electronics

MDPI

Article

HEAP: A Holistic Error Assessment Framework for Multiple Approximations Using Probabilistic Graphical Models

Jiajia Jiao

College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China; jiaojiajia@shmtu.edu.cn

Received: 27 January 2020; Accepted: 20 February 2020; Published: 22 February 2020

Abstract: Approximate computing has been a good paradigm of energy-efficient accelerator design. Accurate and fast error estimation is critical for appropriate approximate techniques selection so that power saving (or performance improvement) can be maximized with acceptable output quality in approximate accelerators. In the paper, we propose *HEAP*, a Holistic Error assessment framework to characterize multiple Approximate techniques with Probabilistic graphical models (PGM) in a joint way. *HEAP* maps the problem of evaluating errors induced by different approximate techniques into a PGM issue, including: (1) A heterogeneous Bayesian network is represented by converting an application's data flow graph, where various approximate options are {precise, approximate} two-state X*-type nodes, while input or operating variables are {precise, approximate, unacceptable} three-state X-type nodes. These two different kinds of nodes are separately used to configure the available approximate techniques and track the corresponding error propagation for guaranteed configurability; (2) node learning is accomplished via an approximate library, which consists of probability mass functions of multiple approximate techniques to fast calculate each node's Conditional Probability Table by mechanistic modeling or empirical modeling; (3) exact inference provides the probability distribution of output quality at three levels of precise, approximate, and unacceptable. We do a complete case study of 3×3 Gaussian kernels with different approximate configurations to verify *HEAP*. The comprehensive results demonstrate that *HEAP* is helpful to explore design space for power-efficient approximate accelerators, with just 4.18% accuracy loss and 3.34×10^5 speedup on average over Mentor Carlo simulation.

Keywords: error assessment; approximate computing; probabilistic graphical models; multiple approximations

1. Introduction

Power and energy efficiency have motivated the coming forth of new approximate techniques for designing accelerators for applications with inherent resilience to errors [1]. By relaxing the quality of output results in image processing, data mining, pattern recognition applications, imprecise calculation, storage and communication can achieve more power saving and performance improvement. This approximate computing paradigm can be applied in different components of application-specified accelerators with different quality-power (or performance) tradeoffs at different abstractions layers [2]. There have been a variety of approximate techniques proposed for gaining power saving or performance improvement, which cover all the sites in approximate computing [3–17], approximate storage [18–30], and approximate communication [31–36]. How to utilize these approximate techniques is a significant issue for designing a cost-effective approximate accelerator.

Error assessment (In the paper, error assessment is to evaluate the actively induced errors impacts on application output quality, where an approximate accelerator adopts some approximate techniques to gain power saving or performance. It is different from the random soft errors caused by high energy particles. Error assessment, error analysis, and error estimation are interchanged to use in the following paper) of approximate accelerators is critical for appropriate approximate techniques selection to maximize the power saving or performance improvement with acceptable output quality. Two kinds of approaches are usually used to characterize the error impacts of approximate techniques on final output quality. First is Mentor Carlo simulation to capture dynamic execution of applications exactly in approximate architectures. It often uses a large number of simulations to get the accurate statistical results [37–39]. This general-purpose approach provides dependable estimation, but requires too much time. The other is fast analytical modeling to characterize applications' output quality in the underlying approximate architectures. It calculates the output quality of approximate configuration with a few formulas very quickly. However, these analytical models are usually limited to certain approximate sites such as approximate adders or approximate multipliers for calculation [40,41], approximate SRAM/DRAM/SSD/PCM [42–45] for storage, and approximate data transmission for communication [31–36]. Particularly, there are a variety of accuracy-power/performance tradeoffs for a specified approximate component. For example, approximate GeAr [15] performs an n-bit addition using multiple sub-adders of smaller size. The most significant r-bits of the sub-adders are considered as resultant bits and used in the result, while the remaining p-bits, the pervious bits, are used to estimate the carry propagation to upper bits. Different parameters (n,r,p) can be configured as different approximate cases. Therefore, the analytical models are fast but hard extensions from local component to the whole architecture directly.

From the above analysis, analytical modeling is faster than Mentor Carlo simulation, and more suitable for early design of approximate accelerator. To break the limits of the traditional analytical modeling only for components, the probabilistic approach is a good alternative [46]. Several probabilistic methods for error estimation in approximate adders or precise scaling have been presented recently [47–49]. The error impacts of approximate adders are evaluated by a compiler-driven error propagation and analysis approach in [47]. Error propagation rules are defined according to the different instruction types, and then Depth-First Search algorithm is used to merge the probability mass functions of all nodes through traversing the application's data flow graph. Precision scaling is characterized by a Bayesian probabilistic approach to predict the relation between component-level functional approximation and application-level accuracy [48]. Each node is a three-state variable of precise, approximate, and unacceptable to represent the precise scaling well. The Bayesian based prediction of [48] is further extended to evaluate any application accuracy and support design exploration well in [49]. High accuracy and low computational time make this Bayesian probabilistic modeling attractive for assessing approximate techniques. However, the mixture of multiple approximate techniques can exploit the fine-grain quality-power tradeoffs, but has not been considered completely.

As a supplementary work to the existing probabilistic approaches, this paper proposes *HEAP*, a Holistic Error assessment framework to characterize multiple Approximate techniques with Probabilistic graphical models (PGM) for application-specified accelerator design. We firstly construct a unified Bayesian network via static application behavior analysis, and then can use mechanistic modeling or simulation driven empirical modeling for PGM node parameters learning of conditional probabilities tables (CPT). Finally, the probability of output quality at different levels can be calculated through exact inference fast and accurately. The main contributions of our work are:

(1) Unified assessment of multiple approximate techniques via PGM. We map error analysis of multiple approximations into a general PGM issue and propose a configurable framework *HEAP* for assessing various approximate techniques via PGM representation, learning, and inference. The heterogeneous Bayesian representation includes two types of nodes, which can configure approximations and track error propagation, respectively. Node learning is accomplished by mechanistic modeling or simulation driven empirical modeling to fill the CPTs of all nodes.

Exact inference can maintain the high accuracy and fast speed of error assessment for multiple approximate techniques.

(2) Design exploration of approximate accelerators through flexible tradeoffs. Using the fast and accurate *HEAP* approach, the best combination of approximate techniques can be selected to maximize the power saving or performance improvement with guaranteed output quality. The configurability of proposed *HEAP* makes approximate techniques selection easy and efficient.

(3) Comprehensive verification by a case study of 3×3 Gaussian kernel. The comprehensive results demonstrate that the proposed *HEAP* achieves 4.18% accuracy loss and 3.34×10^5 speedup on average over Mentor Carlo simulation (1,000,000 samples) and good flexibility in exploiting fine-grain quality-power tradeoffs of multiple approximate techniques. A combined approximate configuration of approximate adder, criticality-aware data cache, and dual-voltage approximate communication performs the maximum power saving under guaranteed output quality.

The structure of this paper is as follows. Firstly, introduction and related work are described in Sections 1 and 2, respectively. Next, Section 3 details the proposed framework *HEAP*, including unified PGM structure construction via static application behavior analysis, node learning for CPTs to characterize approximate techniques, and exact inference to calculate the probability of output quality at three levels. Then, the experiment configuration and results are presented in Section 4. Finally, the conclusion is drawn in Section 5.

2. Related work

In this section, we introduce the related works on a variety of approximate techniques in Section 2.1 and error estimation methods for approximate techniques in Section 2.2, respectively.

2.1. Approximate Techniques

To exploit the inherent error tolerance of applications, various approximate computing approaches are proposed via relaxing the reliability actively for higher performance and lower power consumption. According to the approximate sites, these approximate techniques can be classified into three categories:

(1) Approximate computing. Approximate computing can be achieved by software or hardware. The high-level approximate software can be exploited in a more flexible way, such as approximate aware high-level synthesis [3], neural acceleration for general approximate programs [4], optimization of approximate kernels [5], and floating point [6]. Unlike these approximate software solutions, approximate hardware techniques usually exploit the fine-grain approximation at architecture-level or circuit-level. Recent approximate architectures include SAGE [7], DES [8], SNNAP [9], Neutralizer [10], Brainiac [11], precimonious [12], neural acceleration for GPU [13], and accelerators for machine learning [14]. Approximate circuits focus on the often-used functional units such as approximate adders or approximate multipliers [15–17].

(2) Approximate storage. All layers of the memory hierarchy are covered, including cache, memory, and storage. The approximate caches aim at optimizing the access performance and reducing the cache miss overhead as well as some new types of devices, such as RFVP [18], load value approximation [19], Texture Cache [20], a tunable cache [21], STAxCache [22], Dynamic Energy-Quality [23], and Scratchpad Memory optimize [24]. Approximate memory structures are also proposed by dividing the data into the critical and uncritical regions. The uncritical data region can be set at a lower refresh rate for energy saving, or the potential of Multi-Level Cell is considered, such as Flikker [25] and DrMP [26]. Approximate storage is often for larger capacity for high-density image storage [27–30].

(3) Approximate communication. Approximate Communication techniques are required to reduce communication bottlenecks in large scale parallel systems. The recent survey of approximate communication points that most of the existing techniques related to approximate computing and storage can help to reduce the communication pressure indirectly. The compression,

value approximation, and relaxed synchronization are the popular techniques for approximate communication. The actual approximate communication architectures for parallel system have come out lately. The approximate bus [31] and network on chip architectures [32–36] are proposed for higher network throughput using the critical feature of data type information or dual scaling voltage.

All these carefully designed approximate approaches need fast and accurate error evaluation to verify their effectiveness on exchanging acceptable output quality loss for power or performance benefits.

