Programmable Ultrasound Scan Conversion on a Mediaprocessor-based System

Siddhartha Sikdar¹, Ravi Managuli¹, Tsuyoshi Mitake², Tetsuya Hayashi² and Yongmin Kim¹

¹Image Computing Systems Laboratory
Departments of Electrical Engineering and Bioengineering
University of Washington, Seattle, WA 98195-2500

²Hitachi Medical Corporation
Kashiwa-shi, Chiba-Ken, Japan 277-0804

ABSTRACT

Scan conversion is an important ultrasonic processing stage that maps the acquired polar coordinate data to Cartesian coordinates for display. This requires computationally expensive square root and arctangent calculations for geometric transformation. Previously, we developed an algorithm for implementing scan conversion for gray-scale images using pre-computed lookup tables. In a clinical setting, however, interactive changes of scan conversion parameters, e.g., zoom and sector angle, require these tables to be recomputed often. In this paper, we describe a fast lookup table generation algorithm and its implementation on Hitachi/Equator’s MAP-CA mediaprocessor architecture. In addition, we have extended the gray-scale scan conversion algorithm for color images, which requires interpolation between angular data. For a 600x420 output image, gray-scale scan conversion takes 12 ms while color scan conversion takes 20.3 ms on a 300 MHz MAP-CA. Interactive parameter changes take 102.5 ms for table regeneration. We believe that this high performance is an important step towards making software-based ultrasound programmable systems using mediaprocessors a reality. Such a system would provide more flexibility and improved cost/performance in the future than the existing hardwired solutions.

Keywords: Programmable ultrasound systems; mediaprocessors; scan conversion; fast lookup table generation; fixed-point square root and arctangent computation; partitioned CORDIC; guided DMA.

1. INTRODUCTION

Medical ultrasound systems have many diagnostic modes, such as B, color, and spectral Doppler, and all these modes are routinely used in a clinical environment. Figure 1 shows the high-level block diagram of a typical ultrasound system. The transducer is used to steer the acoustic beam and receive the backscattered signals from the various layers of tissue. These signals are then rectified and combined in the beamformer. The RF demodulator recovers the reflected signal of our interest by removing the high frequency carrier via quadrature demodulation. The B-mode image highlights the tissue characteristics using gray scale and is formed by sending the vectors through the echo processor. The color-mode image is used to highlight blood flow and is formed by sending the color vectors through the color-flow processor. In this mode, a pseudo-color image is overlaid on the B-mode image, where the color code represents the velocity of the blood flow towards or away from the transducer. The scan converter is responsible for geometrically transforming the acquired polar coordinate data in both B and color modes to Cartesian raster data before displaying the output image on a standard monitor. Some processing is performed on the post scan-converted image before final display. The spectral Doppler mode is used for more accurately measuring the velocity of blood of a single point by analyzing the frequency content of the received echoes.

To support B and color modes in real time requires a huge amount of processing power. It has been estimated that 31 to 55 billion operations per second (BOPS) are needed to support the processing requirements in a high-end ultrasound machine depending on whether certain functions like square root and arctangent are implemented in lookup tables or calculated on the fly [1]. Traditional ultrasound systems so far have used special purpose hardware for meeting this large computational
requirement. However, the lack of programmability in dedicated hardware limits the scalability and flexibility of the system for new and evolving diagnostic applications, e.g., panoramic imaging and 3D visualization. Depending upon the application, ASICs, boards and even the complete system may need to be redesigned. In recent years, therefore, there has been an increasing level of interest in ultrasound systems with programmable components since they offer increased flexibility and multi-functionality without the need for hardware redesign [2][3][4].

