Optimal Architectures for Massively Parallel Implementation of Hard Real-time Beamformers

Literature Survey

Thomas Holme and Karen P. Watkins

12 March 1998

EE 382C

Embedded Software Systems

Prof. Brian Evans

Optimal Architectures for Massively Parallel Implementation of Hard Real-time Beamformers

Abstract

Computationally intense spatial filtering, called beamforming, is central to acoustic imaging. Typically, in order to achieve an update rate similar to slow scan television, beamforming must operate in real-time, which leads to massive parallel processing. This paper surveys appropriate work in the field of synchronous data flow (SDF) as applied to the beamforming problem and defines the objectives of a research project. The project is focussed on the application of SDF techniques as a strategy to select the optimal architecture for a beamformer. In order to formulate the proper approach, the applicable theory of SDF is reviewed, including methods used to optimize designs within the SDF domain. The beamforming process is described, and two competing architectures of the beamformer are defined, along with the optimality criteria for selection.

I. Introduction.

Computationally intense spatial filtering called beamforming is central to acoustic imaging. Typically, in order to achieve an update rate similar to slow scan television, beamforming must operate in real-time. The real-time nature of beamforming leads to parallel digital signal processing for state-of-the-art applications in diverse fields such as seismic exploration, aeroacoustic measurements, ultrasonic medical imaging, and underwater acoustics [1].

Many times, these applications require a large number of processors running algorithms in parallel. Efficiency in memory size, processor count, and execution time is critical to achieving an optimal system solution. Modern digital signal processor (DSP) attributes such as on-chip program and data memory, instruction cache, and direct memory access (DMA) can be exploited to achieve optimization. These DSP design features produce a need to minimize code size to fit in on-chip memory, maximize repeated instructions to utilize processor cache, and minimize interprocessor communication overhead in multiprocessor environments.

Modeling and synthesis of the system, using data flow graphs, can actualize these efficiencies. In particular, synchronous data flow (SDF) is appropriate for systems exhibiting hard real-time data independent control flow, such as the beamformers covered in this research. Modeling techniques appropriate for the application will be investigated and the modeling methodology will be selected for characterizing two beamformer architectures.

In this paper, we will first describe the project objectives and our approach, review SDF theory, and then cover methods used to optimize designs within the SDF domain. Following a review of the beamforming process, we will describe the project implementation. Finally, we will comment on the expected benefits of this research.

II. Project Objectives and Approach

The primary goal of this project is to identify the most efficient architecture for a prototypical beamformer by modeling two fundamental processing structures. This project is divided into two phases. The first phase, now complete, consisted of a comprehensive literature search. It was conducted to define the current research level in the two pertinent areas: SDF and beamforming.

In the follow-on second phase, we will model two beamformer architectures focussing on efficiency as described above.

III. SDF Modeling Approach for Optimization.

SDF is a natural representation for signal processing algorithms, such as beamforming, that are amenable to compile time scheduling techniques [2] [3]. In a data flow graph, computations in an algorithm are represented by vertices, called actors. These actors are connected by arcs which pass data in one direction between actors. An actor can only be fired (executed) when a predefined number of data values, called tokens, appear at its inputs. The tokens are placed into first-in first-out (FIFO) queues, called edges. An example of a data flow graph is given in figure 1.

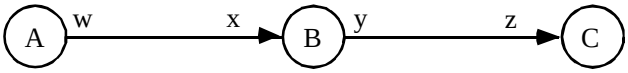
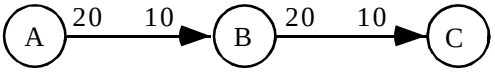


Fig. 1. An Example of a Simple SDF Graph

In this example, A, B and C are actors. The values w, x, y and z represent the number of tokens either generated or consumed by each actor.

