A Fast and Fuzzy Functional Simulator of Inexact Arithmetic Operators for Approximate Computing Systems

Justine Bonnot, Karol Desnos, Maxime Pelcat, Daniel Menard

A Fast and Fuzzy Functional Simulator of Inexact Arithmetic Operators for Approximate Computing Systems

Justine Bonnot
Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164
jbonnot@insa-rennes.fr

Karol Desnos
Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164
kdesnos@insa-rennes.fr

Maxime Pelcat
Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164
Institut Pascal UMR CNRS 6602
mpelcat@insa-rennes.fr

Daniel Menard
Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164
dmenard@insa-rennes.fr

ABSTRACT

Inexact operators are developed to exploit the tolerance of an application to imprecisions. These operators aim at reducing system energy consumption and memory footprint. In order to integrate the appropriate inexact operators in a complex system, the Quality of Service of the approximate system must be thoroughly studied through simulation. However, when simulating on a PC or workstation, the custom bit-level structures of inexact operators are not implemented in the instruction set of the simulating architecture. Consequently, the simulation requires a costly emulation, leading to expensive bit-level simulations. This paper proposes a new "Fast and Fuzzy" functional simulation method for inexact operators whose probabilistic behavior is correlated with the Most Significant Bits of the input operands. The proposed method processes real signal data and simplifies the error model for inexact operators, accelerating the simulation of the system. The modelization accuracy of the error can be controlled by a parameter called fuzzyness degree F. Using the proposed method, the bit-accurate logic-level simulation of inexact operators is replaced by an exact operator to which a pseudo-random error variable is added. Experiments on 16-bit operators show that the proposed simulation method, when compared to a bit-accurate logic level simulation, is up to 44 times faster.

1 INTRODUCTION

According to the Semiconductor Industry Association and Semiconductor Research Corporation, the total energy required for computing will exceed the estimated world's energy production in 2040, if no significant improvement is obtained in terms of energy-aware computing systems [2]. In this context, approximate computing is an active field of research that trades-off the output quality of a system for its energy consumption. Approximate computing benefits from the error resilience of algorithms in signal, image or video processing and data mining fields. Among the numerous existing approximate computing techniques, inexact operators have been designed to overcome the performance limitations of exact operators. Inexact operators are obtained by making a functional approximation. The boolean function of inexact operators is slightly modified to reduce the logic complexity and/or length of critical paths [9], [4], [17]. The main interest of inexact operators compared to a technique such as fixed-point coding is to generate an error with an error rate (ER) lesser than 1 [13]. Inexact operators can then have varied error profiles in terms of ER, the frequency of error occurrences, and error distance (ED) representing the error amplitude.

A challenge when including inexact operators in an application is to evaluate the impact of the approximation on the quality of service (QoS) λ at the output of the application. Potential approximations have to be analyzed to choose the best inexact operators with respect to the different constraints on the application implementation. The approximation design space exploration (ADSE) is large and requires a fast simulation to evaluate approximation impact on QoS.

Two families of state-of-the-art approaches exist to evaluate the errors induced by approximations on an application: 1) Analytical techniques [20], [11]mathematically express error statistics but the link between these statistics and the application QoS is not straightforward. 2) Functional simulation techniques run the approximate system on data and checks the obtained λ. Nevertheless, to mimic the inexact operator behavior, bit-accurate simulations at the logic-level (BALL simulations) are required. BALL simulations are two or three orders of magnitude more complex than classical simulations with native data types, as shown in Section 4.

In this paper, we propose a fast functional simulation technique for inexact operators. The proposed "Fast and Fuzzy" (FnF) simulator simplifies the approximation error model to fasten the simulation. The BALL simulation of an inexact operator is replaced by the exact operation to which is added a pseudo-random variable (PRV) modeling the approximation error. To take into account the correlation between input values and approximation errors, the input operand set is decomposed into subspaces and a different PRV is associated to each subspace. Each PRV is defined to mimic the error in terms of ER and ED generated by the approximation. The proposed simulator is designed to be operator agnostic and is intended to be used during the ADSE process. The FnF simulator is designed to quickly evaluate the impact of different approximations at the hardware level on the QoS of an application.

