Automatic Design Space Exploration for an Error Tolerant Application

Samuel Coward
Visual Technologies Team
Intel Corporation
Folsom, USA
samuel.coward@intel.com

Theo Drane
Visual Technologies Team
Intel Corporation
Folsom, USA
theo.drane@intel.com

Yoav Harel
Visual Technologies Team
Intel Corporation
Folsom, USA
yoav.harel@intel.com

***Abstract*—Creating optimized hardware for error tolerant applications presents significant challenges as well as opportunities. Many algorithms in computer graphics & vision are error tolerant, as their application level correctness ultimately rests on human perception. This error tolerance can be exploited in reducing hardware implementation cost. The challenge is how to explore the space of application level correct designs to determine the optimized hardware architecture. This paper puts forward an approach to automatically explore the space which maximally exploits the acceptable error to minimize hardware cost for a particular graphics algorithm - *Level-Of-Detail*. Results, so far, have shown a 26% hardware area improvement.**

***Index Terms*—floating-point, numerical analysis, power-efficient, design automation, approximate computing, multiple-precision, computer graphics, accuracy hardware tradeoffs**

I. Introduction

In the design of fixed-functions in application-specific integrated circuits (ASICs) there are a myriad of considerations. The *quality of service* provided by the hardware function is determined by its bit-accurate functionality as well as its throughput, latency, area and power consumption. While certain algorithms have a universally accepted specification, *e.g.* the IEEE standard for floating-point arithmetic, others may have an acceptable range of allowable answers or even expected outcomes for only a limited set of allowable inputs. The latter categories are considered as error tolerant applications and present a considerable challenge to standard hardware design methodologies. Designers and architects must navigate four layers of choices; algorithm, number format(s), precision(s) and accuracies. The algorithm has to be chosen which has the *potential* to meet the application requirements. Once the algorithm is chosen, which number format(s) to use throughout the calculation is selected. Then the precision of every internal signal must be chosen and finally the accuracy that each operation must provide. Note that every one of these four layers of choices impacts bit-accurate functionality as well as hardware implementation cost. Given the incredible range of hardware designs that these choices offer, automatic design exploration that explores all four layers is the ultimate goal of design automation. Automatically determining application level correctness may itself be infeasible; for design automation to be possible, a set of executable metrics would need creating. Such metrics may be sufficient for application level correctness but not necessary, however they would enable automatic design space exploration.

In this paper, we focus on the question of precision for a floating-point design in a graphics application. Here precision changes have non trivial effects on design accuracy and hardware implementation cost.

II. Related Work

The key features of such design exploration are:

- Application Level Correctness
  - how the error analysis is performed
- Hardware Implementation Cost
  - whether this be actual synthesis or models of synthesis
- Parameter Space Exploration
  - having parametrized the space of designs, how the space is explored in finding optimized designs

In terms of error analysis, significant work derives from simulating designs with higher precision and extracting minimum, maximum and standard deviations statistics of variables in order to select required precisions, such as [1]. Such approaches fail to capture corner case events, for fixed-point designs, interval arithmetic (IA) has been used to propagate variable ranges to determine necessary precisions. IA can be pessimistic due to correlation between variables, *e.g.* $x - x$ would be considered to reside in a interval containing more than just 0. Overcoming this limitation, Affine Arithmetic was introduced (AA) which keeps first order correlations between variables, [2]. This has been further improved by considering Arithmetic Transforms (AT) which expands AA to include higher order terms while still remaining a linear transformation, [3]. These functions can be efficiently propagated through the design maintaining certain correlation information to provide bounds on the variable values. However, these bounds may still not be tight and these techniques are increasingly computationally expensive. To solve this, SAT and SAT-Modulo-Theory (SMT) have been used to prove or dis-prove bounds are achievable, and thus can be used to orchestrate between IA, AA and AT, [4], [5]. Precision loss has also been considered as introducing random variables into the computation, [6], [7], [8]. Where AA and AT seek to capture correlations, full automatic differentiation has been explored in [9].

