rendering will become non-intuitive to interact with and unrealistic to watch. However, the method could still be used to clean up still images of standard cardiac views.

The paper does also propose an interesting method for automatic tuning of rendering parameters. Figure IV.3d in **Paper IV** actually shows that an optimal threshold could be found based on global and not only local statistics. Even though the visualization based on global statistics has more errors with respect to endocardium visibility, the tuning of the opacity function is still automatic if an iterative search for the minima in Fig. IV.3d is performed. A global opacity transfer function will insure a rendering which is realistic, but it will still be dependent on a successful tracking for the auto-adjustment to work.

An automatically tuned opacity transfer function based on global statistics combined with an improved smoothing scheme is likely to be better suited for cardiac ultrasound imaging than a local opacity transfer function. A local opacity transfer function is just too extreme to be clinically robust.

For future work, it would also be interesting to see if statistics based on the weight vector selected by the Capon or the LCA beamformer could be utilized to control local opacity modulations. It should also be possible to use this statistic as input to image filtering. An adaptively selected weight vector will be symmetric in homogeneous regions, and left-right shifted close to tissue interfaces. In that way we can, with improved confident, apply less smoothing and higher opacity values at tissue interfaces. If this will be different than just looking at neighboring pixels in an edge-preserving image filter is to be seen.

2.5.5 Paper V

The fifth paper stands out from the other papers by not dealing with an adaptive processing technique. However, it is still centered around the topic of implementing algorithms for ultrasound imaging on the GPU. Ultrasound simulations are typically used for research and sometimes also to pre-calculate configurations of a given scan sequence, for instance to achieve a certain pressure in a given area. The continuous growth in computational power will in the future make it possible to evaluate a scan

sequence on the fly. For example to improve the estimation of the mechanical and thermal index. It will also add improved interactivity to ultrasound simulation tools.

Achieving higher processing speed for an existing algorithm by implementing it on a new architecture is not academically relevant in its own. A good analysis has to be given of how the algorithm was modified for the architecture, or how to utilize the gained speed. These are points we have tried to answer in both **Paper I**, **II**, **IV**, and **V**.

In **Paper V** the increased speed made an interactive painting ultrasound simulation program feasible. As of January 2014 the simulator has been download 200 times and counting[2]. It also has close to a 1000 views on Youtube[3].

2.6 Multimedia content

In **Paper II** and **III** there will be references to several videos that can be accessed online[4]. There is one folder for each paper having multimedia content.

2.7 Software

Finally it is worth mentioning that the code for both the Capon beamformer[5] and the ultrasound field simulator[6] have been released under an open source license.

[2] folk.uio.no/jpaasen/huygens
[3] youtube.com/watch?v=rLgsfskliJM
[4] folk.uio.no/jpaasen/thesis
[5] github.com/jpaasen/cos
[6] github.com/jpaasen/hos

References

[1] H. Sutter, "Welcome to the Jungle," 2011. [Online]. Available: http://herbsutter.com/welcome-to-the-jungle/

[2] ——, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," 2004. [Online]. Available: http://www.gotw.ca/publications/concurrency-ddj.htm

[3] J. S. Seland and T. Dokken, "Real-Time Algebraic Surface Visualization," in *Geometric Modelling, Numerical Simulation, and Optimization*, 2007, pp. 163–183.

[4] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Trans. Graph.*, vol. 21, no. 3, p. 703, Jul. 2002.

[5] M. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[6] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100X GPU vs . CPU Myth : An Evaluation of Throughput Computing on CPU and GPU," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, 2010.

[7] K. Kothapalli, D. S. Banerjee, P. J. Narayanan, S. Sood, A. K. Bahl, S. Sharma, S. Lad, K. K. Singh, K. Matam, S. Bharadwaj, R. Nigam, P. Sakurikar, A. Deshpande, I. Misra, S. Choudhary, and S. Gupta, "CPU and/or GPU: Revisiting the GPU Vs. CPU Myth," p. 20, Mar. 2013.

[8] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM Transactions on Graphics*, vol. 23, no. 3, p. 777, Aug. 2004.

[9] B. A. J. Angelsen, *Principles of medical ultrasound imaging and measurements.* Emantec, 2000, ch. 1.

[10] T. Hergum, S. Langeland, E. W. Remme, and H. Torp, "Fast ultrasound imaging simulation in K-space." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 6, pp. 1159–67, Jun. 2009.

[11] K. Thomenius, "Evolution of ultrasound beamformers," in *Proc. IEEE Ultrasonics Symp.*, vol. 2, 1996, pp. 1615–1622.

[12] I. Guracar, "CUDA Accelerated Real Time Signal Processing in High Performance Diagnostic Ultrasound Imaging," in *NVIDIA GPU Technology Conference*, 2013.

[13] J. Kwon, J. H. Song, S. Bae, T.-k. Song, and Y. Yoo, "An effective beamforming algorithm for a GPU-based ultrasound imaging system," in *2012 IEEE International Ultrasonics Symposium*. IEEE, Oct. 2012, pp. 619–622.

[14] M. Tanter and M. Fink, "Ultrafast imaging in biomedical ultrasound." *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 61, no. 1, pp. 102–19, Jan. 2014.

[15] 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. Ferroelectr. Freq. Control*, vol. 58, no. 8, pp. 1698–705, Aug. 2011.

[16] K. E. Thomenius, "Trends w. Ultrasound Scanners Clinical and Research Implications," in *American Association of Physicists in Medicine (AAPM) Annual Meeting*, 2012.

[17] S. Metz, "The Future of Ultrasound Systems and Imaging," in *American Association of Physicists in Medicine (AAPM) Annual Meeting*, 2011.

[18] V. Šoltészová, D. Patel, S. Bruckner, and I. Viola, "A Multidirectional Occlusion Shading Model for Direct Volume Rendering," in *Comput. Graph. Forum*, vol. 29, no. 3, 2010, pp. 883–891.

[19] C. Cole, G. Holley, D. Langdon, S. Maslak, and J. Wright, "Method and apparatus for real-time, concurrent adaptive focusing in an ultrasound beamformer imaging system," Nov. 1996, uS Patent 5,570,691.

[20] J. Capon, "High-resolution frequency-wavenumber spectrum analysis," *Proceedings of the IEEE*, vol. 57, no. 8, pp. 1408–1418, 1969.

[21] F. Vignon and M. R. Burcher, "Capon beamforming in medical ultrasound imaging with focused beams." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 55, no. 3, pp. 619–628, Mar. 2008.

[22] J.-F. Synnevåg, A. Austeng, and S. Holm, "Adaptive Beamforming Applied to Medical Ultrasound Imaging," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 54, no. 8, pp. 1606–1613, Aug. 2007.

[23] J. Mann and W. Walker, "A constrained adaptive beamformer for medical ultrasound: initial results," in *IEEE Ultrason. Symp.*, vol. 2, 2002, pp. 1807–1810.

[24] J.-F. Synnevåg, A. Austeng, and S. Holm, "Benefits of minimum-variance beamforming in medical ultrasound imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 9, pp. 1868–1879, Sep. 2009.

[25] J.-F. Synnevåg, C.-I. C. Nilsen, and S. Holm, "Speckle Statistics in Adaptive Beamforming," in *Proc. IEEE Ultrasonics Symp.* IEEE, Oct. 2007, pp. 1545–1548.

[26] V. Reddy, A. Paulraj, and T. Kailath, "Performance analysis of the optimum beamformer in the presence of correlated sources and its behavior under spatial smoothing," *IEEE Audio, Speech, Language Process.*, vol. 35, no. 7, pp. 927–936, Jul. 1987.

[27] W. Featherstone, "A novel method to improve the performance of Capon's minimum variance estimator," in *Tenth International Conf. on Antennas and Propagation (ICAP)*, 1997, pp. 322–325.

[28] H. L. Van Trees, *Detection, estimation and modulation theory IV: Optimum Array Processing.* John Wiley and Sons, Inc., 2003.

[29] H. So, J. Chen, B. Yiu, and A. Yu, "Medical Ultrasound Imaging: To GPU or Not to GPU?" *IEEE Micro*, vol. 31, no. 5, pp. 54–65, Sep. 2011.

[30] J. Chen, B. Y. S. Yiu, H. K. So, and A. C. H. Yu, "Real-Time GPU-Based Adaptive Beamformer for High Quality Ultrasound Imaging," in *Proc. IEEE Ultrasonics Symp.*, 2011, pp. 1–4.

[31] J. Chen, A. C. H. Yu, and H. K.-H. So, "Design considerations of real-time adaptive beamformer for medical ultrasound research using FPGA and GPU," in *International Conf. on Field-Programmable Technology (FPT)*, 2012, pp. 198–205.

[32] J. Chen, B. Y. Yiu, B. K. Hamilton, A. C. Yu, and H. K.-H. So, "Design space exploration of adaptive beamforming acceleration for bedside and portable medical ultrasound imaging," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, p. 20, Dec. 2011.

[33] B. M. Asl and A. Mahloojifar, "A Low-Complexity Adaptive Beamformer for Ultrasound Imaging Using Structured Covariance Matrix," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 59, no. 4, pp. 660–667, Apr. 2012.

[34] K. Kim, S. Park, Y.-T. Kim, S.-C. Park, J. Kang, J.-H. Kim, and B. MooHo, "Flexible Minimum Variance Weights Estimation Using Principal Component Analysis," in *Proc. IEEE Ultrasonics Symp.*, 2012.

[35] J.-F. Synnevåg, A. Austeng, and S. Holm, "A Low-Complexity Data-Dependent Beamformer." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 58, no. 2, pp. 281–289, Feb. 2011.

[36] C.-I. C. Nilsen and I. Hafizovic, "Beamspace adaptive beamforming for ultrasound imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 10, pp. 2187–2197, Oct. 2009.

[37] I. K. Holfort, F. Gran, and J. A. Jensen, "Broadband minimum variance beamforming for ultrasound imaging." *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 56, no. 2, pp. 314–25, Feb. 2009.

[38] 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 Int. Ultrason. Symp.* IEEE, Oct. 2011, pp. 2448–2451.

[39] K. Diamantis, I. H. Voxen, A. H. Greenaway, T. Anderson, J. r. A. Jensen, and V. Sboros, "A comparison between temporal and subband minimum variance adaptive beamforming," in *Proc. SPIE*, 2014.

[40] K. Brodlie and J. Wood, "Recent Advances in Volume Visualization," *Comput. Graph. forum*, vol. 20, no. 2, pp. 125–148, 2001.

[41] M. Levoy, "Display of Surfaces From Volume Data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29–37, 1988.

[42] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-Time VOLUME GRAPHICS*. A K Peters Ltd., 2006.

[43] I. Viola, A. Kanitsar, and M. E. Gröller, "Importance-Driven Feature Enhancement in Volume Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, pp. 408–418, 2005.

[44] C. D. Correa and K. L. Ma, "Visibility Histograms and Visibility-Driven Transfer Functions," *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[45] S. Marchesin, J.-M. Dischler, and C. Mongenet, "Per-Pixel Opacity Modulation for Feature Enhancement in Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 560–570, 2010.

[46] D. Honigmann, J. Ruisz, and C. Haider, "Adaptive Design of a Global Opacity Transfer Function for Direct Volume Rendering of Ultrasound Data," *Visualization Conference, IEEE*, p. 64, 2003.

[47] J. A. Jensen and N. B. Svendsen, "Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 39, no. 2, pp. 262–267, Jan. 1992.

[48] S. U. Gjerald, R. Brekken, T. Hergum, and J. D'Hooge, "Real-Time Ultrasound Simulation Using the GPU," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 59, no. 5, pp. 885–892, 2012.

[49] T. Reichl, J. Passenger, O. Acosta, and O. Salvado, "Ultrasound goes GPU: real-time simulation using CUDA," *Proceedings of SPIE*, vol. 7261, no. 1, pp. 726 116–726 116–10, 2009.

[50] M. Hlawitschka, R. J. McGough, K. W. Ferrara, and D. E. Kruse, "Fast ultrasound beam prediction for linear and regular two-dimensional arrays." *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 58, no. 9, pp. 2001–12, Sep. 2011.

Paper I

An Optimized GPU Implementation of the MVDR Beamformer for Active Sonar Imaging

Jo Inge Buskenes[1], **Jon Petter Åsen**[2], Carl-Inge C. Nilsen[1] and Andreas Austeng[1]
[1]Department of Informatics, University of Oslo, Oslo, Norway
[2]Medical Imaging Lab (MI-Lab), Norwegian University of Science and Technology, Trondheim, Norway

IEEE Transactions on Oceanic Engineering, Jun 2014.

The Minimum Variance Distortionless Response (MVDR) beamformer has recently been proposed as an attractive alternative to conventional beamformers in active sonar imaging. Unfortunately, it is very computationally complex because a spatial covariance matrix must be estimated and inverted for each image pixel. This may discourage its use unnecessarily in sonar systems which are continuously being pushed to ever higher imaging ranges and resolutions.
In this study we show that for active sonar systems up to 32 channels, the computation time can be significantly reduced by performing arithmetic optimizations, and by implementing the MVDR on a Graphics Processing Unit (GPU). We point out important hardware limitations for these devices, and assess the design in terms of how efficiently it is able to use the GPU's resources. On a quad-core Intel Xeon system with a high-end Nvidia GPU, our GPU implementation renders more than a million pixels per second (1 MP/s). Compared to our initial CPU implementation the optimizations described herein led to a speedup of more than 2 orders of magnitude, or an expected 5-10 times improvement had the CPU received similar optimization effort. This throughput enables real-time processing of sonar data, and makes the MVDR a viable alternative to conventional methods in practical systems.

I.1 Introduction

Data driven methods have been introduced in various imaging fields over the years in attempts to improve image quality. One such method is the Minimum Variance Distortionless Response (MVDR) beamformer. When compared to conventional methods, MVDR is often able to improve image contrast and resolution, as shown in e.g. radar [1], ultrasound [2], and active sonar [3–6].

Despite its inherent potential, the MVDR beamformer has yet to see widespread adoption in the active sonar field. There may be several reasons for this. For one, the method is not inherently robust, and may suffer from a phenomenon called signal cancellation in active systems [7]. Another reason is that in its original form, the computational complexity is cubic with the number of channels, $\mathrm{O}(M^3)$, while conventional beamformers are at $\mathrm{O}(M)$. This is because a spatial covariance matrix is estimated and inverted for each image pixel.

To ensure MVDR robustness, the literature suggests combining means such as temporal and spatial averaging, and regularization [8, 9]. The complexity issue, on the other hand, can be handled by introducing well-founded approximations. For instance, some studies assume data stationarity which allows the formation of a Toeplitz-structured covariance matrix that is simpler to invert [10, 11]. Other studies perform MVDR beamforming in beamspace (the spatial frequency domain), which can be considered sparse due to the limited angular extent of the received beam [8, 12]. Exploiting this can lead to considerable performance improvements, especially in high channel count systems.

In this work, we show that such approximations may not be necessary for sonar systems up to 32 channels, since here the computation time can be reduced by 1-2

orders of magnitude by implementing MVDR on a Graphics Processing Unit (GPU). Similar work may be found in e.g. medical ultrasound imaging, where Chen *et al.* have demonstrated a GPU implementation of the MVDR that operates on real valued data [13, 14]. While this implementation is fast, the real valued data constraint does not allow the creation of asymmetric and shifted responses, and hence prevents the MVDR's potential to be fully reached. Our implementation is not restricted in this manner.

Our implementation is adapted to a well sampled 100 kHz wideband active sonar system operating in the near field. The system has a range resolution comparable to the sample period so our MVDR implementation runs in near single snapshot mode. To deal with this and to avoid signal cancellation we compute and average the covariance estimates from overlapping subarrays [9]. This may resemble techniques such as "subarray MVDR" [15], but unlike that technique we are not using subarrays to reduce complexity but to make it feasible to use MVDR in an active system. In related work, we investigated using the same MVDR GPU implementation in cardiac ultrasound imaging [16, 17], and in active sonar imaging we have compared the performance of this implementation to comparable adaptive methods [18].

This article is outlined as follows: In section I.2 we introduce the concept of adaptive beamforming and provide details on the MVDR method. Then in I.3 we investigate the complexity issue, discuss means for reducing arithmetic complexity, and detail an implementation that makes efficient use of the GPU's parallel resources. The final design is assessed in I.4-I.5, where we provide benchmarks, comparisons with similar CPU implementations, and measures of how efficiently our implementation makes use of the GPU's resources.

I.2 Methods

Consider a sonar imaging scenario where an encoded signal is transmitted to highlight a surface of interest, and assume that the backscattered wavefield is properly sampled using a uniform linear array of receivers. An image can then be formed by coherently combining the receiver outputs to focus at one angle at the time. The principle of adjusting the array's focus in range and direction is commonly referred to as beamforming, and involves assigning suitable delays and weights to the array's channels.

I.2.1 Beamforming

Let the receiver be an M element uniform linear array, and assume that the signature of the transmitted signal has been removed by a matched filter (Fig. I.1). Further assume that the array channels are digitally delayed to focus at a pixel with azimuth angle θ and range sample n, such that the delayed data from the mth channel can be expressed as $x_m[\theta, n]$. To simplify notation, we make the dependence on θ implicit from now on.

By definition the beamformer output $z[n]$ can now be expressed as the weighted

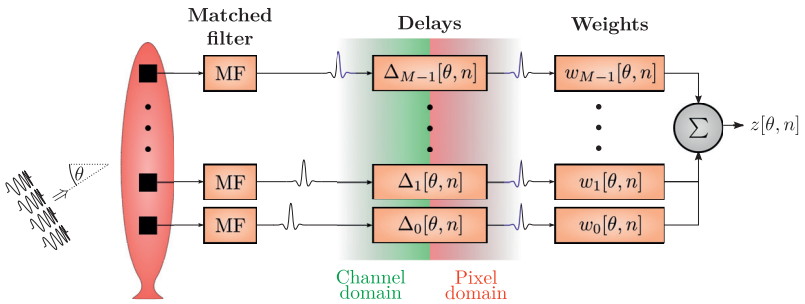


Figure I.1: Beamforming principle. Signal signature is first removed by matched filtering. Then - before summation - a suitable set of delays, Δ, and weights, w, are applied to focus on a pixel of interest at angle and range (θ, n).

sum of all the delayed data samples:

$$z[n] = \boldsymbol{w}^H[n]\boldsymbol{x}[n] = \begin{bmatrix} w_0[n] \\ w_1[n] \\ \vdots \\ w_{M-1}[n] \end{bmatrix}^H \begin{bmatrix} x_0[n] \\ x_1[n] \\ \vdots \\ x_{M-1}[n] \end{bmatrix}, \tag{I.1}$$

where w_m is the weight factor assigned to channel m. With static weights this would be referred to as the conventional delay-and-sum (DAS) beamformer. A large variety of weighting functions exists here for trading lateral resolution for improved noise suppression (contrast), but one always ends up with a compromise between the two [19].

Various adaptive beamformers target this limitation by allowing the weights to change for each pixel to better fit the dynamic nature of the incoming wavefield. In other words, they attempt to use the *a priori* information present in the data to improve image quality. The MVDR beamformer is one such method. It finds the set of complex weights that minimizes the beamformer's expected output power, while ensuring unity gain in the look direction [20]. This is a convex optimization problem that can be solved using Lagrange multipliers to yield the solution:

$$\boldsymbol{w}[n] = \frac{\boldsymbol{R}^{-1}[n]\boldsymbol{a}}{\boldsymbol{a}^T\boldsymbol{R}^{-1}[n]\boldsymbol{a}}, \tag{I.2}$$

where $\boldsymbol{a}$ is a steering vector and $\boldsymbol{R} = E\{\boldsymbol{x}[n]\boldsymbol{x}^H[n]\} \in \mathbb{C}^{M,M}$ is the spatial covariance matrix for the full array. Since we pre-steer our data to every pixel in the image we simplify (I.2) by substituting $\boldsymbol{a}$ with a row vector $\mathbf{1}$ that represents broadside phase-steering. To estimate $\boldsymbol{R}$ we compute a sample covariance matrix $\hat{\boldsymbol{R}}$. In this computation we perform some degree of:

- *spatial averaging* to avoid signal cancellation by decorrelating coherent echoes [9],

- *temporal averaging* over an interval comparable to the pulse length (1-5 samples) to maintain true speckle statistics [21], and
- *diagonal loading* to improve robustness to parameter errors [22, 23].

Combined, these steps will also ensure a numerically well conditioned $\hat{\boldsymbol{R}}$.

We perform *temporal* and *spatial averaging* first and put the result in an intermediate sample covariance matrix $\breve{\boldsymbol{R}}$. To do this we need to segment our array into subarrays. If we let $x_l[n]$ represent the data vector from subarray l,

$$\boldsymbol{x}_l[n] = \begin{bmatrix} x_l[n] & x_{l+1}[n] & \dots & x_{l+L-1}[n] \end{bmatrix}^T, \tag{I.3}$$

then $\breve{\boldsymbol{R}}$ can be calculated as:

$$\breve{\boldsymbol{R}}[n] = \frac{1}{N_K N_L} \sum_{l=0}^{M-L} \sum_{n'=n-K}^{n+K} \boldsymbol{x}_l[n'] \boldsymbol{x}_l^H[n'] \in \mathbb{C}^{L,L}, \tag{I.4}$$

where $N_K = 2K + 1$ is the number of temporal samples to perform averaging over, and $N_L = M - L + 1$ is the number of subarrays.

The final estimate $\hat{\boldsymbol{R}}$ is found by adding a fraction d of the total power of $\breve{\boldsymbol{R}}[n]$ to its diagonal [2]:

$$\hat{\boldsymbol{R}}[n] = \breve{\boldsymbol{R}}[n] + \boldsymbol{I} \frac{d}{L} \mathrm{tr}\{\breve{\boldsymbol{R}}[n]\}, \tag{I.5}$$

where $\boldsymbol{I}$ is an identity matrix, $\mathrm{tr}\{\cdot\}$ represents the matrix trace operation, and $\mathrm{tr}\{\breve{\boldsymbol{R}}[n]\}$ is an estimate of the energy received from this pixel.

Note how subarray averaging led to a size reduction of $\hat{\boldsymbol{R}}$ from $\mathbb{C}^{M,M}$ to $\mathbb{C}^{L,L}$, and hence will produce an L-element weight set when substituted into (I.1). This weight set is applied to all the subarrays, before computing the beamformer output as in (I.1). Or, equivalently, we may apply the weight set to the sum of all the subarrays:

$$z[n] = \boldsymbol{w}^H[n] \sum_{l=0}^{M-L} \boldsymbol{x}_l[n]. \tag{I.6}$$

As summarized in Fig. I.2, the MVDR method is applied to each pixel independently, by:

1. computing the sample covariance matrix $\hat{\boldsymbol{R}}$ in (I.5),
2. computing $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ in (I.2), and
3. computing the beamformer output z in (I.6).

Next we will evaluate these steps in terms of arithmetic complexity, and then discuss their mappability to parallel hardware.

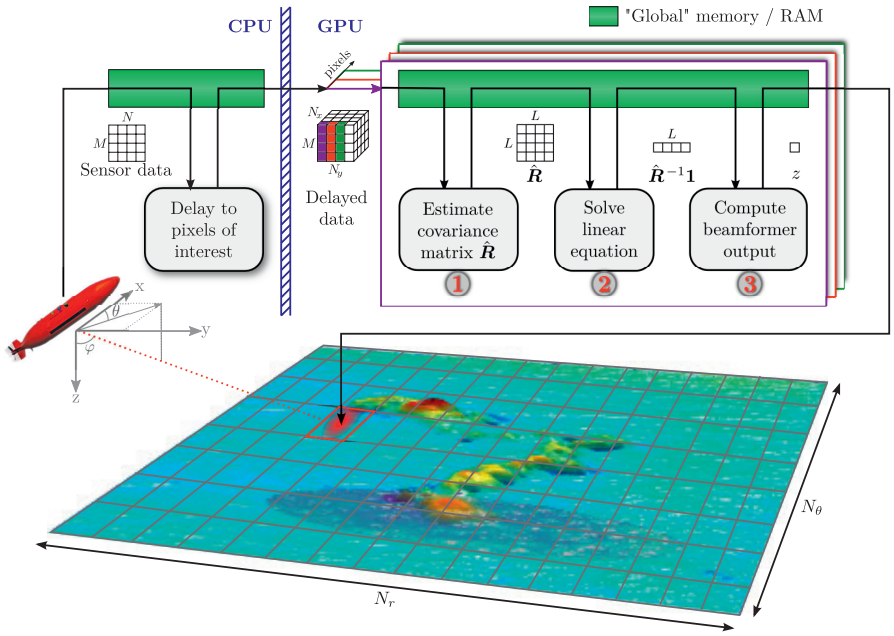


Figure I.2: MVDR beamforming. For each pixel in range and azimuth,
1. an $L \times L$ sample covariance matrix $\hat{\boldsymbol{R}}$ is computed,
2. the term $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ is found using a linear equation solver,
3. and the beamformer output z is computed from (I.6), where $\boldsymbol{w}$ is found by substituting $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ into (I.2).

I.2.2 Computational Complexity

If we neglect spatial and temporal averaging, then the computation of $\hat{\boldsymbol{R}}$ is reduced to a single outer product with complexity of $\mathrm{O}(M^2)$, and the inversion is then of $\mathrm{O}(M^3)$. This might lead us to believe that the inversion step is by far the most complex. But if we implement spatial and temporal averaging as in (I.4), then computing $\hat{\boldsymbol{R}}$ is of $\mathrm{O}(N_K N_L L^2)$ and the inversion of $\mathrm{O}(L^3)$. Computing $\hat{\boldsymbol{R}}$ is now the most complex operation whenever $N_L N_K > L$. To visualize these relations, and include the effects of using complex numbers, we set up complexity formulas that account for the total number of arithmetic operations for each step in the MVDR process (see appendix I.7.1). The only excluded operation was partial pivoting used in the inversion step, but this should contribute marginally to the end result. The entire range of possible subarray sizes from $L \in [1, M]$ was finally evaluated, with temporal averaging set to $K \in \{0, 1, 2\}$, and the number of channels set to $M \in \{8, 16, 32\}$. The results are shown in Fig. I.3.

Note how the computation of $\hat{\boldsymbol{R}}$ completely dominates at smaller subarray sizes, that solving $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ only plays a notable role for larger array and subarray sizes, and that the computation of z has a negligible impact on computation time. Also

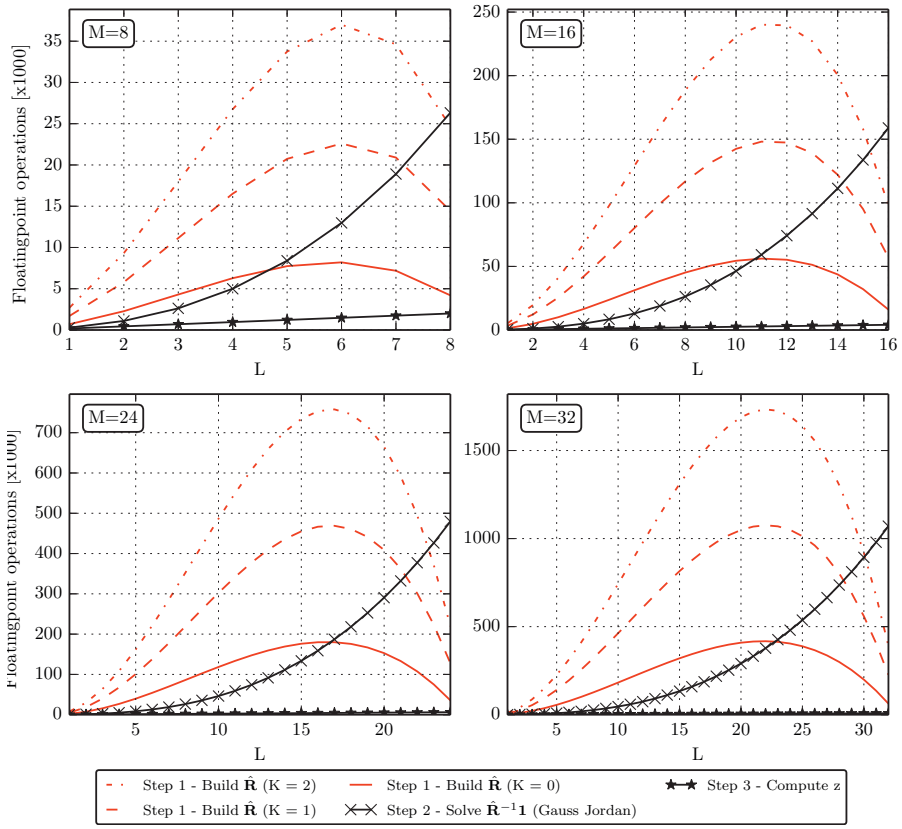


Figure I.3: Per-pixel computational complexity of the steps in MVDR beamforming (before any optimizations). To avoid signal cancellation in an active sonar system we usually set $L < \frac{M}{2}$, in which region the computation of $\hat{\boldsymbol{R}}$ dominates in terms of arithmetic complexity, especially when performing temporal averaging.

notice how temporal averaging comes with a high computational penalty. This is because building $\hat{\boldsymbol{R}}$ is heavy on complex multiplications, which require 3 times as many arithmetic instructions as a complex addition, and a lot of these are repeated unnecessarily.