2.2. Error Estimation Methods

Before designing application-specified approximate accelerators, approximate area recognition is needed to determine the latent approximate locations. After finishing approximate accelerator design, approximate evaluation is required to confirm the guaranteed output quality. The central task of the two aspects is fast and accurate error assessment of various approximate techniques. The error estimation methods for approximate diagram are often classified into two branches:

(1) Mentor Carlo simulation. Mentor Carlo simulation can be further classified into three categories for evaluating approximate techniques: Random, representative, and equivalent. The random way is easy to implement, but low efficiency [1], while the selective representations are usually based on application characterization such as ARC proposed by Chippa [2]. The equivalence based Relyzer [37] uses the static analysis of flow control and dynamic analysis of data flow to capture the similarity degree of different instructions for fewer simulations with fault injection and higher efficiency. On the base of the existing Relyzer, Approxilyzer [38] and gem5-approxilyzer [39] were proposed via exploiting the one bit upset affecting the application-level output quality. However, the optimized Mentor Carlo simulation is still time consuming due to unavoidable simulations.

(2) Analytical modeling. All levels in a design stack can use analytical modeling for fast error assessment. High-level program analysis approaches are presented for specific program or code. These methods almost are based on the grammar and semantics analysis, so that they can provide higher efficiency [6]. Low-level approximate circuits are modeled based on a lookup table technique, which characterizes the statistical properties of approximate hardware and uses a regression-based technique for composing statistics to formulate output quality [40,50]. An analytical error analysis approach for approximate adders can predict, evaluate, and compare the accuracy of various approximate adders and multipliers [41,51]. These existing works mostly focus on low-level error estimation or high-level program analysis.

To both keep the fast speed of analytical modeling and achieve the complete error estimation from various approximate techniques to quality of application output, probabilistic methods have been presented to address the error estimation for approximate computing diagram from a new perspective [46–49]. Preliminary information is provided on a cross-layer framework built on top of a Bayesian model designed to perform component-based reliability analysis of complex systems [46]. A compiler-driven error analysis methodology defines the computing rules of error propagation for different kinds of instruction operations and evaluates the behavior of errors generated from approximate adders in the design of approximate accelerators [47]. Another probabilistic approach predicts the relation between component-level functional approximation and application-level accuracy by precision scaling in a Bayesian network [48,49]. These probabilistic approaches inspired us to design a PGM based configurable framework, *HEAP*, for fast and accurate error assessment of multiple approximate techniques.

Compared with these probabilistic approaches, the proposed *HEAP* provides a holistic framework for multiple approximate techniques, which can be instanced as one configuration for approximate adders [47] or precision scaling [48,49]. Additionally, the case study of 3×3 Gaussian kernel and various results can prove the effectiveness of mapping error assessment problem into a general PGM issue. *HEAP* can also integrate some dynamic simulation information instead of purely mechanistic

modeling to further reduce estimation accuracy loss by 1.61%, which is up to 98–99% accuracy as is provided in [49]. Moreover, *HEAP* can assist approximate accelerator design well through selecting the mixture of multiple approximate techniques in a configurable way easily and efficiently. Therefore, to the best of our knowledge, the proposed *HEAP* is the first to tackle the error assessment of multiple approximate techniques with a joint PGM framework as Section 3 describes, and proves fast and accurate error estimation for assisting cost-effective approximate accelerators design as Section 4 shows.

3. Proposed framework *HEAP*

In this section, we detail *HEAP* the error assessment framework of multiple approximate techniques for application-specified accelerators. First, the general PGM concept is briefly introduced in Section 3.1. Next, the proposed *HEAP* framework overview is described in Section 3.2. Then, *HEAP* three components design is detailed in Section 3.3.

3.1. General PGM Concept

Generally, a complete PGM framework is composed of three parts: Representation, learning, and inference [52].

Representation. This is the fundamental and critical factor for a PGM framework and includes directed graphical models (Bayesian networks in Figure 1a) and undirected graphical models (Markov Random Fields in Figure 1b). In general, a PGM defines a family of probability distributions that can be represented in terms of a graph. Nodes in graph correspond to random variables $X_1, X_2, \ldots, X_n$; the graph structure translates into statistical dependencies (among such variables) that drive the computation of joint, conditional, and marginal probabilities of interest. For example, in Figure 1a, the joint probability or Bayesian factorization is described by the conditional probabilities in Equation (1) as follows:

$$P(X_1, X_2, X_3, X_4, X_5) = P(X_1)P(X_2|X_1)P(X_3|X_1)P(X_5|X_2)P(X_4|X_2, X_3) \quad (1)$$

Figure 1. Two PGM network types. (**a**) Bayesian network; (**b**) Markov Random Fields.

Learning phase. The learning graphic model from data is another important concept for the factorization of the distribution. Learning structure and learning parameters are the two most basic learning tasks as follows:

- Structure learning. As Figure 1a shows, the deterministic Bayesian graph determines the structure in PGM. Without the structure, the joint probability cannot be expressed by the reduced conditional probabilities for the unknown dependencies. Therefore, it is very critical to get the structure accurately for correct factorization.
- Parameter learning. The parameters denote the required CPT for the joint probabilities' calculation. Here, we assume the conditional probability will follow the Bernoulli distribution for the approximation induced bits upset. E.g., the node X_4 in Figure 1 has three parents. It depends on X_1, X_2, X_3, which means the unknown parameters determine the probability of each case.

Inference phase. Based on the available structure and node information, using a given algorithm (e.g., variable elimination and conditioning) for exact inference, we can determine the required marginal probabilities. For example, the marginal probability can be expressed as: $P(X_4 = X_5 = 0 \,|X_1 = 1) = P(X_1, X_2, X_3, X_4, X_5)$. Then, the probability can be determined from the conditional probabilities based on node parameters via the aforementioned efficient inference algorithms.

3.2. HEAP Overview

The proposed *HEAP* can solve the error assessment of multiple approximate techniques via PGM in a joint way. We choose to work within the PGM framework, because this offers unique advantages: (1) Efficiency potential because of PGM low complexity for the high dimensional dependencies converted into small factorization; (2) modularity for easily integrating existing evaluation methods of a single approximate techniques into the PGM modules; (3) perfect match due to the application-specified data flow graph fulfilling the conditional independences.

Based on the above general PGM concept, we map the error propagation due to multiple approximate techniques into the PGM problem as follows, which includes the instanced three main components based on PGM flow shows: (1) Mapping the problem into the fundamental PGM representation; (2) structure and node parameters learning to support the inference; (3) exact inference to guarantee the estimation accuracy.

As Figure 2 shows, the workflow of *HEAP* is also composed of three steps. The given input data flow graph (DFG) of specified application is firstly converted to a heterogeneous Bayesian network by mapping rules. It is noted that, if the data flow graph is not given directly, open source tool Low Level Virtual Machine (LLVM) can compile the application C/C++ source code and provide DFG through intermediate representation (IR) [53]. Based on an available DFG, a Bayesian network is constructed through the simple mapping rules. Keep the variable nodes with three states {P = precise, A = approximate, U = unacceptable}, which uses the same handling policy in [48,49]. For example, X_{1+} denotes the left sum of a and shift temporary. It has three possible states {P, A, U}. To configure the multiple approximate sources, the two states {P = precise, A = approximate} nodes are inserted. For example, X^*_{1+} becomes a new parent of X_{1+} as a new two states node. More details about the representation are depicted in Section 3.3.1.

Then, according to the network dependences between nodes and available multiple approximate techniques, the CPTs are filled with modeling results from approximate library and approximate configuration in Section 3.3.2.

Finally, variable elimination (VE) algorithm [54] uses the Bayesian network and nodes' CPT information to achieve the error distributions via calculating the marginal probability in Section 3.3.3. For example, the probability of precise output is 100%, while it is only 96% under approximate adder configuration, even lower in the mixed approximate configurations. Additionally, the power saving or performance improvement can be further considered to select the appropriate approximate configuration from possible options.

3.3. HEAP Components

Next, we introduce the details of *HEAP*'s three components respectively, including Bayesian network representation, node parameters learning, and exact reference.

3.3.1. Mapping Problem into Bayesian Network Representation

This paper is unlike our previous work [55] using PGM to solve the soft error estimation via basic Bernoulli distribution, where each node occurs one bit upset or not. Here, we should further consider the bit upset causes acceptable approximate or not for the control of output quality. Therefore, node states should be extended to {P = precise, A = approximate, U = unacceptable}. The often-used metric error distance (*ED*) determines the node status P, A, or U in Equation (2). The threshold can be set on

demand, if the error distance between actual value X and approximate value $\widetilde{X}$ is 0, the state is P. *ED* is less than the threshold, and the node state is identified as A. Otherwise, it is U.

$$\text{State}(\widetilde{X})\begin{cases} \text{P} & if\ ED = \left|X - \widetilde{X}\right| = 0 \\ \text{A} & if\ ED = \left|X - \widetilde{X}\right| < ED_{threshhold} \\ \text{U} & if\ ED = \left|X - \widetilde{X}\right| \geq ED_{threshhold} \end{cases} \tag{2}$$

