Computational Bit-width Allocation for Operations in Vector Calculus

Adam B. Kinsman and Nicola Nicolici
Department of Electrical and Computer Engineering
McMaster University,Hamilton,ON L8S4K1,Canada
kinsmaab@mcmaster.ca,nicola@mail.ece.mcmaster.ca

Abstract— **Automated bit-width allocation is a key step required for the design of hardware accelerators. The use of computational methods based on SAT-Modulo Theory to the problem of finite-precision bit-width allocation has recently been shown to overcome challenges faced by the known-art, particularly in the scientific computing domain. However, many such real-life applications are specified in terms of vectors and matrices and they are rendered infeasible by expansion into scalar equations. This paper proposes a framework to include operations from vector calculus and thus it enables tackling applications of practically relevant complexity.**

Keywords: Bit-width allocation, hardware accelerators

I. Introduction

Recent advances in field programmable gate arrays (FPGAs) have tightened the performance gap to application specific integrated circuits (ASICs), motivating research into hardware acceleration [1], [2], [3]. Due to its impact on resources and latency, choice of data representation (allocating the bit-width for the intermediate variables) is a key factor in the performance of such accelerators. Hence, developing structured approaches to automatically determine the data representation is becoming a central problem in high-level design automation of hardware accelerators; either during architectural exploration and behavioral synthesis, or as a pre-processing step to register transfer-level (RTL) synthesis.

The data representation problem has two facets, the precision problem and the range problem. This work is focussed on the range problem. It has been extensively researched in the digital signal processing (DSP) and embedded systems domains [4], yielding two classes of methods: 1) *formal* based primarily on affine arithmetic [5], [6] or interval arithmetic [7]; and 2) *empirical* based on simulation [8], [9], which can be either naive or smart (depending on how the input simulation vectors are generated and used).

Figure 1 summarizes the landscape of techniques that address the bit-width allocation problem. Empirical methods require extensive compute times and produce non-robust bit-widths; while formal methods guarantee robustness, but can over-allocate resources. Despite the success of the above techniques for a variety of DSP and embedded applications, interest has been mounting in custom acceleration of scientific computing. Examples include: computational fluid dynamics [1], molecular dynamics [2] or finite element modeling [3]. Scientific computing brings unique challenges because, in general, robust bit-widths are required in the scientific domain to guarantee correctness, which eliminates empirical methods. Further, ill-conditioned operations, such as division (common in numerical algorithms), can lead to severe over-allocation and even indeterminacy for the existing formal methods based on interval or affine arithmetic.

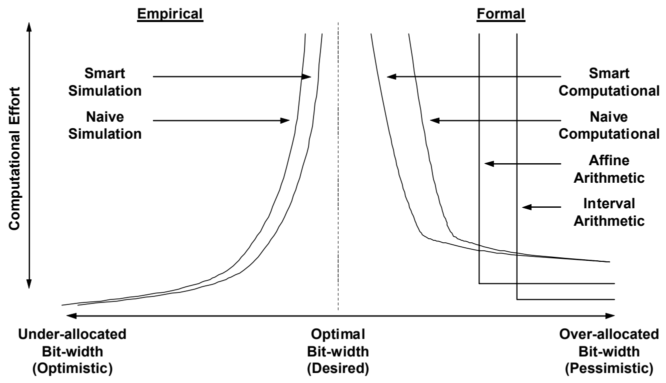


Fig. 1. Tradeoff Between Bit-width Allocation and Computational Effort.

The challenges associated with scientific computing have recently been addressed through a computational approach [10], building on the foundation and the recent developments of computational techniques such as Satisfiability-Modulo Theories (SMT) [11]. This approach is also shown on the right hand side of Figure 1 (as the naive computational approach). While this approach provides robust bit-widths even on ill-conditioned operations, it has limited scalability and is unable to directly address problems involving large vectors that arise frequently in scientific computing.

Given the critical role of computational bit-width allocation for designing hardware accelerators, in this paper we describe a method to automatically deal with problem instances based on large vectors. While the scalar based approach in [10] provides tight bit-widths (provided that it is given sufficient time), its computational requirements are very high. In response, we first introduce a basic vector-magnitude model which, while very favorable in computational requirements compared with full scalar expansion, produces overly pessimistic bit-widths. Subsequently we introduce the concept of block vector, which expands the vector-magnitude model to include some directional information. Due to its ability to tackle larger problems faster, we label this new approach as smart computational in Figure 1. This will enable to effectively scale the problem size, without losing the essential correlations in the dataflow, thus enabling automated bit-width allocation for practical applications.

II. Bit-width allocation in vector calculus

This section details a new algorithmic approach to bit-width allocation for operations in vector calculus.

A. Uniform Vector Bit-width

In order to leverage the vector structure of the calculation and thereby reduce the complexity of the bit-width problem to the point of being tractable, the independence (in terms of range/bit-width) of the vector components as scalars is given up. At the same time, hardware implementations of vector based calculations typically exhibit the following:

- Vectors are stored in memories with the same number of data bits at each address;
- Datapath calculation units are allocated to accommodate the full range of bit-widths across the elements of the vectors to which they apply;

Based on the above, the same number of bits tend to be used implicitly for all elements within a vector, which is exploited by the bit-width allocation problem. As a result, the techniques laid out in this section result in uniform bit-widths within a vector, i.e., all the elements in a vector use the same representation however, each distinct vector will still have a uniquely determined bit-width.