The paper is organized as follows: related works on techniques to model and simulate inexact operators are presented in Section 2. Section 3 details the proposed FnF simulation technique. The efficiency of the FnF simulation in terms of time savings and quality, is exposed in Section 4. Finally, Section 5 concludes the article.

, . https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORKS

Inexact operators are designed by making a functional approximation. The functionality of the operator is modified to reduce e.g. the combinatorial logic complexity and the length of critical paths. The effect on the circuit is a reduction in power consumption and/or area. Inexact operators have been developed to overcome the performance limitations of exact arithmetic operators. Many inexact adders have been proposed ([9], [4], [17]) and their performances are compared in [8]. The comparison between adders is done in terms of normalized mean error distance (NMED), ER, and mean relative error distance (MRED). The error profile of inexact operators is often represented as a Probability Mass Function (PMF).

Different approaches have been proposed to characterize the error of an inexact operator. Probabilistic error models in [11] and [10] derive the link between the inputs and the error occurrence in the operator. These techniques are accurate but custom to particular types of operators. In the case of small input bit-width operators, exhaustive simulations can be launched to give an accurate expression of the PMF. In the more general case, inexact operators are long and complex to simulate since they are working at the logic level. For high word-length operators or to avoid exhaustive simulations, Monte-Carlo simulations can be used to compare the performances of different inexact operators [6]. These simulations strongly depend on the chosen input data distribution and on the number of launched simulations. This technique is also non-deterministic and its execution time depends on the required simulation precision.

Analytical methods provide mathematical expressions of an error metric (ER, NMED or MRED) on inexact operators. The chosen error metric can then be quickly evaluated. For instance, when implementing an approximation based on fixed-point coding, the error metric is the error power. To derive the error power, perturbation theory is used [15]. Perturbation theory is however not applicable to inexact operators because it builds on the hypothesis that errors are small compared to signal, which is not the case in most inexact operators. Several analytical methods have thus been proposed to get the impact of an inexact operator on an application. Interval Arithmetic (IA) can be used [1], but usually overestimates the error bounds. To overcome the pessimistic bounds given by IA, Affine Arithmetic (AA)takes into account the first order dependencies between variables [5]. Nevertheless, these techniques do not give information on the ER of inexact operators. Modified Interval (MIA) and Affine (MAA) Arithmetic have also been proposed in [7]. MIA and MAA require a characterization of the PMF of the inexact operator and derive rules to propagate it through the application. However, the number of terms required to propagate the PMF suffers from range explosion. For instance, to propagate the PMF through 4 Multiply-Accumulate (MAC) operations, 8 million terms are needed [7]. Analytical techniques are completely describing the PMF of an inexact operator but do not give a direct link between the error metric and λ.

The straightforward method to determine the impact of the error on λ is to perform a functional simulation of the application. A functional simulation of an inexact operator simulates the operator at the logic level. Thus, the simulation time is significantly higher than the one obtained with a classical simulation using workstation native data types. The simulation in C code of an Almost Correct Adder (ACA, [17]) on 32-bits takes 197× more time than the floating-point addition natively supported by the processor. When it comes to the Lower-Error Fixed-Width Multiplier (AAM, [16]), a multiplication of two 32-bit operands takes 6250× more simulation time than a floating-point multiplication. Inexact operators are generally more intricate in terms of hardware implementation or memory accesses than their exact implementation.

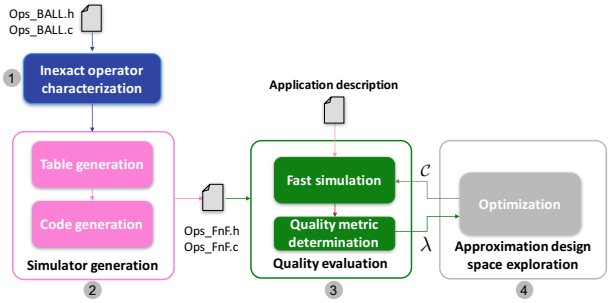


Figure 1: ADSE flow integrating the proposed FnF simulator to evaluate the application QoS with inexact operators

To the best of our knowledge, no method has been proposed to accelerate the simulation of inexact operators by combining exact computation and statistical methods. The goal of the proposed simulator is to replace the BALL simulation of the inexact operator $\widehat{\diamond}$ by the simulation of the exact operator $\diamond$ plus a Pseudo-Random Variable (PRV).