In SDF, when an actor is fired, it consumes a fixed number of tokens on its inputs and produces a fixed number of tokens on its outputs. These parameters are known at compile time. Therefore, we can statically schedule algorithms represented by SDF. By schedule, we mean the task of assigning actors in the data flow graph to processors, ordering execution of these actors on each processor, and determining exactly when an actor fires so that all data precedence constraints are met [3]. A valid schedule is a finite schedule that fires each actor at least once, does not deadlock, and produces no net change in the number of tokens queued on each edge [4]. It can be found by constructing a topology matrix and solving for the repetitions vector which has the characteristic that, when multiplied by the topology matrix, produces a zero vector [5].

The schedule has a great impact on code size. Bhattacharyya, *et al.* illustrates this concept as shown in figure 2 [4]. Several valid schedules exist for this SDF graph. One such schedule is ABCBCCC. In this schedule, all code is in-lined, maximizing the code size. This schedule can be rearranged to group the same actor in repetitive firings called schedule loops, thus approaching minimal code size. If each actor appears only once in the schedule loop, as in schedules 2 and 3 of



1. ABCBCCC	1. 50
2. A(2B(2C))	2. 40
3. A(2B)(4C)	3. 60
4. A(2BC)(2C)	4. 50

a) SDF Graph b) Periodic Schedules c) Buffer Sizes

Fig. 2. Periodic Schedule Listing for SDF Graph

figure 2, no code duplication occurs. These schedules are called single appearance schedules, which result in optimally compact implementation of the SDF graph. In DSP architectures, loops have very low overhead and therefore can be used with virtually no increase in execution time.

The schedule also has a great impact on data buffer size [4]. For example, schedule 2 in figure 2 requires a buffer size of 40 whereas schedule 3 requires a buffer size of 60. This graph is simple. Generally, SDF graphs are more complex and therefore, buffer savings are more significant. Typically, to obtain the smallest memory requirements, the code size should be minimized as a higher priority than buffer size [4]. This strategy will be used to optimize the beamformer.

When more than a single processor is used, additional optimization can be realized while mapping the vertices onto multiple processors. One such technique uses clustering [2]. In clustering, the SDF graph is evaluated for natural grouping of vertices in a manual or automated process to produce hierarchical subgraphs. The subgraphs are scheduled using uniprocessor SDF schedulers. Clustering mitigates the problem of fully expanded graphs exploding in size. An additional problem with clustering SDF graphs is that the resulting graph can deadlock. Pino *et al.* developed the SDF composition theorem that when all conditions are met guarantees the clustered graph will not deadlock. They also demonstrated up to two orders of magnitude increase in execution speed and two orders of magnitude decrease in code size when using this technique [2]. Clustering will be exploited in the beamformer design.

Execution speed can also be improved by reducing interprocessor communication (IPC) overhead. There are three scheduling strategies: fully-static (FS), self-timed (ST), and ordered transactions (OT) [3]. All three scheduling strategies perform the actor assignments and specify the order of firings at compile time. In FS schedules, the exact order and firing times of each actor

are determined at compile time. This minimizes the IPC overhead. However, execution times must be deterministic and exact. Current DSP architectures use cache that may induce variability in execution times. Therefore, worse case execution time estimates must be used. In addition, the FS schedule is inflexible to system timing changes. In ST schedules, we retain the processor assignment and actor ordering used by the FS schedule. Instead of using exact firing times, processors determine when to fire an actor by synchronizing with other processors at run-time. This increases IPC overhead but is tolerant of variable task execution times. OT scheduling is an intermediate approach. The OT method determines the order in which the processors should communicate but does not determine exact times to communicate. The order is enforced at run-time. The IPC overhead is slightly increased from the FS schedule but is decreased from the ST schedule. However, this technique retains the flexibility of the ST schedule. The OT scheduling technique will be used in the beamformer implementation since it matches the data independent execution time and need for flexibility in changing beamformer parameters.