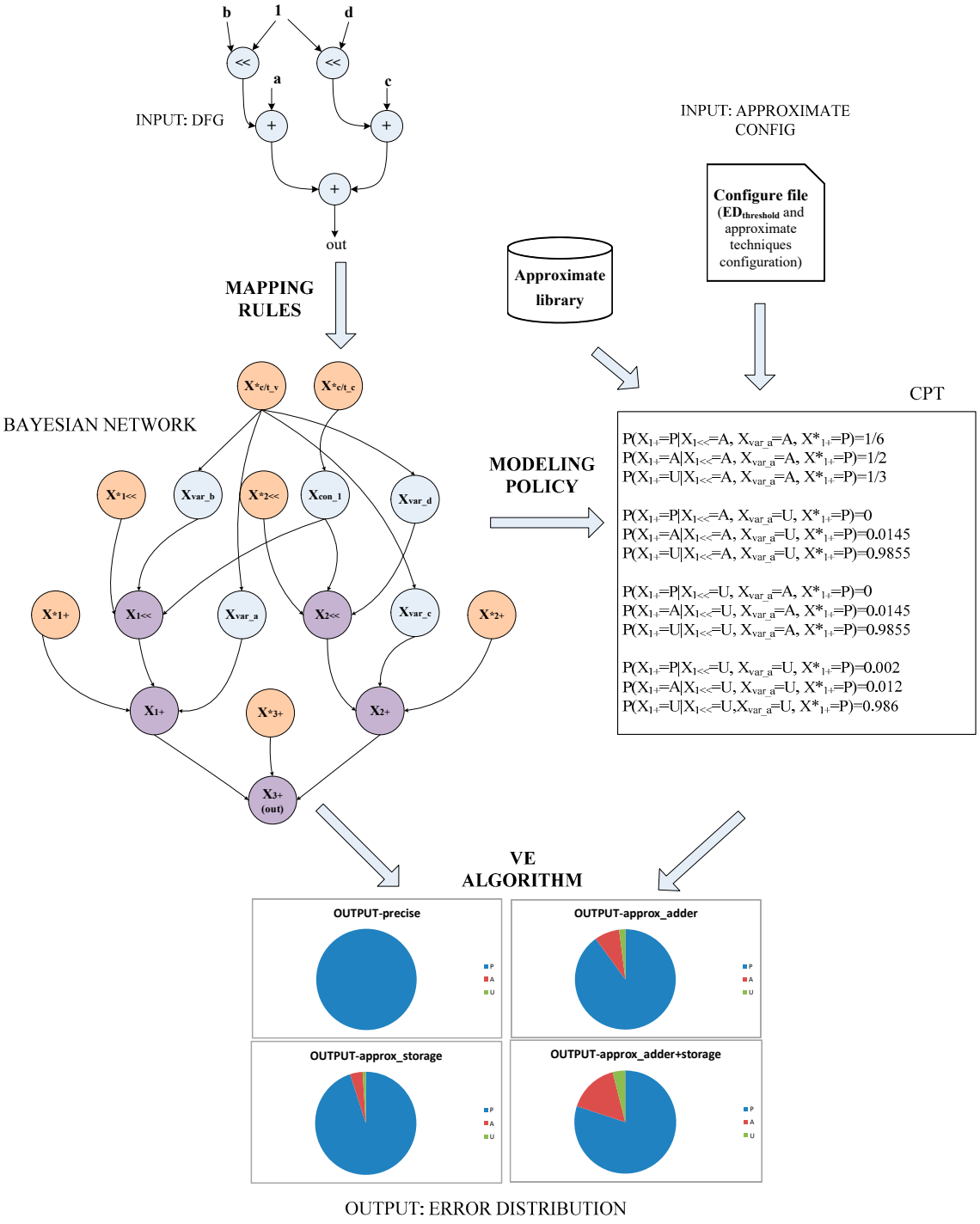


Figure 2. *HEAP* workflow overview.

On the other hand, multiple approximate configurations from different components or different levels are also characterized by inserting the corresponding nodes into the unified Bayesian network. These kinds of nodes are different from input, intermediate, or output variables, which have three states {P, A, U} to pass the error propagation until final output. These approximate technique nodes X^* follow typical Bernoulli distribution and have two states {P = precise, A = approximate}. As Equation (3) depicts, if the node value is assigned to A, the approximate technique is activated. Otherwise, P deactivates the approximate technique.

$$\text{State}(\widetilde{X^*})\begin{cases} \text{P} & \text{if } X^* = 1 \\ \text{A} & \text{if } X^* = 0 \end{cases} \tag{3}$$

In addition, approximate calculation differs from approximate communication and approximate storage. The former is influenced by both operator and operands, while the latter has no functional operation and often changes the value of stored or transmitted variable directly. Consequently, the approximate calculation, like addition, should append a three-state node for storing the computing intermediate result. To keep consistent with the realistic scenario, we assume that the approximate communication and storage are usually configud for input variables.

From the above analysis and description, we can summarize the following three mapping rules to convert an application specified DFG to a Bayesian network as Figure 3 shows. It is noted that Figure 3b only shows the mapping result of the dotted region in Figure 3a for good readability.

(1) Each input in DFG is converted to one X-type node with {P, A, U} three states to track the error propagation, e.g., the node X_{var_a} relates to the input variable *a*;
(2) Each operator in DFG becomes one corresponding variable node X_{1+} with three states {P, A, U} of calculation result and one approximate configuration node X^*_{1+} with two states {P, A}. Similarly, $\{X_{1<<}, X^*_{1<<}\}$, $\{X_{1>>}, X^*_{1>>}\}$ are the corresponding nodes pair of shift left and shift right operators. The number 1 represents the count of this same kind of operation for locating itself in a Bayesian network. Keep the edges between inputs and operator variables like X_{var_a}->X_{1+}. The new edge in bold from the operator configuration node to the operator variable node is also added, e.g., X^*_{1+}->X_{1+};
(3) Insert the non-calculation type approximation node and build the directed edges to each input, e.g., X^*_{s/t_v}-> X_{var_a}. Varying input and constants are usually handled separately for their different storage ways, e.g., X^*_{s/t_c}-> X_{con_1}.

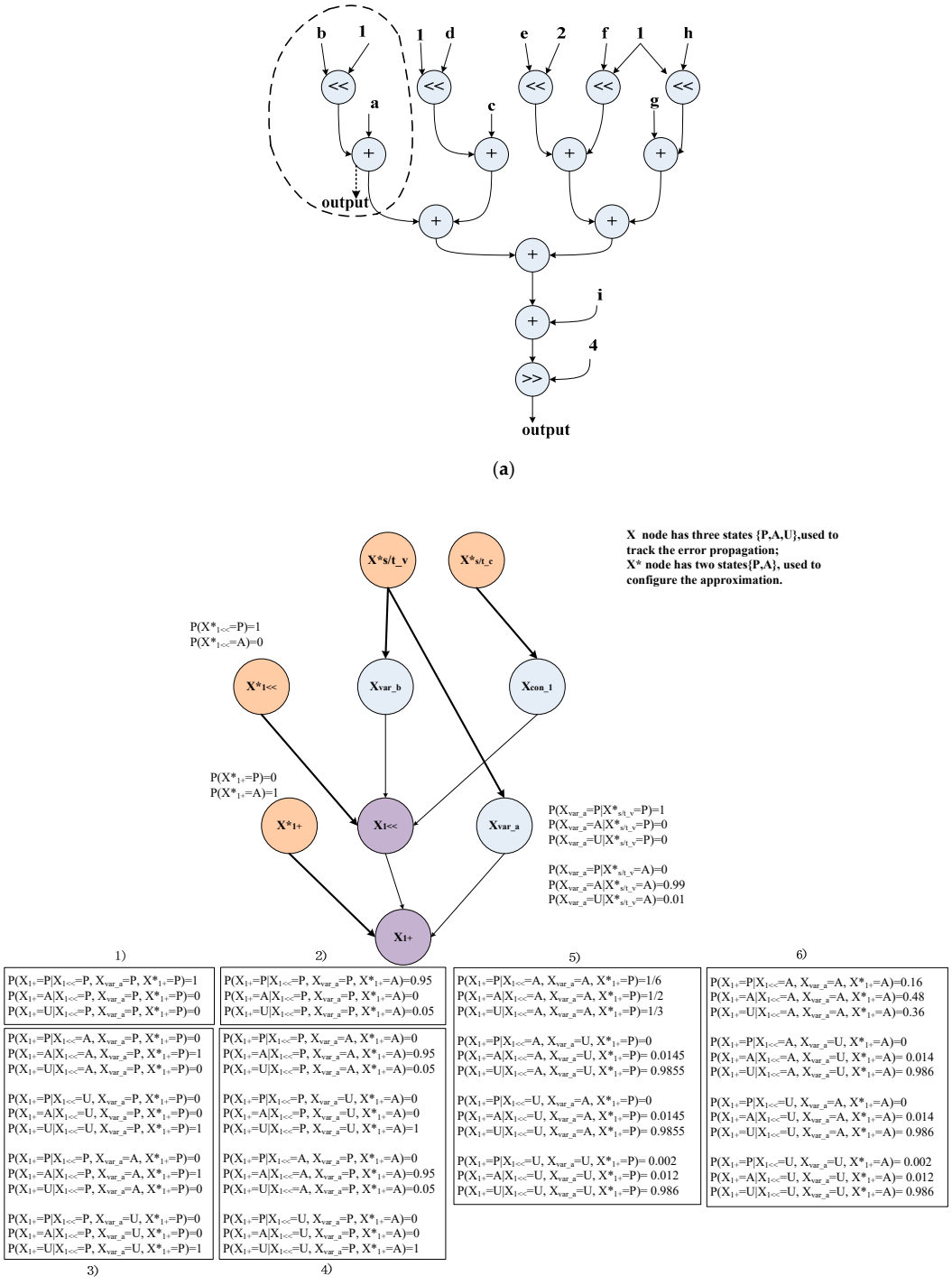
Once we get the Bayesian graph and node information like CPT as Figure 3 shows, the joint probability is expressed as Equation (4).