3 PROPOSED FAST AND FUZZY SIMULATOR

Our proposed simulation technique for inexact operators simplifies the modeling of the approximation error to fasten the simulation of inexact operators within the targeted application.

The proposed FnF simulator is set in the design flow presented in Figure 1. The ADSE, corresponding to Block 4 in Figure 1 aims at finding the best inexact operator for each operation in the application as well as the best parameters for their tuning. For each test of a configuration C, λ is evaluated from the results of the fast simulation carried-out with the FnF simulator (Block 3). The technique proposed for the fast simulation is presented in Section 3.1.

The FnF simulator uses pre-computed tables computed in Block 2. The generation of the tables is presented in Section 3.2. The generation of the tables requires the knowledge of the approximation error statistics. These statistics are provided by the characterization process corresponding to Block 1 in Figure 1. Analytical techniques [11], Monte-Carlo simulations [6] or exhaustive simulations are the different alternatives to obtain these statistics. In the rest of the paper, the statistics on the error have been derived from exhaustive simulations for accuracy and generic purpose.

3.1 Fast and Fuzzy simulation of $x\widehat{\diamond}y$

Let's consider an approximate operator $\widehat{\diamond}$ whose input operands $x \in I_x = [\underline{x}; \overline{x}]$ and $y \in I_y = [\underline{y}; \overline{y}]$ are encoded on N_x and N_y bits respectively. To avoid a coarse error modeling, the input set $I = I_x \times I_y$ is decomposed in subspaces $S_{ij} = I_{xi} \times I_{yj}$ such that $\bigcup_{i,j} S_{ij} = I$. Since the ER is not equal to 1 in each S_{ij}, our method

determines a pseudo-random variable PRV e_{ij} for each S_{ij} with statistical characteristics provided by the inexact operator characterization phase (Block 1). It is important to note that our FnF simulator can take as input the statistics on the error provided by [11], by exhaustive or Monte-Carlo simulations. Our simulator does not output the exact value of the approximate operation but generates the error with the same statistical characteristics as the error at the output of the approximate operator.

Our method reduces the software simulation time of an inexact operator by replacing the BALL simulation of $\widehat{\diamond}$ by the accurate version of the operator $\diamond$ plus a PRV e_{ij} whose statistical characteristics are computed from the error generated by $\widehat{\diamond}$. Figure 2 illustrates this principle.

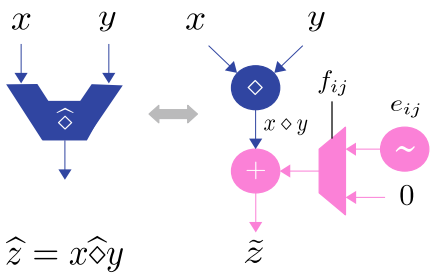


Figure 2: Statistical equivalence between BALL and FnF simulation of $x\widehat{\diamond}y$

According to the ER f_{ij} in S_{ij}, 0 or e_{ij} is added to $x \diamond y$. The input operands sharing the same N-F MSBs (Most Significant Bits) are grouped in the same subspace S_{ij}, and are then associated to the same PRV e_{ij}, hence the error on the result of the simulation. The size of the subspaces S_{ij} controls the modeling grain and is embodied by the user-defined fuzziness degree F. F has an impact on the accuracy of our FnF simulator. Hence, the larger F is, the more different outputs of the simulated operator are summarized with the same PRV e_{ij}, thus increasing the simulation imprecision.

Algorithm 1 Fast and Fuzzy Simulation of $x\widehat{\diamond}y$

```
procedure FnF_◇̂(x, y, N, F)
    x_0 = x >> F                          ▷ Pre-process operands
    y_0 = y >> F
    k = T_idx[x_0][y_0]
    if k == 0 then                        ▷ Error-free test
        return x ◇ y
    else                                  ▷ Pseudo-random number generation
        M_1 = 2^F − 1
        p_ij = generatePRNumber(M_1, x, y)
        e_ij = generateError(p_ij, f_ij, a_ij, b_ij)
        return x ◇ y + e_ij
    end if
end procedure
```