In terms of hardware implementation cost, surrogate cost metrics which are functions of the precisions have been used for fixed-point designs; from the total number of bits in all internal variables, [10], functions linear in the bit-widths, [1], [7] to the total number of partial products bits in all adders and multipliers, [2], [8]. More advanced function fitting using weighted Chebyshev polynomials have been used to model area, delay and energy, [6]. In [11], the frameworks consisted of taking the design through to place and route and then power estimation. The challenge here is the ability to correctly discern whether one particular parametrization is superior to another without the need for computationally expensive and time consuming synthesis.

In terms of parameter space exploration iterative methods have been explored. For initialization, the design which uses the smallest same precision through-out has been used, [1], [6]. Conversely, one approach first maximizes all precisions and then one precision, p_1, is minimized while still maintaining application level acceptability. This is repeated for all parameters, the resultant set p_i is a design which is likely to be incorrect at an application level, but presents a useful seed point, [1]. For the remainder of this paper, the term valid and invalid will be used for parametrizations that are application level acceptable and unacceptable respectively.

For iterations within the parameter space, genetic algorithms and simulated annealing have been explored in [1], [6], [12], [8], [11]. Machine learning optimizers with particle swarm optimization has been explored in [13]. Search directions determined by error effect and expected improvements in hardware cost have been seen in [1]. Fork free areas in the data flow graph of the algorithm have been exploited in [10], this accelerates the exploration by reducing the dimensionality of the search space.

III. The Application: Level-Of-Detail

For our application, we turned to a computer graphics application regarding the rendering of textured objects, [14]. As the view point changes and objects go in and out of view, the level of visible texture detail changes drastically. Real-time minimization of the texture is too computationally expensive. The solution is to store multiple copies of the texture at different levels of minimization, these are referred to as *MipMap Levels*. Then, depending on the object position and orientation, the two relevant minimization levels are determined and averaging performed between them, Fig. 1.

How the MipMap levels need to be processed can be encapsulated in a single fixed-point number, which is called the *level-of-detail* (LOD). Given the LOD is used for subsequent filtering, error in its calculation can certainly be tolerated. As a result, industrial compliance tests for such computer graphics hardware, allow for some error. The LOD can be calculated from how the texture space is mapped to the screen pixel space. This potentially complex mapping is assumed, to reduce computational complexity, to be affine, Fig. 2.

The LOD is derived from the linear transformation matrix. Such hardware has hundreds of input bits and a 12-bit fixed-point output. Given the application and the drastic reduction in bit count, this is a highly error tolerant application. The form of LOD algorithm chosen for this design exploration work has 20 internal precisions, containing a range of floating-point operations, Fig. 3.

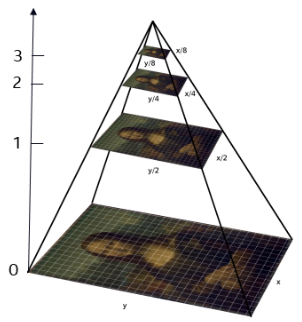
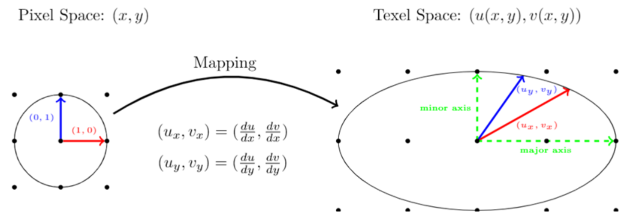


Fig. 1. MipMap Levels.


Pixel Space: (x, y)
Texel Space: $(u(x,y), v(x,y))$
Mapping
(0, 1)
(1, 0)
$(u_x, v_x) = (\frac{du}{dx}, \frac{dv}{dx})$
$(u_y, v_y) = (\frac{du}{dy}, \frac{dv}{dy})$
(u_y, v_y)
(u_x, v_x)
minor axis
major axis
Fig. 2. Texture Mapping.

