

Algorithm Level Timing Speculation for Convolutional Neural Network Accelerators

Thibaut Marty, Tomofumi Yuki, Steven Derrien

▶ **To cite this version:**

Thibaut Marty, Tomofumi Yuki, Steven Derrien. Algorithm Level Timing Speculation for Convolutional Neural Network Accelerators. [Technical Report] RT-0500, Univ Rennes, Inria, CNRS, IRISA, France. 2018, pp.1-17. hal-01811231

HAL Id: hal-01811231
https://inria.hal.science/hal-01811231

Submitted on 8 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

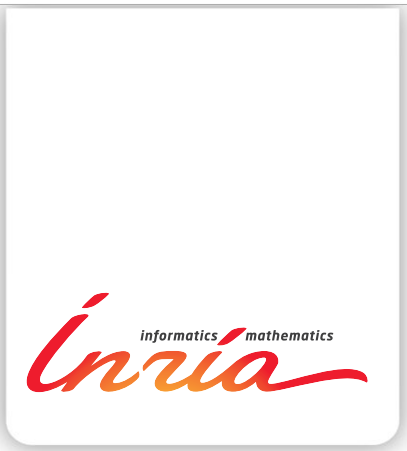


Algorithm Level Timing Speculation for Convolutional Neural Network Accelerators

Thibaut Marty, Tomofumi Yuki, Steven Derrien

TECHNICAL REPORT
N° 0500
June 2018
Project-Team Cairn

ISSN 0249-0803 ISRN INRIA/RT--0500--FR+ENG



Algorithm Level Timing Speculation for Convolutional Neural Network Accelerators

Thibaut Marty, Tomofumi Yuki, Steven Derrien

Project-Team Cairn

Technical Report n° 0500 — June 2018 — 17 pages

Abstract: In this paper, we propose a technique for improving the efficiency of hardware accelerators based on timing speculation (overclocking) and fault tolerance. We augment the accelerator with a lightweight error detection mechanism to protect against timing errors, enabling aggressive timing speculation. We demonstrate the validity of our approach for the convolution layers in Convolutional Neural Networks (CNN). We present an implementation of a fault-tolerant CNN accelerator combined with the lightweight error detection for convolution layers. The error detection mechanism we have developed works at the algorithm level, based on algebraic properties of the computation, allowing the full implementation to be realized using High-Level Synthesis tools. We use a set of Zybo boards to experimentally demonstrate that overclocking boosts the frequency by 17-36% with low chances of error, and that the infrequent errors can be detected with a negligible overhead (only 1000 LUTs).

Key-words: Algorithm Based Fault Tolerance, Timing speculation, High-level synthesis, Convolutional Neural Network

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Spéculation temporelle algorithmique pour accélérateurs de réseaux de neurones convolutifs

Résumé : Dans cet article, nous proposons une technique pour améliorer l'efficacité d'accélérateurs matériels basée sur la spéculation temporelle (overclocking) et la tolérance aux fautes. Nous proposons d'augmenter l'accélérateur avec un mécanisme de détection d'erreur léger pour le protéger contre les erreurs temporelles afin de permettre un overclocking agressif. Nous démontrons la validité de notre approche pour les couches de convolution des réseaux de neurones convolutifs (CNN). Nous présentons une implémentation d'accélérateur combiné avec une détection d'erreur légère pour les couches de convolution. Le mécanisme de détection d'erreur que nous avons développé fonctionne au niveau algorithmique : il est basé sur des propriétés algébriques du calcul, ce qui permet une implémentation intégralement avec des outils de synthèse de haut niveau (HLS). Nous utilisons un ensemble de cartes Zybo pour montrer expérimentalement que nous pouvons accroître la fréquence de 17 à 36% avec une faible probabilité d'erreur et que ces erreurs rares sont détectées avec un surcout négligeable (seulement 1000 LUT).

Mots-clés : Tolérance aux fautes au niveau algorithmique, spéculation temporelle, synthèse de haut niveau, réseaux de neurones convolutifs

1 Introduction

The use of embedded systems for various computing tasks is already widespread, and will continue to grow. The key challenge for hardware designers is to produce designs that are efficient, in terms of many metrics such as speed, energy, area cost, and so on, under limited time (short time-to-market). In combination with the increasing maturity of High-Level Synthesis tools, we have the opportunity to explore many system-level design choices in search for efficient designs.

Circuit-level timing speculation, also known as overclocking, is one possible approach to boost the efficiency of such hardware. However, timing speculation may lead to incorrect/corrupted results due to timing anomalies that typically occur within long combinational paths. For carry-chain based arithmetic circuits, these long paths contribute to the most significant bits of the results of an operation. As a consequence, such timing errors cause large numerical errors in the computation [1], [2], [3]. Although many applications are known to be robust to noise (i.e., frequent errors with small amplitude), occasional large errors can have devastating effect even for such applications.

Furthermore, the frequency of error depends on a number of factors, including the intensity of overclocking, operating temperature, variability within and across boards, input data, and so on. This makes it extremely difficult to determine a "safe" overclocking speed analytically or empirically. Therefore, overclocking must be applied conservatively or the infrequent large errors must be tolerated by the application.

