Design Space Exploration of Adaptive Beamforming Acceleration for Bedside and Portable Medical Ultrasound Imaging

Junying Chen
Department of Electrical and Electronic Engineering
The University of Hong Kong, Hong Kong
jychen@eee.hku.hk

Billy Y. S. Yiu
Medical Engineering Program
The University of Hong Kong, Hong Kong
bysyiu@hku.hk

Brandon K. Hamilton
Department of Electical Engineering
The University of Cape Town, South Africa
brandon.hamilton@uct.ac.za

Alfred C. H. Yu
Medical Engineering Program
The University of Hong Kong, Hong Kong
alfred.yu@hku.hk

Hayden K.-H. So
Department of Electrical and Electronic Engineering
The University of Hong Kong, Hong Kong
hso@eee.hku.hk

ABSTRACT

The use of adaptive beamforming is a viable solution to provide high-resolution real-time medical ultrasound imaging. However, the increase in image resolution comes at an expense of a significant increase in compute requirement over conventional algorithms. In a bedside diagnosis setting where plug-in power is available, GPUs are promising accelerators to address the processing demand. However, in the case of point-of-care diagnostics where portable ultrasound imaging devices must be used, alternative power-efficient computer systems must be employed, possibly at the expense of lower image resolution in order to maintain real-time performance. This paper presents an initial design space exploration on viable compute architectures that might address the drastically different requirements between bedside and portable medical ultrasound imaging systems using adaptive beamforming. The design and implementation of a GPU accelerator that provides over 45x performance improvement over the equivalent C implementation on a single CPU is presented. Furthermore, and implementation of the beamforming algorithm on a high-performance mobile platform based on an ARM Cortex A8 mobile processor in combination with the built-in NEON accelerator is also presented. The mobile platform delivers over 270x reduction in power consumption when compared to the GPU platform at an expense of much reduced performance. The tradeoffs between power, performance and image quality among the target platforms are studied and future research directions in power-efficient architectures for high-performance medical ultrasound systems are presented.

1. INTRODUCTION

In clinical practice, ultrasound has established itself as the medical imaging modality of choice for real-time bedside scanning [7]. With advances in portable system design, this modality has also expanded its application domain into point-of-care diagnostics that are conducted in remote out-hospital settings [3]. Despite its real-time imaging capability and broad clinical applicability, ultrasound has drawn reservations from some clinicians over its second-class image quality as compared to those seen in other modalities like magnetic resonance imaging. To address this issue, the ultrasound community have been striving to develop novel beamforming routines that can improve the imaging contrast and resolution [8]. One possible solution is to use apodization during beamforming to suppress contrast-reducing sidelobes inherently introduced from the delay-and-sum procedure used for receive focusing. Recent investigations have indeed shown that adaptive computation of apodization weights based on local signal statistics can further improve sidelobe suppression without affecting the focusing quality [5, 6]. Nevertheless, the adaptive beamforming framework is not yet transferable to everyday clinical use because the computational power needed for adaptive apodization calculations is seemingly beyond that available on existing scanners where system resources are often deployed on a frugal basis.

In a bedside diagnosis setting where plug-in power is available, the use of graphics processing units (GPUs) is a promising cost-effective accelerating technology that provides the much-needed processing power. In this work, we present our initial design and implementation of an adaptive beamforming algorithm using an NVIDIA GTX 480 GPU as an accelerator. When compared to the unoptimized C implementation on a single CPU system, a speedup of 192x has been demonstrated, providing real-time performance to our ultrasound imaging system if minor image degradation is allowed. If compared to the optimized single-CPU C implementation, an acceleration of 45x is achieved. How-

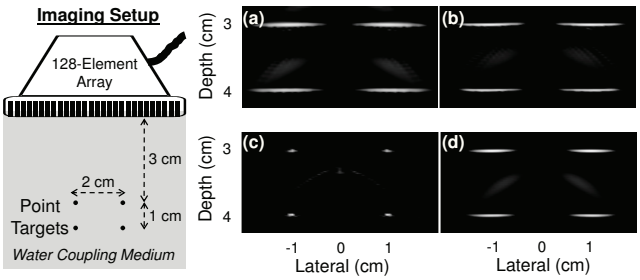

Figure 1: Images obtained with four beamforming methods (32-channel receive apertures): (a) sum without focus delay; (b) delay and sum (DAS); (c) DAS with adaptive apodization (minimum variance, L=16); (d) DAS with fixed Hanning apodization. Data was acquired by laterally sweeping an unfocused beam (32-channel transmit aperture, 5 MHz ultrasound).

ever, the considerable power consumption of the combined CPU and GPU system makes this system less than ideal for use in portable settings where battery life must also be treated as a primary design constraint.

