Optimal Configuration of an Embedded Parallel System for Synthetic Aperture Radar Processing *

Jeffrey T. Muehring and John K. Antonio
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409-3104
{jmuehrin, antonio}@cs.ttu.edu

Abstract–The creation of a synthetic aperture radar (SAR) image involves processing radar return signals in real-time using a computing platform on board the aircraft that houses the SAR system. In such environments, it is important to minimize the total power consumption of all onboard systems. This is especially true for applications that utilize small unmanned aircraft or satellites. In this paper, a mathematical optimization technique is formulated – based on nonlinear programming – for determining the optimal (i.e., minimal consumed power) configuration of an onboard parallel computing platform for SAR processing. The target hardware for this study is a Mercury Race System that is assumed to be configurable using a combination of two types of daughtercards: one type has six processors and a total of 32MB of memory; the other type has two processors and a total of 64MB of memory.

1 Introduction

Because radar is a ranging instrument, the resolution associated with a single radar return depends on the width of the transmitted pulse; the shorter the pulse, the higher the resolution. However, generating short radar pulses requires high power [2]. In many applications where very high-resolution radar images are desired, there are hard constraints on the allowable size, weight, and power of the radar system (e.g., satellites and unmanned aircraft). Thus, radar systems that can generate extremely narrow pulses are not feasible in such applications because of their associated large size and/or high power requirements.

Synthetic aperture radar (SAR) is a processing technique for achieving high-resolution images from relatively small and low-power radar systems. Specifically, SAR involves the processing of multiple low-resolution radar returns to emulate a high-resolution return. Typical applications for SAR include ground surveillance and terrain mapping. Advantages of using SAR instead of optical imaging techniques include radar's immunity to weather and lighting conditions. Image resolutions for typical SAR applications can range from 50 m down to 0.5 m [3]. Due to space limitations, detailed background information on the theoretical foundations of SAR processing is not included here; however, there are numerous excellent books on the topic (e.g., see [2]).

In addition to the size and power associated with the radar equipment itself, the size and power of the computing platform used to perform the SAR processing can also become significant. Minimizing the power of the computing platform used for SAR processing, for a given radar system, is the focus of this paper.

SAR processing can be parallelized and performed on an embedded parallel computing platform. As a first step toward deciding how to configure such a computing platform, the aggregate required processing throughput associated with a given set of system parameters can be derived (see [3] for details). However, the throughput requirement alone does not uniquely specify how to configure the embedded computer. As described in more detail in Section 2, the computational strategy assumed here involves using the technique of sectioned fast convolution [5]. The choice of the "section size" used in this technique dictates the relative efficiency of the processors used and the amount of memory required. In general, a large section size implies better computational efficiency at the expense of requiring more memory. To further complicate the issue, there are practical constraints on how the embedded computer can be configured. Specifically, the number of processors and amount of memory for a configuration must be realizable by using combinations of different types of daughtercards. In this paper, two types of daughtercards are assumed to be available: one that has six processors and a total

*This work was supported by Rome Laboratory under grant number F30602-96-1-0098.

of 32MB of memory and one that has two processors and a total of 64MB of memory.

The proposed formulation involves the derivation of a parameterized objective function that defines the power consumption of the embedded computer. This objective function depends on radar-dependent parameters, application-dependent parameters, processor-dependent parameters, a software-dependent parameter, and configuration-dependent parameters (i.e., the number of daughtercards of each type). For a fixed set of radar-, application-, and processor-dependent parameters, values of the software- and configuration-dependent parameters are determined that minimize the derived objective function (i.e., the consumed power of the embedded computer).

The rest of the paper is organized in the following manner. In Section 2, an overview of the basic computational strategy is provided and mathematical relationships among the underlying parameters are derived. Based on these mathematical relationships, the proposed optimization problem is formulated in Section 3. A solution technique for the proposed optimization problem and numerical studies are included in Section 4 to illustrate the utility of the proposed approach.

2 Computational Framework

The basic computational framework assumed here is the same as that described in [3]. The description given here is an overview; for more details refer to [3].

Processors are divided into range and azimuth processors. That is, every processor is dedicated exclusively to the processing of data either in the range or azimuth direction. The range direction is perpendicular to the line of flight and the azimuth direction is parallel to the line of flight.

