Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Multi-dimensional optimization for approximate near-threshold computing[*]

Jing WANG[†1], Wei-wei LIANG[1], Yue-hua NIU[2], Lan GAO[1], Wei-gong ZHANG[†‡3]
[1]*College of Information Engineering, Capital Normal University, Beijing 100056, China*
[2]*Institution of Spacecraft System Engineering China Academy of Space Technology, Beijing 100094, China*
[3]*Beijing Advanced Innovation Center for Imaging Theory and Technology, Beijing 100048, China*
[†]E-mail: jwang@cnu.edu.cn; zwg771@cnu.edu.cn

Received Feb. 26, 2020; Revision accepted May 13, 2020; Crosschecked Sept. 29, 2020

Abstract: The demise of Dennard's scaling has created both power and utilization wall challenges for computer systems. As transistors operating in the near-threshold region are able to obtain flexible trade-offs between power and performance, it is regarded as an alternative solution to the scaling challenge. A reduction in supply voltage will nevertheless generate significant reliability challenges, while maintaining an error-free system that generates high costs in both performance and energy consumption. The main purpose of research on computer architecture has therefore shifted from performance improvement to complex multi-objective optimization. In this paper, we propose a three-dimensional optimization approach which can effectively identify the best system configuration to establish a balance among performance, energy, and reliability. We use a dynamic programming algorithm to determine the proper voltage and approximate level based on three predictors: system performance, energy consumption, and output quality. We propose an output quality predictor which uses a hardware/software co-design fault injection platform to evaluate the impact of the error on output quality under near-threshold computing (NTC). Evaluation results demonstrate that our approach can lead to a 28% improvement in output quality with a 10% drop in overall energy efficiency; this translates to an approximately 20% average improvement in accuracy, power, and performance.

Key words: Approximate computing; Near-threshold computing; Output quality predictor; Energy; Performance
https://doi.org/10.1631/FITEE.2000089 **CLC number:** TP302.1

1 Introduction

The "power wall" challenge has impeded the development of computer systems. As near-threshold-voltage computing (NTC) techniques (Kaul et al., 2012; Kozhikkottu et al., 2014) which make transistors operate in the near-threshold region can lead to a considerable flexible trade-off between power and performance, developments in this area provide a new approach for solving the power wall problem. In the case of NTC, 50% of relative energy-saving is obtained at the cost of 20% performance loss; compared with super-threshold computing, the same gain incurs an immense 50% or a higher delay overhead. Although NTC has been a promising and energy-efficient solution for power-constrained environments, earlier research has nevertheless shown that this approach also encompasses significant reliability issues, including increased susceptibility to process variation. As supply voltage drops, the number of failure cells increases drastically. In particular, NTC exacerbates the sensitivity of minimum supply voltage to process variation, leading to a high error rate. Several techniques have been proposed to deal

[‡] Corresponding author

[*] Project supported by the National Natural Science Foundation of China (Nos. 62076168 and 61772350), Beijing Nova Program (No. Z181100006218093), and the Research Fund from Beijing Innovation Center for Future Chips (No. KYJJ2018008)

ORCID: Jing WANG, https://orcid.org/0000-0003-3653-7013; Wei-gong ZHANG, https://orcid.org/0000-0003-3969-5607

with increasing NTC errors, including new types of correction codes, reconfiguration, and hardware redundancies. These techniques still require near-perfect executions to guarantee correctness, and imply a high design cost at both device and microarchitecture levels.

Relaxing these requirements will enable significant savings; approximate computing technologies leverage error-tolerance properties of applications and the perceptual limitations of humans to trade off computation quality (e.g., accuracy) against computational effort (e.g., energy) in error-tolerant applications such as media processing and the emerging recognition, mining, and synthesis (RMS) applications (Reagen et al., 2018). These applications generally process inexact inputs obtained from non-traditional sources such as sensors, and the associated algorithms are often stochastic in nature; thus, these applications generally require only acceptable results instead of precise outputs.

There are opportunities to improve the performance and efficiency if requirements for absolute correctness can be relaxed. Thus, at the architectural level, conventional devices can be pushed to their operational limits by reducing voltage scaling V_{dd}. As an approximate method, this approach can provide levers that trade off quality for efficiency. A processor is able to invest more energy when accurate data processing is required, but it can save energy when less accurate processing is permitted. This effectively reduces the average energy consumption and creates an extra degree of freedom for system-level power management. At the same time, researchers have proposed numerous strategies for use in the approximate approach, including precision scaling (Tian et al., 2015), loop perforation (Sidiroglou-Douskos et al., 2011), load value approximation (Sutherland et al., 2015), skipping tasks, multiple inexact program versions, inexact faulty hardware, and neural network (NN) based accelerators. In this context, NNs exhibit significant parallelism and can be accelerated efficiently by dedicated hardware to enable performance and energy benefits. In one earlier study, Esmaeilzadeh et al. (2012) proposed the approximate computing technique that works by training an NN to mimic approximate code regions and replace the original code with an invocation of a new low-power process unit. This technique avoids changes to the instruction set architecture (ISA) of the processor, enables the use of neural acceleration in devices that are already available commercially, and accelerates a broad range of applications.

We simultaneously apply an NN-based approximation at the software level with a voltage-scaled implementation of computation in this study. The NN-based approximation will transform the approximate code region into a neural representation. The approximate code region is therefore a hot spot and is able to tolerate imprecision in operations and data. To combine NN-based approximation with voltage scaling, we leverage NN inherent fault-tolerance characteristics to determine the faults induced by a near-threshold voltage (NTV). An accuracy requirement limits the topology complexity of NN, while an aggressive reduction in complexity also has a negative effect on the output quality. As network complexity is reduced to the lower threshold limits, high energy efficiency of NTC provides an opportunity for further energy saving. However, it is extremely difficult, if not impossible, to develop a general approximate computing framework with guaranteed quality, which is applicable to all types of error-tolerant applications. We therefore study particular types of approximate strategy, and develop a specific quality-guaranteed dynamic system configuration.

The fault-tolerant capability of an NN will increase in concert with the topology complexity. This means that an adjustment in NN complexity provides an opportunity for voltage scaling. A more complicated NN will lead to more multiplications and add operations, resulting in energy and performance overheads. The challenge in this area is then to choose a proper voltage level and NN complication to obtain an optimal trade-off among output quality, energy, and performance. A multi-dimensional optimization mechanism is necessary for reaching multi-objective goals. The combined technique system used here is shown in Fig. 1. Our approach encompasses the following four steps:

1. We propose a multi-dimensional optimization approach for approximate computing. This approach simultaneously applies NTC with an NN-based approximate strategy. This enables us to take advantage of NTC energy efficiency, and to leverage NN's features to accelerate the excution and tolerate NTC-

induced faults.

2. We design a framework to forecast the performance, output quality, and energy of an NN–NTC combined approximate system. An optimization algorithm is used to obtain proper voltage and NN complication based on predicted performance, energy, and output quality.

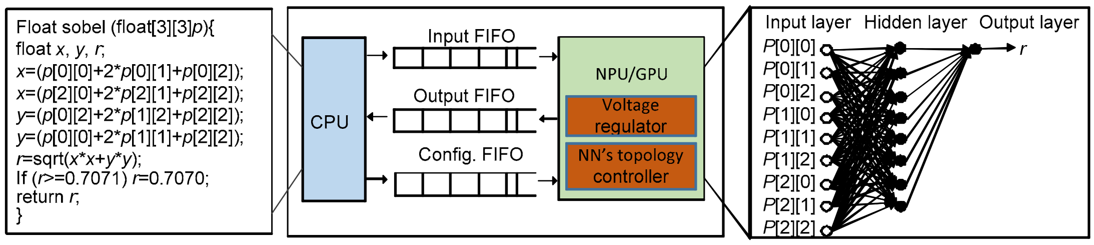
3. We propose an output quality predictor to evaluate the quality of a system. Via error injection and propagation analysis, we determine the output quality of each instruction group at a given voltage level and network topology. The quality bucket of each group at distinct approximate levels is then established. Output quality can be acquired according to the "quality bucket" distribution of an application.

4. We model power as a function of voltage level and NN complexity. This means that NN complexity can be quantified by the time of computation and memory access. We therefore design an NN model parser, which can compute the amount of arithmetic computation and memory access times given the model description and input data size.

We evaluate the energy, performance, and output quality of our approach. Results show that our optimization framework is aware of application features to tune the energy and performance breakdown of each benchmark to obtain the best trade-off among performance, energy, and output quality.