In this work, we propose to combine timing speculation with lightweight error-detection to make overclocking a viable option. Error-detection is necessary to prevent the high impact errors from affecting the final output, and it must be lightweight so that the gains by overclocking is not nullified. Although many low level error detection techniques exist, they either have prohibitive area or performance overhead [4], [5] or do not provide enough error coverage [6]. We therefore propose a higher level error detection scheme by building on earlier results on Algorithm Based Fault Tolerance [7]. ABFT offer lightweight error detection techniques, and are widely used in High Performance Computing as a protection from both soft and hard errors [8], [9].

We use Convolutional Neural Networks as a case study to demonstrate our approach. CNN is a variant of multi-layered neural networks that are known to work well for image/video processing where the main idea is to construct features from local information through convolutions. CNN models used in state-of-the-art applications are computation intensive and often need to be accelerated on GPUs or FPGAs to achieve high performance and/or obtain better energy efficiency [6], [10], [11], [12], [13], [14], [15]. The core computations of CNNs have abundant parallelism, both task-level and fine-grained, that can be efficiently mapped to these accelerators. Furthermore, CNNs are known to be tolerant to noise. The reasons above make CNN an interesting class of computation to target.

Specifically, our contributions are the following:

- An in-depth quantitative analysis of the impact of overclocking on the performance and accuracy of CNN accelerators on FPGAs.
- A low overhead error detection for convolution layers in CNNs based on algorithmic properties.
- An implementation taking advantage of the above.

The remainder of this paper is organized as follows. Section 2 provides background on timing speculation techniques and CNNs. We describe our proposed accelerator with lightweight error detection in Section 3. We demonstrate the approach with a prototype implementation in

Section 4, and then discuss our results and related work in Section 5. We conclude and give directions for future work in Section 6.

2 Background

In this section, we introduce the background of our work.

2.1 Timing Speculation through Overclocking in FPGAs

The minimum clock period at which a given FPGA design is expected to work is obtained from a static timing analysis. This analysis assumes the worst case scenario, and hence the design may be operated on a slightly higher operating frequency without much risk of observing an incorrect behavior. However, this technique, widely known as *overclocking* or *timing speculation*, also has many pitfalls:

- Variability among chips and operating conditions makes it difficult to determine how much overclocking can be tolerated. The fact that errors do not manifest often (or the inability to observe errors in a given setup) does not mean that an error will never happen.
- The impact of timing errors on the circuit output is difficult to evaluate a priori. Unlike errors arising from truncation/quantization, the impact is not limited to least significant bits. Thus, it may result in large numerical errors, which may compromise the design functionality.

There is little work on the impact of overclocking on FPGA performance, and its consequences to the circuit output. A notable exception is the work by Shi et al. [3] that compares the performance/accuracy trade-off obtained by reduced arithmetic precision (through quantization) with that by overclocking. They have shown that overclocking can give gains competitive to quantization. However, the quality metric used was a statistical measure (SNR) that can mask the difference between frequent, low amplitude, error; and infrequent, high amplitude, error. Li et al. [2] show that even a single bit flip in higher order bits can cause CCNs to miss-classify images. We have confirmed this result with out prototype impementation as reported in Section 4.6. These results show that infrequent large errors have large impact on CNN outputs, necessitating error detection to be combined with overclocking.

2.2 Convolutional Neural Networks

In this paper, we are interested in the forward pass of CNNs, i.e., when a trained network is applied to new inputs. Accelerating the training of NNs is an interesting topic on its own, and is not in the scope of this paper.

In a typical configuration, a forward pass of CNNs process three-dimensional matrices in a pipelined manner through multiple *layers*. The input is usually an image, having its height and width plus the color being the third dimension, often called depth. For classification, the final output is a one-dimensional vector of length M indicating the likelihood of each category, which can be viewed as a $1 \times 1 \times M$ matrix.

Layers can be of two different types:

Convolution layers These act as local feature extractors. Given a $P \times Q \times N$ input matrix x, it computes a $R \times C \times M$ output y. Each of the M two-dimensional outputs are computed as a three-dimensional convolution over x with a kernel (weights) of size $K \times K \times N$. The convolution

may be strided by some factor S. The R and C dimensions of the output matrix may become smaller than the input for a non-unit stride, or depending on the padding of the boundaries (and also as a result of sub-sampling layers mentioned below). Different layers take different values of the parameters described - the output $R \times C \times M$, called feature maps, can be viewed as an "image" where the depth is replaced with extracted features.

Fully-Connected layers These are usually employed as the last stages of the pipeline where the size of the input matrix has become significantly smaller than the original image. They perform a collection of dot-products with weights and produce a one-dimensional vector as outputs. They are identical to the hidden layers in regular ANNs and can be computed as matrix-matrix products.

Additionally, activation functions (e.g., rectifier) or sub-sampling (e.g., max-pooling) may be considered as layers in CNN. Since these layers are inexpensive and can be merged with the preceding layer, we do not discuss them in this paper.

State-of-the-art CNN model are known to be computationally demanding (a single run of a VGG16 model requires more than 30 GOP), with more than 90% of the computational workload spent on the convolutions layers. Our approach aims at accelerating the convolution layer.

Given a $P \times Q \times N$ input matrix x and $K \times K \times N \times M$ matrix holding the weights w, a convolution layer outputs a $R \times C \times M$ matrix y through the following equation:

$$y_{r,c,m} = \sum_{n=0}^{N-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} x_{Sr+i,Sc+j,n} \cdot w_{i,j,n,m} \tag{1}$$

3 Proposed Approach

In this section, we first describe our accelerator from a system level perspective, and explain how we use timing-speculation (overclocking) to improve its computational performance. We then describe our lightweight error detection algorithm—the corner stone of our speculation technique—and describe how the error detection is mapped to hardware.