In an attempt to explore the design space of adopting adaptive beamforming algorithms in portable systems, our algorithm was retargeted to a high-performance mobile platform. An ARM Cortex A8 processor [2], with the help of its NEON SIMD coprocessor is used as a case study. When compared to the GPU solution, a 270x improvement in power consumption has been observed. However, the absolute performance of the mobile platform remains far from capable of handling real-time image processing. Further development of a more power-efficient compute platform is still in progress.

As such, the goal of this work is not to present a complete solution to real-time ultrasound imaging using adaptive beamforming. Rather, we take a system designer's perspective and aim to:

- Explore the performance benefit of GPU accelerators with power consumption unconsidered;
- Explore the performance of high-performance, low-power mobile platforms;
- Identify future research directions in the power-performance tradeoff space.

In the next section, background information about adaptive beamforming for ultrasound imaging will first be provided. The GPU and mobile CPU implementations of the algorithm will then be presented in Section 3 and Section 4 respectively. In Section 5, we study the power and performance tradeoff of the two systems and will make suggestions for future research and conclude in Section 6.

2. ADAPTIVE BEAMFORMING IN MEDICAL ULTRASOUND

2.1 Background Concepts

Ultrasound imaging is based upon a pulse-echo sweep sensing mechanism that acquires images by sweeping an ultrasound beam laterally across an imaging view [7]. In modern ultrasound systems where array transducers are employed, the beam sweep is facilitated by sequentially activating and deactivating different array channels; on receive, pulse-echo data are acquired over the same set of array channels to form images. To achieve focusing at each image pixel, a procedure known as delay-and-sum (DAS) beamforming is typically used [8]. It computes a pixel value as an absolute sum of each receive channel's pulse-echo data sample whose index position corresponds to the time-of-flight between the pixel and that channel. This procedure is performed for every pixel in the scanline, and it is repeated over all scanlines to form an image.

Figs. 1a and 1b shows an example of an ultrasound image obtained with and without DAS for a field-of-view that encompasses a series of point-target pairs placed at two different depths (pixel magnitude shown in dB scale). As can be seen, DAS beamforming can improve the resolvability of the point-target pairs. Nevertheless, artifacts in the form of spurious gray streaks can still be found: a problem that is brought about by unwanted sidelobes inherent in the DAS process [8].

2.2 Adaptive Beamforming: The Principles

The overall goal of adaptive beamforming is to improve the ultrasound image quality through suppressing unwanted sidelobes. It works by adaptively applying apodization to each channel such that the DAS procedure becomes a weighted summation process. It has been recently shown that if the output pixel variance can be minimized with respect to the local signal statistics, then the sidelobes can be suppressed without affecting the resolving power [5, 6]. Fig. 1c illustrates how adaptive apodization can lead to sharper visualization of the point-target pairs. It is worth noting that fixed apodization weights like Guassian and Hanning windows may be used as well, but their impact is less effective given that the output pixel variance is not minimized (see Fig. 1d).

In terms of its formulation, the adaptive beamforming approach first segments an M-channel receive aperture into a series of lag-one overlapping sub-apertures with L channels [5, 6]. Typically, the sub-aperture size is chosen to be between $M/4$ and $M/2$ to maintain a suitable balance with the total number of sub-apertures. The pixel value for position P_o is then defined as the absolute average of all sub-apertures' individual weighted sum for the corresponding delayed channel-data samples. Such a value can be mathematically expressed as follows:

$$z(P_o) = \frac{1}{M-L+1} \sum_{k=0}^{M-L} \mathbf{w}^H(P_o)\mathbf{x}_k(P_o), \qquad (1)$$

where $\mathbf{w}(P_o)$ is the $L \times 1$ apodization vector for a sub-aperture, and $\mathbf{x}_k(P_o)$ is the $L \times 1$ ensemble of focus-delayed channel-data samples in the k^{th} sub-aperture (between channels k and $k+L-1$).