2 Multi-dimensional optimization framework

We illustrate the framework of our multi-dimensional optimization approach in Fig. 2. This framework takes a default approximate level, including voltage level and the complication of neural network architecture. We also incorporate a specific usage scenario (i.e., a platform and resource budget including energy, accuracy, and performance) that includes inputs and generates an adapted voltage and an NN architecture as outputs. The choice of voltage and NN depends on constraints of energy, accuracy, and performance. However, extracting these metrics through network training and direct hardware measurements is excessively time-consuming. Thus, to speed up this process, we bypass training and measurement processes by leveraging accuracy, latency, and energy predictors (Fig. 2). This framework contains three predictors: a performance predictor to predict system delay, an energy predictor to provide the system's total power consumption, and an output quality loss predictor (Fig. 3). Voltage and NN topology can be tuned to satisfy the requirement in performance/energy/quality budget or target. This means that voltage and NN topology are input variables, and performance, energy, and quality are functions of these two variables. Based on our framework, given the performance or energy target, a simulated



```
Float sobel (float[3][3]p){
float x, y, r;
x=(p[0][0]+2*p[0][1]+p[0][2]);
x=(p[2][0]+2*p[2][1]+p[2][2]);
y=(p[0][2]+2*p[1][2]+p[2][2]);
y=(p[0][0]+2*p[1][1]+p[2][2]);
r=sqrt(x*x+y*y);
If (r>=0.7071) r=0.7070;
return r;
}
```

Fig. 1 An NN–NTC combined approximate approach

NN: neural network; NTC: near-threshold computing; FIFO: first input first output; CPU: central processing unit; NPU: neural-network processing unit; GPU: graphics processing unit

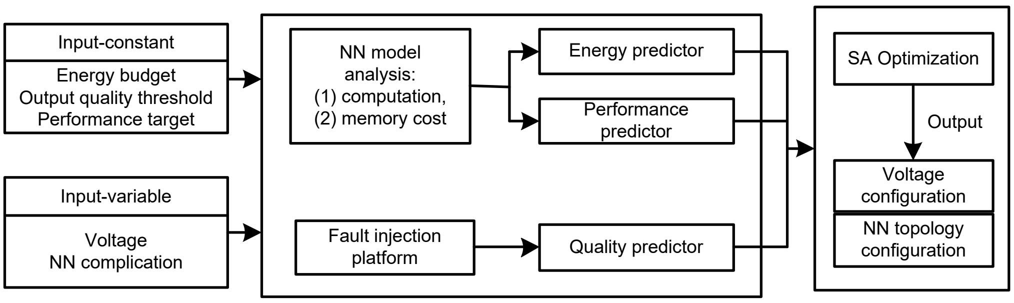


Fig. 2 Overview of the framework design (SA: simulated annealing; NN: neural network)

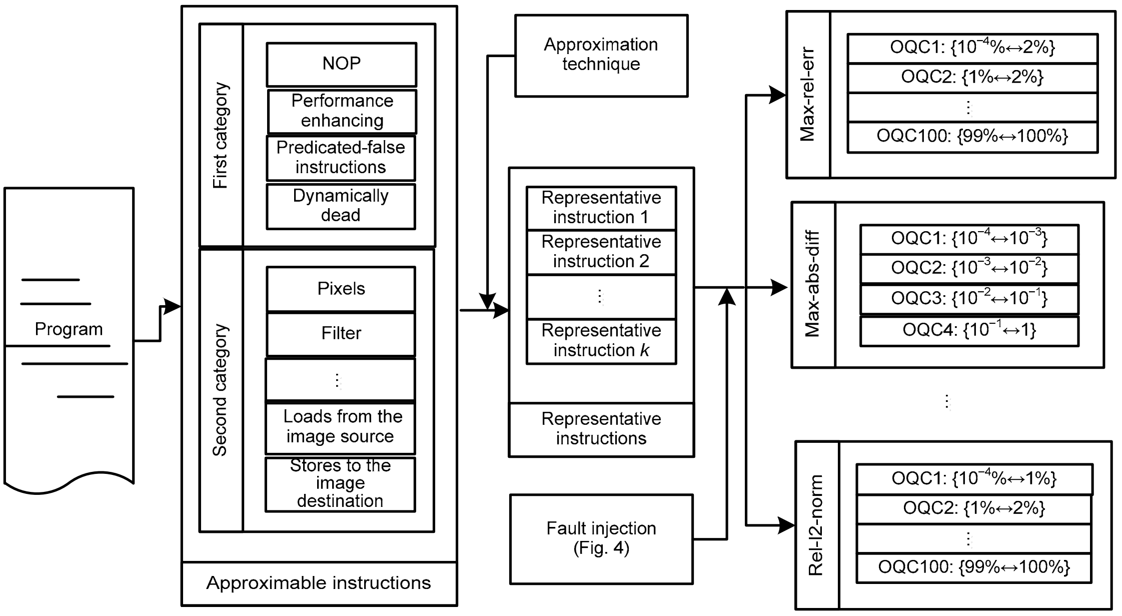

Program
First category
NOP
Performance enhancing
Predicated-false instructions
Dynamically dead
Second category
Pixels
Filter
⋮
Loads from the image source
Stores to the image destination
Approximable instructions
Approximation technique
Representative instruction 1
Representative instruction 2
⋮
Representative instruction k
Representative instructions
Fault injection (Fig. 4)
Max-rel-err
OQC1: $\{10^{-4}\% \leftrightarrow 2\%\}$
OQC2: $\{1\% \leftrightarrow 2\%\}$
⋮
OQC100: $\{99\% \leftrightarrow 100\%\}$
Max-abs-diff
OQC1: $\{10^{-4} \leftrightarrow 10^{-3}\}$
OQC2: $\{10^{-3} \leftrightarrow 10^{-2}\}$
OQC3: $\{10^{-2} \leftrightarrow 10^{-1}\}$
OQC4: $\{10^{-1} \leftrightarrow 1\}$
⋮
Rel-l2-norm
OQC1: $\{10^{-4}\% \leftrightarrow 1\%\}$
OQC2: $\{1\% \leftrightarrow 2\%\}$
⋮
OQC100: $\{99\% \leftrightarrow 100\%\}$

Fig. 3 Output quality predictor design (NOP: no operation; OQC: output quality class)

annealing (SA) optimization algorithm can be used to determine a suitable voltage level and the NN topology configuration. This can then be used to obtain the best performance, energy, and output quality trade-off.

2.1 Output quality predictor

An output quality predictor processes application outputs, and evaluates quality as a set of numerical values. As different applications have diverging characteristics, it is difficult for a single NN architecture to run optimally on all different platforms. The quality predictor must therefore be application-specific although the framework itself is general. The predictor developed here can be used to assess the impact of an instruction-level error on the output quality. We develop this approach based on Approxilyzer's (Venkatagiri et al., 2016) main insight that errors propagate "similarly" through the control and data flow paths in the program. These are then likely to generate program outputs of similar quality. Thus, instructions that can be approximated can be divided into a series of groups. To analyze the impact of errors on output quality, we design a fault injection platform and inject errors on representative approximated instructions from each group. The errors introduced into the application are derived from observations of effects of various approximate computing techniques, including the error probability, magnitude, and predictability. We also assess the impact of a single or a class of approximate computing techniques on application output quality. Fault injection based on the error probability, magnitude, and predictability generates a quality profile for the application as well as insight into its resilience.

2.1.1 Output quality analysis

The approximate technique presented here should work on code regions that can be approximated. If this is not the case, then approximation at any place could lead to catastrophic failures including out-of-bound memory accesses. In other words, approximation should never affect critical data and operations. Approximate computing techniques should therefore be targeted towards resilient computations while avoiding sensitive ones. We identify potentially resilient computations using software ACCEPT (Sampson et al., 2015). Instructions that can be approximated are classified into two categories, the first of which has no influence on the output. This category includes four instruction types: (1) "no operation" (NOP) instructions which have no influence on the architectural state; (2) performance enhancing instructions such as prefetch and branch, which predict hint instructions (the invalid execution of these, including prefetching, does not change program semantics); (3) predicated-false instructions, which discard results so that incorrect ones will not have impact on the program output; (4) dynamically dead instructions (the destination registers of these remain unused and they can therefore be considered dynamically dead).