The recent introduction of high performance mediaprocessors targeted for multimedia applications opened up unprecedented possibilities for the real-time implementation of ultrasound algorithms on programmable platforms that are flexible, scalable and cost effective [2]. Mediaprocessors have incorporated the latest advances in computer architecture and signal processing technology to deliver high computational throughput in a programmable framework by utilizing instruction-level and data-level parallelism. Even though mediaprocessor compilers have improved greatly in the last decade, careful mapping of the algorithm to the underlying processor architecture is essential in order to exploit the vast amount of available computational resources on a modern mediaprocessor and to achieve performance comparable to existing custom hardware-based solutions [5]. We have been investigating the optimal mapping of important ultrasound algorithms on modern mediaprocessors.

One of the key image processing algorithms in an ultrasound system is scan conversion [7]. This processing stage requires the computation of expensive square root and arctangent. The processing requirement of scan conversion alone was estimated to be about 13 billion operations per second (BOPS) when square root and arctangent operations are computed at run time and 4.65 BOPS when lookup tables are used for these operations [1]. With the introduction of mediaprocessors like TI TMS320C80 and MAP1000, our research demonstrated the feasibility of performing real-time B-mode scan conversion on a programmable platform [7]. In order to achieve high performance, expensive square root and arctangent computations were avoided by pre-computing the address transformation between the rectangular and polar coordinate systems. This allowed us to do real-time B-mode scan conversion as long as the geometric transformation parameters, such as zoom, sector angle, steering angle and region of interest (ROI), did not change. However, in a clinical environment, whenever the sonographer changes the parameters of the display, it results in a different polar to rectangular address transformation. Thus, the corresponding addresses between the rectangular and polar coordinate systems have to be recomputed. Since ultrasound is a real-time imaging modality, we need to ensure that the recomputation can be achieved as fast as possible. The computation of the address transformation currently takes about 3 seconds on a 450 MHz Pentium II processor. Because this implementation computes square root and arctangent in floating-point arithmetic, a significant speed-up cannot be achieved by a direct implementation on a fixed-point mediaprocessor. This latency between the time the display parameters are changed and the time they take effect is too long and would not be welcome by the clinicians. We believe that a maximum latency of ~200 ms would display output images with the new parameters almost instantaneously, thus would be more tolerable to the clinicians. We have developed a fast algorithm to compute the address transformation in fixed-point arithmetic. These transformations are stored in a table and then used by a DMA (direct memory access) engine for doing intelligent data loads and stores in parallel with the core computation engine. The address transformation tables can be reused over successive frames if the user does not change the scan conversion parameters.

The scan conversion process for color-mode images, while similar to that for B-mode images, has some unique requirements. In color mode, multiple vectors at each spatial location (known as ensembles) are collected and the velocity estimated by determining the average change in phase using an autocorrelation technique [6]. We have developed an interpolation scheme for angular data in order to obtain the correct result independent of the angle.

In this paper, we describe our mapping of scan conversion for both B-mode and color-mode images on a modern commercial mediaprocessor, the MAP-CA from Hitachi and Equator Technologies [8]. The rest of this paper is organized as follows: we first briefly introduce the MAP-CA architecture followed by a detailed discussion on the scan conversion process. We then describe our scan conversion implementation and compare it with previously proposed ideas. Finally, we present the performance of our algorithm and discuss trade-offs between accuracy and performance for scan conversion specific computations.
Figure 1. Block diagram of an ultrasound system.