From optimization principles, it can be shown that Eq. 1 would have minimum output variance if $\mathbf{w}(P_o)$ is

defined as follows [5]:

$$\mathbf{w}(P_o) = \frac{\mathbf{R}^{-1}(P_o)\mathbf{a}}{\mathbf{a}^H\mathbf{R}^{-1}(P_o)\mathbf{a}}, \quad (2)$$

where $\mathbf{R}^{-1}(P_o)$ is the covariance matrix inverse for the focus-delayed channel-data samples and $\mathbf{a}$ is simply an $L \times 1$ vector of ones. In matrix algebra, one common way of estimating the covariance matrix $\mathbf{R}(P_o)$ is to compute the mean outer product between each data ensemble and its conjugate transpose [5]. This can be mathematically expressed as follows when given $M - L + 1$ sub-apertures:

$$\mathbf{R}(P_o) = \frac{1}{M-L+1} \sum_{k=0}^{M-L} \mathbf{x}_k(P_o)\mathbf{x}_k^H(P_o). \quad (3)$$

In the adaptive beamforming framework, one of the major computation bottlenecks is the need to estimate the covariance matrix for each pixel. If this process can be parallelized in real-time, realizing adaptive beamforming algorithms in ultrasound imaging can be made to be more feasible.

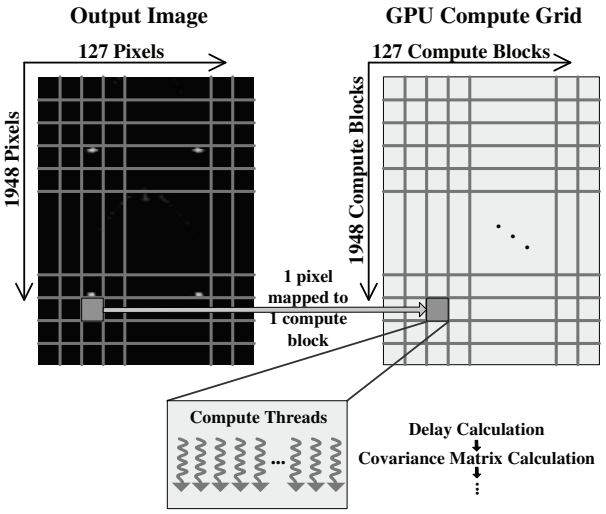

Figure 2: Mapping of output image pixels to GPU compute blocks.

3. GPU IMPLEMENTATION

This section presents the design and implementation of our adaptive beamforming algorithm on a GPU-based accelerator platform. An off-the-shelf high-performance GPU was used for its commodity and relatively low cost. The goal of this design is to deliver the highest performance with little consideration on power consumption. It is assumed that such platform will be used for bedside diagnosis where plug-in power supply is available.

3.1 Implementation Platform

Our adaptive beamforming algorithm was implemented on an NVIDIA GeForce GTX 480 graphics display card hosted in a desktop computer with a 2.4 GHz Intel Core 2 Quad Q6600 CPU and 8 GB DDR2 RAM running the Ubuntu 9.10 environment. The GTX 480 was chosen as it was the first commercially available GPU that realized the Fermi architecture [4]. 15 streaming multiprocessors (SMs) were present on the GPU. Each SM in turn included 32 streaming processor cores, making a maximum of 480 executing cores available during runtime. NVIDIA CUDA toolkit version 3.1 was used for compilation.

3.2 Algorithm Parallelization

To fully exploit the compute power of the GPU, a two-level parallelization process was applied to the adaptive beamforming algorithm to match the CUDA programming model as shown in Figure 2.

On the top-level, a coarse-grained parallelization was employed in which the algorithm was split into blocks of identical computation that operated on different sections of raw ultrasound scanlines. Each one of the 127×1948 output pixels was mapped to exactly one thread block, collectively forming a CUDA compute grid. Such mapping was possible because the beamforming process of each image pixel was independent of other pixels in the image. The amount of compute blocks that may be activated in our implementation was therefore equal to the number of pixels in our target output image.