After radar returns have been sampled and converted to digital signals, samples are typically read into memory at a rate of 5-50 Msamples/s [3]. By visualizing memory as a 2-dimensional grid, a row of memory contains the returns from a single radar pulse, whereas a column contains returns of different pulses from the same range. Memory is therefore sequentially filled a row at a time. When a sufficient number of rows have been filled, this data is sent to a range processor. These blocks of data are sent to the range processors in a round-robin fashion. After a number of range processors have processed data, the conglomerate block of data is "corner-turned," or matrix-transposed, and then sent to the azimuth processors. Note that the number of range and azimuth processors need not be the same. The matrix transposition of the data dictates that the azimuth processors receive the range-processed rows as columns and the unprocessed columns of the azimuth direction as rows.

Processing of the samples in the range direction primarily involves convolving the data with a reference kernel. The most efficient method of performing this convolution is with the use of FFTs, which is known as a fast convolution [5, 7]. It is assumed that the entire vector of range samples for a given pulse return is processed as a single section of data.

The azimuth processors perform similar operations on the data as the range processors (i.e., fast convolution) but with one important difference: the length of the data stream in the azimuth direction is indefinite whereas in the range direction it is of a fixed length. Therefore the data cannot be convolved as a single entity in the azimuth dimension. Sectioned fast convolution [5] provides a method for processing data streams of indefinite length. For such a data stream, the data is divided into sections of arbitrary length. A section is then convolved with the prestored kernel as in the case of a regular fast convolution. However, overlapping the sections by an amount equal to the kernel size and performing fast convolutions on each overlapped section yields the same result as if the entire data stream were convolved at once. But there is a price to be paid in computational efficiency for using this method. A portion (of length equal to the kernel size) of each convolution resultant must be discarded. Therefore, computational efficiency decreases as the ratio of the section of new data to the kernel size decreases.

Besides memory, another limiting factor to the size of the new data to be convolved is the $O(N \lg N)$ time complexity of the standard FFT algorithm. An important objective is to balance computational efficiency with memory requirements. For instance, selecting a section size that maximizes computational efficiency alone, without regard for concomitant memory requirements, may be unfavorable due to high power consumption of the required memory. Accounting for this tradeoff is an important aspect of the model presented in this section.

A fast convolution consists of an N-point FFT, N complex multiply operations, and an N-point inverse-FFT, where N is the number of data points to be processed, including any overlap. The complexity of this computational load is therefore $L = O(N \lg N + N)$. The exact number of floating point operations generally depends on processor- and implementation-specific details. For the purposes of this paper, SHARC processors are assumed, for which the exact

number of floating point operations is given by [3]:

$$L = 10N \lg N + 6N.$$

The computational load per sample is obtained by dividing L by the number of new data points processed, which reflects the efficiency of the calculation. For range processing this load per sample, ϕ_r, due to the fast convolution is given by

$$\phi_r = \frac{10F_r \lg F_r + 6F_r}{S_r},$$

where F_r is the FFT size for the range and S_r is the number of points in the range to be processed. These two values can differ because of the stipulation in the FFT algorithm that requires the FFT size to be a power of two (i.e., $F_r = 2^k$). Although this implies some inefficiency, it is usually still faster than using a direct convolution algorithm based on the exact sequence length.

The number of range points S_r is equal to the range swath R_s divided by the desired resolution δ (this is an intuitive result based on the physical interpretations of R_s and δ). Using this expression, the equation for ϕ_r becomes

$$\phi_r = \frac{\delta F_r(6 + 10 \lg F_r)}{R_s}.$$

Similarly, the azimuth processing load per sample due to the fast convolution is given by

$$\phi_a = \frac{F_a(6 + 10 \lg F_a)}{S_a},$$

where F_a is the azimuth FFT size and S_a is the section length. It should be noted that for both range and azimuth processing, the reference kernels are prestored and dependent only upon physical parameters of the system.

To compute the number of processors required for both range and azimuth processing, the total computational load must be computed. The fast convolution comprises the majority of the load. However, several other operations are also involved, including fix-to-float conversion, complex signal formation, motion compensation, magnituding, and the matrix transpose already mentioned [3]. It is important to realize that different operations can take different amounts of time, even if they are considered to be a "single floating point operation." Therefore, calculating the total computational load requirement per data sample involves dividing the number of real operations per sample of each type by their respective tested throughputs for a given type of processor. This value multiplied by the sample rate yields the total number of processors required.

