Real-Time Sonar Beamforming on a MasPar Architecture

José Salinas, W. Robert Bernecky
Systems Architecture and Software Development Branch
Naval Undersea Warfare Center Division Newport
Code 2153, Bldg. 80
New London, CT 06320

Abstract

This paper presents a novel approach for performing real-time sonar beamforming on linear sensor arrays using the MasPar SIMD architecture. The beamforming problem is defined as a three dimensional solution space by generating a cube structure with sonar array elements as one dimension, the required beams in another dimension, and the time samples in the third dimension. The given approach maps the problem cube into the MasPar structure using a modified one-to-one mapping and uses two MasPar Fortran 90 intrinsic array functions to generate the solutions to the beams. Simulation results are provided for different array and beam sizes.

1. Introduction

As the capacity of commercial parallel processing architectures continues to increase, research in the area of parallel algorithm mapping has become increasingly important. This is especially true of many applications which require large amounts of computations, data, and IO rates [4]. When dealing with real time applications, it is also imperative that results be computed within predetermined time deadlines. One such type of application is sonar beamforming [2,5].

Beamforming is defined as the process of generating a set of beam patterns (or beams) with some directionality properties by manipulating a set of signal values read in from a sensor array structure [1,3]. In the past, time-domain beamforming was accomplished by buffering the sensor signals and introducing a set of time delay circuits in the beamformer hardware to compensate for the time delays between separated elements listening to the same sound source [6,7].

Traditionally, beamforming has required customized hardware tailored to a particular sonar array in order to achieve the required performance. However, as the size of new array designs continues to increase, the cost of designing, manufacturing, and maintaining beamforming hardware has become economically unattractive. One solution is to use COTS computer systems for these applicatons. Since a large number of the computations involved in computing the beams of an array are independent and disjoint, commercial fine grained parallel architectures are very well suited for complex beamforming algorithms [2].

This paper focuses on implementing a beamformer algorithm for use in a linear array of transducer elements on a commercial MasPar SIMD system. A cyclic parallel mapping approach is used to map the three dimensional cube into the structure of the MasPar machine.

2. Problem definition

2.1 Beamforming model and assumptions

Figure 1 shows the model used for beamforming on linear arrays of transducer elements. The model consists of a single unique signal sound source (S) located at an infinite distance from a linear array of n transducer elements. S is modeled as a sinuosoidal sound wave traveling with a velocity c, a frequency f, and wavelength $\lambda = c/f$. The sound speed c is assumed to be approximately 1530 m/s but can vary depending on propagation factors.

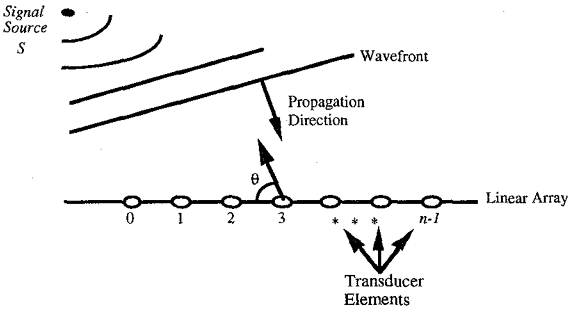

Figure 1: Beamforming model.

Within the array, an inter-element separation d is defined as half the wavelength ($\lambda/2$) of S. Since S is assumed to be at an infinite distance, wavefronts arriving at the array will be parallel to each other and form a planar wavefront.

In this case, all sensors are assumed to have equal response curves for all signal values. An ideal transmission medium is also assumed for the signal propagation. The output signal of the array consists of the value from each transducer element output at the array sampling frequency.

If S is located at an angle which is not perpendicular to the array, the signal wavefronts will reach the array sensors at different times proportional to the angle of arrival, θ. Time-domain beamforming consists of delaying the input sensor values by introducing a time delay for each element so that the maximum array response for a particular bearing is achieved when these values are added. Since a signal received at the array elements consists of coherent values, the maximum sensitivity of the array for a particular direction is achieved when the outputs of the sensors are added in phase.