IV. Time Delay Interpolation Beamforming

The beamformer function is essentially a spatial filter that processes discrete signals from a number of sensors in order to produce a directionally-sensitive response. The individual sensors are distributed in space to provide sensitivity to sound arriving over the intended field of view. Near field conditions of spherical wave propagation are assumed. The signals arriving from the desired look angle are reinforced by imposing time delays on each sensor so that the signal formed by the summation of all sensor signals is maximized (figure 3). This is referred to as steering the

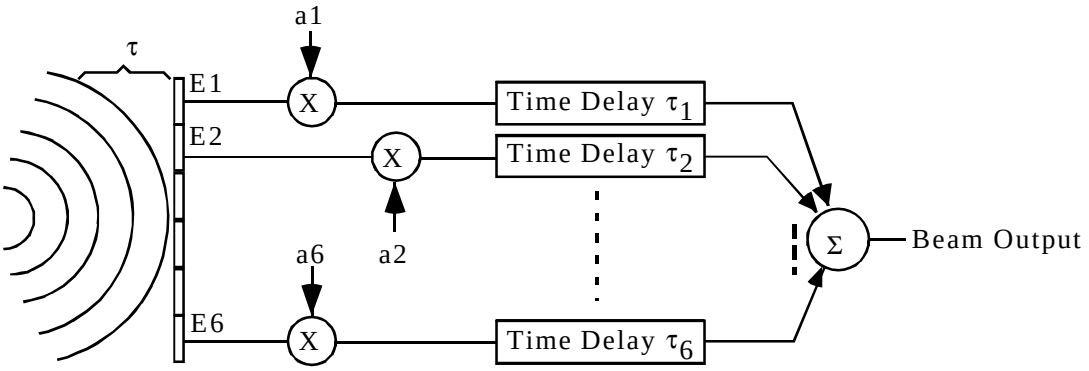


Fig. 3. Time Delay Imposed for Time-of-Arrival Compensation for Beamformer Function

beam. The resulting beam enhances the signal level of sound arriving from the intended angle compared to the background noise level. In modern systems, the time delays are implemented with digital signal processing. However, the sampling rate required to adequately describe the temporal signal may be too coarse for the resolution of the time delays needed to accurately steer the beam to the desired direction. If the system were designed such that the signals from all sensors were sampled at the higher rate required by the spatial requirement, a severe penalty in the number of processors and the speed of operation would result. The concept of interpolation beamforming is used to mitigate the spatial sampling requirement and control the amount of processing needed to form accurately steered beams [1].

In interpolation beamforming, the signals from each sensor is sampled at or above the Nyquist rate dictated by the bandwidth of the signal (figure 4). These signals are augmented with inter-