$$\begin{aligned} &P\left(X^*_{s/t\_v},\ X^*_{s/t\_c}, X^*_{1\ll}, X^*_{1+}, X_{1\ll}, X_{1+}, X_{var\_a}, X_{var\_b}, X_{con\_1}\right) \\ =\ &P\left(X^*_{s/t\_v}\right) P\left(X^*_{s/t\_c}\right) P\left(X^*_{1\ll}\right) P\left(X^*_{1+}\right) P(X_{var\_a} \left| X^*_{s/t\_v}\right.) P(X_{var\_b} \left| X^*_{s/t\_v}\right.) \\ &P(X_{con\_1} \left| X^*_{s/t\_c}\right.) P(X_{1\ll} \left| X_{var\_b},\ X_{con\_1},\ X^*_{1\ll}\right.) P(X_{1+} \left| X_{var\_a},\ X_{1<<},\ X^*_{1+}\right.) \end{aligned} \tag{4}$$

Based on the joint probability in the above Equation (4), inference can obtain different marginal probabilities for multiple approximate configurations. For example, the marginal probability of output node X_{1+} with unacceptable results $P(X_{1+} = U\ |X^*_{s/t\_v} = X^*_{s/t\_c} = X^*_{1<<} = P,\ X^*_{1+} = A)$ provides the error propagation under approximate addition configuration in accelerator. Similarly, the marginal probability of output node X_{1+} with unacceptable results $P(X_{1+} = U\ |X^*_{s/t\_c} = X^*_{1<<} = P,\ X^*_{s/t\_v} = X^*_{1+} = A)$ under the mixed configuration of approximate addition and approximate input in accelerator. The approximate input may be achieved by approximate communication or storage, or even other equivalent approximate techniques.

3.3.2. Structure and Node Parameters Learning

This section provides the two learning tasks: Structure learning and node parameters learning for the following inference. (1) Structure learning. As Figure 3 shows, we use the simple mapping rules to construct the Bayesian network structure from a determined DFG. Therefore, the application specified DFG is converted into an exact Bayesian representation for further inference. (2) Node parameters learning.



(**a**)

(**b**)

Figure 3. Error estimation of approximate techniques issue mapped into PGM based framework in *HEAP*. (**a**) DFG to represent a 3 × 3 Gaussian kernel. Variable input *a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i* should be loaded

from data storage while 1,2,4 are constant as a part of instruction. Shift left (<<), shift right (>>), and addition (+) are operators. The directed edge gives the bearing of data flow; (**b**) HEAP Bayesian network representation. Purple nodes come from DFG completely, while additional pink nodes are for approximate communication or storage, and turquoise nodes are inserted for approximate calculation. The bold lines are new, while the normal line inherits from DFG.

Firstly, the X*-type root nodes are used to configure the possible approximation as described in Section 3.3.1. This kind of node's CPT has only two cases, as Figure 3b shows. Precise shift operation is achieved by setting the CPT of node $X^*_{1<<}$ to $P(X^*_{1<<} = P) = 1$, $P(X^*_{1<<} = A) = 0$. If activate approximate addition, $P(X^*_{1+} = P) = 0$, $P(X^*_{1+} = A) = 1$.

The rest X-type nodes' CPTs are more complex because of dependences from their parents. Particularly, the mixture of multiple approximate techniques further increases the difficulties of CPT calculation for X-type nodes. We classify the calculations into six categories according to the number of imprecise parents and parent type as follows:

(1) All precise (zero imprecise nodes).

If all parent nodes are precise, this part of CPT is determined easily. The current node must be precise; therefore, the corresponding probability is 1, while the approximate and unacceptable probabilities are both zero as Figure 3b shows.

(2) Approximate operator.

Considering the variety of multiple approximate techniques, we can model each given approximate technique through Probability Mass Function (PMF) in greater detail. PMF denotes the probability P of a discrete random variable X, to be equal to a determined value x. This is expressed as $pX(x) = P(X = x)$. This mechanistic modeling can be finished by fast theoretical analysis (or use simulation profiling to build empirical modeling, it is more accurate but time consuming. More related information is discussed in Section 4.3 and the future work of Section 5). For example, PMF of an 8-bit GeAr adder [15] in Figure 4 can give the corresponding CPT if two operands are precise. These models of approximate techniques are stored in an approximate library for reuse.

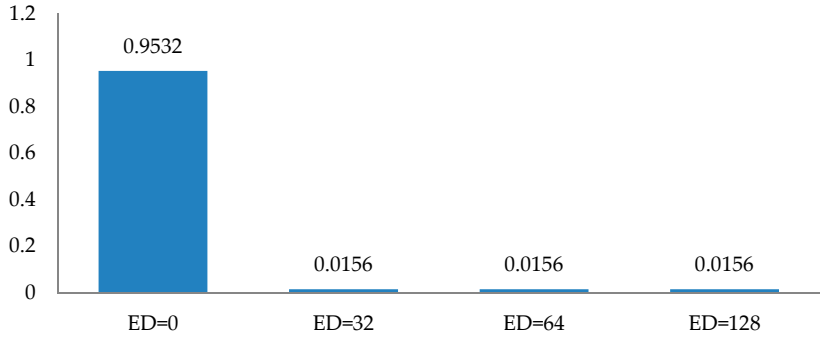


Figure 4. 8-bit approximate adder GeAr Probability Mass Function.

When the $ED_{threshhold}$ is set to 4, the partial information of X_{1+} can be determined by the first-order approximate impact in Equation (5). Similarly, the complete CPTs of nodes X_{var_a}, X_{var_b}, X_{con_1} can be available.

$$\begin{cases} P\left(X_{1+} = P \middle| X_{1\ll} = P,\ X_{var\_a} = P,\ X^*_{1+} = A\right) = 0.95 \\ P\left(X_{1+} = A \middle| X_{1\ll} = P,\ X_{var\_a} = P,\ X^*_{1+} = A\right) = 0.00 \\ P\left(X_{1+} = U \middle| X_{1\ll} = P,\ X_{var\_a} = P,\ X^*_{1+} = A\right) = 0.05 \end{cases} \tag{5}$$

(3) One imprecise operand.

This calculation is based on operator analysis. $log2(ED_{threshhold})$ determines the acceptable lowest bits upset in X-type node. For example, $ED_{threshold}$ is set to 4. One bits uet in the lowest 2 bits can be marked as A, otherwise, it is P. Due to the addition operator in X_{1+}, the same ED is transmitted from one source operand X_{var_a} to the destination operand X_{1+}. The corresponding CPT information can be determined in Equation (6). However, it is noted that the CPT information should be changed with the varying operator. For example, if a shift operator, the bits upset from imprecise operand may be masked and result in a different CPT. Similarly, the unacceptable operand case can be handled.

$$\begin{cases} P\left(X_{1+} = P \middle| X_{1\ll} = P,\ X_{var\_a} = A,\ X^*_{1+} = P\right) = 0 \\ P\left(X_{1+} = A \middle| X_{1\ll} = P,\ X_{var\_a} = A,\ X^*_{1+} = P\right) = 1 \\ P\left(X_{1+} = U \middle| X_{1\ll} = P,\ X_{var\_a} = A,\ X^*_{1+} = P\right) = 0 \end{cases} \tag{6}$$

(4) Two imprecise operands.

If multiple approximate operands are together, we can finish the CPT calculation through exhaustion of all possible cases. The process occurs only once, but can be reused many times in the same kind of operator nodes via an approximate library. For exampl $ED_{threshhold}$is set to 4, and Table 1 lists all eighteen cases for X_{1+} with approximate $X_{1<<}$ as well as approximate X_{var_a}. We can find out three P's, six U's, and twelve A's. Therefore, the CPT information can be determined. This process also considers the error masking of two concurrent approximations into a precise result in Table 1. For example, one operand $X_{1<<}$ has positive 1 of ED, while the other X_{var_a} has the same negative value -1. The addition makes the offset of two approximate operands with each other. Similarly, one approximate plus one unacceptable operand or two unacceptable operands could be handled in an extended way. The difference lies in CPTs of the two cases, which are influenced by bit width. Without loss of generality, we give a mechanistic model for an arbitrary bit width n in in Tables 2 and 3, respectively.

Table 1. Conditional probabilities table (CPT) calculation under twopproximate operands with addition.

$ED(X_{1<<})$	$ED(X_{var\_a})$	$ED(X_{1+})$	*State of* X_{1+}
1	1	0	P
		2	A
1	2	1	A
		3	A
1	3	2	A
		4	U
2	1	1	A
		3	A
2	2	0	P
		4	U
2	3	1	A
		5	U
3	1	2	A
		4	U
3	2	1	A
		5	U
3	3	0	P
		6	U
P: 1.5/9 = 1/6	A: 4.5/9 = 1/2	U:3/9 = 1/3	

Table 2. CPT calculation under one approximate operand plus one unacceptable operand.

$ED(X_{1<<})$, $ED(X_{var\_a})$	Bit Width		$ED(X_{1+})$	State of X_{1+}
	n	8		
$ED(X_{1<<}) = 1$ $ED(X_{var\_a}) = 4$ OR $ED(X_{var\_a}) = 1$ $ED(X_{1<<}) = 4$	$(ED_{threashold} - 3) \times 2$	2	3	A
			5	U
$ED(X_{1<<}) = 2$ $ED(X_{var\_a}) = 4,5$ OR $ED(X_{var\_a}) = 2$ $ED(X_{1<<}) = 4,5$	$(ED_{threashold} - 2) \times 2$	4	1	A
$ED(X_{1<<}) = 3$ $ED(X_{var\_a}) = 4,5,6$ OR $ED(X_{var\_a}) = 3$ $ED(X_{1<<}) = 4,5,6$	$(EDt_{hreashold} - 1) \times 2$	6	2	A
Others	$(2^n - ED_{threashold} + 12) \times (ED_{threashold} - 1)$	744	>3	U
Total	$(2^n - ED_{threashold})(ED_{threashold} - 1)$	756		{P,A,U}
P: 0	A: 11/756 = 0.0145	U:745/756 = 0.9855		

Table 3. CPT calculation under two unacceptable operands with addition.