2. THE MAP-CA MEDIAPROCESSOR ARCHITECTURE

Figure 2 shows the simplified block diagram of MAP-CA whose architecture is optimized for image and video processing [8]. The processing core consists of two clusters, a 32-kbyte 4-way set-associative data cache, and a 32-kbyte 2-way set-associative instruction cache. Each cluster has 64 32-bit general-purpose registers, 16 predicate registers, two pairs of 128-bit registers, an Integer Arithmetic Logic Unit (IALU), and an Integer Graphics Arithmetic Logic Unit (IGALU). The IALU can perform either a 32-bit fixed-point arithmetic operation or a 64-bit load/store operation while the IGALU can perform 64-bit partitioned arithmetic operations. Many IGALU instructions can specify different data partitions (each with 8, 16 or 32 bits), providing a rough performance improvement of 8×, 4× or 2× via data-level parallelism, respectively. The IGALU also supports very powerful instructions, e.g., inner_product, which in one instruction can perform multiple multiplications in parallel and accumulate the results into a 32-bit register. These two clusters are capable of executing four different instructions (e.g., two on IALUs and two on IGALUs) per clock cycle where instructions are as sophisticated as inner_product. The MAP-CA includes more than 700 instructions with many multimedia-optimized extensions, e.g., rounding, saturation, and shifting modes for fixed-point operations, which are very useful in fixed-point processing of multimedia data. The processor core currently runs at 300 MHz and the external memory is 133-MHz SDRAM. In addition to the processing core and its instruction set, the MAP-CA has a sophisticated DMA controller, called Data Streamer, which supports various data transfer modes.

Figure 2. Simplified block diagram of MAP-CA.
3. SCAN CONVERSION

The ultrasound transducer sends acoustic beams and receives the echoes along radial directions. In order to form a 2D image, the transducer sweeps the beam along angular directions within a sector. The echoes from the tissue are sampled along each radial direction corresponding to different depths in the tissue and these N samples are stored in memory as a vector corresponding to each angular direction. Figure 3 shows how the transducer acquires data and Figure 4 (a) shows how the acquired data are stored in memory. To display the acquired image with the correct geometric alignment, an image warping transformation needs to be performed. Analytically, this transformation corresponds to a polar coordinate to Cartesian coordinate transformation. The inverse mapping method is used for performing this transformation where for every output pixel, a corresponding address is calculated in the input image and input vector samples in the neighborhood of this address are fetched and interpolated to obtain the value of the output pixel.

A vector sample in memory is addressed by its angle $\psi$ from the vertical line (y axis) and its radial distance $r$ from the transducer. Using inverse mapping technique for every pixel of the output image, the corresponding vector sample address in the input image is calculated according to the following equations

$$r = \sqrt{x^2 + y^2}$$  \hspace{1cm} (1)

$$\psi = \arctan(x / y)$$  \hspace{1cm} (2)

The $\psi$ and $r$ are computed for all the active pixel locations in the output image. Furthermore, $\psi$ and $r$ are checked to see if they lie outside of the input image boundary for each output pixel.

The values of $r$ and $\psi$ computed above need not be integers whereas the addresses of vector samples are. Therefore, the inverse mapping process will give us an address location, which might fall in between the integral vector sample locations. Some interpolation scheme is required to compute the value of the output pixel from the neighboring input vector samples. To perform interpolation, the relative distances of the nearby input vector sample locations to the inverse-mapped address must be calculated. These fractional distances are then used in computing a weighted average of the input vector samples. Various interpolation functions and window sizes have been applied in ultrasonic scan conversion, such as linear interpolation, cubic spline, sinc, or Bessel functions [11].

![Figure 3. Ultrasound image scan conversion. Directions of acoustic vectors in a sector scan.](image-url)
Although the mediaprocessors have high processing power, a direct implementation on mediaprocessor of the scan conversion algorithms found in specialized boards and/or ASIC chips would not necessarily lead to real-time performance [11]. If scan conversion is to be implemented in real time on programmable processors, new algorithms need to be explored. We have developed such a new scan conversion algorithm for maximizing a mediaprocessor’s computational resource.

4. ALGORITHM MAPPING ONTO MAP-CA

We have divided the scan conversion process into several stages in order to optimize the performance on the MAP-CA mediaprocessor and meet the real-time frame rate requirements of a typical ultrasound machine. In order to save on computation of expensive square root and arctangent operations in the core loop for every frame, the input vector sample addresses are pre-computed for a single output frame in a LUT, which is subsequently utilized for successive frames. The following are the main processing stages in our scan conversion algorithm.