Range and azimuth processing have unique load requirements in addition to the fast convolution load and are noted by the constants α_r and α_a, respectively. The required number of range processors is then defined by

$$P_r = Q(\alpha_r + \frac{\phi_r}{\gamma}), \tag{1}$$

where Q is the sample rate and γ is the throughput in Mflops for a fast convolution based on the assumed processor type used. Similarly, the number of azimuth processors required is given by

$$P_a = Q(\alpha_a + \frac{\phi_a}{\gamma}). \tag{2}$$

It can be shown that the sample rate is determined by the following equation [3]:

$$Q = \frac{vR_s}{\delta^2},$$

where v is the velocity of the platform. If this expression is substituted for Q and the expressions for ϕ_r and ϕ_a are also applied, then Eqs. (1) and (2) become

$$P_r = \frac{v(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^2} \tag{3}$$

$$P_a = \frac{vR_s(\alpha_a + \frac{F_a(6+10 \lg F_a)}{\gamma S_a})}{\delta^2}. \tag{4}$$

The total memory required for range processing is a product of the number of range processors, P_r, and the number of range samples, S_r. This value represents the number of complex range samples that are stored in memory at a given instant, each complex sample consisting of 16 bytes. Therefore the total range memory required is

$$M_r = 16P_rS_r, \tag{5}$$

or equivalently,

$$M_r = \frac{16R_sv(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^3}. \tag{6}$$

Azimuth memory needs dominate total system memory, requiring a double-buffer (for the matrix transpose operation) and an output image buffer, both of size $S_r(S_a + K_a)$, where K_a denotes the length of the azimuth reference kernel. The double-buffer must store complex values; the output image buffer stores reals. The total azimuth memory requirement in bytes is expressed as

$$M_a = 10S_r(S_a + K_a). \tag{7}$$

The value of K_a can be expressed in terms of basic parameters of the radar. Let λ be the wavelength of the radar. The value for K_a is derived in [3] to be:

$$K_a = \frac{\lambda R}{2\delta^2}.$$

Substituting this expression and $S_r = R_s/\delta$ into Eq. (7) yields

$$M_a = \frac{R_s(\lambda R + 2\delta^2 S_a)}{\delta^3}. \quad (8)$$

3 Formulation of an Optimal Configuration Problem

The final equations derived above for P_r, P_a, M_r, and M_a, given by Eqs. (3), (4), (6), and (8), depend on many different types of basic system parameters. These basic parameters can be divided into four major categories:

- radar-dependent parameters: R (range), R_s (range swath), and λ (wavelength);
- application-dependent parameters: δ (desired resolution) and v (platform velocity);
- processor-dependent parameters: α_r, α_a, and γ; and
- software-dependent parameter: S_a.

From Eqs. (3), (4), (6), and (8), it appears that there is also a dependence on the parameters F_r (range FFT size) and F_a (azimuth FFT size). However, recall that F_r and F_a are functions of S_r and $S_a + K_a$, respectively, and S_r and K_a can both be expressed in terms of basic radar- and application-dependent parameters.

For the purposes of this section, denote the total processor requirement $(P_r + P_a)$ and the total memory requirement $(M_r + M_a)$ as P and M. To formulate an optimal configuration problem, it is assumed that all radar-, application-, and processor-dependent parameters are specified, and S_a is to be determined. To emphasize this dependence solely on the parameter S_a, P and M are denoted by $P(S_a)$ and $M(S_a)$. The question that naturally arises is how to optimally choose the value of S_a? More fundamentally, how does the value of S_a affect the resulting configuration of the computing platform and its value of consumed power? Recall that the desired objective is to minimize the total power consumption of the computing platform.

A possible (yet unrealistic) approach would be to model consumed power of the computing platform as

$$\kappa P(S_a) + \beta M(S_a),$$

where κ and β are constants that represent power requirements on a per processor and per byte of memory basis, respectively. Determining a value of S_a, say S_a^*, which minimizes this function could be used to define an optimal configuration – i.e., a configuration that has $P(S_a^*)$ processors and $M(S_a^*)$ bytes of memory.

Modeling total consumed power as described above is unrealistic because it allows configurations to have arbitrary numbers of processors and amounts of memory. This would require, in general, that such a configuration be realized at the chip-level, i.e., customized boards may have to be developed to support the derived optimal configurations.