The second category of instructions encapsulates data that are safe to approximate because of an application's fault tolerance capability. In other words, when adopting different approximate techniques or degrees, this category of instructions might lead to different impacts on outputs within an acceptable range of output quality. In multimedia applications, for example, data types like pixels and filter coefficients can be relaxed due to the perceptual limitations of users. In the Sobel edge detection algorithm, for example, instructions holding data about the filter and pixels can also be approximated (Sampson et al., 2015).

Tuning voltage requires a regulator. This means that voltage scaling is limited due to the area overhead, while changing levels is not continuous. The choice of voltage level can therefore be considered as numerable; the number of unique network topologies within a search space can still be large; however, considering the trade-off between topology complexity and accuracy, NN topology choices are therefore also numerable. If a system trades off 10% complexity for only 1% output accuracy, for example, the energy and performance overhead will overwhelm the accuracy gain. We use efficient sample generation here in combination with a predictor training method (Chippa et al., 2013). This enables us to select sample architectures from across the overall space and encompass high sampling density in the area of "samples of interest." The numbers of voltage levels and NN approximate degrees are limited, however, and thus it is reasonable to obtain the output quality at a possible voltage level and NN topology through error injection and propagation analysis. To simulate faults at a certain voltage level and an approximate degree, errors should be injected based on representative approximated instructions from each group.

The nature of output from an application varies from one to another. This means, for example, that a sobel output is an image, while one from jmeint is a boolean. It is therefore necessary to have an application-specific quality metric to effectively assess the quality loss of each application output. To quantify the output quality, three quality metrics are applied to different types of applications. The first of these is max-abs-diff, which gives the maximum absolute difference between golden and fault outputs. The second is max-rel-err, which calculates the maximum relative error between the golden and faulty outputs. The third is rel-l2-norm, a metric which directly compares two others (Yazdanbakhsh et al., 2017).

These metrics are used to quantify the effect on output quality (Fig. 3); they divide the output quality into a series of "quality buckets." In terms of each combination of approximate voltages at the hardware level and the NN architecture at an application level, we collect quality errors across all samples in the representative workload. We then take the average value of these quality metrics as the trained impact.

2.1.2 Fault injection platform

Scaling down voltage saves energy but also causes small issues and impacts output accuracy. Indeed, at a given voltage operating point, it is the case that each bit-cell exhibits a normal distribution and a random occurrence error rate, ranging between 1 and 0. Traditional fault injection frameworks based on dynamic instrumentation can experience kernel slowdowns, making them prohibitively slow to sweep many fault patterns across a range of fault rates that vary by orders of magnitude. Current software-based models also tune voltage at a coarse-grained level, and consider hardware as a whole; our method is distinct as voltage can be tuned at a fine-grained level. We therefore propose a fault injection platform based on a simulation backend to evaluate the reliability of our method (Fig. 4). The framework injects fault and tuning voltage on real hardware design in the EDA tool, and runs popular benchmarks written by C program language. This framework injects errors based on prior research into the relationship between voltage and the single-bit error rate (Karpuzcu et al., 2012), and can guide lower-level tools regarding where to conduct a detailed micro-architectural fault analysis. The hardware we simulate is a multi-core system; this encompasses a standard peripheral component interconnect (PCI), a network, and debug support unit (DSU) devices. This workload communicates with hardware through PCI and network interface, while our fault injection software communicates with hardware through DSU.

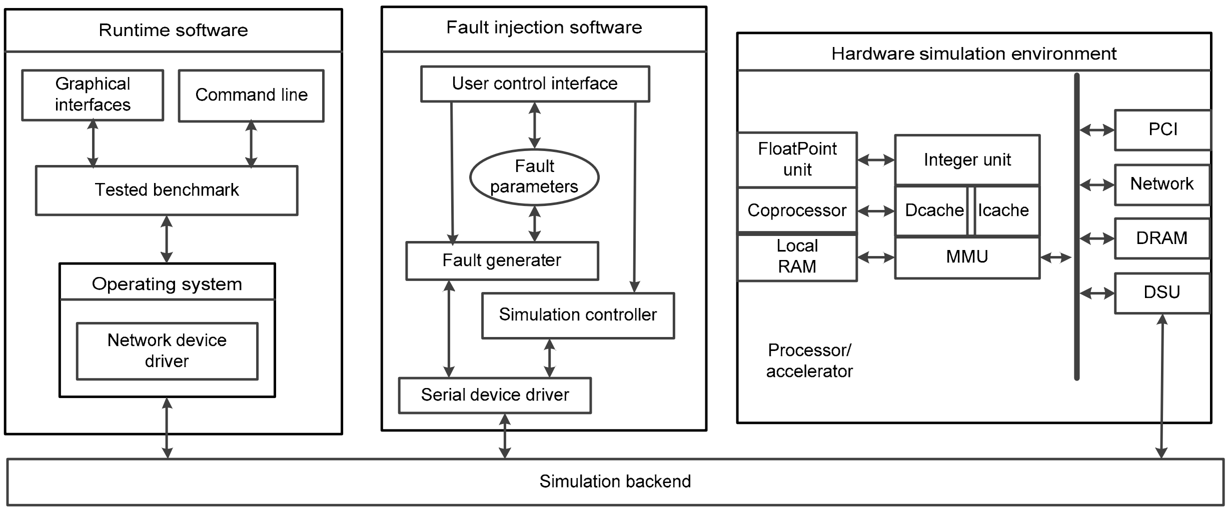


Fig. 4 Fault injection platform framework

PCI: peripheral component interconnect; RAM: random access memory; DRAM: dynamic RAM; MMU: memory management unit; DSU: debug support unit

The simulation backend comprises a connection between the hardware simulator and software system. This is implemented via a foreign language interface (FLI) within the EDA tool. Indeed, via the simulation backend, this software generates signal stimulus inputs, and an injected value is assigned to the signal logic of the target processor or accelerator in the hardware emulation environment, and receives the system status for debugging. The runtime software, including the application benchmark and the system software, interacts with end users via the graphical interface or command line. The runtime software then communicates with the underlying hardware through a network or a serial device. The fault injection software receives related parameters, including fault time and location through the user-controlled interface. This software also obtains a list of signals from hardware through a simulation backend. A depth-first search method that iterates over the list is used to identify and record signals hierarchically. Establishing a hierarchical resource pool can help users directly find the location of the fault. This software generates a fault time and location for the hardware signal. Generated fault and simulation information will be sent to the target processor. This fault injection software receives the state of the processor through the simulation backend after a fault is injected. The hardware receives software commands, and sends execution status through an internal DSU via a trace cache. This simulation backend has a virtual network card and a serial interface. The simulation backend can send the control command and fault injection information to the DSU; this then returns information to different destinations according to the content of the information. Data transmission and command scheduling are achieved through correct information distribution, while the information on this application is sent to the runtime software; fault injection information is sent to the software. In the case of voltage point, we run the entire test set on sample network architectures to obtain the classification error along with the bit error rate modeled using the software Varius-NTV (Karpuzcu et al., 2012).

2.2 Energy predictor design

We model power as a function of supply voltage and NN complexity. Extracting energy and latency of an NN architecture through direct measurement, however, remains very challenging. Platform-specific latency measurements can be slow and difficult to parallelize, particularly when the number of available devices is limited. Large-scale latency measurements might therefore prove expensive and lead to a computational bottleneck. To speed up this process, we construct a predictor for energy and latency on the target device to enable fast and reliable energy and lantency estimations for the NN candidates based on the complexity of NN, a quantification of the amount of computation and memory access times. We design an NN model parser based on the observation that the types of operators involved in the networks are limited; thus, the amount of computation and the memory access times can be formulated.