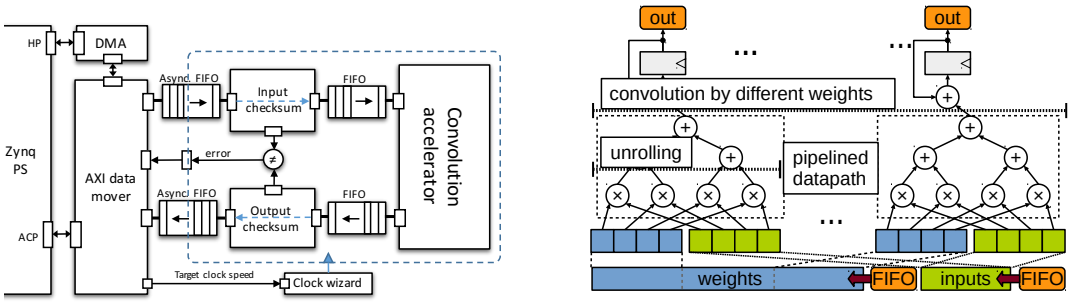
3.1 Algorithm Level Timing Speculation

Our approach builds on our ability to detect errors impacting the result of a convolution layer. This error detection mechanism uses two checksums, one computed from the output, and another computed directly from inputs, which we refer to as output-checksum and input-checksum, respectively. Note that it uses only two checksums, unlike matrix multiplication ABFT. This allows to reduce the checksum computation complexity, thus reducing the error detection overhead.

We use the same system level architecture as described by Zhang et al. [14] that use a decoupled access/execute model in the form of macro-pipelining. The actors pipeline the computation in blocks called *tiles* that partition the computation into smaller chunks to fit on-chip memory. Our speculative execution scheme therefore also operate at the tile level.

Our modified architecture is depicted in Figure 1 where the notables differences from the original [14] are:

- Data exchange between our accelerator and the rest of the system is based on asynchronous FIFOs to enable a different clock domain for our convolution accelerator. Note that data-transfers to/from memory are not overclocked, and are therefore error free.



(a) System-Level overview.

(b) The convolution kernel based on the design by Zhang et al. [14]

Figure 1: Timing-Speculative Convolution Accelerator. At the system-level, we have a macro-pipeline decoupling the I/O with the main kernel, equipped with a clock management hardware. The convolution kernel has three sources of parallelism. The main convolution (or dot-product) has ample parallelism, which is used as the fine-grained parallelism (unrolling in HLS). This datapath is also aggressively pipelined to process different inputs. Furthermore, this datapath itself is replicated for convolutions by different weights to the same input. The factor of unrolling and/or replication controls the throughput of the accelerator.

- Two additional components are used to compute the input-checksum and output-checksum on the data stream flowing into/out of the accelerator. How the checksums are computed is described later in this section.
- Overclocking is enabled through a clocking wizard component which is managed in software. In our setup, the clock wizard offers a 0.1 MHz resolution.

The efficiency of speculation depends on two factors:

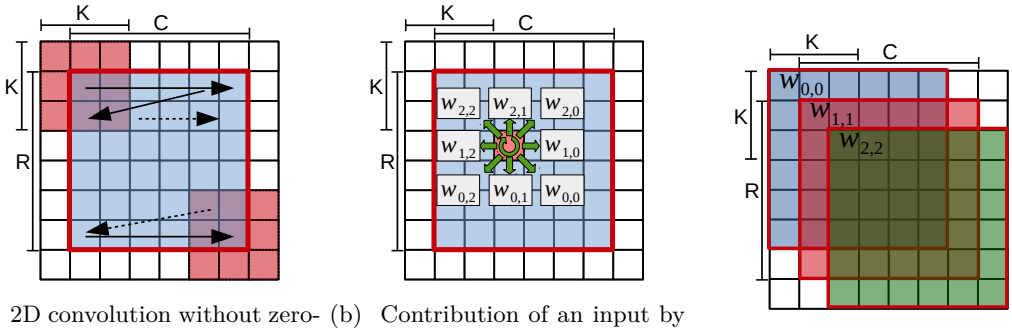
- The overhead (in terms of area/latency) of error detection.
- The performance penalty caused by a misspeculation, which corresponds to the re-computation cost weighted by its probability of occurrence.

We show in Section 4 with our prototype implementation that the error detection itself does not incur any performance overhead, and has negligible area overhead. We discuss the cost of misspeculation in the next section.

3.2 Failure Recovery Cost

Once tile computation has completed, the accelerator compares the input-checksum with the output-checksum. If checksums do not match, the tile is recomputed at normal (safe) frequency. An important point is that the execution of all tiles can be done in parallel (and therefore in any order). A faulty tile does hence not stall the accelerator macro-pipeline : its output is simply discarded. The faulty tile only needs to be later fed back to the pipeline for a safe re-execution.

When the faulty tile is reprocessed, the clock wizard must be reprogrammed twice to run the tile at some lower frequency. On our system, the total reprogramming T_p cost is less than $T_\mathrm{p} = 80\,\mu\mathrm{s}$ while the execution of a tile (without overclocking) T_s ranges from $100\,\mu\mathrm{s}$ to $3300\,\mu\mathrm{s}$. The total cost of failure recovery in our design is $2T_\mathrm{p} + T_\mathrm{s}$, we can conclude that the cost of misspeculation is negligible as long as the error rate is low.