$ED(X_{1<<})$, $ED(X_{var\_a})$	Bit Width		$ED(X_{1+})$	State of X_{1+}
	n	8		
$(2^n$-$ED_{threashold})$ $ED(X_{1<<}) = ED(X_{var\_a})$	$2^n - ED_{threashold}$	252	0	P
			2*$ED(X_{var\_a})$	U
$ED(X_{1<<}) - ED(X_{var\_a}) = 1$ OR $ED(X_{1<<}) - ED(X_{var\_a}) = -1$	2^n – 1-$ED_{threashold}$	251	1	A
$ED(X_{1<<}) - ED(X_{var\_a}) = 2$ OR $ED(X_{1<<}) - ED(X_{var\_a}) = -2$	2^n – 2-$ED_{threashold}$	250	2	A
$ED(X_{1<<}) - ED(X_{var\_a}) = 2$ OR $ED(X_{1<<}) - ED(X_{var\_a}) = -2$	2^n – 3-$ED_{threashold}$	249	3	A
Others	$(2^n - ED_{threashold})$ $(2^n - ED_{threashold} - 4) + 6$	62502	>3	U
Total	$(2^n - ED_{threashold}) \times 2$	63504		{P,A,U}
P: 126/63504 = 0.002	A: 750/63504 = 0.0012	U: 62628/63504 = 0.986		

(5) Approximate operator + one imprecise operand.

The mixture of category (2) and (3) makes this (5). If there is one approximate operand, the CPT information is determined by its PMF and one approximate operator in Equation (7).

$$\begin{cases} P\left(X_{1+} = P \middle| X_{1\ll} = P,\ X_{var\_a} = A,\ X^*_{1+} = A\right) = 0.00 \\ P\left(X_{1+} = A \middle| X_{1\ll} = P,\ X_{var\_a} = A,\ X^*_{1+} = A\right) = 0.95 \\ P\left(X_{1+} = U \middle| X_{1\ll} = P,\ X_{var\_a} = A,\ X^*_{1+} = A\right) = 0.05 \end{cases} \tag{7}$$

When there is an unacceptable operand, the CPT information is determined by a higher priority of status U, and equal to one unacceptable operand in Equation (8).

$$\begin{cases} P\left(X_{1+} = P \middle| X_{1\ll} = P,\ X_{var\_a} = U,\ X^*_{1+} = A\right) = 0 \\ P\left(X_{1+} = A \middle| X_{1\ll} = P,\ X_{var\_a} = U,\ X^*_{1+} = A\right) = 0 \\ P\left(X_{1+} = U \middle| X_{1\ll} = P,\ X_{var\_a} = U,\ X^*_{1+} = A\right) = 1 \end{cases} \tag{8}$$

(6) Approximate operator + two imprecise operands.

The mixture of category (2) and (4) makes this (6). If there are two approximate operands, the CPT information is determined by weighted computing in Table 4. Similarly, the rest of the cases can be solved via considering the operator *ED* with operands *ED* together, as the CPTs of nodes in Figure 3b shows.

Table 4. CPT calculation under two approximate operands with addition.

$ED(X_{1<<})$	$ED(X_{var\_a})$	$ED(X^*_{1+})$	$ED(X_{1+})$	State of X_{1+}
$(0, ED_{threashold})$	$(0, ED_{threashold})$	0(95%)	=0	P(0.95/6)
			$<= ED_{threashold}$	A(0.95/2)
			$>= ED_{threashold}$	U(0.95/3)
$(0, ED_{threashold})$	$(0, ED_{threashold})$	>3(0.05)	$>= ED_{threashold}$	U(0.05)
P:0.95/6 = 0.16		A:0.95/2 = 0.48	U:0.95/3 + 0.05 = 0.36	

Based on the above complete calculation policy of six categories, the local error occurrence distribution of each approximate technique can be expressed by the node CPT information. If mechanistic models for some existing approximate techniques are available, they can be reused directly. If not, the simulation-based profiling can be adopted to achieve the empirical models. The mechanistic or empirical even hybrid models help to fill the approximate characteristics into CPTs for the further inference.

3.3.3. Exact VE Inference

Exact variable elimination (VE) [54] is chosen as the inference algorithm in this paper. Because VE merges and adds a series of decomposition factors one by one in the PGM graph, which can achieve accurate solutions.

The detailed algorithm description is shown in Figure 5, including four inputs: The conditional probability table of nodes, that is, the CPTs of all nodes, the observation node X, and the query node list Y, where Y_0 is the observation value corresponding to Y; elimination order; the output is the edge probability P (X|Y = Y_0). More descriptions and accuracy proofs of the VE algorithm can be found in the literature [54]. The complexity of VE can be measured by the number of times of numerical multiplication and numerical addition. An optimal elimination order can bring the lowest complexity, but how to find the optimal elimination order itself is still an NP problem. In this paper, the VE algorithm uses the topological order that the PGM structure depends on as the elimination order.

Procedure VE $(\Gamma, X, Y, Y_0, \rho)$

***Inputs*:** Γ is the list of conditional probabilities (CPTs of all nodes) in a Bayesian network;
X denotes a list of query variables; Y is a list of observed variables;
Y_0 represents the corresponding list of observed values;
ρ is an elimination ordering for variables outside $X \bigcup Y$.
Output: $P(X|Y=Y_0)$.

1. Set the observed variables in all factors to their corresponding observed values.
2. **While** ρ is not empty
3. Remove the first variable z from ρ,
4. Call sum-out(Γ, z).
5. **Endwhile**
6. Set **h=** the multiplication of all the factors on Γ.
//h is a function of variables in **X**.
7. Return $h(X)/\sum_X h(X)$. // Renormalization

Figure 5. Variable elimination (VE) algorithm persudo code.

4. Experiments and Results Analysis

4.1. Accuracy and Speed Evaluatoin

To verify the effectiveness of proposed framework *HEAP*, we implement C++ language-based Mentor Carlo simulation (MC) with 1000000 samples and proposed framework *HEAP* using academic version of PGM library SMILE [56] for 3 × 3 Gaussian kernel, respectively. During the application execution of MC, we insert the bits upset, following the PMF of configured approximate techniques, such as approximate adder in Figure 4. Three different configurations are compared between MC and proposed *HEAP* in Table 5. It is noted that 1,000,000 samples of MC are divided into 10 groups with 100,000 samples to compute the error bar. The results show the error bar is very small and ranges from −0.004% to 0.002% around the mean value. The results in Figure 6 show the error distribution differences between MC and proposed *HEAP*.

Table 5. Approximate configurations for estimation accuracy and speed comparison.

Approximate Configurations	Approximate Techniques	ALS
approx_adder	Approximate addition for intermediate operation (GeAr adder)	1.53%
approx_s/t_var	Approximate communication or storage for Input variables	6.38%
approx_adder+s/t_var	Mixture of the above two techniques	4.64%
Average		4.18%

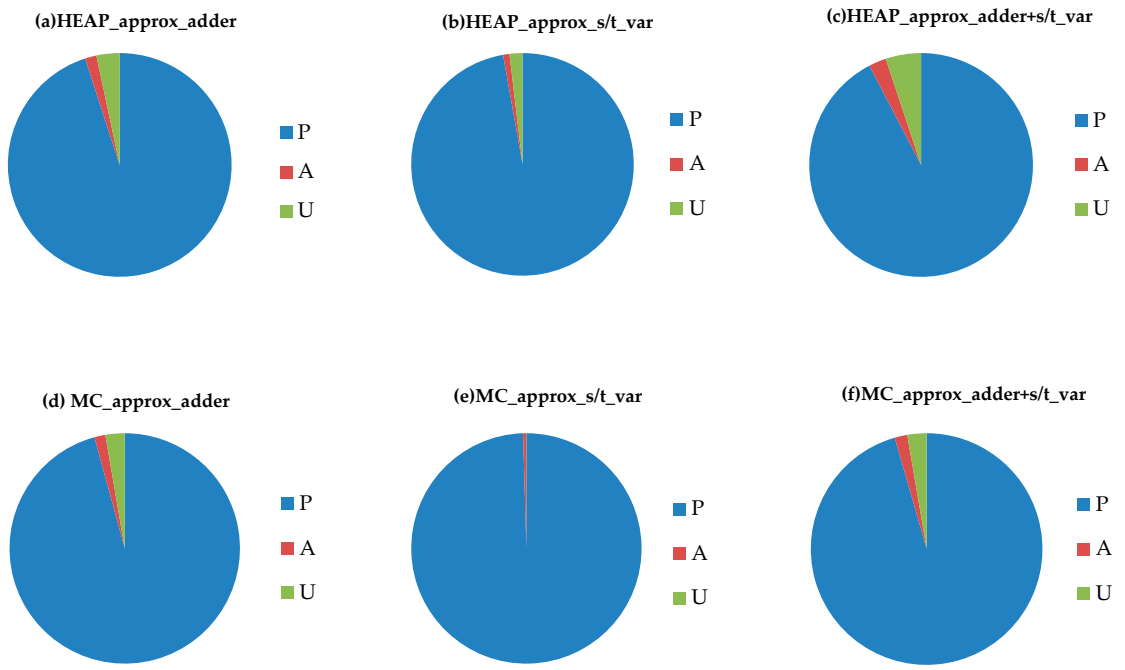


Figure 6. Estimation accuracy comparison between *HEAP* and MC.