In reality, it is more practical to constrain the set of configurations to those that are realizable using commercially available boards that contain differing numbers of processors and amounts of memory. For this study, the computing platform is assumed to be based on a Mercury Race System that is configurable using a combination of two possible types of daughtercards: (1) the S2T16B, which has a total of six SHARC processors and 32MB of memory and (2) the S1D64B, which has a total of two SHARC processors and 64MB of memory. Each of these card types has a corresponding maximum power consumption rating: the type 1 card is rated at 12.2 watts and the type 2 card is rated at 9.6 watts [6]. Under this framework, the total power consumption is modeled based on the number of cards of each type utilized.

Let C_1 and C_2 denote the number of type 1 and type 2 cards utilized, respectively. Thus, the function for total consumed power, denoted as W, is defined as

$$W = 12.2C_1 + 9.6C_2. \quad (9)$$

Next, two required constraint equations naturally follow based on the values of $P(S_a)$ and $M(S_a)$:

$$6C_1 + 2C_2 \geq P(S_a) \quad (10)$$

$$32C_1 + 64C_2 \geq M(S_a). \quad (11)$$

These constraint equations insure that the total number of processors in the configuration is no less than the total number of required processors and the total amount of memory in the configuration is no less than the total amount of memory required. In this framework, values for the parameters C_1 and C_2 must be optimized (in addition to the value of the parameter S_a). Although the parameter S_a does not explicitly appear in the objective function that is to be minimized, i.e., W, its effect is implicit through the constraint equations.

To summarize, the proposed optimization problem is stated as follows: find nonnegative integer values

for C_1, C_2, and S_a such that W is minimized and constraint Eqs. (10) and (11) are satisfied.

4 Solving the Optimal Configuration Problem

4.1 *Proposed Solution Technique*

As formulated, the proposed optimization problem can be classified as an integer programming problem. Solving such optimization problems can be computationally intensive (see [4] for a summary on integer programming techniques).

Instead of directly applying an integer programming technique, an alternative approach is proposed here for solving the formulated optimization problem. Notice that the objective and the constraint equations are *nearly* continuous functions of the optimization variables C_1, C_2, and S_a. If the objective and constraints were continuous, then nonlinear programming techniques (e.g., see [1]) could be applied. Such approaches often have fast convergence properties. The only discontinuous portion in the formulation is due to the definition of F_a, which is a discontinuous function of S_a. (Recall that F_a is defined as the smallest integer power of two that is greater than $S_a + K_a$.) This discontinuous function prevents the direct application of nonlinear programming. However, by selecting F_a as an integer power of two, and adding a constraint to ensure that $K_a + S_a$ is no greater than this selected value, the discontinuity can be removed. Thus, in addition to the constraints given by Eqs. (10) and (11), the following constraint equation is added

$$K_a + S_a \leq F_a, \tag{12}$$

where the value of $F_a = 2^k \geq K_a$ is fixed (the value of K_a is known based on the values of the specified basic parameters). Thus, to ensure optimality, it may be necessary to solve several constrained optimizations based on different feasible values for F_a. In practice, however, only a few values for F_a need to be tried: from the smallest feasible value up to the point at which the optimal value of S_a is such that $K_a + S_a < F_a$ (i.e., the constraint becomes inactive).

4.2 *Numerical Studies*

The solution technique proposed in the previous subsection is applied to find optimal configurations based on four different sets of application-dependent parameters: (1) $\delta = 1$, $v = 300$; (2) $\delta = 1$, $v = 200$; (3) $\delta = 1.5$, $v = 300$; and (4) $\delta = 1.5$, $v = 200$ (the units for δ and v are meters and meters/s, respectively). For all four cases considered, the radar-dependent parameters and processor-dependent parameters were fixed at the following values: $R = 10^5$, $R_s = 2 \times 10^4$, $\lambda = 0.03$, $\alpha_r = 0.3528$, $\alpha_a = 0.9068$, and $\gamma = 94$. These values are derived in [3] based on a Mercury Race System configured using SPARC processors.

Intuitively, case 1 represents the most computationally demanding scenario of the four cases considered – it has the largest platform velocity and the finest desired SAR resolution. Case 4, on the other hand, represents the other end of the spectrum – it is the scenario with the smallest velocity and coarsest resolution. Thus, it would be expected that case 1 have the highest power consumption requirement and case 4 the lowest – this intuition is confirmed in the numerical studies described next.

