SAND--89-2985C

JUN 1 4 1990

DE90 012160

Implementation of the Phase Gradient Algorithm

Daniel E. Wahl
Paul H. Eichel and Charles V. Jakowatz, Jr.

Sandia National Laboratories
Albuquerque, New Mexico, 87185

ABSTRACT

The recently introduced Phase Gradient Autofocus (PGA) algorithm is a non-parametric autofocus technique which has been shown to be quite effective for phase correction of Synthetic Aperture Radar (SAR) imagery. This paper will show that this powerful algorithm can be executed at near real-time speeds and also be implemented in a relatively small piece of hardware. A brief review of the PGA will be presented along with an overview of some critical implementation considerations. In addition, a demonstration of the PGA algorithm running on a 7"x10" printed circuit board containing a TMS320C30 digital signal processing (DSP) chip will be given. With this system, using only the 20 range bins which contain the brightest points in the image, the algorithm can correct a badly degraded 256x256 image in as little as 3 seconds. Using all range bins, the algorithm can correct the image in 9 seconds.

1. INTRODUCTION

Synthetic Aperture Radar (SAR) is a widely used imaging technique in which a high degree of azimuth resolution is achieved by synthetically creating a large antenna aperture. As resolution requirements become more stringent, phase errors which are introduced by atmospheric turbulence or by platform motion must be extracted in order to produce a focused final image.

Several methods currently exist which automatically extract these phase errors and allow better focusing of SAR images. These methods however either require *a priori* knowledge of the phase error order such as with the map-drift[1,2] methods or assume a strong isolated point target is present in the scene.

The recently introduced Phase Gradient Autofocus (PGA) algorithm[3] is non-parametric and requires no assumptions about strong isolated point targets existing in the scene. The key advantage to the PGA algorithm is that it utilizes the fact that the phase error is redundant over all range bins. This enables the algorithm to be an optimal estimator of the phase derivative[4]and works well even with images that contain high clutter background.

The PGA algorithm is iterative in nature and compute intensive. These facts may discourage many system integrators from considering implementing the PGA algorithm into any real-time SAR image display system. Due to the rapidly expanding microprocessor and digital signal processor (DSP) market, the computer technology is available to implement the PGA algorithm in a real-time environment and on a relatively small piece of hardware.

This paper along with the associated hardware demonstrates that the PGA algorithm can be implemented in a small piece of hardware and can correct the phase errors of the SAR image on the order of real-time speeds (seconds). The PGA demonstration hardware utilizes a TMS320C30 DSP chip to calculate and correct the phase errors of a badly degraded - high clutter image.

The following sections are organized as follows: Section 2 gives a brief overview of the PGA algorithm while sections 3 and 4 illustrate how the software and hardware were implemented. Finally section 5 concludes with various performance results along

---

with some suggestions on how implement the algorithm running multiple processors.

2. OVERVIEW OF PGA

The PGA algorithm assumes that the phase degraded image is given in range-compressed format. This data is described by the equation:

$$f_n(t) = |f_n(t)|\exp\{j[\varphi_n(t) + \varphi_e(t)]\} \quad \text{(EQ 1)}$$

Where the subscript n refers to the n*th* range bin and t is the time across the synthetic aperture. Note that $|f_n(t)|$ and $\varphi_n(t)$ denote the magnitude and phase respectively. The phase error given by $\varphi_e(t)$ is common to all range bins at each aperture position. A brief outline of the algorithm implementation is given below:

1. Zero current cumulative phase error array $\varphi_c(t)$.
2. Load in the range compressed image into a local array.
3. Apply $\varphi_c(t)$ to the image.
4. Form fully compressed image by doing 1-dimensional inverse FFT's along the azimuth direction.
5. Find the brightest point in each range bin from the fully compressed image.
6. Circularly shift each range bin in the fully compressed image such that the brightest point is in the center.
7. Select a symmetric window about the origin to capture support of the degraded point spread function (PSF).
8. Calculate the modified range compressed image by doing 1-dimensional FFT's along the azimuth direction.
9. Denoting the modified range compressed image by $g_n(t)$ where the subscript n denotes the n*th* range bin, determine the phase derivative along the azimuth direction by the equation:

$$\dot{\varphi}(t) = \frac{\Sigma_n \mathrm{Im}[g_n^*(t)\dot{g}_n(t)]}{\Sigma_n |g_n(t)|^2} \quad \text{(EQ 2)}$$

10. Next, integrate the phase derivative

$$\varphi(t) = \sum_{k=0}^{t} \dot{\varphi}(k) \quad \text{(EQ 3)}$$

and add to the cumulative phase error

$$\varphi_c(t) = \sum_{k=0}^{N-1} \varphi(k) \quad \text{(EQ 4)}$$

11. The RMS phase error calculated at the current iteration is given by:

$$\chi = \frac{\sqrt{\sum_{k=0}^{N-1} |\varphi(k)|^2}}{N} \quad \text{(EQ 5)}$$

12. If $\chi > \max_{rms}$ go to 2
    If $\chi < \max_{rms}$ stop.

Where $\max_{rms}$ is a predefined maximum allowable RMS phase error across the synthetic aperture.