A new metric, Accuracy Loss Sum (ALS) is defined to measure the output distribution inaccuracy of proposed HEAP over MC in Equation (9) as follows.

$$ALS = \sum_{i=\{P,A,U\}} abs\Big(P(State(outputs) = i)_{HEAP} - P(State(outputs) = i)_{MC}\Big) \quad (9)$$

The ALS values for three approximate configurations are calculated based on Figure 6 and listed in Table 5. We can figure out that the inaccuracy ranges from 1.53% to 6.38%, and on average 4.18%. Among of three approximate configurations, approx_adder has the least accuracy loss, down to 1.53%, while approx_s/t gets the most accuracy loss 6.38%. In *HEAP* design, Bayesian network and variable elimination inference are both exact, while the nodes' CPTs are filled with modeling information in

Section 3.3. For these experiments, left or right shifters adopt conservative mechanistic modeling. This handling policy makes undetermined cases in an unacceptable status. The errors in input variables like a,b,c with approx_s/t_var configuration will propagate to the output through six shifters, while approx_adder configuration only go through one shifter in the structure of Figure 3a. Therefore, it is reasonable that the error paths with more nodes related to imprecise shifters modeling will cause higher accuracy loss. Lower accuracy loss can be achieved by more accurate empirical modeling and will be discussed in Section 4.3. Additionally, the mixed configuration has a medium accuracy loss 4.64% less than a single configuration of approx_s/t_var. This obviously results from the more accurate approx_adder modeling.

The estimation speed comparison between proposed *HEAP* and MC is shown in Figure 7. 1,000,000 samples in MC consumes thousands of seconds, while the proposed *HEAP* makes good uses of the high-speed advantage of PGM framework and consumes about 5 milliseconds. Therefore, *HEAP* has 334043.6× speedup on average over MC.

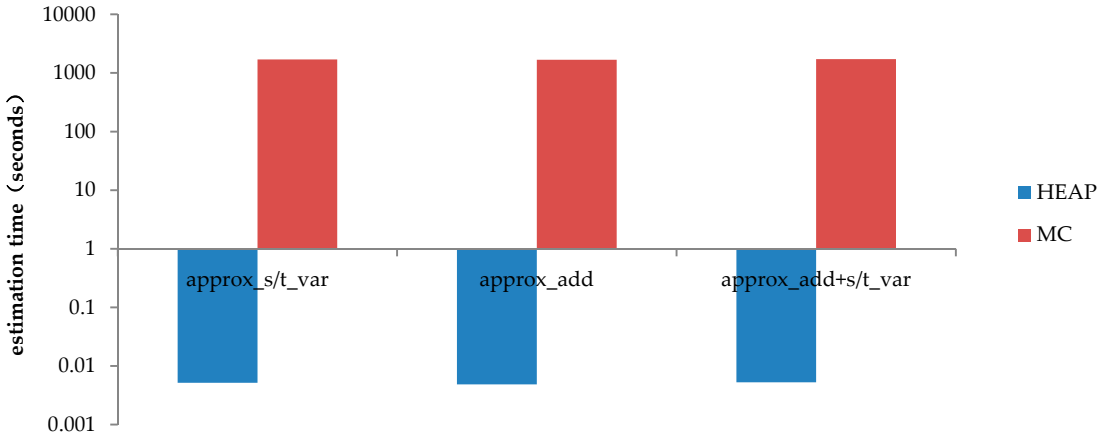


Figure 7. Estimation speed comparison between *HEAP* and MC.

In all, the proposed *HEAP* provides fast and accurate error assessment, achieving 4.18% accuracy loss and five orders of magnitudes speedup over MC. Moreover, this configuration approx_s/t_var of approximate input variables in *HEAP* can be considered equivalent to precise scaling in [48,49]. The estimation accuracy loss and computational time in millisecond of *HEAP* mostly keeps the same magnitude with the works in [48,49]. The tiny accuracy difference mainly results from the calculation policy of CPTs, our mechanistic modeling versus empirical modeling. This will be further discussed in Section 4.3.

4.2. *Approximate Techniques Selection Using HEAP*

Once we confirm the accuracy of proposed framework *HEAP*, we can use it to do more analysis so that the good approximate accelerator design can be achieved via balancing output quality and power saving (or performance improvement) well.

Firstly, we can evaluate each single approximate technique and identify whether it is suitable for the application-specified fast accelerator. Figure 8 gives the error distributions of approximate storage or communication, approximate left or right shifter, and one approximate adder. We can figure out that approximate storage (or communication) and adder are good choices for approximate accelerator, while shifters bring higher probability of unacceptable output.

Secondly, we explore the approximation degree selection of a single approximate technique. Figure 9 shows the increasing number of approximate GeAr adder effects on the application specified accelerator. The X-axis represents the number of approximate adders, while Y-axis represents the probability of acceptable output. The maximum configuration has three approximate adders to guarantee a 90% acceptable output. This configuration can be considered the error propagation of

varying adder number and keep high fidelity, with [47] from the estimation accuracy over Mentor Carlo simulations.

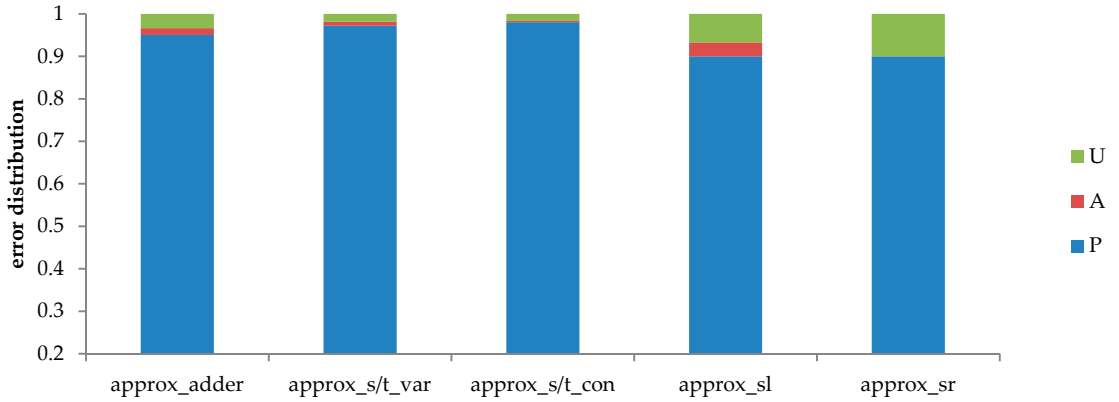


Figure 8. Different approximate techniques comparison.

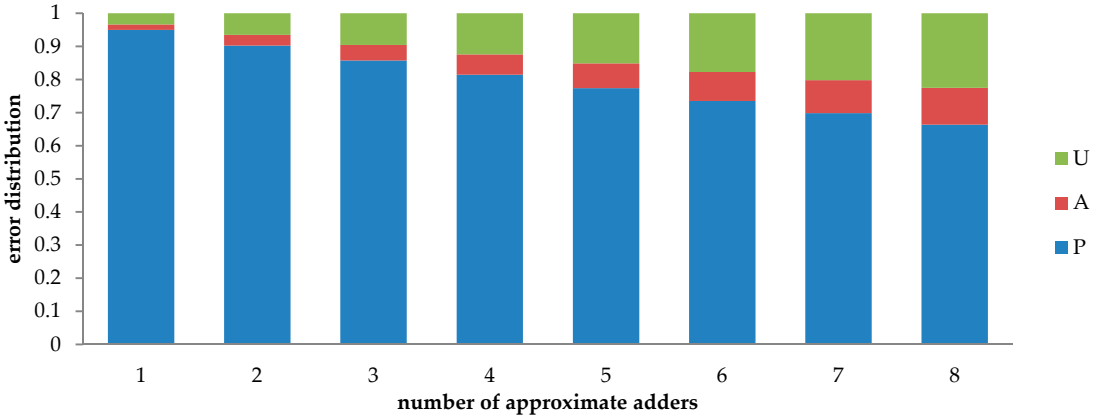


Figure 9. Different approximate degrees of a single approximate technique comparison.

Finally, we explore the combinations of multiple different approximate techniques. Figure 10 show the error distribution results of different mixed approximate configurations in Table 6. All these configurations can satisfy a 90% acceptable output. To select the best mixture, we can consider the relevant power saving or performance improvement.

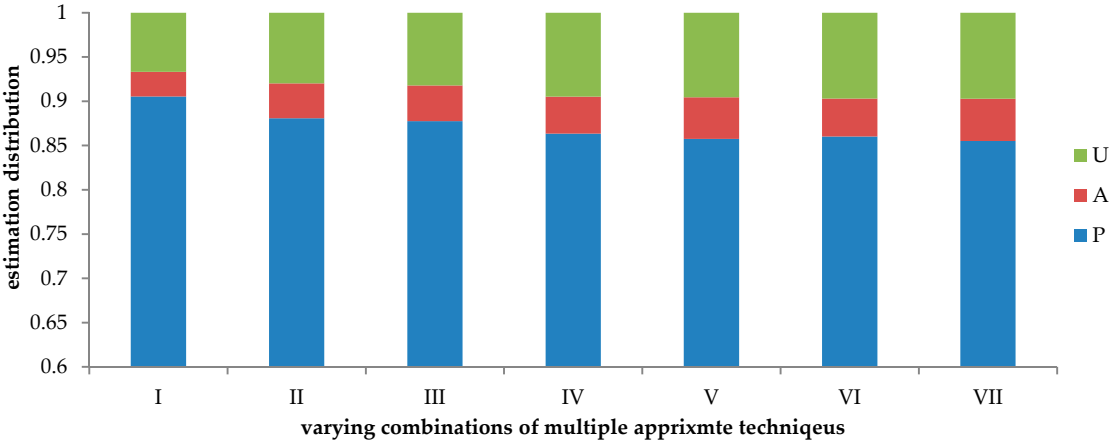


Figure 10. Different combinations of multiple approximate techniques comparison.