(a) 2D convolution without zero-padding (b) Contribution of an input by each weight (c) Input data groups by weight

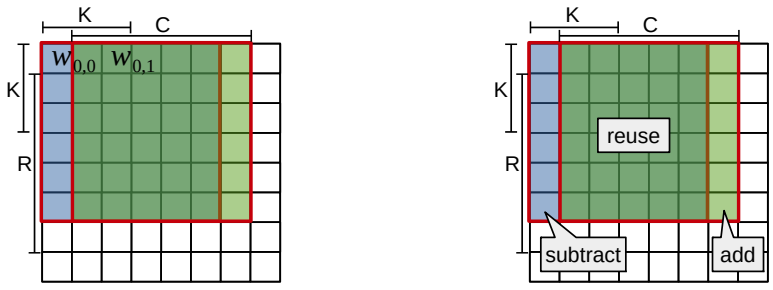
Figure 2: Illustration of the factorization for 2D case. As depicted in Figure 2a, 2D convolution can be viewed as a dot-product between the weights and the neighboring inputs with a sliding window. An alternative view shown in Figure 2b is that an input value is used to compute K^2 output values, contributing to each of them through multiplication by different weights. We can thus group the input data into (overlapping) subsets based on the weights, which is what is shown for three weight values in Figure 2c. Since sum of convolution outputs is completely linear, the multiplication can be factorized to save work.

The rest of the section describes our checksum computation technique. We start with a simpler case for 2D convolution kernel, and then generalize to convolution layers in CNNs.

3.3 Intuition with 2D Convolution

The 2D case is missing the depth dimension in the processed matrices, but the main ideas carry over to the 3D case. Given the 2D output y, the output-checksum is:

$$\sigma = \sum_{r=0}^{R} \sum_{c=0}^{C} y_{r,c}$$



(a) Two neighboring groups (b) Reuse pattern

Figure 3: The reuse between two input groups corresponding to weights $w_{0,0}$ and $w_{0,1}$. The sum of all elements in group $w_{0,1}$ can be computed by addition/subtraction of columns from that of $w_{0,0}$, instead of repeating $R \times C$ additions.

Substituting the definition of y (Eqn. 1 without m, n and $S = 1$) gives the direct computation from the inputs:

$$\rho = \sum_{r=0}^{R} \sum_{c=0}^{C} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} (x_{r+i,c+j} \cdot w_{i,j})$$

The goal is to simplify the computation of ρ so that the checksum comparison can be performed with reduced cost.

The additional two-dimensional summation provides two sources for simplification. The combination of the two simplifications described below reduces the cost of computing the checksum for 2D case from $O(RCK^2)$ to K^2 multiplications and $O(RC)$ additions.

3.3.1 Factorization

Multiplications can be factored out to eliminate RC multiplications. This may be viewed as a reordering of the summations followed by factorization:

$$\rho = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} w_{i,j} \left(\sum_{r=0}^{R} \sum_{c=0}^{C} x_{r+i,c+j} \right)$$

A graphical illustration is given in Figure 2. This reduces the number of multiplications from K^2RC to K^2.

3.3.2 Reuse in Summations

The groups of summations after factorization have significant overlaps, which can be used to reduce the number of additions. This simplification concerns the computation of the inner two summations:

$$X_{i,j} = \sum_{r=0}^{R} \sum_{c=0}^{C} x_{r+i,c+j}$$

Note that each value of X is a summation over slightly different regions of x due to the offsets by i and j. These values of X takes K^2RC additions when computed naïvely. However, once a value of X for a specific instance of i, j is computed, the remaining instances can be computed by only $O(C)$ or $O(R)$ additions as explained in Figure 3.

There are multiple ways to rewrite the definition of X to take advantage of this reuse. One example is as follows:

$$X_{i,j} = \begin{cases} \sum_{r=0}^{R} \sum_{c=0}^{C} x_{r,c} & : \begin{matrix} i = 0 \\ j = 0 \end{matrix} \\ X_{i-1,j} + \sum_{c=0}^{C} x_{R-1+i,c+j} - \sum_{c=0}^{C} x_{i-1,c+j} & : i > 0 \\ X_{i,j-1} + \sum_{r=0}^{R} x_{r+i,C-1+j} - \sum_{r=0}^{R} x_{r+i,j-1} & : \begin{matrix} i = 0 \\ j > 0 \end{matrix} \end{cases}$$

In the above, the $R \times C$ summation for $X_{0,0}$ is performed first. Then the remaining instances of $X_{i,j}$ is computed by addition and subtraction of one row/column. The $R \times C$ summation is not repeated for all each $X_{i,j}$ ($K \times K$ instances) reducing the complexity by $O(K^2)$ in exchange for $2R$ or $2C$ additions.