- Address generation
- DMA data transfer to fetch input vector samples in parallel with the computation
- Interpolation
  - B mode: linear interpolation
  - Color mode: angular interpolation
- DMA data transfer of processed pixels off-chip

Each of these stages is described in detail in the following sections.

4.1 Address Generation

The MAP-CA is a fixed-point mediaprocessor and does not possess fast floating-point computation capability. Therefore, for fast regeneration of address mappings that involve computation of square root and arctangent, an efficient fixed-point technique is needed while balancing the trade-off between accuracy and performance. The two commonly-used techniques for fixed-point square root and arctangent computation are iterative approximation and polynomial approximation. We first describe both these techniques and then discuss the accuracy-performance trade-off.
The Coordinate Rotation Digital Computer (CORDIC) [12] is an iterative algorithm to calculate the square root and arctangent in fixed-point arithmetic. It utilizes a successive approximations approach in which the vector corresponding to a point in the 2D plane is rotated in steps until it is aligned with the horizontal axis. The cumulative sum of all the rotation steps gives the angle of the vector with respect to the horizontal line, while the magnitude of the vector is given by the horizontal coordinate. Every iteration of the algorithm rotates the vector by $\arctan(1/2^i)$. With an increase in the number of iterations, the angle of rotation is smaller and hence more accuracy is obtained.

The following is a fragment of CORDIC pseudo code that illustrates the successive approximations process:

\[
\text{for } i = 0 \text{ to } N-1 \\
\quad dx = X / 2^i \\
\quad dy = Y / 2^i \\
\quad da = \arctan(1/2^i) \\
\quad \text{if } Y \geq 0 \text{ then} \\
\qquad X -= dy \\
\qquad A -= da \\
\qquad Y += dx \\
\quad \text{else} \\
\qquad X += dy \\
\qquad A += da \\
\qquad Y -= dx \\
\quad \text{endif} \\
\text{next}
\]

\[
\text{mag} = 0.607 \times X; \\
\text{phase} = A;
\]

The output of the above loop gives the magnitude and phase of the complex number $X+jY$. The number of iterations ($N$) determines the accuracy of the computation.

A direct implementation of this algorithm on a mediaprocessor is inefficient due to the following reasons:
1. The algorithm cannot utilize the instruction-level and data-level parallelism.
2. The conditional branching instructions have a heavy branching penalty for each pass through the iteration.
3. Because of branch conditions, both clusters cannot be utilized since the MAP-CA is a lock-stepped VLIW processor with a single program counter.

Since this successive approximation has to be carried out for every active pixel in the output display, it is very important to reduce the number of cycles it takes to complete each iteration. Thus, we modified the algorithm to exploit instruction-level and data-level parallelism on the MAP-CA. The MAP-CA instruction set provides logical comparison and "bit-select" instructions to help eliminate branching penalties. The number of operations in each path of the branch is small. Thus, by calculating both paths of the branch and using the bit-select instruction to select the correct result depending upon the value of a mask generated by a comparison operation, we can avoid branch instructions altogether. This helps us to avoid expensive branch penalties. At the same time, it can now take advantage of instruction-level parallelism by utilizing both clusters simultaneously.

We use the following notation from now on: a fixed-point $M$-bit $sQN$ number means that for this signed number $N$ bits out of $M$ are used for storing the fractional part, and $M-N-1$ bits are used for the integer part with one sign bit. The following snippet of code shows the iterations of the CORDIC algorithm for 32-bit $sQ16$ fixed-point representation:

\[
\text{for } i = 1; i < 16; i++ \\
\quad da.s32.lo = *(atan_LUT_sQ16+i); \\
\quad da.s32.hi = da.s32.lo; \\
\quad dy.s64 = \text{hmpv}_rs_ps32_re(y.s64,i); \quad \text{// dy} = y >> i \\
\quad dx.s64 = \text{hmpv}_rs_ps32_re(x.s64,i); \quad \text{// dx} = x >> i
\]
sig_y.s64 = hmpv_cge_ps32(y.s64,zero.s64);  \( y \geq 0? \)