I.3 Mapping the MVDR to a GPU

An important feature of MVDR beamforming, as with beamforming in general, is that each pixel can be computed independently and in an identical fashion. Furthermore, a typical sonar image may contain millions of pixels. This represents a level of data parallelism that appears very well suited for a massively parallel architecture such as the GPU.

We decided to investigate the feasibility of running MVDR on a GPU by mapping it to a Nvidia GeForce Quadro 6000, which performs similarly to the consumer grade Geforce GTX 480. This is a high-end Compute Unified Device Architecture (CUDA) enabled GPU based on Nvidia's Fermi architecture, which as now been superseded by the Kepler architecture. The code was written in Nvidia's "C for CUDA" framework, which adds a few extra features to the C language to specify how to distribute work across a large number of threads. It would have been possible to use GPUs from AMD and the cross-platform OpenCL framework from the Khronos group instead, but CUDA has a batch linear equation solver [24] that no OpenCL library provides. As will be discussed later, this is a key component in our design.

To use Nvidia's own terminology, the Quadro 6000 is comprised of 14 streaming multiprocessors (SMs), each having 32 CUDA cores that execute a common program called a kernel. Combined these cores deliver a peak performance of more than 1 Tflop/s (appendix I.7.2). In practice this performance is hard to obtain, but one can get fairly close by balancing the load evenly on all cores, and by trying to avoid that some cores are forced to idle due to a pending data transfer or thread synchronization. As the steps in the MVDR method require different strategies for achieving this, we have designed a different kernel and configuration for each of them. We will pay particular attention to building $\hat{\boldsymbol{R}}$, since the potential for gaining overall speedups are greatest here.

I.3.1 Computing the Spatial Covariance Matrix, $\hat{\boldsymbol{R}}$

As observed in Fig. I.3, a direct computation of $\hat{\boldsymbol{R}}$ by implementing (I.4) and (I.5) is the greatest computational burden. We target this issue by first *minimizing arithmetic operations*, then aim to get as close to the peak *arithmetic throughput* on the GPU as possible.

Minimizing Arithmetic Operations

In Fig. I.4 we illustrate how to reduce the arithmetic complexity of building $\hat{\boldsymbol{R}}$ in a system with $M = 5$ sensors, a subarray size of $L = 3$ and temporal averaging is set to $K = 1$. Here the spatial sum is carried out before the temporal sum. To reduce arithmetic operations we may:

1. exploit the fact that $\hat{\boldsymbol{R}}$ is Hermitian positive semi-definite to compute only one half of it,
2. avoid redundant multiplications, and
3. avoid redundant summations by first computing the upper row of $\hat{\boldsymbol{R}}$ and then add/subtract to these iteratively to find the diagonals of $\hat{\boldsymbol{R}}$.

To study the effect of these optimizations we altered the complexity formulas to take them into account. Exploiting symmetry and performing iterative summing (step 1 and 3) is always desirable, but avoiding redundant multiplications (step 2) comes at the cost of increased memory consumption. When all optimizations are applied

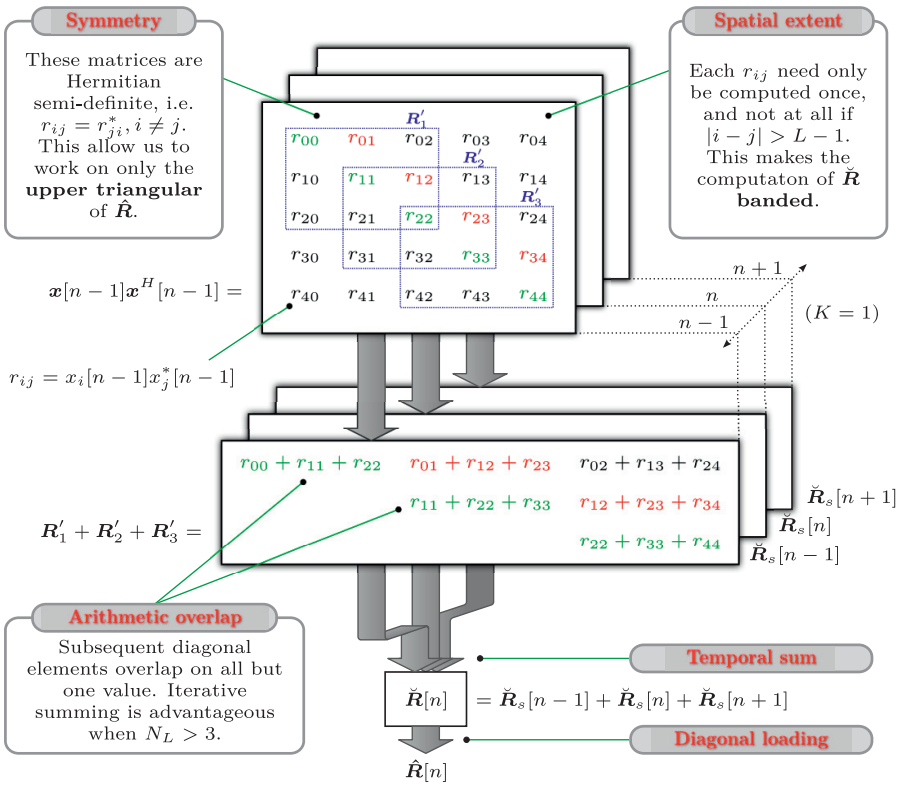


Figure I.4: Step 1: Building $\hat{\boldsymbol{R}}$. This is a visualization of how $\hat{\boldsymbol{R}}$ could be built in a case with $M = 5$ sensors, with subarray size $L = 3$ and temporal averaging set to $K = 1$. Here $\boldsymbol{R}_l'$ is the sample covariance matrix for the lth subarray, and $\breve{\boldsymbol{R}}$ is the average of N_K of these. Note that instead of performing the temporal sum last as here, one could take more temporal samples into consideration in the computation of each r_{ij}.

we call it a "minimum instructions" solution, and when step 2 is left out we call it a "minimum memory" solution. Fig. I.5 compares the two solutions, and formulas may be found in appendix I.7.1. Here the complexity curves are relative to the reference implementation in Fig. I.3. Both solutions reduce the complexity considerably, and recomputing the multiplications does not make that much of a difference. We selected the memory minimized version since our algorithm is severely memory bound, as we will see next.

Arithmetic Throughput

When we compute $\hat{\boldsymbol{R}}$, we very rarely perform more than 1-3 floating point operations for every float read or written to memory. Unfortunately, the GPU prefers kernels to

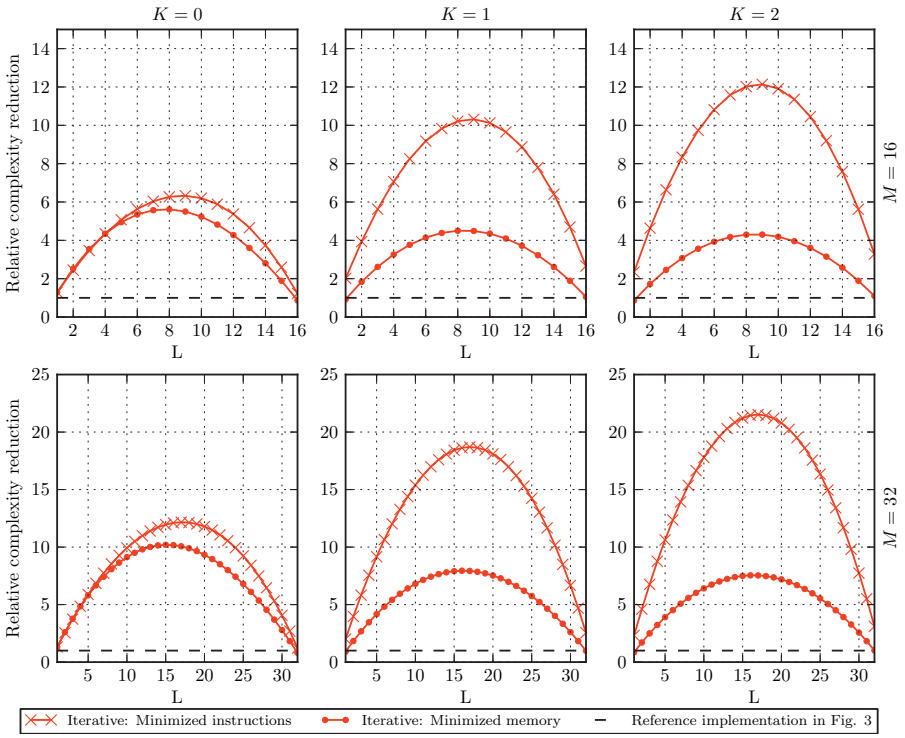


Figure I.5: Arithmetic optimization of computing $\hat{\boldsymbol{R}}$: Relative reduction in arithmetic complexity compared to the initial implementation shown in Fig. I.3 (higher is better). Note how the arithmetic count is reduced by a factor 4-10 in the memory optimized case, and by a factor 6-22 in the instruction optimized case.

be more arithmetically intensive than this. This can be inferred from Table I.1, where the peak bandwidth of the three types of GPU memory are compared to the peak arithmetic throughput (see appendix I.7.2 for derivations). Global memory (RAM), in which all data must reside at some point, is at best only able to move one float for every 30 floats processed by the CUDA cores, and potentially much worse if care is not taken to use "coalesced" memory access patterns (that avoid memory bank conflicts).

	$\mathbf{B_{arith}}$	$\mathbf{B_{mem}}$	$\mathbf{B_{mem}/B_{arith}}$
Arithmetic	1.03 Tflop/s		
Global memory		36 Gfloats/s	1:30
Shared memory		257 Gfloats/s	1:4
Registers		>1.5 Tfloats/s	>3:2 [25]

Table I.1: Nvidia Quadro 6000: Memory throughput, B_{mem}, compared to arithmetic throughput, B_{arith}.

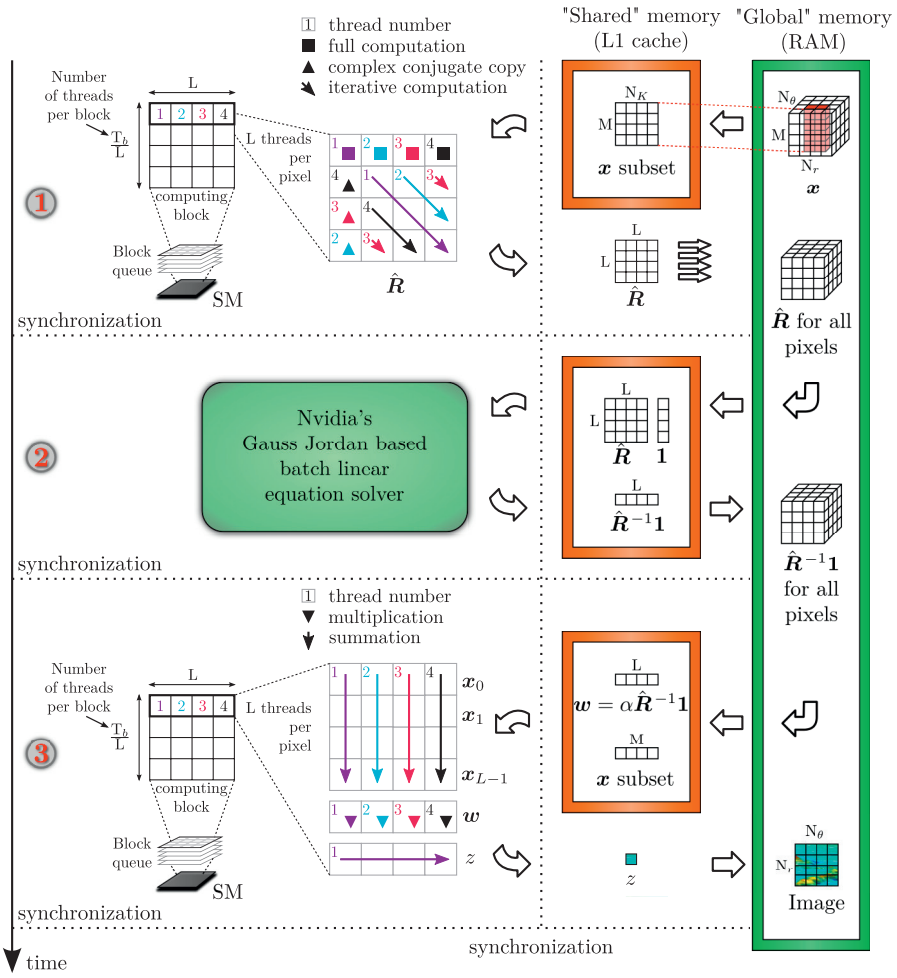


Figure I.6: MVDR implementated on a GPU. We do this in 3 steps, where each step process the full image before moving on to the next step. *Step 1*: The sample covariance matrix $\hat{\boldsymbol{R}}$ is formed by threads running along its diagonals. This allows spatial averaging to be implemented in a computationally efficient manner and minimizes inter-thread communication. *Step 2*: $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ is computed using the heavily optimized batched linear equation solver from Nvidia [24]. *Step 3*: The beamformer output z is computed in a straight forward fashion using L threads that first sum the subarrays and then apply the MVDR weighting function. A single thread finally sum all the channels up.

This is why the usage of global memory should be minimized. Shared memory, on the other hand, is a very fast level 1 cache that is shared by all threads on an SM. At peak performance it is almost able to keep up with the computing units, but to obtain this

performance we must make sure that:

1. the data needed to compute a pixel can fit in shared memory, while
2. the access patterns we use promote maximum bandwidth.

These challenges are very closely linked. Since the Quadro 6000 architecture is of compute capability 2.0, it has 48 kB shared memory (L1 cache) and 128 kB registers per streaming multiprocessor (SM) [26]. This memory is shared by all active threads on that SM, a number that should be no less than 768 [27]. This is to expose a sufficient level of data parallelism to ensure that memory latency is completely hidden (in accordance with Little's law). If we divide the shared memory evenly on all 768 threads we find that each thread should store no more than 8 single precision complex numbers in shared memory, and 24 stored in registers. This should make it apparent why computing a single pixel per thread is a bad idea, and why we need to keep each thread as light on memory consumption as possible.

In Fig. I.6 we illustrate how we get around these challenges. First we make each SM compute entire range lines of pixels, then load the subset of $\boldsymbol{x}$ that all these pixels depend upon into shared memory. This lets us perform temporal averaging without having to read additional channel data.

Second, we assign L threads per pixel to traverse the diagonals of $\hat{\boldsymbol{R}}$. On the top row a full computation is carried out, then that row is saved back to global memory following a coalesced access pattern that maximizes global memory bandwidth. For subsequent rows the threads move along the diagonals while performing iterative summations; the result from the previous element on the diagonal is updated by adding and removing the correlation coefficients that enters and exits the sum, respectively. To minimize memory consumption, we compute the coefficients again every time we need them. When a thread has finished up a diagonal, we have them wrap around to compute one of the diagonals in the lower triangular of $\hat{\boldsymbol{R}}$. Since $\hat{\boldsymbol{R}}$ is conjugate symmetric, the values in the leftmost column is obtained by a complex conjugate copy of the relevant value in the first row. This is slightly non-optimal as it causes divergent branching, but is needed because the solver is Gauss Jordan based and thus require a full $\hat{\boldsymbol{R}}$. Combined these steps balance the load evenly on all threads, is almost completely free of arithmetic redundancy, and consumes less memory.

I.3.2 Solving $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$

As intuition might suggest the matrix product $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ can be carried out by first inverting the matrix $\hat{\boldsymbol{R}}$, then performing the inner product with $\mathbf{1}$. However, one can also solve the linear equation $\hat{\boldsymbol{R}}\boldsymbol{b} = \mathbf{1}$ for $\boldsymbol{b}$, where $\boldsymbol{b} = \hat{\boldsymbol{R}}^{-1}\mathbf{1}$. This is the preferred approach since the solution is obtained directly and without the added computational cost of the final inner product.

An important thing to note, however, is that unlike the problems most GPU libraries tries to solve we do not attempt so solve large linear equations or invert large matrices; our matrices are small, but we have a very large number of them. Fortunately Nvidia recently released a highly optimized batched linear equation solver

tailored for this particular task. It is Gauss Jordan based and supports partial pivoting and complex numbers. The only downside to using it in our application is that it does not exploit the Hermitian property of our sample covariance matrix. A better choice would be a solver based on Cholesky decomposition. These are designed for Hermitian positive semi-definite matrices such as our covariance matrix, and require only half the arithmetic operations of a Gauss Jordan solver.

I.3.3 Computing z

The beamformer output z is computed on a per-pixel basis by first computing the MVDR weights $\boldsymbol{w}$, which are merely scaled versions of $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ (I.2). Then the sum in (I.6) is found by assigning a group of L threads to respective elements of x_l, which then proceed to accumulate these for all N_L subarrays. The resulting data vector is finally multiplied with the weight vector using the same threads, and then a single thread is used to sum these products to obtain z.

I.3.4 Implementation summary

The GPU design is summarized and compared to our CPU designs in table I.2.

	GPU	**CPU**
Pixel processing	In batches	One by one
Threads per pixel	Multiple	One
1. Building $\hat{\boldsymbol{R}}$:		
Values computed	All	Only upper triangular
Optimization	Minimized memory consumption	Minimized number of instructions
2. Computing $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$:		
Method	Nvidia's batch Gauss Jordan solver	Custom inline Cholesky solver
Cache utilization	Efficient	Not efficient
Data transfers	Minimum	Moderate

Table I.2: Implementation summary.

I.4 Images and Benchmarks

To demonstrate the imaging capability of the MVDR beamformer, we have processed experimental datasets from the $M = 32$ element Kongsberg Maritime HISAS1030 sonar mounted on a HUGIN AUV [28]. HISAS1030 is a high resolution synthetic aperture sonar with an array length of 1.2 m, operating frequency of 100 kHz, and bandwidth of 30 kHz. The element size and spacing is 2.5λ and the opening angle is 25°. To produce the image shown in Fig. I.7 the sonar was operated in sidescan mode. The studied object is the 1500 dwt oil tanker wreck Holmengraa. It is 68 m long and

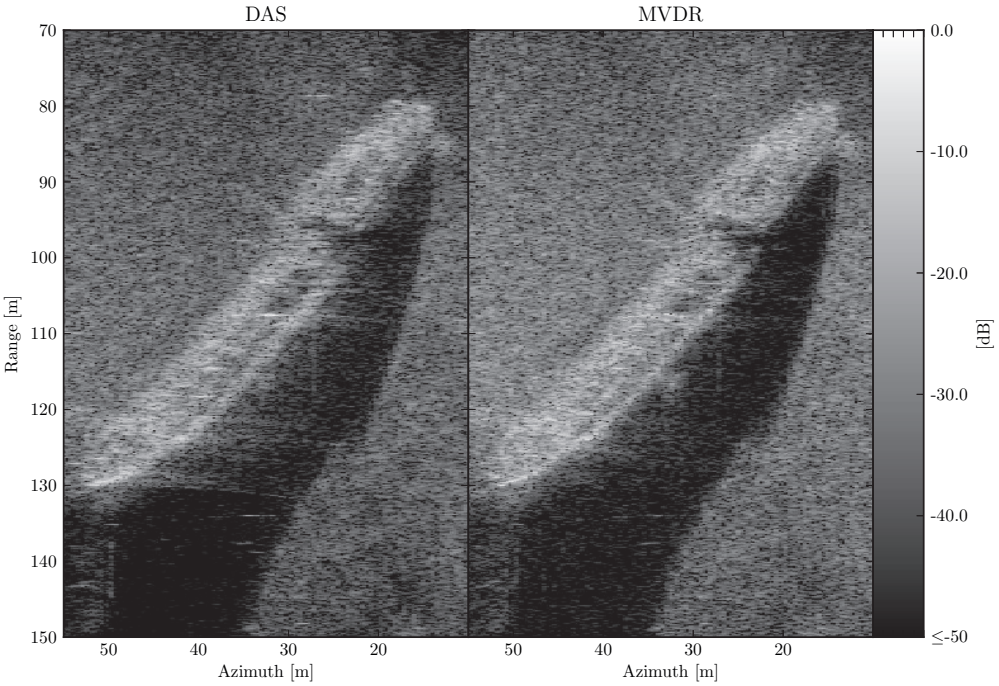


Figure I.7: HISAS sidescan sonar (SSS) image of the shipwreck Holmengraa that lies on a slanted seabed at 77 m depth outside of Horten, Norway. The image was processed with $M = 32$, $L = 16$, $K = 1$ and $d = 1\%$. The image was linearly upinterpolated by a factor 2 in azimuth making its total size 1.46 megapixels (MP). Note how MVDR improves edge definition and reduces noise in shadow regions.

9 m wide, and lies at a slanted seabed at 77 m depth outside of Horten, Norway. The 1 megapixel (MP) MVDR image were here processed with parameters $L = 16$, $K = 1$, and $d = 1\%$.

The computational performance of our implementation was first assessed on a test system with a quad-core Intel Xeon E5620, 64 GB of RAM and an Nvidia Quadro 6000 card. The results were obtained by processing a 1 MP image from the data from a 32 channel array, for all subarray sizes L, and for $K \in \{0, 1, 2\}$. Run-time measurements are shown in Fig. I.8, the run-times of memory-only and arithmetic-only GPU kernels are depicted in Fig. I.9, and an estimate of computation efficiency is presented in Fig. I.10. We were also granted a test drive at the Boston HPC centre on a machine with an Intel Xeon E5-2670, 32 GB RAM and an Nvidia K20. The results from this run are shown in Fig. I.11.

As seen in the run-time comparisons presented in Fig. I.8, the GPU method became 2-3 orders of magnitude faster than the C implemtation we started out with. The remaining bottleneck in the final design is the inversion step, which is typically 5 times slower than the build step. In most cases the processing speed of the Quadro

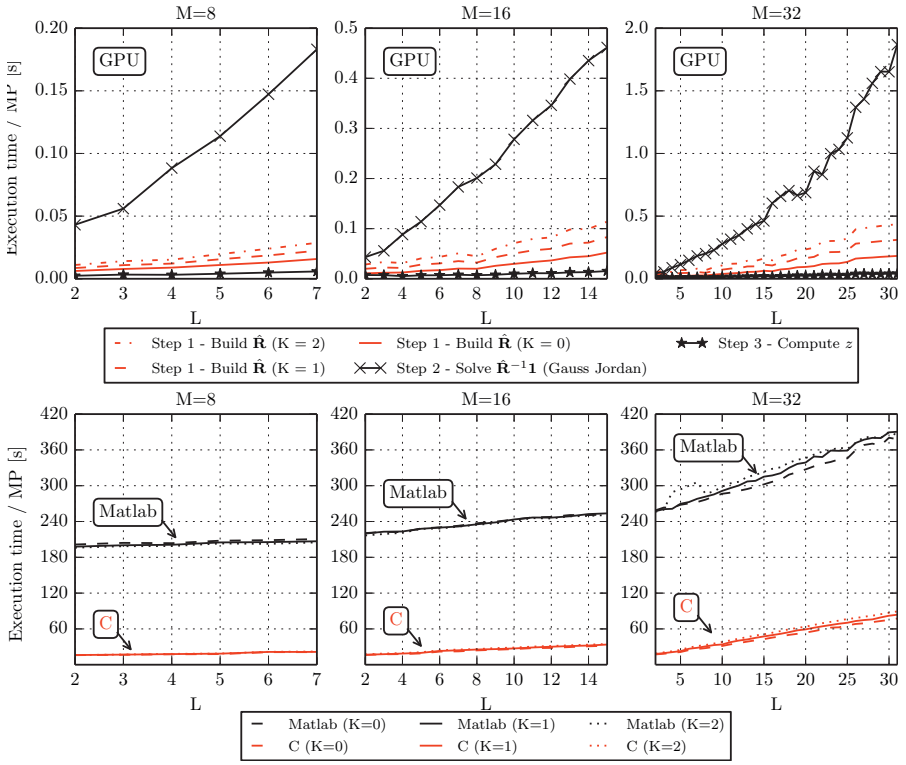


Figure I.8: MVDR benchmarks. A 1 MP image from a $M = 32$ channel array was processed for all L, and for $K \in \{0, 1, 2\}$.
Top: The time the GPU spent on building $\hat{\boldsymbol{R}}$, solving $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$, and computing z. Note the major speedup of building $\hat{\boldsymbol{R}}$ when compared to the complexity plot in Fig. I.3.
Bottom: Compared to a reference MATLAB and single thread C implementation running on a CPU the GPU offered a speedup of 2-3 orders of magnitude, but these numbers are somewhat misleading. If the C implementation was properly optimized we expect the GPU to be no more than a factor 5-10 faster, even if its theoretical peak performance is 20 times higher than that of the CPU.

6000 is above 1 MP/s, and this was only improved by a factor 1.5-2 when run on the new K20 GPU. The benchmarks of our memory-only and arithmetic-only kernels show that the kernels spend roughly the same time on both these tasks, so optimising only one of these further will have marginal impact. All GPU kernels were compiled with `nvcc` at optimization level `O2`. Excluded from these benchmarks is the data transfer time from CPU to GPU, which account for 2-20% of the total processing time. We believe it makes little sense to include them since it keeps getting easier to perform these data transfers in parallel by offloading the task to the Direct Memory Access (DMA) controllers present on modern GPUs. Furthermore, the data rates in active

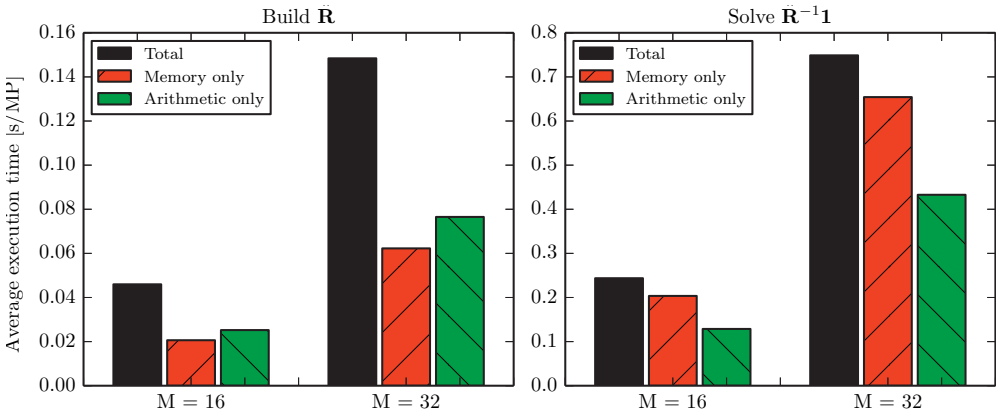


Figure I.9: Execution time of an arithmetic-only and a memory-only version of the MVDR code. A dataset from an $M = 32$ array was processed for all L using $K = 1$, and the mean execution time for a 1 megapixel (MP) image was used here. From this plot we can infer that the kernel building $\hat{\boldsymbol{R}}$ is memory bound, as the time the kernel spends performing memory transactions is higher than the corresponding time it spends carrying out arithmetical operations. Furthermore, when the total runtime is larger than the restricted kernels this can largely be attributed to latency, which we can see that building $\hat{\boldsymbol{R}}$ suffers from with the chosen parameters.

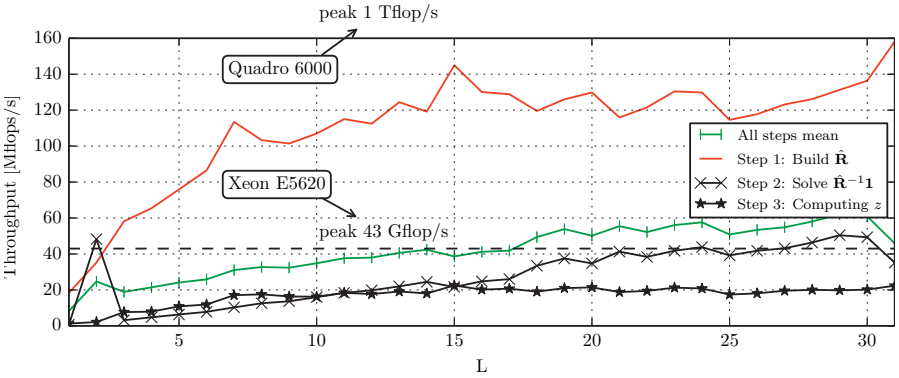


Figure I.10: Code efficiency. An estimate of the number of floating point operations per second (Flop/s), found by dividing the theoretical complexity curves by actual run-times. This is a crude measure as it does not include any other instructions than the actual arithmetic operations in the MVDR computation.

sonar are relatively low compared to the bandwidth available for these transfers.