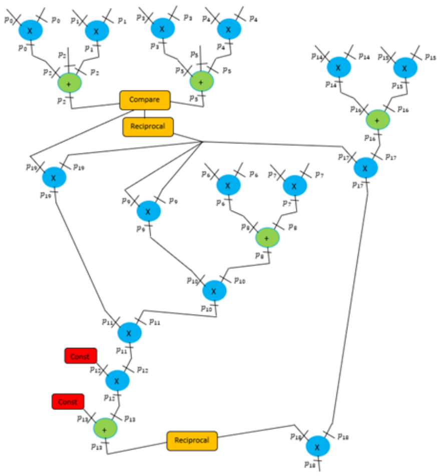


Fig. 3. Level-Of-Detail Data Flow Graph.

IV. Proposed Framework

Following such approaches as in [13], our proposed framework provides a method for exploring the parameter space. Our exploration needs to determine whether a given parametrization is valid and its hardware implementation cost. To overcome run-time limitations of logic synthesis and challenges in exploring the parameter space, our framework also introduces two novel contributions:

- Hardware Implementation Cost - a hybrid of actual synthesis and a model of synthesis results (a *proxy*).
- Parameter Space Exploration - a set of initialization points is constructed and a directed walk which is guaranteed to remain on the boundary of valid parametrizations.

The overall framework can be found in Fig. 4.

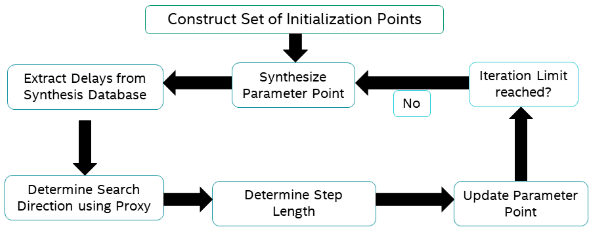

Construct Set of Initialization Points
Extract Delays from Synthesis Database
Synthesize Parameter Point
Iteration Limit reached?
No
Determine Search Direction using Proxy
Determine Step Length
Update Parameter Point

Fig. 4. Framework for Parameter Space Exploration via Directed Walks.

The details of how validity, hardware implementation cost and parameter space exploration is determined now follows.

A. Application Level Correctness (Validity)

The LOD hardware that is being optimized can be considered to be of the form:

$$f(\boldsymbol{x}, \boldsymbol{p}). \tag{1}$$

Where $\boldsymbol{x}$ are the design inputs and $\boldsymbol{p}$ are the set of mantissa width values that parametrize the design. An existing implementation sets all mantissa widths to be 23 (matching single precision floating-point), denoted as $\boldsymbol{p}_{\mathbf{max}}$. The application requires 8 fractional bits of precision when averaging between mipmap levels, hence the LOD value has 8 fractional bits. For this proof of concept work, the value of $f(\boldsymbol{x}, \boldsymbol{p}_{\mathbf{max}})$ will be considered to be *the reference* value and acceptable deviation can be no worse than 1 unit in the last place, 2^{-8} in this case. Thus a parametrization $\boldsymbol{p}$ is valid if:

$$|f(\boldsymbol{x}, \boldsymbol{p}_{\mathbf{max}}) - f(\boldsymbol{x}, \boldsymbol{p})| < 2^{-8} \quad \forall \boldsymbol{x}. \tag{2}$$

Python 2.7.15's scipy.optimize.minimize function was used to determine the extrema of $f(\boldsymbol{x}, \boldsymbol{p}_{\mathbf{max}}) - f(\boldsymbol{x}, \boldsymbol{p})$ and hence determine whether the validity condition was satisfied.

B. Hardware Implementation Cost

With a 20 dimensional parameter space and a logic synthesis of a particular parametrization taking between 3-4 hours of run-time, a hybrid of both [6] and [11] was taken; mixing actual synthesis with models of synthesis. Total gate count was chosen as the hardware implementation cost metric of interest. For significantly different parts of the parameter space, a full logic synthesis was performed. For localized changes to the parameter values a model of total gate count was used. Firstly multiple synthesis runs were performed to establish how each component within the LOD algorithm responded to changes in delay and mantissa width. The result were the curves found in Fig. 5.