Subsequently, a low-level, fine-grained parallelization was employed in which the computation within the thread block was further assigned to a set of parallel compute threads. For this purpose, independent computations within each beamforming sub-function were assigned to different compute threads. Consider the calculation of covariance matrix as an example. Since the calculation of a covariance matrix element was independent of that of other elements, each of them was handled by one independent compute thread in our implementation. Other beamforming sub-functions were parallelized similarly into computing threads within the thread block. The number of concurrent parallel threads per compute block is termed the thread block size in NVIDIA's terminology. The thread block size, the number of active blocks in the grid that may be scheduled to run, together with their usages of shared registers and memories all play an non-trivial role in determining the overall performance. Our experience showed that although the Fermi architecture allowed a maximum thread block size of 1024, our current implementation activated only 128 compute threads within a compute block as it resulted in better performance than with larger thread block sizes.

3.3 Memory Management

Figure 3 highlights the various components in the memory hierarchy of the Fermi architecture, including register files, shared memories, L1 and L2 caches, as well as global memory. Since the access speed to different types of memory varies considerably, the choice of memory type as temporary storage has a significant effect on the overall performance.

In our current implementation, the slow global memory was used exclusively as buffers to hold the top-level input and output data of the entire beamforming process. Within each streaming multiprocessor, the 64 KB

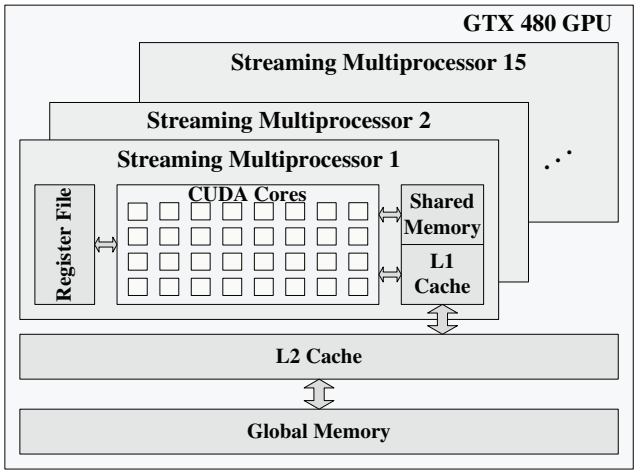

Figure 3: Memory hierarchy of Fermi architecture.

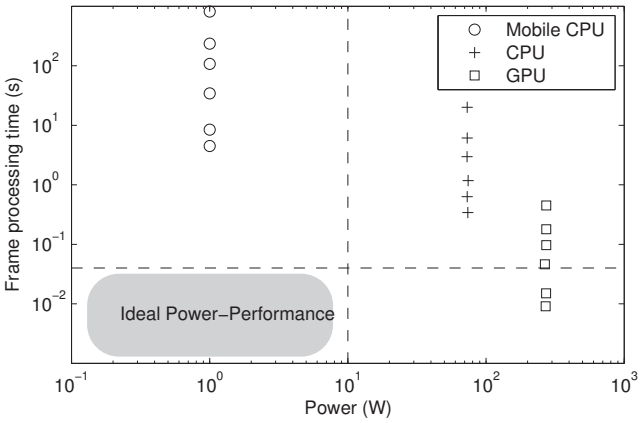

Figure 4: Performance and power consumption of 3 target architectures. The area towards the lower left corner indicates the ideal power-efficiency required for portable ultrasound systems.

of configurable on-chip memory was configured as 48 KB shared memory with 16 KB L1 cache. Compared to the other configuration that provided 16 KB of shared memory, the configuration with a larger shared memory was chosen as we will be using the large shared memory as buffers between sub-functions of the beamforming algorithm.

When a thread block was scheduled to execute on an SM, the individual pixel beamforming process of the block began by reading in related input data from the global memory. During the beamforming process, buffers allocated in the shared memory in the SM were used to store the results of beamforming sub-functions. To ensure the highest availability of shared memory, registers were used exclusively for intermediate results within these sub-functions. The output of the pixel beamforming was written back to the global memory at the end of the beamforming process when the register file and shared memory occupied by the thread block were also released.