Table 6. Multiple approximate techniques selection via comparing power.

Type	Approx_s/t_con		Approx_s/t_var		Approx_adder Total Number = 8		Approx_sl Total Number = 5 + 1		Power Saving $\sum W_i P_i$
	W_i	P_i	W_i	P_i	W_i	P_i	W_i	P_i	
	12%	50%	19%	63%	69% × 80%	60% × P_j	69% × 20%	60% × P_j	
I	√		√		√(number = 1,P_j = 1/8)				22%
II			√				√(number = 1,P_j = 1/6)		13%
III			√		√(number = 2,P_j = 2/8)				20%
IV	√		√				√(number = 1,P_j = 1/6)		19%
V					√(number = 3,P_j = 3/8)				12%
VI	√		√		√(number = 2,P_j = 2/8)				26%
VII					√(number = 1,P_j = 1/8)		√(number = 1,P_j = 1/6)		10%

Here, we assume that the full power breakdown includes 19% memory access, 12% communication, and 69% computing [57]. This percent is the weight value, denoted as W_i. approx_s/t_var is configured by approximate data cache [18], and can save power up to 63%, while approx_s/t_con is approximate communication with about 50% power saving [35]. The power saving percent is denoted as P_i. Bitwise operation like shift left or shift right is simpler and consumes lower power than addition. Therefore, we divide the computing power into 80% addition and 20% shift. Each approximate component is assumed to reduce power by 60% [58]. The number of configuring approximate components also influences the power saving through P_j. The best mixture of multiple approximate techniques is composed of two approximate adders, approx_s/t_var and approx_s/t_con. It can provide a 91.8% acceptable output in Figure 10, with up to 26% power saving through $\sum W_i P_i$ in Table 6. These results demonstrate that the mixed configuration can exploit fine-grain tradeoff between output quality degradation and power saving.

4.3. Discussion About HEAP Optimization

To improve the estimation accuracy, there is no doubt that the more accurate CPT is required. Simulation based empirical modeling can be used instead of mechanistic modeling. However, it is noted that the simulation-based modeling consumes more design time and developer efforts. Therefore, we recommend the hybrid modeling to fill the CPTs, where most information is determined by mechanistic modeling, and little critical information like one or two approximate operands case can be handled by simulation profiling. Taking an 8-bit left shifter for example, an empirical modeling policy is used for two cases: One approximate operand and two approximate operands. The rest keep the mechanistic modeling. This hybrid modeling reduces average ALS of three configurations in Table 5 down to 1.61%. This result is considerably consistent with 98–99% accuracy of Bayesian modeling in [49]. The appropriate tradeoffs can be compromised between modeling efforts and effectiveness to support various approximate techniques.

HEAP has good configurability. Section 4.2 has demonstrated the easy configuration of different approximate techniques, approximate degree, and a mixture of multiple approximate techniques using X*-type nodes. Additionally, the X*-type nodes can be modified, inserted, removed, or even merged in a free-style. For example, the two approximate adders have different PMFs so that their related CPT will be update and make a new error distribution. If all of the adders adopt a unified configuration, all of the nodes can be merged into only one. Similarly, other operations can be finished to guarantee the good configurability. This characteristic of heterogeneous Bayesian network in *HEAP* can support arbitrary combination of multiple approximate techniques as Figure 10 and Table 6 gives, where a single approximate technique is also one particular instance and can also be evaluated very well, as Figures 8 and 9 shows.

Therefore, the proposed *HEAP* provides a fast and accurate error assessment approach to designing cost-effective approximate accelerators.

5. Conclusions

In this paper, we proposed a fast and accurate error assessment framework *HEAP* for approximate accelerator design that can estimate the probability of multiple approximate technique impacts on output quality at different levels (precise, approximate, and unacceptable) in a configurable manner using probabilistic graphical models. *HEAP* consists of a heterogeneous Bayesian network representation, which has two different types of nodes, X*-type and X-type, to configure approximate options and track corresponding error propagation, respectively. This is followed by approximate library driven node parameters learning, which characterizes the uniform probability mass function of available approximate techniques through mechanistic models or empirical models. Based on the ready Bayesian network and node parameters of conditional probability tables, the exact variable elimination inference can calculate the marginal probability of output quality at three levels of precise, approximate, and unacceptable under given approximate configurations quickly and accurately. Compared with Mentor Carlo simulation, the proposed *HEAP* framework can achieve just 4.18% accuracy loss and 3.34×10^5 speedup for 3×3 Gaussian kernel. The good configurability of *HEAP* also makes itself able to estimate different approximate techniques and even their diverse combinations. Therefore, the proposed *HEAP* can provide flexible quality-power tradeoffs through estimating multiple approximate techniques quickly and select the best approximate configuration for maximum power saving with acceptable output quality.

In the future, *HEAP* will be extended to support more popular approximate techniques and exploit fine-grain tradeoff among the variety of combinations efficiently in two directions: (1) Insert mechanistic and empirical models of more approximate solutions into our existing approximate library of *HEAP* for good variety. This work needs the probability mass function of available approximate options, which should cover as many of the approximate techniques as possible, approximate hardware architecture or approximate software approach, low-level approximate adder circuits, or high-level approximate Cache hierarchy; (2) adopt heuristic algorithm or evolution optimization to search the best solution from more approximate options smartly and quickly. The large Bayesian network size causes a huge solution space to configure the approximate techniques for best tradeoffs. These efficient search algorithms are required instead of exhaustive method. The enhanced *HEAP* will not only help designers to decide the appropriate quality-power (or performance) tradeoffs faster and more accurately, but also further shortens the time-to-market of an approximate accelerator very well.

Author Contributions: Conceptualization, J.J.; project administration, writing, and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China grant No. 61502298, No. 61603245 and No.71702100.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Li, X.; Yeung, D. Application-level correctness and its impact on fault tolerance. In Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, Scottsdale, AZ, USA, 10–14 February 2007; pp. 181–192.
2. Chippa, V.K.; Chakradhar, S.T.; Roy, K.; Raghunathan, A. Analysis and characterization of inherent application resilience for approximate computing. In Proceedings of the 50th Annual Design Automation Conference, New York, NY, USA, 2–6 June 2013; p. 113.
3. Li, C.; Luo, W.; Sapatnekar, S.S. Joint precision optimization and high level synthesis for approximate computing. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference, Piscataway, NJ, USA, 8–12 June 2015; pp. 1–6.

4. Esmaeilzadeh, H.; Sampson, A.; Ceze, L.; Burger, D. Neural acceleration for general-purpose approximate programs. *IEEE Micro* **2013**, *33*, 16–27. [CrossRef]
5. Misailovic, S.; Carbin, M.; Achour, S.; Qi, Z.; Rinard, M.C. Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. *ACM SIGPLAN Not.* **2014**, *49*, 309–328. [CrossRef]
6. Sampson, A.; Dietl, W.; Fortuna, E.; Gnanapragasam, D.; Ceze, L.H.; Grossman, D. EnerJ: Approximate data types for safe and general low-power computation. *ACM SIGPLAN Not.* **2011**, *46*, 164–174. [CrossRef]
7. Samadi, M.; Lee, J.; Jamshidi, D.A.; Hormati, A.H.; Mahlke, S.A. Sage: Self-tuning approximation for graphics engines. In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, Davis, CA, USA, 7–11 December 2013; pp. 13–24.
8. Chippa, V.; Raghunathan, A.; Roy, K.; Chakradhar, S. Dynamic effort scaling: Managing the quality-efficiency tradeoff. In Proceedings of the 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), New York, NY, USA, 5–9 June 2011; pp. 603–608.
9. Moreau, T.; Wyse, M.; Nelson, J.; Esmaeilzadeh, H.; Ceze, L.; Oskin, M. SNNAP: Approximate computing on programmable socs via neural acceleration. In Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 7–11 February 2015; pp. 603–614.
10. Grigorian, B.; Reinman, G. Accelerating divergent applications on simd architectures using neural networks. *ACM Trans. Arch. Code Optim.* **2015**, *12*, 2. [CrossRef]
11. Grigorian, B.; Farahpour, N.; Reinman, G. BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing. In Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, USA, 7–11 February 2015; pp. 615–626.
12. Rubio-González, C.; Nguyen, C.; Nguyen, H.D.; Demmel, J.; Kahan, W.; Sen, K.; Bailey, D.H.; Iancu, C.; Hough, D. Precimonious: Tuning assistant for floating-point precision. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13), Denver, CO, USA, 17–22 November 2013; pp. 1–12.
13. Yazdanbakhsh, A.; Park, J.; Sharma, H.; Lotfi-Kamran, P.; Esmaeilzadeh, H. Neural acceleration for gpu throughput processors. In Proceedings of the 48th International Symposium on Microarchitecture, Waikiki, Hawaii, USA, 5–9 December 2015; pp. 482–493.
14. Du, Z.; Palem, K.; Lingamneni, A.; Temam, O.; Chen, Y.; Wu, C. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In Proceedings of the 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), Singapore, 20–23 January 2014; pp. 201–206.
15. Shafique, M.; Ahmad, W.; Hafiz, R.; Henkel, J. A low latency generic accuracy configurable adder. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; pp. 1–6.
16. Hanif, M.A.; Hafiz, R.; Hasan, O.; Shafique, M. QuAd: Design and analysis of quality-area optimal low-latency approximate adders. In Proceedings of the 54th Annual Design Automation Conference 2017, Austin, TX, USA, 18–22 June 2017; pp. 1–6.
17. Rehman, S.; Prabakaran, B.S.; El-Harouni, W.; Shafique, M.; Henkel, J. Heterogeneous approximate multipliers: Architectures and design methodologies. In *Approximate Circuits*; Springer: Berlin, Germany, 2019; pp. 45–66.
18. Yazdanbakhsh, A.; Pekhimenko, G.; Thwaites, B.; Esmaeilzadeh, H.; Mutlu, O.; Mowry, T.C. RFVP: Rollback-free value prediction with safe-to-approximate loads. *ACM Trans. Arch. Code Optim.* **2016**, *12*, 62. [CrossRef]
19. Miguel, J.S.; Badr, M.; Jerger, N.E. Load value approximation. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, Cambridge, UK, 13–17 December 2014; pp. 127–139.
20. Sutherland, M.; San Miguel, J.; Jerger, N.E. Texture cache approximation on GPUs. In Proceedings of the Workshop on Approximate Computing Across the Stack. In conjunction with HiPEAC, Amsterdam, The Netherlands, 19–21 January 2015.
21. Själander, M.; Nilsson, N.S.; Kaxiras, S. A tunable cache for approximate computing. In Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures, Paris, France, 8–10 July 2014; pp. 88–89.