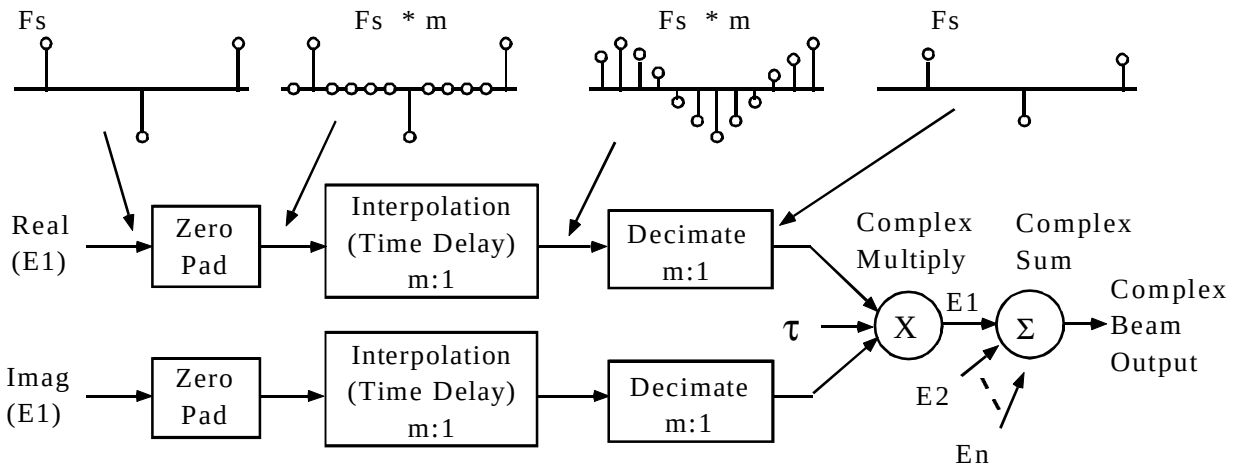


Fig. 4. Time Delay Interpolation Beamforming with Complex Samples

sample zeroes (zero padding) to effectively increase the input sample rate. Next, a low pass filter is used to interpolate between the real samples, thus creating an accurate representation of the input signal now sampled at the higher frequency. Typically, this low pass filter is implemented as a finite impulse response (FIR) digital filter [6]. Following summation of the individual sensor signals, the output is decimated to return to the original sampling rate. In actual practice, only the non-zero samples are multiplied by the filter coefficients. In addition, the filter coefficients selected for a given sensor signal are chosen to impose the relative delay needed between this

sensor signal and the others in the array to steer the beam in the intended direction. This results in computational efficiency while maintaining the accuracy of the formed beam.

Further efficiencies can be achieved by analyzing the point at which the interpolation is invoked. In systems forming a large number of beams, where the number of beams is larger than the number of sensors, it has been demonstrated that interpolation of each sensor signal is most efficient. However, if the number of beams is less than the number of sensors, it can be more efficient to zero pad the input signals, sum the signals for each beam and then interpolate the resulting beam signals [6]. Application of this architecture is limited to those systems in which imposed time delays are not altered during the time that the signals are observed. In cases where the time delays are modified as the signals are received, the time delays must be imposed on the individual sensors and the interpolation/filtering executed at the sensor level. The beamformer we are implementing operates under this constraint.

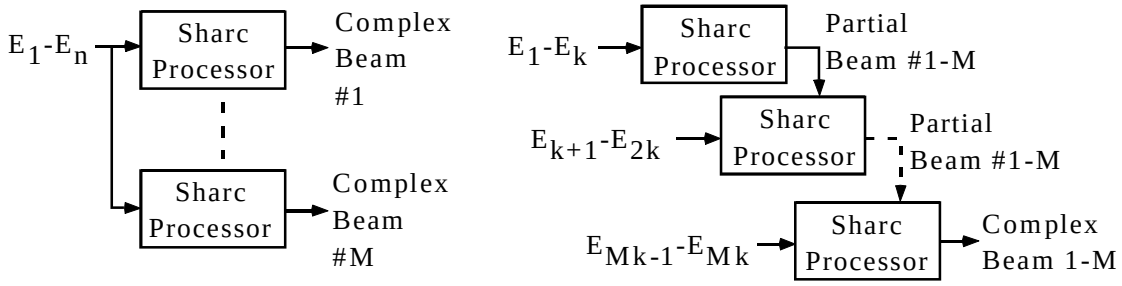
In acoustic systems using very high frequencies, the bandwidth of the information is a small percentage of the operating frequency. If first order (Nyquist) sampling were used to define the high frequency signal, the information bandwidth would be highly oversampled. Therefore, additional efficiencies in sampling rates can be achieved by employing a sampling technique that preserves the information in the envelope of the signal by adequately oversampling that bandwidth. This technique, called second order or quadrature sampling, has been extensively used in this context [7]. This technique down-converts the bandpass signal to complex baseband, producing a real and imaginary sampled signal at a combined sampling rate that is much less than that required to retain the high frequency waveform. By performing a precise sampling process, this procedure characterizes a bandpass signal, x(t), with uniformly spaced samples of its quadrature components. Therefore, x(t) can be expressed as

$$x(t) = x_I(t) \cos(2\pi f_0 t) - x_Q(t) \sin(2\pi f_0 t) \quad (1)$$

where $x_I(t)$ and $x_Q(t)$ denote the in-phase and quadrature components of x(t) and f_0 denotes the center of the passband [1]. This sampling will be used in the beamformer for efficiency.