B. Vector Magnitudes

The approach to dealing with problems specified in terms of vectors centers around the fact that no element of a vector can have (absolute) value greater than the vector magnitude i.e., for a vector $\mathbf{x} \in \mathbb{R}^n$:

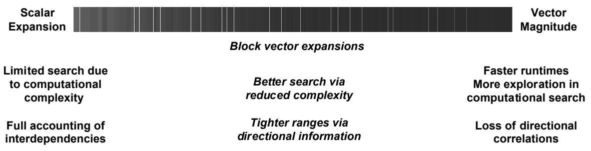
$$\mathbf{x} = (x_0, x_1, ..., x_{n-1})$$
$$||\mathbf{x}|| = \sqrt{\mathbf{x}^T\mathbf{x}}$$
$$|x_i| \leq ||\mathbf{x}|| \ , \ 0 \leq i \leq n-1$$

Starting from this fact, we can create from the input calculation a vector-magnitude model which can be used to obtain bounds on the magnitude of each vector, from which the required uniform bit-width for that vector can be inferred.

Creating the vector-magnitude model involves replacing elementary vector operations with equivalent operations bounding vector magnitude, Table I contains the specific operations used in this paper. When these substitutions are made, the number of variables for which bit-widths must be determined can be significantly reduced as well as the number of expressions, especially for large vectors.

TABLE I
Magnitude bounding operations.

Name	Vector Operation	Magnitude Model
Dot Product	$\mathbf{x}\cdot\mathbf{y}$	$\lvert\lvert\mathbf{x}\rvert\rvert\lvert\lvert\mathbf{y}\rvert\rvert\cos(\theta_{xy})$
Addition	$\mathbf{x}+\mathbf{y}$	$\sqrt{\lvert\lvert\mathbf{x}\rvert\rvert^2+\lvert\lvert\mathbf{y}\rvert\rvert^2+2\mathbf{x}\cdot\mathbf{y}}$
Subtraction	$\mathbf{x}-\mathbf{y}$	$\sqrt{\lvert\lvert\mathbf{x}\rvert\rvert^2+\lvert\lvert\mathbf{y}\rvert\rvert^2-2\mathbf{x}\cdot\mathbf{y}}$
Matrix Multiplication	$A\mathbf{x}$	$\sigma\lvert\lvert\mathbf{x}\rvert\rvert$ $\lvert\sigma_A^{min}\rvert \leq \sigma \leq \lvert\sigma_A^{max}\rvert$
Pointwise Product	$\mathbf{x}\circ\mathbf{y}$	$\varepsilon\lvert\lvert\mathbf{x}\rvert\rvert\lvert\lvert\mathbf{y}\rvert\rvert$ $0 \leq \varepsilon \leq 1$



Scalar Expansion
Vector Magnitude
Block vector expansions
Limited search due to computational complexity
Better search via reduced complexity
Faster runtimes More exploration in computational search
Full accounting of interdependencies
Tighter ranges via directional information
Loss of directional correlations

Fig. 2. Goal of Block Vector Representations.

The entries of Table I arise either as basic identities within, or derivations from, vector arithmetic. The first entry is simply one of the definitions of the standard inner product, and the addition and subtraction entries are resultant from the parallelogram law [12]. The matrix multiplication entry is based on knowing the singular values σ_i of the matrix and the pointwise product comes from: $\sum x_i^2 y_i^2 \leq (\sum x_i^2)(\sum y_i^2)$.

While significantly reducing the complexity of the range determination problem, the drawback to using this method is that directional correlations between vectors are virtually unaccounted for. For example, vectors $\mathbf{x}$ and $A\mathbf{x}$ are treated as having independent directions, while in fact the true range of vector $\mathbf{x}+A\mathbf{x}$ may be restricted due to the interdependence. In light of this drawback, the next section proposes a means of restoring some directional information without reverting entirely to the scalar expansion based formulation.

C. Directionality via Block Vectors

As discussed in the previous section, bounds on the magnitude of a vector can be used as bounds on the elements, with the advantage of significantly reduced computational complexity. In essence, the vector structure of the calculation (which would be obfuscated by expansion to scalars) is leveraged to speed up range exploration. These two methods of vector-magnitude and scalar expansion form the two extremes of a spectrum of approaches, as illustrated in Figure 2. At the scalar side, there is full description of interdependencies, but much higher computational complexity which limits how thoroughly one can search for the range limits. At the vector-magnitude side directional interdependencies are almost completely lost but computational effort is significantly reduced enabling more efficient use of range search effort. A tradeoff between these two extremes is made accessible through the use of block vectors, which this section details.

Simply put, expanding the vector-magnitude model to include some directional information amounts to expanding from the use of one variable per vector (recording magnitude) to multiple variables per vector, but still fewer than the number of elements per vector. Two natural questions arise: what information to store in those variables? If multiple options of similar compute complexity exist, then how to choose the option that can lead to tighter bounds?