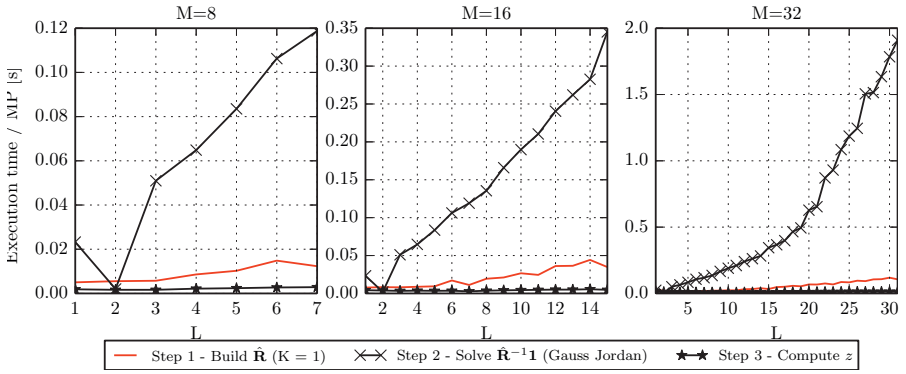


Figure I.11: MVDR benchmarks from Boston HPC centre with the new high-end Nvidia K20 Kepler GPU. The exact same scenario and code as in Fig. I.8 was used here. With no code alterations the performance was only improved marginally compared to running on the Quadro 6000.

I.5 Discussion

In accordance with previous studies, Fig. I.7 demonstrates the MVDR beamformer's ability to produce images with suppressed interference and improved detail resolution. Compared to the DAS beamformer, the ship's edges appear sharper and the shadows less noisy. In this scenario the MVDR's performance was not particularly sensitive to parameter adjustments. Similar performance was obtained with arbitrary combinations of $L \in \{12, 16, 24\}$, $K \in \{1, 2, 3\}$ and $d \in [0.01, 0.05]$, and no adjustments had to be made when processing other parts of the scene.

As observed in Fig. I.8 the combination of minimizing arithmetic operations and implementing the MVDR on a GPU lead to a significant improvement in processing speed. Compared to a MATLAB and single thread C implementation an order 2 and 3 speedup was achieved, respectively. Note that we do not consider this comparison fair. In fact, the theoretical peak throughput of this GPU is roughly 20 times higher than that the CPU in question, meaning the potential of the CPU was far from reached in our initial implementations. The key reason for this is that the GPU design process pixels in batches, which allows us to minimize data transfers and maximize the use of fast memory cache. In other aspects the designs are similar, both CPU implementations compute $\hat{\boldsymbol{R}}$ efficiently, and they make use of either an optimized custom Cholesky based solver or one from the Intel Math Kernel Library (MKL). Unfortunately we had little time to write a custom batch based solver and covariance builder for the CPU, but we expect the speed difference to be 5-10 times if we had compared two such designs.

Note from Fig. I.8 how the building of $\hat{\boldsymbol{R}}$ now only takes up a fraction of the total processing time, and recall that it was the other way around when we presented the theoretical complexity curves in Fig. I.3. Also, the benchmark curves for building $\hat{\boldsymbol{R}}$ now seem linear. The main reason for this is that the optimization step negated

much of the extra complexity introduced by averaging, i.e. reduced the complexity from $\mathrm{O}(N_K N_L L^2)$ to less than $\mathrm{O}(L^2)$. Another reason for the run-time to appear more like $\mathrm{O}(L)$ is likely that our design makes better use of the GPU's resources when there is more processing involved per pixel. The inversion step, on the other hand, gains less from being implemented on a GPU. This is because its nature is less data parallel. In particular, the back substitution step involved in its computation is mostly sequential. This difference can be observed in Fig. I.3. Alternative solvers, such as one based on Cholesky decomposition that exploits the Hermitian property of $\hat{\boldsymbol{R}}$, can in theory reduce complexity by a factor two, but we question whether this result can be obtained in practice using a GPU. The CPU implementations perform Cholesky based inversion, but this does not explain why the C and MATLAB implementation have a total run-time that is linearly dependent on L. We believe this happens because these implementations are dominated by the calculation of the covariance matrix and data movement, and not by the inversion step.

When optimizing a GPU design it is important to know whether it is limited by memory bandwidth or arithmetical throughput. To measure this we made two versions of the MVDR kernel, one that only performs arithmetic operations, and one that only performs memory operations. Then we benchmarked these kernels and compared their run-times to the total run-time (Fig. I.9). A first thing to note is how these kernels seem equally occupied performing memory and arithmetic operations. This is a good sign, but since we consistently minimized memory consumption at the expense of some extra computations when building $\hat{\boldsymbol{R}}$, we know it to be bound by memory bandwidth. It also has a problem with latency, which can be inferred from the total run-time being significantly higher than that of the two single-function kernels. Since the GPU hardware can carry out memory and arithmetic operations concurrently and independently, this gap indicates that the GPU sometimes does "nothing". In our case this is due to synchronization hold-ups when performing temporal averaging, which is carried out in a sequential manner.

Even with all our efforts we were only able to obtain processing rates of 40 GFlop/s on average in the desirable range of subarray sizes (Fig. I.3). This is mainly due to the inversion step, since we for the most part obtained more than 100 GFlop/s when building $\hat{\boldsymbol{R}}$. While these numbers are slightly underestimated, they are regardless far away from the Quadro 6000's peak of 1 TFlop/s. Again, we believe this is due to the design being bound by memory bandwidth. This belief is further supported by the test results from running the code on the new K20 Kepler card, where we saw a modest factor 1.5-2 speedup. Although the Kepler card peaks at 3.5-4 Tflop/s, its shared memory bandwidth is only approximately twice that of the Quadro 6000, hence matches the observed speed-up. However, it is likely that the Kepler card would perform better had we optimized our design for it. For scientific computing the Kepler boards are considered more attractive, with the introduction of features such as kernels calling kernels, more registers, more shared memory, and better mechanisms for hiding or removing data transfers.

I.6 Conclusion

The MVDR beamformer is an algorithm capable of producing images with improved detail resolution and contrast compared to conventional DAS beamforming. The downside is its inherent need for robustification and the high computational complexity associated with estimating and inverting the spatial covariance matrix.

We have shown that for systems consisting of up to 32 channels the problem can be largely mitigated by building the covariance matrix in a clever way, and by making use of the massive computational power available in modern GPUs. We were able to improve upon the run-time of a single thread C-implementation by roughly two orders of magnitude. For most choices of parameters the GPU was able to create images at $\sim$1 MP/s at an average data processing rate of $\sim$40 GFlop/s. This is less than 5% of the peak performance of the GPU, but we believe it to be near optimal given the constraints of memory bandwidth and the sequential nature of some parts of the MVDR implementation.

All in all, the MVDR maps well to the GPU since the computations involved are independent on the pixel level, and partially also within each pixel. The GPU allows MVDR to be used in real-time processing of sonar data, and makes the MVDR a viable alternative to conventional methods in practical systems.

I.7 Appendices

I.7.1 MVDR Complexity Formulas

To estimate the number of floating point operations needed to MVDR beamform a single pixel, we formed expressions that accumulate the number of complex arithmetic operations found in the MVDR process. From observing the generated assembly code we inferred that each complex addition and multiplication would require $O_\mathrm{a} = 2$ and $O_\mathrm{m} = 6$ floating point operations, respectively. The formulas are listed for each step below for reference.

Building $\hat{\boldsymbol{R}}$. The initial complexity of this step can be inferred directly from (I.4) and (I.5):

$$O_{\substack{\mathrm{Build}\,\hat{\boldsymbol{R}} \\ \mathrm{initial}}} = \underbrace{O_\mathrm{m} N_\mathrm{k} N_\mathrm{l} L^2}_{\text{Multiplications}} + \underbrace{O_\mathrm{a}(N_\mathrm{k} + N_\mathrm{l} - 2)L^2}_{\text{Additions}} + O_\mathrm{dload}, \tag{I.7}$$

where $O_\mathrm{dload} = (2L-1)O_\mathrm{a} + O_\mathrm{m}$ is the minor cost of performing diagonal loading. If we apply the optimization strategies discussed in section I.3.1 we can arrive at the

following instead:

$$O_{\substack{\text{Build } \hat{\boldsymbol{R}} \\ \text{min arith}}} = \underbrace{O_{\text{m}} \frac{M + N_{\text{l}}}{2} L}_{\text{Multiplications}} + \underbrace{O_{\text{a}}(N_{\text{k}} - 1)(N_{\text{l}} - 1)L}_{\text{First row additions}} + \underbrace{\frac{(L-1)(L-2)}{2} \Big[2O_{\text{a}} + 2(N_{\text{k}} - 1)O_{\text{a}} \Big]}_{\text{Iteration additions}} + O_{\text{dload}} \tag{I.8}$$

Of the solutions discussed, this is the least expensive in terms of arithmetic instructions. However, if memory bandwidth is a limiting factor a better solution is to recompute multiplications where they are needed:

$$O_{\substack{\text{Build } \hat{\boldsymbol{R}} \\ \text{min mem}}} = \underbrace{O_{\text{m}} N_{\text{k}} N_{\text{l}} L}_{\text{First row multiplications}} + \underbrace{O_{a}(N_{\text{k}} - 1)(N_{\text{l}} - 1)L}_{\text{First row additions}} + \underbrace{\frac{(L-1)(L-2)}{2} \Big[2O_{\text{a}} + 2(N_{\text{k}} - 1)O_{\text{a}} + 2N_{\text{k}} O_{\text{m}} \Big]}_{\text{Iteration multiplications and additions}} + O_{\text{dload}}. \tag{I.9}$$

Solving $\hat{\boldsymbol{R}}^{-1}\mathbf{1}$ is achieved by using a batched Gauss Jordan solver with support for complex numbers and partial pivoting. Its complexity - with partial pivoting excluded - can be expressed as:

$$O_{\text{Solve } \hat{\boldsymbol{R}}^{-1}\mathbf{1}} = \sum_{r=0}^{L} \Bigg[\underbrace{\big(L - r\big)\big((L - r + 2)O_{\text{a}} + (L - r + 3)O_{\text{m}}\big)}_{\text{Reduction}} + \underbrace{(r-1)O_{\text{a}} + rO_{\text{m}}}_{\text{Backsubstitution}} \Bigg], \tag{I.10}$$

where r is a running variable r that indexes rows in the augmented matrix $\big[\hat{\boldsymbol{R}}|\mathbf{1}\big]$.

Computing z is very simple once the covariance matrix $\hat{\boldsymbol{R}}$ is built and inverted, and has an near negligible impact on performance:

$$O_{\text{Compute } z} = O_{\text{a}}(2L - 2)2 + O_{\text{m}}3L. \tag{I.11}$$

I.7.2 GPU Throughput

In the context of determining whether an implementation is computationally bound or memory bound, one should first compare the target platform's sustained computational throughput to sustained memory throughput. Let us start with the former.

The Quadro 6000 has 32 CUDA cores per SM, each operating at a rate of 1148 MHz and being able to perform 2 floating point operations (flop) per clock cycle when multiply-add instructions are used. The theoretical peak arithmetic throughput is then given as:

$$\begin{aligned} \mathrm{B}_{\mathrm{arith}} &= 2 \cdot 1148 \text{ Mflop/s/core} \cdot 32 \text{ cores/SM} \cdot 14 \text{ SMs} \\ &= 1.03 \text{ Tflop/s}. \end{aligned} \tag{I.12}$$

Now let us compare this to the memory throughput. The "global" GDDR5 memory bus is 384 bit wide, and operates at 3 Ghz where 2 bits are sent every cycle. Its peak bandwidth is then:

$$\begin{aligned} \mathrm{B}_{\mathrm{gmem}} &= \frac{2 \cdot 3 \text{ Gbit/s} \cdot 384 \text{ bit}}{8 \text{ bit/byte}} \\ &= 144 \text{ GB/s } (36 \text{ Gfloats/s}). \end{aligned} \tag{I.13}$$

The shared memory, on the other hand, is organized into 32 banks per SM, each being 32 bit wide and operating at 1148 MHz where 1 bit is sent per cycle. Its peak aggregated bandwidth is then:

$$\begin{aligned} \mathrm{B}_{\mathrm{smem}} &= \frac{\frac{1148}{2} \text{ Mbit/s} \cdot 32 \text{ bit/bank} \cdot 32 \text{ banks/SM} \cdot 14 \text{ SMs}}{8 \text{ bit/byte}} \\ &\approx 1.03 \text{ TB/s } (257 \text{ Gfloats/s}). \end{aligned} \tag{I.14}$$

The bandwidths are compared in Tab. I.1. Note that even when using shared memory at least 4 floating point operations must be carried out per float transferred to the CUDA cores, otherwise the algorithm will be memory bound and the peak arithmetic throughput can not be reached.

Acknowledgment

The authors would like to thank Kongsberg Maritime and the Norwegian Defence Research Establishment (FFI) for providing experimental data, and thank Nvidia for providing support for running the batched linear equation solver as well as for granting us a testdrive at the Boston HPC center.

References

[1] G. R. Benitz, "High-Definition Vector Imaging," *Lincoln Laboratory Journal*, vol. 10, no. 2, pp. 147–170, 1997.

[2] J.-F. Synnevag, A. Austeng, and S. Holm, "Adaptive beamforming applied to medical ultrasound imaging." *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 54, no. 8, pp. 1606–13, Aug. 2007.

[3] A. E. A. Blomberg, A. Austeng, and R. E. Hansen, "Adaptive Beamforming Applied to a Cylindrical Sonar Array Using an Interpolated Array Transformation," *IEEE Journal of Oceanic Engineering*, vol. 37, no. 1, pp. 25–34, Jan. 2012.

[4] A. E. A. Blomberg, R. E. Hansen, S. A. V. Synnes, and A. Austeng, "Improved interferometric sonar performance in shallow water using adaptive beamforming," in *Proceedings of the International Conference & Exhibition on Underwater Acoustic Measurements (UAM), Kos, Greece, June*, 2011.

[5] S. Dursun, A. Austeng, R. E. Hansen, and S. Holm, "Minimum variance beamforming in active sonar imaging," in *Proceedings of the 3rd International Conference & Exhibition on Underwater Acoustic Measurements: Technologies and Results*, B. r. e. John S. Papadakis & Leif, Ed., 2009, pp. 1373–1378.

[6] K. Lo, "Adaptive Array Processing for Wide-Band Active Sonars," *IEEE Journal of Oceanic Engineering*, vol. 29, no. 3, pp. 837–846, Jul. 2004.

[7] B. Widrow, K. Duvall, R. Gooch, and W. Newman, "Signal cancellation phenomena in adaptive antennas: Causes and cures," *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 3, pp. 469–478, May 1982.

[8] H. L. Van Trees, *Optimum Array Processing.* New York, USA: John Wiley & Sons, Inc., Mar. 2002.

[9] T. Kailath and T.-J. Shan, "Adaptive beamforming for coherent signals and interference," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, no. 3, pp. 527–536, Jun. 1985.

[10] B. M. Asl and A. Mahloojifar, "A low-complexity adaptive beamformer for ultrasound imaging using structured covariance matrix." *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 59, no. 4, pp. 660–7, Apr. 2012.

[11] A. Jakobsson, S. Marple, and P. Stoica, "Computationally efficient two-dimensional Capon spectrum analysis," *IEEE Transactions on Signal Processing*, vol. 48, no. 9, pp. 2651–2661, 2000.

[12] C.-I. C. Nilsen and I. Hafizovic, "Digital beamforming using a GPU," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, Apr. 2009, pp. 609–612.

[13] J. Chen, Y. Yiu, and H. So, "Real-time GPU-based adaptive beamformer for high quality ultrasound imaging," *IEEE Ultrasonics Symposium*, vol. 1, no. 1, pp. 1–4, 2011.

[14] J. Chen, B. Y. Yiu, B. K. Hamilton, A. C. Yu, and H. K.-H. So, "Design space exploration of adaptive beamforming acceleration for bedside and portable medical ultrasound imaging," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, p. 20, Dec. 2011.

[15] D. Chapman, "Partial adaptivity for the large array," *IEEE Transactions on Antennas and Propagation*, vol. 24, no. 5, pp. 685–696, Sep. 1976.

[16] J. P. Åsen, J. I. Buskenes, C.-I. C. Nilsen, A. Austeng, and S. Holm, "Implementing Capon Beamforming on the GPU for Real Time Cardiac Ultrasound Imaging," in *Proceedings IEEE Ultrasonics Symposium*, 2012.

[17] J. P. Åsen, J. I. Buskenes, C.-I. Colombo Nilsen, A. Austeng, and S. Holm, "Implementing capon beamforming on a GPU for real-time cardiac ultrasound imaging." *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 61, no. 1, pp. 76–85, Jan. 2014.

[18] J. I. Buskenes, J. P. Åsen, C.-I. C. Nilsen, and A. Austeng, "Adapting the minimum variance beamformer to a graphics processing unit for active sonar imaging systems," *The Journal of the Acoustical Society of America*, vol. 133, no. 5, p. 3613, 2013.

[19] F. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[20] J. Capon, "High-resolution frequency-wavenumber spectrum analysis," *Proceedings of the IEEE*, vol. 57, no. 8, pp. 1408–1418, 1969.

[21] J.-F. Synnevag, A. Austeng, and S. Holm, "Benefits of minimum-variance beamforming in medical ultrasound imaging," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 56, no. 9, pp. 1868–1879, Sep. 2009.

[22] H. Cox, R. Zeskind, and M. Owen, "Robust adaptive beamforming," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 10, pp. 1365–1376, Oct. 1987.

[23] J. N. Maksym, "A robust formulation of an optimum cross-spectral beamformer for line arrays," *The Journal of the Acoustical Society of America*, vol. 65, no. 4, p. 971, 1979.

[24] Nvidia, "Nvidia registered developers program." [Online]. Available: https://developer.nvidia.com/registered-developer-programs

[25] V. Volkov, "GTC: Better Performance at Lower Occupancy," 2010. [Online]. Available: http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf

[26] Nvidia, *CUDA C Programming Guide v4.2*, 2012. [Online]. Available: http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation

[27] ——, *CUDA C Best Practices Guide v4.1*, 2012. [Online]. Available: http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation

[28] R. E. Hansen, H. J. Callow, T. O. Sæ bø, S. A. Synnes, P. E. Hagen, G. Fossum, and B. Langli, "Synthetic aperture sonar in challenging environments: Results from the HISAS 1030." in *Proceedings of the 3rd International Conference & Exhibition on Underwater Acoustic Measurements: Technologies and Results*, 2009.

Paper II

Implementing Capon Beamforming on a GPU for Real-Time Cardiac Ultrasound Imaging

Jon Petter Åsen[1], Jo Inge Buskenes[2], Carl-Inge C. Nilsen[2], Andreas Austeng[2] and Sverre Holm[1,2]
[1]Medical Imaging Lab (MI-Lab), Norwegian University of Science and Technology, Trondheim, Norway
[2]Department of Informatics, University of Oslo, Oslo, Norway

IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 61, pp. 76-85, Jan 2014.

Capon beamforming is associated with a high computational complexity, which limits its use as a real-time method in many applications. In this paper we present an implementation of the Capon beamformer that exhibits real-time performance when applied in a typical cardiac ultrasound imaging setting. To achieve this performance we make use of the parallel processing power found in modern Graphics Processing Units (GPUs), combined with beamspace processing to reduce the computational complexity as the number of array elements increases.
For a three dimensional beamspace we show that processing rates supporting real-time cardiac ultrasound imaging are possible, meaning that images can be processed faster than the image acquisition rate for a wide range of parameters. Image quality is investigated in an *in-vivo* cardiac dataset. These results show that Capon beamforming is feasible for cardiac ultrasound imaging, providing images with improved lateral resolution both in element-space and beamspace.

II.1 Introduction

Capon beamforming [1] has recently been applied to medical ultrasound imaging with promising results in several studies [2–6]. Most of this work focuses on the improved lateral resolution and contrast obtained with the Capon beamformer. However, some interesting trade-offs have also been introduced. In [7], Synnevåg *et al.* explains how image resolution can be maintained at higher framerates, with smaller probes, or for deeper penetration. Despite these benefits, widespread adoption of the Capon beamformer has not been seen due to its high computational complexity [8]. In this paper we present an implementation of the Capon beamformer exhibiting real-time performance when applied in a typical cardiac ultrasound imaging setting. To achieve this performance we make use of the parallel processing power found in modern Graphics Processing Units (GPUs) (found in almost all PCs), combined with novel methods to reduce the computational complexity as the number of array elements increases.

Previous work on parallel implementations of the Capon beamformer have been focused on radar and passive sonar systems using custom made systolic array processors [9–11]. For medical ultrasound, Chen *et al.* [12–14] implemented a Capon beamformer on the GPU that produces real apodization weights based on real data. However, the Capon beamformer is meant to process complex data. Under the constraint of using real data, the Capon beamformer is not allowed to micro-steer the main lobe, and therefore its full potential is not reached. Micro-steering of the main lobe is an important property of the Capon beamformer in order for it to improve structural details and edge definitions in an ultrasound image [7, Fig 9.]. Complex arithmetic leads to more operations per sample, but also allows the sampling frequency to be reduced. The number of samples needed per image can therefore be minimized. All results in this paper are reported with the use of baseband (or I/Q) data, where the sampling frequency is near the system bandwidth.

In recent work we have applied a GPU-based Capon beamformer for both active

sonar imaging [15, 16] and cardiac ultrasound imaging [17]. Sonar has a less strict real-time requirement than medical ultrasound imaging, which means that most array configurations can be handled in real time using the GPU. In [17] and in this paper we show that Capon beamforming applied in medical ultrasound imaging, and in particular cardiac ultrasound imaging, demands frame rates that not even a cluster of high-end GPUs can provide. Since the execution time of Capon beamforming is cubic with respect to the number of array elements, we need methods that reduce the computational complexity for large arrays. Various approaches to low-complexity Capon beamforming for medical ultrasound imaging have been proposed in the literature [18–21]. In this paper we extend the work done in [17] and [16] by utilizing a beamspace transformation. The beamspace transform has been proposed used in ultrasound imaging by Nilsen and Hafizovic [22] in order to maintain high frame rates as the number of elements increases while maintaining almost the same lateral resolution and sidelobe reduction capabilities of element-space Capon when applied in narrow-beam systems.

In the next section we present background information on Capon beamforming in both element-space and beamspace. We also give an introduction to the GPU computing model. In Section II.3 we give a summary of our parallel implementation of the element-space Capon beamformer, before the GPU implementation of the beamspace Capon beamformer is presented in Section II.4. Benchmark results and resulting images for both implementations are presented in Section II.5 and II.6, together with a discussion of trading image quality for speed in Section II.7. Finally we discuss the results in Section II.8, and draw conclusions in Section II.9.

II.2 Background

II.2.1 Capon Beamforming

The conventional form of array beamforming in medical ultrasound imaging is delay-and-sum (DAS). Here, the signal recorded at element number $m \in [0, M-1]$ at time n is delayed with an appropriate delay $\Delta_m[n]$ to focus and steer the ultrasound beam. This delayed data is here denoted $x_m[n]$. The beamformer output $z[n]$ is then the sum of all M elements weighted with a factor w_m,

$$z[n] = \sum_{m=0}^{M-1} w_m^* x_m[n] = \mathbf{w}^H \mathbf{x}[n], \tag{II.1}$$

where $(\cdot)^H$ is the Hermitian transpose. The weight vector, or window, $\mathbf{w}$ is usually real in DAS beamforming, and is applied to trade resolution for a lower side lobe level. Although different windows can be used in different areas of the images, $\mathbf{w}$ is usually predefined.

As shown in Fig. II.1, the basic idea of an adaptive beamformer is to form a complex weight vector based on the received data. The Capon beamformer produces weights that minimize the beamformer output power while maintaining unity gain in the focus direction. The result is that interference impinging from other directions

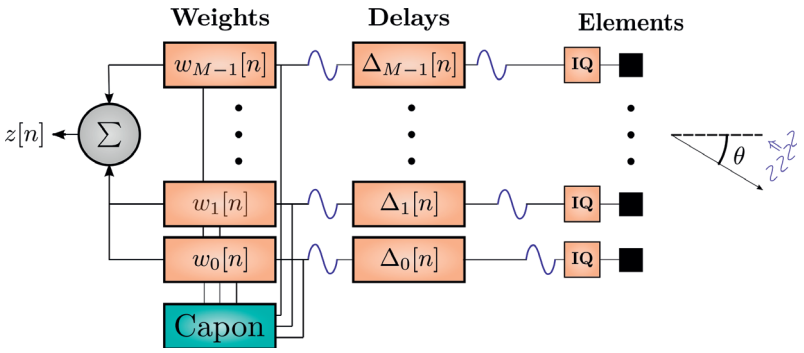


Figure II.1: Capon beamforming. The impinging signal is demodulated, and aligned in phase by applying steering and focusing delays $\Delta_m[n]$. Adaptive weights, $\mathbf{w}[n]$, are then calculated based on in-phase data, and finally the output is formed by a Capon-weighted coherent sum.

is suppressed [2]. As with coarse-fine beamformers [23], the complex weights cause a micro-steering which is minute compared to the initial steering and focusing step, therefore the beamformer also works well for broadband ultrasound signals.

The Capon beamformer, following the approach in [7], can be divided into three main steps: estimation of a sample covariance matrix, solving a system of linear equations, and calculation of the beamformer output. The sample covariance matrix is estimated as

$$\breve{\mathbf{R}}[n] = \frac{1}{N_L N_K} \sum_{n'=n-K}^{n+K} \sum_{l=0}^{N_L-1} \mathbf{x}_l[n'] \mathbf{x}_l[n']^H, \tag{II.2}$$

where $N_K = 2K + 1$ is the number of temporal samples included, $\mathbf{x}_l = [x_l[n], \ldots, x_{l+L-1}[n]]$ is the lth subarray of length L, and $N_L = M - L + 1$ is the number of subarrays. Note that the covariance matrix dimension is $L \times L$. The covariance matrix is loaded with a diagonal factor ϵ to ensure numerical stability and increased robustness:

$$\hat{\mathbf{R}}[n] = \breve{\mathbf{R}}[n] + \epsilon[n]\mathbf{I} = f(\mathbf{x}[n]). \tag{II.3}$$

The operator $f(\cdot)$ represents the process of estimating $\hat{\mathbf{R}}$ from the input $\mathbf{x}$. A factor proportional to the output power is often used to reduce the need for parameter adjustments, and the trace of $\breve{\mathbf{R}}$,

$$\epsilon[n] = d \times \frac{\text{trace}\{\breve{\mathbf{R}}[n]\}}{L}, \tag{II.4}$$

has been applied in much of the recent literature on Capon beamforming for medical ultrasound imaging [2, 6, 22, 24]. It has also been applied to HF and VHF antenna processing [25].

Next, a linear system of equations has to be solved,

$$\mathbf{b}[n] = \hat{\mathbf{R}}[n]^{-1}\mathbf{a} \in \mathbb{C}^L, \tag{II.5}$$

where the steering vector $\mathbf{a} = \mathbf{1}_L = [1_0, 1_1, ..., 1_{L-1}]^T$ since $\mathbf{x}$ is pre-delayed.

Finally the Capon weight vector is formed from (II.5) as

$$\mathbf{w}[n] = \frac{\mathbf{b}[n]}{\mathbf{1}^H\mathbf{b}[n]} \in \mathbb{C}^L, \tag{II.6}$$

and the beamformer output is calculated by combining all N_L subarrays weighted with the same set of adaptive weights. This is formally stated as

$$z[n] = \frac{1}{N_L}\mathbf{w}[n]^H \sum_{l=0}^{N_L-1} \mathbf{x}_l[n] \tag{II.7}$$

$$= 1/N_L \, (\mathbf{w}[n] * \mathbf{1}_{N_L})^H \mathbf{x}[n], \tag{II.8}$$

where $*$ is a non-truncated convolution across elements. Note how we can either choose to sum all subarrays (II.7), or accumulate a weighting per element (II.8).

In passive systems, assuming stationary statistics, the sample covariance matrix in (II.2) is typically averaged over a long sequence of samples[26]. Since medical ultrasound imaging is an active, short-pulsed system, there is a limited amount of data from which $\hat{\mathbf{R}}$ can be estimated. The estimated matrix is therefore often found to be both inaccurate and poorly conditioned. Another challenge with active systems is the high correlation between signal and interference, resulting in signal cancellation [27]. Thus, the beamformer output can be lower than expected, even when there is signal present and the distortionless-response criterion is applied.

In order to get a well-conditioned $\hat{\mathbf{R}}$, avoid signal cancellation, and to get DAS-like speckle, $\hat{\mathbf{R}}$ has to be averaged over $L \leq M/2$ long subarrays and $N_K \sim T_p/T_s$ time samples [28], where, T_p is the pulse length in seconds and T_s is the sampling period. For our baseband data, one pulse length (1.5λ) is around 3 samples ($K = 1$). Typical values for d is between 1/10 and 1/100.