Algorithm 1 details the proposed simulation process for $N_x = N_y = N$ and Figure 3 illustrates the flow of our simulation. The first step of the FnF simulation is to *pre-process* the input operands x and y by extracting their N-F MSBs. This leads to the values x_0 and y_0 respectively. x_0 and y_0 indicate to which subspace S_{ij} x and y belong to. The index $k = \mathcal{T}_{idx}[x_0][y_0]$ indicates if $x\widehat{\diamond}y$ may generate an error. $\mathcal{T}_{idx}$ is a precomputed table that indicates if an error occurs in S_{ij}. Consequently, if k is equal to zero, no error is generated and the accurate version of $x \diamond y$ is directly returned, thus avoiding any simulation time overhead. If k is different from zero, a set of statistical characteristics (a_{ij}, b_{ij}, f_{ij}) is used to compute the PRV e_{ij}. The pre-computed table $\mathcal{T}_{err}[k]$ stores the sets of statistical characteristics for each S_{ij} where an error occurs. Section 3.2 presents the generation of the two abovementioned tables. The PRV e_{ij} is computed with the following expression

$$e_{ij} = a_{ij} + b_{ij} * p_{ij}, \tag{1}$$

with p_{ij} a pseudo-random number. In the FnF simulator, p_{ij}, is geerated from the F LSBs of the input operands. Indeed, the LSBs of a signal can be considered as a white random additive noise non-correlated with the input signal as derived in [19]. The F LSBs of x and y are concatenated and finally scrambled by a `xor` operation with a constant K. The purpose of these operations is to map the input operands $(x, y) \in I_x \times I_y$ to $p_{ij} \in [0; 2^{2F}[$ in a bijective way.

Finally, an asset of our simulator is that the execution time depends on the number of errors committed by the original inexact operator. The less errors an operator generates, the faster our FnF simulator.

3.2 Automated simulator construction

The main contribution is to reduce the simulation time to get the value $x\widehat{\diamond}y$ with the method detailed in Section 3.1. The error modeling is simplified using the tables $\mathcal{T}_{idx}$ and $\mathcal{T}_{err}$ to store the characteristics of the PRVs used to compute the output value. The generation of these tables is done only once for each operator and off-line.

Table $\mathcal{T}_{idx}$ indicates if a combination of input operands does or not generate an error. The size of $\mathcal{T}_{idx}$ is $2^{N_x+N_y-2F}$. The N_x-F and $N_y - F$ MSBs of the operands are masked and operands obtaining the same mask value are mapped to the same subspace S_{ij} in $\mathcal{T}_{idx}$. Equation 2 presents the computation of $\mathcal{T}_{idx}$.

$$\mathcal{T}_{idx}[i,j] = \begin{cases} 0 & \text{if } \forall x, y \in S_{ij},\ x\widehat{\diamond}y = x \diamond y \\ k & \text{else.} \end{cases} \tag{2}$$

$\mathcal{T}_{err}$ stores the ED characteristics embodied by the affine form (a_{ij}, b_{ij}), and the threshold t_{ij} used to generate an error with the same ER as in S_{ij}. The size of $\mathcal{T}_{err}$ then depends on the number of subspaces where an error occurs.

To generate an error with the same statistical characteristics as the inexact operator, a system with two equations for three unknowns is set. The threshold t_{ij} is then computed to be equal to the number of errors to generate. We then have the system $\mathcal{S}$ in Equation 3 to solve, with f_{ij} representing the ER in S_{ij}, A_{ij}, the maximum ED in S_{ij}, and μ_{ij} the mean ED in S_{ij}.

$$\mathcal{S}: \begin{cases} t_{ij} & = 2^{2*F}(1 - f_{ij}) \\ A_{ij} & = a_{ij} + b_{ij} * \max_{i,j}(p_{ij}) \\ \mu_{ij} * f_{ij} & = \frac{\sum_{p=0}^{2^{2*F}-1} a_{ij}+b_{ij}*p}{2^{2*F}} \end{cases} \tag{3}$$

$\mathcal{S}$ is developed to extract a_{ij} and b_{ij} in Equation 4. Indeed, the maximum value of the pseudo-random number p_{ij} is $2^{2F} - 1$ and