2.2.1 Power model

Energy is a key challenge for the development of the computer industry. We need to establish an energy predictor to guide multi-dimensional optimization in the NTC system. The power consumption of processing NNs includes computation cost P_c and memory access cost P_m (Wang et al., 2014). P_c can be formulated as the total power cost of multiply/add operations per clock cycle, as $P_\mathrm{c}=\mu_\mathrm{add}C_\mathrm{add}+\mu_\mathrm{mul}C_\mathrm{mul}$. In this expression, μ_add (μ_mul) and C_add (C_mul) denote the power cost per add operation (multiply operation) and the total number of add operations (multiply operations), respectively. Similarly, P_m depends on the storage scheme. The total power consumption per clock cycle, P, can be modeled as

$$P = P_\mathrm{c} + P_\mathrm{m} = \mu_\mathrm{add}C_\mathrm{add} + \mu_\mathrm{mul}C_\mathrm{mul} + \mu_\mathrm{mem}C_\mathrm{mem}, \quad (1)$$

where μ_mem and C_mem denote the power cost per bit when accessing memory and the total number of bits, respectively. The power cost of one neuron, μ_i (i=1, 2, …), is considered to be a function of supply voltage v_i: $\mu_i \propto v_i^2 \beta_i$, where β_i is a hardware-specific constant. We therefore use the linear approximation method (Tavakkoli-Moghaddam et al., 2008) to obtain the value of β_i, including β_add, β_mul, and β_mem of the adder, multiplier, and memory, respectively. Moreover, after neuron-level voltage scaling, the voltages of the adder, multiplier, and memory are different per operation. We therefore extend Eq. (1) to

$$\begin{aligned} P = &\sum v_j^2 \left(\beta_\mathrm{add} \cdot C_{\mathrm{add}_j} + \beta_\mathrm{mul} \cdot C_{\mathrm{mul}_j}\right) \\ &+ v_\mathrm{mem} \sum \beta_\mathrm{mem} \cdot C_\mathrm{mem}, \end{aligned} \quad (2)$$

where v_j denotes the number of different voltages allocated on the j^th neuron (j=1, 2, …) and v_mem is the voltage of the memory system, unified for all neurons. The numbers of add and multiply operations are calculated based on the analysis of network models.

2.2.2 Neural network parser

Modeling of power reveals that this variable is a function of the supply voltage and complexity of the NN. In other words, prediction of power depends on the estimation of NN computation and memory access analyses. Although deep NNs (DNNs) can derive different network models by changing their hierarchical structures as well as the number of and connections between neurons, types of operators (layers) involved in the networks are limited, and generally include convolution, full connection, and pooling. Thus, the computation of each layer is determined regardless of how the input data change, and computation and memory access can be formulated. We propose an NN model parser; given a model description, this parser can compute the amount of arithmetic computation and memory access times.

The NN model consists of operators with different functions. The arithmetic computation type, computation amount, and memory access are different for each type of operator. We analyze six classical NNs, including Alexnet, Inception, VGG19, VGGish, single shot multibox detector (SSD), and Attention, and find 10 typical operators. As previously mentioned, the operator computation features and memory access features are fixed for each layer in DNN. We formulate the numbers of architecture operations and memory access times for each layer; this formulation is a function of input data size and operator parameters such as the size of the convolution kernel and stride of the two-dimensional convolution layer. We save the formulation of each operator in a feature description file.

The model parser we present analyzes the number of layers and the execution order. A text file description of the NN model is then converted into a representation of the general model through the TensorBoard visualization component embedded in TensorFlow. This general model representation contains operator types, number of operators, and their order. In this context, each layer is regarded as one node, and the input and output between them are regarded as the directed edge; thus, the network model transforms to a directed acyclic graph (DAG).

The second step is NN computation and memory access feature extraction. We use the graph search algorithm to traverse DAG and calculate the amount of computation and memory access times of each node based on the formulations in the feature description file. The operator feature extraction process is as follows. First, we determine whether the operator is defined in the feature description file. In this case, it has been loaded and we calculate the number of basic operations such as multiplication and addition using the input data size and the formula definition of the operator in the feature description file. The second step is to detecte whether there is a built-in feature analysis method inside the system; if so, call the extraction function to parse the operator; otherwise, the null feature is output. Finally, the parser gives the amount of arithmetic computation and memory access times, which can be used to predict the power of the approximate system.

2.3 Performance predictor

We use the minimum delay of NN to represent performance. In general, the reduction in voltage can adversely affect the system performance due to the increasing time to complete a task. Moreover, the delay of execution depends on the hardware mapping scheme of neurons. As the structure of NN becomes increasingly complex, it is difficult for all neurons to execute in parallel for a layer at the same time. The fixed dimensions of the computational grid may need some layers to be partitioned, as the dimensions of the layer may be smaller than, match, or exceed the size of hardware. In cases where the layers' dimensions exceed the function unit size, the function units must be used iteratively to compute different parts of these layers, and we call each use of all on-chip function units a "run." Then, the final computational delay is the sum of all runs. For each run, the lowest voltage of the neurons in the workload will determine the maximum delay, represented by the delay at normal supply voltage D_i multiplied by the magnification of the delay under a low voltage. Hence, the delay of the i^{th} layer is the sum of the maximum delays for j execution runs of this layer, as

$$\begin{aligned}&\text{perlayer\_max}\{D\}_i\\&=\max\left\{D_1\cdot G_1, D_2\cdot G_2,\cdots,D_j\cdot G_j\right\}.\end{aligned}\tag{3}$$

Finally, the maximum delay of NN, which has n layers, is the accumulation of each layer's delay, and can be formulated as

$$\text{NN\_Delay}=\sum_{i=1}^{n}\text{perlayer\_max}\{D\}_i.\tag{4}$$

In particular, the delay of each layer can be decomposed into computation delay D_c and memory access delay D_m. Then, D_m can be classified into two categories, buffer access delay D_b and main memory access on buffer misses D_miss. The first part of data access latency D_b may overlap with D_c. Owing to the limited space of weight and data cache, D_miss cannot be hidden by D_c. Therefore, the maximum delay of each layer is expressed as

$$D_i=\max\left\{\{D_\text{c}\}_i,\{D_\text{b}\}_i\right\}+\{D_\text{miss}\}_i,\tag{5}$$

where i denotes the i^{th} layer in the NN. The calculation of delay is based on a single delay model of the logic gate using NN, which comprehensively captures the process, voltage, and temperature variation along with the input slew and output load. As the memory access pattern on a specific hardware platform is fixed and the miss rate can be predicted before the inference process, we can calculate D_miss.

3 Optimization

Real-world applications face very different constraints. In one example, real-time video frame analysis may have a very strict latency constraint, whereas Internet-of-Things (IoT) edge device designers may care more about runtime energy consumption for a longer battery life. It is infeasible to have one approximate degree that meets all these constraints. This makes it necessary to adapt the approximate degree to the specific use scenarios before deployment. Our optimization is targeted to find the best selection of voltage levels and the NN topology for applications to obtain the best three-dimensional (3D) trade-off.

We adopt two case studies as examples in this approach. In the first case, we minimize delay given accuracy, performance, and power constraints. In this case, P–E-R means that performance is the objective function while energy and output quality are constraint conditions, and P–E means that performance is the objection while energy is the constraint. This is consistent with previous work but without considering output quality (Azizi et al., 2010). Secondly, we maximize the power saving given accuracy and performance constraints. E–P-R represents the fact that while energy is the objective function, performance and output quality are constraints. E–P represents the fact that energy is the objective function while performance is the constraint. E–P does not consider output quality though.

Taking P–E-R as an example, the optimization problem can be solved using the SA algorithm, which is a stochastic optimization approach based on the Monte-Carlo iterative solution strategy. SA simulates the optimization problem as an annealing process of solid matter in physics. This finds the global optimal solution of the objective function as temperature decreases. The greatest advantage of SA is that it can jump out of the local optimal solutions and tends ultimately to the global optimal solution. The optimization problem is converted into exploring the solution space, and then finding the best approximate degree by a range of moves. Algorithm 1 shows the process of solving the P–E-R case. We initialize the system with a series of configurations $C=\{c_1, c_2, \ldots, c_n\}$, where c_i ($i=1, 2, \ldots, n$) is a combination of the voltage level and the NN topology.

Algorithm 1 Approximate degree selection

```
Initialize: $C \leftarrow C_0$; $T \leftarrow T_0$.
1  Calculate IPS←IPS_PREDICTOR($C$);
2  while $T>\varepsilon$ do
3    while the number of moves<$M$ do
4      $C_{\text{new}}$=ANNEAL_MOVE(C);
5      $\text{Performance}_{\text{new}}$←IPS_PREDICTOR($C_{\text{new}}$);
6      $\text{ENERGY}_{\text{new}}$←ENERGY_PREDICTOR($C_{\text{new}}$);
7      OQ $\text{LOSS}_{\text{new}}$←OQ_PREDICTOR($C_{\text{new}}$);
8        if ($\text{Performance}_{\text{new}}$>Performance)∩($\text{Energy}_{\text{new}}$
             <$\text{Energy}_{\text{budget}}$)∩($\text{OQLOSS}_{\text{new}}$<
             $\text{OQLOSS}_{\text{threshold}}$)∪($\text{IPS}_{\text{new}}$ reduces with
             a certain probability) then
9            $C \leftarrow C_{\text{new}}$; Performance←$\text{Performance}_{\text{new}}$;
10       end if
11   end while
12   $T \leftarrow \alpha T$;
13 end while
```

The algorithm maximizes Performance through annealing moves ANNEAL_MOVE(), which can verify whether $\text{Performance}_{\text{new}}$ is better than the old Performance, whether $\text{Energy}_{\text{new}}$ is lower than $\text{Energy}_{\text{budget}}$, and whether $\text{OQLOSS}_{\text{new}}$ is lower than $\text{OQLOSS}_{\text{threshold}}$. Thus, if $\text{Performance}_{\text{new}}$ is worse, the algorithm decreases the annealing temperature. This can also make occasional uphill moves to better explore the solution space. The algorithm can finally generate a configuration which can maximize the performance and guarantee the energy budget and quality threshold at the same time.