II.2.2 Beamspace Processing

Nilsen and Hafizovic [22] have proposed to apply beamspace processing to reduce the required computations of the Capon beamformer when applied to medical ultrasound imaging. Beamspace is an alternative representation of the per-element data, transformed into a fan of beams covering the sector illuminated by transmit beams. The transformation is formally stated as

$$\mathbf{x}_{\mathrm{BS}}[n] = \mathbf{B}\mathbf{x}[n], \qquad \mathbf{B} \in \mathbb{C}^{N,M} \tag{II.9}$$

$$[\mathbf{B}]_{p,q} = \frac{1}{\sqrt{M}} e^{-j2\pi pq/M} \quad N \leq M. \tag{II.10}$$

The steering vectors in $\mathbf{B}$, the so-called Butler or normalized discrete fourier transform (DFT) matrix, determine each beam's direction in space. Beams where no interference is present can therefore be removed by simply removing rows from $\mathbf{B}$.

For an imaging system using narrow transmit and receive beams, like cardiac ultrasound imaging, the received signal will be concentrated in a narrow band in wavenumber space. Hence, a small percentage of the beams will contain almost all the received energy. Nilsen and Hafizovic concluded that as little as three beams around (and including) the transmit direction were adequate to produce results comparable to applying Capon beamforming in element space. Hence, for equally pitched arrays the inversion steps is found to be constant with respect to the number of array elements.

The beamspace version of the Capon beamformer is derived by replacing the covariance matrix in (II.6) with the beamspace covariance matrix

$$\hat{\mathbf{R}}_{\mathrm{BS}} = f(\mathbf{x}_{\mathrm{BS}}) = f(\mathbf{B}\mathbf{x}), \tag{II.11}$$

where $\mathbf{B} \in \mathbb{C}^{N_b,L}$, N_b is the number of selected beams, and $f(\cdot)$ represent the covariance estimation process as defined in (II.3).

In the remaining sections, the methods described in this section and Section II.2.1 will be referred to as beamspace Capon (BS-Capon) and element-space Capon (ES-Capon) respectively. The name Capon will refer to both methods.

II.2.3 GPU Compute Model

In this section we give a brief discussion of GPU computing, laying the basis for our discussions on parallel implementation strategies. The Capon beamformers have been implemented using the CUDA GPU framework by Nvidia [29], however the challenges and proposed methods described in this paper are valid for other GPU framworks as well, like e.g. OpenCL. Independent of framework, the key point is that GPUs are designed to execute large sets of small concurrent tasks, like beamforming a pixel in a large image, much faster than a CPU. While the CPU has few cores, but logic to process complex threads, the GPU has many cores, but each core needs many (preferably simple) threads to keep it occupied and by that achieve maximum throughput.

In CUDA programming a kernel function is executed over a large set of threads, where the kernel function describes the work to be done in each thread. As depicted in Fig. II.2, threads are organized in a grid of thread-blocks, with maximal size of 1024 threads per block (e.g. 32×32). Threads inside a block support low-cost synchronization if needed. From a CUDA perspective the GPU consist of one or more streaming multi-processors (SM), where each SM is capable of executing multiple arithmetic operations per clock cycle.

An SM has a limited amount of shared memory (or near-core cache) and registers. Therefore, to achieve maximum GPU occupancy (defined as having the maximum number of threads residing on each SM), a conservative use of registers and shared memory per thread is required. Each thread should perform enough instructions per transferred byte in order to hide memory latency, and the problem should be divided into enough threads in order to hide instruction latency. Global GPU memory

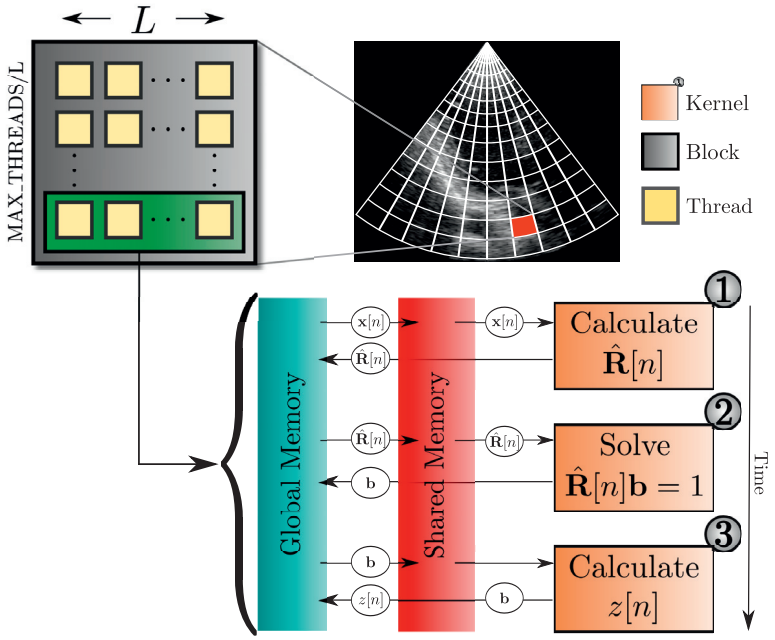


Figure II.2: Calculation of Capon adaptive weights mapped to GPU architecture. A recorded echo across the array for a given range and angle (selected cell) is assigned to a group of L threads that can run independently of all other groups. Note that several samples can be processed per thread-block if L is small. The task of these L threads is to calculate the Capon beamformer output in the three depicted steps.

transactions, in addition to CPU-GPU transfers, are slow and must be minimized. All of these constraints combined pose a real challenge as the target problem has to map well to the GPU framework on both a grid, block, thread, and memory consumption level in order to benefit from parallel acceleration. In the next two sections, we will describe how this mapping can be done for the Capon beamformers.

II.3 Parallel ES-Capon

In this section we will give a brief summary of our parallel implementation of the ES-Capon beamformer as presented in [17] and [15]. This implementation was our starting point when the BS-Capon beamformer was implemented.

If the beamformer in (II.7) is analyzed in combination with the matrix equation in (II.5) and the covariance estimation process in (II.3), we see that the calculation of a set of Capon weights is independent across the outputs $z[n]$. As shown in Fig. II.2, a first level of granularity is therefore to divide the image into a grid, where the output in each cell can be calculated independently. Further, based on memory dependencies

and to hold memory consumption per thread at a minimum, a group of threads should process one output in the final image. Continuing with the calculation of one weight vector, the algorithm can be broken down into three main steps that, to some extent, can be divided further into a set of parallel processes. That is, for each data vector $\mathbf{x}$ in the image:

1) Calculate the sample covariance matrix $\hat{\mathbf{R}}$ as in (II.3).
2) Solve the linear system of equations as in (II.5).
3) Calculate $\mathbf{w} = \mathbf{b}/\mathbf{1}_L^H\mathbf{b}$ and the amplitude as in (II.7).

The computational complexities of these three steps are $O(N_K N_L L^2)$, $O(L^3)$, and $O(L)$, respectively. As M (and thereby L) increases, the first two steps quickly become unmanageable, and real-time performance in cardiac ultrasound imaging can only be achieved by exploring methods with reduced complexity.

II.3.1 Calculation of Multiple Sample Covariance Matrices

Calculation of one element of $\hat{\mathbf{R}}$ is from (II.2) found to be independent of calculation of the other elements. A natural level of granularity would then be to assign one thread to each element of $\hat{\mathbf{R}}$ [13]. This will minimize global reads and writes needed per thread, but since we want to avoid block-to-block synchronization, the maximum supported subarray length is limited to 32 by this approach. Since most medical ultrasound arrays have 64 elements or more, an implementation not limited by $L_{max} = M/2 = 32$ is preferable.

In [15, 17] we demonstrated how the cubic complexity with L can be reduced by realizing that calculations in (II.2) overlap across subarrays. The complexity of calculating $\hat{\mathbf{R}}$ is by this reduced to $O(N_K(L(N_L - 2) + 2L^2))$. For $L = M/2$ this reduces to $O(N_K M^2)$. Maximum supported L is also increased to the maximum number of threads per block (currently 1024).

II.3.2 Solving Multiple Small Linear Systems

In [17] we discussed how it is not given that solving on the GPU is faster than solving on the CPU in general. However, there are two arguments for solving on the GPU. First the CPU, which might already run a lot of ultrasound-related algorithms, is offloaded. Second, the transfer of several thousand covariance matrices over the PCI-Express bus, from the CPU to the GPU, is avoided.

For the results presented in this paper, we have used a GPU implementation of batched Gauss-Jordan elimination made by Nvidia to solve $\hat{\mathbf{R}}\mathbf{b} = \mathbf{1}_L$, where $\mathbf{b} = \hat{\mathbf{R}}^{-1}\mathbf{1}_L$, for all pixels in an image in one kernel launch. The solver is available online for registered CUDA developers [30].

II.3.3 Compute Beamformer Output

The final output can be formed by either reducing the data down to length L as in (II.7), or increasing the weight vector to length M as in (II.8). Since we selected to

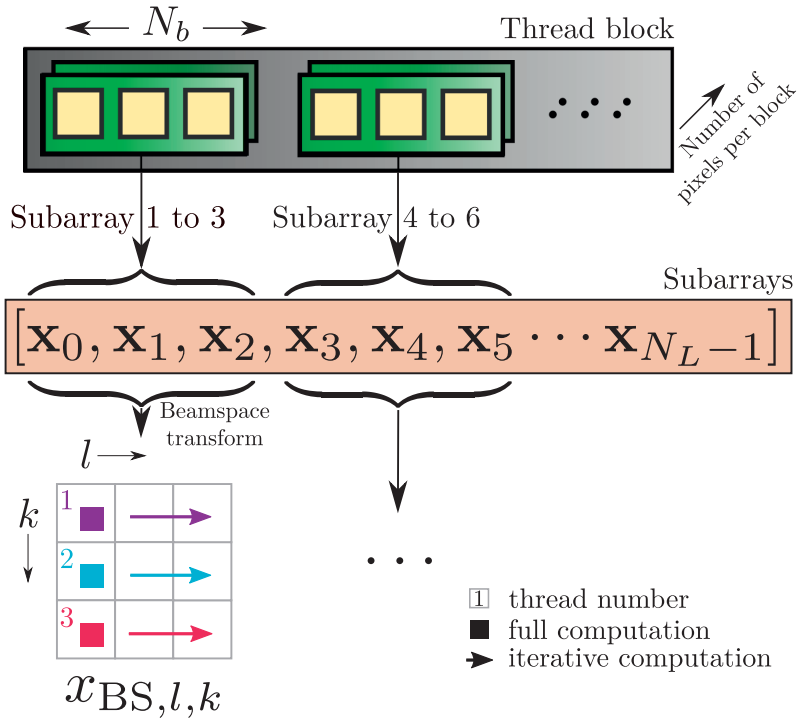


Figure II.3: Calculation of the sliding beamspace transform. A group of N_b threads is assigned a set of subarrays which they transform to beamspace in a sliding window fashion. In this example $N_b = 3$, l is the current subarray and k is the current beam.

use L threads per output, the data is reduced to length L by letting thread r calculate

$$\breve{x}_r = \sum_{k=r}^{r+N_L-1} x_k \text{ for } r \in [0, L-1]. \tag{II.12}$$

Each thread now computes $z_r = w_r^* x_r$, and finally one thread computes and outputs $z = \frac{1}{N_L} \sum_{r=0}^{L-1} z_r$ to global memory.

II.4 Parallel BS-Capon

In this section we will derive a parallel implementation of the BS-Capon beamformer, and comment on how it deviates from the implementation in Section II.3. The three main steps described in Section II.3 are present here as well, but in addition we need to transform the per-element data to beamspace. This can be done using a FFT, but for few beams ($N_b < \log L$) a matrix multiply with the truncated Butler matrix is less costly.

In order to reduce the complexity of the matrix inversion problem the transformation must be performed before or after the covariance matrix has been estimated. If the transformation is applied before, one transformation is needed per subarray. This results in subarrays that no longer overlap. The amount of input data are therefore changed from M to $N_L N_b$ elements per sample, but all remaining steps should get lower computational complexity as long as $N_b < L$.

The other option is to transform the element space covariance matrix to beamspace:

$$\hat{\mathbf{R}}_{\mathrm{BS}} = \mathbf{B}\hat{\mathbf{R}}\mathbf{B}^H. \tag{II.13}$$

The advantage of this approach is that minor changes have to be made to the pipeline described in Section II.3, but it will only speed up the solving step if the solution $\mathbf{b}_{BS}$ is transformed back to element space. In this paper we will focus on the first approach, and how it has been implemented on the GPU. In both cases solving has to be performed against a beamspace steering vector

$$\mathbf{a}_{\mathrm{BS}} = \mathbf{B}\mathbf{1}_L = \sqrt{L}\mathbf{e}_1, \tag{II.14}$$

where $\mathbf{e}_i$ has the value one in the ith position and zeroes elsewhere. Note that the factor $\sqrt{L}$ was not included in [22].

As mentioned, transforming subarrays into beamspace yields subarrays that no longer overlap. This means that the spatial overlap previously exploited when calculating the sample covariance matrix in element space is no longer available. On the other hand, since N_b should be small ($N_b < 32$), one thread assigned to each element in $\hat{\mathbf{R}}_{\mathrm{BS}}$ is now an option. A kernel similar to the one described in [13] has therefore been implemented. The computational complexity of this step is now $O(N_K N_L N_b^2)$, which for $L = M/2$ is less than the complexity of building $\hat{\mathbf{R}}$ in element space as long as $N_b < \sqrt{M}$.

The overlap between subarrays that were exploited when estimating the covariance matrix in element space, can now be utilized when calculating the beamspace transform. Given a M long data vector divided into N_L subarrays, the beamspace transformation of the lth subarray for the kth beam is from (II.9) found to be:

$$x_{\mathrm{BS},l,k} = \mathbf{B}_k \mathbf{x}_l \qquad k \in [0, N_b - 1] \tag{II.15}$$

$$= \frac{1}{\sqrt{L}} \sum_{p=i}^{l+L-1} x_p e^{-j2\pi k(p-l)/L}, \tag{II.16}$$

where $\mathbf{B}_k$ is the kth row of $\mathbf{B}$.

This formula has been implemented using a normalized sliding DFT (SDFT)[31]. The SDFT allows us to still save instructions by utilizing the overlap between subarrays, but with an increase in task level granularity. In this situation, it might be tempting to use one thread per beam (index k) that calculates this beam for all subarrays. However, since the number of selected beams usually is small ($N_b \ll M$) we will end up with a large memory-per-thread ratio (M/N_b). Using one thread per beam is therefore not feasible for small-N_b-large-M configurations. One solution, as

depicted in Fig. II.3, is to combine SDFT with full computations. A number of sub thread groups of size N_b is created that targets different sets of subarrays which they transform using the SDFT algorithm. In that way, the number of threads per sample is increased and thus the memory-per-thread ratio is decreased.

After solving $\mathbf{b}_{\mathrm{BS}} = \sqrt{L}\hat{\mathbf{R}}_{\mathrm{BS}}^{-1}\mathbf{e}_1$, the beamspace weight vector is calculated as

$$\mathbf{w}_{\mathrm{BS}} = \frac{\mathbf{b}_{\mathrm{BS}}}{\sqrt{L}\mathbf{e}_1\mathbf{b}_{\mathrm{BS}}} \in \mathbb{C}^{N_b} \tag{II.17}$$

using one thread per beam, and applied to the beamspace data summed across subarrays,

$$z[n] = \mathbf{w}_{\mathrm{BS}}^H \sum_{l=0}^{L-1} \mathbf{x}_{\mathrm{BS},l}. \tag{II.18}$$

As presented in (II.3), adaptive diagonal loading is performed by loading the sample covariance matrix with the average energy per element, that is trace$\{\breve{\mathbf{R}}\}/L$. When data are transformed into beamspace, a high percentage of the total energy is mapped to a small subset of beams. Therefore, if a set of high energy beams is selected, the average energy per beam will be higher than the average energy per element (trace$\{\breve{\mathbf{R}}\}/L <$ trace$\{\breve{\mathbf{R}}_{\mathrm{BS}}\}/N_b$). This means that less diagonal loading (smaller d) has to be applied when the covariance is estimated among beams and not elements.

To summarize, by transforming data to beamspace the complexity of all subsequent steps is potentially reduced. Calculation of the covariance matrix is now $O(N_K N_L N_b^2)$, the inversion step is $O(N_b^3)$, and calculation of weights and the beamformer output are both $O(N_b)$. The added complexity of SDFT is $O(N_b L)$. To maximize performance, large adjustments to the processing pipeline described in Section II.3 is required.

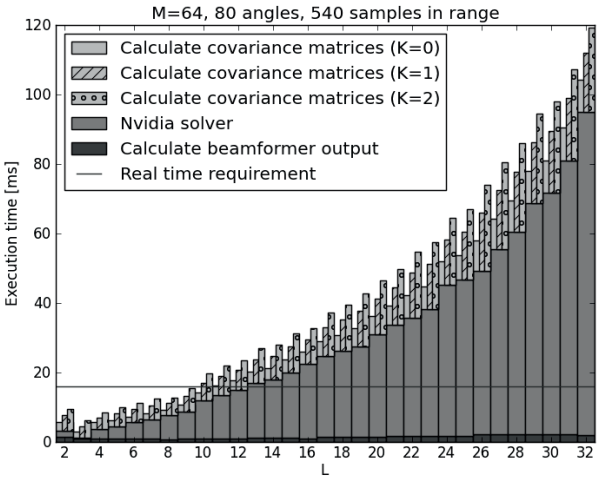
II.5 Benchmarks

In this section we present benchmarks of the implementations presented in Sections II.3 and II.4 performed on a Nvidia Quadro 6000 graphics card (14 SMs, 1 TFlops theoretical single precision performance, 6 GB global memory, and 144 GB/s memory bandwidth). Problem sizes are calculated based on 64 and 96 element phased arrays with a 3.4 MHz center frequency. Assuming a system bandwidth of 80% (relative to the center frequency) we need a complex sampling frequency of 2.7 MHz, and when imaging down to a depth of 15.4 cm we have approximately 540 samples per range line. The angular sampling is given by the Rayleigh criterion divided by two [32],

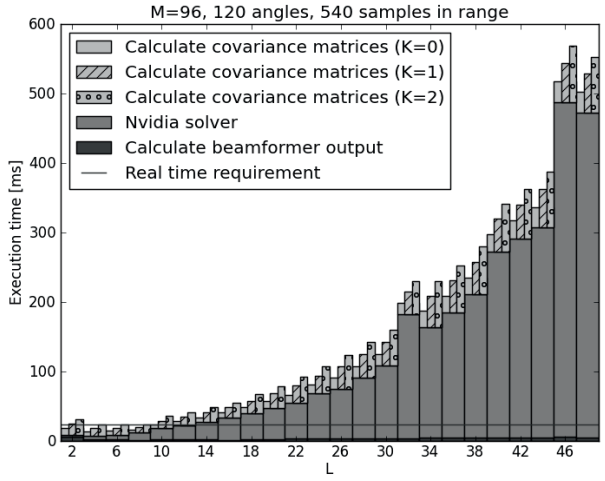
$$\delta\theta = \lambda/(2Mp), \tag{II.19}$$

where λ is the pulse wavelength and p is the element pitch. To sample a 70° sector with 64- and 96-element arrays with $p = \lambda/2$ we therefore need approximately 80 and 120 receive beams respectively.

The presented execution times do not include transfer of data from CPU- to the GPU-side, since data are assumed to be pre-delayed and available in global GPU



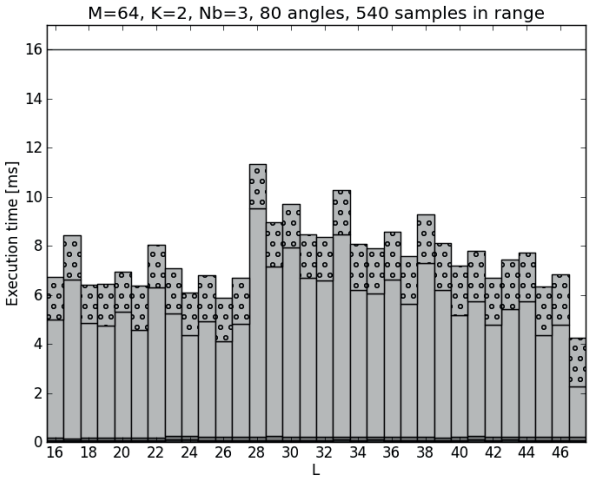
(a)



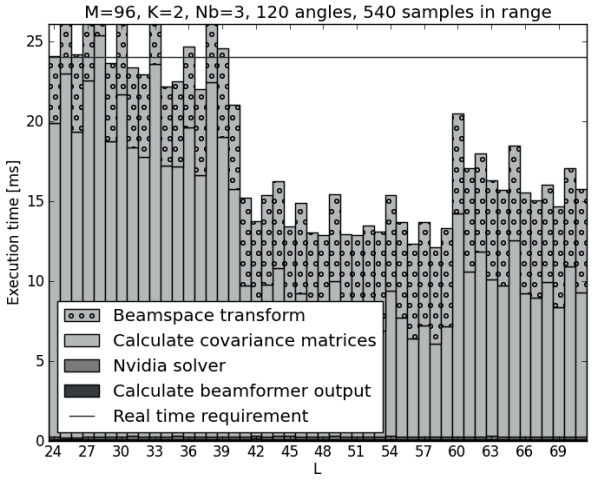
(b)

Figure II.4: Benchmarks of GPU Capon beamforming in element space for a cardiac image covering a 70° wide 15.4 cm deep sector. Execution time is plotted as a function of subarray length L. The figures show execution times for the three steps listed in Section II.3: Calculating covariance matrices with three different choices of temporal averaging, solver, and the final beamformer sum.

memory. This is a valid assumption, since the imaging scenarios presented in this paper are well below the 8.0 GB/s limit of PCI Express 2.0. What is included are



(a)



(b)

Figure II.5: Benchmarks of GPU beamspace Capon beamforming for a cardiac image covering a 70° and 15.4 cm deep sector. Execution time is plotted as a function of subarray length L. If L had been fixed at $M/2$ and execution time was plotted against N_b, the solver would have shown the same quadratic growth as in Fig. II.4. When N_b is fixed, calculating the covariance matrix and the BS-transform is inverse proportional and proportional to L respectively. The large jumps in execution times are caused by adaptive kernel launch configurations, which depends on M, L and N_b.

all calculations, memory transfers from global to shared GPU memory, and shared memory access. The delay step has previously been shown to take far less time than what we report for both Capon beamformers [13, 33].

Benchmarks of the ES-Capon beamformer are presented in Fig. II.4. For each subarray size, the total execution time is split into one bar for each of the three steps described in Section II.3, and the calculation of $\hat{\mathbf{R}}$ is presented for three different values of K. For $M = 64$, $L = 32$ and $K = 2$, a typical setup for high-quality cardiac imaging, processing throughput is 8.3 frames per second (FPS). For $M = 96$, $L = 48$ and $K = 2$ the processing throughput drops below 0.5 FPS.

In Fig. II.5 we present similar benchmarks for the BS-Capon beamformer. The figure only presents results with use of temporal averaging ($K = 2$). For $M = 64$, $L = 32$, $K = 2$ and $N_b = 3$, processing throughput is 125 FPS (8 ms). For $M = 96$, $L = 48$, $K = 2$ and $N_b = 3$, processing throughput is 77 FPS (13 ms). In all four benchmarks there is a line indicating the time it takes to acquire the underlying image using one receive line per transmit. Hence, below this line we are capable of performing real-time processing.

II.6 In-vivo cardiac images

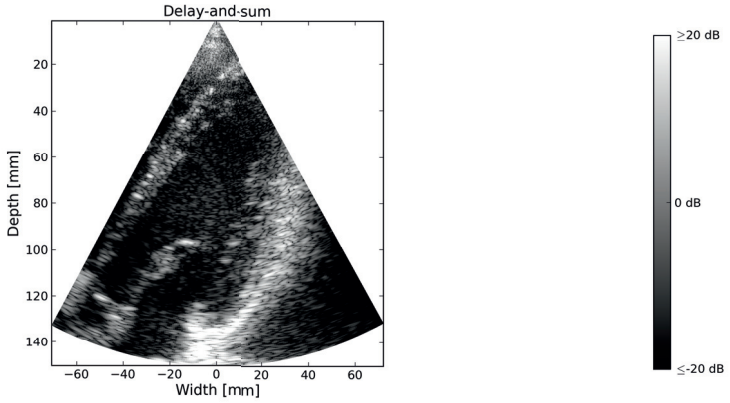
Fig. II.6 presents results of applying ES- and BS-Capon on an *in-vivo* image of the left ventricle. The image was acquired using a 64-element and $p = 0.65\lambda$ pitched phased array in harmonic mode (1.7 MHz at transmit and 3.4 MHz at receive). The image frame is extracted from a cardiac cycle, that can be viewed in its full length in the attached videos **Media-Movie 1, 2 and 3**. The region of interest is 70° with a minimal lateral sampling of 98 receive beams given by (II.19). The image was acquired using 50 transmit beams and "Synthetic Transmit Beamforming" was used on receive [32].

The data set has been processed with the following settings for the ES-Capon beamformer: $L = 32$, $K = 2$, and $d = 0.1$. These settings have previously been shown to provide improved lateral resolution while maintaining DAS-like speckle [28]. For BS-Capon the following parameters have been used: $L = 32$, $K = 2$, $d = 0.01$, and $N_b = 3$.

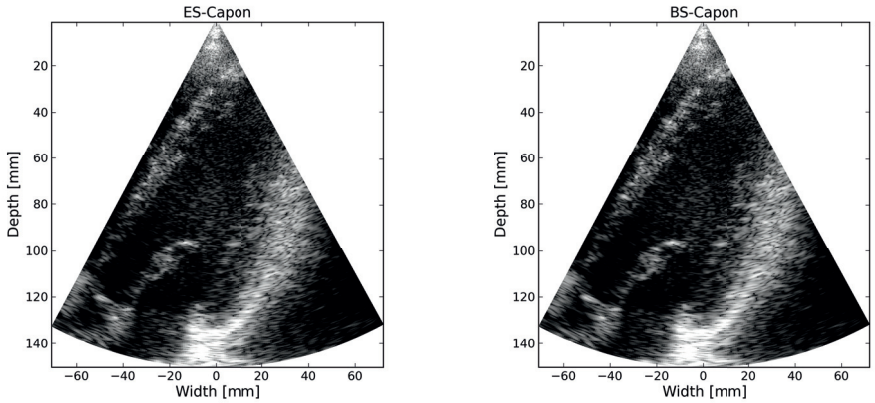
II.7 Trading resolution for speed

As seen from Fig. II.4 the execution time, for the solver in particular, is reduced when L is reduced. The same is true when N_b is reduced in beamspace. So to gain an increase in processing throughput we can reduce L in element space or N_b in beamspace. In this section we investigate how adjusting L for ES-Capon and N_b for BS-Capon impacts the lateral resolution.

In Fig. II.7, the minimum distance between two point targets (while preserving a 6 dB saddle point in between the two) is plotted against L (when processed with ES-Capon) and N_b (when processed with BS-Capon). To produce the result in Fig. II.7 $K = 2$ and $d = 0.01$ has been used on both methods in order to obtain the same



(a)



(b) (c)

Figure II.6: *In-vivo* harmonic cardiac ultrasound image acquired using a 3.5 MHz, 64 elements phased array. Number of receive beams for the 70° sector is 98, and number of samples in range is 610. Dynamic range is -20 dB to 20 dB with mean value represented as 0 dB. The image is the first frame in the attached videos. a) Delay-and-sum with uniform weights **Media-Movie 1**. b) Element-space Capon ($L = 32, K = 2, d = 0.1$) **Media-Movie 2**. c) Beamspace Capon ($L = 32, K = 2, N_b = 3, d = 0.01$) **Media-Movie 3**.

resolution for $L = N_b = M/2$. In Fig. II.6, the diagonal loading factor for BS-Capon was adjusted to make the two results equal for $L = 32$ and $N_b = 3$.

The phantom used contained two point targets with 30 dB amplitude located at 90

mm depth. The two point targets were placed inside speckle, with 0 dB in amplitude, covering an area from $\pm 90^\circ$ in azimuth. The phantom was simulated using Field II [34, 35]. The simulation was performed using a 64-element, $p = \lambda/2$ pitch, 10 μm kerf, 80% bandwidth, and 10 mm high phased array. The transducer was excited with a 3.4 MHz and 1.5λ long pulse with transmit focus at 90 mm. The precision of the presented measurements is 0.12 mm, dictated by the simulated frame rate of 25 FPS, and the relative speed of the two point scatterers, 3 mm/s. To avoid having to deal with signal cancellation and errors caused by not hitting each point with a receive beam, 16x oversampling has been used on transmit.

For high values of L and N_b the two methods are equal, but for lower values, BS-Capon provides improved resolution compared to using small subarrays. We see that the resolution for BS-Capon gradually decreases due to an automatic increase in diagonal loading (cf. end of Section II.4). If we compensate by reducing d, BS-Capon will provide resolution equal to ES-Capon (with $L = M/2$) for all values of N_b except one [22]. For $N_b = 1$ the resolution is worse than for DAS because of a triangular apodization caused by subarray averaging.