As an example to the above, consider a simple 3x3 matrix multiplication $\mathbf{y} = A\mathbf{x}$, where $\mathbf{x},\mathbf{y} \in \mathbb{R}^3$ and $\mathbf{x} = [x_0, x_1, x_2]^T$. Figure 3 shows an example matrix A (having $9.9 \leq \sigma_A \leq 50.1$) as well as constraints on the component ranges of $\mathbf{x}$. In the left case in the figure, the vector-magnitude approach is applied: $||\mathbf{x}||$ is bounded by $\sqrt{2^2+2^2+10^2} = 10.4$, and the bound on $||\mathbf{y}||$ is obtained as $\sigma_A^{max}||\mathbf{x}||$ which for this example is $50.1 \times 10.4 \approx 521$.

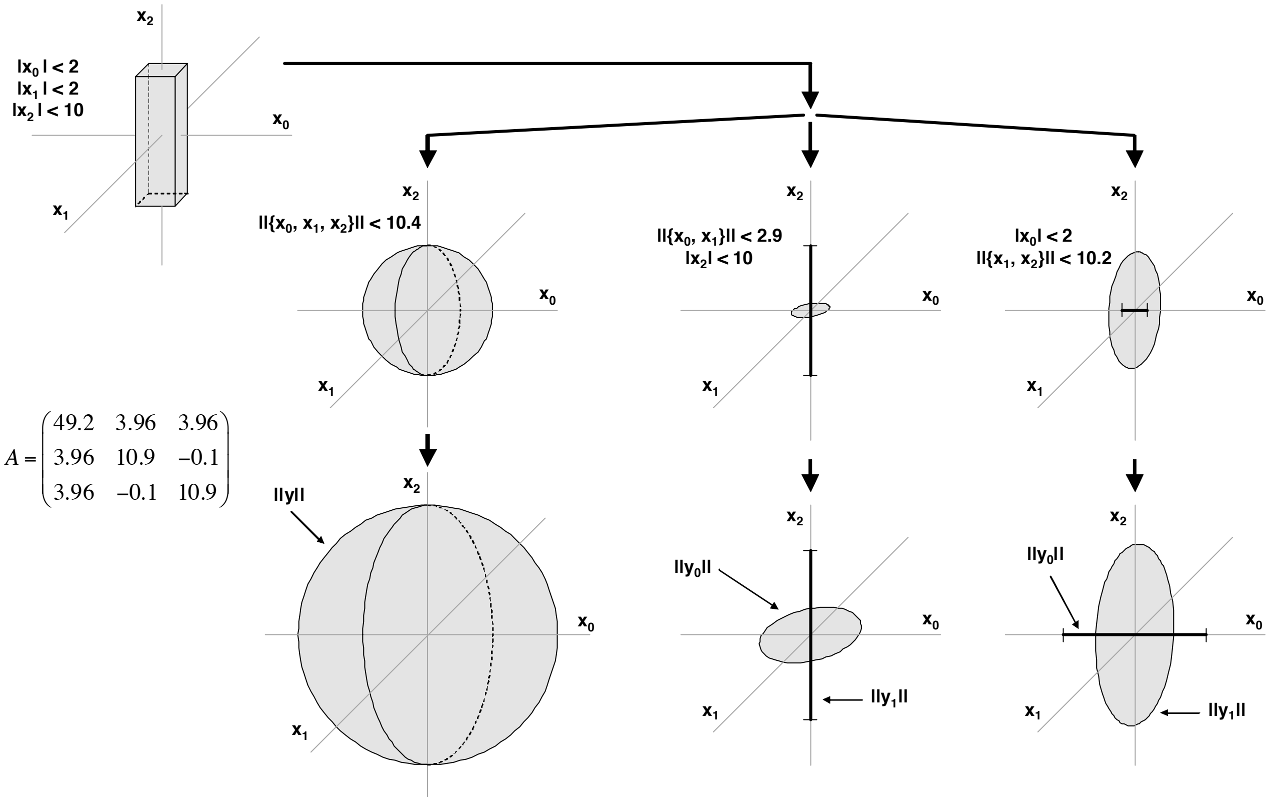


Fig. 3. Vector-Magnitude and Partitioned Block Vector Example.

The vector-magnitude estimate of ≈ 521 can be seen to be relatively poor, since true component bounds for matrix multiplication can be relatively easily calculated, ≈ 146 in this example. The inflation results from dismissal of correlations between components in the vector, which we address through the proposed partitioning into block vectors. The middle part of Figure 3 shows one partitioning possibility, around the component with the largest range, i.e. $\mathbf{x_0} = [x_0, x_1]^T, \mathbf{x_1} = x_2$. The input bounds now translate into a circle in the $[x_0, x_1]$-plane and a range on the x_2-axis. The corresponding partitioning of the matrix is:

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{y_0} \\ \mathbf{y_1} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_1} \end{bmatrix}$$

where $10.4 \leq \sigma_{A_{00}} \leq 49.7$ and $0 \leq \sigma_{A_{01}}, \sigma_{A_{10}} \leq 3.97$ and simply $A_{11} = \sigma_{A_{11}} = 10.9$. Using block vector arithmetic [13] (which bears a large degree of resemblance to standard vector arithmetic) it can be shown that $\mathbf{y_0} = A_{00}\mathbf{x_0} + A_{01}\mathbf{x_1}$ and $\mathbf{y_1} = A_{10}\mathbf{x_0} + A_{11}\mathbf{x_1}$. Expanding each of these equations using vector-magnitude in accordance with the operations from Table I, we obtain:

$$||\mathbf{y_0}|| = \sqrt{(\sigma_{A_{00}}||\mathbf{x_0}||)^2 + (\sigma_{A_{01}}||\mathbf{x_1}||)^2 + 2\sigma_{A_{00}}\sigma_{A_{01}}||\mathbf{x_0}||||\mathbf{x_1}||}$$
$$||\mathbf{y_1}|| = \sqrt{(\sigma_{A_{10}}||\mathbf{x_0}||)^2 + (\sigma_{A_{11}}||\mathbf{x_1}||)^2 + 2\sigma_{A_{10}}\sigma_{A_{11}}||\mathbf{x_0}||||\mathbf{x_1}||}$$

As the middle part of Figure 3 shows, applying the vector-magnitude calculation to the partitions individually amounts to expanding the $[x_0, x_1]$-plane circle, and expanding the x_2 range, after which these two magnitudes can be recombined by taking the norm of $[\mathbf{y_0}, \mathbf{y_1}]^T$ to obtain the overall magnitude $||\mathbf{y}|| = \sqrt{||\mathbf{y_0}||^2 + ||\mathbf{y_1}||^2}$. By applying the vector-magnitude calculation to the partitions individually, the directional information about the component with the largest range is taken into account. This yields $||\mathbf{y_0}|| \leq\approx 183$ and $||\mathbf{y_1}|| \leq\approx 154$, thus producing the bound $||\mathbf{y}|| \leq\approx 240$.

The right hand portion of Figure 3 shows an alternative way of partitioning the vector, with respect to the basis vectors of the matrix A. Decomposition of the matrix used in this example reveals the direction associated with the largest σ_A to be very close to the x_0 axis. Partitioning in this way (i.e., $\mathbf{x_0} = x_0, \mathbf{x_1} = [x_1, x_2]^T$) results in the same equations as above for partitioning around x_2, but with different values for the sub-matrices: $A_{00} = \sigma_{A_{00}} = 49.2$ and $0 \leq \sigma_{A_{01}}, \sigma_{A_{10}} \leq 5.61$ and $10.8 \leq \sigma_{A_{11}} \leq 11.0$. As the figure shows, we now have range on the x_0-axis and a circle in the $[x_1, x_2]$-plane which expand according the same rules (with different numbers) as before yielding this time: $||\mathbf{y_0}|| \leq\approx 137$ and $||\mathbf{y_1}|| \leq\approx 124$ producing a tighter overall magnitude bound $||\mathbf{y}|| \leq\approx 185$.

Given the larger circle resulting from this choice of expansion, it may seem surprising the bounds obtained are tighter in this case. However, consider that in the x_0 direction (very close to the direction of the largest σ_A), the bound is overestimated by $2.9/2 \approx 1.5$ in the middle case while it is exact in the rightmost case. Contrast this to the overestimation of the magnitude in the $[x_1, x_2]$-plane of only about 2% yet, different basis vectors for A could still reverse the situation. Clearly the decision how to partition a vector can have significant impact on the quality of the bounds. Consequently, the next section discusses an algorithmic solution for this decision.

D. Partition Selection

Recall that in the cases from the middle and right hand portions of Figure 3, the magnitude inflation is reduced, but for two different reasons. Considering this fact from another angle, it can be restated that the initial range inflation (which we are reducing by moving to block vectors) arises as a result of two different phenomena; 1) inferring larger ranges for individual components as a result of one or a few components with large ranges as in the middle case or 2) allowing any vector in the span defined by the magnitude to be scaled by the maximum σ_A even though this only really occurs along the direction of the corresponding basis vector of A.

In the case of magnitude overestimation resulting from one or a few large components, the impact can be quantified using range inflation: $f_{vec}(\mathbf{x}) = ||\mathbf{x}||/||\hat{\mathbf{x}}||$, where $\hat{\mathbf{x}}$ is $\mathbf{x}$ with the largest component removed. Intuitively, if all the components have relatively similar ranges, this expression will evaluate to near unity, but removal of a solitary large component range will have a value larger than and farther from 1, as in the example from Figure 3 (middle) of $10.4/2.9 = 3.6$

Turning to the second source of inflation based on σ_A, we can similarly quantify the penalty of using σ_A over the entire input span by evaluating the impact of removing a component associated with the largest σ_A. If we define α as the absolute value of the largest component of the basis vector corresponding to σ_A^{max}, and $\hat{A}$ as the block matrix obtained by removing that same component's corresponding row and column from A we can define: $f_{mat}(A) = \alpha\sigma_A^{max}/\sigma_{\hat{A}}^{max}$, where $\sigma_{\hat{A}}^{max}$ is the maximum across the σ^{max} of each sub-matrix of $\hat{A}$. The α factor is required to weight the impact of that basis vector as it relates to an actual component of the vector which A multiplies. When σ_A^{max} is greater than the other σ_A, f_{mat} will increase (Figure 3-right) to $0.99 \times 49.2/11.0 = 4.4$. Note finally that the partition with the greater value ($4.4 \geq 3.6$) produces the tighter magnitude bound ($185 < 240$).