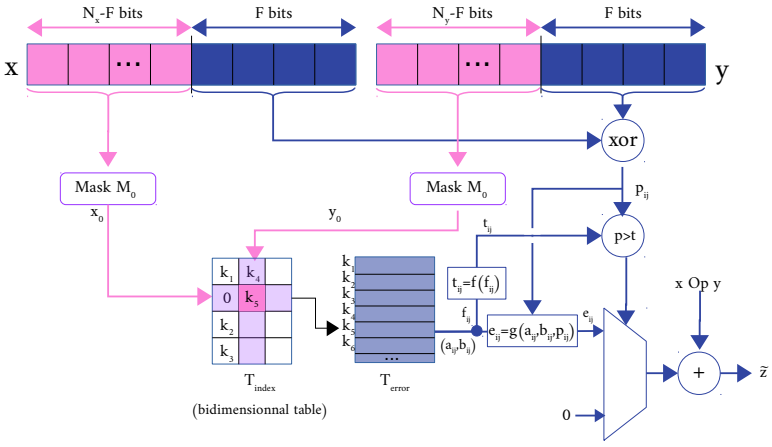


Figure 3: Graph flow of the proposed Fast and Fuzzy simulation

the third equation of the system can be separated to highlight an arithmetic sequence whose first term is t_{ij} and common term 1.

$$\begin{cases} b_{ij} & = \frac{2^{2F+1}\mu_{ij}f_{ij} - 2A_{ij}(2^{2F} - 1 - t_{ij})}{(2^{2F} - 1 - t_{ij})(2^{2F} - 3 + t_{ij} - 2^{2F+1})} \\ a_{ij} & = A_{ij} - b_{ij}(2^{2F} - 1) \end{cases} \tag{4}$$

The error e_{ij} can also be modeled by a constant value if b_{ij} is set to zero, but the accuracy of our simulator decreases. In this case, the threshold t_{ij} is computed for each S_{ij} by equalizing the mean ED of $\widehat{\diamond}$ with the mean ED generated by the FnF simulation of $\widehat{\diamond}$, as presented in Equation 5.

$$\mu_{ij} * f_{ij} = a_{ij} * (1 - \frac{t_{ij}}{2^{2*F}}) \tag{5}$$

$$\Leftrightarrow \quad t_{ij} = 2^{2*F} \cdot (1 - \frac{\mu_{ij} \cdot f_{ij}}{a_{ij}}) \tag{6}$$

Besides, an inexact operator can be designed to generate errors always lower or equal to $x \diamond y$. In this case, e_{ij} cannot be greater than the value of the operation $x \diamond y$, since the output of the simulation is $x \diamond y + e_{ij}$. In this case, the generated error e_{ij} is equal to $\max(-\min_{x,y \in S_{ij}}(x \diamond y), A_{ij})$. The maximum ED is not always generated. Consequently, to keep the same mean ED, the threshold t_{ij} has to be lower to generate errors more often. The worst possible case is obtained when e_{ij} is always equal to $x \diamond y$ in S_{ij}: in this case, the value of t_{ij} is presented in Equation 7. Else, e_{ij} is always equal to A_{ij}, the maximum ED in S_{ij}.

$$t_{ij} = 2^{2*F} \cdot (1 - \frac{\mu_{ij} \cdot f_{ij}}{-\min_{x,y \in S_{ij}}(x \diamond y)}) \tag{7}$$

4 EXPERIMENTAL RESULTS

The FnF simulator proposes to simulate an inexact operator using a simple error model. To evaluate the impact on the simulation of the proposed method, three points have to be highlighted: the time savings offered by the simulator for the simulation of a single operation presented in Section 4.1, the trade-off simulation time/accuracy of the proposed simulation depending on the fuzzyness degree F, detailed in Section 4.2 and the accuracy of the QoS evaluation compared to the time savings in an ADSE process, extended in Section 4.3. The results have been obtained on a processor Intel i7-6700 with 32GBytes of RAM.

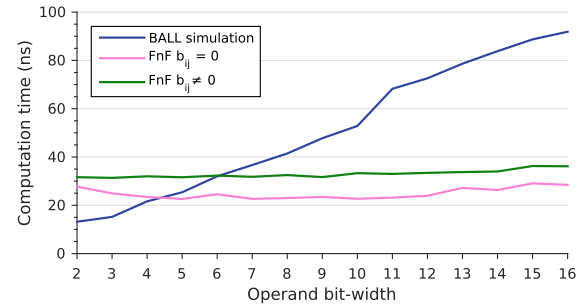
4.1 Simulation time savings for the FnF simulator

To quickly test the impact of an inexact operator in an application, the easiest solution is to simulate the application in software with BALL simulation. The simulation time of our FnF simulator is then compared with the software BALL simulation time obtained with the C code from the App Test framework [3] for a single operation and various input operand bit-width. The time required to generate the tables (Block 4 in the flow presented in Figure 1) is less than 140s for the tested 16-bit inexact operators.