In element space, resolution decays rapidly since the effective adaptive aperture is proportional to L. For $L = N_b = 3$, BS-Capon provides almost 1 mm improved resolution over DAS and ES-Capon. However, note that the resolution of the Capon beamformers strongly depends on the image signal-to-noise ratio (SNR) and the choice of parameters. Thus, different SNR scenarios and choices of parameters will lead to different results. It is also worth noting that ES-Capon beamforming in this example performs worse than DAS for small values of L. For these small subarrays, the method is not able to steer zeros at each point target or accurately micro-steer the main lobe. Subarray averaging further impose a triangular or trapezoid apodization function, which leads to worse resolution if the introduced adaptivity can not compensate for the increase in main lobe width. For $L = 1$ the ES-Capon is equal to DAS.

II.8 Discussion

In this paper we have presented a parallel implementation of the BS-Capon beamformer for high-resolution cardiac ultrasound imaging. The results show that it is now possible to apply BS-Capon in real-time in cardiac ultrasound imaging by utilizing the power of a GPU. According to Fig. II.5 it is also possible to process two parallel receive lines per transmit beam (for both $M = 64$ and $M = 96$. $L = M/2$).

For ES-Capon, processing rates for high quality settings are still too high to support real-time cardiac ultrasound imaging. From Fig. II.4a we see that for $L = M/2 = 32$ and $K = 1$, the total execution time is 110 ms (9 FPS). Taking into account that we use complex data, but less samples per image, these results are similar to the performance reported by Chen et al. [13].

A major part of the total execution time for ES-Capon comes from solving the system of linear equations. This limitation is targeted by reducing the sample covariance matrix size in beamspace, and as shown in Fig. II.5, inverting the 3×3 beamspace covariance matrix is then barely visible. The time it takes to construct all

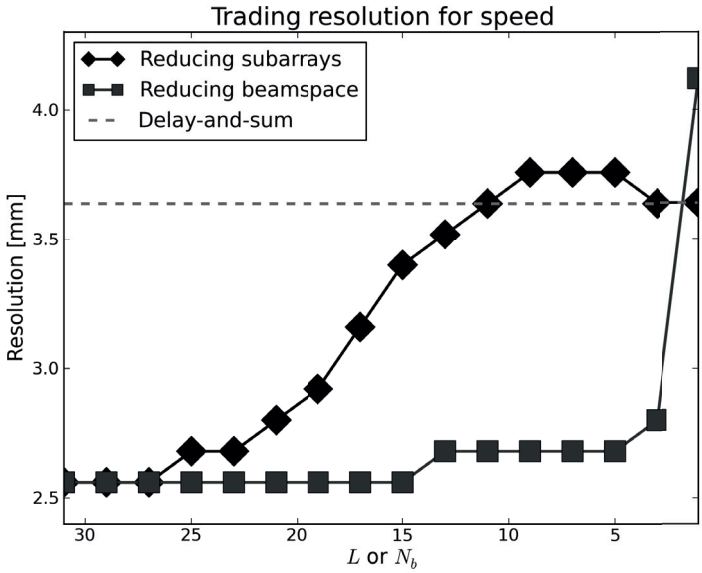


Figure II.7: Trade-off between resolution and speed by reducing subarray size or reducing beamspace dimension. As L or N_b gets smaller, the execution time is reduced (Fig. II.4 and II.5). When N_b is varied, L is fixed on 32.

covariance matrices is also reduced, but for few beams this is now the major contributor to the total execution time. If real-time is defined as 10 FPS [13] almost all execution times in both Fig. II.4a and II.5 are real-time. More natural for cardiac ultrasound imaging is to define real-time as the time it takes to acquire the underlying image. With this requirement the images formed using a 64 and 96 element array have a real-time requirement of 62.5 and 41.7 FPS respectively using single line acquisition. Each benchmark plot has a line indicating this real-time requirement. With beamspace processing, all processing times for large subarrays are below this line.

This paper further shows how to obtain a parallel formulation of the BS-Capon beamformer, and how to implement it in a GPU framework. The per-pixel granularity suggests that the beamformer is easily dividable across multiple GPUs. It is therefore possible to achieve higher frame rates if several GPUs are used simultaneously. However, there is usually a space-cost-energy limitation for how many GPUs that can be assembled in an ultrasound scanner.

Turning to image quality we see from Fig. II.7 and the attached videos, that Capon beamforming provides improved lateral resolution of moving point targets both in element- and in a reduced beam-space ($N_b = 3$). In the cardiac recording (Fig. II.6), structures are thinner and better resolved when the Capon beamformers are applied.

One example is the mitral valve located at 10 cm depth and -1.5 cm width. On the other hand we observe a loss in contrast, especially in the septum (Fig. II.6, 6 cm depth, -2 cm width). This loss can be partially explained by the presence of highly correlated signals and mismatch between the assumed steering vector and signal. This is the same issue that made us use 16x oversampling when producing Fig. II.7. Future work should focus on how large this oversampling needs to be and how it should be conducted in order to still achieve real time processing.

In the end we want to point out that cardiac ultrasound has not been selected as the target modality in this paper because Capon beamforming shows large improvements in image quality in particular. It has been selected because it is the most computationally demanding modality, measured in floating points operations per second. The rationale being that if BS-Capon can run in real time for cardiac ultrasound it can run in real time for all medical ultrasound modalities. We believe that Capon beamforming may show more improvements in other imaging modalities where detection of small point targets is important.

II.9 Conclusion

In this paper a parallel formulation and implementation of the beamspace Capon beamformer have been presented, and the beamformer has successfully been implemented on the GPU. For a three dimensional beamspace, processing rates supporting real-time cardiac ultrasound imaging was shown to be possible, meaning that processing rates higher than the image acquisition rate are obtained for a wide range of parameters. Image quality was investigated in an *in-vivo* cardiac dataset. These results showed that Capon beamforming is feasible for cardiac ultrasound imaging, providing images with improved lateral resolution both in element- and beamspace.

Acknowledgment

The authors would like to thank Prof. K. Kristoffersen, GE Vingmed Ultrasound and Norwegian University of Science and Technology, for valuable comments.

References

[1] J. Capon, "High-resolution frequency-wavenumber spectrum analysis," *Proceedings of the IEEE*, vol. 57, no. 8, pp. 1408–1418, 1969.

[2] J.-F. Synnevåg, A. Austeng, and S. Holm, "Adaptive Beamforming Applied to Medical Ultrasound Imaging," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 54, no. 8, pp. 1606–1613, Aug. 2007.

[3] A. Austeng, T. Bjåstad, J.-F. Synnevåg, S.-E. Måsøy, H. Torp, and S. Holm, "Sensitivity of minimum variance beamforming to tissue aberrations," in *Proc. IEEE Ultrasonics Symp.*, 2008, pp. 1072–1075.

[4] F. Vignon and M. R. Burcher, "Capon beamforming in medical ultrasound imaging with focused beams." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 55, no. 3, pp. 619–628, Mar. 2008.

[5] F. Viola and W. Walker, "Adaptive signal processing in medical ultrasound beamforming," in *Proc. IEEE Ultrasonics Symp.*, 2005, pp. 1980–1983.

[6] S. Mehdizadeh, A. Austeng, T. F. Johansen, and S. Holm, "Minimum Variance Beamforming Applied to Ultrasound Imaging With a Partially Shaded Aperture," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 59, no. 4, pp. 683–693, Apr. 2012.

[7] J.-F. Synnevåg, A. Austeng, and S. Holm, "Benefits of minimum-variance beamforming in medical ultrasound imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 9, pp. 1868–1879, Sep. 2009.

[8] H. So, J. Chen, B. Yiu, and A. Yu, "Medical Ultrasound Imaging: To GPU or Not to GPU?" *IEEE Micro*, vol. 31, no. 5, pp. 54–65, Sep. 2011.

[9] J. G. McWhirter and T. J. Shepherd, "Systolic array processor for MVDR beamforming," *Proc. IEE Radar and Signal Processing*, vol. 136, no. 2, pp. 75–80, 1989.

[10] M. Moonen, "Systolic MVDR beamforming with inverse updating," *Proc. IEE Radar and Signal Processing*, vol. 140, no. 3, pp. 175–178, 1993.

[11] P. Sinha, A. D. George, and K. Kim, "Parallel algorithms for robust broadband MVDR beamforming," *Journal of Computational Acoustics*, vol. 10, pp. 69–96, 2002.

[12] J. Chen, B. Y. Yiu, B. K. Hamilton, A. C. Yu, and H. K.-H. So, "Design space exploration of adaptive beamforming acceleration for bedside and portable medical ultrasound imaging," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, p. 20, Dec. 2011.

[13] J. Chen, B. Y. S. Yiu, H. K. So, and A. C. H. Yu, "Real-Time GPU-Based Adaptive Beamformer for High Quality Ultrasound Imaging," in *Proc. IEEE Ultrasonics Symp.*, 2011, pp. 1–4.

[14] J. Chen, A. C. H. Yu, and H. K.-H. So, "Design considerations of real-time adaptive beamformer for medical ultrasound research using FPGA and GPU," in *International Conf. on Field-Programmable Technology (FPT)*, 2012, pp. 198–205.

[15] J. I. Buskenes, J. P. Åsen, C.-I. C. Nilsen, and A. Austeng, "An optimised GPU implementation of the MVDR beamformer for active sonar imaging," *Submitted to IEEE Journal of Oceanic Engineering*, 2013.

[16] ——, "Adapting the Minimum Variance beamformer to a Graphics Processing Unit for Active Sonar Imaging systems," in *Proceedings of Meetings on Acoustics*, 2013.

[17] J. P. Åsen, J. I. Buskenes, C.-I. C. Nilsen, A. Austeng, and S. Holm, "Implementing Capon Beamforming on the GPU for Real Time Cardiac Ultrasound Imaging," in *Proc. IEEE Ultrasonics Symp.*, 2012, pp. 2133–2136.

[18] J.-F. Synnevåg, A. Austeng, and S. Holm, "A Low-Complexity Data-Dependent Beamformer." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 58, no. 2, pp. 281–289, Feb. 2011.

[19] B. M. Asl and A. Mahloojifar, "A Low-Complexity Adaptive Beamformer for Ultrasound Imaging Using Structured Covariance Matrix," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 59, no. 4, pp. 660–667, Apr. 2012.

[20] A. Jensen and A. Austeng, "An approach to multibeam covariance matrices for adaptive beamforming in ultrasonography," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 59, no. 6, pp. 1139–1148, Jun. 2012.

[21] K. Kim, S. Park, Y.-T. Kim, S.-C. Park, J. Kang, J.-H. Kim, and B. MooHo, "Flexible Minimum Variance Weights Estimation Using Principal Component Analysis," in *Proc. IEEE Ultrasonics Symp.*, 2012.

[22] C.-I. C. Nilsen and I. Hafizovic, "Beamspace adaptive beamforming for ultrasound imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 10, pp. 2187–2197, Oct. 2009.

[23] K. Thomenius, "Evolution of ultrasound beamformers," in *Proc. IEEE Ultrasonics Symp.*, vol. 2, 1996, pp. 1615–1622.

[24] S.-L. Wang and P.-C. Li, "MVDR-based coherence weighting for high-frame-rate adaptive imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 10, pp. 2097–2110, Oct. 2009.

[25] W. Featherstone, "A novel method to improve the performance of Capon's minimum variance estimator," in *Tenth International Conf. on Antennas and Propagation (ICAP)*, 1997, pp. 322–325.

[26] H. Krim and M. Viberg, "Two decades of array signal processing research: the parametric approach," *IEEE Signal Process. Mag.*, vol. 13, no. 4, pp. 67–94, 1996.

[27] V. Reddy, A. Paulraj, and T. Kailath, "Performance analysis of the optimum beamformer in the presence of correlated sources and its behavior under spatial smoothing," *IEEE Audio, Speech, Language Process.*, vol. 35, no. 7, pp. 927–936, Jul. 1987.

[28] J.-F. Synnevåg, C.-I. C. Nilsen, and S. Holm, "Speckle Statistics in Adaptive Beamforming," in *Proc. IEEE Ultrasonics Symp.* IEEE, Oct. 2007, pp. 1545–1548.

[29] Nvidia, "CUDA Toolkit Documentation," 2013. [Online]. Available: http://docs.nvidia.com/cuda/index.html

[30] ——, "Registered Developers Website," 2013. [Online]. Available: https://developer.nvidia.com/user

[31] R. Lyons, "dsp tips & tricks - the sliding DFT," *IEEE Signal Process. Mag.*, vol. 20, no. 2, pp. 74–80, Mar. 2003.

[32] T. Hergum, T. Bjåstad, K. Kristoffersen, and H. Torp, "Parallel beamforming using synthetic transmit beams," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 54, no. 2, pp. 271–280, Feb. 2007.

[33] J. Kwon, J. H. Song, S. Bae, T.-k. Song, and Y. Yoo, "An effective beamforming algorithm for a GPU-based ultrasound imaging system," in *Proc. IEEE Ultrasonics Symp.* IEEE, 2012, pp. 619–622.

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

[35] J. A. Jensen, "Field: A program for simulating ultrasound systems," *Medical & Biological Engineering & Computing*, vol. 34, no. Supplement 1, Part 1, pp. 351–353, 1996.

Paper III

Capon Beamforming and Moving Objects - An Analysis of Lateral Shift-Invariance

Jon Petter Åsen[1], Andreas Austeng[2] and Sverre Holm[1,2]
[1]Medical Imaging Lab (MI-Lab), Norwegian University of Science and Technology, Trondheim, Norway
[2]Department of Informatics, University of Oslo, Oslo, Norway

IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, , vol. 61, no. 7, pp. 1152-1160, Jul 2014.

If an ultrasound imaging system provides a presentation of a moving object which is sensitive to small spatial shifts, the system is said to be locally spatially shift-variant. This can, for instance, happen if the axial or lateral sampling is insufficient. The Capon beamformer has been shown to provide increased lateral resolution in ultrasound images. Increased lateral resolution should demand denser lateral sampling. However, in previous literature on Capon beamforming for medical ultrasound imaging only single-frame scenarios have been simulated. Temporal behaviour and effects caused by the increased resolution and lack of oversampling have therefore been neglected. In this paper, we analyse the local lateral shift-invariance of the Capon beamformer when imaging moving objects. We show that insufficient lateral sampling makes an imaging system based on the Capon beamformer laterally shift-variant. Different methods for oversampling on transmit and receive are then discussed and investigated to improve on the Capon beamformer. It is shown that lateral shift-invariance can be improved by oversampling based on phase rotation on receive. This without affecting the acquisition frame rate and with a minor change in processing complexity.

III.1 Introduction

In the last decade several authors have applied the Capon beamformer [also known as the minimum variance beamformer] for medical ultrasound imaging [1–3]. The method has been shown to improve lateral resolution and edge sharpness in ultrasound images [1, 4, 5]. This is accomplished by adaptively adjusting the array weights based on element-to-element covariance to minimize interference. The method is, however, also highly sensitive to model errors [6–9]. This has led to the development of several robust versions of the Capon beamformer with e.g., spatial smoothing [10] or diagonal loading [11] of the covariance estimate, or application of steering vector uncertainty sets [12, 13]. Spatial smoothing and diagonal loading, together with temporal smoothing of the covariance estimate, have been applied in earlier work on Capon beamforming for medical ultrasound imaging [4]. The first two trade resolution for robustness and temporal smoothing helps to preserve speckle statistics for high resolution parameters [14]. Steering vector uncertainty sets, however, have not been applied for medical ultrasound imaging.

A steering vector uncertainty set reduces signal-nulling which appears if the signal of interest is present during the covariance estimation process and if this estimate is later used with a steering vector that does not match the signal propagation vector exactly. This self-nulling is actually how the Capon beamformer obtains its super resolution capabilities, but the signal can also get lost in the process. In [11], it was shown that most methods based on steering vector uncertainty sets are equivalent to diagonal loading where the loading factor depends on the steering vector uncertainty. In that way, they make up for a given uncertainty by making the Capon beam wider. Hence, the maximal achievable resolution is decreased if the initial steering vector spacing was too coarse. In medical ultrasound imaging, this initial set of steering vectors is governed by the number of receive beams. In this paper, we will make

use of an additional set of steering vectors to reduce the self-nulling effect by means of oversampling. The maximal achievable resolution is then maintained without the presence of severe self-nulling.

In [15], Cox presents in great detail how the amount of mismatch between the selected steering vector and the signal propagation vector impacts the beamformer output in simple situations. He also states the following important fact about the Capon beamformer (the $\mathbf{k}_3$ processor):

> "*... a* $\mathbf{k}_3$ *processor requires more closely spaced beams than a ...* $\mathbf{k}_1$*-processor in order to avoid serious signal suppression effects being introduced on signals arriving from directions between the beams*",

where $\mathbf{k}_1$ is the delay-and-sum (DAS) beamformer. In Fig. III.1 we have plotted the two-way response for DAS and the Capon beamformer using [15, Eqs. (26) and (32)]. A 64-element linear array with $\lambda/2$ element spacing, where λ is the wavelength, is scanned over a sector equal to four times the array resolution using narrow band excitation. Located in the farfield of the sector is an object moving parallel to the array surface. The plot shows the high precision scanned output when the object is located at plus and minus half the required beam spacing for DAS [See (III.8)]. The effect of typical subarray averaging and diagonal loading have been added to the Capon beamformer output [subarray length of 32 and a diagonal loading factor of 0.01]. We see that a DAS beamformer, using the required beam spacing, could yield a 1.9 dB scalloping loss [16] if the object is moving parallel to the array. However, if the Capon beamformer is used with the same beam density, the loss could be as large as 27 dB. This imaging system is then highly sensitive to small spatial shifts, either by the object or the array itself. A 1.9 dB scalloping loss for the Capon beamforming is obtained for this specific example when the beam spacing or object distance is decreased with a factor of 25.

In previous literature on Capon beamforming for medical ultrasound imaging, only single frame scenarios have been simulated. The scalloping loss effect has therefore been passed over in silence by either oversampling on transmit, by carefully positioning point scatterers exactly on transmit-receive beams, or by ignorance. In our first work in which Capon beamforming was applied on a loop of images containing moving objects [17, 18], we commented on the effect, but no solution was presented other than oversampling on transmit when simulating. The same was done by Jensen and Austeng [19].

In this paper, we investigate the issue further. We show that when a moving point scatterer is imaged with the Capon beamformer with the same beam spacing which is sufficient for DAS, the predicted scalloping loss is observed both in simulations and *in vitro* data. The problem is then to find what can be done to improve on the situation without affecting the acquisition frame rate and processing throughout.

The next section gives an introduction to Capon beamforming together with a short introduction to lateral sampling and shift-invariance. In Section III.3 we study the scalloping loss yield by the Capon beamformer and its local lateral shift-invariance property. Different methods for reduced scalloping loss and by that improved local lateral shift-invariance are presented in Section III.4. Then, in Section III.5, we present

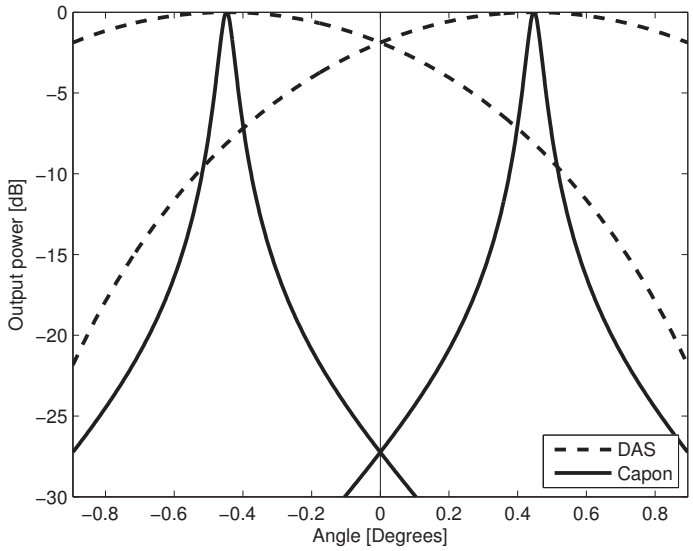


Figure III.1: Estimated drop in output power mid-way between receive beams for delay-and-sum and Capon beamforming (with subarray length of 32 and a diagonal loading factor of 0.01) applied in a far-field-narrowband setting.

the results of applying the most promising method on *in vitro* data. Finally, we discuss our findings in Section III.6 and give our conclusions in Section III.7.

III.2 Background

III.2.1 Standard Beamforming

The standard form of time-domain array beamforming is the DAS beamformer, for which the output z is calculated as

$$z[n] = \sum_{m=0}^{M-1} w_m^* x_m[n - \Delta_m[n]] = \mathbf{w}^H \mathbf{x}[n], \tag{III.1}$$

where M is the number of elements or channels in the array, $\Delta_m[n]$ is the per-element focusing and steering delay, and $\mathbf{w}$ is a weight vector or window function. The weight vector is typically selected to trade resolution for a lower side-lobe level. Hence, the weights are typically real and their K-space response is then symmetric.

III.2.2 Capon Beamforming

The Capon beamformer produces a weight vector, like the one in (III.1), based on the impinging signal and an optimization criterion. The problem is as follow [20]:

$$\min_{\mathbf{w}} E\{|z[n]|^2\} \approx \min_{\mathbf{w}} \mathbf{w}^H \hat{\mathbf{R}} \mathbf{w} \tag{III.2}$$

$$\text{subjected to } \mathbf{w}^H \mathbf{a} = 1, \tag{III.3}$$

where $\hat{\mathbf{R}}$ is a sample covariance matrix. Hence, the resulting weight vector minimizes the output power while maintaining unit gain in the steering direction $\mathbf{a}$.

The sample covariance matrix can be estimated for an active broadband system in the following way [4]:

$$\breve{\mathbf{R}}[n] = \frac{1}{N_L N_K} \sum_{n'=n-K}^{n+K} \sum_{l=0}^{N_L-1} \mathbf{x}_l[n'] \mathbf{x}_l[n']^H, \tag{III.4}$$

where K is selected proportional to the pulse length, $N_K = 2K+1$, $N_L = M - L + 1$, and $\mathbf{x}_l$ is a L-long subarray $[x_l[n], \ldots x_{l+L}[n]]$. Finally the matrix is loaded with a diagonal factor for numerical stability and increased robustness,

$$\epsilon = d * \text{trace}\{\breve{\mathbf{R}}\}/L \tag{III.5}$$

$$\hat{\mathbf{R}} = \breve{\mathbf{R}} + \epsilon \mathbf{I}. \tag{III.6}$$

The solution to the minimization problem in (III.2) is

$$\mathbf{w}[n] = \frac{\hat{\mathbf{R}}[n]^{-1}\mathbf{a}}{\mathbf{a}^H \hat{\mathbf{R}}[n]^{-1}\mathbf{a}} \in \mathbb{C}^L. \tag{III.7}$$

Note that one matrix has to be constructed and inverted for each data vector $\mathbf{x}[n]$ received by the system. From (III.7) it is also clear that complex data [in-phase and quadrature data] can lead to complex weights. The weights can therefore have an asymmetric K-space response with a shifted main lobe and zeros at arbitrary positions.

III.2.3 Lateral Sampling and Shift-Invariance

The required lateral sampling in an ultrasound image for a given center frequency, f_c, and bandwidth, B, is given by the aperture size, D, and the wavelength of the center frequency, $\lambda_c = c/f_c$, where c is the speed of sound. The required Nyquist beam spacing is then:

$$\delta\theta < \frac{\lambda_c}{2qD}, \tag{III.8}$$

where the factor two is the effect of a two-way acquisition [21], and where q is an oversampling factor which can be applied to reduce folding of frequencies between f_c

and $f_c + B/2$. In this paper, we will use the q-factor to perform oversampling before Capon beamforming.

Equation (III.8) was used to calculate the object positions in Fig. III.1. Therefore, if (III.8) is not fulfilled, the resulting scalloping loss for the DAS beamformer could become visible in the image. The system output will then be sensitive to small spatial shifts. We say that the system is locally spatially shift-variant. Since Capon beamforming improves the lateral resolution we will focus on sensitivity to small lateral shifts, or in other words, the local lateral shift-invariance property of the imaging system.

In an ultrasound image with 50 dB dynamic range mapped to 256 gray levels, a 1 dB loss corresponds to 5 gray levels. This is approximately equal to the visibility threshold [Weber fraction of 2%] for grayscale images. A loss larger than 1 dB could therefore end up being visible to the observer.

To visualize the local lateral shift-invariance of a given imaging system, we will make use of lateral shift-variance plots (LSV-plots) [22]. An LSV-plot is constructed for a given imaging system by imaging a point scatterer moving laterally at constant speed. For each lateral position, the rms of the image data is calculated in range, and the resulting beam profiles are stacked in to an image. The LSV-plots in this paper are displayed as a contour plot with contours on -1, -2, -3, -6, -12 and -24 dB. The maximum for each beam profile is also marked with a black dot, and the amplitude variation among these points is presented in a subplot next to the LSV-plot. The LSV-plots based on simulations have been up-interpolated to 128 samples in each direction using cubic interpolation if the number of receive beams is less than 128.

To further quantify the shift-invariance we have calculated the mean absolute error (MAE) between the maximum point position in each beam profile and a straight line $[f(x) = x]$. We will refer to this measure as P-MAE. The MAE of the maximum points' amplitude versus the data maximum is also measured. We refer to this measure as A-MAE. The first measure quantifies the amount of geometric distortion [22]; and the latter quantifies the peak gain variation, or in other words, the scalloping loss. The maximum amplitude deviation, or maximum scalloping loss, along this point trace is also calculated. We present all these numbers together with the LSV-plot.

A given imaging system will then be locally laterally shift-invariant if the contours are diagonal and if the amplitude variation among the maximum points is less than 1 dB. Then there is minimal geometric distortion (low P-MAE) and the scalloping loss will not be visible for the observer (low A-MAE).

Figure III.2a presents such a LSV-plot for an imaging system using the DAS beamformer with a beam spacing equal to (III.8). Dotted vertical lines indicate position of transmit-receive beam pairs. The point scatterer response has been simulated using Field II [23, 24] using a 96 element phased array with a 22 mm azimuth aperture, which was excited with a 2.5 MHz and 1.5 cycles long pulse. Transmit focus in azimuth and elevation were configured to be equal and the point scatterer was moved through this point. The transmit focus is where the ultrasound beam is at its narrowest; hence shift variance effects will appear at this location first. The beam density was calculated based on (III.8) with a speed of sound equal to 1540. We see that the contours in Fig. III.2a are approximately diagonal (low P-MAE) and that

the amplitude variation among the maximum points is minimal (low A-MAE). We therefore conclude that the system is locally laterally shift-invariant.

In Fig. III.2b the beam density is reduced by a factor of two compared to Fig. III.2a. The geometric distortion is now increased to a level where the contours are not diagonal. The maximum scalloping loss is almost 4 dB. Hence, the imaging system is no longer locally laterally shift-invariant. We observe that the maximum peak is both geometrically distorted and attenuated when the point is located between two beams and the distance between the beams is too high.

III.3 Local Lateral Shift-Invariance of the Capon Beamformer

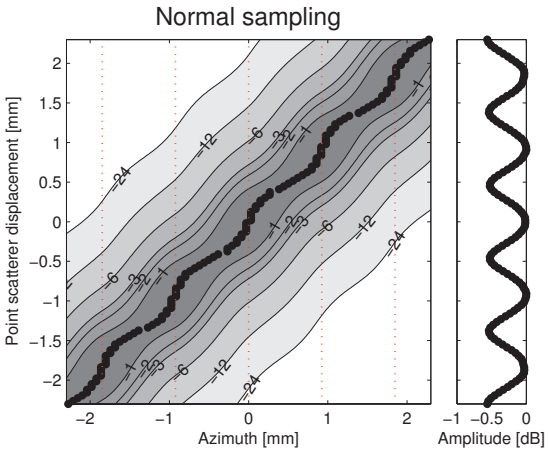
We will now investigate the local lateral shift-invariance of the Capon beamformer with high resolution parameters [4, 18]. In Fig. III.3a the same point scatterer response as presented in Fig. III.2a has been weighted with Capon weights calculated using (III.7) with $L = M/2 = 48$, $K = 1$ and $d = 1/100$. We see how the signal cancellation caused by the difference between the signal propagation vector and the assumed steering vector increases as the point moves away from a given transmit-receive beam. As a consequence the scalloping loss is tremendous, and not far from the predicted value in Fig. III.1.

The diagonal factor d and the effective adaptive aperture size L can be used to control the mainlobe width of the Capon beamformer. In the extreme cases, $d \gg 1$ and $L = 1$, the Capon beamformer becomes either a triangle or uniform-weighted DAS beamformer respectively, and a beam spacing according to (III.8) would then be sufficient. In Fig. III.3b we show how the scalloping loss is reduced when the amount of diagonal loading is increased.

We observe that Fig. III.3b now contains a scaled version of the pattern presented in Fig. III.2b. From Fig. III.3 it should be clear that Capon beamforming either requires increased lateral sampling, or the resolution has to be reduced to the same level as DAS to make the imaging system locally laterally shift-invariant. It would be unwise to do all the computation involved with Capon beamforming with a high d value and end up with a resolution equal to DAS. However, we see that diagonal loading can be a useful tool if resolution has to be traded for the need of oversampling. The same is true for the subarray length parameter.