22. Ranjan, A.; Venkataramani, S.; Pajouhi, Z. STAxCache: An approximate, energy efficient STT-MRAM cache. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 356–361.
23. Frustaci, F.; Blaauw, D.; Sylvester, D.; Alioto, M. Approximate SRAMs with dynamic energy-quality management. *IEEE Trans. Very Large Scale Integ. Syst.* **2016**, *24*, 2128–2141. [CrossRef]
24. Gilani, S.Z.; Kim, N.S.; Schulte, M. Scratchpad memory optimizations for digital signal processing applications. In Proceedings of the 2011 Design, Automation & Test in Europe, Grenoble, France, 14–18 March 2011; pp. 1–6.
25. Liu, S.; Pattabiraman, K.; Moscibroda, T.; Zorn, B.G. Flikker: Saving DRAM refresh-power through data partitioning. In Proceedings of the Architectural Support for Programming Languages and Operating Systems (ASPLOS), Lausanne, Switzerland, 16–20 March 2011.
26. Zhang, X.; Zhang, Y.; Childers, B.R.; Yang, J. DrMP: Mixed precision-aware dram for high performance approximate and precise computing. In Proceedings of the 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT), Portland, OR, USA, 9–13 September 2017; pp. 53–63.
27. Guo, Q.; Strauss, K.; Ceze, L.H.; Malvar, H.S. High-density image storage using approximate memory cells. *ACM SIGPLAN Not.* **2016**, *51*, 413–426. [CrossRef]
28. Malvar, H.S. Fast progressive image coding without wavelets. In Proceedings of the DCC 2000, Data Compression Conference, Snowbird, UT, USA, 28–30 March 2000; pp. 243–252.
29. Wallace, G.K. The JPEG still picture compression standard. *IEEE Trans. Consum. Electron.* **1992**, *38*, xviii–xxxiv. [CrossRef]
30. Dufaux, F.; Sullivan, G.J.; Ebrahimi, T. The JPEG XR image coding standard [Standards in a Nutshell]. *IEEE Signal Process. Mag.* **2009**, *26*, 195–204. [CrossRef]
31. Stevens, J.R.; Ranjan, A.; Raghunathan, A. AxBA: An approximate bus architecture framework. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 5–8 November 2018; pp. 1–8.
32. Boyapati, R.; Huang, J.; Majumder, P.; Yum, K.H.; Kim, E.J. Approx-noc: A data approximation framework for network-on-chip architectures. *ACM SIGARCH Comput. Arch. News* **2017**, *45*, 666–677. [CrossRef]
33. Ahmed, A.B.; Fujiki, D.; Matsutani, H.; Koibuchi, M.; Amano, H. AxNoC: Low-power approximate network-on-chips using critical-path isolation. In Proceedings of the Twelfth IEEE/ACM International Symposium on Networks-on-Chip, Turin, Italy, 4–5 October 2018; p. 6.
34. Raparti, V.Y.; Pasricha, S. DAPPER: Data aware approximate NoC for GPGPU architecture. In Proceedings of the Twelfth IEEE/ACM International Symposium on Networks-On-Chip, Turin, Italy, 4–5 October 2018; p. 7.
35. Ascia, G.; Catania, V.; Monteleone, S.; Palesi, M.; Patti, D.; Jose, J. Approximate wireless networks-on-chip. In Proceedings of the 2018 Conference on Design of Circuits and Integrated Systems (DCIS), Lyon, France, 14–16 November 2018; pp. 1–6.
36. Tatas, K. High-performance 3D NoC bufferless router with approximate priority comparison. In Proceedings of the 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 7–9 May 2018; pp. 1–4.
37. Hari, S.K.S.; Adve, S.V.; Naeimi, H.; Ramachandran, P. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. *ACM SIGPLAN Not.* **2012**, *47*, 123–134. [CrossRef]
38. Venkatagiri, R.; Mahmoud, A.; Hari, S.K.S.; Adve, S.V. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, Taiwan, 15–19 October 2016; p. 42.
39. Venkatagiri, R.; Ahmed, K.; Mahmoud, A.; Misailovic, S.; Marinov, D.; Fletcher, C.W.; Adve, S.V. Gem5-approxilyzer: An open-source tool for application-level soft error analysis. In Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Portland, OR, USA, 24–27 June 2019; pp. 214–221.
40. Chan, W.T.J.; Kahng, A.B.; Kang, S.; Kumar, R.; Sartori, J. Statistical analysis and modeling for error composition in approximate computation circuits. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 47–53.
41. Mazahir, S.; Ayub, M.K.; Hasan, O.; Shafique, M. Probabilistic error analysis of approximate adders and multipliers. In *Approximate Circuits*; Springer: Berlin, Germany, 2019; pp. 99–120.

42. Yang, L.; Murmann, B. SRAM voltage scaling for energy-efficient convolutional neural networks. In Proceedings of the 2017 18th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 14–15 March 2017; pp. 7–12.
43. Ganapathy, S.; Teman, A.; Giterman, R.; Burg, A.; Karakonstantis, G. Approximate computing with unreliable dynamic memories. In Proceedings of the 2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS), Grenoble, France, 7–10 June 2015; pp. 1–4.
44. Sampson, A.; Nelson, J.; Strauss, K.; Ceze, L.H. Approximate storage in solid-state memories. *ACM Trans. Comput. Syst.* **2014**, *32*, 1–23. [CrossRef]
45. Fang, Y.; Li, H.; Li, X. SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write. In Proceedings of the 2012 IEEE 21st Asian Test Symposium, Niigata, Japan, 19–22 November 2012; pp. 131–136.
46. Vallero, A.; Savino, A.; Politano, G.; Chatzidimitriou, A.; Tselonis, S.; Kaliorakis, M.; Gizopoulos, D.; Riera, M.; Canal, R.; Gonzalez, A.; et al. Early component-based system reliability analysis for approximate computing systems. In Proceedings of the 2nd Workshop on Approximate Computing (WAPCO), Accra, Ghana, 29 April 2016; pp. 1–4.
47. Castro-Godínez, J.; Esser, S.; Shafique, M.; Pagani, S.; Henkel, J. Compiler-driven error analysis for designing approximate accelerators. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1027–1032.
48. Traiola, M.; Savino, A.; Barbareschi, M.; Carlo, S.D.; Bosio, A. Predicting the impact of functional approximation: From component-to application-level. In Proceedings of the 2018 IEEE 24th International Symposium on On-Line Testing and Robust System Design (IOLTS), Platja d'Aro, Spain, 2–4 July 2018; pp. 61–64.
49. Traiola, M.; Savino, A.; Di Carlo, S. Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications. *Microelectron. Reliab.* **2019**, *102*, 113309. [CrossRef]
50. Lee, S.; Lee, D.; Han, K.; Shriver, E.; John, L.K.; Gerstlauer, A. Statistical quality modeling of approximate hardware. In Proceedings of the 17th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 15–16 March 2016.
51. Qureshi, A.; Hasan, O. Formal probabilistic analysis of low latency approximate adders. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *38*, 177–189. [CrossRef]
52. Koller, D.; Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*; MIT Press: Cambridge, CA, USA, 2009.
53. Lattner, C.; Adve, V. LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the International Symposium on Code Generation and Optimization, 2004, CGO 2004, San Jose, CA, USA, 20–24 March 2004; pp. 75–86.
54. Cozman, F.G. Generalizing variable elimination in Bayesian networks. In Proceedings of the Workshop on Probabilistic Reasoning in Artificial Intelligence, Editora Tec Art São Paulo, Brazil, 20 November 2000; pp. 27–32.
55. Jiao, J.; Juan, D.C.; Marculescu, D.; Fu, Y. Exploiting component dependency for accurate and efficient soft error analysis via probabilistic graphical models. *Microelectron. Reliab.* **2015**, *55*, 251–263. [CrossRef]
56. SMILE: Structural Modeling, Inference, and Learning Engine. Available online: https://www.bayesfusion.com/smile/ (accessed on 28 November 2019).
57. Howard, J.; Dighe, S.; Vangal, S.R.; Ruhl, G.; Borkar, N.; Jain, S.; Erraguntla, V.; Konow, N.; Riepen, M.; Gries, M.; et al. A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE J. Solid State Circuits* **2010**, *46*, 173–183. [CrossRef]
58. Gupta, V.; Mohapatra, D.; Park, S.P.; Raghunathan, A.; Roy, A. IMPACT: Imprecise adders for low-power approximate computing. In Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design, Fukuoka, Japan, 1–3 August 2011; pp. 409–414.