a1.s64 = hmpv_add_ps32(a.s64,da.s64);        \( a1 = a+da \)

a2.s64 = hmpv_sub_ps32(a.s64,da.s64);  \( a2 = a-da \)

a.s64 = hmpv_bitslct_64(sig_y.s64,a1.s64,a2.s64);   \( a = (y\geq0)?a1:a2 \)

x1.s64 = hmpv_add_ps32(x.s64,dy.s64);   \( x1 = x+dy \)

x2.s64 = hmpv_sub_ps32(x.s64,dy.s64);  \( x2 = x-dy \)

x.s64 = hmpv_bitslct_64(sig_y.s64,x1.s64,x2.s64); \( x = (y\geq0)?x1:x2 \)

y1.s64 = hmpv_add_ps32(y.s64,dx.s64);  \( y1 = y+dx \)

y2.s64 = hmpv_sub_ps32(y.s64,dx.s64);  \( y2 = y-dx \)

y.s64 = hmpv_bitslct_64(sig_y.s64,y2.s64,y1.s64); //\( y=(y>0)y1:y2 \)

We have used the C language along with MAP-specific intrinsics, where the intrinsics are hints to the compiler on what instructions to use for performing a certain operation. For example, the hmpv_bitslct_64 intrinsic above will be converted by the compiler into a bitslct.64 assembly instruction during compilation. The use of intrinsics gives us direct access to the powerful instruction set and the specialized computational resources of mediaprocessors from a high level. Also, to make use of the data-level parallelism available on the MAP-CA mediaprocessor, we have used partitioned operations to calculate several addresses in parallel per iteration. The partition is indicated by the suffix on an intrinsic, e.g., hmpv_add_ps32 takes two 64-bit arguments partitioned into two signed 32-bit values and performs additions to generate two 32-bit results in a 64-bit register. Thus, we can perform two 32-bit signed additions with a single instruction. The suffix on the variables in the code above indicates the size, e.g., da.s64 is a 64-bit signed variable containing two 32-bit partitioned values, da.s32.lo and da.s32.hi. The use of intrinsics is essential to optimally utilize partitioned operations, as even modern compilers have difficulty in generating these partitioned operations from a high level language. The code shown above computes both branches of the conditional statement in the pseudo code and uses the bitslct_64 instruction to choose one of the two results based on the value of a mask. The mask itself is generated by the comparison instruction cge_ps32. A combination of the comparison and bit-selection instructions helps us avoid conditional branching instructions and overhead.

The smaller the number of bits required in representing an address, the more addresses can be computed in parallel. Since we use fixed-point arithmetic, the number of bits used in representing a fixed-point number is related to the desired accuracy. So, there is a trade-off between computation speeds and accuracy of computation. Inaccuracies in the computation are manifested as block artifacts in the displayed image.

The second method we studied is polynomial approximation. We estimate the square root, reciprocal and arctangent using polynomials. In order to calculate the square root and arctangent, we used a polynomial of the form \( a_0 + (a_1 - a_2 \times x) \times x \) where the coefficients \( a_0, a_1 \) and \( a_2 \) are chosen based on the most significant bits of \( x \) (most significant 3 bits for square root and most significant 4 bits for arctangent of a 32-bit sQ16 fixed-point number). This computation can be efficiently implemented using the inner_product instruction, which performs four multiplications and three additions in a single instruction.