4 Evaluation

4.1 Experiment setup and metrics

We evaluate our optimization framework by running the Axbench benchmark, which covers a diverse set of domains such as machine learning, scientific computation, signal processing, image processing, robotics, and compression. AxBench comes with the necessary annotations to mark the approximate region of the code and the application-specific quality metric to assess the output quality of each application. Conventional voltage scaling sets normal voltage to 1.1 V and 810 mV for low-voltage environments. We further scale the voltage down to the threshold voltage, which is set based on two principles: first, V_{dd} should stop scaling before transistors encounter the first uncorrectable error; second, V_{dd} should support the minimum frequency to ensure that the application performance in terms of throughput is not reduced significantly when the voltage decreases from super-threshold computing (STC) to NTC. Silvano et al. (2014) conducted experiments considering the effect of variation of threshold voltage on transistor delay, and mentioned that the minimum frequency is 400 MHz to guarantee performance. Thus, in our detailed evaluation, we perform the evaluation in a 22-nm technology with the lowest V_{dd} as 600 mV, and the minimum frequency of 400 MHz at NTV. We use the Sniper simulator (Carlson et al., 2011) to evaluate the performance. Sniper is a parallel, high-speed, and accurate simulator. It allows fast and accurate simulation and trading off simulation speed for accuracy to allow a range of flexible simulation options when exploring different homogeneous and heterogeneous multi-core architectures. We employ HSIM-VCS co-simulation, which contains the fault injection module, to verify the voltage tuning effect on power and reliability. In this study, we use Varius-NTV to model the reliability under NTC. Varius-NTV is a microarchitecture-level model to study the failure modes at NTC with the impact of process variation. Moreover, to quantify the trade-off among accuracy, power, and performance, we define a metric EEPI to select the optimal solution to achieve a high accuracy with low power consumption and delay:

$$\text{EEPI} = \text{Error} \cdot \text{Energy} \cdot \text{Performance}. \qquad (6)$$

4.2 Application resilience and energy-efficiency analysis

Each application has its unique error-resilient behavior, and we try to capture those features that differentiate an application from others. This is achieved by analyzing the application sensitivity at the approximate level from the accuracy and energy efficiency perspectives. EPI is a metric expressing the energy efficiency, and it can be obtained through the consumption of energy divided by performance request (IPS). The error expresses the output quality loss rate. We use MSE to quantize the NN complexity. Fig. 5 demonstrates the change in EPI. As we can observe, the EPIs of fft and jpeg change dramatically, indicating that they are sensitive to the approximate degree from the perspective of energy efficiency. Fig. 6 demonstrates how the approximate degree impacts the accuracy. jeint and jpeg are flat; therefore, they are not sensitive to the change in approximate degree. fft, inversek2j, and blackscholes are sensitive to the change in approximate degree.

To summarize, a four-quadrant figure (Fig. 7) illustrates that a different benchmark shaves a different susceptibility to the change in approximate degree from the accuracy and energy efficiency perspectives. The high degree of resilience existing in jpeg and jmeint applications emphasizes the scope and potential of approximate computing.

4.3 Optimization results and analysis

To examine the effectiveness of our strategy, we evaluate two cases, the low-power case and high-performance case. To analyze the potential of our strategy under different workloads, we run three combinations of workloads: (1) error-susceptible-dominant workloads, which are composed mainly of error-sensitive applications, such as fft; (2) error-non-susceptible-dominant workloads, which are composed mainly of error-insensitive workloads, such as jpeg; (3) balanced workloads, where the numbers of error-sensitive workloads and error-insensitive workloads are equal.

4.3.1 Low-power case study

In this subsection, we evaluate the output quality and EEPI under different performance thresholds. In

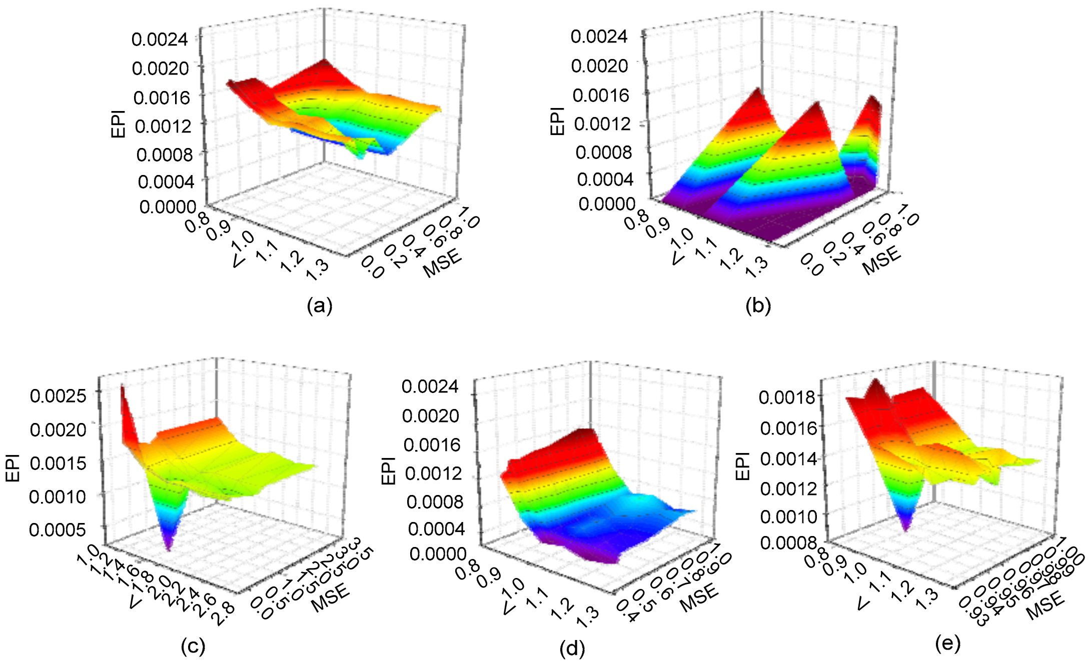


Fig. 5 The degree of approximation and voltage level influence on EPI: (a) blackscholes; (b) fft; (c) inversek2j; (d) jpeg; (e) jmeint

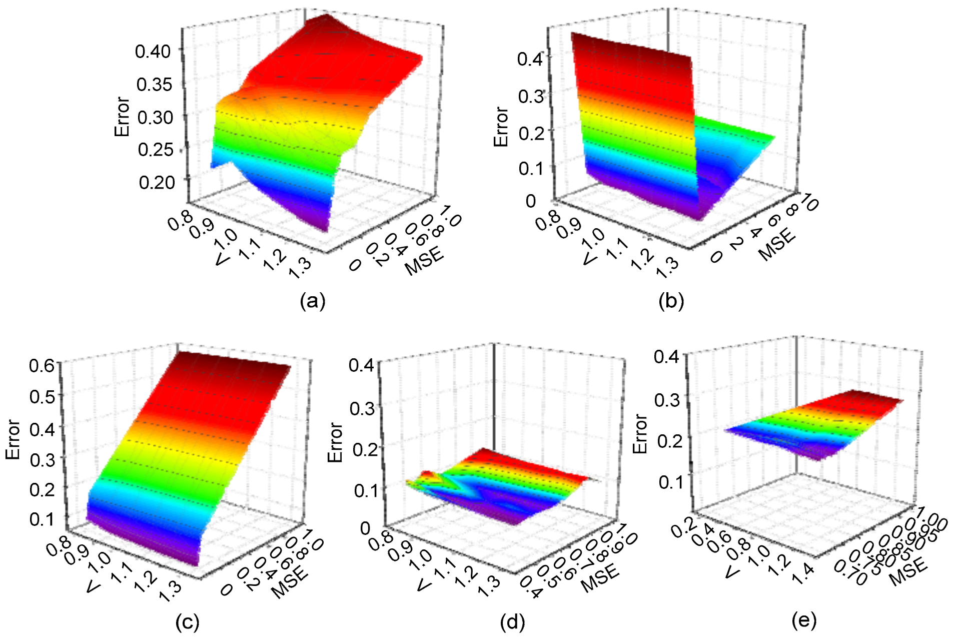


Fig. 6 The degree of approximation and the voltage level influence on the error: (a) blackscholes; (b) fft; (c) inversek2j; (d) jpeg; (e) jmeint