When dealing with sonar array systems, multiple beams can be computed in parallel. Typically, the number of beams computed is one more than the number of sensors using Cosine spacing (i.e. the separation between two consecutive beams is the inverse Cosine of a value between -1 and 1). This generates beams with closer spacing at $\theta=90$ degrees to account for narrower beams at the broadside look direction.

2.2 Beamforming Domain Cube

The beamforming problem is bounded by three variables: the number of array elements, the number of beams to generate, and the amount of storage needed to hold the buffer values for each element at the particular array sampling frequency. A three dimensional cube representation is derived using each of these three variables as different dimensions of the cube stored in a buffer structure.

Using this structure, the beamforming problem is defined as a set of beamforming operations on the cube. Each cube intersection point, $v_{i,j,k}$, in the cube represents the input values for beam j, at time k for the transducer element i.

The beam values are computed by adding the set of sensor values at some point in the time buffer to achive the required directionality for that beam. Beams in a linear array are completely independent, therefore, computations can be done across different *element* × *time* "rows" of the cube in parallel.

2.3 SIMD mapping

By defining the beamforming problem as a cube, a mapping of the cube to a MasPar array can be derived using a modified one-to-one mapping approach. The mapping consists of assigning each cube value $v_{i,j,k}$ of the cube to the local memory of a PE in the PE array (either in a register or in memory). In the case of the MasPar system, the 2D array of PEs is assigned to the "faces" (*element* × *beam* slices) of the cube in a one-to-one manner using a cyclic approach. Subsequent slices corresponding to other time indices will get mapped to subsequent locations in the PE memory. Hence, each PE at (i,j) will contain the input values for the entire time buffer for sensor i and beam j.

When the number of beams or elements is larger than the size of the PE array, a cyclic mapping approach will map values on the PEs using more than one *beam* × *element* value per PE. Values not covered by the initial mapping will be mapped on to subsequent memory layers in the same one-to-one approach. This will guarantee the entire problem space is covered by the smaller PE array.

3. General beamforming algorithm

A general parallel beamforming algorithm based on the problem cube is shown in Table 1. The algorithm consists of a main infinite loop which reads in the sensor values of the array, computes the beams, and outputs the results. Array values for each sensor are read into the algorithm at the array sampling frequency, therefore, the beam computations have to be performed in real time before the next set of values is available at the next time step. Additionally, each of the sensor values can be multiplied by an optional shading coefficient which modifies the beam pattern.

Table 1: General SIMD algorithm.

```
1. initialize current time
2. compute the initial delay values for each
   element x beam combination
do forever
   3. read in sensor values from the array
   4. multiply sensor values by shading
       coefficient (if applicable)
   5. insert values into the problem cube at
       the current time index and spread to
       at all beam positions
   do for each beam
       do for each sensor position
           6. compute the position of the
               value corresponding to the
               delay for the current beam and
               element
           7. retrieve the value at that
               position
       end do
       8. add all the sensor values retrieved
           to generate the beam value for the
           current beam
   end do
   9. output all the beam values
   10. update current time (using a circular
       buffer approach)
   11. update the delay values for each beam
       x element combination
end do
```

Once the values have been read in, they are distributed across all the beam indices. This will assure all the beams have access to the same input values during the beam computations. Forming each beam consists of adding the input value for all the sensors corresponding to the time index which gives the maximum response at the current bearing. These values are then retrieved and make up the beam partial sum values. The final beam results are computed by adding the set of values retrieved.

If this algorithm is executed sequentially, the computational complexity for each iteration of the main loop depends on the the two nested loops. Given a set of n array elements and a set of m beams, the computational complexity of each iteration is given by **O**(nm). If m is defined as $n+1$, then the running time becomes **O**(n^2).

4. Algorithm on the MasPar

Once the mapping of the cube on the SIMD machine has been implemented. The beamforming computations can be applied to the mapped values to generate the beams. An intrinsic data movement instruction and an array reduction function were used for implementing the input spreading and partial sum additions. This algorithm is described in Table 2 using pseudo Fortran 90 code.