3.4 Lightweight Checksum for 3D Convolution Layers

For the 3D case, the output y has the third dimension corresponding to the different kernels applied to the input. The checksum is over all three dimensions of the output:

$$\sigma = \sum_{m=0}^{M-1} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} y_{m,r,c} \tag{2}$$

Substituting Eqn. 1 (again, assuming unit stride) gives the direct checksum computation from the inputs:

$$\rho = \sum_{m=0}^{M-1} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} \left(\sum_{n=0}^{N-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} x_{n,r+i,c+j} \cdot w_{m,n,i,j} \right)$$

Reordering of the summations permits the factorization of the multiplication by weights:

$$\rho = \sum_{n=0}^{N-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \left(\left(\sum_{r=0}^{R-1} \sum_{c=0}^{C-1} x_{n,r+i,c+j} \right) \cdot \sum_{m=0}^{M-1} w_{m,n,i,j} \right)$$

Note that in the 3D case, the different convolution kernels applied to the same input (the m dimension) can be first added together, since m is invariant to the expression involving x.

$$\rho = \sum_{n=0}^{N-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \left(X_{n,i,j} \cdot \sum_{m=0}^{M-1} w_{m,n,i,j} \right) \tag{3}$$

where X takes the same form as the 2D case, except for the additional n dimension, which does not have any reuse.

We have assumed unit stride to simplify the presentation. We do not give the full detail due to space reasons, but the same principles apply to non-unit strides as well. The main difference is in the simplification of the summations. With non-unit strides, the input groups (Figure 2c) also becomes strided. This reduces the reuse across the input groups, but this is natural since non-unit stride essentially corresponds to sub-sampling, i.e., the number of uses of each input for computing the output is reduced.

3.5 Implementation of Checksum Calculations

The output-checksum is implemented as a simple accumulator over convolution outputs, with a small area overhead compared to the rest of the datapath. The hardware component responsible for computing the input-checksum (Eqn. 3) is more costly, as it involves a multiplier, storage for partial sums, and non-trivial control logic. As depicted in Figure 1a, this component operates directly on the input stream.

Both of these components are significantly less complex compared to the main kernel. The main performance constraint is to ensure that the data processing rate matches that of the convolution kernel. The amount of parallelism in the convolution kernel affects the rate of input consumption as well as output production. The checksum calculations need sufficient parallelism to keep up with this rate. This ensures that the input-checksum, computed in parallel with the main convolution computation, do not impact the overall latency of the accelerator.

The rates of inputs/outputs may be viewed as the number of data in a single FIFO element. As the throughput of the convolution kernel increases, the input/output elements become wider vectors holding more values. The checksum calculations must then operate on vectors of inputs and outputs.

The output-checksum has a trivial parallelization to process the output vectors. Since the output-checksum is an accumulation, the computation is independent except for the aggregation at the end.

4 Proof-of-Concept

We present our empirical study in this section. The study consist of two parts: (i) quantifying area overhead in contrast to adding more parallelism (ii) study of variability.

4.1 Experimental Platform

The whole accelerator was designed using SDSoC (2016.3), demonstrating the suitability of our approach to modern FPGA tools that operate at higher level of abstractions.

We targeted two boards from the Xilinx Zynq-7000 family in our experiments: Zybo and ZC706 (XC7Z045). The motivation for using Zybo is because we have more than 8 boards available for our experiments. This allowed us to perform a large number of experiments to quantify the error rate with different frequencies in high resolution, and to study the degree of variability across boards. ZC706 is significantly larger than Zybo, providing design points that are more relevant for use cases that require high throughput.

The hardware designs used in this section targets the fifth layer in the AlexNet [15] CNN architecture with the following configuration: $N = 192$, $M = 128$, $R = C = 13$, $K = 3$, and $S = 1$. A preliminary experience showed that this layer was the one that most impact classification rate when impacted by timing errors. Note that the third and fourth layers of AlexNet also have similar configurations with bigger N and/or M.

4.2 Accelerator Design Space

The accelerator design have a number of parameters:

- Tile size (T_n, T_m, T_r, T_c). As explained in Section 3.1, our CNN accelerator processes convolution layers in smaller units called tiles. These parameters define the subset of the output calculation to be executed by a single execution of the accelerator. The corresponding indices in Eqn. 1 are partitioned by the values specified by these parameters to define a tile. These parameters control the computation to communication ratio of the accelerator.
- Unrolling Factor (U_n, U_m). These factors control the degree of parallelism. The pipelined datapath performs U_n instances of the multiplication each cycle, which is controlled as the unroll factor of the innermost loop in HLS. This datapath is replicated U_m times to process convolutions by different kernels in parallel.
- Word Length (WL). We use fixed-point arithmetic in our accelerator, which naturally give rise to another tuning parameter: number of bits used to represent each number. We use the same word length for all variables.

All designs are synthesized with highest target frequency that can be met. This ensures that excessive slacks in the timing are avoided.

We explored various combinations of word lengths and unroll factors. They both have significant influence on the area cost of the convolution accelerator without our proposed error detection. The tile sizes were selected to be the largest tile that can accommodate the unroll factors we tried. Having larger tiles increases BRAM usage forcing some of our designs to not fit on the target board.

Table 1: Area results for different design parameters on ZC706 and Zybo. ABFT enabled designs include the cost for all components: the convolution kernel, input-checksum, and output-checksum. The area overhead of ABFT is negligible across all designs - at most 3.7% additional LUTs for ZC706 and 5.9% for Zybo. BRAM/DSP/GOP are unaffected. GOPS is giga operations per second for executing 96 tiles, including the latency of read/write actors.

(a) ZC706 area results for tiles of size $128 \times 192 \times 13 \times 13$.