The computation of the square root and arctangent is required for every active pixel location within the output image. However, the active pixel locations within the output image are not known a priori. Since we can compute the bounding lines of the sector using the scan conversion parameters, we can avoid the address computation for pixels that lie outside the sector. Using this information, we have reduced the number of computations to only those pixels within the active region, giving us a significant savings, especially when the sector is narrow and also for color mode where the region of interest might be only a small portion of the entire image.
4.2 Transfer of input vector samples to on-chip memory

The relationship between the input and output addresses in scan conversion is nonlinear. For example, the sequential output pixels \(a, b, c, d\) and \(e\) in Figure 5(a) correspond to the non-sequential input vector samples \(A, B, C, D\) and \(E\), respectively, in Figure 5(b). Non-sequential accesses to memory increase the memory access latency, causing the processor to halt and wait for the data to arrive. To minimize such processor stalls, we use the DMA controller (called Data Streamer) on the MAP-CA to fetch the relevant vector samples from off-chip to on-chip memory independent of the processor. The utilization of the DMA controller on the MAP-CA is similar to that of the Transfer Controller on the TI TMS320C80 as described in the previous implementation of scan conversion [7]. There are several operational modes in the Data Streamer. One of them is the guided DMA mode, which uses a LUT (guide table) of memory addresses to move data from one location of memory to another memory location. Figure 6 shows an example of the guided DMA mode of data transfer. The guide table contains the inverse-mapped addresses of the input vector samples (e.g., 0x300, 0x200, 0x100 etc). The Data Streamer uses the addresses in the guide table to fetch the input vector samples from noncontiguous locations (at offsets of 0x300, 0x200, 0x100 within the input image) in off-chip memory and stores them in contiguous locations in on-chip memory before the processor performs the interpolation. For each output pixel, neighboring input vector samples are fetched for interpolation (shown as the shaded areas in Figure 5(b) for 2x2 interpolation). The Data Streamer is made to work concurrently with the processor by utilizing the double-buffering scheme [7] where the processor performs interpolation on a current block of data that is present on-chip, while the Data Streamer fetches the next block of data from off-chip to on-chip memory. Since the data movements are performed concurrently with the computation, the processor stalls due to the memory access latency are significantly reduced.

![Guided DMA for prefetching into contiguous on-chip memory locations.](image-url)
4.3 Interpolation

To calculate the output pixel values, neighboring input vector samples are interpolated. Since the guided DMA brings only the input vector samples corresponding to active pixels, the interpolation will be calculated only for active pixels and not for pixels that lie outside the sector. Several interpolation schemes have been used previously [11]. The most popular interpolation schemes used for scan conversion are 2x2 bilinear interpolation and 4x2 interpolation (where 4 neighboring lateral vector samples and 2 neighboring axial vector samples are interpolated). 2x2 bilinear interpolation is illustrated in Figure 7 while 4x2 interpolation is shown in Figure 8. Using the inner_product instruction, we can compute the 2x2 interpolation with a single-cycle throughput on the MAP-CA.

\[ X = (1-x_{frac}) \cdot (1-y_{frac}) \cdot A + (1-x_{frac}) \cdot y_{frac} \cdot B + (1-y_{frac}) \cdot x_{frac} \cdot C + x_{frac} \cdot y_{frac} \cdot D \]  

(3)

In color mode, the velocity of the blood is obtained by measuring the phase shift between successive echoes reflected by the moving blood [6]. Since the phase data are angular, linear interpolation will give inaccurate results in certain cases, an example of which is shown in Figure 9 for two vectors. The correct value of the interpolation of two angles should be such that the resultant vector lies on the shortest arc between the phases. For correct interpolation, we need to compute the linear interpolation as a sum of the shortest arc distances between two angular vectors.

\[ i_{shortArc} = \pi - |\pi - \phi_i - \phi_{i+1}|. \]  

(4)

Nikolaidas and Pitas [13] proposed the following method to calculate the shortest arc distance between two vectors:
A direct implementation of this equation on the MAP-CA would require several operations. We efficiently perform this computation using fixed-point arithmetic taking advantage of the ability of 2’s complement arithmetic to overflow (or wrap around) the number range:

\[ \text{Arc}_{\text{short}} = \phi_i - \phi_{i+1} \]  \hspace{1cm} (5)

This method assumes that our measurements for \( \phi \) are scaled to the full dynamic range of our signed 2’s complement number (e.g., with 8-bit data, maximum red \( \pi = 0111 1111 \) and maximum blue \( -\pi = 1000 0000 \)). Then, if we compute the signed subtraction of two vectors, \( \phi_i - \phi_{i+1} \), ignoring the overflow flag and use the 8-bit result, we are guaranteed to have the shortest arc distance between the two vectors. This method produces a signed result where the sign bit indicates the relative direction of the arc distance. Therefore, we reduce the multiple absolute magnitude and subtraction operations in Eq. (4) to a single subtraction in Eq. (5).

