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

Final Report

Thomas Holme and Karen P. Watkins

8 May 1998

EE 382C

Embedded Software Systems

Prof. Brian Evans

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

Abstract

This paper reports an experimental analysis of real-time computational architectures applied to a digital time delay beamformer implementation. The goal of this research has been to identify the most efficient multiprocessor utilization for a prototypical beamformer by modeling the signal processing and applying selected multiprocessor scheduling algorithms. The analytic method centered on modeling the most computationally intense core of the beamformer signal processing in Ptolemy. First, the Synchronous Dataflow (SDF) domain model used to implement the beamformer core is presented. By exercising the model, the computational results were compared to an existing equation-based model for validation. Secondly, the SDF model was translated into a Code Generation in C (CGC) domain model, in order to evaluate multiprocessor scheduling performance. Four well-known automated scheduling strategies were experimentally applied: declustering, a classical list scheduler, dynamic-level scheduler, and hierarchical scheduler. A manual heuristic schedule was also postulated and evaluated. Several key metrics were applied in judging optimality. For this application, it is demonstrated that the hierarchical scheduler holds measurable advantage over the other algorithms considered, including the manual method. Resulting metrics are reported along with an analysis of the underlying performance drivers.

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. Digital signal processor (DSP) attributes such as on-chip memory, instruction cache, and direct memory access (DMA) can be exploited to achieve optimization.

Modeling and synthesis of the system, using dataflow graphs, can actualize these efficiencies. In particular, Synchronous Dataflow (SDF) is appropriate for systems exhibiting hard real-time data-independent control flow, such as beamformers. The primary goal of this research has been to identify the most efficient multiprocessor utilization for a prototypical beamformer by modeling the signal processing and applying selected multiprocessor scheduling algorithms. Here, we determine the optimal scheduling for the process where the criterion for optimality is minimization of the required cycle counts with a constrained number of processors.

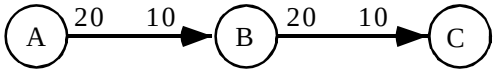
In this paper, we first review SDF theory and multiprocessor scheduling algorithms. A prototypical beamformer problem is defined along with its corresponding SDF model. We invoke selected multiprocessor scheduling algorithms in order to find an optimized solution. The results are presented along with an analysis of the performance of the algorithms.

II. 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 SDF, when an actor is fired, it consumes and produces a fixed number of tokens. Since these parameters are known at compile-time, algorithms represented by SDF can be statically scheduled. By scheduling, we mean the task of assigning actors in the dataflow graph to processors, ordering execution of these actors,

and determining when an actor fires so that all data precedence constraints are met [3]. A valid schedule is finite and fires each actor at least once, does not deadlock, and produces no net change in the number of tokens queued on each edge [4].

The schedule has a great impact on code size. Bhattacharyya, *et al.* illustrates this concept as shown in Figure 1 [4]. This figure lists several valid schedules including schedules with single and multiple appearances of the actors. If each actor appears only once in the schedule, no code duplication occurs, which results in an optimal graph.



a) SDF Graph

	b) Periodic Schedules	c) Buffer Sizes
1.	ABCBCCC	50
2.	A(2B(2C))	40
3.	A(2B)(4C)	60
4.	A(2BC)(2C)	50

Fig. 2. Periodic Schedule Listing for SDF Graph

When more than a single processor is used, additional optimization can be realized while mapping the nodes onto multiple processors. Prior research into optimal multiprocessor schedulers has produced several algorithms including declustering (DC), a classic list scheduler based on Hu's work (HU) [5], dynamic-level scheduling (DLS), and the hierarchical scheduling algorithm. The DC algorithm targets systems using homogeneous architectures that communicate through global shared memory. This is a multi-pass algorithm that attempts to minimize interprocessor communication (IPC) cost by scheduling all communications as well as all computations [6]. The HU algorithm assigns each node a priority and places the nodes in a list, sorted in order of decreasing priority. A global time clock regulates the scheduling process by assigning nodes onto processors as they become available. This algorithm does not consider IPC costs [7]. The DLS algorithm modifies the HU algorithm by eliminating the global clock in an effort to reduce IPC costs. This allows the algorithm to consider all processors, including processors still "busy", as candidates for scheduling at each step [7].

The hierarchical scheduling algorithm clusters nodes in a manual or automated process to optimize the schedule [2]. The clustered subgraphs are then scheduled onto single processors using uniprocessor SDF schedulers. Clustering mitigates the problem of fully expanded graphs

exploding in size but can introduce deadlock during this process. This algorithm guarantees that the clustered graph will not deadlock by applying the SDF composition theorem.

III. Time Delay Interpolation Beamforming

The beamformer function studied here is a spatial filter that processes discrete signals from a number of near-field sensors in order to produce a directionally-sensitive response. 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. 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 point the beam to the desired direction accurately. The concept of interpolation beamforming is used to mitigate the spatial sampling requirement and control the amount of processing needed to form accurately pointed beams [1].

In interpolation beamforming, the signals from each sensor are sampled at or above the Nyquist rate dictated by the bandwidth of the signal (Figure 2). These signals are augmented with inter-sample zeroes (zero padding) to effectively increase the input sample rate. Next, a lowpass finite impulse response (FIR) filter is used to interpolate between the samples, thus creating an accurate representation of the input signal now sampled at the higher frequency [8]. Following summation of the individual sensor signals, the output is decimated to return to the original sampling rate. In practice, only the non-zero samples are multiplied. In addition, the filter coefficients selected for a given sensor signal are chosen to impose the relative delay needed between this sensor and the others in the array to point the beam in the intended direction.

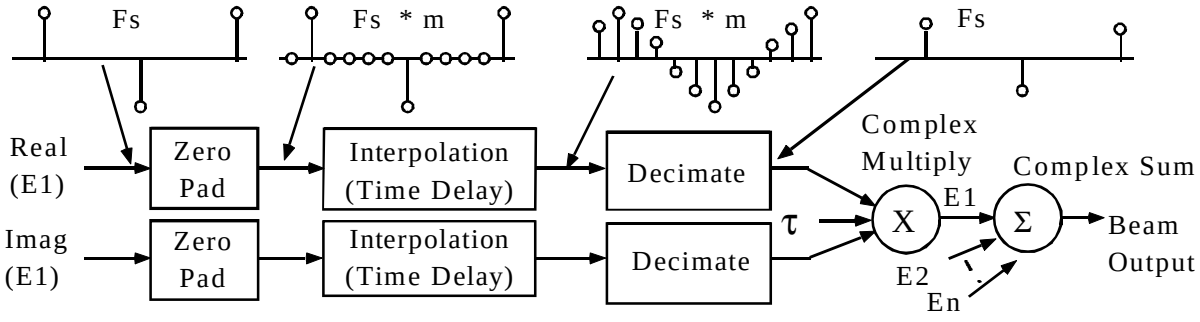


Fig. 2. Time Delay Interpolation Beamforming with Complex Samples

IV. SDF Modeling of the Beamformer

In this research, the beam formation described above was modeled in the SDF domain using Ptolemy. The prototypical system under study produces 160 beams using 32 sensor inputs per beam. The formation of four beams was modeled and verified (Figure 3). This nucleus of 4 beams must be replicated 40 times to produce the full system.

Simulated sensor input data were generated using MATLAB. In order to simulate the system sampling, real and imaginary data points were generated and read from files into Ptolemy during simulation. The data was distributed in the model as inputs to four sets of galaxies and stars, forming beams one through four. In each set, the core galaxy was replicated 32 times. This galaxy consisted of two multiply-accumulate (MAC) stars and one complex multiply star. Each MAC star executed an eight-tap FIR filter. Previously computed coefficients for each FIR filter were used to impart the required time delay. The output of each galaxy set was a partially formed beam. The fully-formed beam was produced by summing all partial-beam real and imaginary data samples. This model was verified by comparing the beam output sample points to a conventional equation-based model using MATLAB. Excellent agreement was demonstrated.

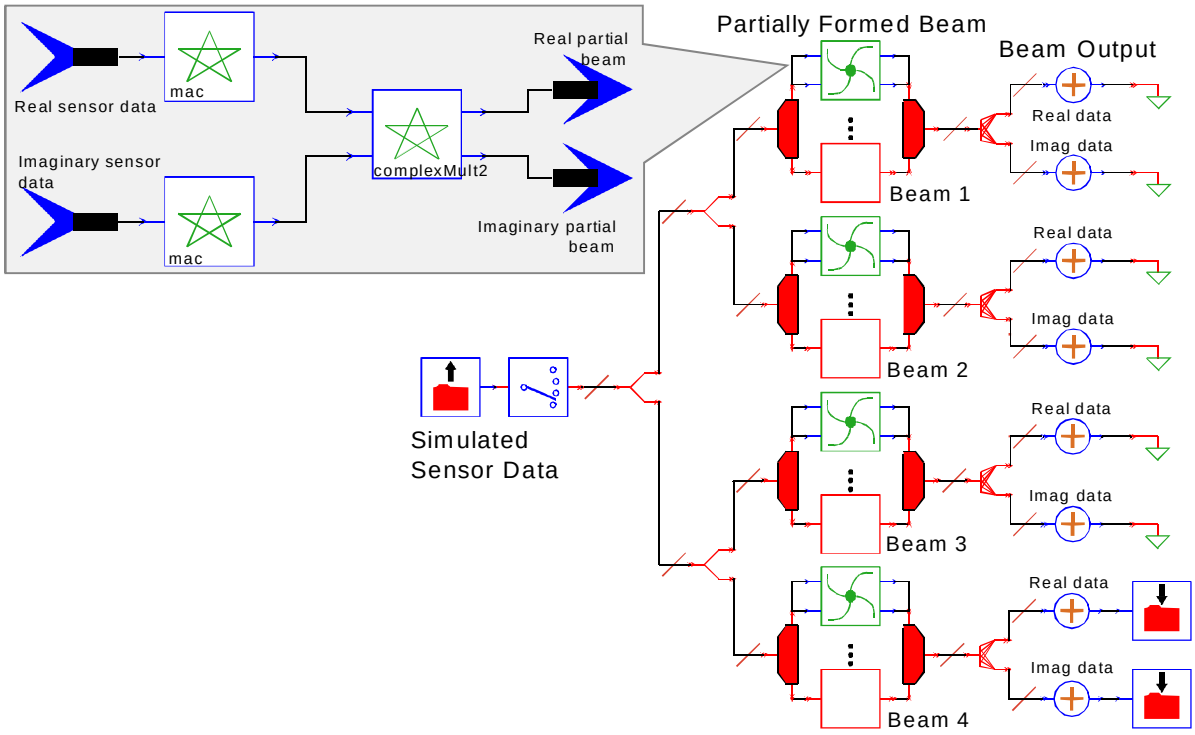


Fig. 3. SDF Beamformer Model Created in Ptolemy

V. Code Generation Domain Modeling and Results

The SDF model was translated into the Code Generation in C (CGC) domain. Using this domain, the multiprocessor scheduling algorithms described earlier were invoked. For this example, the number of processors was constrained to four. Execution times for each node were input to the algorithms to achieve realistic schedules. Here, assembly code execution times were entered into the model for the processor of interest, the Analog Devices SHARC DSP.

The CGC model was exercised several times invoking various multiprocessor scheduling algorithms, namely, DC, DLS, HU, and hierarchical. In addition, a manual scheduling method was tested. This method confined the formation of one beam to one processor. Each schedule was evaluated for total execution time, remaining free cycles, and processor utilization.

The schedules for each algorithm are shown as summary Gantt charts in Figure 4, with the exception of DC. The DLS and HU algorithms produced nearly identical schedules. In both algorithms, one processor was used to calculate one of the complex components of the FIR filter for 16 sensors for all four beams. Both algorithms then assigned a mixture of complex multiplies and additions across all processors to form the four complete beams. Each algorithm apportioned

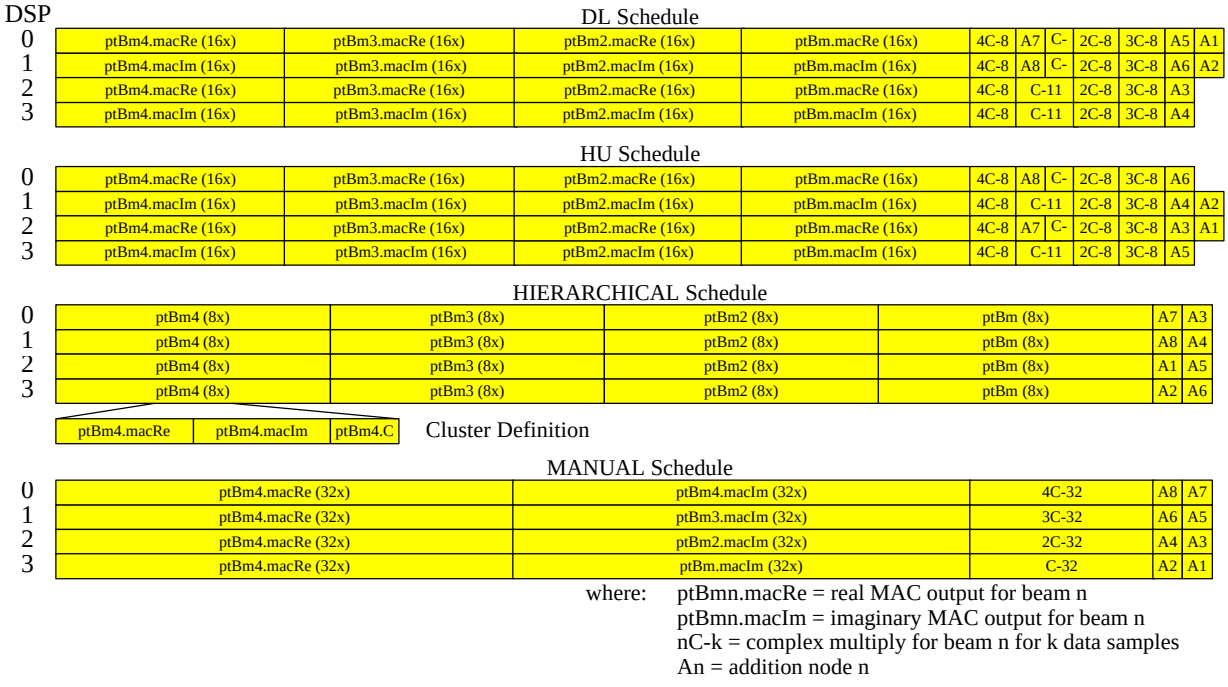


Fig. 4. Schedule Results

the mixture differently.

The hierarchical schedule clustered the two MAC stars and the complex multiply star shown as a galaxy in Figure 3. Each processor executed this clustered galaxy for eight of the 32 sensor signals used to form a beam, for all four beams. Finally, each processor performed two separate addition functions. This schedule has the benefit that all code is identical across all processors. Therefore, incremental implementation and testing can be executed on a single processor.

In the manual schedule, for a single beam, each processor calculated all real components of the FIR filter, followed by all imaginary components. It then concluded by calculating the complex multiply and the additions. This schedule shares the benefit that all code is identical on all processors.

The execution throughput, or makespan, is by far the most valued metric in this analysis. A low makespan value indicates high performance. The results for all algorithms are shown in Table 1. As indicated, the DC algorithm produced the highest makespan value, far exceeding all other schedules. In cases of excessive communication costs, this algorithm will only split up tasks if it can reduce the IPC costs. In the limit, this leads to scheduling the entire structure onto a single processor [6]. Indeed, the beamformer prototype induced this solution and the algorithm assigned all nodes to a single processor, which produced unacceptable load sharing. Therefore, this algorithm will not be discussed further. The DLS and HU schedules produced identical makespan values and were also the highest values of the remaining four. The manual schedule produced an intermediate value. The hierarchical schedule produced the optimal makespan.

In a real time system, the repetition time of the complete processing epoch translates to a fixed upper limit on the number of processor cycles available for execution. As an indicator of

Table 1. Resulting Performance Metrics

Schedule Algorithm	Makespan (# cycles)	Remaining Free Cycles	Processor Utilization
DC	4929		
DLS	1234	56	95.66%
HU	1234	56	95.66%
Hierarchical	1222	68	94.73%
Manual	1228	62	95.19%

efficiency, the metric of remaining free cycles has been used to determine if the required throughput has been met. Large values of remaining free cycles are considered desirable. The number of remaining free cycles is shown in Table 1 with processor utilization also listed. Again the hierarchical schedule is shown to be most efficient. The differences in the remaining free cycles are small across all algorithms. However, since the processor utilization is near maximum, schedule optimization is even more crucial to achieving the required throughput.

VI. Analysis

The hierarchical scheduling algorithm produced the optimal schedule for the real-time interpolation beamformer, producing the fastest throughput and the largest number of free cycles. It also produced a schedule in which identical code resides on all processors. Clearly, the results are attributable to the strategy embodied in the algorithm. Examining the details of the scheduling output, this method first clustered the multiply-accumulate pair with its corresponding complex multiply. It then scheduled this cluster onto a single processor. This permitted the algorithm to use uniprocessor schedulers that provided further optimization such as producing a single appearance schedule. By clustering the core computational nodes in the model, namely the MAC stars and the complex multiply stars, this algorithm permitted efficiencies not available to the other algorithms. By clustering these nodes, the algorithm internalized any communications between these nodes, thus eliminating the associated overhead. This is analogous to having on-chip memory in the DSP where intermediate values are held in internal registers that become immediately available for following computations. The net effect was a shorter execution time, as indicated by the simulation results.

The performance of the DLS and HU algorithms is tied to the mechanism by which nodes are assigned to processors. Nodes are assigned a priority that determines order of execution placed in an ordered list. In HU, nodes are assigned to processors for execution as they become ready. The DLS algorithm considers all processors and includes weighting for IPC costs and current processor task load. Not withstanding these differences, both algorithms are slaved to this prioritized list, and therefore suffer the resulting inefficiencies.

VII. Benefits Derived from the Research

Several direct benefits resulted from this research. The comparison of scheduling algorithms in a severely time-constrained application produced valuable increases in throughput. The optimal schedule reported here has been shown to have higher performance than an intuitive *ad hoc* manual method. Indeed, through this research, the goal of producing a schedule with a constrained number of processors and minimal cycle count has been met. In addition, the resulting schedules permit insight into the processing flow, parallelism, and bottlenecks that may not be apparent on the surface.

In the broader view, traditional system design has been by trial and error. When throughput was not realized, more processors were added, increasing cost and complexity. By modeling a system prior to implementation, with attention to execution times and accurate IPC costs, early insight into final system topology and dataflow is achievable. Modeling provides optimal schedules derived from numerical analysis that cannot be achieved through manual methods.

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, Mar. 1997.
4. S. Bhattacharyya, P. Murthy, and E. A. Lee, “Optimized Software Synthesis for Synchronous Dataflow”, *Proc. of Application Specific Array Processors ’97 Conference*, Zurich, Switzerland, July 1997.
5. T. C. Hu, “Parallel Sequencing and Assembly Line Problems”, *Oper. Research*, vol. 9, pp. 841-848, Nov. 1961.
6. G. Sih and E. A. Lee, “Declustering: A New Multiprocessor Scheduling Technique”, *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 6, pp. 625-637, Jun. 1993.
7. G. Sih and E. A. Lee, “A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures”, *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175-187, Feb. 1993.
8. 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.