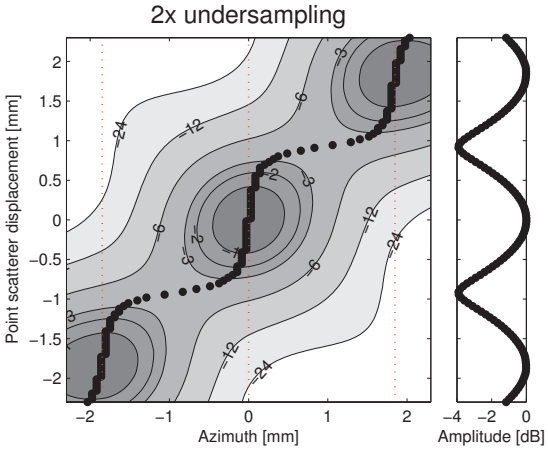
III.4 Oversampling Methods

In this section, we will discuss different methods aimed at improving the local lateral shift-invariance of the Capon beamformer. From the previous section, it should be clear that we somehow need to increase the lateral sampling to make the Capon imaging system locally laterally shift-invariant. In this section, we will investigate how this oversampling should be conducted, how large it must be, and how the proposed methods will affect the acquisition frame rate and the processing complexity.



P–MAE: 0.07 mm A–MAE: 0.25 dB (max –0.56 dB)

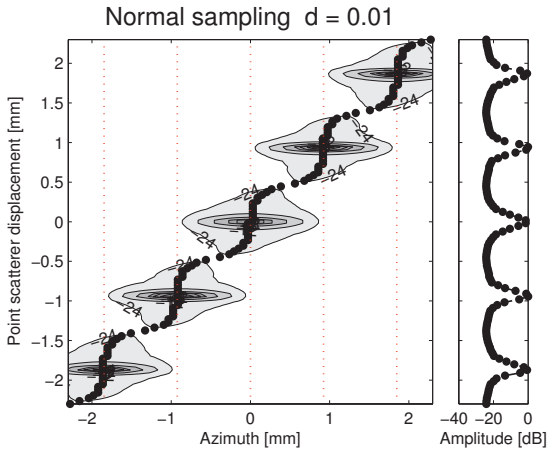
(a)



P–MAE: 0.28 mm A–MAE: 1.29 dB (max –3.91 dB)

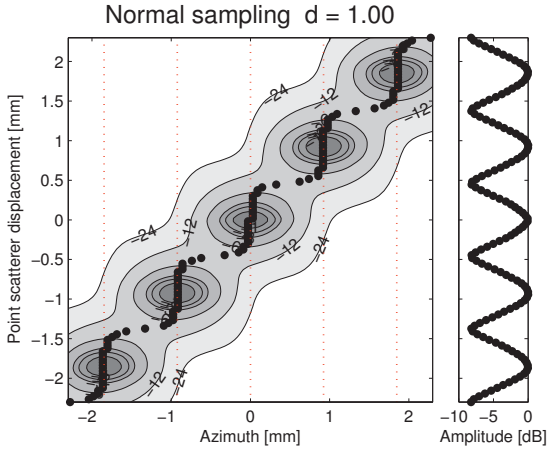
(b)

Figure III.2: LSV-plots of delay-and-sum beamforming. If contours are diagonal and the amplitude variation among the peak points (black dots) is small, the imaging system is said to be locally laterally shift-invariant. Dotted vertical lines are plotted where transmit-receive beams are located. a) Sampling density according to (III.8). The system is locally laterally shift-invariant. b) Undersampling by a factor of two. The system is no longer locally laterally shift-invariant.



P–MAE: 0.14 mm A–MAE: 18.50 dB (max –24.14 dB)

(a)



P–MAE: 0.17 mm A–MAE: 3.61 dB (max –8.22 dB)

(b)

Figure III.3: LSV-plots of Capon beamforming with $L = M/2 = 48$ and $K = 1$, and with sampling corresponding to (III.8). a) Capon beamforming with $d = 1/100$ as diagonal loading. The system is not locally laterally shift-invariant. b) The diagonal loading factor d is increased to 1. The scalloping loss is reduced, but the system is still not locally laterally shift-invariant.

III.4.1 Oversampling on Transmit

A straightforward way of obtaining sufficient sampling in an active imaging system with an equal number of transmit (tx) and receive (rx) beams is to increasing the number of tx-beams until the system is locally laterally shift-invariant. In Fig. III.4a and Fig. III.4b we have increased the tx-beam density by q-factors of 4 and 16 respectively. As expected, the geometric distortion and scalloping loss decrease with increased lateral sampling. Note that Fig. III.4b is zoomed-in compared to Fig. III.3 to better see the beam-to-beam variation.

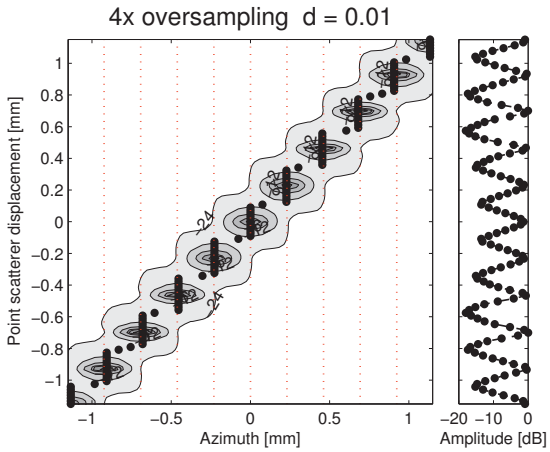
From Fig. III.4b, we see that 16 times oversampling reduces the scalloping loss significantly. Even though high-frequency variations are reduced, the maximum is still larger than 1 dB. An even higher level of oversampling is therefore required to have a completely local lateral shift-invariant Capon beamformer when a moving point scatterer is simulated without noise and with the same parameters as in Fig. III.3. Note that a diagonal loading factor of 1/100 will introduce a white noise component in the sample covariance matrix 20 dB lower than the signal present. Hence, the covariance has been estimated from data which appeared to have an SNR of 20 dB.

Even though increasing the number of transmits improves the lateral shift-invariance, it will also reduce the acquisition frame rate with a factor equal to the oversampling factor. The processing complexity will also increase equally. For these two reasons, oversampling on transmit is something which is desirable to avoid in real time ultrasound applications, and therefore we will not discuss this method further.

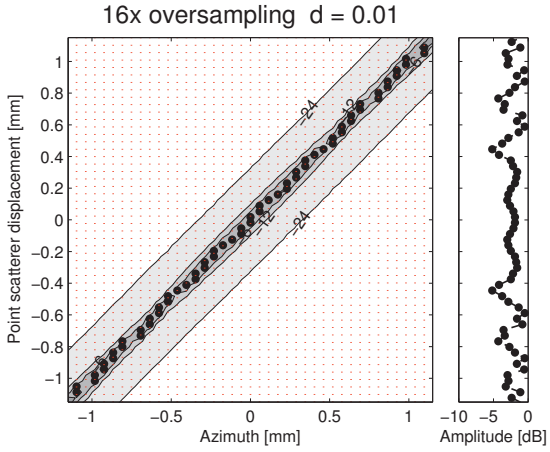
III.4.2 Oversampling by Parallel Receive Beamforming

Parallel receive beamforming (PRB) means that multiple narrow receive beams (rx-beams) are constructed from one broad tx-beam. Then, by reducing the number of tx-beams and relying on multiple rx-beams to get sufficient sampling on receive, it is possible to achieve high frame rate imaging. However, PRB also introduces geometrical distortion which, to some degree, can be compensated for by using, e.g., synthetic transmit beamforming (STB) [22, 25]. In [26], it is shown that STB works well with Capon beamforming, something which is also verified in [18]. In the following discussion, we consider corrected PRB as being equal to oversampling on transmit when it comes to image quality. The LSV-plots for PRB are therefore not included.

In cardiac ultrasound imaging, PRB is typically used to keep the number of rx-beams high while the number of tx-beams are reduced. When using PRB for oversampling, we would like to maintain the same number of tx-beams while the number of rx-beams are increased. The acquisition frame rate will then not be affected. On the other hand, more rx-beams will increase the number of covariance matrices and thereby the number of matrix-inversions with a factor equal to the number of parallel receive lines. The processing times reported in [18] will then also increase with the same factor. Because more than 16 times oversampling is required to achieve lateral-shift-invariant and high-resolution imaging of a moving point scatterer, none of the implementations described in [18] would be possible in real time. We therefore need a different way of oversampling our imaging sector.



(a)



(b)

Figure III.4: Capon beamforming as described in Fig. III.3 with oversampling on transmit. The figure is zoomed-in compared with Fig. III.3 to better see the beam-to-beam variation. a) 4 times oversampling ($q = 4$). b) 16 times oversampling ($q = 16$). The maximum scalloping loss is reduced, but it is still too high for the system to be locally laterally shift invariant.

III.4.3 Oversampling by Phase Rotation

The Capon beamformer formula in (III.7) gives a direct opportunity for oversampling using phase rotation. The steering vector $\mathbf{a}$ has in previous work been set to $\mathbf{1}$ because ultrasound data must be pre-delayed. However, the steering vector can also, as in narrow band applications, be varied over a set of predefined vectors. In a narrow band setting, the beam can make a sweep across the whole imaging sector through phase steering. This is also possible in broadband applications as long as the phase rotation is less than one pulse length (See coarse-fine beamforming in [27] and its references). The maximal steering angle is approximately given by

$$\theta_{max} = 2\alpha\delta\theta, \tag{III.9}$$

where α is the pulse length in terms of λ, and 2 and $\delta\theta$ is the oversampling factor and beam spacing from (III.8). For our simulations $\theta_{max} = 3\delta\theta$. It is therefore possible to make a sweep based on phase rotation between a given beam and half-way to its two nearest neighbours without introducing large errors. Phase rotation is actually what the Capon beamformer uses internally to micro-steer away from bright point scatterers and to enhance edges.

For a uniform linear array located along the x-axis the steering vector $\mathbf{a}_\theta$ should be calculated as

$$\mathbf{a}_\theta = \begin{bmatrix} e^{-j\frac{2\pi}{\lambda_c}x_0\sin(\theta)} \\ e^{-j\frac{2\pi}{\lambda_c}x_1\sin(\theta)} \\ \vdots \\ e^{-j\frac{2\pi}{\lambda_c}x_{L-1}\sin(\theta)} \end{bmatrix} \tag{III.10}$$

where x_i is the element position and θ is swept from $-\delta\theta/2$ to $\delta\theta/2$. Note that the steering vector must be calculated for an L element symmetric subarray, even though the full array has M elements. This subarray should have the same element pitch, $|x_i - x_{i-1}|$, as the full array.

The steering angles can be spread out in many different ways. If the number of steering angles is N_θ we have selected a uniform distribution of θ with $N_\theta/2$ angles on each side of the original steering direction when N_θ is even.

Figure III.5 shows the phase rotation method applied on the same scan sequence as in Fig. III.3 with 4 and 16 times oversampling (q-factor) on receive respectively. Note that the figure is zoomed-in compared with Fig. III.3. As with oversampling on transmit, there is some variation when the point moves in and out of focus. The maximum scalloping loss is therefore large. The extreme local variations are, however, reduced significantly. Comparing Fig. III.4 with Fig. III.5, the errors introduced by the phase rotation method are best seen in sidelobes.

Since the phase rotation method operates on the same channel data as in Fig. III.3a, there will be no reduction in the acquisition frame rate by applying this method. In addition, since the same number of covariance matrices are constructed and inverted, there also is no extra cost involved. The only additional cost will be the multiplication of the extra steering vectors with the inverted covariance matrix. If the equation $\hat{\mathbf{R}}\mathbf{a} =$

$\mathbf{b}$, as in [18], is solved using the Gauss-Jordan algorithm, we can solve the problem for all steering vectors at once by row reducing the matrix $[\hat{\mathbf{R}}, \mathbf{a}_{\theta_1}, \mathbf{a}_{\theta_2}, \cdots \mathbf{a}_{\theta_{N_\theta}}]$ to the form $[\mathbf{I}, \mathbf{b}_{\theta_1}, \mathbf{b}_{\theta_2}, \cdots \mathbf{b}_{\theta_{N_\theta}}]$. This is an $O(L^3)$ operation as long as N_θ is not much greater than L.

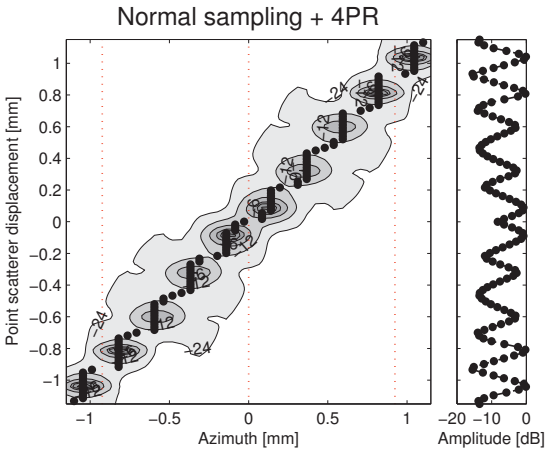
III.5 In Vitro Phantom Data

In Fig. III.6 we present the result of applying the Capon beamformer on *in vitro* data with and without oversampling by phase rotation. The data was acquired using a modified Vivid E9 high-end cardiac ultrasound scanner (GE Vingmed Ultrasound, Horten, Norway) equipped with a M5S-D phased transducer operating in harmonic mode at 3.34 MHz. The transmit beam density was set to match (III.8) based on this frequency, and 111 tx-beams where therefore distributed on a 65°sector. Transmit focus was located at 8 cm range. Note that in harmonic imaging, the generated transmit beam at the harmonic frequency will be wider and have lower sidelobes than the beam resulting from transmitting at the harmonic frequency [28]. The lateral sampling is therefore increased compared with our simulations. A tissue mimicking phantom containing wire targets and anechoic cysts was imaged, and the data was acquired while the probe was moved across the phantom manually. Fig. III.6a shows the first frame out of 14 in the acquired loop processed with DAS beamforming.

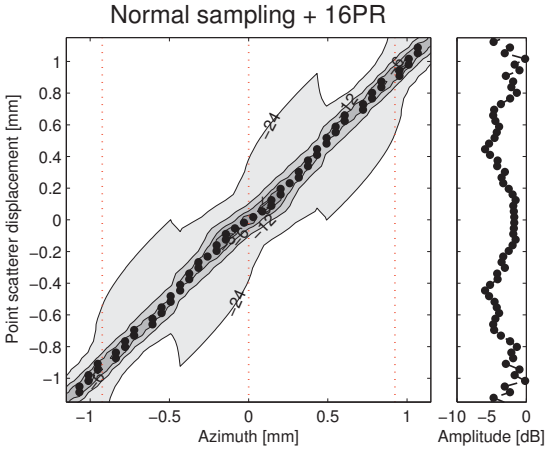
One point scatterer in Fig. III.6a has been selected [white box], and then used to generate LSV-plots. The point scatterer is located close to the transmit focus. Figure III.6b shows the resulting LSV-plot when using the DAS beamformer. The scalloping loss is less than 1 dB, and the system is by that locally laterally shift-invariant. As a consequence point scatterers have a consistent appearance when subjected to motion **[Media Movie 1]**.

Figure III.6c shows the same point scatterer response processed with the Capon beamformer without oversampling. For the frame where the point scatterer is at -6 degrees the receive beam direction is very close to the signal propagation vector. Hence, we see a high amplitude value at this location. In almost all the other frames the signal loss is massive (at most 10 dB). The result is blinking point scatterers when subjected to motion **[Media Movie 2]**.

Figure III.6d presents the result of applying sixteen times oversampling by phase rotation. We observe how the amplitude of the point scatterer is better preserved in all frames, and the scalloping loss is not far from our 1 dB threshold. Note that the oscillation along the diagonal does not indicate shift-variance. It is a result of lack of control when the probe was manually dragged across the phantom. To get a diagonal without peaks the probe has to be dragged across the phantom with a speed which matches the acquisition frame rate. In addition, because of the increase in lateral resolution, the Capon beamformer requires smaller frame-to-frame movements than DAS. Looking at the video of Capon beamforming with sixteen times oversampling by phase rotation **[Media Movie 3]** the blinking is now highly reduced. The same can be seen from Fig. III.7 where point scatterers appear brighter in the image with oversampling by phase rotation.



(a)



(b)

Figure III.5: Capon beamforming as described in Fig. III.3 with oversampling based on phase rotation. The figure is zoomed-in compared with Fig. III.3 to better see the beam-to-beam variation. a) 4 times oversampling on receive ($q = 4$). b) 16 times oversampling on receive ($q = 16$).

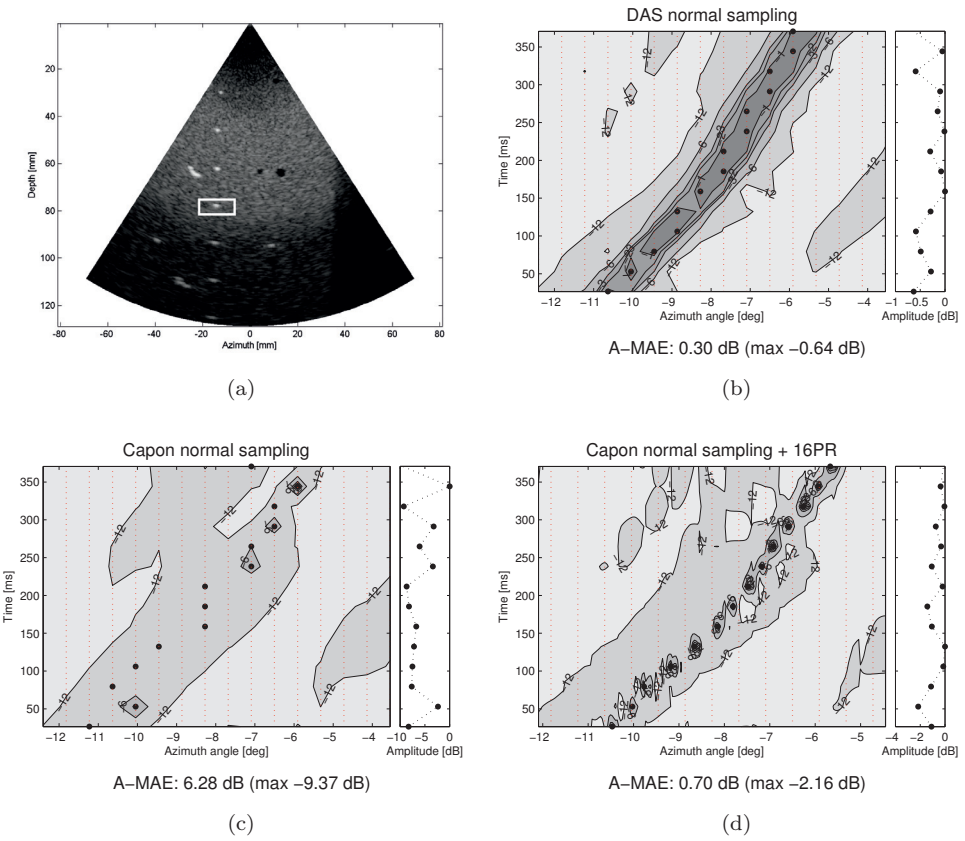
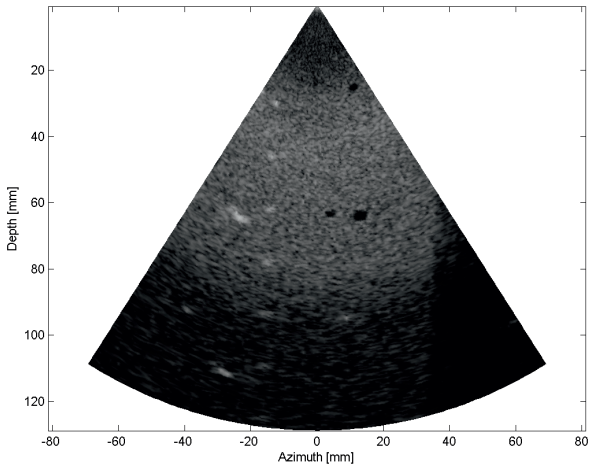


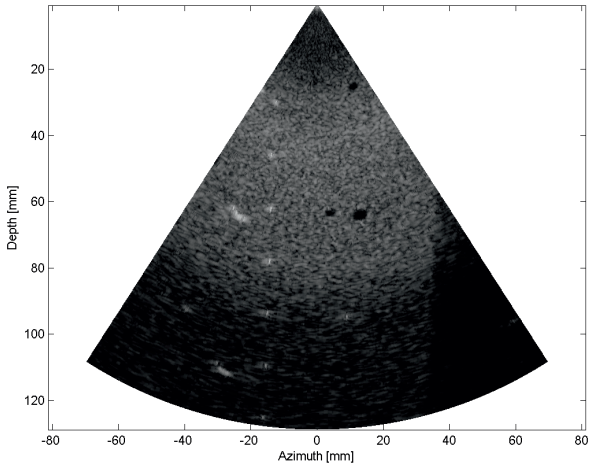
Figure III.6: LSV-plots generated from *in vitro* phantom data where the probe has been manually dragged across the phantom. See the supplementary material for videos of the whole loop processed with the different beamformers. Capon beamforming has been performed with the same settings as in Fig. III.3a. P-MAE is not shown due to a lack of probe movement control. a) First frame in the recorded loop processed with delay-and-sum beamforming [Media Movie 1]. A white box indicates the point scatterer used to construct the LSV-plots. b) LSV-plot for delay-and-sum beamforming. c) LSV-plot for Capon beamforming without oversampling. d) LSV-plot for Capon beamforming with sixteen times oversampling using phase rotation.

III.6 Discussion

Faced with an imaging scene containing bright point scatterers or sharp edges, one could wonder how large the oversampling factor needs to be to avoid visible scalloping losses. In his seminal paper, Cox [15] derives the required beam density for a given output SNR. For instance 47 dB requires a beam density for the Capon beamformer



(a)



(b)

Figure III.7: First frame from Fig. III.6a processed with Capon beamforming. a) Without oversampling **[Media Movie 2]**. b) With sixteen times oversampling using phase rotation **[Media Movie 3]**. Notice how the point scatterers appear brighter in the image with oversampling.

ten times the one given by (III.8) [with $q = 1/2$]. Since resolution for the Capon beamformer depends on SNR we would recommend to first use the dynamic range in the image as guidance. The amount of diagonal loading will also dictate the maximal

achievable resolution since $10 \log d$ can be interpreted as the maximal input SNR seen by the Capon beamformer. Fig. III.1 shows that 25 times oversampling is required for a d-value of 1/100. When *in vitro* data contains additional noise this oversampling requirement is reduced.

Relating the presented phase rotation method to earlier work we find that it is similar to a steering vector uncertainty set. However, where these set-based methods typically have a criterion for selecting the best output after applying all the steering vectors contained in the set, oversampling by phase rotation outputs all. It can also be thought of as a narrowband approach to parallel receive beamforming where Capon's method is applied on all receive beams.

Oversampling by phase rotation can therefore introduce similar block artifacts as parallel receive beamforming (PRB). To remove these artifacts we could distribute our steering vectors in an overlapping scheme [like rx-beams are spread out for STB]. We then get a phase rotation version of STB where e.g. eight phase rotated beams are averaged into four. Whether the geometric distortion, or stripes, seen at the bottom of the sector in the video with 16 times oversampling **[Media Movie 3]** is removed by such technique is left for future work.

When oversampling is applied it becomes evident that the high-resolution settings used in this paper might be too extreme, both with respect to the level of required oversampling and the point scatterer's appearance. Traditionally in medical ultrasound imaging, the point spread function is always wider laterally than in range. With the Capon beamformer this is now changed. A more symmetric point spread function while preserving speckle statistics is obtained with the parameters $L = M/4$ and $K = 0$ [14]. The level of required oversampling and the computational complexity are also reduced. Note that oversampling is not only essential to keep the amplitude level of moving point scatterers. It should also help to improve contrast and edge definitions. If oversampling is not applied, homogeneous speckle regions could get a too-low average value and the edge delineation of a moving object will fluctuate in the same way as a moving point scatterer would have sparkled.

Finally, we want to stress the fact that this demand for oversampling does not only affect the Capon [or minimum variance beamformer]. Other methods claiming super resolution, like the Eigenspace method, low complexity beamformer etc. [29–32], would also require oversampling when improved lateral resolution is obtained. The same applies for high resolution spectral estimation [33].

III.7 Conclusion

In this paper we have investigated the lateral shift-invariance property of the Capon beamformer when imaging moving objects. It was shown that Capon beamforming with the same beam density as required for delay-and-sum beforming results in blinking point scatterers when the system is subjected to motion. We then discussed and investigated different methods based on oversampling on transmit and receive to reduced this effect. Finally, we showed that the local lateral shift-invariance could be improved by oversampling based on phase rotation on receive. This without affecting

the acquisition frame rate and with a minor change in processing complexity. In simulations, this method reduced the maximum scalloping loss by 16 dB. On *in vitro* data, the reduction was around 7 dB and the point scatterer blinking was highly reduced.

Acknowledgment

The authors thanks A. Sørnes at GE Vingmed Ultrasound, Horten, Norway for help in acquiring the *in vitro* data.

References

[1] J.-F. Synnevåg, A. Austeng, and S. Holm, "Adaptive Beamforming Applied to Medical Ultrasound Imaging," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 54, no. 8, pp. 1606–1613, Aug. 2007.

[2] F. Vignon and M. R. Burcher, "Capon beamforming in medical ultrasound imaging with focused beams." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 55, no. 3, pp. 619–628, Mar. 2008.

[3] F. Viola and W. Walker, "Adaptive signal processing in medical ultrasound beamforming," in *Proc. IEEE Ultrasonics Symp.*, 2005, pp. 1980–1983.

[4] J.-F. Synnevåg, A. Austeng, and S. Holm, "Benefits of minimum-variance beamforming in medical ultrasound imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 9, pp. 1868–1879, Sep. 2009.

[5] J. Chen, B. Y. S. Yiu, H. K. So, and A. C. H. Yu, "Real-Time GPU-Based Adaptive Beamformer for High Quality Ultrasound Imaging," in *Proc. IEEE Ultrasonics Symp.*, 2011, pp. 1–4.

[6] X. Mestre and M. Lagunas, "Finite sample size effect on minimum variance beamformers: optimum diagonal loading factor for large arrays," *IEEE Transactions on Signal Processing*, vol. 54, no. 1, pp. 69–82, Jan. 2006.

[7] B. Widrow, K. Duvall, R. Gooch, and W. Newman, "Signal cancellation phenomena in adaptive antennas: Causes and cures," *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 3, pp. 469–478, May 1982.

[8] M. Wax and Y. Anu, "Performance analysis of the minimum variance beamformer," *IEEE Transactions on Signal Processing*, vol. 44, no. 4, pp. 928–937, Apr. 1996.

[9] ——, "Performance analysis of the minimum variance beamformer in the presence of steering vector errors," *IEEE Transactions on Signal Processing*, vol. 44, no. 4, pp. 938–947, Apr. 1996.

[10] T.-J. Shan and T. Kailath, "Adaptive beamforming for coherent signals and interference," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, no. 3, pp. 527–536, Jun. 1985.

[11] Jian Li, Petre Stoica, and Zhisong Wang, "On robust capon beamforming and diagonal loading," *IEEE Transactions on Signal Processing*, vol. 51, no. 7, pp. 1702–1715, Jul. 2003.

[12] R. Lorenz and S. Boyd, "Robust minimum variance beamforming," *IEEE Transactions on Signal Processing*, vol. 53, no. 5, pp. 1684–1696, May 2005.

[13] M. Rubsamen and M. Pesavento, "Maximally Robust Capon Beamformer," *IEEE Transactions on Signal Processing*, vol. 61, no. 8, pp. 2030–2041, Apr. 2013.

[14] J.-F. Synnevåg, C.-I. C. Nilsen, and S. Holm, "Speckle Statistics in Adaptive Beamforming," in *Proc. IEEE Ultrasonics Symp.* IEEE, Oct. 2007, pp. 1545–1548.

[15] H. Cox, "Resolving power and sensitivity to mismatch of optimum array processors," *J. Acoust. Soc. Am.*, vol. 54, no. 3, pp. 771–785, 1973.

[16] F. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[17] J. P. Åsen, J. I. Buskenes, C.-I. C. Nilsen, A. Austeng, and S. Holm, "Implementing Capon Beamforming on the GPU for Real Time Cardiac Ultrasound Imaging," in *Proc. IEEE Ultrasonics Symp.*, 2012, pp. 2133–2136.

[18] ——, "Implementing capon beamforming on a GPU for real-time cardiac ultrasound imaging." *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 61, no. 1, pp. 76–85, Jan. 2014.

[19] A. Jensen and A. Austeng, "An approach to multibeam covariance matrices for adaptive beamforming in ultrasonography," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 59, no. 6, pp. 1139–1148, Jun. 2012.

[20] J. Capon, "High-resolution frequency-wavenumber spectrum analysis," *Proceedings of the IEEE*, vol. 57, no. 8, pp. 1408–1418, 1969.