A modification of this algorithm is a mode where only *k* range bins are used to calculate the phase corruption instead of using all range bins. The *k* range bins are selected if they contain one of the *k* brightest points in the image. With this implementation *k* was selected to be 20.

3. SOFTWARE IMPLEMENTATION

The steps for the software algorithm implementation is given in section 2. A block diagram illustrating the iterative sequence is shown in Figure 1.

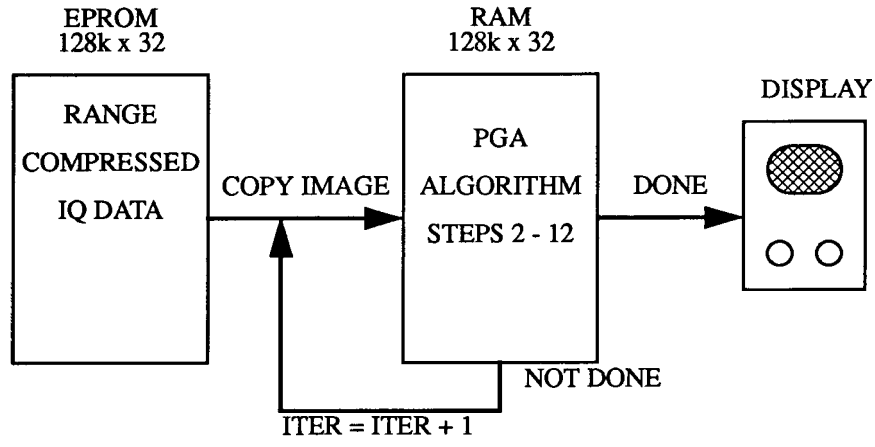

Figure 1. Illustration of software implementation in PGA demonstration hardware

With this implementation, the range compressed phase corrupted image is stored in EPROM. At each iteration, this image is copied to local RAM and the current cumulative phase correction is applied. The image is then fully compressed and the PGA algorithm is applied to calculate a new incremental phase error. This phase error is then added to the cumulative phase error which is utilized in the next iteration. Upon completion, the cumulative phase error $\varphi_e(t)$ is the calculated phase error for the image across the aperture.

The code was written in C and TMS320C30 assembly language. The source code consists of 956 lines of C and 904 lines of assembly code. The resultant compiled code is 3k. The C code comprises 2k of the compiled code while the assembly portion makes up the remaining 1k. The assembly portion of the code includes subroutines which the algorithm utilized extensively. These include FFT routines, copy routines and video display routines.

The amount of general purpose memory available for intermediate buffers and scratch-pad memory was the on-chip 2k RAM located on the TMS320C30. This amount of memory proved to be sufficient for a 256x256 SAR image. This memory was divided into two 512 size buffers, three 256 size buffers and the remaining area was used by the stack and local variable storage.

The amount of execution time spent in each phase of the algorithm in relation to the total execution time per iteration is shown in table 1. Table 1 also indicates how much of the code is written in assembly for each particular task. The phases of the algorithm were subdivided into 5 portions. The *copy* portion equates to step 2 in section 2. The *correct* portion equates to step 3. The *azimuth compress* portion equates to step 4. The *rotate* portion equates to steps 5 - 7. The *phase* portion which actually

calculates the phase error includes steps 8 - 12.

FUNCTION	% OF TOTAL FLOPS	% ASSEMBLY CODE
COPY	2.4%	100%
CORRECT	9.8%	0%
AZ COMPRESS	18.5%	74%
ROTATE	12.3%	60%
PHASE	57%	60%

Table 1. Summary of PGA execution time

With this implementation, it is difficult to determine the amount of time the algorithm requires at each iteration. As the algorithm progresses the size of the window selected to capture the PSF at each iteration decreases. This window size then dictates the size of FFT's to use throughout the rest of that particular iteration. A smaller FFT size will decrease the amount of time it requires to execute the PGA algorithm for that iteration.