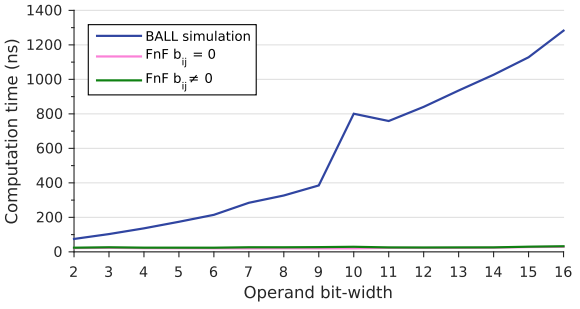
Figure 4a represents the simulation times for the ACA and Figure 4b for the Lower-Error Fixed-Width Multiplier (AAM), an approximate array multiplier presented in [16]. The time for simulating a single operation is represented depending on the operand bit-width. The obtained times have been **averaged** for all the possible configurations of carry-chain length (for the ACA) and all the possible values of F for both operators. A BALL simulation of the ACA on 16-bit takes 300 more time than classical simulation of the exact operator with native data types, and the BALL simulation of the AAM, for the multiplication of two 16-bit operands takes 4200 more time than a simulation with native data types.

The simulation times for the FnF simulator are represented for $b_{ij} \neq 0$ and $b_{ij} = 0$, the affine modeling being slightly longer to simulate than when $b_{ij} = 0$. For the ACA, the FnF simulation is always faster than the BALL one with operand bit-width greater than 5 if $b_{ij} = 0$, else greater than 6. On a single 16-bit addition, the BALL simulation takes 3.5 more time than the FnF simulation. The gain for this operator is reasonable since the design of the ACA is quite simple and can easily be reproduced with C-code. However, when it comes to the AAM, whose design is much more complex than the one of the ACA, the FnF simulation is always faster and

the BALL simulation takes 44 more time than the FnF simulation on a single 16-bit operation.



(a) ACA inexact operator



(b) AAM inexact operator

Figure 4: Simulation time for the FnF and the BALL simulation of one operation.

4.2 Trade-off simulation time/quality

The number of Fuzzy bits F embodies a trade-off between the simulation time, the size of the tables to store and the quality of the simulation. Figure 5 represents the Normalized Rooted Mean Squared Error (NRMSE) of the FnF simulation of the ACA compared to the BALL simulation of the inexact operator, $\delta_{FnF/BALL}$, for two input operand bit-widths. The NRMSE of the inexact operator simulated with BALL simulation compared to the exact operator, $\delta_{BALL/ex}$, is equal to 0.35. When $F = N-1$, a single PRV is used to model the errors, $\delta_{FnF/BALL}$ and $\delta_{BALL/ex}$ are of the same order of magnitude. When $F = 0$, the simulator behaves like a Look-Up Table (LUT) containing the error due to inexact operation. Thus, $\delta_{FnF/BALL}$ is equal to zero. Besides, the relative difference between the FnF simulation and BALL simulation stays lower than 10% for $N - F$ higher than 4.

The fuzzyness degree F has also an impact on the simulation time as demonstrated in Figure 6 for the AAM. The FnF simulation time is represented for different input bit-widths, depending on the fuzzyness degree. The highest F, the smallest the tables and the least cache misses. For instance, for $N = 13$, F has to be at least equal to 3 to be faster than the BALL simulation (simulation time gain of 2.4) and the maximum simulation time gain if $F = N - 1$ is equal to 49.2.