To utilize the signed shortest arc distance, the interpolation computation is also modified. The color mode 2x2 interpolation has the form:

\[ \phi_{\text{out}} = L_1 \phi_1 + L_2 \phi_2 + L_3 \phi_3 + L_4 \phi_4 \]  \hspace{1cm} (6)

where the coefficients sum to one, i.e., \( L_1 + L_2 + L_3 + L_4 = 1 \). The above equation is manipulated into the following equivalent form for a 2x2 angular interpolation:

\[ \phi_{\text{out}} = \phi_1 + L_A (\phi_2 - \phi_1) + L_B (\phi_3 - \phi_2) + L_C (\phi_4 - \phi_3) \]  \hspace{1cm} (7)

where the new coefficients are computed from the old coefficients:

\[ \begin{align*}
L_C &= L_4 \\
L_B &= L_3 + L_4 \\
L_A &= L_2 + L_3 + L_4
\end{align*} \]

\( \phi_1 \) can be thought of as the basis vector and the other terms are the weighted shortest arc distances from this basis vector. Eq. (7) also uses the shortest arc distance math, ignoring the overflow and keeping the signed 16-bit result. Eq. (7) can be implemented efficiently on the MAP-CA using the \textit{shift\_pu8}, \textit{subtract\_ps8}, and \textit{inner\_product} instructions.

### 4.4 Transfer of processed data to off-chip memory

The interpolation is only performed for the active pixels and a group of M pixels are processed at a time for efficiency, where M is determined based on on-chip memory size. After interpolation, the processed output pixels have to be stored back to memory. Direct storing to memory will incur cache misses since the locations of the M active pixels may cross cache-line boundaries depending on the geometry of the active region. To minimize the processor stall cycles due to cache misses during storing the results off-chip, we use the Data Streamer to store the processed data concurrently with the processing in a fashion similar to what was described in Section 4.2. The active pixel locations in the output image are determined during the calculation of the inverse-mapped input addresses by checking the input image boundary conditions.

These active output address locations are run-length encoded in a LUT (i.e., the LUT contains a number of addresses and for each address, a variable number of pixels to be stored contiguously starting at that memory address). Figure 10 shows an example of the run-length encoding. If the active region is narrow, as in the top of the sector for group 1, multiple start addresses and run lengths are stored for each group of M pixels. If, on the other hand, the active image contains a continuous run of more than M pixels, the run is broken after M pixels and a new start address is stored as is the case for group N. We use the LUT to do DMA transfers as described before. However, since a variable number of pixels need to be stored for each address, a separate data transfer is performed for each starting address in the LUT where the size of the transfer is the number of continuous pixels corresponding to that starting address. This scheme allows us to prevent redundant stores for the background pixels and also reduce the processor stall cycles due to memory store latency.
5. RESULTS AND DISCUSSION

5.1 Scan Conversion

Since we use the inverse mapping technique, the computational performance of scan conversion is dependent upon the number of active pixels in the output image. This in turn depends upon the parameters, like sector angle, zoom and pan. On the MAP-CA processor, for an input image of 384x1024 and an output image of 600x420, we obtained the performance of 31.3 cycles per active pixel for B mode and 52.5 cycles per active pixel for color mode. For a sector angle of 90 degrees and zoom to fit the 600x420 output image (116,000 active pixels), these correspond to 12 ms for B mode and 20.3 ms for color mode @300 MHz. However, the core computation loop involving the interpolation took only 8.7 cycles per active pixel for B mode (3.3 ms for the image mentioned above) and 14.7 cycles per active pixel for color mode (5.6 ms). This indicates that the scan conversion performance is limited by the time taken to transfer data between off-chip and on-chip memories and not on the computing engine of the MAP-CA.