Algorithm 1 shows the steps involved in extracting magnitude bounds (and hence bit-widths) of a vector based calculation. Taking the specification in terms of the calculation steps, and input variables and their ranges, *VectorMagnitude-Model* on line 1 creates the base vector-magnitude model as in Section II-B. The algorithm then proceeds by successively partitioning the model (line 5) and updating the magnitudes (line 7) until either no significant tightening of the magnitude bounds occurs (as defined by *GainThresh* in line 9) or until the model grows to become too complex (as defined by *SizeThresh* in line 10). Note that the *DetermineRanges* function (line 7) is based upon existing range analysis techniques, such as affine arithmetic or the computational method from [10]. The *Partition* function (line 5) utilizes the impact functions f_{vec} and f_{mat} to determine a good candidate for partitioning. Computing the f_{vec} function for each input vector is inexpensive, while the f_{mat} function involves two largest singular(eigen) value calculations for each matrix. It is worth noting however that a partition will not change the entire *Model*, and thus many of the f_{vec} and f_{mat} values can be reused across multiple calls of *Partition*.

```
Input : VectorCalculation, InputVariables,
        InputRanges, IntermediateVariables,
        SizeThresh, GainThresh
Output: IntermediateMagnitudes
1  Model = VectorMagnitudeModel(VectorCalculation,
       InputVariables, InputRanges, IntermediateVariables);
2  UpdatedMagnitudes=DetermineRanges(Model);
3  Flag = True;
4  while Flag do
5      Model = Partition(Model);
6      IntermediateMagnitudes = UpdatedMagnitudes;
7      UpdatedMagnitudes = DetermineRanges(Model);
8      Gain=max( UpdatedMagnitudes /
           IntermediateMagnitudes );
9      if $Gain < GainThresh$ then
           Flag = False;
       end
10     if $size(Model) > SizeThresh$ then
           Flag = False;
       end
   end
11 RETURN IntermediateMagnitudes;
```

Algorithm 1: Vector-Magnitude.

III. Case Studies

In this section we discuss 4 case studies, which show the proposed vector based method built on top of HySAT [11].

A. Analytic Center

The analytic center of a set of inequality constraints maximizes a distance metric from all constraint boundaries. This example comes from convex optimization, when using a distance metric based on a logarithmic penalty function; solving for the analytic center will give rise to calculations such as those below [14].

$$z[i] = d[i]\mathbf{a}[i] \cdot (\mathbf{C} - \mathbf{P}) \qquad d[i] = 1/(b[i] - \mathbf{a}[i] \cdot \mathbf{C})$$

The inequality constraints are defined by $\mathbf{a}[i], b[i]$ and $\mathbf{C}$ is the analytic center. The values $z[i]$ reflect a penalty of moving $\mathbf{C}$ to $\mathbf{P}$ with respect to $\mathbf{a}[i], b[i]$. For the specific case of 5 inequality constraints in $\mathbb{R}^3$, the vector equations can be expanded into scalar equations ($i \in \{1,2,3,4,5\}$):

$$q_1[i] = b[i] - (a_x[i]C_x + a_y[i]C_y + a_z[i]C_z) \qquad q_2[i] = 1/q_1[i]$$

$$z[i] = q_2[i]\,(a_x[i]\,(C_x - P_x) + a_y[i]\,(C_y - P_y) + a_z[i]\,(C_z - P_z))$$

Consider ranges of $C_x, C_y, C_z, P_x, P_y, P_z \in [-100, 100]$, and $a_x[i], a_y[i], a_z[i], b[i] \in [-10, 10]$. The equivalent vector-magnitude model is as follows:

$$q_1[i] = b[i] - ||\mathbf{a}||||\mathbf{C}||\cos(\theta_{aC}) \qquad q_2[i] = 1/q_1[i]$$

$$q_3[i] = \sqrt{||\mathbf{C}||^2 + ||\mathbf{P}||^2 - 2||\mathbf{C}||||\mathbf{P}||\cos(\theta_{CP})}$$
$$z[i] = q_2[i]||\mathbf{a}||q_3[i]\cos(\theta_{aq_3})$$

for $0 \leq ||\mathbf{a}|| \leq 17.4$ and $0 \leq ||\mathbf{C}||, ||\mathbf{P}|| \leq 173.3$.

TABLE II
AFFINE VS. SAT-MODULO FOR ANALYTIC CENTER.