The formulated optimization problem was solved using a routine from the Optimization Toolbox of MATLAB called `constr`. This routine was executed interactively (in MATLAB's command line mode) on a Sun SparcStation, and the response time for solving each optimization was almost immediate (less that one second).

To illustrate the advantage associated with allowing configurations to have two types of cards (i.e., heterogeneous configurations), optimizations were also conducted in which only one card type is allowed (i.e., homogeneous configurations). Mathematically, finding an optimal homogeneous configuration corresponds to setting the value of either C_1 or C_2 to zero and solving the resulting optimization. Tables 1, 2, and 3 summarize the results of the numerical studies that were conducted. Table 1 shows the results for the optimal heterogeneous configurations, in which both types of cards are allowed (i.e., solving the optimization as described in the previous subsection). Tables 2 and 3 show the results of optimal homogeneous configurations, in which C_2 and C_1, respectively, were defined to be zero in the formulation. In all tables, the optimal values of S_a, C_1, C_2, and F_a as well as the corresponding optimal value of the consumed power are tabulated for each of the four cases considered.

Notice that optimal values of S_a and F_a given in Table 2 are substantially less than those in Table 3. This is logical considering that the memory to processor ratio for the type 2 card is much higher than that for the type 1 card, and memory requirements grow linearly with the value of S_a (refer to Eq. (7)). In reality, of course, a fractional number of cards cannot be installed in an actual configuration. Thus, the values for C_1 and C_2 would need to be rounded up to the nearest integers so that the processor and memory constraints are satisfied.

Table 1: Optimal Heterogeneous Configurations: Type 1 and 2 Cards

case no. ($\delta : v$)	S_a	C_1	C_2	F_a	Power (in watts)
1 (1 : 300)	548	4.8	4.2	2048	94.7
2 (1 : 200)	548	2.1	5.3	2048	77.3
3 (1.5 : 300)	357	1.5	1.4	1024	31.9
4 (1.5 : 200)	357	0.7	1.8	1024	26.0

Table 2: Optimal Homogeneous Configurations: Type 1 Cards Only

case no. ($\delta : v$)	S_a	C_1	C_2	F_a	Power (in watts)
1 (1 : 300)	259	11.0	0	2048	134.4
2 (1 : 200)	175	10.5	0	2048	127.9
3 (1.5 : 300)	174	3.5	0	1024	42.8
4 (1.5 : 200)	118	3.3	0	1024	39.9

Table 3: Optimal Homogeneous Configurations: Type 2 Cards Only

case no. ($\delta : v$)	S_a	C_1	C_2	F_a	Power (in watts)
1 (1 : 300)	2154	0	11.4	4096	109.7
2 (1 : 200)	1559	0	9.6	4096	91.8
3 (1.5 : 300)	1342	0	4.2	2048	40.2
4 (1.5 : 200)	975	0	3.4	2048	32.9

5 Conclusions

A formal approach for optimally configuring an embedded computing platform for SAR processing was introduced. The formulation allows the platform to be configured using a combination of two types of cards. The variables that are optimized include the number of cards of each type and an FFT section size parameter. The advantage – in terms of minimizing consumed power – of optimally utilizing two card types (instead of restricting configurations to have only one card type) was illustrated through numerical studies. Also, an intuitive correspondence between optimal power consumption requirement and application-dependent parameters (i.e., SAR image resolution and platform velocity) was illustrated.

Acknowledgments

The authors thank T. Einstein for his help in explaining some of the finer points of the computational framework assumed in this paper, which was originally described in [3]. We also thank A. G. Antonio for sharing with us his expertise in radar systems and SAR processing.

References

[1] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Second Edition, John Wiley & Sons, New York, NY, 1993.

[2] J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar: Systems and Signal Processing*, John Wiley & Sons, New York, NY, 1991.

[3] T. Einstein, "Realtime Synthetic Aperture Radar Processing on the RACE Multicomputer," Application Note 203.0, Mercury Computing Systems, Inc., Chelmsford, MA, 1995.

[4] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, Sixth Edition, McGraw-Hill, New York, NY, 1995.

[5] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[6] "SHARC DSP Compute Nodes (3.3-Volt)," Mercury Computing Systems, Inc., Chelmsford, MA, Sept. 1995.

[7] J. S. Walker, *Fast Fourier Transforms*, Second Edition, CRC Press, New York, NY, 1996.