[21] T. Hergum, S. Langeland, E. W. Remme, and H. Torp, "Fast ultrasound imaging simulation in K-space." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 6, pp. 1159–67, Jun. 2009.

[22] T. Hergum, T. Bjåstad, K. Kristoffersen, and H. Torp, "Parallel beamforming using synthetic transmit beams," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 54, no. 2, pp. 271–280, Feb. 2007.

[23] J. A. Jensen and N. B. Svendsen, "Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 39, no. 2, pp. 262–267, Jan. 1992.

[24] J. A. Jensen, "Field: A program for simulating ultrasound systems," *Medical & Biological Engineering & Computing*, vol. 34, no. Supplement 1, Part 1, pp. 351–353, 1996.

[25] B. Denarie, T. A. Tangen, I. K. Ekroll, N. Rolim, H. Torp, T. Bjåstad, and L. Løvstakken, "Coherent plane wave compounding for very high frame rate ultrasonography of rapidly moving targets." *IEEE Trans. Med. Imag.*, vol. 32, no. 7, pp. 1265–76, Jul. 2013.

[26] A. Rabinovich, Z. Friedman, and A. Feuer, "Multi-Line Acquisition With Minimum Variance Beamforming in Medical Ultrasound Imaging," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 60, no. 12, 2013.

[27] K. Thomenius, "Evolution of ultrasound beamformers," in *Proc. IEEE Ultrasonics Symp.*, vol. 2, 1996, pp. 1615–1622.

[28] R. Fedewa, K. Wallace, M. Holland, J. Jago, G. Ng, M. Rielly, B. Robinson, and J. Miller, "Spatial coherence of backscatter for the nonlinearly produced second harmonic for specific transmit apodizations," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 51, no. 5, pp. 576–588, May 2004.

[29] J.-F. Synnevåg, A. Austeng, and S. Holm, "A Low-Complexity Data-Dependent Beamformer." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 58, no. 2, pp. 281–289, Feb. 2011.

[30] C.-I. C. Nilsen and I. Hafizovic, "Beamspace adaptive beamforming for ultrasound imaging." *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 56, no. 10, pp. 2187–2197, Oct. 2009.

[31] S. Mehdizadeh, A. Austeng, T. F. Johansen, and S. Holm, "Eigenspace based minimum variance beamforming applied to ultrasound imaging of acoustically hard tissues." *IEEE Trans. Med. Imag.*, vol. 31, no. 10, pp. 1912–21, Oct. 2012.

[32] K. Kim, S. Park, Y.-T. Kim, S.-C. Park, J. Kang, J.-H. Kim, and B. MooHo, "Flexible Minimum Variance Weights Estimation Using Principal Component Analysis," in *Proc. IEEE Ultrasonics Symp.*, 2012.

[33] I. Ekroll, H. Torp, and L. Løvstakken, "Spectral doppler estimation utilizing 2-D spatial information and adaptive signal processing," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 59, no. 6, pp. 1182–1192, Jun. 2012.

Paper IV

Adaptive Volume Rendering of Cardiac 3D Ultrasound Images - Utilizing Blood Pool Statistics

Jon Petter Åsen[1], Erik Steen[2], Gabriel Kiss[3,4], Anders Thorstensen[1,5], Stein Inge Rabben[2]

[1]Medical Imaging Lab (MI-Lab), Norwegian University of Science and Technology, Trondheim, Norway

[2] GE Vingmed Ultrasound, Horten, Norway

[3] Norwegian University of Science and Technology, ISB - Department of Circulation and Medical Imaging, Trondheim, Norway

[4] St. Olavs Hospital, Trondheim, Norway

[5] Department of Cardiology, St. Olavs Hospital, Trondheim, Norway

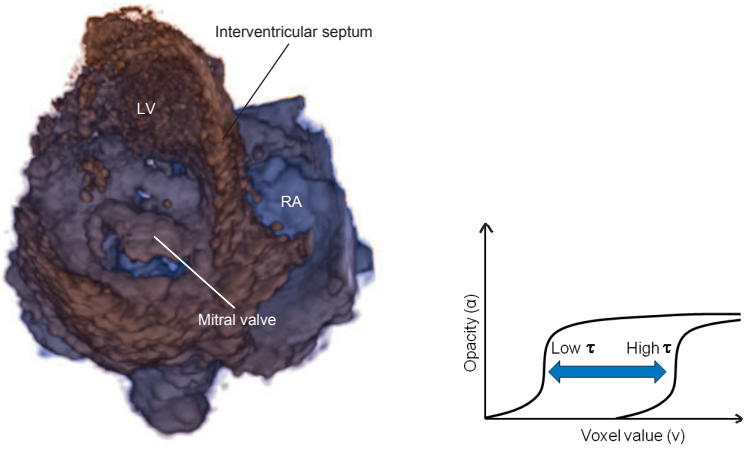
Proc. SPIE Medical Imaging 2012, vol. 8320, pp. 832008.

In this paper we introduce and investigate an adaptive direct volume rendering (DVR) method for real-time visualization of cardiac 3D ultrasound. DVR is commonly used in cardiac ultrasound to visualize interfaces between tissue and blood. However, this is particularly challenging with ultrasound images due to variability of the signal within tissue as well as variability of noise signal within the blood pool. Standard DVR involves a global mapping of sample values to opacity by an opacity transfer function (OTF). While a global OTF may represent the interface correctly in one part of the image, it may result in tissue dropouts, or even artificial interfaces within the blood pool in other parts of the image. In order to increase correctness of the rendered image, the presented method utilizes blood pool statistics to do regional adjustments of the OTF. The regional adaptive OTF was compared with a global OTF in a dataset of apical recordings from 18 subjects. For each recording, three renderings from standard views (apical 4-chamber (A4C), inverted A4C (IA4C) and mitral valve (MV)) were generated for both methods, and each rendering was tuned to the best visual appearance by a physician echocardiographer. For each rendering we measured the mean absolute error (MAE) between the rendering depth buffer and a validated left ventricular segmentation. The difference d in MAE between the global and regional method was calculated and t-test results are reported with significant improvements for the regional adaptive method ($\bar{d}_{A4C} = 1.5 \pm 0.3$ mm, $\bar{d}_{IA4C} = 2.5 \pm 0.4$ mm, $\bar{d}_{MV} = 1.7 \pm 0.2$ mm, d.f. $= 17$, all $p < 0.001$). This improvement by the regional adaptive method was confirmed through qualitative visual assessment by an experienced physician echocardiographer who concluded that the regional adaptive method produced rendered images with fewer tissue dropouts and less spurious structures inside the blood pool in the vast majority of the renderings. The algorithm has been implemented on a GPU, running an average of 16 fps with a resolution of 512x512x100 samples (Nvidia GTX460).

IV.1 Introduction

Direct volume rendering (DVR)[1] is commonly used in cardiac ultrasound to visualize interfaces between tissue and blood[2–4]. DVR has the advantage over other methods like 2D-slicing that it presents a large part of the volumetric data in a single view. Figure IV.1a presents such a volume rendering of a high quality cardiac ultrasound volume. Anatomical structures of special interest are highlighted; the left ventricle (LV), the mitral valve which separates LV from the left atrium (LA), and the interventricular septum which separates the LV from the right ventricle (RV). As can be seen, interpreting cardiac ultrasound data is difficult and requires a lot of training. Clear tissue-blood boundaries are therefore of great importance. However, this is challenging to achieve with ultrasound images due to variability of the signal within tissue as well as variability of noise signal within the blood pool.

At the core of DVR, optical properties (colour and opacity) are derived from a transfer function and integrated along rays of light. One ray for each pixel in the final



(a) (b)

Figure IV.1: a). An example of direct volume rendering (DVR) of cardiac ultrasound. Highlighted anatomical structures are: The left ventricle (LV), the right atrium (RA), mitral valve and the interventricular septum. b). Shows two opacity transfer functions (OTF) with a fuzzy transition between voxels excluded from (values below τ) or included in the rendering (values above τ). Adjusting τ (blue arrow) will either include or exclude more data.

image. Applying the transfer function is also known as classification since features are visually separated through assignment of different colours and opacities. On the other hand, selecting a transfer function with low classification error is considered a challenge and will always be related to the data at hand [5]. Figure IV.1b shows an example of a monotonically increasing opacity transfer function (OTF) for cardiac ultrasound DVR. Unlike other medical imaging modalities like CT and MR, cardiac ultrasound data are assumed to consist of only two classes; tissue and the blood pool signal. Tissue is expected to have higher intensity than blood, but in practice the two class-distributions overlap. The OTF in Figure IV.1b is therefore designed to give a fuzzy threshold between samples excluded from and samples included in the rendering. Selecting the right threshold, as for transfer functions design in general, is therefore a challenge in cardiac ultrasound visualization.

There are several reasons for the overlap between tissue and blood pool signal in ultrasound images. Sidelobes of the ultrasound beam results in signal from off-beam structures (clutter noise). Speckle pattern, caused by interference between point scatterers results in high signal variance in homogeneous regions. Multiple reflections between tissue structures (reverberations) causes false echoes. Distortion of the ultrasound wave front caused by varying speed of sound in inhomogeneous tissue (aberrations) reduce the signal amplitude. Interfaces of high reflectivity (lungs, ribs

and calcifications) and how tissue layers are aligned with the ultrasound beam result in drop-outs and varying tissue signal. Attenuation of the ultrasound signal with depth gives high variation in tissue intensity across the volume. While an optimal global OTF may represent the tissue-blood interface correctly in one part of the image, the high variation in tissue intensities may result in tissue dropouts, or even artificial interfaces within the blood pool in other parts of the image. The latter artefact is visible in the upper part of LV in Figure IV.1a. Figure IV.1a also shows a large dropout in the lateral wall. These problems are to some extent solved by clipping, but standard clipping also has its limitations due to planar geometry. The curved shape of cardiac chambers calls either for other clipping geometries or a way to better assign high opacity to voxels associated with tissue and low opacity otherwise[6].

A lot of work has been done in the visualization community to improve the results given by standard DVR. Special attention has been devoted to the problem of selecting the right transfer function[7–13] and exploration of illustrative or non-photorealistic techniques to bring important information into focus[14–18]. Recently, Marchesin et al. [19] introduced a relevance function for feature enhancement in volume rendering. The paper differs from earlier contributions by also suggesting how to derive a relevance map without having any prior knowledge of voxel priorities. However, the use of local gradients as relevance indicators together with the per-ray constant opacity, yielding transparent renderings, makes it unsuitable for cardiac ultrasound. In our approach we adapt the ideas given by Marchesin et al. on how to do local opacity modulations, but with a relevance map better suited for cardiac ultrasound. For medical applications many adaptive and semi-adaptive DVR-like techniques have been proposed for enhancing anatomical structures[20–23]. For ultrasound, Hönigmann et al.[24] proposed how to design a global adaptive OTF used to visualize volumetric fetal images. They also discussed the possibility of calculating one OFT for different regions in the rendering. Such a regional OTF can be derived from e.g. local edge detection or histograms. The method proposed in this paper regionally adapts the OTF by utilizing blood pool statistics derived from a real-time LV endocardial tracking algorithm[25]. The purpose is to increase the correctness of the rendered image by reducing artificial tissue dropouts and spurious structures inside the blood pool.

IV.2 Methods

IV.2.1 Blood pool based regional adaptive opacity transfer function

Selecting the correct global OTF for cardiac ultrasound visualization can be viewed as a global thresholding problem. Therefore we define the opacity threshold τ for a monotonically increasing OTF α as the highest sample value v where all samples with lower value than v will get zero opacity, $\tau = \max_i(v_i),\ \alpha(v_j) = 0\ \ \forall\, j < i$, hence they will be rendered transparent. As OTF we use a truncated polynomial:

$$\alpha(v) = \begin{cases} 0, & v < \tau, \\ \min(a(v-\tau)^\gamma, k), & v \geq \tau. \end{cases} \tag{IV.1}$$

Global transparency is controlled by $a \in [0, \infty\rangle$, and in this paper $a = 80$ has been selected to give a steep OTF. The truncation and gamma value are set to $k = 0.5$ and $\gamma = 2$ respectively. A second order polynomial has been selected instead of a standard linear OTF to increase the transparency of values near the opacity threshold. These values have a higher probability for being noise, and we let the opacity reflect this.

Since the global distributions of tissue and blood pool voxels usually overlap, adjusting τ will always involve a tradeoff between the number of true-positive and false-positive tissue voxels. If the opacity threshold is set too high, tissue surfaces will start to fall apart. However, if set too low, low level noise will get high opacity and the rendering will end before the actual tissue border is reached. Figure IV.2a depicts the problem of having a global OTF (in orange). If rays r_i and r_j have noise (n_r) and tissue (s_r) levels given by n_i, s_i, n_j, s_j, with $n_i < s_i < n_j < s_j$, there are two obvious choices for τ. The global opacity threshold τ can be selected as $n_i < \tau < s_i$ or $n_j < \tau < s_j$. The first choice makes n_j occlude s_j, however the second choice will render s_i transparent. This will result in an artificial hole in the final rendering along r_i. With our regional adaptive OTF, the idea is to let the opacity threshold τ vary across the framebuffer, making us able to define individual opacity thresholds for r_i and r_j (shown in green in Figure IV.2a).

In optical character recognition there has been a long tradition to utilize adaptive thresholding schemes to deal with variable background levels in 2D images[26], thus maximizing the amount of foreground output. One simple adaptive method is based on estimates of regional statistics. The local image threshold $\tau_{x,y}$ is calculated as

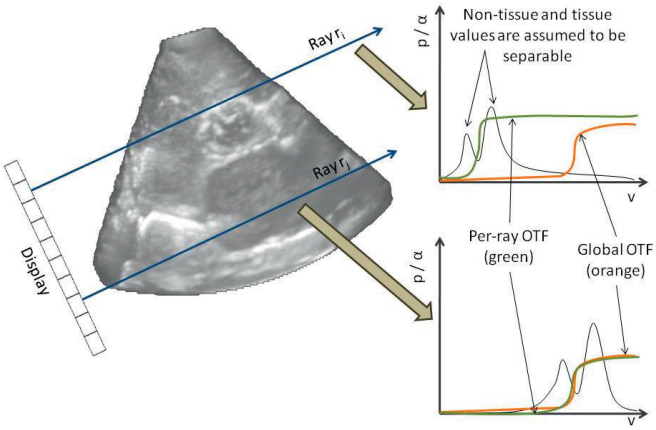
$$\tau_{x,y} = \mu_{x,y} + c\sigma_{x,y}, \tag{IV.2}$$

where c is a constant, and $\mu_{x,y}$ and $\sigma_{x,y}$ are mean and standard deviation estimates calculated in a window w around (x,y). This yields what is known as a thresholding surface, where the method assumes that the foreground has higher expected value than the background inside w. Hence, foreground pixels are located above the thresholding surface.

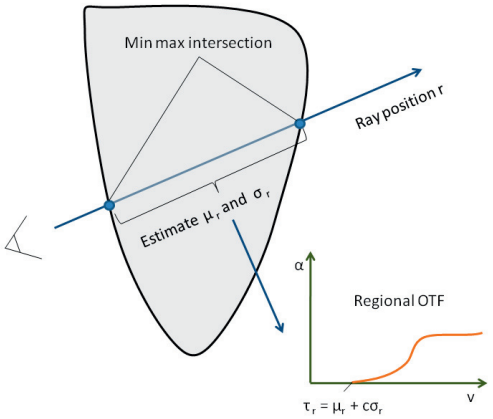
To address our problem with high variability in tissue intensity in ultrasound images, we have designed a regional adaptive OTF by estimating the regional means and standard deviations of Equation IV.2 from the blood pool of the LV. The blood pool was delineated by a state-estimation algorithm capable of tracking the LV endocardium in real-time[25]. The temporal state of the LV is returned by the framework as a time series of polygon meshes. How we utilize this information to do regional opacity adjustments is depicted in Figure IV.2b. For a given frame we estimate the signal mean and standard deviation, μ_r and σ_r, along rendering ray r. Note that only the blood pool, the intersection between r and the given LV delineation, is used to estimate μ_r and σ_r. The estimates are then used to calculate a regional opacity threshold τ_r given as

$$\tau_r = \mu_r + c\sigma_r, \tag{IV.3}$$

where c is a global constant adjusting τ_r around μ_r with a certain number of standard deviations. The local threshold τ_r together with Equation IV.1 form our blood pool based regional adaptive OTF. Rays that do not intersect the given LV delineation are rendered using a global OTF.



(a)



(b)

Figure IV.2: a) Shows ideal bimodal histograms of two density evolutions along ray r_i and r_j. If we apply a global OTF (orange) on the given case we must select one ray were we want to separate tissue from blood. However, with a regional adapted OTF (green) we can find individual thresholds τ_r that minimizes the amount of misclassified samples along both rays. b) Estimated values of mean μ_r and standard deviation σ_r inside the blood pool are used to regionally adjust the OTF. The regional opacity threshold $\tau_r = \mu_r + c\sigma_r$ is calculated for each ray r intersecting a given left ventricular delineation.

IV.2.2 Parameter space

The proposed method in Equation IV.3 introduces a parameter c, which controls how many standard deviations we want τ_r to deviate from the per-ray blood pool mean μ_r. This parameter is clearly a substitution for the global opacity threshold, and adjusting c will in the same way control the level of data visible in the rendered image. Setting $c = 0$ will render approximately 50% of the blood pool samples transparent. Since we have a robust estimate of blood pool statistics, tissue signals are expected to have $\mu_{tissue} > \mu_r$ and $c > 0$ should be used.

The shape of the OTF is of great importance for the final appearance of a volume rendering. An OTF with a gentle slope will for instance result in a transparent look. In the proposed method we are more interested in the opacity threshold than the actual shape of the OTF, therefore we use the simple shifted and truncated second degree polynomial from Equation IV.1 as a basis for the local OTF. Because it is a simple closed form expression, it can be used to quickly calculate on-the-fly opacity values.

For thresholding 2D images, the adaptive method in Equation IV.2 requires a window size that is directly related to the size of foreground objects. In our method, this window size depends on the ray-blood-pool intersection, and ideally no foreground (tissue) is included in the estimates. The regional opacity threshold τ_r is used along the full length of r, not just the intersection. Intersections with too few samples to give reliable statistics are rendered using a global OTF.

IV.2.3 Spatial and temporal regularization

Introducing adaptive visualization has some negative side effects. We trade spatial and temporal coherence in the presentation of anatomical structures for a better presentation of endocardial tissue. This does not pose any problems as long as the structure of interest along a given ray is associated with a value larger than the blood pool mean. Otherwise we risk rendering this structure transparent. The latter situation will often arise if e.g. the endocardium, mitral valve or papillary muscles are included in the estimated statistics. Given the properties of the real-time tracking framework, the two last structures, will to some extent, always be included. To exclude the endocardium we highly depend on a successful LV endocardial tracking. All segmentation algorithms will be prone to errors and two regularization schemes have therefore been tested. First, we do spatial smoothing of estimates across neighbouring rays. This will constrain the regional adaptivity. Second, we do temporal smoothing over successive frames to reduce the per-frame adaptivity and potential flickering[27]. The smoothing has been implemented as averaging filters on the framebuffer containing regional statistics. All regional adaptive renderings and statistics have been produced without any regularization in this paper. However the regularization schemes have been tested, and we will comment on how they perform.

	Apical 4-chamber	Inverted apical 4-chamber	Mitral valve
$\bar{d}$	1.5 ± 0.3 mm	2.5 ± 0.4 mm	1.7 ± 0.2 mm

Table IV.1: Statistics showing a significant improvement ($p < 0.001$, d.f. = 17) in mean absolute error (MAE) for the proposed regional adaptive method in three views. The variable $\bar{d}$ is the average difference plus-minus standard error between MAE_{global} and $\text{MAE}_{regional}$ in 18 subjects. All renderings are from end-diastole and were tuned to the best visual appearance by a physician echocardiographer.

IV.2.4 Rendering setup

All renderings have been produced with Phong lighting and the data have been pre-smoothed with anisotropic diffusion[28, 29]. For ray-casting it is normal to stop integrating when no significant transparency is left (known as early-ray-termination). We have chosen to terminate each ray when only 5% transparency is left. The rendering resolution has been set to 512x512x100.

IV.2.5 Measuring rendering accuracy

As a measure of rendering accuracy, we have calculated the mean absolute error (MAE) between the rendering depth buffer (given by early-ray-termination) and a validated left ventrical segmentation. The reference segmentation was performed by an experienced physician echocardiographer using the AutoLVQ tool from GE Vingmed Ultrasound. The MAE measure will increase if the rendering contains tissue dropouts, or if spurious structures occlude endocardial tissue. A minimal MAE therefore exists when a majority of rays end their traversal near the tissue-blood border depicted by the reference segmentation. A view-aligned clipping plane has been applied to produce standard long-axis and mitral valve views, clipping away half of both the ventricle and the reference segmentation. Clipping of the volume from behind has also been applied to reduce the impact of non-terminated rays versus rays terminated by clutter noise.

IV.3 Results

IV.3.1 Quantitative evaluation

The global and regional adaptive methods were compared using a dataset of apical recordings from 18 subjects. The dataset consisted of 8 healthy volunteers and 10 subjects with recent myocardial infarctions and were acquired using the GE Vivid 7 Dimensions system (GE Vingmed, Horten, Norway) and a 2.5MHz matrix array transducer (GE Vingmed, Horten, Norway), with the subjects in the left lateral decubitus position. Scans were taken from the apical imaging window, in harmonic mode, from 4 up to 6 QRS triggered sub-volumes, during an end-expiratory breath-hold. The depth and angle of the ultrasound sector were adjusted such that the entire

	Apical 4-chamber	Inverted apical 4-chamber	Mitral valve
Better	17	17	12
Equal	1	1	4
Worse	0	0	2

Table IV.2: Ratings given by an experienced physician echocardiographer when comparing the blood pool based regional adaptive OTF with a global OTF in three views.

LV was covered.

For each recording, three renderings from standard views (apical 4-chamber (A4C), inverted A4C and mitral valve (MV)) were generated for both the global and regional adaptive methods. Each rendering was then tuned to the best visual appearance by a physician echocardiographer. For each rendering we measured the MAE between the rendering depth buffer and a validated LV segmentation. The results of these measurements are reported in Table IV.1. The null hypothesis that $\bar{d}$ is less than or equal to zero, i.e. $\text{MAE}_{global} \leq \text{MAE}_{regional}$, can be rejected ($p < 0.001$) for all views using t-statistics with d.f. = 17.

The algorithm has been implemented on a GPU using CUDA. It runs at an average speed of 16 fps with an Nvidia GTX 460 graphics card. Note that this card has half the texture performance in CUDA compared to Direct3D. To estimate statistics we need to add a second pass through the volume to the rendering pipeline, reducing performance from 26 fps for a global OTF to 16 fps. As mentioned the rendering resolution is 512x512x100.

IV.3.2 Qualitative evaluation

For each pair of renderings, an experienced physician echocardiographer was asked to answer the following questions regarding image quality: *Does the blood pool based regional adaptive OTF produce a rendered image with less tissue dropouts and spurious structures inside the blood pool compared with the global OTF?*. This qualitative evaluation is summarized in Table IV.2. As seen, he concluded that the regional adaptive method performed better in 46 of the 56 renderings, and only worse in two.

IV.3.3 Examples

Figures IV.3a, b and c show three renderings of the same apical 4-chamber view of one subject. With a low opacity threshold (Figure IV.3a), a lot of clutter noise is picked up in the apex of the LV (circle). With a threshold corresponding to the minimum MAE in Figure IV.3d, the clutter noise is removed (Figure IV.3b), but a tissue dropout (circle) has now increased in size. With the blood-pool based method (Figure IV.3c) clutter noise inside the LV is suppressed, giving a clear view of the endocardial tissue and a papillary muscle (arrow) without reducing the thickness of the mitral valve (square). The extent of the mentioned dropout is also reduced (circle). In Figure IV.3d we

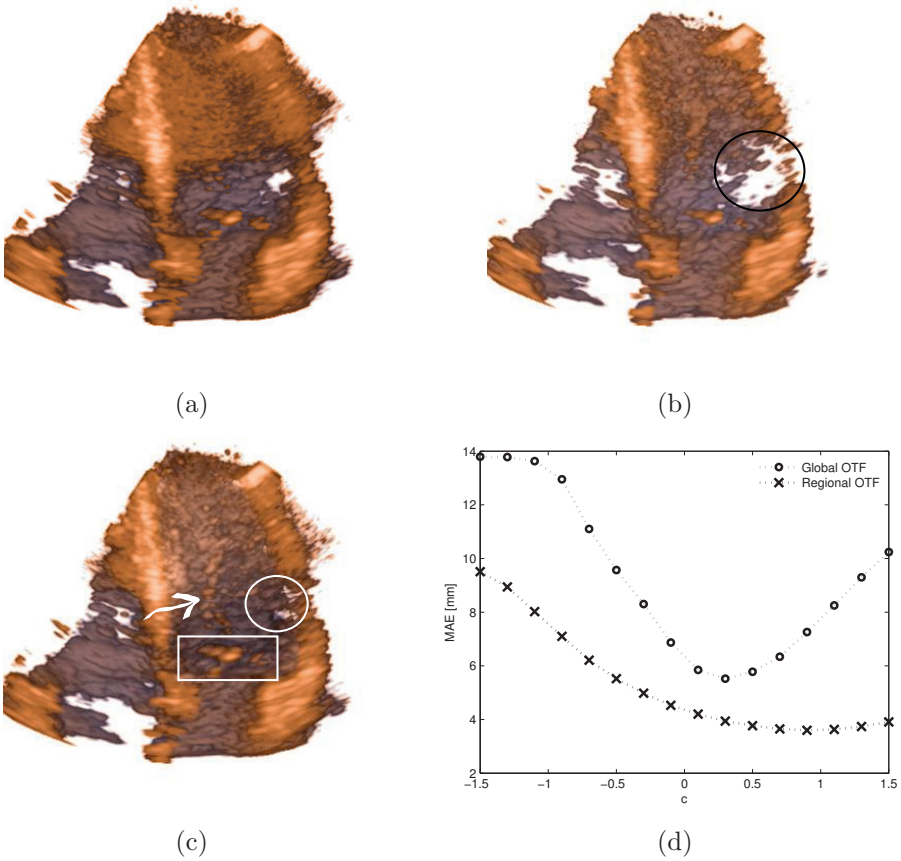


Figure IV.3: a) Global OTF with a low opacity threshold τ. A lot of noise is visible inside the LV (circle). b) Global OTF with τ corresponding to the lowest MAE in Figure d. The image contains less noise, but also a large tissue dropout (circle). c) Blood pool based regional adaptive OTF with c=0.5 which is close to the minimum in d. The papillary muscle is now more visible (arrow) and the tissue dropout (circle) is smaller without pruning the mitral valve (sqare). d) Parameter exploration in the view shown in a, b and c. MAE is plotted versus the c-value. For comparison, the global OTF has been converted into a function of c using the global mean and standard deviation.

present a parameter exploration of the c-value introduced with the regional adaptive OTF. For comparison the global opacity threshold is substituted with a function of c, global mean and standard deviation. The figure shows that for a given view the MAE measure has a local minimum for both the global and regional methods, and that the regional OTF has a smaller MAE than the global OTF for all c.

IV.4 Discussion

A regional adaptive OTF based on blood-pool statistics has been proposed. The purpose was to better handle the high variation of signal within tissue as well as variability of noise signal inside the blood pool. Both quantitative and qualitative results reported in Section IV.3 shows that this has been accomplished.

We believe that the significant reduction in MAE for the regional adaptive method in Table IV.1 is caused by a reduced amount of rendering outliers. By outliers we mean rendering rays that stop far away from the endocardial boundary depicted by the reference segmentation, thus having a high absolute error. We observe that the difference in MAE in millimetre is small, between 1.5 and 2.5 mm for the three views in Table IV.1, compared to normal average end-diastole myocardial thickness (around 10mm). However, regional improvements in order of centimetres can be observed in Figure IV.3c (circle). These large improvements are smoothed out by the MAE measure, but remains as a significant statistical improvement ($p < 0.001$) in MAE-difference between the regional and global adaptive method in all three views (Table IV.1).

Table IV.2 shows that in 46 pair of renderings the regional adaptive OTF based on blood pool statistics provided renderings with less tissue dropouts and spurious structures inside the blood pool. However, Table IV.2 also shows that the regional adaptive method was ranked worse in the MV view for two subjects, and in four MV-views the two methods were equal. On the other hand, the MV statistics in Table IV.1 are the most significant among the three views. One explanation to these diverging results is: When doing the qualitative assessment the physician echocardiographer watched the whole cardiac cycle, where the statistics in Table IV.1 are calculated from a single end-diastolic frame. In end-diastole the mitral valve is closed, which means that the MV is outside our tracking model and therefore not included in the blood pool statistics. When the MV is inside the tracking model, thus included in the estimates, visual artefacts can occur since the MV will be treated as being a part of the blood pool and might get rendered transparent. Another explanation is that all the recordings are from an apical imaging view. From this view the MV is normal to the ultrasound beam. The MV has because of this better image quality than e.g. the LV endocardium and thus less variation is observed in the MV-view statistics than for the long-axis views (Table IV.1). When the image quality is good it is hard to make any regional improvement compared with a global OTF. Opposite, the endocardial boundary has usually lower image quality than the MV from the apical imaging window. For the two 4-chamber views the statistics therefore has higher variation and for the inverted 4-chamber, that had large acoustic dropouts in many of

the recordings, the improvement is much larger.