Output	Scalar Affine		Scalar SAT-Modulo	
	Range	Bits	Range	Bits
$q_1[i]$	[-3010 , 3010]	13	[-3011 , 3011]	13
$q_2[i]$	∞	–	[-101 , 101]	8
$z[i]$	∞	–	[-3.6e5 , 3.6e5]	20
Output	Vector Affine		Vector SAT-Modulo	
	Range	Bits	Range	Bits
$q_1[i]$	[-3010 , 3010]	13	[-3011 , 3011]	13
$q_2[i]$	∞	–	[-101 , 101]	8
$q_3[i]$	∞	–	[-347 , 347]	10
$z[i]$	∞	–	[-6.1e5 , 6.0e5]	21

Table II shows ranges and equivalent bit-widths for the analytic center calculation when using affine arithmetic and the SAT-Modulo Theory (SMT) approach. Because in this and in the next case study no matrix multiplications exist, the block vectors are not required (only the vector-magnitude results are shown). Up to q_2, the vector-magnitude model gives identical results to the scalar expansion, but overestimates z, a result of losing the correlations. Note also that the singularity is circumvented by adding the constraint $q_1^2 \geq 0.0001$, which is convenient in the SMT approach (as detailed in [10]), but for which no mechanism exists in affine arithmetic.

B. Euclidian Projection

A second convex optimization/analysis based case study is Euclidian projection of a point/points onto a hyperplane. For a hyperplane $\mathbf{a} \cdot \mathbf{x} + b = 0$, a point $\mathbf{x0}$ will have a projection:

$$P(\mathbf{x0}) = \mathbf{x0} + \frac{b - \mathbf{a} \cdot \mathbf{x0}}{\mathbf{a} \cdot \mathbf{a}} \mathbf{a}$$

If we consider for the case study $\mathbf{x}$ substituted for $\mathbf{x0}$ and $\mathbf{a}, \mathbf{x} \in \mathbb{R}^5$, once again the vector equations can be expanded into scalar equations:

$$q_1 = \textstyle\sum_{i=0}^{4} a_i x_i; \quad q_2 = \sum_{i=0}^{4} a_i^2$$
$$q_3 = b - q_1; \quad q_4 = q_3 / q_2$$
$$z_i = x_i + q_4 a_i, \quad i \in \{0,1,2,3,4\}$$

where $-100 \leq x_i \leq 100$ and $-10 \leq a_i \leq 10$ for $i \in \{0,1,2,3,4\}$ and $-10 \leq b \leq 10$. The vector-magnitude model can also be formulated:

$$q_1 = ||\mathbf{a}||\,||\mathbf{x}|| \cos(\theta_{ax}); \quad q_2 = ||\mathbf{a}||^2$$
$$q_3 = b - q_1; \quad q_4 = q_3 / q_2$$
$$||\mathbf{z}|| = \sqrt{||\mathbf{x}||^2 + (q_4||\mathbf{a}||)^2 + 2||\mathbf{x}||q_4||\mathbf{a}|| \cos(\theta_{ax})}$$

with $0 \leq ||\mathbf{x}|| \leq 223.7$ and $1 \leq ||\mathbf{a}|| \leq 22.4$.

Table III shows ranges and equivalent bit-widths for Euclidian projection under the conditions described above. Similarly to analytic center, the vector model keeps up well with the scalar model for the first intermediates and in fact, as of q_4, the vector model actually surpasses the scalar model to provide *better* results, since it can be more thoroughly searched due to lower computational complexity. Note also that singularity avoidance is unnecessary due to the $||\mathbf{a}|| \geq 1$ constraint in the specification. Nonetheless, since no mechanism exists in affine arithmetic for capturing such a constraint, it cannot handle the division.

TABLE III
AFFINE VS. SAT-MODULO FOR EUCLIDIAN PROJECTION.

Output	Scalar Affine		Scalar SAT-Modulo	
	Range	Bits	Range	Bits
q_1	[-5000 , 5000]	14	[-5001 , 5001]	14
q_2	[0 , 500]	9	[0 , 501]	9
q_3	[-5010 , 5010]	14	[-5011 , 5011]	14
q_4	∞	–	[-348 , 348]	10
z_i	∞	–	[-646 , 630]	11
Output	Vector Affine		Vector SAT-Modulo	
	Range	Bits	Range	Bits
q_1	[-5003 , 5003]	14	[-5003 , 5003]	14
q_2	[-114 , 501]	9	[0 , 501]	9
q_3	[-5013 , 5013]	14	[-5013 , 5013]	14
q_4	∞	–	[-235 , 235]	9
z_i	∞	–	[-270 , 271]	10

C. Conjugate Gradient Method

The third case study is based on a single iteration of the Conjugate Gradient method for solving linear systems of equations. This method has many applications, examples include finite element method analysis and solutions to partial differential equations. A single (in fact the first) iteration can be formulated as follows. Given a matrix A, and a vector $\mathbf{b}$ with an initial guess for $\mathbf{x}$ (which solves $A\mathbf{x} = \mathbf{b}$) of $\mathbf{x_0}$ then let:

$$A = \begin{bmatrix} 5.665 & 2.630 & 1.088 \\ 2.630 & 9.624 & 2.647 \\ 1.088 & 2.647 & 6.329 \end{bmatrix}$$

$$\mathbf{r} = \mathbf{b} \qquad \mathbf{d} = \mathbf{x_0} \qquad \mathbf{q} = A\mathbf{d}$$
$$\alpha = \frac{\mathbf{r}^T\mathbf{r}}{\mathbf{d}^T\mathbf{q}} \qquad \mathbf{r}' = \mathbf{r} - \alpha\mathbf{q} \qquad \beta = \frac{\mathbf{r}'^T\mathbf{r}'}{\mathbf{r}^T\mathbf{r}}$$
$$\mathbf{d}' = \mathbf{r}' + \beta\mathbf{d}$$

and $\mathbf{r}', \mathbf{d}'$ feed into the next iteration. Note, because of the matrix multiplications, this case study (as well as the next one) implements also the block vector from Section II-C.