4. MOBILE PROCESSOR IMPLEMENTATION

To explore the performance tradeoff of very low power platforms for use in portable ultrasound imaging systems, the adaptive beamforming algorithm presented in Section 2 was implemented on a high-performance mobile CPU system.

4.1 Implementation Platform

The ARM Cortex A8 processor present on the custom-made RHINO platform [1] was used as our implementation target. The RHINO platform was chosen as its processor system was connected to an FPGA accelerator that may be utilized in our future work. In this work, only the ARM system was utilized.

The ARM processor running at 600 MHz with 32 KB of L1 and 256 KB of L2 caches is a high-performance mobile CPU targeted for applications such as smartphones and tablet computers. On the RHINO platform, with 256 MB of DDR2-200 RAM attached, a standard Linux 2.6 kernel system was in place.

The processor includes an accelerator coprocessor, called NEON, which supports 128-bit SIMD instructions for both integer and floating point operations. The NEON SIMD coprocessor supports up to 16 operations simultaneously. It shares the same floating-point registers used in ARM vector floating point (VFP) coprocessor extensions, which provide low-cost single-precision and double-precision floating-point computation. The NEON architecture includes three execution pipelines for integer and fixed point SIMD instructions, and two execution pipelines for advanced SIMD floating point instructions. There is a direct interface between the integrated L2 cache and the NEON coprocessor unit for data streaming.

4.2 Algorithm Implementation

The implementation of our beamforming algorithm on the ARM platform was a straightforward compilation of the same C source code that served as a basis for our GPU implementation. To improve performance without incurring major increase in power consumption, only the NEON coprocessor was utilized.

The code was cross-compiled using the CodeSourcery G++ Lite Edition compiler for ARM. The compiler supports optimizations and autovectorizations for the ARMv7 architecture and 128-bit NEON quadword vector registers, as well as VFPv3 scalar hardware floating-point code. Using this, the integer and floating-point code within loops were transformed automatically to make use of the NEON SIMD instructions for processing multiple data elements simultaneously.

5. RESULTS & DISCUSSIONS

To evaluate the performance and power consumption of our target systems, our minimum-variance (MV) adaptive beamforming algorithm was implemented with 32- and 64-element receive apertures ($M = 32$ and $M = 64$), each utilizing subarray lengths of half and quarter of the number of receive elements, i.e., $L = M/2$

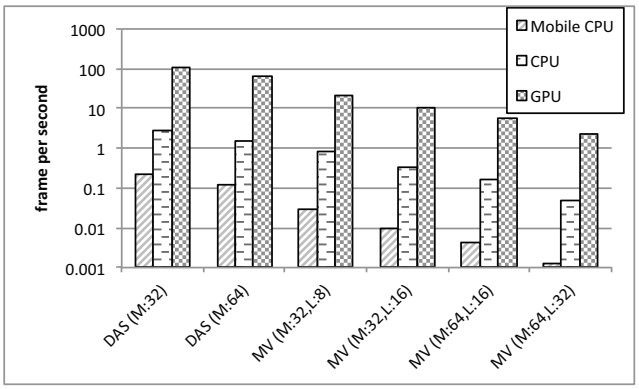

Figure 5: Performance of mobile CPU, CPU and GPU executing adaptive beamforming with $M = \{32, 64\}, L = \{M/2, M/4\}$ and with delay-and-sum of channels.

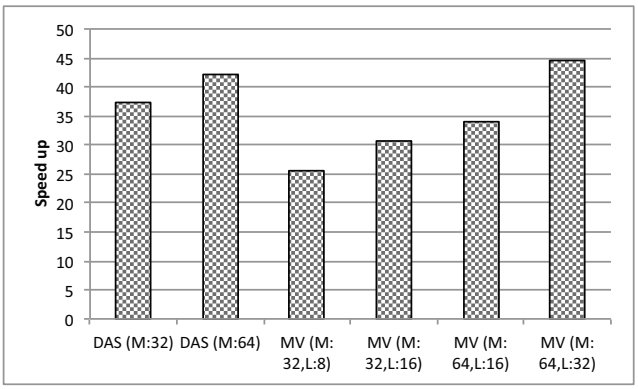

Figure 6: Speed up of the GPU implementation when compared to the CPU implementation.

and $L = M/4$. Furthermore, a non-adaptive beamforming algorithm based on conventional delay-and-sum (DAS) was implemented to examine the performance impact of the adaptive algorithm.