V. Project Implementation

In this project, two possible beamformer architectures will be evaluated. The primary difference between the two architectures centers on the method of mapping computations onto processors. The two architectures are shown in figure 5. In the first architecture (figure 5a), data



a) Each processor forms one beam
No. Processors = M Beams

b) Each processor forms full set of partial beams
No. Processors = No. sensors/k

Fig. 5. Two Beamformer Architectures To Be Evaluated

from all sensors are input into each processor. Each processor then imposes the appropriate time delays, filters and sums the data, and outputs a single complex beam. In this architecture, the number of processors required equals the total number of beams to be formed. In the second architecture (figure 5b), data from one or more sensors are input into each processor. Each processor then imposes the appropriate time delays for all beams produced, filters and sums the data. The output of each processor is a partial beam. The complete beam set is fully formed in the last processor in the chain. In this architecture, the number of processors required equals the number of sensors in the array divided by the number of sensors input to each processor. Clearly, the architecture that yields the least number of processors is the most efficient implementation.

The basis of computation is the Analog Devices Sharc DSP. The Sharc is designed for parallel processing with six high speed communication channels using independent DMA controllers per channel. In addition, this processor has one megabit of on-chip memory. Due to space limitations, no more than 160 processors can be allocated to form the requisite 160 beams using 32 sensors per beam. The processors will be spread over multiple boards, each housing 18 Sharcs per board.

In this project, the two architectures described above will be modeled using SDF domain in Ptolemy. Modeling will focus on the most intense portion of the problem: the interpolation

filtering/decimation/summation structure. Optimization techniques described in this paper will be used. The execution time of each architecture will be measured and bottlenecks will be identified. Based upon the modeling, the processor count will be evaluated. The optimal architecture will be selected where the criterion for optimality is minimization of the number of processors while achieving the required throughput. For this architecture, Ptolemy will be used to generate the C source code and where bottlenecks occur, assembly language will be written.

VI. Anticipated Intellectual Reward

Several benefits will be derived from this research. For example, efficiency of the candidate algorithms will be evaluated by quantitative metrics rather than by the *ad hoc* approach taken in the past. In addition, innovative processor topologies may evolve from the modeling process, based upon the insight into the potential parallelism evident in the SDF graph. More explicitly, final implementation of the system onto processors will be enhanced through this process. Finally, future designs can be based on this design approach, leveraging the knowledge and insight gained.

References

1. R. A. Mucci, "A Comparison of Efficient Beamforming Algorithms", *IEEE Trans. on Acoust., Speech, Signal Processing,* vol. ASSP-32, no. 3, pp. 548-557, June 1984.
2. J. L. Pino, S. Bhattacharyya, and E. A. Lee, *A Hierarchical Multiprocessor Scheduling Framework for Synchronous Dataflow Graphs*, UCB/ERL M95/36, May 30, 1995.
3. S. Sriram and E. A. Lee, "Determining the Order of Processor Transactions in Statically Scheduled Multiprocessors", *Journal of VLSI Signal Processing*, vol. 15, no. 3, pp. 207-220, March 1997.
4. S. Bhattacharyya, P. Murthy, and E. A. Lee, "Optimized Software Synthesis for Synchronous Dataflow", *Proc. of ASAP '97 Conference*, Zurich, Switzerland, July 1997.
5. E.A. Lee and D. G. Messerschmidtt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing", *IEEE Trans. on Computers*, vol. C-36, no. 1, pp. 24-35, January 1987.
6. R. G. Pridham and R. A. Mucci, "A Novel Approach to Digital Beamforming", *Journal of Acoust. Soc. Amer.*, vol. 63, no. 2, pp. 425-434, February 1978.
7. D. C. Horvat, J. S. Bird, and M. M. Goulding, "True Time-Delay Bandpass Beamforming", *IEEE Journal of Oceanic Engineering*, vol. 17, no. 2, pp. 185-192, April 1992.