Taking constraints on the inputs as $0.1 \leq \mathbf{x_0} \leq 104$ and $0.1 \leq \mathbf{b} \leq 260$, Table IV shows the ranges of the intermediates obtained through affine arithmetic and SMT. In this case, due to the increased complexity of the scalar formulation caused by the matrix multiplication, the scalar method overestimates the ranges (this is because of the timeouts due to the problem size). While there is no guarantee that the ranges obtained through the vector method are optimal, they are significantly better than the scalar ones once again because the reduced complexity of the formulation enables more thorough search of the solution.

The significant reduction in bit-width from the vector method in this case study arises largely because the eigenvalues of the matrix are fairly uniformly distributed, and the input ranges of the vectors are uniform over the elements. This algorithm also has inherently weak directional interdependencies between intermediate variables, they are more strongly correlated in terms of magnitude, which further accounts for the success of the vector-magnitude approach. Because of this however, no significant gains are made by applying the block-vector approach, unlike the next study.

TABLE IV

AFFINE VS. SAT-MODULO FOR CONJUGATE GRADIENT.

Output	Scalar Affine		Scalar SAT-Modulo	
	Range	Bits	Range	Bits
d_i	[-104 , 104]	8	[-104 , 104]	8
r_i	[-260 , 260]	10	[-260 , 260]	10
q_i	[-1550 , 1550]	12	[-1075 , 1075]	12
rTr	[0 , 2.03e5]	19	[0 , 7.74e4]	17
dTq	[-1.38e5 , 3.72]	20	[0 , 1.44e5]	18
α	∞	–	[0 , 7.78e8]	30
r'_i	[-1.21e12 , 1.21e12]	42	[-1.02e11 , 9.40e10]	38
$r'Tr'$	[-2.06e24 , 2.70e24]	83	[0 , 2.08e22]	75
β	∞	–	[0 , 2.23e17]	58
d'_i	[-2.32e19 , 2.32e19]	66	[-8.24e18 , 7.33e18]	64

Output	Vector Affine		Vector SAT-Modulo	
	Range	Bits	Range	Bits
$\|\|\mathbf{d}\|\|$	[0.1 , 104]	8	[0.1 , 104]	8
$\|\|\mathbf{r}\|\|$	[0.1 , 260]	10	[0.1 , 260]	10
$\|\|\mathbf{q}\|\|$	[0.42 , 1300]	12	[0.42 , 1300]	12
$\mathbf{r}^T\mathbf{r}$	[0 , 6.77e4]	18	[0 , 6.77e4]	18
$\mathbf{d}^T\mathbf{q}$	[0 , 1.36e5]	19	[0 , 1.36e5]	19
α	∞	–	[0 , 1.60e6]	22
$\|\|\mathbf{r}'\|\|$	∞	–	[0 , 2.01e6]	22
$\mathbf{r}'^T\mathbf{r}'$	∞	–	[0 , 4.04e12]	43
β	∞	–	[0 , 5.97e7]	27
$\|\|\mathbf{d}'\|\|$	∞	–	[0 , 4.18e7]	27

D. FFT Based Correlation

The last case study is based on a fast method for computing correlation via sum-of-square-differences for 2-dimensional (2D) data, as applied to object tracking. An application and its dataflow are detailed in [15], which has been reproduced below. The inputs f,g are 2D arrays of data values, referred to by [15] as the search and reference window respectively and the final correlation result ssd is:

$$ssd = \mathscr{F}^{-1}\{(sinc \circ \mathscr{F}\{f \circ f\}) - 2(\mathscr{F}\{f\} \circ \mathscr{F}\{g\}^*)\}$$

where recall from Table I that $\circ$ is the element-wise product of arrays, $sinc$ is the 2D sinc function of appropriate size, and $\mathscr{F}$ is computed using the Fast Fourier Transform (FFT).

Two points are of primary interest in Table V. First, note that the vector method overestimates the range for $\mathscr{F}\{ssd\}$, this is due to the strong directional correlation of the pointwise matrix produce. However, the range for ssd is actually smaller, due to the same phenomenon of the previous case study, i.e., the scalar instance becomes too complex that it cannot be feasibly searched and thus overestimates the bit-width. In the bottom portion of the table, block vectors have been implemented leading to tightening of the range of the range for $\mathscr{F}\{ssd\}$, while the range of ssd is unaffected due to the directional independence of the FFT in the final step.

In all experiments, run times were on the order: scalar SMT-10's of min.; vector/block SMT-min.; Affine-seconds.