From these sample data points, linear regression was performed to create an analytic gate count function G, quadratic

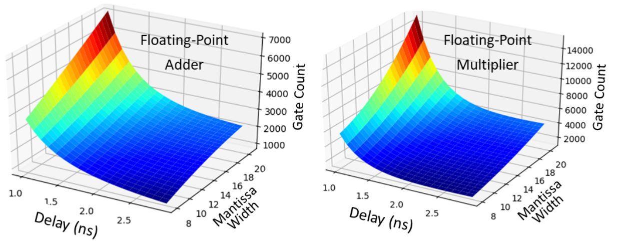


Fig. 5. Area-Delay Curves for Floating-Point Addition and Multiplication.

in the mantissa width p and inverse quadratic in delay d which determined the constants a_i:

$$G(d, p) = a_0 + \frac{a_1}{d} + \frac{a_2}{d^2} + a_3 p + \frac{a_4 p}{d} + \frac{a_5 p}{d^2} + a_6 p^2 + \frac{a_7 p^2}{d} + \frac{a_8 p^2}{d^2}. \tag{3}$$

Once an actual synthesis run is performed for parametrization $\boldsymbol{p}$, the delays over each component is extracted d_i and a model for the expected gate count difference by changing parametrization $\boldsymbol{p}$ to $\boldsymbol{p'}$ is (summing over all the components in the design):

$$\sum_i G_i(d_i, p_i') - G_i(d_i, p_i). \tag{4}$$

C. Parameter Space Exploration

The optimal design will be found on the boundary of valid space, as this is where the error tolerance will be maximally exploited and reduction of any particular mantissa width will result in an invalid design. To explore this boundary, a set of qualitatively distinct initialization points are sought. First considering the design defined by $\boldsymbol{p}_{\mathbf{max}}$; a point on the boundary can be generated by taking one parameter, reducing it until the design is only just valid and then repeating for all other parameters in turn. Given the design has 20 parameters, this process could generate $20! \approx 10^{18}$ initialization points by considering the parameters in different orders. Generating all such initializations is infeasible, so a method for determining qualitatively different elements within this set is required. This is achieved by partitioning the parameters, as in Fig. 6 into 5 connected groups. From these groups, $5! = 120$ initialization points are created by considering all possible orderings of these groups during the minimization process. Within each group a fixed ordering of minimization occurs. This approach can deliver qualitatively different initialization points, independent of the number of parameters.

Having chosen the initialization parametrizations, directed walks from all of these points are considered in parallel. From a parametrization $\boldsymbol{p}$, a logic synthesis run is performed and the delays across each component are extracted d_i. The analytic gate count function is used to determine the best search direction:

$$-\nabla_{\boldsymbol{p}} \sum_i G_i(d_i, p_i). \tag{5}$$

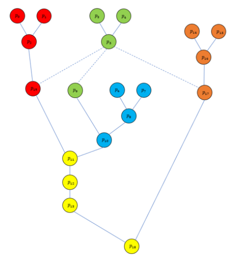


Fig. 6. Parameter Graph for the LOD Calculation.

This direction should point into invalid space. Direction vectors whose elements sum to zero are likely to move along the boundary. Hence an $offset$ is added to (5) such that the sum of its elements is 0. The updated parametrization is then:

$$\boldsymbol{p'} = \boldsymbol{p} + \alpha \left(offset\mathbf{1} - \nabla_{\boldsymbol{p}} \sum_i G_i(d_i, p_i) \right). \quad (6)$$

Where α determines how far in the new search direction to travel. Given the edge of valid space is of interest, α is chosen to guarantee that $\boldsymbol{p'}$ lies on the boundary.

V. The Result

The generated framework was a fully closed loop optimization; interim results evolved 120 initialization points across 8 iterations of the directed walking algorithm. The gate count for 8 initialization points can be found in Fig. 7. From these interim results, a parametrization with 53 fewer mantissa bits than the original design was found; whose gate count was reduced by 25.7%. The growth of gate count during the iterations present in this figure indicates that the steps being taken exceed the trust region of the analytic gate count function. The best parametrization came from one of the initialization points.