4.3 ADSE for a stereo vision application

Designing an application with approximate computing, several parameters have to be tuned during the ADSE. During this phase,

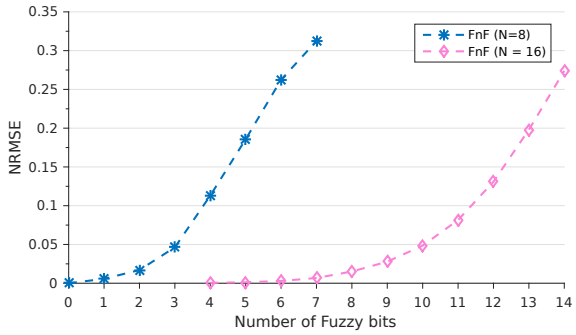


Figure 5: NRMSE between the FnF simulation and the BALL simulation of $\widehat{\diamond}$ (ACA with a carry chain length cut at $N/2$)

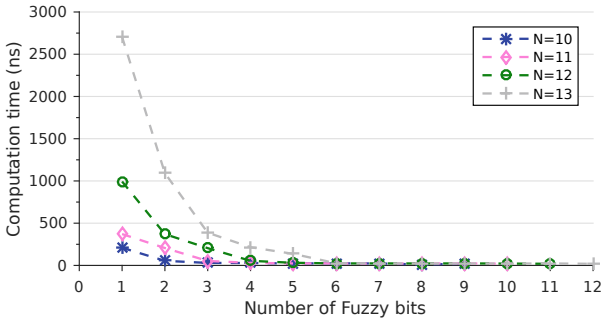


Figure 6: FnF simulation time depending on the fuzzyness degree F for the AAM

the application is run several times to measure the quality of the result at the output of the application with the chosen parameters, to finally select the parameters leading to the best performances with respect to the designer's constraints. To give an idea of the intensiveness of this phase, several tunable parameters are given hereafter (non exhaustive list). The percentage of replaced operators, the replaced operations, the inexact operator that is used (ex: AAM or AWM), if the operator is tunable, the parameter to tune (ex: carry-chain length for the ACA) and finally, the set of inputs on which the algorithm has to be tested, are tunable parameters.

To illustrate the DSE phase, the stereomatching algorithm is considered [12]. This computer vision algorithm outputs a depth map given two rectified input images. Approximate computing is particularly interesting for this algorithm since it is computationally expensive. In this experiment, only the most computation intensive kernel of this algorithm is considered for the ADSE. Only taking into account the percentage of operators as well as the operations to replace, but no tuning of the operators, 158 simulations are needed. If the replacement of multiply operations is solely considered, testing all the possible configurations with the BALL simulation takes **8.82 hours**. The simulation using the FnF simulator only requires **12.71 minutes**.

This test has been done using solely an input image from a test set extracted from the Middlebury database [14]. To validate the

approximation on the stereomatching algorithm, the test needs to be done on a complete data set, hence increasing the number of simulations to perform. Several configurations have been tested and are presented in Table 1. The quality metric used to measure the difference between FnF and the BALL simulation is the SSIM presented in [18]. The terms r_{ADD} and r_{MULT} represent the ratio of respectively addition and multiplication which use an inexact operator. The gain $G_t = \frac{t_{BALL}}{t_{FnF}}$ in term of simulation time of the proposed approach compare to a BALL simulation is provided. The accuracy degradation of the application QoS evaluation is provided through Δ_a that measures the relative difference between the SSIM of both output images obtained with FnF and BALL simulations. The multiply operations have been replaced by the AAM and the additions have been replaced by the ACA with a carry-chain length equal to 7. Both operators are on 16-bit. The most important accuracy degradation for QoS evaluation depends on which operation is impacted by the approximation, but the degradation is always lower than 6%. The FnF simulator leads to an accurate QoS evaluation and allows to save a non negligible simulation time by dividing the simulation time between 1.38 to 87.59.

F	r_{ADD} (%)	r_{MPY} (%)	G_t	Δ_{acc}
5	0	14.3	65.59	$< 0.1\%$
5	0	14.3	66.01	$< 0.1\%$
5	0	14.3	65.71	$< 0.1\%$
5	0	28.6	84.44	$< 0.1\%$
5	0	28.6	83.37	$< 0.1\%$
5	0	28.6	87.59	$< 0.1\%$
8	20	0	1.38	5.69%
5(AAM) - 8(ACA)	20	14.3	52.24	0.578 %

Table 1: ADSE for the stereomatching algorithm. Simulation time gain G_t and quality degradation Δ_{acc} of the proposed approach compared to a BALL simulation for different configurations r_{ADD} and r_{MULT}

5 CONCLUSION

This paperhas proposed a simulator of inexact arithmetic operators that simplifies the modeling of the approximation error to fasten the simulation. The "Fast and Fuzzy" simulator explores the functional accuracy of operators in potentially large applications. The simulator is demonstrated to induce an acceptable loss of simulation accuracy in exchange for large simulation speedups. Simulation speedups of 3.5× and 44× are observed respectively for an inexact 16-bit adder and an inexact 16-bit multiplier.