IV. CONCLUSION

This paper shows how to deal with vectors when allocating bit-widths, with an impact on both area and performance. This is a central problem when designing hardware accelerators. The long-term implications on the implementation flow for both ASICs and FPGAs may be significant because this is a key step before RTL synthesis, which traditionally lacked automation for robust solutions, as required by scientific applications that are presently migrated to hardware.

TABLE V

AFFINE VS. SAT-MODULO FOR FFT-BASED CORRELATION.

Output	Scalar Affine		Scalar SAT-Modulo	
	Range	Bits	Range	Bits
$f \circ f$	[0 , 6.56e4]	17	[0 , 6.56e4]	17
$F = \mathscr{F}\{f\}$	[-512 , 1024]	12	[-512 , 1024]	12
$\mathscr{F}\{f \circ f\}$	[-1.64e5 , 2.63e5]	20	[-1.32e5 , 2.63e5]	20
$F \circ G^*$	[-2.63e5 , 1.05e6]	22	[-2.63e5 , 1.05e6]	22
$\mathscr{F}\{ssd\}$	[-1.93e6 , 1.05e6]	22	[-1.99e6 , 5.25e5]	22
ssd	[-1.06e8 , 9.23e7]	28	[-1.01e8 , 8.01e7]	28

Output	Vector Affine		Vector SAT-Modulo	
	Range	Bits	Range	Bits
$f \circ f$	[0 , 4.20e6]	24	[0 , 4.20e6]	24
$F = \mathscr{F}\{f\}$	[0 , 2.05e3]	13	[0 , 2.05e3]	13
$\mathscr{F}\{f \circ f\}$	[0 , 4.20e6]	24	[0 , 4.20e6]	24
$F \circ G^*$	[0 , 4.20e6]	24	[0 , 4.20e6]	24
$\mathscr{F}\{ssd\}$	[0 , 7.55e7]	28	[0 , 7.55e7]	28
ssd	[0 , 7.55e7]	28	[0 , 7.55e7]	28

Output	Block Vector Affine		Block Vector SAT-Modulo	
	Range	Bits	Range	Bits
$f \circ f$	[0 , 4.20e6]	24	[0 , 4.20e6]	24
$F = \mathscr{F}\{f\}$	[0 , 2.05e3]	13	[0 , 2.05e3]	13
$\mathscr{F}\{f \circ f\}$	[0 , 4.20e6]	24	[0 , 4.20e6]	24
$F \circ G^*$	[0 , 2.10e6]	23	[0 , 2.10e6]	23
$\mathscr{F}\{ssd\}$	[0 , 3.78e7]	27	[0 , 3.78e7]	27
ssd	[0 , 7.55e7]	28	[0 , 7.55e7]	28

REFERENCES

[1] K. Sano, T. Iizuka, and S. Yamamoto, "Systolic Architecture for Computational Fluid Dynamics on FPGAs," in *Proc. Int. Symp. on Field-Programmable Custom Comp Machines*, 2007, pp. 107–116.

[2] R. Scrofano, M. Gokhale, F. Trouw, and V. Prasanna, "Accelerating Molecular Dynamics Simulations with Reconfigurable Computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 6, pp. 764–778, June 2008.

[3] R. Mafi, S. Sirouspour, B. Moody, B. Mahdavikhah, K. Elizeh, A. Kinsman, N. Nicolici, M. Fotoohi, and D. Madill, "Hardware-based Parallel Computing for Real-time Haptic Rendering of Deformable Objects," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2008, p. 4187.

[4] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable Computing: Architectures and Design Methods," *IEE Proceedings - CDT*, pp. 193–207, March 2005.

[5] J. Stolfi and L. de Figueiredo, "Self-Validated Numerical Methods and Applications," in *Brazilian Mathematics Colloq Monograph*, 1997.

[6] D.-U. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, "Accuracy-Guaranteed Bit-Width Optimization," *IEEE Transactions on CAD*, vol. 25, no. 10, pp. 1990–2000, Oct 2006.

[7] R. Moore, *Interval Analysis*. Prentice Hall, 1966.

[8] C. Shi and R. Brodersen, "An Automated Floating-point to Fixed-point Conversion Methodology," in *Proc. International Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2003, pp. 529–532.

[9] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou, "Low-Power Optimization by Smart Bit-Width Allocation in a SystemC-Based ASIC Design Environment," *IEEE Transactions on Computer-Aided Design*, pp. 447–455, March 2007.

[10] A.B. Kinsman and N. Nicolici, "Finite precision bit-width allocation using SAT-modulo theory," in *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2009, pp. 1106–1111.

[11] M. Franzle and C. Herde, "HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems," *Formal Methods in System Design*, vol. 30, no. 3, pp. 178–198, June 2007.

[12] G. Arfken, *Mathematical Methods for Physicists, 3rd Edition*. Orlando, FL: Academic Press, 1985.

[13] G. H. Golub and C. F. V. Loan, *Matrix Computations, 3rd Edition*. John Hopkins University Press, 1996.

[14] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[15] L. M. C. Ruey-Yuan Han, "Fast courier transform correlation tracking algorithm with background correction," US Patent number: 6970577, November 2005.