WL	Um×Un	Type	BRAM	DSP	FF	LUT	GOPS
8	8×16	base	30%	7%	0.7%	3.7%	22.1
		ABFT	30%	7%	1.3%	6.0%	
	16×16	base	32%	14%	1.1%	6.7%	41.6
		ABFT	32%	14%	1.8%	9.0%	
12	8×16	base	43%	14%	0.3%	3.6%	22.5
		ABFT	45%	14%	1.1%	6.8%	
	16×16	base	45%	28%	0.3%	6.1%	42.2
		ABFT	45%	28%	1.2%	9.1%	
	32×16	base	49%	57%	0.5%	10.7%	75.2
		ABFT	49%	57%	1.3%	13.8%	
16	8×16	base	57%	14%	0.3%	4.6%	22.5
		ABFT	59%	14%	1.4%	8.3%	
	16×16	base	60%	28%	0.4%	7.9%	42.2
		ABFT	60%	28%	1.4%	11.6%	
	32×16	base	63%	57%	0.6%	14.0%	75.2
		ABFT	63%	57%	1.6%	17.6%	
Available			1090	900	437K	219K	

(b) Zybo area results for tiles of size $32 \times 32 \times 13 \times 13$.

WL	Um×Un	Type	BRAM	DSP	FF	LUT	GOPS
8	4×8	base	37%	20%	1.8%	5.8%	5.8
		ABFT	37%	20%	3.7%	10.3%	
	8×8	base	67%	60%	2.9%	6.4%	10.5
		ABFT	67%	60%	4.7%	10.9%	
	16×8	base	100%	80%	1.8%	17.1%	17.7
		ABFT	100%	80%	3.7%	21.6%	
12	4×8	base	47%	40%	2%	6.9%	5.8
		ABFT	47%	40%	4.1%	12.2%	
	8×8	base	73%	80%	2%	9.4%	10.5
		ABFT	73%	80%	4.1%	14.7%	
	16×8	base	100%	100%	6.1%	69.8%	17.7
		ABFT	100%	100%	8.3%	74.3%	
16	4×8	base	47%	40%	2.2%	8%	5.8
		ABFT	47%	40%	4.6%	13.9%	
	8×8	base	73%	80%	2.2%	11.5%	10.5
		ABFT	73%	80%	4.6%	17.2%	
Available			120	80	35K	18K	

4.3 Area Overhead

In our approach, we add additional hardware for detecting errors to enable more aggressive overclocking. Thus, the area overhead must be smaller than simply increasing the parallelism using the additional resource spent on error detection. Otherwise, it will be simpler and better to use this additional hardware to improve the accelerator performance by adding more parallelism without relying on overclocking and error detection.

We can expect significantly lower cost compared to the main computation due to the algorithmic simplifications presented in Section 3. In particular, the simplification eliminates *a factor of* $R \times C$ multiplications that translates to 169 times less multiplications for Alexnet's layer 5.

These multiplications are computed by a single multiplier implemented with LUTs, as other operators. LUTs are also used for needed memories. Thus, no BRAM or DSP are used for the input-checksum computation.

As expected, the area overhead is extremely small on all the designs across a number of designs we have synthesized, reported in Table 1. For ZC706 that can fit larger tiles than Zybo, the overhead is at most 3.7%. This is negligible in contrast to the proportional area requirement by simply adding additional parallelism, which corresponds to higher unroll factors (Um and Un).

4.4 Cost of Miss-speculation

4.5 Study of Variability

In this section, we report our empirical study on two sources of variability. The main message is that due to many different sources of variability, statically selecting an overclocking frequency must be done conservatively. Our online error detection mechanism provides an opportunity to perform overclocking more aggressively.

4.5.1 Inter-Board Variability

Process variation is a well-known phenomenon that causes the same circuit to behave differently across boards. This gives significant variations across boards with respect to timing errors as well. As illustrated in Figure 4,

4.5.2 Data Variability

Timing errors are expected to be sensitive to data. This is because the critical path is data dependent. For typical arithmetic operations, inputs with small absolute values (i.e., high order bits are zero) have shorter critical paths. We empirically quantify this variability by executing our accelerator with input data masked to have higher order bits set to zero.

The precise setup is as follows. We take the design for Zybo synthesized for 16 bits input data and ran with varying frequencies for 400 images each. The input data were masked to have varying effective width (dynamic range). We collect the number of erroneous values in the output image with respect to the error-free execution using masked input data.

The result presented in Figure 5 shows the expected behavior; input data with smaller effective width result in errors at a much higher frequency.

4.5.3 Importance of Dynamic Adjustment

We have empirically study two sources of variability. The timing error behavior changes due to many factors such as temperature, data dynamic range, tile size, and target frequency set during

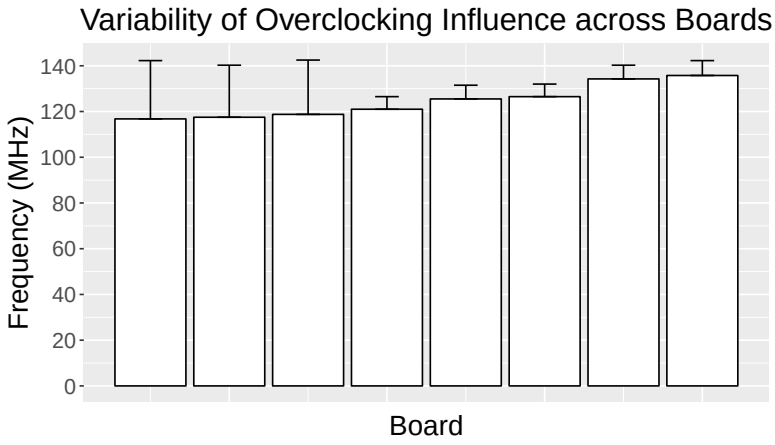


Figure 4: This figure shows how inter-board variability impacts the achievable overclocking rate. Each bar correspond to the highest overclocking rate achievable of a given Zybo board (a given overclocking rate is considered achievable if it translates as a tile error rate lower than 0.2%). In addition, we also illustrate how this maximum overclocking is impacted by data variability by plotting the best-case vs worst case interval for different data masking scenarios