This project has received funding from the French Agence Nationale de la Recherche under grant ANR-15-CE25-0015 (ARTEFaCT project).

REFERENCES

[1] G Alefeld. 1999. Interval arithmetic tools for range approximation and inclusion of zeros. In *Error Control and Adaptivity in Scientific Computing*. Springer, 1–21.

[2] Semiconductor Industries Association and Semiconductor Research Corporation. 2015. Rebooting the IT Revolution, a Call for Action. https://www.src.org/newsroom/rebooting-the-it-revolution.pdf. (2015).

[3] Benjamin Barrois, Olivier Sentieys, and Daniel Menard. 2017. The hidden cost of functional approximation against careful data sizing: a case study. In *Proceedings of DATE*. European Design and Automation Association, 181–186.

[4] Vincent Camus, Jeremy Schlachter, and Christian Enz. 2016. A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 127.

[5] Claire Fang Fang, Rob A Rutenbar, Markus Püschel, and Tsuhan Chen. 2003. Toward efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling. In *Proceedings of the 40th annual Design Automation Conference*. ACM, 496–501.

[6] Jiawei Huang and John Lach. 2011. Exploring the fidelity-efficiency design space using imprecise arithmetic. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*. IEEE, 579–584.

[7] Jiawei Huang, John Lach, and Gabriel Robins. 2011. Analytic error modeling for imprecise arithmetic circuits. *Proc. SELSE* (2011).

[8] Honglan Jiang, Jie Han, and Fabrizio Lombardi. 2015. A comparative review and evaluation of approximate adders. In *Proceedings on Great Lakes Symposium on VLSI*. ACM.

[9] Yongtae Kim, Yong Zhang, and Peng Li. 2013. An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press.

[10] Sana Mazahir, Osman Hasan, Rehan Hafiz, and Muhammad Shafique. 2017. Probabilistic Error Analysis of Approximate Recursive Multipliers. *IEEE Trans. Comput.* (2017).

[11] Sana Mazahir, Osman Hasan, Rehan Hafiz, Muhammad Shafique, and Jorg Henkel. 2017. Probabilistic error modeling for approximate adders. *IEEE Trans. Comput.* 66, 3 (2017), 515–530.

[12] Judicaël Menant, Muriel Pressigout, Luce Morin, and Jean-Francois Nezan. 2014. Optimized fixed point implementation of a local stereo matching algorithm onto C66x DSP. In *Design and Architectures for Signal and Image Processing (DASIP), 2014 Conference on*. IEEE.

[13] Jongsun Park, Jung Hwan Choi, and Kaushik Roy. 2010. Dynamic bit-width adaptation in DCT: an approach to trade off image quality and computation energy. *IEEE transactions on very large scale integration (VLSI) systems* 18, 5 (2010).

[14] Daniel Scharstein and Richard Szeliski. 2003. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, Vol. 1. IEEE.

[15] Changchun Shi and Robert W Brodersen. 2004. A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary inputs. In *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, Vol. 3. IEEE, III–373.

[16] Lan-Da Van, Shuenn-Shyang Wang, and Wu-Shiung Feng. 2000. Design of the lower error fixed-width multiplier and its application. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47, 10 (2000).

[17] Ajay K Verma, Philip Brisk, and Paolo Ienne. 2008. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Proceedings of the conference on Design, automation and test in Europe*. ACM.

[18] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004).

[19] Bernard Widrow and István Kollár. 2008. Quantization noise. *Cambridge University Press* 2 (2008).

[20] Yi Wu, You Li, Xiangxuan Ge, and Weikang Qian. 2017. An Accurate and Efficient Method to Calculate the Error Statistics of Block-based Approximate Adders. *arXiv preprint arXiv:1703.03522* (2017).