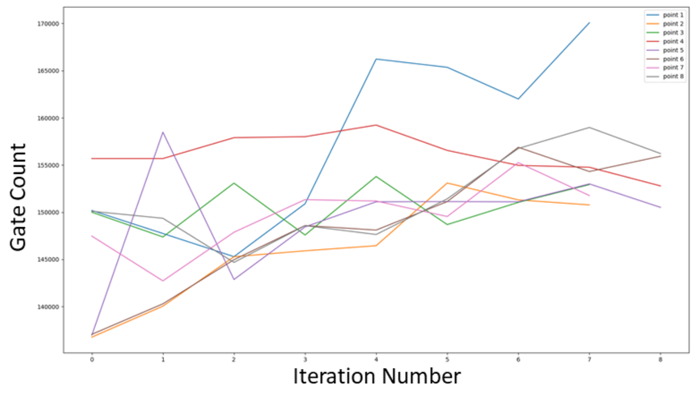


Fig. 7. Gate Count Results for 8 Directed Walks.

VI. Conclusion & Future Work

This paper has put forward a framework for precision optimization for error tolerant hardware consisting of directed walks from a set of initial points which explore the boundary of application level acceptable designs. Future work includes improving the robustness of the error modeling and error extrema computation. The directed walks are currently extended beyond the trust region of the analytic gate count function; further work is required to establish an improved interplay between logic synthesis and analytic models of the expected gate count. The interim results suggest that exploring more of the initialization point space may deliver a more robust optimization strategy. Further work is required on how best to perform parameter clustering for initial point creation. The framework can be extended to consider more advanced parametrizations of the design space to include accuracies and other number formats. It is hoped that the framework presented in this paper will contribute to the development of general robust frameworks for error tolerant hardware optimization.

References

[1] M. . Cantin, Y. Savaria, and P. Lavoie, "A comparison of automatic word length optimization procedures," in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 2, May 2002, p. II.

[2] D. . Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides, "Accuracy-guaranteed bit-width optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1990–2000, Oct 2006.

[3] Y. Pang, K. Radecka, and Z. Zilic, "An efficient hybrid engine to perform range analysis and allocate integer bit-widths for arithmetic circuits," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, Jan 2011, pp. 455–460.

[4] Y. Pang and K. Radecka, "An efficient algorithm of performing range analysis for fixed-point arithmetic circuits based on sat checking," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 1736–1739.

[5] A. B. Kinsman and N. Nicolici, "Finite precision bit-width allocation using sat-modulo theory," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 1106–1111.

[6] A. Ahmadi and M. Zwolinski, "A symbolic noise analysis approach to word-length optimization in dsp hardware," in *2007 International Symposium on Integrated Circuits*, Sep. 2007, pp. 457–460.

[7] Yu Pu and Yajun Ha, "An automated, efficient and static bit-width optimization methodology towards maximum bit-width-to-error tradeoff with affine arithmetic model," in *Asia and South Pacific Conference on Design Automation, 2006.*, Jan 2006, p. 6.

[8] A. Ahmadi and M. Zwolinski, "Area word-length trade off in dsp algorithm implementation and optimization," in *2005 The 2nd IEE/EURASIP Conference on DSPenabledRadio (Ref. No. 2005/11086)*, Sep. 2005, p. 16.

[9] A. A. Gaffar, O. Mencer, W. Luk, P. Y. K. Cheung, and N. Shirazi, "Floating-point bitwidth analysis via automatic differentiation," in *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.*, Dec 2002, pp. 158–165.

[10] J. Chung and L. Kim, "Bit-width optimization by divide-and-conquer for fixed-point digital signal processing systems," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3091–3101, Nov 2015.

[11] A. A. Gaffar, J. A. Clarke, and G. A. Constantinides, "Powerbit - power aware arithmetic bit-width optimization," in *2006 IEEE International Conference on Field Programmable Technology*, Dec 2006, pp. 289–292.

[12] M. . Cantin, Y. Savaria, D. Prodanos, and P. Lavoie, "An automatic word length determination method," in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, vol. 5, May 2001, pp. 53–56.

[13] M. Kurek and W. Luk, "Parametric reconfigurable designs with machine learning optimizer," in *2012 International Conference on Field-Programmable Technology*, Dec 2012, pp. 109–112.

[14] P. S. Heckbert, "Fundamentals of texture mapping and image warping," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-89-516, Jun 1989. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1989/5504.html