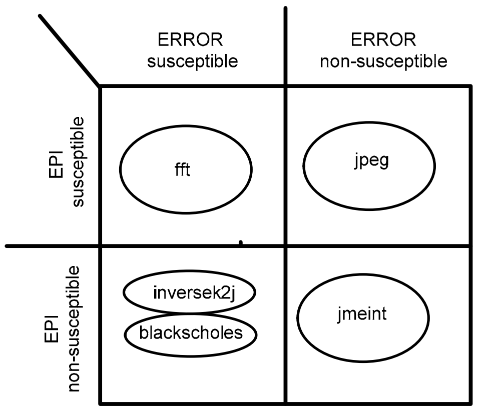


Fig. 7 Susceptibility to EPI and ERROR

Fig. 8, the *x* axis represents the performance request, ERROR represents the output quality, and EEPI represents the trade-off among energy, performance, and output quality. As we can observe, when running "balanced workload," our method can obtain a 30% improvement in output quality with a 10% drop in overall energy efficiency, compared with the E–P-baseline, and the EEPI for "balanced workload" can improve by 20% on average. When running "error-susceptible-dominant workload," our method can obtain a 33% improvement in output quality, with a 6% drop in overall energy efficiency, and EEPI can improve by 25% on average. The above experiment results show that our 3D optimization can obtain significant improvement in output quality with a little energy/performance overhead.

4.3.2 High performance case study

As illustrated in Fig. 9, we estimate the output quality loss and EEPI for all the workloads under different power budgets, adopting case 2 optimization schemes in Table 1. As we can observe, when running the "balanced workload," our method can obtain a 28% improvement in output quality with an 11% drop of overall energy efficiency compared with the P–E-baseline. The EEPI for "balanced workload" can improve by 19% on average. When running "error-susceptible-dominant workload," our method can obtain a 30% improvement in output quality with a 7% drop of overall energy efficiency, and the EEPI for "error-susceptible-dominant workload" can improve by an average of 23%. When running "error-non-susceptible-dominant workload," our method can obtain a 25% improvement in output quality with a 12% drop in overall energy efficiency. The EEPI for "error-non-susceptible-dominant workload" can improve by 16% on average. The above experiment results show that our 3D optimization can achieve better performance, energy, and output quality trade-off.

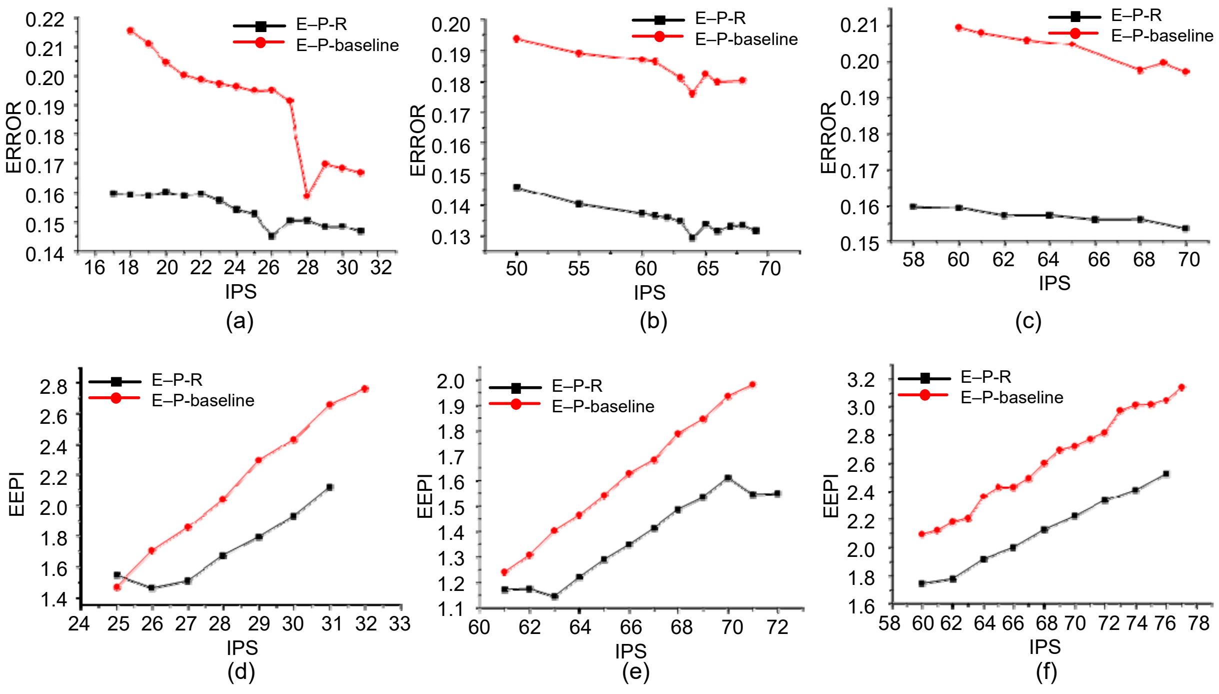


Fig. 8 Energy efficiency, reliability, and EEPI of each workload adopting the E–P-R or E–P optimization scheme under different performance requests: (a) ERROR for balanced workload; (b) ERROR for error-susceptible-dominant workload; (c) ERROR for error-non-susceptible-dominant workload; (d) EEPI for balanced workload; (e) EEPI for error-susceptible-dominant workload; (f) EEPI for error-non-susceptible-dominant workload

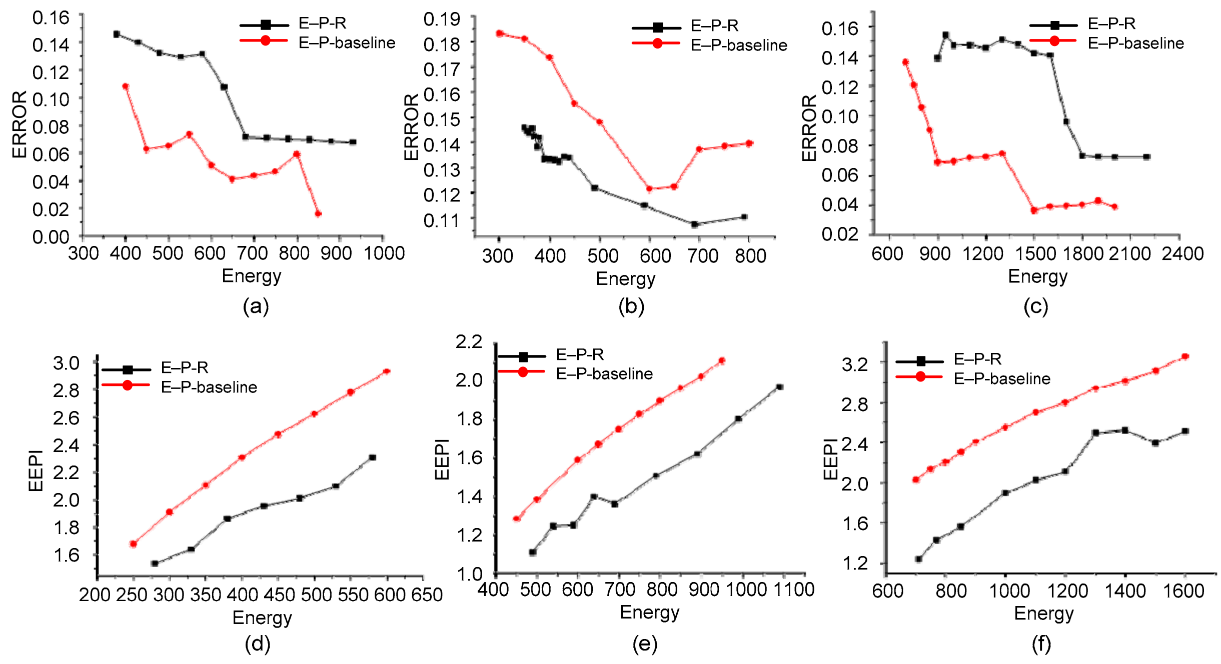


Fig. 9 Energy efficiency, reliability, and EEPI of each workload adopting the P–E-R or P–E optimization scheme under different power budgets: (a) ERROR for balanced workload; (b) ERROR for error-susceptible-dominant workload; (c) ERROR for error-non-susceptible-dominant workload; (d) EEPI for balanced workload; (e) EEPI for error-susceptible-dominant workload; (f) EEPI for error-non-susceptible-dominant workload