Table 2: MasPar algorithm.

```
1. initialize variable: TIME
2. intialize arrays: CUBE,  SHADE_COEFF,
   PARTIAL_SUM, RESULTS
3. compute initial delay values for each
   element × beam combination and store in
   DELAY_VALUES
do forever
    4. CUBE(:,0, TIME ) = input values from
       the array in parallel
    5. CUBE(:,0, TIME ) = CUBE(:,0,TIME) *
       SHADE_COEFF -- if applicable
    6. CUBE(:,:, TIME) = SPREAD( A, dim = 2,
       copies = MaxBeams )
    7. PARTIAL_SUM = CUBE(:,:,
       DELAY_POSITION)
    8. RESULTS = SUM( PARTIAL_SUM, dim = 1 )
    9. output RESULTS in parallel
    10. update TIME, DELAY_POSITION (using
        circular buffer approach)
end do
```

This algorithm uses the two built-in MasPar Fortran 90 functions: SPREAD and SUM. The main data structure is the CUBE array. This is a 3D array with elements × beams × time dimensions which holds the input values for each element and beam combination for up to MAX_TIME samples. A TIME scalar variable is used to keep track of the current time in the procedure as well as to index into the time dimension of the CUBE array. The SHADE_COEFF array is a one dimensional array which contains the optional shading coefficients for each of the elements. The initial delay for each of the sensors and beams is given by the two dimensional DELAY_POSITION array. This array depends on the physical position of the element within the array and the bearing of the beam. Beam partial sums are stored in the PARTIAL_SUM array. Finally the RESULTS array is a one dimensional array which is used to output the set of beams.

The first steps initializes all the necessary arrays in the procedure before any beamforming computations are done. An infinite loop computes the set of beams for the current time at each iteration. Input values are read into the first row of the CUBE array in parallel at the position corresponding to the current time as shown in Figure 2(a). An optional shade coefficient multiplication can also be done at this stage.

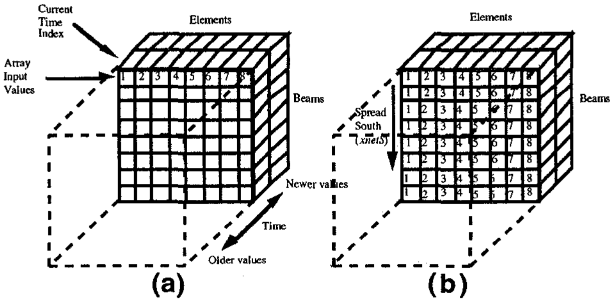

Figure 2: (a) Array input value in CUBE, (b) distribution of values across the beams.

Once the values have been stored in the first row, the elements are distributed to the rest of the beams using a SPREAD function. The function will take as input an array and replicate it across a given dimension. In this case, the input values consist of the array element values which need to be copied along the beam dimension. The SPREAD function will use the MasPar built-in MPL (MasPar Programming Language) communication instructions to accomplish the data transfer as shown in Figure 2(b).

Once all the beam rows have the new input values, the delayed element values can be retrieved in parallel using the element delays indices computed in the initialization phase and stored in the DELAY_POSITION array. For each element × beam position, the DELAY_POSITION values will be used to index into the CUBE array along the time dimension using a table lookup approach as shown in Figure 3. The PARTIAL_SUM array will be used for storing the delay values retrieved.

The final stage is to compute the beam results using the values in the PARTIAL_SUM array. The array reduction function SUM is used to add the delay values along the element dimension and storing the results in the RESULTS array. This array will contain the beam values for the current time which are then output from the beamforming procedure in parallel.

The final step is to update the TIME variable and DELAY_POSITION array. The TIME variable can be updated by adding 1 or resetting to 0 if it has passed the index limit of the time dimension of the CUBE array. Similarly, all the DELAY_POSITION values are updated by 1 or reset to 0 in parallel if the position is greater than the maximum time index. A test and reset approach is implemented for updating the TIME variable and DELAY_POSITION array to reduce execution time.

For the complexity analysis, the main bottlenecks of the algorithm are the two built-in functions SPREAD and SUM. These functions cannot be done in constant time and depend on the size of the array. All other steps can be done in $O(1)$ time. Assuming an efficient implementation of these functions, then the SUM and SPREAD functions can be accomplished in $O(\log n)$. Hence, the total complexity for the main loop body is given by $O(\log n)$.

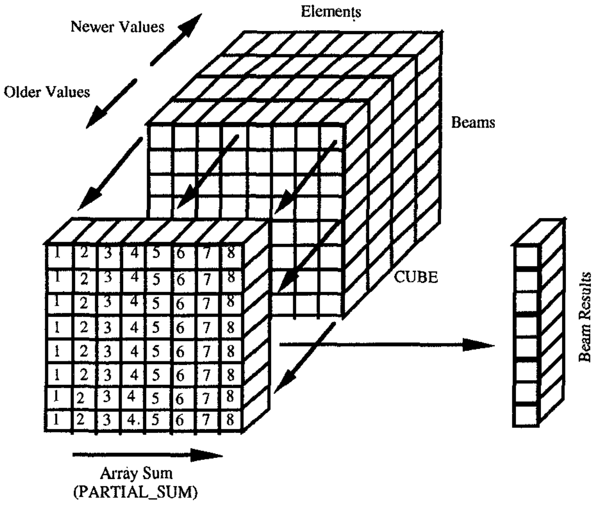

Figure 3: Beam Results Computations

5. Performance results

The beamforming algorithm was executed on a *64* × *64* MasPar MP-2 machine. Several different combinations of beams and elements were used with a simulated signal source. The computations were executed for 100,000 iterations and the average execution time computed. These results are given in Table 3.

The best execution time was achieved for array sizes and beam numbers less than or equal to the size of the machine (i.e. 64 × 64). Since the mapping used for the algorithms was a one-to-one mapping of element and beam values to PE elements, the execution time for arrays which are smaller than the maximum size of the machine is the same as for arrays which are the size of PE array. Similarly, if the problem is larger than the size of the PE array and is a multiple of the machine size, then PEs will have to do more than one computation for each element × beam combination, hence, the performance will degrade accordingly.

Table 3: Performance results

Elements	Beams	Time (μsec)
32	32	152
64	64	152
65	65	774
128	128	601
192	192	1117
256	256	1571
256	512	2981

The worst performance was achieved for arrays which are not a multiple of the machine. In these cases, the compiler used array masking in order to compute operations on the array which are mapped to a subset of the PEs. If the array × beam combination is not a power of 64, then dummy elements and beams can be inserted to improve the beamforming performance.

6. Conclusion

This paper introduced a novel approach for performing time-domain beamforming on linear sonar arrays using a MasPar SIMD architecture. The approach is based on mapping the beamforming problem into a set of operations on a 3 dimensional cube structure composed of all element × beam × time values needed for computing the beams in parallel. A MasPar MP-2 system was used to implement the beamforming algorithm using the built-in functions for array communication and arithmetic provided by the MasPar Fortran 90 language to achieve all the beamforming computations in parallel. Simulation results were provided for different element and beam combinations.

7. References

[1] A. B. Baggeroer, "Sonar Signal Processing," in *Applications of Digital Signal Processing*, A. V. Oppenheim, Ed., Prentice-Hall, Englewood Cliffs, NJ, 1978.

[2] W. R. Bernecky, "Linear Time-Domain Beamformer for a Spherical Array Using SIMD," *NUWC-NPT Technical Report*, 1995.

[3] B. Blesser and J. M. Kates, "Digital Processing in Audio Signals," in *Applications of Digital Signal Processing*, A. V. Oppenheim, Ed., Prentice-Hall, Englewood Cliffs, NJ, 1978.

[4] D. Foster, *Designing and Building Parallel Programs*, Addison-Wesely, New York, 1995.

[5] R. Kneipfer, "Beamforming - An Overview of Its History and Status," *NUWC-NPT Technical Report*, 1992.

[6] J. A. Nuttall, "Adaptive-Adaptive Narrowband Subarray Beamforming," *NUWC-NPT Technical Memorandum*, 1994.

[7] N. L. Owsley, "Sonar array processing," in *Array Signal Processing*, ch. 3, S. Haykin, Ed., Prentice-Hall, Englewood Cliffs, NJ, 1985.