4. HARDWARE IMPLEMENTATION

A TMS320C30 DSP chip was utilized in the PGA demonstration hardware to phase correct a 256x256 size complex image. This chip can execute 33.3 MFLOPS with a 60-ns single-cycle instruction execution time. The block diagram of the system is shown in Figure 2.

A total of 3 types of memory exist on the board. Bank 0 contains 32k x32 bits of EPROM space for program code. Banks 2 and 3 have EPROM space to store the range compressed complex data of the uncorrected demonstration image. These EPROMS are 128k x 32 bits which will store a complex image of the size 256x256 pixels. Banks 4 and 5 provide the RAM for most of the on-board memory accesses. These RAMS are also 128k x 32 bits.

The TMS320C30 has two buses that operate independently. This design utilizes the first bus to output data to a video DAC which will drive a video monitor. The monitor is used to display the uncorrected and corrected image. The second bus is used to interface to all other memories and I/O devices.

At each iteration, the algorithm will copy the range compressed data from the EPROMS in banks 2 and 3 into the RAM in banks 4 and 5. The rest of the time is spent accessing the on-board rams in banks 4 and 5 or the internal memory to the DSP chip (this comprises steps 2-10 in section 2). The access time for the program code in bank 0 is not critical since the TMS320C30 contains a program cache which minimizes the effect of slow external program memory.

In order to take full advantage of the power of the TMS320C30 it is necessary to acquire memory devices with at least 25ns access time. This will allow the TMS320C30 to avoid wait states. Given the above description it is easily seen that the RAMS in bank 4 and 5 be fast enough to result in 0-wait state access time from the DSP chip. The impact of slower memory for banks 0, 2 and 3 is not as significant with this implementation. The final configuration for this board resulted in 0-wait state memory for banks 4-5, 1-wait state memory for bank 0, and 2-wait states for banks 2-3.

It was decided to use a video DAC to satisfy the image display requirement. This solution required only 2 additional chips on board (video DAC and latch). The disadvantage to this solution is that the TMS320C30 is not available for calculations while it is displaying an image on the video monitor. Thus, the degraded SAR image is displayed before the PGA algorithm is applied, then once the image is corrected, the corrected image is displayed.

The resulting design was implemented on a 4-layer pc board, The size of the board is 7" x 10". The total IC chip count was 30

including the TMS320C30. The maximum power drawn is 15 watts.