Using the frame rate requirements of a typical ultrasound system, we observed that the scan conversion takes up only 70% of the processing power of a single MAP-CA processor. Our performance of 12 ms for B-mode scan conversion (for 116,000 active pixels) is comparable to the previously-reported performance of 23.8 ms (~250,000 active pixels) on MAP-1000 [9] and 17.2 ms (~101,800 active pixels) on the TMS320C80 [10].

5.2 Address LUT Generation

Since we use an iterative technique for calculation of square root and arctangent operations, there is a trade-off between performance and accuracy. A smaller number of iterations would generate the LUTs in a shorter time, but the pixel addresses would be less accurate. The fixed-point representation of floating-point numbers is also critical to this issue of performance versus accuracy.

We index the pixels in the input image according to the vector and sample addresses as shown in Figure 11. The accuracy is measured in terms of the maximum error in the inverse-mapped vector or sample addresses in the input image, as compared to the corresponding values calculated using floating-point arithmetic. For 8 iterations of the CORDIC algorithm, we found that the inverse-mapped vector addresses could be off by at most one vector location or one sample location. However, 8 iterations of CORDIC require the addresses to be represented in 32-bit sQ8 because at least 9 bits are needed in the integer portion to be able to describe the dynamic range of addresses for our image size. This prevented us from using 16-bit...
partitioned operations. The total time taken for LUT generation in this case was 68.2 ms. To use 16-bit partitioned operations and have at least 9 bits for the integer portion, we could represent the numbers in 16-bit sQ6 by limiting the number of CORDIC iterations to 6. This introduced errors of up to two sample locations and up to six vector locations after appropriate scaling. This produced noticeable blocking artifacts in the image. Even though the performance was improved, we rejected this method because of errors and artifacts. For higher accuracy, we used 16 iterations of CORDIC with the numbers represented in 32-bit sQ16, which gave errors of less than one vector or sample location with minimal blocking artifacts. Using 16 iterations of the CORDIC, it took 102.5 ms. The polynomial approximation method using 32-bit sQ12 gave us slightly more error than the 16-bit CORDIC. At the same time, it was about 17% faster since it does not involve iterations. We have adopted the CORDIC algorithm with 16 iterations because it has the least amount of error among the alternatives we studied. The performance of the LUT generation algorithm using 16 iterations of CORDIC (102.5 ms) is well within the acceptable limit of ~200 ms. This gives us room to further improve the accuracy in the future if needed while still staying within the acceptable limit.

![Figure 11. Vector and sample addresses in the input image.](image_url)

6. CONCLUSION

Due to the high computational requirements, ultrasound systems have been implemented using special purpose algorithm-specific hardwired boards with limited programmability. Mediaprocessors can sustain the computational throughput required by ultrasound image processing algorithms. We have mapped scan conversion, which is a key ultrasound processing stage, on a modern mediaprocessor, MAP-CA. Because of our unique scan conversion algorithm mapping and the high performance architecture of MAP-CA, it requires less than a single programmable processor to perform scan conversion. In addition, scan conversion is I/O-bound (i.e., it takes more time to move data between on-chip and off-chip memories compared to the necessary computation). This is interesting because hardwired scan conversion boards thus far have been designed mainly to support huge computational requirements. On the programmable MAP-CA processor using our algorithm mapping techniques, the scan conversion computation is no longer a bottleneck, but data I/O is. This indicates that in the future we can perform more sophisticated interpolation for better quality images without any increase in the total execution time. Also, it means that future improvements in memory bandwidths are likely to impact the performance of scan conversion more than improvements in processor clock speed.

7. REFERENCES