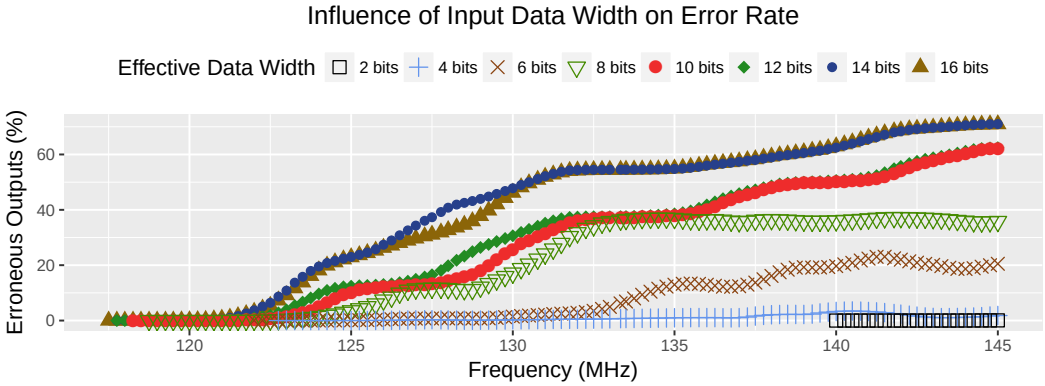


Figure 5: Percentage of corrupted outputs as a function of operating frequency over 400 images. Effective Data Width of n means $16 - n$ most significant bits were set to zero. Data points with zero observed error are not plotted - the bottom points have small but non-zero error rate.

HLS to name a subset. The static timing analysis gives a frequency where the timing can be guaranteed to be met. However, the analysis is necessarily conservative due to the variability and also because it must consider worst case scenarios. Our approach allows for run-time adjustment of the frequency to overclock as much as possible, without risking timing errors to influence the final output.

It may seem possible to perform an off-line calibration step before-hand for certain sources of variability, such as inter-board variation. However, the probabilistic behavior of timing errors that depends on many parameters makes it extremely challenging to find a "safe" frequency. The number of samples that needs to be collected to statistically select frequencies are likely to be prohibitively high, forcing conservative choices to be made.

In constrast, our approach enables much more aggressive overclocking for a very low area overhead (ony a few percent additional LUT utilization).

4.6 Impact on Classification Accuracy

Since neural networks are known to be tolerant to noises—frequent, low amplitude errors—one may question if the timing errors cause notable impact on the final quality of the CNN or not. Li et al. [2] have performed an extensive study for various networks including AlexNet [15] in the context of soft-errors. They show that errors in the higher order bits significantly degrades the accuracy of neural networks.

We measured the frequency of flips for each bit while processing AlexNet images with publicly available model[1] on our accelerator. The accelerator was overclocked to produce a small number of errors per image (in the order of 100 corrupted values per image). By using overclocking frequencies with small error rates, we avoid having multiple timing errors while computing a single output. We confirmed that most of the corrupted values are different from the error-free execution by one and only one bit. This gives us an empirical estimate of the likelihood of bits being flipped under moderate overclocking beyond error-free frequencies.

The empirical probability of each bit being flipped derived from 5124k erroneous values collected are summarized below:

MSB (bit 15)	bit 14	bit 13	bit 12-0 combined
27%	16%	24%	33%

This shows that there are significant chances for the high order bits to be flipped through timing errors. This makes sense as timing errors are expected in critical paths that commonly affect high order bits (e.g., carry chains).

We used a software implementation[2] to simulate the impact on classification results based on the above probability. The final impact on the result depends on the layer where the error occurs. The last two layers (4th and 5th) have the most significant influence. We observed that injecting a small number of errors (20 per image) to the last layer causes more than 10% miss-classifications when compared to error-free execution.

We have also observed that extreme overclocking basically produces random values as outputs. The impact on classification accuracy is much higher in such situations.

Note that the probability reported above is only a rough empirical estimate. We can expect them to change due to various factors: variability due to data, temperature, etc.; the low-level decisions during HLS and/or P&R; and so on. It is difficult to have precise analytical models as we do not have full control over low-level designs synthesized by HLS tools.

[1] `https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet`
[2] `https://github.com/tiny-dnn/tiny-dnn`

5 Discussion and Related Work

5.1 Comparison with Existing ABFT Techniques

Our approach builds on the ABFT technique for matrix operations [7]. In fact, the convolution layer and the fully-connected layer can be viewed as a collection of matrix products, making the classical ABFT directly applicable. Most ABFT extensions employ a variant of the original method by identifying pieces of computations that can be viewed as matrix operations [16], [17].