Besides the GPU and mobile CPU implementations, the same algorithm was also implemented on the host CPU of the GPU platform. The implementation was a straightforward compilation of the source code using `gcc` version 4.4. Level 3 (`-O3`) optimization with autovectorization was enabled to exploit the performance benefit of the Streaming SIMD extension (SSE) on the Intel processor.

To measure power consumption of the GPU and CPU implementations, a wirelessly connected power monitor was used to measure the power drawn from the main AC socket. The measured power consumption therefore included power consumed by the entire supporting platform, including the power loss at the AC transformer. The GPU was turned off when the power of the CPU implementation was measured. On the RHINO platform, power consumption of the mobile processor system was measured directly using on-board power sensors.

5.1 Results Overview

Figure 4 gives an overall summary of the main implementation results of this paper, which shows three clusters of results from the three target platforms. Each data point in the graph denotes the time required to complete adaptive beamforming for one image under a specific set of system parameters (M, L). The shaded area bounded by the dotted lines in the lower-left corner indicates the power consumption and performance of the ideal platform for our ultrasound application. For smooth, real-time processing, video frame rates of 25 to 30 frames per second are required. Furthermore, for battery operated portable systems, moderate power and energy consumption is needed to ensure reliable operations.

5.2 Performance

Figure 5 shows the performance of our mobile CPU, CPU and GPU implementations. It is clear from Figure 5 that our GPU implementation consistently outperform the other two CPU implementations. In fact, the speed advantage of the GPU platform becomes more prominent as the computational requirement increases with M and L as shown in Figure 6. In the case of $M = 64, L = 32$, where the computational requirement is highest, the GPU implementation achieve a 45x speedup over the CPU implementation.

However, despite the speedup over our simple CPU implementation, the GPU implementation falls short of delivering real-time performance in all cases except when image quality is sacrificed with $M = 32, L = 8$.

The performance of the mobile platform, with an ARM Cortex A8 and its associated NEON SIMD accelerator was the lowest as expected. The platform by itself is unable to provide satisfying processing power to implement our real-time adaptive beamforming in software. In the case of the most computationally demanding case of $M = 64, L = 32$, a 40x slowdown was observed when compared to the desktop CPU processor.

5.3 Power Efficiency

One of the motivations to consider using a mobile CPU platform for performing adaptive beamforming was its low power consumption that was ideal for portable systems. Results show that the active power consumed by the mobile platform, the CPU platform and the GPU platform consume around 1W, 73W, and 240W on average respectively. Here, the ARM-based platform has a clear advantage in terms of power consumption.

However, the low power consumption also results in the considerably lower performance of the mobile platform. As a result, to truly reflect the efficiency of a platform, the performance-to-power ratios of the ARM platform and the GPU platform are plotted in Figure 7, normalized to that of the CPU platform.

Figure 7 shows that in spite of the 2 orders of magnitude higher power consumption of the GPU platform, it actually out-performed the low-power mobile platform in terms of power-performance ratio. To understand the results in Figure 7, note that the power-performance ratio is indeed equal to the energy required to process each frame of image:

$$\frac{\text{power}}{\text{performance}} = \frac{\text{energy per second}}{\text{frame per second}} = \frac{\text{energy}}{\text{frame}}. \quad (4)$$

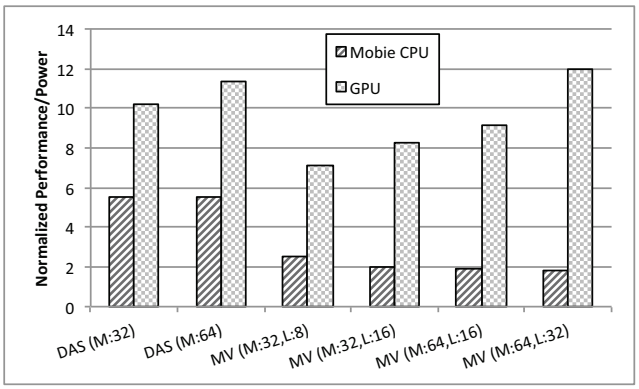

Figure 7: Performance-power ratios of mobile CPU and GPU normalized to the desktop CPU platform.