Table 1 Schemes to minimize energy consumption or maximize performance

Case	Target	Constraint	Strategy
1	Performance	Energy and reliability	P–E-R
		Energy	P–E
2	Energy	Performance and reliability	E–P-R
		Performance	E–P

4.3.3 Power breakdown analysis

We evaluate the energy breakdown of each benchmark with different performance budgets to analyze why E–P-R can obtain a better trade-off compared with the E–P-baseline. The results are shown in Fig. 10. The energy consumption of the error-sensitive application should not decrease; otherwise, the user experience will become worse because of the loss of accuracy. Compared with the E–P-baseline, the energy consumed by jpeg increases by 24% on average, while the energy consumed by jmeint decreases by 16% on average. From the application characteristic analysis, jpeg and jmeint are both error-insensitive, while jpeg is EPI-sensitive and jmeint is EPI-insensitive; i.e., when we adopt the same approximate level, jpeg has a larger potential for energy reduction than jmeint; thus, by the optimization mechanism more energy is distributed to jpeg to achieve a better trade-off. We can conclude that our framework can selectively adjust the optimization effect when more difficult or precise application computations are needed.

5 Related work

The challenge of the "power wall" has impeded the development of computer systems. The NTC technique is developed as a new direction to solve the problem. To obtain a better trade-off between performance and energy consumption in the NTC system, Teodorescu and Torrellas (2008) proposed variation-aware algorithms for application scheduling and power management. They used linear programing to find the best voltage and frequency levels for each core by maximizing the performance at a given power budget. Azizi et al. (2010) gave an architecture-circuit framework to obtain energy-performance trade-off. Santriaji and Hoffmann (2016) proposed a hard-ware control system for GPU to minimize the energy consumption while guaranteeing the performance.

However, NTC faces new challenges of reliability. To address the reliability problem, software-

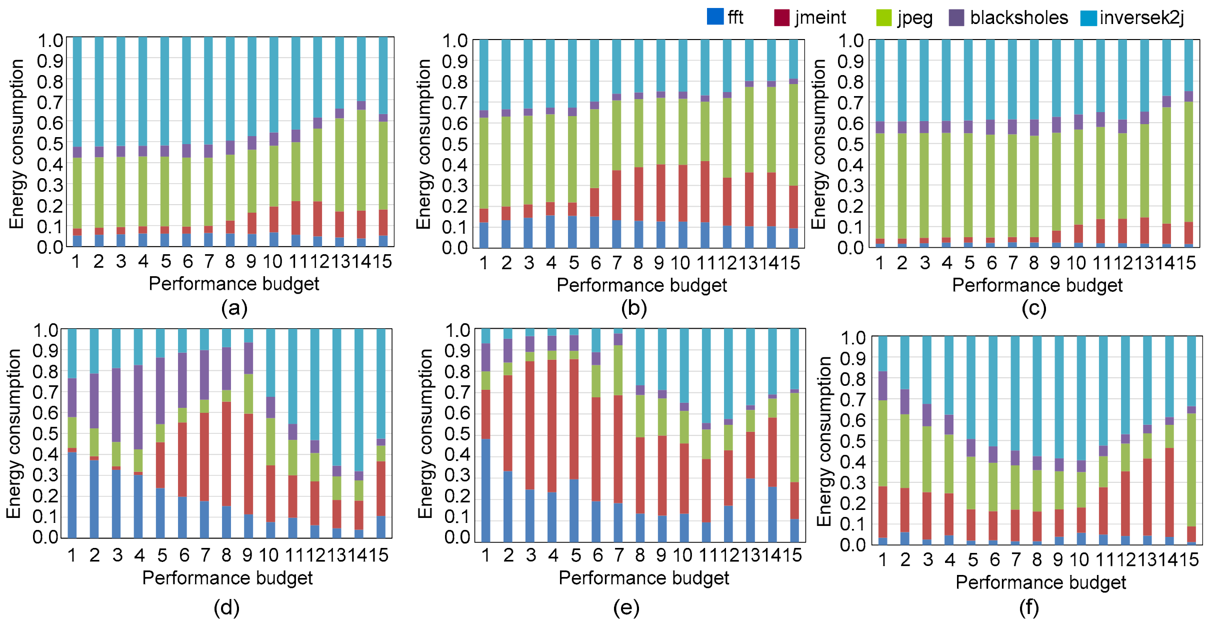


Fig. 10 The energy breakdown of each workload: (a) E–P-R for balanced workload; (b) E–P-R for error-susceptible-dominant workload; (c) E–P-R for error-non-susceptible-dominant workload; (d) E–P for balanced workload; (e) E–P for error-susceptible-dominant workload; (f) E–P for error-non-susceptible-dominant workload (References to color refer to the online version of this figure)

level (Huang and Abraham, 1984), architecture-level (Tavakkoli-Moghaddam et al., 2008), process-level redundancy (Shye et al., 2007; Ferreira et al., 2011), and circuit- or logic-level (Das et al., 2009) techniques have been proposed. Zhang and Chakrabarty (2006) achieved fault tolerance by check pointing, and obtained energy saving by dynamic voltage scaling. Zhao et al. (2008) proposed on-line (dynamic) algorithms that detect early completions and adjust the task frequencies at runtime to improve the overall reliability. PEARL is a novel modeling framework which investigates the problem of assigning optimal voltage-frequency settings to individual segments within example workflows (Wang et al., 2015). Song et al. (2015b) presented two variance reduction techniques for proactive reliability management: proportional dynamic voltage-frequency scaling (DVFS) and coordinated thread swapping. Dynamic reliability variance management (DRVM) (Song et al., 2015a) was proposed to obtain the trade-offs among energy, performance, and reliability.

The target of previous methods is to keep the system error free. This will incur high costs on both performance and energy consumption. Nowadays, many researchers have proposed approximate computing to relax the effort to guarantee reliability (Liu et al., 2011; Grigorian et al., 2015; Zhong, 2015; Wunderlich et al., 2016). These approximate techniques can also obtain energy-performance trade-off. The research on approximate techniques can be classified into three categories: circuit level, architectural level, and algorithmic level. IMPACT (Gupta et al., 2011) was designed for approximate computing at the circuit level. Esmaeilzadeh et al. (2012) selected and trained an NN to mimic a region of imperative code. They designed an NPU to accelerate the execution of the offloading approximate code region at the architecture level. At the algorithmic level, Paraprox can make different data patterns adopt different approximation methods through looking up the table (Samadi et al., 2014).

The above optimization methods focus on voltage scaling, but do not take the approximate technique into account to reduce the overhead. To guarantee the output quality, the approximate technique tunes only the approximate degree, but does not consider voltage scaling. We propose a multi-dimensional optimization mechanism to tune the approximate degree and voltage scaling together to obtain a better trade-off among energy, performance, and output quality.

6 Conclusions

We proposed a multi-dimensional optimization approach for approximate computing to reduce the fault-tolerant overhead in an NTC system. This approach combines the NTC- and NN-based approximate strategies. We took advantage of the energy efficiency of NTC, and leveraged the features of NN to accelerate the execution and tolerate the NTC-induced faults. We designed a framework to forecast the performance, output quality, and energy of this NN–NTC combined approximate system. Then, an optimization algorithm was used to obtain the proper voltage and NN complication based on the predicted performance, energy, and output quality. The proposed output quality predictor determines the output quality of each instruction group at a given voltage level and network topology through error injection and propagation analysis. We modeled the power as a function of the voltage level and NN complexity. The complexity of NN was quantified by the amount of computation and memory access times, which was obtained by an NN model parser. The evaluation results showed that these two approximation techniques can be employed in a synergistic manner; our optimization framework can be aware of the features of different applications and tune the energy and performance allocation of each benchmark to obtain the best trade-off.

Contributors

Jing WANG and Wei-gong ZHANG designed the research. Wei-wei LIANG, Yue-hua NIU, and Lan GAO processed the data. Jing WANG and Wei-wei LIANG drafted the manuscript. Lan GAO helped organize the manuscript. Wei-gong ZHANG and Yue-hua NIU revised and finalized the paper.