We emphasize that the lightweight checksum calculation proposed in this paper is not a direct application of the classical ABFT. We use algorithmic invariants in collections of convolutions to further reduce the cost of checksums. This is evident in the fact that we reduce the algorithmic complexity by two-fold, exploiting reuse in two-dimensions, whereas the original ABFT brings one-degree savings.

The algebraic properties used in ABFT relies on associativity and commutativity. Since floating-point operations are not commutative, the application of these techniques requires additional care. For instance, a small threshold value to decide that the checksums match may be needed. This is not a concern for hardware CNN accelerators, since the clear trend is to use short integers encodings for data and/or weights [18].

5.2 Other Techniques for Timing Error Detection

The potentially high impact of timing errors has led to several circuit-level techniques for its detection and correction.

The Razor technique [4] uses a dual flip-flop scheme to detect timing violation. The idea consists in delaying the clock signal by a small offset at the input of the second flip-flop and compare the two sampled values. If these two values differ, then a timing error has occurred. This approach is however not well suited to modern FPGA design flows since it often requires manual floor-planning to obtain the expected behavior [19], [20]. Another issue is that the error detection overhead can be significant when the number of flip-flops that needs to be protected in fine-grain FPGA pipelined datapath is large.

Other error detection techniques, such as those based on Residue Numbering Systems could be used. RNS protection provides low overhead error detection [21] for convolution kernels. However, they only offer limited protection against typical timing errors that impact several outputs at a time.

6 Conclusions and Future Work

In this paper, we propose timing speculation coupled with lightweight error detection as an approach to further improve the performance of hardware accelerators for CNNs. We have demonstrated the efficacity of our approach with a prototype implementation and an extensive empirical study. In addition, our approach is very well-suited for implementation in high-level design tools such as Vivado HLS/SDSoC, which is becoming more and more attractive for productivity reasons.

We believe that similar techniques can be applied to many other application domains (bioinformatics, iterative solvers, etc.) by taking advantage of existing ABFT techniques or by devising new algorithms tailored for this task. This is part of our ongoing work.

Acknowledgment

This work was partially supported by European project ARGO. Ce projet bénéficie du soutien financier de la Région Bretagne.

References

[1] S. Chaudhuri, J. S. J. Wong, and P. Y. K. Cheung, "Timing speculation in FPGAs: Probabilistic inference of data dependent failure rates," in *2011 International Conference on Field-Programmable Technology*, Dec. 2011, pp. 1–8.

[2] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, 2017, 8:1–8:12.

[3] K. Shi, D. Boland, and G. A. Constantinides, "Accuracy-Performance Tradeoffs on an FPGA through Overclocking," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, Apr. 2013, pp. 29–36.

[4] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36, 2003, pp. 7–.

[5] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: Software Implemented Fault Tolerance," in *Proceedings of the International Symposium on Code Generation and Optimization*, ser. CGO '05, 2005, pp. 243–254.

[6] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2015, pp. 1–6.

[7] K.-H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *Ieee transactions on computers*, vol. C-33, no. 6, pp. 518–528, Jun. 1984.

[8] Z. Chen, "Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13, 2013, pp. 167–176.

[9] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, "Algorithm-based Fault Tolerance for Dense Matrix Factorizations," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '12, 2012, pp. 225–234.

[10] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 257–260.

[11] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16, 2016, pp. 16–25.

[12] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, Jul. 2009, pp. 53–60.

[13] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ Deep Learning Accelerator on Arria 10," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, 2017, pp. 55–64.

[14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15, 2015, pp. 161–170.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1097–1105.

[16] A. Roy-Chowdhury, N. Bellas, and P. Banerjee, "Algorithm-based error-detection schemes for iterative solution of partial differential equations," *Ieee transactions on computers*, vol. 45, no. 4, pp. 394–407, Apr. 1996.

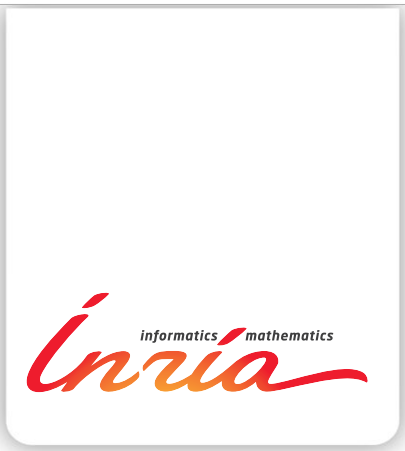
[17] G. Bosilca, A. Bouteiller, T. Herault, Y. Robert, and J. Dongarra, "Composing resilience techniques: ABFT, periodic and incremental checkpointing," *International journal of networking and computing*, vol. 5, no. 1, pp. 2–25, Jan. 10, 2015.

[18] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16, 2016, pp. 26–35.

[19] R. Ragavan, C. Killian, and O. Sentieys, "Adaptive Overclocking and Error Correction Based on Dynamic Speculation Window," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2016, pp. 325–330.

[20] A. Brant, A. Abdelhadi, D. H. H. Sim, S. L. Tang, M. X. Yue, and G. G. F. Lemieux, "Safe Overclocking of Tightly Coupled CGRAs and Processor Arrays using Razor," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, Apr. 2013, pp. 37–44.

[21] S. J. Piestrak and P. Patronik, "Design of Fault-Secure Transposed FIR Filters Protected Using Residue Codes," in *2014 17th Euromicro Conference on Digital System Design*, Aug. 2014, pp. 575–582.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803