Although the GPU platform runs at 270x the power consumption of the mobile platform, it has over 3 orders of magnitude higher processing performance. As a result, the GPU spends less time on processing each frame, resulting in lower energy consumption. Consequently, for instance in the case of $M = 64$ and $L = 32$, the GPU is performing 6.6 times better than the mobile CPU platform in terms of energy consumption per frame.

5.4 Power-Performance Tradeoffs

Refer back to Figure 4, none of the three platforms under consideration is capable of fulfilling both the power and performance requirements at the same time. Although our current GPU implementation can deliver real-time realization of adaptive beamforming for small aperture sizes, it is unable to deliver the same when using larger aperture sizes. Furthermore, even though the energy consumed by the GPU platform to process each image frame is lowest among the three, the actual power consumption of exceeding 270W makes them unsuitable for portable applications.

To that end, although mobile CPU platform may operate at close to 1W power consumption, it is not capable of performing any real-time processing of the image data. Alternative power-efficient high-performance processing architecture is thus needed to realize high-fidelity portable ultrasound imaging systems.

5.5 Future Directions

To process our target ultrasound images in real-time with high resolution, the performance of our current GPU implementation must be improved by another order of magnitude. Given the parallel nature of image formation, a simple way to obtain the needed performance in the future would be to employ more than 1 GPU as accelerators. Otherwise, given the current implementation is already approaching the performance limit of the GPU, improvements at the algorithm level, or radically different algorithm implementation designs will be needed.

Obtaining such performance at reasonable power consumption is more challenging. One way to address that is to investigate the use of low-power mobile graphics accelerators for image processing. The ARM processor SOC on the RHINO platform contains a PowerVR SGX 3D graphics accelerator that may be used for that purpose. Finally, implementing the beamforming algorithm directly on FPGAs is also a possible power-efficient solution.

6. CONCLUSIONS

In this paper, we have presented an initial design space exploration of implementing next-generation adaptive beamforming ultrasound imaging for both bedside diagnosis and out-of-hospital clinical uses. The performance and power consumption of a high-performance GPU based accelerator, a high-performance CPU only system, as well as a high-performance mobile CPU system was investigated. The GPU system had a clear performance advantage with improvements of 45x and 1810x over the desktop CPU and mobile CPU systems. On the other hand, in terms of power consumption, the power-optimized mobile CPU system out-performed the GPU and CPU system by factors of 272 and 70 respectively. However, when the effects of power consumption and performance are considered together, the performance-to-power ratio of the GPU system was surprisingly the highest among the three, making it the most power-efficient system under consideration.

Out of the three systems, only the GPU accelerated system was capable of real-time image processing at a reduced image resolution. However, the associated power consumption made it unsuitable for use in portable systems. In the future, we plan to utilize the FPGA located on the RHINO system considered in this work as an accelerator to achieve real-time performance at a manageable power consumption level. We will also further exam acceleration using lower-power GPU as an alternative if image quality can be sacrificed for power.

7. REFERENCES

[1] A. Langman et. al. Reconfigurable hardware interface for computing and radio, 2011.

[2] ARM Limited. *Cortex-A8 Technical Reference Manual*. ARM, Cambridge, United Kingdom, 2010.

[3] B. P. Nelson and K. Chason. Use of ultrasound by emergency medical services: A review. *J. Emerg. Med.*, 1:253–259, 2008.

[4] NVIDIA Co. *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. NVIDIA, Santa Clara, USA, 2010.

[5] J. F. Synnevag, A. Austeng, and S. Holm. Adaptive beamforming applied to medical ultrasound imaging. *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, 54(8):1606–1613, August 2007.

[6] J. F. Synnevag, A. Austeng, and S. Holm. Benefits of minimum-variance beamforming in medical ultrasound imaging. *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, 56(9):1868–1879, September 2009.

[7] P. N. T. Wells. Ultrasound imaging. *Phys. Med. Biol.*, 51:R83–R98, 2007.

[8] T. A. Whittingham. Medical diagnostic applications and sources. *Prog. Biophys. Mol. Biol.*, 93:84–110, 2007.