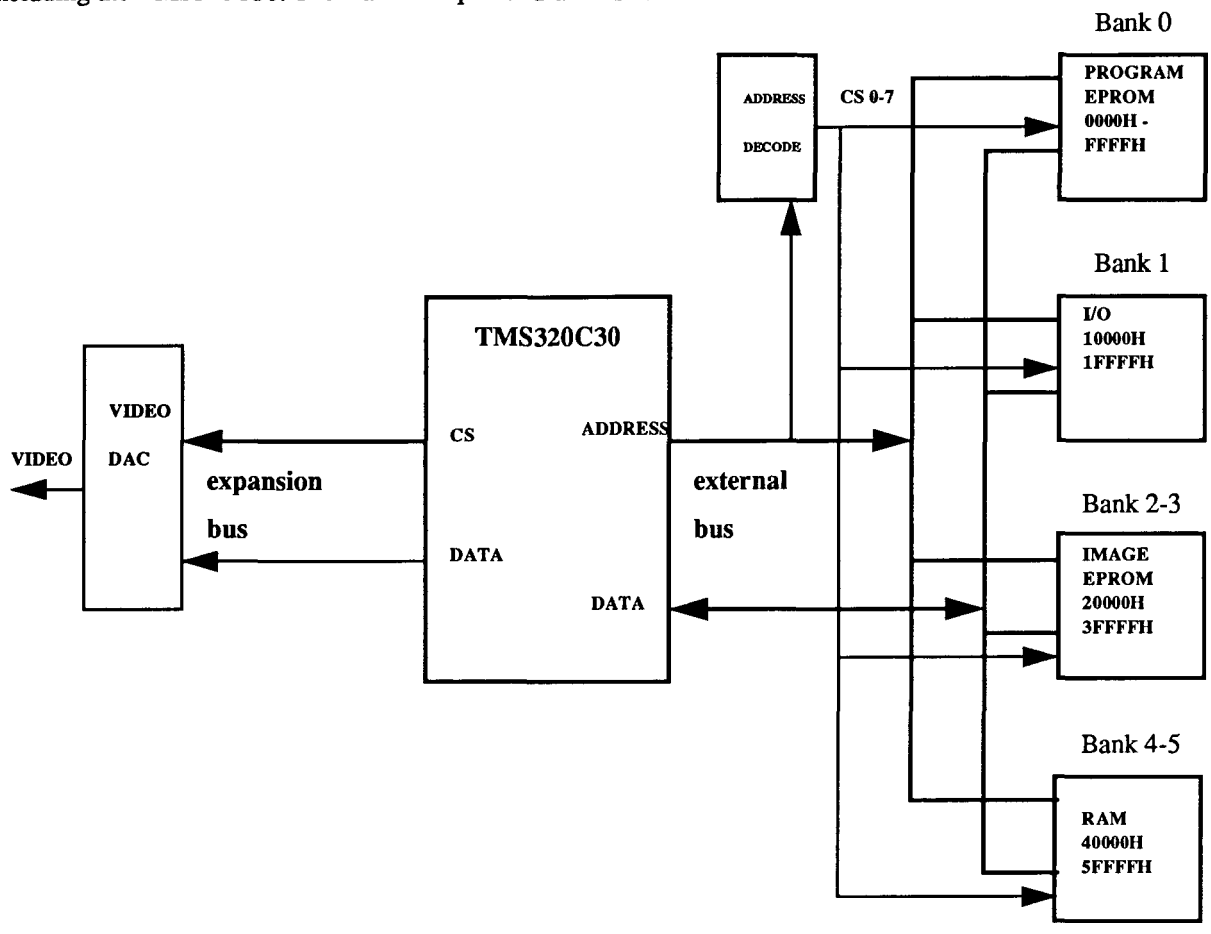

Figure 2. Block diagram of PGA hardware implementation

5. DISCUSSION AND CONCLUSION

The primary performance characteristics of the PGA hardware and software configuration described above are:

1. The size of the final printed circuit board is 7" x 10".
2. The total IC chip count is 30.
3. Total maximum power drawn is 15 watts.
4. Execution performance on a 256x 256 image:
   - 4 iterations with an RMS error of 0.223
   - 9 seconds for all range bins
   - 3 seconds for the 20 brightest point mode

The corrected image used for this hardware demonstration resulted in no noticeable difference between the 20 brightest point

mode and the mode that used all range bins. Based on this observation, a real-time implementation may incorporate only the *n* brightest point mode which would result in a well focused image with much less execution time. Note, there is no optimum value for *n*. The best number for *n* is dependant on the number of bright targets and the amount of background clutter present in the image. This value is usually determined heuristically.

The 3 second execution time is dependant on the size and the amount of clutter contained in the image that is being corrected. The image used here contains a high degree of background clutter which will increase the number of iterations required to correct the image. Typically, most images do not require more than 3 iterations to correct.

In order to achieve faster execution times, the PGA algorithm can easily be implemented in a parallel architecture. This is accomplished by subdividing the number of range bins of the image into portions. The number of range bin portions is equivalent to the number of processors available. If the *n* brightest point mode were used, each processor would execute on a submultiple of the *n* range bins. Steps 3-9 of the algorithm given in section 2 could then execute in parallel. Each processor would execute these steps on the range bins that it was assigned to. Each step would be processed in parallel, but the next step would not commence until each processor completes the current step.

Note that when implementing the algorithm in parallel, the outline given in section 2 cannot be applied directly. Some algorithm steps cannot be completely implemented in parallel but may be processed in a pseudo-parallel nature. For example, in step 9, each processor may form the numerator and denominator in eq. 2. At completion, the results of each process may then be combined to form an estimate of the phase derivative as shown in equations 6-8.

$$\dot{\varphi}(t) = \frac{\Sigma_p [\Phi_p(t)]}{\Sigma_p [\Gamma_p(t)]} \qquad \text{(EQ 6)}$$

$$\Phi_p(t) = \Sigma_{np} \mathrm{Im}[g^*_{np}(t)\, g_{np}(t)] \qquad \text{(EQ 7)}$$

$$\Gamma_p(t) = \Sigma_{np} |g_{np}(t)|^2 \qquad \text{(EQ 8)}$$

where *p* is the processor and *np* indicates the range bins assigned to processor *p*.

6. REFERENCES

1. W.D. Brown and D.C. Ghiglia, J. Opt. Soc. Am. A 5, 924 (1988)

2. D.C. Ghiglia and W. D. Brown, J. Opt. Coc. Am. A 5, 942 (1988)

3. P. H. Eichel, D.C. Ghiglia, and C.V. Jakowatz, Jr., "Speckle processing method for synthetic-aperture-radar phase correction", in *Optics Letters*, volume 14, number 1, January 1, 1989.

4. P. H. Eichel and C. V. Jakowatz, Jr., "Phase-gradient algorithm as an Optimal estimator of the phase derivative", in *Optics Letters*, volume 14, number 20, October 15, 1989