The images in Figure IV.3 demonstrate that our regional adaptation of the OTF provides a better tradoff between artificial tissue dropouts and spurious structures compared with a global OTF. In Figure IV.3d this improvement is quantified as $\text{MAE}_{global}(c) > \text{MAE}_{regional}(c), \forall c \in [-1.5, 1.5]$. This illustrate our findings that the global OTF did not produce a smaller MAE than the regional OTF in any pair of renderings if both methods where tuned to the lowest MAE. The gentle slope of $\text{MAE}_{regional}(c)$ in Figure IV.3d is another important observation. This indicates that the proposed method is less sensitive to changes in c, and therefore less tuning is often required. This observation was also done by the physician echocardiographer. The rendering in Figure IV.3c has a c-value corresponding to the minimum of $\text{MAE}_{regional}(c)$ in Figure IV.3d. We have often found the c-value at the local minimum of the MAE measure to be approximately the same c-value as both we and the physician echocradiographer would have selected in a given view. A fully automatic method could therefore iterate towards the local minimum by steepest descent, removing the need for manual tuning of c. It should be kept in mind that the success of this approach highly depends on a correct segmentation of the LV. This approach will also not be real-time, since an accurate segmentation of the LV usually involves manual adjustments and processing times in the order of seconds. One option is to use the real-time acquired LV delineation, used to estimate blood pool statistics, to measure rendering accuracy as well. A different endocardial detection scheme should then be used to estimate blood-pool statistics, to avoid a too close coupling between the LV delineation used for estimation and validation.

As mentioned, no spatial regularization was applied to the results in Section IV.3. However, spatial smoothing will decrease the variation in τ_r across the image, and with increased filter kernels the results will become equivalent to applying an OTF with a global adaptive opacity threshold τ_g based on the whole blood pool. With increased spatial smoothing less visual artefacts have been observed at the cost of a higher MAE. As no flickering has been observed, temporal smoothing had no positive effect. E.g. the rapid movement of the mitral valve caused artificial dropouts in all views when temporal smoothing was applied.

IV.4.1 Limitations

The choice of using a deformable model to indicate the blood pool location imposes some limitations for the proposed method. As mentioned, some important cardiac structures are included in the depicted blood pool and might result in visual artefacts. However, how the LV endocardial boundary is derived is not relevant for the result in this paper. Local edge detectors combined with regularization may work equally well for limiting the regional estimation of means and standard deviations to the blood pool.

IV.5 Conclusion

In this paper we have introduced and investigated an adaptive DVR method for real-time visualization of cardiac 3D ultrasound. Compared to a global OTF we have shown, both with quantitative and qualitative results, that the regional adaptive method, based on blood pool statistics, is capable of reducing tissue dropouts and spurious structures inside the blood pool. The need for manual tuning is reduced by adapting the OTF to the data at hand, potentially leading to a fully automatic and data-dependent OTF.

References

[1] M. Levoy, "Display of Surfaces From Volume Data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29–37, 1988.

[2] E. Steen and B. Olstad, "Volume Rendering of 3D Medical Ultrasound Data Using Direct Feature Mapping," *IEEE Transactions on medical imaging*, vol. 13, no. 3, 1994.

[3] S. I. Rabben, *Technical Principles of Transthoracic Three-Dimensional Echocardiography*, 1st ed. Springer, 2011, ch. 2.

[4] G. Kiss, E. Steen, J. P. Åsen, and H. G. Torp, "GPU volume rendering in 3D echocardiography: real-time pre-processing and ray-casting," in *IEEE International Ultrasonics Symposium Proceedings*. IEEE, 2010.

[5] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. S. Avila, K. M. Raghu, R. Machiraju, and J. Lee, "The Transfer Function Bake-Off," *IEEE Computer Graphics and Applications*, vol. 21, no. 3, pp. 16–22, 2002.

[6] S. I. Rabben, S. Berg, and E. N. Steen, "Methods and apparatus for volume rendering," 2008.

[7] G. Kindlmann and J. W. Durkin, "Semi-automatic generation of transfer functions for direct volume rendering," in *Proceedings of the 1998 IEEE symposium on Volume visualization - VVS '98*. New York, New York, USA: ACM Press, Oct. 1998, pp. 79–86.

[8] P. Sereda, A. V. Bartroli, I. W. O. Serlie, and F. A. Gerritsen, "Visualization of Boundaries in Volumetric Data Sets Using LH Histograms," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 2, pp. 208–218, 2006.

[9] S. Wesarg, M. Kirschner, and M. Khan, "2D Histogram based volume visualization: combining intensity and size of anatomical structures," *International journal of computer assisted radiology and surgery*, vol. 5, no. 6, pp. 655–666, 2010.

[10] J. Woodring and H.-W. Shen, "Semi-Automatic Time-Series Transfer Functions via Temporal Clustering and Sequencing," *Computer Graphics Forum*, vol. 28, no. 3, 2009.

[11] M. Haidacher, D. Patel, S. Bruckner, A. Kanitsar, and M. E. Groller, "Volume visualization based on statistical transfer-function spaces," in *2010 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, Mar. 2010, pp. 17–24.

[12] C. D. Correa and K. L. Ma, "Visibility Histograms and Visibility-Driven Transfer Functions," *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[13] Y. Wang, J. Zhang, W. Chen, H. Zhang, and X. Chi, "Efficient opacity specification based on feature visibilities in direct volume rendering," *Computer Graphics Forum*, vol. 30, no. 7, pp. 2117–2126, Sep. 2011.

[14] I. Viola, A. Kanitsar, and M. E. Gröller, "Importance-Driven Feature Enhancement in Volume Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, pp. 408–418, 2005.

[15] S. Bruckner, S. Grimm, A. Kanitsar, and M. E. Gröller, "Illustrative Context-Preserving Exploration of Volume Data," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1559–1569, 2006.

[16] C. Rezk-Salama and A. Kolb, "Opacity Peeling for Direct Volume Rendering," *Computer Graphics Forum*, vol. 25, no. 3, pp. 597–606, Sep. 2006.

[17] M. M. Malik, T. Möller, and M. E. Gröller, "Feature peeling," in *Proceedings of Graphics Interface 2007 on - GI '07*. New York, New York, USA: ACM Press, May 2007, p. 273.

[18] S. Bruckner and M. E. Gröller, "Instant Volume Visualization using Maximum Intensity Difference Accumulation," *Computer Graphics Forum*, vol. 28, no. 3, pp. 775–782, Jun. 2009.

[19] S. Marchesin, J.-M. Dischler, and C. Mongenet, "Per-Pixel Opacity Modulation for Feature Enhancement in Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, pp. 560–570, 2010.

[20] D. Borland, J. P. Clarke, J. R. Fielding, I. I. Taylor, and M. Russell, "Volumetric depth peeling for medical image display," in *Proceedings of SPIE*, vol. 6060, 2006, pp. 35–45.

[21] S. Lindholm, P. Ljung, C. Lundström, A. Persson, and A. Ynnerman, "Spatial Conditioning of Transfer Functions Using Local Material Distributions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1301–1310, 2010.

[22] C. Lundström, P. Ljung, and A. Ynnerman, "Local Histograms for Design of Transfer Functions in Direct Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 1570–1579, 2006.

[23] Q. Zhang, R. Eagleson, G. M. Guiraudon, and T. M. Peters, "High-quality anatomical structure enhancement for cardiac image dynamic volume rendering," in *Proceedings of SPIE*, vol. 6918, no. 1. SPIE, Mar. 2008, pp. 691 834–691 834–10.

[24] D. Honigmann, J. Ruisz, and C. Haider, "Adaptive Design of a Global Opacity Transfer Function for Direct Volume Rendering of Ultrasound Data," *Visualization Conference, IEEE*, p. 64, 2003.

[25] F. Orderud, "A Framework for Real-Time Left Ventricular Tracking in 3D+T Echocardiography, Using Nonlinear Deformable Contours and Kalman Filter Based Tracking," *IEEE Computers in Cardiology*, 2006.

[26] W. Niblack, *An introduction to digital image processing*. Birkeroed, Denmark, Denmark: Strandberg Publishing Company, 1985.

[27] B. Petersch, M. Hadwiger, H. Hauser, and D. Hönigmann, "Real time computation and temporal coherence of opacity transfer functions for direct volume rendering of ultrasound data," *Computerized Medical Imaging and Graphics*, vol. 29, no. 1, 2005.

[28] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[29] E. Steen and B. Olstad, "Volume rendering in medical ultrasound imaging based on nonlinear filtering," in *IEEE Winter Workshop on Nonlinear Digital Signal Processing*. IEEE, 1993.

Paper V

Huygens on Speed: Interactive Simulation of Ultrasound Pressure Fields

Jon Petter Åsen[1] and Sverre Holm[1,2]
[1]Medical Imaging Lab (MI-Lab), Norwegian University of Science and Technology, Trondheim, Norway
[2]Department of Informatics, University of Oslo, Oslo, Norway

Proc. IEEE Ultrasonics Symposium 2012, pp. 1643-1646.

The introduction of graphics processing unit (GPU) computing has made it possible to speed up computationally demanding algorithms. One of these algorithms is the calculation of pressure fields from acoustic transducers. Here, the execution time often limits the number of elements and field points we can select in order to get results back in finite time.
In this paper we present a simple GPU-based simulator capable of simulating high resolution pressure fields at interactive frame rates. The simulator is based on the same principle as the Ultrasim toolbox, where responses from several point sources are accumulated in a set of observation points, hence solving the Rayleigh-Sommerfeld integral. The cumulative sum for each observation point is independent of all other observation points, making the problem perfect for GPU processing. For the simulator we provide both a Paint-like interface for interactive drawing of uniform linear arrays and free-hand shapes, and a Matlab interface for precise scripting of element positions and observation points.
The presented GPU simulator was compared both with a multi threaded C-version, Ultrasim, and Field II. Compared with Ultrasim we report a 400 times speedup when simulating a varying number of source points on a 150 K points observation grid. The test system consisted of a low-end GPU (Nvidia Quadro 600) and an Intel i7-870 2.93 GHz quad-core CPU.

V.1 Introduction

Software for simulation of pressure fields from ultrasound transducers is an important tool for gaining knowledge on how a transducer performs. This brings both great insight to researchers and helps lowering production costs. A downside is that calculating large fields is often found to be time consuming, and the number of field points and the level of transducer approximation have to be adjusted low enough to have the result back in finite time. The introduction of graphics processing unit (GPU) computing has made it possible to speed up computationally demanding algorithms. In this paper we present a simple GPU-based simulator capable of simulating high resolution pressure fields at interactive frame rates.

Today there exist a vast collection of ultrasound simulators. There is Field II by Jensen[1], considered the gold standard of ultrasound simulation. However, there are also simulators that provides different functionality and perform different trade-offs than Field II. One example is the Ultrasim toolbox[2]. As for Field II, this is also a Matlab plugin. However, where Field II is mostly used to simulate ultrasound images and beams, Ultrasim can only simulate snapshots of transducer transmit fields. Another difference, is the way these toolboxes are organized. Field II is a collection of functions that makes it easy to run large batched simulations where different acquisition setups are to be investigated. Ultrasim, on the other hand, is an interactive tool with a graphical user interface controlling all simulation parameters. Ultrasim is therefore more accessible if one only need to simulate transducer transmit pressure. Field II and Ultrasim are based on two different concepts, the spatial impulse response and Huygens' principle respectively. In addition there are also simulators based on the

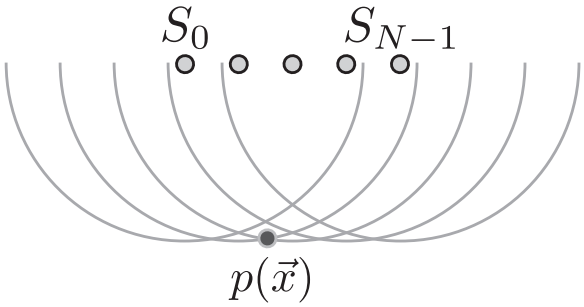


Figure V.1: Huygens' principle showing the superposition of spherical waves from many point sources.

angular spectrum method [3] capable of simulating 3D harmonic fields. And lately, simulators, capable of simulating 3D ultrasound images in orders of seconds have also been introduced [4].

Like Ultrasim, the presented simulator, Huygens on Speed (HOS), is based on Huygens' principle [5] as depicted in Fig. V.1. Hence, solving the Rayleight-Sommerfeld integral[2]:

$$\phi = \frac{1}{2\pi} \int_S \frac{u_n(r_0, t - r/c)}{r} dS. \tag{V.1}$$

where u_n, the normal velocity, is integrated over the transducer surface S weighted with the reciprocal of distance from the field point to the transducer.

We easily see that each calculation at a given field point is independent of the result at all other field points. The problem is therefore perfect for GPU computing, since one thread on the GPU can process one field point independent of all other threads. The only inputs needed are the field point and source point properties. Examples of other simulators implemented using GPU computing includes [6] and [7]. The first simulates 3D pressure fields at high speeds using a fast near field method, and the latter is capable of simulating ultrasound images in real time using convolution and an assumption of a shift-invariant imaging system.

The simulator presented in this paper is designed to be an interactive tool for drawing and exploring different array geometries. It is our belief that the simulator can be used both in research and academic teaching, as a neat way of demonstrating array beamforming principles.

V.2 Method

The presented simulator works by discretizing the surface S found in the Rayleigh-Sommerfeld integral into a set of point sources, $\{S_i\}_{i=0}^{N-1}$. As shown in Fig V.1, the total field pressure, p, at position $\mathbf{x}$ can then be found as the coherent sum of N

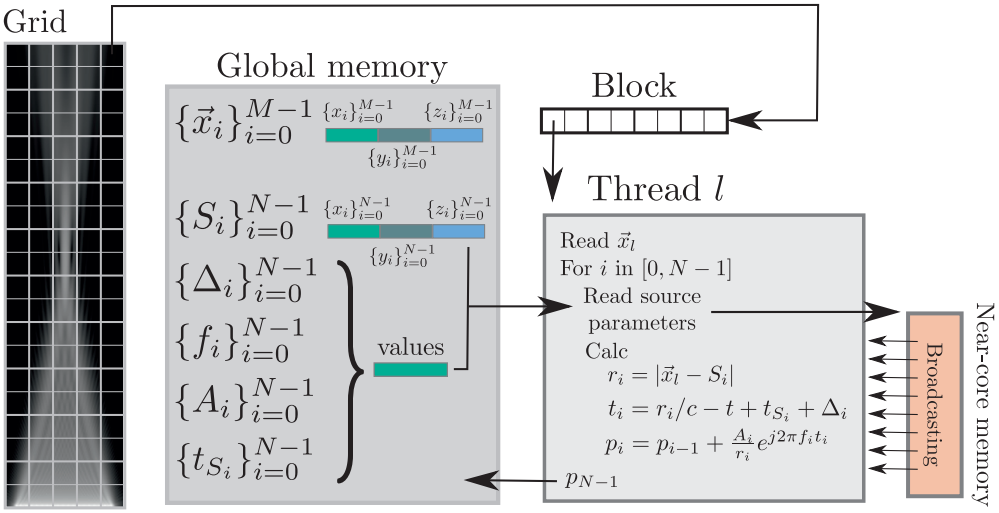


Figure V.2: Implementation of Huygens' principle on the GPU. One thread is launched per observation point, here presented as an azimuth plane. Each thread then reads the corresponding observation point, and collaborates on loading source point properties to near-core memory. From near-core memory, source properties are broadcast to registers when the kernel iterates over the set of sources. Note the coalesced memory layout of $\mathbf{x}$ and S in global memory. The figure reflects the notation used in Section V.2.

monochromatic spherical waves:

$$p(\mathbf{x}) = \sum_{i=0}^{N-1} \frac{A_i}{r_i} e^{2\pi j f_i t_i}. \tag{V.2}$$

Here, A_i is the source amplitude or apodization, f_i is the source frequency, r_i is the distance from $\mathbf{x}$ to S_i, and t_i is the wave travelling time given by $t_i = r_i/c - t + t_{S_i} + \Delta_i$, where c is the speed of sound, t is the snap shot time stamp, t_{S_i} is the source time stamp, and Δ_i is the steering and focusing delay. Hence, a simple simulator based on (V.2) works by rendering the contribution from a set of point sources with their corresponding properties in a set of observation points. How the source and observation points are positioned and what they represent is left for the user to decide.

V.3 Design

From (V.2) we see that field energy at a given observation point, $\mathbf{x}$, can be found independently from all other locations in space, and just as important, the result for each observation point is written to different bits of memory. Such a problem is perfect considering the GPU's ability of processing hundreds of pixels/threads in parallel. The

implementation of (V.2) used in the presented simulator is depicted in Fig V.2. One thread is launched per observation point. The reason for using this level of granularity is that the number of observation points is usually large compare to the number of source points. We also end up with enough instructions per thread to hide memory and instruction latency. On the other hand, if we have more source points than observation points, we should parallelize across source points instead. However, the result from each source point has to be written to equal positions in memory, making barriers that would significantly lower the method's throughput. Continuing the discussion of Fig. V.2, the threads inside a given block work together to load all source points and their corresponding properties to near-core memory. Since all threads inside a block work on the same source simultaneously, we make use of broadcasting to feed one source to all threads in a single read per property. Note also how observation and source points are organized in global memory for coalesced reads, and by that maximizing global memory throughput.

As front-end to the presented simulator we provide both a Matlab scripting environment and a Paint-like user interface for interactive drawing of arrays and freehand shapes. The Matlab interface is minimal but highly flexible. The functional interface takes as arguments the values listed in global memory in Fig. V.2, and returns the calculated pressure for each observation point. Constructing and discretizing transducer arrays into points and calculating delays etc. is left for the user. However, an example script of how this could be done is provided.

After increased speed, the main contributions in this work is how the user interface has been composed to make an interactive and interesting experience out of exploring acoustical arrays and their properties. The user interface works like a paint program, where point sources are added based on mouse interaction. There are several supported modes of interaction, and extensions to more advanced modes are obviously possible. However, when selecting modes of drawing, precision in source point positions have not been our focus. For that, one have the Matlab interface. Supported modes include click-and-drag of $n\lambda$-spaced arrays and freehand drawing. For both these modes there exist both pulsed and continuous wave modes.

In click-and-drag mode, equidistant point sources are created along a straight line from where the left mouse button was clicked to where it is released using the specified λ-spacing. The result is an array with one point source per element. Hence, there is no element directivity. In freehand drawing, a new point source is added for each mouse-moved event that occurs when the left mouse button is down, just like freehand drawing in Paint.

After point sources have been added, they can all be focused in a common field point by right-click-and-drag. This can be used e.g. to illustrate how ultrasound arrays are able to sweep the beam in space, and how the beam can be focused at different depths. In pulsed-wave mode this can be used to illustrate how a pulse evolves when moving through the focus. Other concepts that can be interactively visualized both in pulsed and continuous wave mode include among other; frequency dependent resolution and depth of field, the effect of apodization, grating lobes and side lobes, near- and far-field transitions, and multiple focus zones.

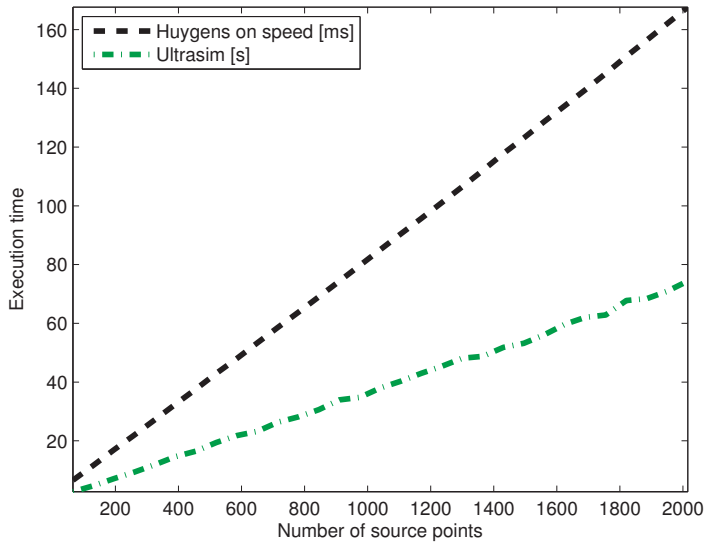


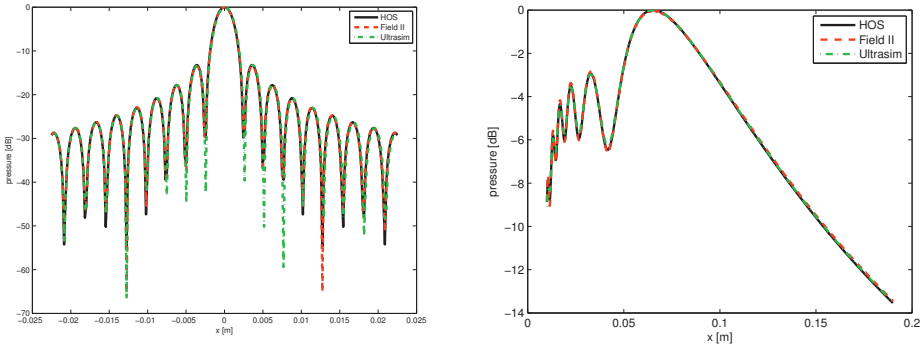
Figure V.3: Benchmark of HOS and Ultrasim. Test setup consisted of 150 K observation points.Note that execution times for the presented simulator are in order of milliseconds whereas execution times for Ultrasim are reported in seconds. The target platform was a low-end Nvidia GPU (Quadro 600, 96 cores GPU) and a high-end Intel quad-core CPU (i7-870, 2.93 GHz).

V.4 Results and Discussion

Implementing Huygens' principle on the GPU results in a significant speedup, that again allows for interactive simulations of dense pressure fields from acoustical arrays. We show this in Fig. V.3 by presenting a benchmark made between HOS and Ultrasim. Note that execution times are reported in seconds for Ultrasim and milliseconds for HOS. Thus the average speedup is 400x in favour of HOS across the selected range of source points.

Fig. V.4a shows the in-focus (at 80mm) lateral pressure from a 2.5MHz, 64 element (32λ wide and 10λ high, 5 azimuth $\times$ 20 elevation points/subelements per aperture) uniform linear array calculated with HOS, Field II and Ultrasim. The beam profiles are strikingly similar. In Fig. V.4b we also see that all three simulators yield the same result along an axial cut through focus. The continuous pressure field calculated with Field II was found by taking the fourier transform of the total transmit beam energy at the center frequency.

Fig. V.5 shows the azimuth pressure field calculated using HOS (a) and Ultrasim (b). The resolution is 5.6 observation points per mm for both, hence 250K observation points in total. HOS and Ultrasim used 2 seconds and 3 minutes 40 seconds



(a) Lateral pressure

(b) Axial pressure

Figure V.4: Profiles of pressure generated by HOS, Ultrasim and Field II. The simulated array has 64 elements, a center frequency of 2.5 MHz, $\lambda/2$ pitch, and $\lambda/2$ wide and 10λ high elements. Uniform apodization is used and the array is focused at 8 cm depth. 5 source points (mathematical elements in Field II) were used in azimuth and 20 in elevation.

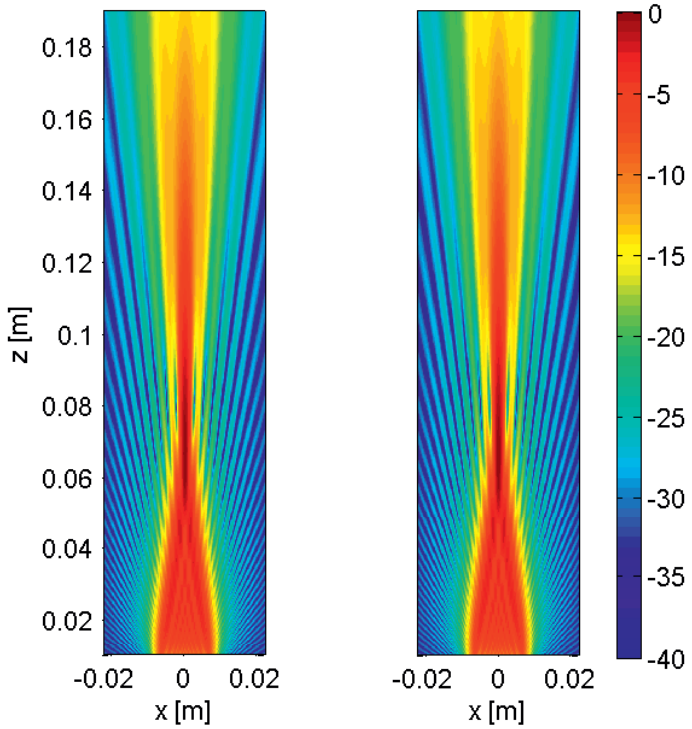
respectively in order to generate the image, hence a 110x speedup. The hardware used in this comparison was an Nvidia GT520, 48 cores GPU, and a i7-870 Intel quad-core CPU. A multi threaded C-version of the presented simulator used 1 minute and 45 seconds to generate the same image. Field II used 2 minutes 51 seconds. Note that Field II has to calculate the spatial impulse response for all observation points, and is therefore not directly comparable to Ultrasim's snap-shot-mode. The spatial impulse response can be calculated using HOS by iterating in time. Using the available low-end GPU and high-end CPU, the presented simulator and Field II are equally fast on this operation (without using streams on the GPU). However, for massive parallel calculation of the spatial impulse response one might want to apply a different parallelization scheme and level of granularity than what is described in this paper.

Fig. V.6 shows how arrays of any shape can be interactively drawn using the provided user interface. A video of how the interactive user interface works can be found at [8].

The simulator can be downloaded from the same site [8]. The source code has been made available at github [9]. Contributions are welcome.

Acknowledgment

The authors would like to thank Tore Bjåstad, Torbjørn Hergum and Hans Torp for the initial idea of creating the HOS simulator.



(a) HOS (b) Ultrasim

Figure V.5: Azimuth pressure field in dB generated by the presented simulator and Ultrasim for the same array simulated in Figure V.4. The resolution is 5.6 observation points per mm for both (250K observation points in total).

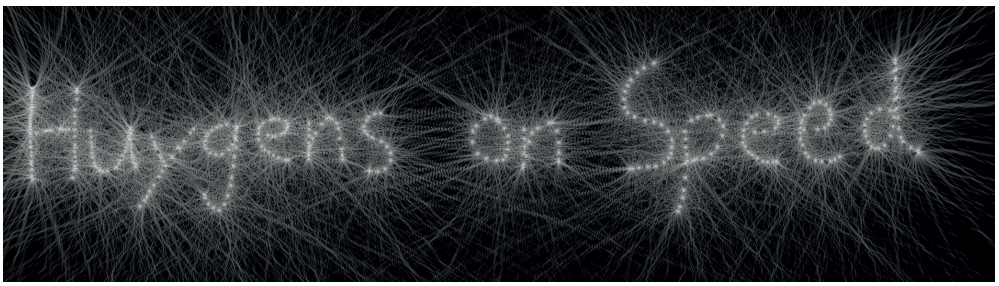


Figure V.6: Free hand drawing of source points using the interactive user interface.

References

[1] J. A. Jensen and N. B. Svendsen, "Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers." *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 39, no. 2, pp. 262–267, Jan. 1992.

[2] S. Holm, "Ultrasim - a toolbox for ultrasound field simulation," in *Nordic Matlab conference*, 2001.

[3] F. Prieur, T. F. Johansen, S. Holm, and H. Torp, "Fast simulation of second harmonic ultrasound field using a quasi-linear method." *The Journal of the Acoustical Society of America*, vol. 131, no. 6, pp. 4365–75, Jun. 2012.

[4] T. Hergum, S. Langeland, E. W. Remme, and H. Torp, "Fast ultrasound imaging simulation in K-space." *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 56, no. 6, pp. 1159–67, Jun. 2009.

[5] "Treatise on Light by Christiaan Huygens." [Online]. Available: http://www.gutenberg.org/ebooks/14725

[6] M. Hlawitschka, R. J. McGough, K. W. Ferrara, and D. E. Kruse, "Fast ultrasound beam prediction for linear and regular two-dimensional arrays," in *IEEE International Ultrasonics Symposium*. IEEE, Oct. 2010, pp. 2199–2202.

[7] S. U. Gjerald, R. Brekken, T. Hergum, and J. D'Hooge, "Real-Time Ultrasound Simulation Using the GPU," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 59, no. 5, pp. 885–892, 2012.

[8] "Huygens on Speed." [Online]. Available: http://folk.uio.no/jpaasen/huygens/

[9] "Huygens on Speed git repository." [Online]. Available: https://github.com/jpaasen/hos.git