Compliance with ethics guidelines

Jing WANG, Wei-wei LIANG, Yue-hua NIU, Lan GAO, and Wei-gong ZHANG declare that they have no conflict of interest.

References

Azizi O, Mahesri A, Lee BC, et al., 2010. Energy-performance tradeoffs in processor architecture and circuit design: a

marginal cost analysis. *ACM SIGARCH Comput Arch News*, 38(3):26-36.
https://doi.org/10.1145/1816038.1815967

Carlson TE, Heirman W, Eeckhout L, 2011. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. Proc Int Conf for High Performance Computing, Networking, Storage and Analysis, p.1-12.
https://doi.org/10.1145/2063384.2063454

Chippa VK, Chakradhar ST, Roy K, et al., 2013. Analysis and characterization of inherent application resilience for approximate computing. 50th ACM/EDAC/IEEE Design Automation Conf, p.1-9.
https://doi.org/10.1145/2463209.2488873

Das S, Blaauw D, Bull D, et al., 2009. Addressing design margins through error-tolerant circuits. 46th ACM/IEEE Design Automation Conf, p.11-12.
https://doi.org/10.1145/1629911.1629917

Esmaeilzadeh H, Sampson A, Ceze L, et al., 2012. Neural acceleration for general-purpose approximate programs. 45th Annual IEEE/ACM Int Symp on Microarchitecture, p.449-460.
https://doi.org/10.1109/MICRO.2012.48

Ferreira K, Stearley J, Laros JH, et al., 2011. Evaluating the viability of process replication reliability for exascale systems. Proc Int Conf for High Performance Computing, Networking, Storage and Analysis, p.1-12.
https://doi.org/10.1145/2063384.2063443

Grigorian B, Farahpour N, Reinman G, 2015. BRAINIAC: bringing reliable accuracy into neurally-implemented approximate computing. IEEE 21st Int Symp on High Performance Computer Architecture, p.615-626.
https://doi.org/10.1109/HPCA.2015.7056067

Gupta V, Mohapatra D, Park SP, et al., 2011. IMPACT: IMPrecise adders for low-power approximate computing. IEEE/ACM Int Symp on Low Power Electronics and Design, p.409-414.
https://doi.org/10.1109/ISLPED.2011.5993675

Huang KH, Abraham JA, 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Trans Comput*, C-33(6):518-528.
https://doi.org/10.1109/TC.1984.1676475

Karpuzcu UR, Kolluru KB, Kim NS, et al., 2012. VARIUS-NTV: a microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. IEEE/IFIP Int Conf on Dependable Systems and Networks, p.1-11.
https://doi.org/10.1109/DSN.2012.6263951

Kaul H, Anders M, Hsu S, et al., 2012. Near-threshold voltage (NTV) design—opportunities and challenges. Proc 49th Annual Design Automation Conf, p.1149-1154.
https://doi.org/10.1145/2228360.2228572

Kozhikkottu V, Venkataramani S, Dey S, et al., 2014. Variation tolerant design of a vector processor for recognition, mining and synthesis. Proc Int Symp on Low Power Electronics and Design, p.239-244.
https://doi.org/10.1145/2627369.2627636

Liu S, Pattabiraman K, Moscibroda T, et al., 2011. Flikker: saving DRAM refresh-power through critical data partitioning. Proc 16th Int Conf on Architectural Support for Programming Languages and Operating Systems, p.213-224. https://doi.org/10.1145/1950365.1950391

Reagen B, Gupta U, Pentecost L, et al., 2018. Ares: a framework for quantifying the resilience of deep neural networks. Proc 55th ACM/ESDA/IEEE Design Automation Conf, p.1-6.
https://doi.org/10.1109/DAC.2018.8465834

Samadi M, Jamshidi DA, Lee J, et al., 2014. Paraprox: pattern-based approximation for data parallel applications. Int Conf on Architectural Support for Programming Languages and Operating Systems, p.35-50.
https://doi.org/10.1145/2541940.2541948

Sampson A, Baixo A, Ransford B, et al., 2015. ACCEPT: a Programmer-Guided Compiler Framework for Practical Approximate Computing. Technical Report No. UW-CSE-15-01, University of Washington, USA.

Santriaji MH, Hoffmann H, 2016. GRAPE: minimizing energy for GPU applications with performance requirements. 49th Annual IEEE/ACM Int Symp on Microarchitecture, p.1-13. https://doi.org/10.1109/MICRO.2016.7783719

Shye A, Moseley T, Reddi VJ, et al., 2007. Using process-level redundancy to exploit multiple cores for transient fault tolerance. 37th Annual IEEE/IFIP Int Conf on Dependable Systems and Networks, p.297-306.
https://doi.org/10.1109/DSN.2007.98

Sidiroglou-Douskos S, Misailovic S, Hoffmann H, et al., 2011. Managing performance vs. accuracy trade-offs with loop perforation. Proc 19th ACM SIGSOFT Symp and 13th European Conf on Foundations of Software Engineering, p.124-134. https://doi.org/10.1145/2025113.2025133

Silvano C, Palermo G, Xydis S, et al., 2014. Voltage island management in near threshold manycore architectures to mitigate dark silicon. Design, Automation & Test in Europe Conf & Exhibition, p.1-6.
https://doi.org/10.7873/DATE.2014.214

Song W, Mukhopadhyay S, Yalamanchili S, 2015a. Architectural reliability: lifetime reliability characterization and management of many-core processors. *IEEE Comput Arch Lett*, 14(2):103-106.
https://doi.org/10.1109/LCA.2014.2340873

Song W, Mukhopadhyay S, Yalamanchili S, 2015b. Managing performance-reliability tradeoffs in multi-core processors. IEEE Int Reliability Physics Symp, p.3C.1.1-3C.1.7. https://doi.org/10.1109/IRPS.2015.7112707

Sutherland M, San Miguel J, Enright Jerger N, 2015. Texture cache approximation on GPUs. University of Toronto, Toronto, Canada. http://www.eecg.toronto.edu/~enright/TexCacheApprox.pdf

Tavakkoli-Moghaddam R, Safari J, Sassani F, 2008. Reliability optimization of series-parallel systems with a choice of redundancy strategies using a genetic algorithm. *Reliab Eng Syst Saf*, 93(4):550-556.

https://doi.org/10.1016/j.ress.2007.02.009

Teodorescu R, Torrellas J, 2008. Variation-aware application scheduling and power management for chip multiprocessors. Int Symp on Computer Architecture, p.363-374. https://doi.org/10.1109/ISCA.2008.40

Tian Y, Zhang Q, Wang T, et al., 2015. ApproxMA: approximate memory access for dynamic precision scaling. Proc 25th Edition on Great Lakes Symp on VLSI, p.337-342. https://doi.org/10.1145/2742060.2743759

Venkatagiri R, Mahmoud A, Hari SKS, et al., 2016. Approxilyzer: towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. 49th Annual IEEE/ACM Int Symp on Microarchitecture, p.1-14. https://doi.org/10.1109/MICRO.2016.7783745

Wang L, Rivers JA, Gupta MS, et al., 2014. Resilience and real-time constrained energy optimization in embedded processor systems. 10th Workshop on Silicon Errors in Logic-System Effects.

Wang L, Vega AJ, Buyuktosunoglu A, et al., 2015. Power-efficient embedded processing with resilience and real-time constraints. IEEE/ACM Int Symp on Low Power Electronics and Design, p.231-236. https://doi.org/10.1109/ISLPED.2015.7273519

Wunderlich HJ, Braun C, Schöll A, 2016. Pushing the limits: how fault tolerance extends the scope of approximate computing. IEEE 22nd Int Symp on On-line Testing and Robust System Design, p.133-136. https://doi.org/10.1109/IOLTS.2016.7604686

Yazdanbakhsh A, Mahajan D, Esmaeilzadeh H, et al., 2017. AxBench: a multiplatform benchmark suite for approximate computing. IEEE Des Test, 34(2):60-68. https://doi.org/10.1109/MDAT.2016.2630270

Zhang Y, Chakrabarty K, 2006. A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems. IEEE Trans Comput-Aid Des Int Circ Syst, 25(1):111-125. https://doi.org/10.1109/TCAD.2005.852657

Zhao BX, Aydin H, Zhu DK, 2008. Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems. IEEE Int Conf on Computer Design, p.633-639. https://doi.org/10.1109/ICCD.2008.4751927

Zhong LL, 2015. BROAD: Bold and Reliable Online Approximate Computing Framework for Diverse Applications. MS Thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA.