计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2018.20180295
55(9):2002-2015,2018

近似存储技术综述

吴 宇[1] 杨 涓[1] 刘人萍[1] 任津廷[1] 陈咸彰[1,2] 石 亮[1] 刘 铎[1]
[1](重庆大学计算机学院 重庆 400044)
[2](重庆大学通信学院 重庆 400044)
(wuyucqu@163.com)

Survey on Approximate Storage Techniques

Wu Yu[1], Yang Juan[1], Liu Renping[1], Ren Jinting[1], Chen Xianzhang[1,2], Shi Liang[1], and Liu Duo[1]
[1](*College of Computer Science, Chongqing University, Chongqing* 400044)
[2](*College of Communication, Chongqing University, Chongqing* 400044)

Abstract With the rapid development of cloud computing and Internet of things, how to store the explosively growing data becomes a challenge for storage systems. In tackling this challenge, approximate storage technology draws broad attention for its huge potential in saving the cost of storage and improving the system performance. Approximate storage techniques trade off the accuracy of the outputs for performance or energy efficiency taking advantages of the intrinsic tolerance to inaccuracies of many common applications. In this way, the applications improve their performance or energy efficiency while meeting the user requirements. Therefore, how to exploit the features of storages and fault-tolerant applications to improve data access performance, decrease space overhead, and reduce energy consumption is becoming a key problem for storage systems. In this paper, we first introduce the definition of approximate storage technology and show the techniques for identifying the approximate areas in the data. Then, we elaborate the approximate storage techniques for CPU cache, main memory, and secondary storage, respectively. We discuss the advantages and disadvantages of these approximate storage techniques along with the corresponding application scenarios. In the end of this paper, we summarize the features of approximate storage techniques and discuss the research directions of approximate storage techniques.

Key words approximate storage; high performance; error tolerance; efficient access; energy saving

摘 要 随着云计算和物联网业务的快速发展,如何存储爆发式增长的数据成为存储系统的一个巨大挑战.为解决这一问题,近似存储作为一种解决存储资源紧张的必要手段越来越受到关注,它通过利用某些应用程序固有的容错特性,在输出结果的精度和应用的性能间进行权衡,以在满足用户需求的同时提升性能和能效.因此,如何针对不同的存储与应用的特点,通过近似存储数据解决访问性能低、空间开销大和能耗高等问题,已成为存储系统的研究热点.首先介绍近似存储技术的定义与近似区域的识别技

收稿日期:2018-04-16;**修回日期**:2018-06-08
基金项目:国家自然科学基金面上项目(61672116);重庆市基础与前沿研究项目(cstc2016jcyjA0332);中国博士后科学基金项目(2017M620412)
This work was supported by the General Program of the National Natural Science Foundation of China (61672116), the Chongqing Basic and Frontier Research Program (cstc2016jcyjA0332), and the China Postdoctoral Science Foundation (2017M620412).
通信作者:陈咸彰(xzchen@cqu.edu.cn)

术;接着分别阐述适用于高速缓存、内存和外存3个存储层次的近似存储技术,并分析其优缺点与应用范围;最后总结近似存储的特点,并探讨存储系统中近似存储技术的进一步研究方向.

关键词　近似存储;高性能;容错;高效访问;节能

中图法分类号　TP391

随着云计算与物联网业务的快速发展,Dennard缩放定律[1]及Moore定律带来的功耗与性能优化逐渐难以满足新兴应用对计算机系统性能与功耗不断增长的需求.同时,物联网等新兴应用产生的信息量呈指数型增长趋势,使得计算机需要存储爆炸增长的数据,对存储系统的性能、功耗和可用存储资源带来巨大的挑战.

为了解决存储领域面临的资源短缺问题,Google、微软和阿里巴巴等大规模数据密集型企业建立配备有数千台高性能服务器的大型数据中心.然而,大型数据中心能耗极高,每年需耗费上千亿千瓦时电量,服务器数量的增长率仅为数据量增长率的五分之一[2],简单地扩大存储规模不能解决存储资源紧张问题.此外,现代计算机系统的处理器通过多核并行计算的方式,理论上能以指数提高计算效率,然而存储器访问数据的物理极限导致并行计算的效率不能以指数提升[3].资源过度配置的传统方法不能有效地解决计算机存储资源紧张的问题.因此,空间开销大、能耗高、访问性能低已成为存储领域面临的巨大挑战.

近似存储技术作为缓解存储难题的一个重要途径,近年来受到越来越多的关注.作为近似计算的子领域,近似存储严格遵循近似计算牺牲输出质量以换取高性能的思想.与传统的精确存储数据技术不同,近似存储技术利用应用程序固有的容错特性,通过近似存储数据或读取近似数据的方式,在满足用户需求的条件下降低应用的输出质量,从而减少存储空间开销、提升访问效率和降低存储能耗的优势.

当前广泛使用的大量应用,如图像渲染、信号处理、增强现实、数据挖掘、语言识别、多媒体处理、机器学习和传感数据分析等,对不精确的输出具有内在容忍度[4-9],用户可以接受降低一定质量的输出结果.因此,计算机系统可以针对这些应用使用近似存储技术,获取性能、能耗、存储寿命等方面的优化.

近似存储技术是近年来存储相关工作的研究重点,将近似应用于程序运行的存储阶段能提升存储访问效率、增大存储容量、降低存储能耗,未来可能成为解决存储领域难题的必要手段.

1　近似存储技术的研究背景

1.1　近似存储的定义

近似存储作为一种解决存储资源紧张的必要技术,利用应用程序固有的对不精确输出的容错特性,通过近似存储数据或读取近似数据的方式,权衡输出结果精确度和应用性能,以在满足用户需求的同时提升性能和能效.

近似存储是近似计算的子领域,近似计算的思想是通过降低输出质量以获取应用的高性能,降低输出质量的方式是通过改变程序运行过程中的计算模式,而近似存储是只针对计算模式中与存储相关的部分,包括数据的存储与读取.目前,针对近似技术的研究大多不是只针对存储,但有相当一部分研究涉及到近似存储,本文主要针对涉及存储并非只针对存储的近似技术作介绍.

图1展示了近似存储技术与传统存储技术的差异.从应用领域看,近似存储技术只适用于容错应用,而传统存储使用于容错应用和精确应用;从数据的处理方式看,近似存储可以近似地缓存数据或者从内存近似加载指令,传统存储只能缓存精确的数

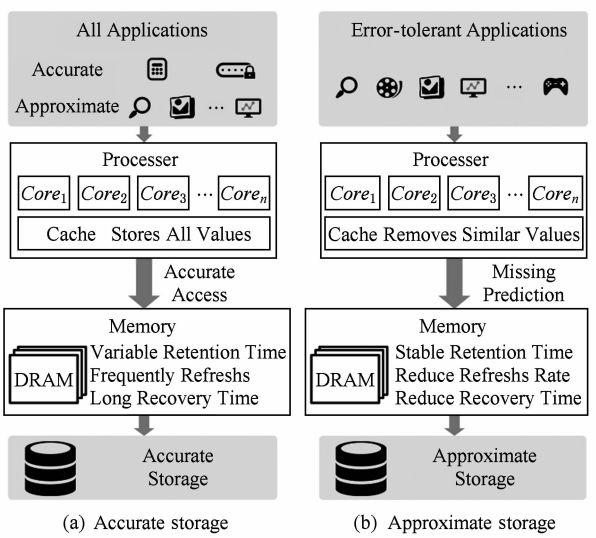


Fig. 1　Differences of accurate and approximate storage

图1　精确存储与近似存储方式差异

据;从数据的存储方式看,近似存储可以在主存和外存中存储近似数据,而传统存储必须存储精确的数据.

近似存储最根本的因素是应用程序能容忍不精确的输出.诸如数字信号处理,图像、音频和视频处理,无线通信,网络搜索和数据分析等应用在识别、挖掘和合成(mining and synthesis, RMS)[10]等新兴应用领域都能容忍不精确质量的输出.近似存储技术的应用领域如表1所示.近似存储具有无限潜力,推动计算的发展以适应信息高速增长的变化.利用应用程序的容错性质提升性能、节省能源,缓解存储资源紧张的难题.

Table 1 Approximate Storage Technology Applications

表1 近似存储技术应用领域

Application Area	References
Mathematics	ISA Support for Approximate[11], RFVP[12], STAxCache[13], DrMP[14], Optimization of Approximate Kernels[15]
Games	Approximate Storage[5], ISA Support for Approximate[11], Flikker[16]
Multimedia	Approximate Storage[5], RFVP[11], STAxCache[13], DrMP[14], Approximate Kernels[15], Flikker[16], Load Value Approximator[17], QuARK[18], RAIDR[19], High-Density Image Storage[20], PTC[21], JPEG[22], JPEG XR[23], Optimization of Memory Storage Technology[24]
Data analysis	Approximate Storage[5], RFVP[12], STAxCache[13], DrMP[14], Optimization of Approximate Kernels[15], Load Value Approximator[17], QuARK[18], RAIDR[19], Uncertain 〈T〉[25]
Route optimization	Flikker[16], QuARK[18], RAIDR[19], Uncertain 〈T〉[25]
Recognition	Approximate Storage[5], ISA Support for Approximate[11], STAxCache[13], Uncertain 〈T〉[25]
Machine learning	RFVP[12], DrMP[14], RAIDR[19], Uncertain 〈T〉[25]
Language processing	RFVP[12], Flikker[16]
Signal processing	RFVP[11], STAxCache[12]

应用程序内在的容错因素来源于2个方面:1)应用程序的数据集存在显著的噪声和冗余[4,26-27].现实世界纷繁复杂、大量重复的信息以及通过传感器等不精确器件采集数据都导致输入数据集冗余.在输入数据存在噪声和冗余的情况下,精确计算与存储得到的结果本质上是非精确的,同时浪费大量的资源.2)程序本质的计算模式,如统计聚合、迭代改进等会衰减或纠正由近似引起的误差[4].

1.2 近似存储的发展动力

近似存储牺牲了应用输出结果的精度,与传统的追求高精度输出结果的观点相悖,但是近似存储仍然作为一种新兴的技术在存储领域快速发展,并越来越受到关注.近似存储的发展动力主要体现在4个方面:

1) 海量数据存在大量冗余.从移动设备到数据中心,数据量不断增长,所需的存储资源不断增加,但是存储资源的增长速度不能与信息的增长速度相提并论.这些爆炸式增长的数据存在大量冗余,极大地浪费了紧缺的存储资源.如何在不增加存储资源的情况下,利用有限的存储资源存储更多的数据被越来越多的研究人员所关注.利用近似存储可以有效地解决数据冗余,提升存储空间的利用率.

2) I/O瓶颈问题有待解决.简单地增加存储器资源不能从根本解决存储资源紧张的问题,反而会由于存储器容量的增加,加大整体的能耗.且在计算能力与需求不断提高的现状下,处理器与存储器之间的性能差距也越来越大,I/O瓶颈问题也越来越严重.利用近似存储可以减小存储器带宽与处理器带宽的差距.

3) 存储器件需要更高的稳定性.深入到存储器内部,随着半导体技术进入纳米级别,电路集成度不断提高,电路集成度越高越容易受到环境的影响.alpha粒子、宇宙射线的照射等环境是造成瞬时故障的主要原因,据统计,存储系统中发生的故障98%都是瞬时故障[28].电路对具有先进技术节点的参数变化和故障敏感度会随着电路集成度的增加而增加,传统的无故障计算为实现容错和纠错需要在不同级别的层次上设计保护带和冗余,浪费了大量的能源[26].因此可以通过近似存储技术降低数据精度要求,节约能源,获得性能的提升.

4) 用户能容忍不精确输出.对用户而言,由于人类感知能力有限,完全精确和近似精确的计算结果可能没有任何差异.例如有损压缩利用人类对图像或声波中的某些频率成分不敏感的特性,允许压

缩过程中损失一定的信息;视频播放过程中某一特定帧的丢失可能不会被用户发现.没有唯一的黄金输出的限制,对应用的改进提供了动力.与人类感官相关的应用,放松精度的限制,提供足够精确的结果而不是绝对精确的结果,对性能的提升非常必要.用户的关注重点不一定是应用程序的精确度,如智能手机、平板电脑等常用的日常设备,人们对电池的续航时间的关心通常会超过对某一应用程序的精确度.

2　近似存储概述

近似存储作为近似计算的分支领域,其一般过程与近似计算的一般过程[26]相似,可分为3个阶段:近似区域识别[29]、执行近似存储操作、评估近似输出质量.近似存储作为一种能有效解决存储资源紧张的关键技术,并非适用于所有的存储数据.本节将简要介绍近似存储的一般过程和近似存储的重要前提,即近似区域识别,并简要总结近似存储及近似计算的相关技术.

2.1　近似存储的一般过程

近似存储作为近似计算的一个分支,其过程与近似计算的过程类似,可以分为3个阶段,如图2所示.识别可近似区域并标识可近似代码段是近似存储的前提,对近似区域数据执行近似存储操作是核心,控制与评估输出质量是对近似存储方案的可靠性保证.本节将介绍近似存储一般意义上的3个阶段,并对涉及的相关技术作简单总结.

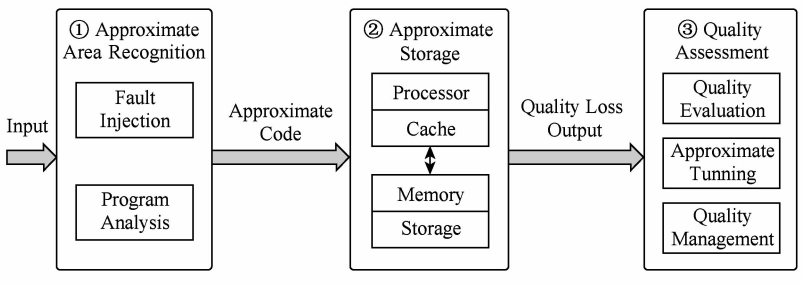


Fig. 2　The process of approximate storage

图2　近似存储的一般过程

表2分别对近似区域识别技术、近似存储技术、近似计算技术、质量控制与评估方案作总结.近似区域识别总结了具有代表性的一些技术,无论近似计算还是近似存储,都需要先识别近似区域,表2中介绍的技术基本适用于近似存储;近似存储总结近似计算中与涉及到存储的技术,表2中所列举的技术有的是部分涉及存储,如文献[16,24,30]等都不只是针对存储作近似;近似计算列举一些与存储关联

Table 2　Approximate Storage Related Technique Summary

表2　近似存储相关技术总结

Technology Type	References
Approximate Area Recognition	Analysis of Application Resilience[4], Application-Level Fault Tolerance[6], Relyzer[31], Approxilyzer[32], Shoestring[33], Assuring Application-Level Correctness[34], EnerJ[35]
Approximate Storage	Approximate Storage[5], ISA Support for Approximate[11], RFVP[12], STAxCache[13], DrMP[14], Optimization of Approximate Kernels[15], Flikker[16], Load Value Approximator[17], QuARK[18], RAIDR[18], High-Density Image storage[20], PTC[21], JPEG[22], JPEG XR[23], Memory storage technology[24], Uncertain ⟨T⟩[25], Partially-Forgetful Memories[36], Texture Cache[37], Cache for Approximate[38], Tunable Cache[39], SRAM for Error-Tolerant Applications[40], Dynamic Energy-Quality[41], Scratchpad Memory Optimizate[42], Precision-Aware Soft Error[43], Efficient Reliabilit DRAM[44], Lossy Memory I/O[45]
Approximate Computing	Neural Acceleration for General Approximate Programs[6], Neural Acceleration for GPU[7], Architecture Support Approximation[11], Optimization of Approximate Kernels[15], SAGE[46], DES[47], SNNAP[48], Neuralizer[49], BRAINIAC[50], Precimonious[51], The Fusion of Orchestrated Services[52], Quality Programmable Vector Processors[53], Optimization of Floating-point[54], Approximate Adder[55], Approximate Multiplier[56], Accelerators for Machine-Learning[57], Approximate Logic Synthesis[58]
Quality Control and Evaluation	Analysis of Application Resilience[4], Application-Level Fault Tolerance[6], Approxilyzer[32], SAGE[46], DES[47], Relyzer[48], Statistical Guarantees[59], Proactive Control[60], Quality of Service Profiling[61], Rumba[62], Reduce Runtime Fault Recovery Overhead[63]

较小甚至不涉及存储的近似技术,近似存储作为近似计算的小部分,涉及的技术远少于近似计算,这部分内容扩展对近似存储的认识,不作过多介绍;质量控制与评估总结一些具有一定通用性的评估方案,表 2 中所列举的评估方案都不是只针对近似存储,一般来说更适用于近似计算,通常这些近似计算中涉及到存储方面的近似工作.

1) 识别近似区域. 识别近似区域代码段是近似存储技术的必要前提. 应用程序的代码段,并非都能作近似处理,因此需要标识近似代码段,然后对其执行近似存储操作. 近似区域代码通常有 2 个特征:①代码段能容忍错误,因为具有容错特性的近似代码段在执行近似操作后输出质量降低不明显;②代码段的执行频率高,一般地说,高频执行的代码占整体开销的比重大,近似存储能提升程序的整体性能并减少程序运行的整体开销. 将在 2.2 节详细地介绍近似区域识别技术.

2) 执行近似存储操作. 近似存储技术使可近似代码段能近似存储或从存储器获取近似数据. 程序执行过程中,存储操作占据大部分开销,通过近似存储,达到减少能耗、节约资源、提高效率的目的. 虽然识别可近似代码段有额外开销,但通过近似存储能有效地减少大量开销,从而在整体上降低能耗. 将在第 3 节更详细地介绍近似存储技术.

3) 控制与评估输出质量. 近似存储的方法本质上是通过降低质量来换取更高的效率和更少的资源消耗,因此必须控制与评估近似输出结果,确定输出质量是否可接受. 输出质量的评估标准有 2 种:①由开发人员直接指定输出质量阈值,这比较简便;②被终端用户可接受的输出. 实现第 2 个评估标准具有挑战性,因为不同的用户对输出质量的标准不同,因此难以确定统一的输出质量阈值.

实际上并非所有的近似存储或近似计算过程都严格按照以上流程实现,但所有近似计算和近似存储都必须经历以上过程. 有的近似计算方案在执行近似操作的过程中监视输出质量,并不断回滚来调整输出质量,如 SAGE[46]、动态工作度量 DES[47];上述使用故障注入方法识别近似区域的近似计算,在确定近似区域代码段时候直接根据输出质量决定. 而 SNNAP[48], Neuralizer[49], BRAINIAC[50], Precimonious[53] 和专用 SAGE 静态编译器[46] 等近似计算技术都遵循上述流程. 文献[64]中的近似存储方法也遵循上述流程,该方法为可近似的数据设计近似内存,通过维护故障映射表,将近似数据映射到近似内存执行. 表 2 列举了近似存储的相关技术,包括近似区域的识别、质量控制与评估、近似存储技术与近似计算技术. 将在第 3 节更详细地介绍近似存储技术.

2.2 近似区域的识别

作为近似计算的重要分支,近似存储的一个重要步骤也是对近似区域的识别. 即便是可近似的应用程序,也存在某些程序段在执行近似操作后导致应用输出存在较大误差,产生用户无法接受的结果. 所以,近似存储的一个关键问题是要安全地区分可近似代码段与不可近似码段,保证只对可近似代码区域执行近似存储操作. 识别近似区域有 2 种方法[26,65],即故障注入与程序分析. 不同的近似存储技术应根据需求,选择合适的近似区域识别方法标识可近似代码段.

2.2.1 故障注入

故障注入,即向原始程序注入故障,观察注入故障后对程序运行时的影响,注入故障后,不会导致程序崩溃或运算结果出错的代码段标识为可近似的. 故障注入的方法具有一般性,但耗时较长,它先向原始程序注入故障,然后观察它对程序执行结果的影响. 注入的故障覆盖范围越广、故障类型越多,分析得到的弹性代码段越精确. 故障注入的方法经历了 3 个阶段:

1) 随机故障注入. Li 等人提出的故障注入方法[6]向 3 个硬件结构,即物理寄存器文件、获取队列和问题队列,随机注入故障.

2) 代表性故障注入. Chippa 等人提出的 ARC[4] 框架选择具有代表性的指令进行故障注入实验,描述近似代码段的特征.

3) 等价类故障注入. Relyzer[31] 方法与 ARC 不同,仅针对等价类错误进行故障注入,对那些检测到故障与输出结果为无声数据损坏(silent data corruption, SDC)的代码段又利用静态和动态的控制流和数据流启发式方法捕获到不同指令类型的相似故障,把它们归为等价类. Venkatagiri 等人在 Relyzer 基础上提出了 Approxilyzer[32],关注所有高精度动态指令的 1 位错误对输出结果的影响.

2.2.2 程序分析

程序分析,一般是根据程序的语法和语义,通过观察与分析,找到程序的可近似区域代码. 相对于故障注入,程序分析虽然不具备一般性,但是效率更高,如 Feng 等人提出的 Shoestring[33], Cong 等人[34] 和

Sampson 等人[35]提出的程序分析方法.文献[34]中的程序分析方法无法分析所有的系统故障,文献[35]中的方法只针对 Java 语言.

2.3　质量控制与评估

近似存储的方法通过降低质量以换取更高的效率和更少的能耗,然而质量不可能无限制地降低,因此必须控制与评估近似输出结果的质量,确定输出质量是否可接受.

输出质量的控制与评估很少有通用的设计方案,一般地说,评估工作与近似存储技术紧密相关,无论是近似计算还是近似存储,都根据需求设计相应的评估方案.动态的输出质量控制方案在程序执行过程中监视输出质量,并不断回滚来调整输出质量,如 SAGE[46]、动态工作度量 DES[48]等.

输出质量的评估标准有 2 种:1)由开发人员直接指定输出质量阈值,这比较简便,本文列举的近似存储技术基本都是采用这种方案;2)被终端用户可接受的输出,AxGames[66]是一种通用的评估用户可接受质量损失阈值的方案.实现第 2 个评估标准具有挑战性,因为不同的用户对输出质量的标准不同,因此难以确定统一的输出质量阈值.

3　近似存储技术

近似存储技术通过少量降低程序执行的质量,达到提高存储访问性能、减少空间开销或降低能耗的目的.现有近似存储技术分别适用于高速缓存、主存、外存 3 个存储层次,它们分别用于优化存储访问效率、降低空间开销、提高可靠性、减少存储单元的磨损或降低能耗.表 3 总结了近年来具有代表性的近似存储技术,从存储层次上对这些技术进行归纳,并总结这些技术的主要改进目标.

Table 3　Summary of Approximate Storage Techniques

表 3　近似存储技术总结

Application Layer	Approximate Storage Technique	Access Efficiency	Reduce Space Overhead	Improve Reliability	Reduce Wear	Save Energy
Cache	RFVP[16], Load Value Approximation[17]	√				
	Texture Cache[37], Cache For Approximate [38]	√	√			
	A Tunable Cache[39]		√	√		
	Dynamic Energy-Quality[41]	√				√
	Scratchpad Memory Optimizate[42]		√			√
	Precision-Aware Soft Error [43]			√		
	STAxCache[13], QuARK[18]	√	√	√	√	√
Memory	Flikker[16], RAIDR[19]	√		√		√
	DrMP[14]	√				√
Storage	High-Density Image Storage[20], PTC[21], JPEG[22], JPEG, XR[23],		√			
	Approximate Storage[5]	√			√	
Mixed Layer	Memory Storage Technology[23], ISA Support for Approximate[10]	√	√			
	Approximate Kernels [14], Uncertain ⟨T⟩[24]	√				√

Note: "√" represents the main improvement goals for approximate storage technology.

如图 3 所示,近似存储技术按所属的存储层次可分为 3 类:1)面向高速缓存的近似存储技术,主要研究如何提高高速缓存的访问效率和减少高速缓存的存储开销;2)面向内存的近似存储技术,主要针对 DRAM 动态刷新存储单元的特点优化访存性能与存储能耗;3)面向外存的近似存储技术,主要针对闪存特性优化数据存储开销和存储单元的磨损.

3.1　针对高速缓存的近似技术

缓存作为处理器与存储器之间的交换缓冲区,具有访问速度快的特点,但是缓存的高速访问仍然无法跟上处理器的处理速度,同时还存在缓存容量小和能耗高的缺点.利用近似存储技术能克服缓存的这些缺点,本节将针对如何提高访问效率、减小能耗和缓存存储开销,介绍高速缓存的近似存储技术.

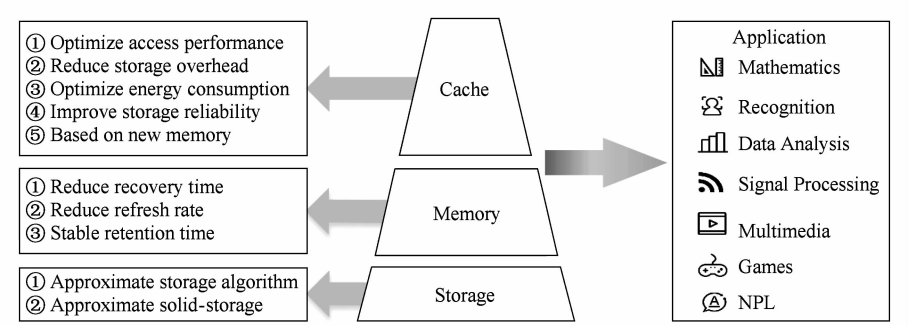


Fig. 3 Approximate classification based on storage hierarchy

图 3 基于存储层级的近似技术分类

英特尔出产的芯片多处理器(chip multiprocessor, CPM)中,高速缓存已占芯片面积的 30%[67],现代处理器利用多层缓存层次结构提高访问效率,大型服务器应用程序工作集通过构建更大的缓存来提高性能,但都未从根本上解决问题. 因此,利用近似存储技术降低高速缓存的能耗、提高性能势在必行.

3.1.1 提高存储访问效率

由于处理器的并行处理,导致存储带宽无法跟上处理器带宽. 为了减少带宽差距,需要提高存储器的访问效率. 利用近似存储技术,在缓存缺失时近似估计需要从存储器加载的数据,隐藏缓存未命中导致的延迟,提高存储访问的效率.

Load value 近似器[16]作为一种高效存储访问技术,主要针对图形应用程序中的 load 指令在访问缓存缺失时执行近似操作,执行过程如图 4 所示. 在编译阶段,访问高速缓存未命中时,不需要从内存读取数据,而是利用 load value 近似器近似地预测数据,从而不间断地在处理器中执行程序,大大减少了内存访问延迟. 与传统预测器不同,访问缓存未命中时,近似器会同时访问内存,将读取到的数据作为 load value 近似器的输入进行训练,以评估近似值预测的正确性. 因为图形应用程序的高容忍度,能够接受不精确的输出,因此可以消除回滚机制. 如果近似器的精度足够高,还可以放松置信区间对缺失值进行估计. load value 近似器在输出质量忽略不计的情况下,能显著节约能源和提高效率.

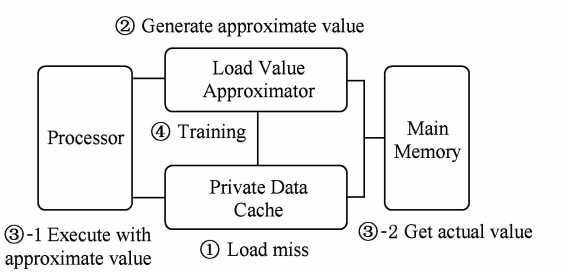


Fig. 4 Load value approximation overview[16]

图 4 Load value 近似器概述[16]

RFVP[11]也是一种预测缺失值来提高存储器访问效率的近似存储技术,其原理是在 L1 cache 缺失时预测缺失值来避免检测与重新执行,且仅针对那些执行次数较多却能导致大量缓存缺失率的 load 指令进行预测. 因为 GPU 中的每个数据请求都是单指令多数据流(single instruction multiple data, SIMD),它为多个并发线程生成数据,为每个线程执行预测会产生大额开销,因此 RFVP 利用相邻线程访问值的相似性来设计仅具有 2 个并行专用预测器的多值预测器. 多值预测器合成了 2 个 two-delta[36]预测器,一个用于线程 0~15,另一个用于线程 16~31. RFVP 技术能提高存储访问效率,解决处理器与存储器带宽延迟的问题.

3.1.2 降低缓存存储开销

高速缓存虽然访问速度快,但其容量较小,经常出现缓存访问缺失的情况,但由于缓存价格昂贵以及工艺的局限性,不可能无限制扩大缓存,在有限容量的缓存中存储更多的数据是解决这一问题的有效方案,在传统的 cache 器件和新型的非易失性器件中都能设计近似存储方案降低数据存储的开销.

1) 基于传统缓存的相似值存储

基于相似值的解决方案主要是利用缓存值的相似性,删除重复值和冗余数据,减少存储数据.

Sutherland 等人提出的近似存储技术[37]适用于各种数据密集型应用,它利用 load value 近似器和 texture 硬件生成近似值. 其中 texture 是位于线程和高速缓存之间的存储单元,能够在多个相邻高速缓存条目之间插值. 利用 texture 的内插功能,使用浮点数据作为高速缓存的索引. 基于空间局部性与时间局部性,可以选取 2 个从全局存储器连续读取的值之差进行近似,其中差值用训练集生成.

在数兆字节的最后一级缓存(last level cache, LLC)中,许多相同或相似的值可能会跨多个块缓存,极大地浪费了缓存的容量. Miguel 等人观察到最后一级高速缓存中大多数的缓存值表现出相似性,具体来说,跨缓存块的值不相同但是相似,若近似地将相似的数据看作冗余数据,删除重复和冗余数据,可更好的利用缓存的存储空间. 由此,Miguel 等人设计了一种新颖的 LLC 架构——Doppelganger[38],一种专门适用于近似计算的缓存架构,将存在相似值的高速缓存块(即使跨缓存块也可以)作为单个数据块存储在缓存中,其整体架构如图 5 所示. 通过识别相似的高速缓存块,并把这些块与数据数组中的单个条目关联起来,使得多个块共享一个数据条目,虽然这牺牲了数据的精确度,但却能在更小的数据阵列中存储更多非冗余的数据.

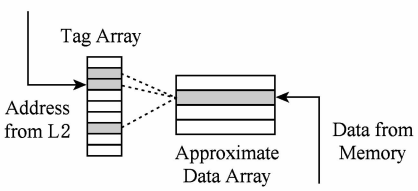


Fig. 5　Doppelganger: Tags of approximately similar blocks are associated with the same data entry[38]

图 5　Doppelganger 将相似块近似到相同数据条目上[38]

2) 基于新型器件的近似值存储

随着工艺特征尺寸的逐渐减小,互补金属氧化物半导体(complementary metal oxide semiconductor, CMOS)缩放面临着诸多挑战,如高泄漏功耗、低可靠性、低密度等问题. 研究人员正开发新的器件来取代它,新型器件为计算机系统的设计带来了新的机会. 基于新型器件缓存架构的近似值存储方案能有效解决上述问题.

Sjalander 等人提出的新型器件缓存架构[39],可以存储近似数据和精确数据,根据片上高速缓存的存储资源对精度需求的不同,以可调精度的方式对资源进行分配. 当应用程序放松对数据的精度的要求时,数据会以四进制的形式近似存储在四元存储单元,精确的数据以二进制的形式存储在存储单元中,精确的数据需要两倍的存储单元数量. 高速缓存行只能存储近似数据或精确数据,不能混合存储 2 种数据. 为了支持四进制的近似存储架构,他们设计了新的寻址方案. 将近似计算与新兴多值设备结合,使相同的近似存储单元数存储更多的数据.

3.1.3　降低缓存能耗

SRAM 单元主要用作高速缓存,它使用晶体管存储数据,存在高泄漏功耗的问题,降低电源电压的近似存储方案能节约能源.

Kumar 对降低 SRAM 电源电压进行研究发现,其错误率主要由读取翻转和写入失败的概率决定[68]. 对 90 nm 和 60 nm 工艺下的 SRAM 采取电源电压减低技术可以降低 60%的泄漏功率. 电源电压缩放是降低能耗非常受欢迎的技术,但为保证数据的正常执行,SRAM 的最小电源电压 V_{min} 应高于电源电压的阈值 V_{DD},因此找到合适的电源电压是关键. 图 6 展示了图形应用程序的误码率(bit error rate, *BER*)和信噪比(peak signal-to-noise ratio, *RNSR*)与 V_{DD} 的关系.

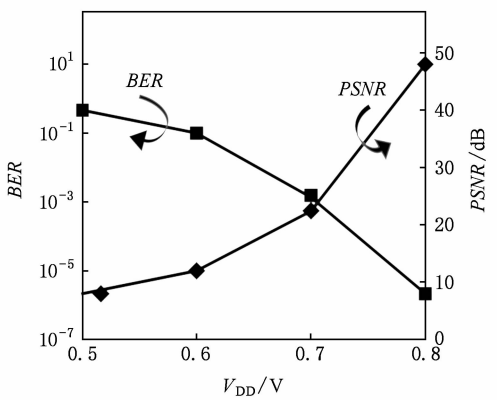


Fig. 6　Measured *BER* and resulting *PSNR* versus V_{DD}[42]

图 6　测量误码率和信噪比与 V_{DD} 的关系[42]

Frustaci 等人提出了具有动态能源-质量管理作用的 SRAM[41],提出了利用 DUAL-V_{DD} 最低对有效位(least significant bit, LSB)的降压技术. 低位精度可以被接受时,禁用对应于给定数量的最低有效位线,以线性地降低质量的能源.

Gilani 等人提出的一种暂存器存储器设计[42]能减少片上存储器的面积,通过权衡必须的最小工作电压($V_{DD\,min}$)和应用输出质量设计存储器. 此外,对于数字信号处理应用,允许将电压调整到 $V_{DD\,min}$ 以下进一步节能,与短字长存储器相比,这种方法的平均误差更低;而对于错误敏感的应用程序,可以通过重新配置存储器的组织结构,以较低的 $V_{DD\,min}$ 实现更高的内存容量.

3.1.4　提高存储可靠性

由于 SRAM 单元容易受到粒子撞击,造成位翻转,导致数据错误,因此保护 SRAM 免受粒子撞击的近似存储技术能提高数据存储的可靠性.

Palframan 等人提出的方法[43]是一种为 GPU 的执行逻辑和寄存器文件提出的精确感知保护方法,他们提出的架构优化了整数计算的错误覆盖率,并结合选择性逻辑强化、目标检查器电路和智能寄存器文件编码,他们的方法通过保护免受粒子撞击,提高了存储访问的可靠性和性能.

3.1.5 基于新型存储器的综合近似方案

目前高速缓存使用的主流器件是 SRAM 单元,它使用晶体管存储数据,但是存在高泄漏功耗、低靠性与低密度等问题. 而新型存储器件,如 STT-MRAM,具有接近零泄漏、高耐用性、高密度的性质,因此使用新兴的新型存储器件代替 SRAM 在缓存层进行近似存储是一种能有效解决多种问题的方案. 新型非易失性存介质将颠覆传统的缓存结构[69],然而目前研究这种方法的人不多,但是这种设计思路具有很好的应用前景,将来可能成为主流的设计手段.

Monazzah 等人提出了一种基于近似计算,使用可靠性-能量旋钮细粒度地对近似 STT-MRAM 缓存实现质量可调配的软/硬件方法:QuARK[17]. 利用程序员对应用程序源代码中的可近似数据的声明,将该信息传递给以硬件方式实现的 2 个主要的 STT-MRAM 可靠性-能量旋钮:读取电流和写入电流. 对于可近似数据的读写操作,2 个旋钮都将设置为满足编译器制定的近似的可接受水平的最低电流. 若输入质量不被接受,可在运行时改变对该数据精度的要求,从而提高可靠性水平.

Ranjan 等人提出 STAxCache[12],一种基于 STT-MRAM 的近似二级缓存架构. 其保留了传统缓存的灵活性,同时允许在程序内存地址空间的不同区域进行不同级别的近似. 该架构扩展了指令集体系结构(instruction set architecture, ISA),并提供接口,使程序员能快速的指定质量要求. 整个架构采用低保留和高保留的高速缓存方式组合形成的异构高速缓存加固,每个高速缓存组合都使用质量可配置的数据阵列,以实现各种精度级别的读写操作. 该架构还提供一张质量表和指令感知的高速缓存控制器. 质量表用于捕获地址空间区域的质量要求,跟踪每个区域跳过刷新的次数. 控制器用于强制执行特定的质量约束.

3.2 针对内存的近似技术

目前针对内存的近似存储技术大多是针对 DRAM 的动态刷新等特性所作的优化,不适用于基于新型非易失性存储器(non-volatile storage, NVM)的内存. 1)NVM 具有无需动态刷新和可持久化保存数据等 DRAM 不具备的特性,使得针对 DRAM 的近似存储技术所解决的问题不复存在. 例如,NVM 不需要动态刷新存储单元,所以无需降低存储单元刷新频率和保留时间的近似存储技术. 2)NVM 存在 DRAM 没有的缺陷,例如读写不对称和写耐久度低. 因此,针对 NVM 特性的近似存储技术还有待探索.

DRAM 单元将数据存储为电容器上的电荷,三星对 DRAM 行业进行研究确定了 3 个主要的威胁:延长恢复时间、频繁刷新和可变保留时间(VRT, variable retention time)[70]. 本节针对这 3 个威胁对 DRAM 的相关近似技术进行了介绍.

1) 减少恢复时间

由于晶体管-电容器之间触点的电阻增加以及较小的晶体管的驱动能力的降低导致需要更长的时间将电量充至目标电压,这种现象称为延长恢复时间.

Zhang 等人提出了一种新的细粒度精确感知的 DRAM 恢复调度技术 DrMP[13],并介绍了 DrMP 的 3 种变体:DrMP-A,DrMP-P 和 DrMP-U. 它们都能大幅减少恢复时间,DrMP-A 通过将重要数据位映射到快速行段来支持近似计算,以减少恢复时间提高性能,并降低了应用级错误. DrMP-P 将内存行配对在一起(一行具有快速回复时间、一行具有较慢的回复时间),以减少精确计算的平均恢复时间. DrMP-U 结合了 DrMP-A 和 DrMP-P,可以更好地交换性能,能耗和计算精度.

2) 降低刷新频率

DRAM 单元必须进行定期的刷新以防止数据丢失. 这些刷新操作干扰了内存访问,随着 DRAM 器件容量的增加,DRAM 刷新的负面影响也随之增加. DRAM 的刷新操作所消耗的能源占 DRAM 总能量消耗的很大一部分,其刷新频率通常由少量的弱单元决定. 为了减少 DRAM 能源消耗,可以通过近似存储技术降低其刷新频率,降低刷新率可能会导致数据的错误率的上升,但是却能以较低的错误率大幅度的降低功耗.

Liu 等人提出近似感知的 DRAM 系统 Flikker[15],如图 7 所示. Flikker 将数据划分为关键和非关键组,运行时系统将数据分配到不同的内存部分,包含关键数据的内存部分以常规的刷新频率刷新,而包含近似(非关键组)数据的行刷新率被降低.

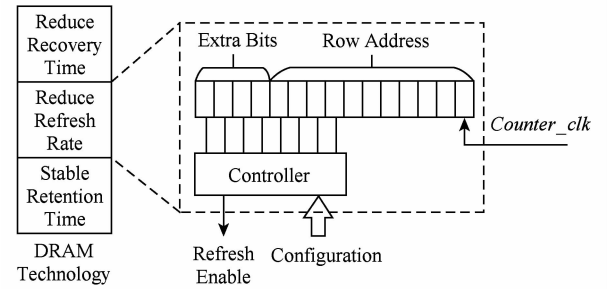


Fig. 7　DRAM approximate technique[15]

图 7　DRAM 近似技术[15]

Liu 等人提出了感知智能 DRAM 刷新机制 RAIDR[18],根据其保留数据所需的刷新率将 DRAM 行分组为多个“容器”,每个容器中的行以不同的速率刷新. RAIDR 还在内存控制器中存储了保留时间,避免了修改 DRAM 设备的需要.

3) 稳定的保留时间

DRAM 单元可以安全地保留数据而不被刷新的时间量称为单元的保留时间. 在当前的系统中,所有 DRAM 单元都以保证电池完整性所需的频率刷新. 可变保留时间对利用保留时间的差异减少不必要刷新带来了难度.

Liu 等人从 5 家主要 DRAM 供应商的 248 种商品 DDR3 DRAM 芯片中收集保留时间信息. 他们发现了 2 种重要现象[71]:1)数据模式依赖,即每个 DRAM 单元的保留时间受存储在其他 DRAM 单元中的数据的显着影响;2)可变保留时间,DRAM 单元的保留时间随时间不可预知地变化. 对未来 DRAM 扩展和保留时间测量机制提供了方向.

Jung 等人提出一个全面的模拟环境[44],用于对近似 DRAM 进行研究. 他们依靠 SystemC 事务级模型(transaction level model, TLM)进行快速准确的模拟,构建基于模块化的 DRAMSys 框架,主要由 4 个关键部分组成:DRAM 和核心模型、DRAM 功耗模型、散热模型、DRAM 保留错误模型.

3.3　针对外存的近似技术

外存具有容量大和访问速度慢的特点,针对外存的近似存储技术更关注如何存储更多的存储数据和减少存储器的磨损. 本节针对上述 2 个问题,阐述了基于外存的相关近似存储技术,如图 8 所示.

3.3.1　近似固态存储技术

固态存储器既可以作为外存存储永久数据,又能够作为主存存储瞬时数据. 固态存储器的缺点表现在 2 个方面:1)磨损不均衡导致寿命不长,厂商标称单级单元(single-level unit, SLC)类型的闪存寿命只有 10 万次,多级单元(multi-level unit, MLC)类型的寿命更短,只有 1 万次[72];2)为了向存储单元存储或检索更多的状态导致的长时间等待延迟.

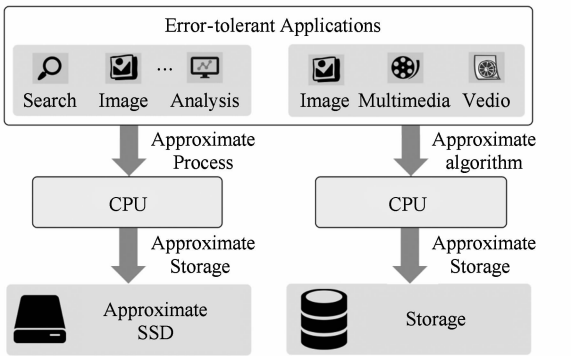


Fig. 8　Approximate storage technique for storage

图 8　针对外存的近似存储技术

Sampson 等人针对上述问题,提出一种近似固态存储技术[5],它能在质量损失忽略不计的情况下,通过将近似数据映射到耗尽硬件纠错资源的块上,以减轻磨损、延长固态存储器的使用寿命;并通过近似配置多级单元,以降低存储或检索的延迟.

最近,Qing 等人提出一种能扩大存储容量的高密度图像近似存储算法[19],主张不同的比特单元对应不同的近似程度,这种算法以图像编码算法 PTC[20]为基础,与近似存储系统[5]协作,以增加存储单元的密度,存储更多的数据.

3.3.2　降低存储开销的近似技术

诸如图像与视频这样的视觉应用程序在存储时耗费大量的存储空间,尤其是监控类视频更要求码率恒定与持续服务[73]. 为了节约资源,可以近似存储数据,以减少空间开销. 压缩存储能有效节约资源,例如著名的 JPEG[56],然而这种商用压缩技术仅用于压缩纹理和颜色数据. 后来有研究人员在 JPEG 的基础上提出了 JPEG XR[21].

Dufaux 等人提出的图像近似存储算法[22]利用 GPGPU 工作负载中浮点数的特性,在截断其最低有效位后对浮点数无损压缩. 使用这种方法要先研究满足需要的微体系结构,其必须支持 GPU 和片外存储器之间传输数据的无损压缩技术. 这种方法对图像整体计算精度的影响很小.

3.4　近似存储技术小结

从存储的层次结构,缓存、主存、外存 3 个方面介绍近似存储技术. 针对高速缓存,主要介绍了提高缓存访问效率、减小缓存空间开销、降低缓存能耗、提高缓存可靠性的近似存储技术;针对主存,主要对

如何克服 DRAM 延长恢复时间、频繁刷新和可变保留时间的 3 个缺点分别介绍了相应的近似存储技术;针对外存,主要介绍了近似固态存储技术与降低存储开销的算法.

近似存储作为一项新兴的技术,是近年来存储相关工作的研究重点,本文介绍了部分近年来具有代表性的近似存储技术.一些其他与近似存储相关的技术,应用于存储的多个层次,如表 3 中 Mixed Layer 所示,Chisel[14]优化运行在近似硬件平台的内核以自动选择可能存储在近似内存中的近似指令和数据,Uncertain〈T〉[24]为近似数据在近似硬件平台上设计一种近似编程语言以映射到近似器件运行和存储,文献[23]的技术基于 SIMD 体系结构通过重用程序运行中的局部结果以减小错误恢复的成本,文献[10]通过重新构建新的指令集使程序能近似计算与存储.

4 总结与展望

近似存储技术作为解决存储资源紧张的必要手段,是近年来存储相关工作的研究重点.本文简要介绍了近似存储以及近似存储的一般过程,并依据存储架构的主要层次,分别从基于高速缓存、内存、外存 3 个方面,详细介绍近似存储技术的研究现状.

近似存储技术对应用程序中可近似数据采取不精确的存储方法,提高存储空间的利用率、加速存储访问效率、节约能源消耗、减少器件磨损,从而提升存储器性能,以解决目前存储领域面临的存储资源紧张、能源消耗高、读取速度低等问题.

但是,目前的近似存储研究工作仍然存在不足与待改进之处体现在 4 个方面:

1) 目前的近似存储技术为了区分程序的近似与精确部分,需要对程序进行预处理.然而,现有预处理方法复杂高且不具有一般性.因此,简便高效、通用的近似区域查找方法是未来近似存储领域的一项重要研究内容.

2) 现有提高存储访问效率的近似存储技术集中在缓存层面,用于缩小内存与处理器之间的访问速度差距.然而,如何利用近似存储技术缩小内外存之间的数据访问速度差异是亟待解决的问题.

3) 近年来,新型非易失性存储器件飞速发展,展现出 DRAM 级数据访问速度、数据掉电不丢失和存储密度大等优点,是可以用作高速缓存、内存或外存的下一代存储设备.然而,基于新型非易失性存储器的近似技术研究进展缓慢.面向各个存储层次,研究优化新型非易失性存储器中的数据访问性能、存储开销和存储磨损度的近似存储技术是一个极具前景的研究方向.

4) 由于每个用户对输出质量损失的接受程度不同,在近似存储技术权衡数据精确度与输出质量时,如何找到一个令大部分用户满意的质量下降阈值是一个难题.为此,保证近似存储可靠性的质量下降阈值设定方法是未来必须研究的近似存储关键技术.

参 考 文 献

[1] Dennard R H, Rideout V L, Bassous E, et al. Design of ion-implanted MOSFET's with very small physical dimensions [J]. IEEE Journal Solid State Circuits, 1974, 9(5): 256-268

[2] Gantz J, Reinsel D. Extracting value from chaos [OL]. [2018-04-15]. https://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf

[3] Ceze L, Sampson A. Approximate computing: Unlocking efficiency with hardware-software co-design [J]. ACM, 2017, 20(3): 12-16

[4] Chippa V K, Chakradhar S T, Roy K, et al. Analysis and characterization of inherent application resilience for approximate computing [C] //Proc of the 22nd Design Automation Conf. Piscataway, NJ: IEEE, 2013: 113: 1-113:9

[5] Sampson A, Nelson J, Strauss K, et al. Approximate storage in solid-state memories [J]. ACM Trans on Computer Systems, 2014, 32(3): 9:1-9:23

[6] Li X, Yeung D. Application-level correctness and its impact on fault tolerance [C] //Proc of the 18th High Performance Computer Architecture. Piscataway, NJ: IEEE, 2007: 181-192

[7] Yazdanbakhsh A, Park J, Sharma H, et al. Neural acceleration for GPU throughput processors [C] //Proc of the 48th IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2015: 482-493

[8] Samadi M, Jamshidi D A, Lee J, et al. Paraprox: Pattern-based approximation for data parallel applications [C] //Proc of the 19th Int Conf on Architectural Support for Programming Languages & Operating Systems. New York: ACM, 2014: 35-50

[9] Sidiroglou-Douskos S, Misailovic S, Hoffmann H, et al. Managing performance vs. accuracy trade-offs with loop perforation [C] //Proc of the 19th ACM SIGSOFT Symp and the 13th European Conf on Foundations of Software Engineering. New York: ACM, 2011: 124-134

[10] Chen Y K, Chhugani J, Dubey P, et al. Convergence of recognition, mining, and synthesis workloads and its implications [J]. Proceedings of the IEEE, 2008, 96(5): 790-807

[11] Esmaeilzadeh H, Sampson A, Ceze L. Doug burger: Architecture support for disciplined approximate programming [C] //Proc of the 17th Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2012: 301-312

[12] Yazdanbakhsh A, Pekhimenko G, Thwaites B, et al. RFVP: Rollback-free value prediction with safe-to-approximate loads [J]. ACM Transactions on Architecture & Code Optimization, 2015, 12(4): 62:1-62:13

[13] Ranjan A, Venkataramani S, Pajouhi Z, et al. STAxCache: An approximate, energy efficient STT-MRAM cache [C] //Proc of DATE's 17. New York: ACM, 2017: 356-361

[14] Zhang XianWei, Zhang Youtao, Childers B R, et al. DrMP: Mixed precision-aware DRAM for high performance approximate and precise computing [C] //Proc of the 22nd Int Conf on Parallel Architectures and Compilation Techniques. Piscataway. Piscataway, NJ: IEEE, 2017: 53-63

[15] Misailovic S, Carbin M, Achour S, et al. Reliability and accuracy-aware optimization of approximate computational kernels [C] //Proc of the 29th Conf on Object-Oriented Programming Systems, Languages, and Applications. New York: ACM, 2014: 309-328

[16] Liu S, Pattabiraman K, Moscibroda T, et al. Flikker: Saving DRAM refresh-power through critical data partitionin [J]. Architectural Support for Programming Languages and Operating Systems, 2011, 39(1): 213-224

[17] Miguel J S, Badr M, Jerger N E. Load value approximation [C] //Proc of MICRO's 14. Piscataway, NJ: IEEE, 2014: 127-139

[18] Monazzah A M H, Shoushtari M, Miremadi S G, et al. QuARK: Quality-configurable approximate STT-MRAM cache by fine-grained tuning of reliability-energy knobs [C] //Proc of the 26th Int Conf on Low Power Electronics and Design. Piscataway, NJ: IEEE, 2017: 1-6

[19] Liu J, Jaiyen B, Veras R, et al. RAIDR: Retention-aware intelligent DRAM refresh [C] //Proc of the 40th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2012: 1-12

[20] Qing Guo, Strauss K, Ceze L, et al. High-density image storage using approximate memory cells [C] //Proc of the 21st Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 413-426

[21] Malvar H S. Fast progressive image coding without wavelets [C] //Proc of DCC'00. Piscataway, NJ: IEEE, 2000: 243-252

[22] Wallace G K. The JPEG still picture compression standard [J]. Communications of the ACM, 1991, 34(4): 30-44

[23] Dufaux F, Sullivan G J, Ebrahimi T. The JPEG XR image coding standard [J]. IEEE Signal Processing Magazine, 2009, 26(6): 195-199

[24] Rahimi A, Benini J, Rajesh K, et al. Concurrent instruction reuse to correct timing errors in SIMD architecture [J]. IEEE Trans on Circuits and Systems II: Analog & Digital Signal Processing, 2013, 60(12): 847-851

[25] Bornholt J, Mytkowicz T, McKinley K S. Uncertain〈T〉: A first-order type for uncertain data [C] //Proc of the 19th Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2014: 51-66

[26] Xu Q, Mytkowicz T, Kim N S. Approximate computing: A survey [J]. IEEE Design & Test, 2016, 33(1): 8-22

[27] Han J, Orshansky M. Approximate computing: An emerging paradigm for energy-efficient design [C] //Proc of the 18th European Test Symp. Piscataway, NJ: IEEE, 2013: 1-6

[28] Dubrova E. Fault tolerant design: An introduction [OL]. [2018-04-15]. http://www.malekinezhad.ir/KTH.pdf

[29] Carbin M, Misailovic S, Rinard M C. Verifying quantitative reliability for programs that execute on unreliable hardware [J]. ACM SIGPLAN Notices, 2013, 48(10): 33-52

[30] Misailovic S, Carbin M, Achour S, et al. Reliability-and accuracy-aware optimization of approximate computational kernels [C] //Proc of the 29th Conf on Object-Oriented Programming Systems, Languages, and Applications. New York: ACM, 2014: 309-328

[31] Hari S K S, Adve S V, Naeimi H, et al. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults [J]. Computer Architecture News, 2012, 40(1): 123-134

[32] Venkatagiri R, Mahmoud A, Hari S K S, et al. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency [C] //Proc of the 49th IEEE/ACM International Symp on Microarchitecture. Piscataway, NJ: IEEE, 2016: 1-14

[33] Feng S, Gupta S, Ansari A, et al. Shoestring: Probabilistic soft error reliability on the cheap [C] //Proc of the 15th Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2010: 385-396

[34] Cong J, Gururaj K. Assuring application-level correctness against soft errors [C] //Proc of the 9th IEEE/ACM Int Conf on Computer-Aided Design. Piscataway, NJ: IEEE, 2011: 150-157

[35] Sampson A, Dietl W, Fortuna E, et al. EnerJ: Approximate data types for safe and general low-power computation [J]. ACM SIGPLAN Notices, 2011, 46(6): 164-174

[36] Eickemeyer R J, Vassiliadis S. A load-instruction unit for pipelined processors [J]. IBM Journal of Research & Development, 1993, 37(4): 547-564

[37] Sutherland M, Miguel J S, Jerger N E. Texture Cache Approximation on GPUs [OL]. [2018-04-15]. http://www.eecg.utoronto.ca/~enright/TexCacheApprox.pdf

[38] Miguel J S, Albericio J, Moshovos A, et al. Doppelgänger: A cache for approximate computing [C] //Proc of the 50th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2017: 50-61

[39] Sjalander M, Nilsson N S, Kaxiras S. A tunable cache for approximate computing [C] //Proc of the 10th IEEE/ACM Int Symp on Nanoscale Architectures. Piscataway, NJ: IEEE, 2014: 88-89

[40] Frustaci F, Khayatzadeh M, Blaauw D, et al. SRAM for error-free and error-tolerant applications with dynamic energy-quality management in 28nm CMOS [C] //Proc of ISSCC'14. Piscataway, NJ: IEEE, 2014, 244-245

[41] Frustaci F, Blaauw D, Sylvester D, et al. Approximate SRAMs with dynamic energy-quality management [J]. IEEE Trans on Very Large Scale Integration Systems, 2016, 24 (6): 2128-2141

[42] Gilani S Z, Kim N S, Schulte M. Scratchpad memory optimizations for digital signal processing applications [C] // Proc of the 14th Design, Automation & Test in Europe Conf. Piscataway, NJ: IEEE, 2011: 1-6

[43] Palframan D J, Kim N S, Lipasti M H. Precision-aware soft error protection for GPUs [C] //Proc of the 20th High Performance Computer Architecture. Piscataway, NJ: IEEE, 2014: 49-59

[44] Jung M, Mathew D M, Weis C, et al. Efficient reliability management in SoCs-an approximate DRAM perspective [C] //Proc of the 21st Asia and South Pacific Design Automation Conf. Piscataway, NJ: IEEE, 2016: 390-394

[45] Sathisha V, Schulte M J, Kim N S. Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads [C] //Proc of the 27th Parallel Architectures and Compilation Techniques. New York: ACM, 2012: 325-334

[46] Samadi M, Lee J, Jamshidi D A, et al. SAGE: Self-tuning approximation for graphics engines [C] //Proc of the 50th IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2017: 13-24

[47] Chippa V, Raghunathan A, Roy K, et al. Dynamic effort scaling: Managing the quality-efficiency tradeoff [C] //Proc of the 52nd Design Automation Conf. New York: ACM, 2011: 603-608

[48] Moreau T, Wyse M, Nelson J, et al. SNNAP: Approximate computing on programmable SoCs via neural acceleration [C] //Proc of the 21st High Performance Computer Architecture. New York: ACM, 2015: 603-614

[49] Grigorian B, Reinman G. Accelerating divergent applications on SIMD architectures using neural networks [J]. ACM Trans on Architecture and Code Optimization, 2015, 12(1): 2:1-2:23

[50] Grigorian B, Farahpour N, Reinman G. BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing [C] //Proc of the 21st High Performance Computer Architecture. Piscataway, NJ: IEEE, 2015: 615-626

[51] Rubio-González C, Nguyen C, Hong D N, et al. Precimonious: Tuning Assistant for Floating-Point Precision [M]. New York: ACM, 2013: 27:1-27:12

[52] ohansson L, Karppinen A, Wanner L. The fusion of meteorological-and air quality information for orchestrated services using environmental profiling [C] //Proc of the 16th Int Conf on Information Fusion. Piscataway, NJ: IEEE, 2013: 1638-1644

[53] Venkataramani S, Chippa V K, Chakradhar S T, et al. Quality programmable vector processors for approximate computing [C] //Proc of the 50th IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2017: 1-12

[54] Schkufza E, Sharma R, Aiken A. Stochastic optimization of floating-point programs with tunable precision [C] //Proc of the 35th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2014: 53-64

[55] Liang, Jinghang, Lombardi F. New metrics for the reliability of approximate and probabilistic adders [J]. IEEE Trans on Computers, 2013, 62(9): 1760-1771

[56] Liu Cong, Jie Han, Lombardi F. A low-power, high-performance approximate multiplier with configurable partial error recovery [C] //Proc of DATE'14. Piscataway, NJ: IEEE, 2014: Article No. 95

[57] Du Zidong, Palem K, Lingamneni A, et al. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators [C] //Proc of the 55th Design Automation Conf. Piscataway, NJ: IEEE, 2014: 201-206

[58] Shin D, Gupta S K. Approximate logic synthesis for error tolerant applications [C] //Proc of DATE'10. Piscataway, NJ: IEEE, 2010: 957-960

[59] Mahajan D, Yazdanbakhsh A, Park J, et al. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration [J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 66-77

[60] Xin S, Lenharth A, Fussell D S, et al. Proactive control of approximate programs [C] //Proc of the 21st Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 607-621

[61] Misailovic S. Quality of service profiling [C] //Proc of the 32nd ACM/IEEE Int Conf on Software Engineering. New York: ACM, 2010: 25-34

[62] Khudia D S, Zamirai B, Samadi M, et al. Rumba: An online quality management system for approximate computing [J]. ACM SIGARCH Computer Architecture News, 2015, 43 (30): 554-566

[63] Hosseini F S, Fotouhi P, Yang C, et al. Leveraging compiler optimizations to reduce runtime fault recovery overhead [C] //Proc of the 54th Design Automation Conf. New York: ACM, 2017: Article No. 20

[64] Shoushtari M, Banaiyanmofrad A, Dutt N. Exploiting partially-forgetful memories for approximate computing [J]. IEEE Embedded Systems Letters, 2015, 7(1): 19-22

[65] Mittal S. A survey of techniques for approximate computing [J]. Computing Surveys, 2016, 48(4): 62:1-62:33

[66] Park J, Amaro E, Mahajan D, et al. AxGames: Towards crowdsourcing quality target determination in approximate computing [C] //Proc of the 21st Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 623-636

[67] Kadjo D, Kim H, Gratz P, et al. Power gating with block migration in chip-multiprocessor last-level caches [C] //Proc of DATE'13. Piscataway, NJ: IEEE, 2013: 93-99

[68] Kumar A. SRAM leakage-power optimization framework: A system level approach [OL]. [2018-04-15]. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-182.pdf

[69] Xu Yuanchao, Yan Junfeng, Wan Hu, et al. Review of new non-volatile storage security and privacy issues [J]. Journal of Computer Research and Development, 2016, 53(9): 1930-1942 (in Chinese)
(徐远超, 闫俊峰, 万虎, 等. 新型非易失存储的安全与隐私问题研究综述[J]. 计算机研究与发展, 2016, 53(9): 1930-1942)

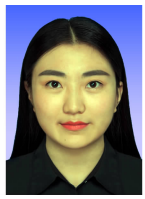
[70] U. Kang, et al. Co-architecting controllers and DRAM to enhance DRAM process scaling [OL]. [2018-04-15]. http://www.cs.utah.edu/thememoryforum/kang.pdf

[71] Liu J, Jaiyen B, Kim Y, et al. Implications for retention time profiling mechanisms [J]. ACM SIGARCH Computer Architecture News, 2013, 41(3): 60-71

[72] Chen Zhiguang, Xiao Yu, Liu Fang, et al. A high-performance and reliable storage system for disk SSD backup [J]. Journal of Computer Research and Development, 2013, 50(1): 80-89 (in Chinese)
(陈志广, 肖侬, 刘芳, 等. 一种用磁盘备份SSD的高性能可靠存储系统[J]. 计算机研究与发展, 2013, 50(1): 80-89)

[73] Cao Shunde, Hua Yu, Feng Dan, et al. High performance distributed storage system for massive high definition video data [J]. Journal of Software, 2017, 28(8): 1999-2009 (in Chinese)
(操顺德, 华宇, 冯丹, 等. 面向海量高清视频数据的高性能分布式存储系统[J]. 软件学报, 2017, 28(8): 1999-2009)

Wu Yu, born in 1995. Master candidate. Her main research interests include approximate computing and storage.



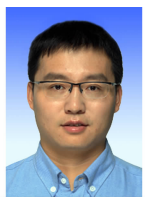
Yang Juan, born in 1995. Master candidate. Her main research interests include approximate computing and storage.

Liu Renping, born in 1991. PhD candidate. His main research interests include approximate computing and hardware accelerator.

Ren Jinting, born in 1994. PhD candidate His main research interests include approximate computing and optimizing compiler.

Chen Xianzhang, born in 1989. PhD. His main research interests include new-generation memory systems, file systems, embedded systems and software, and cloud computing.

Shi Liang, born in 1987. PhD, associate professor. His main research interests include flash memory, embedded systems, and emerging non-volatile memory technology.



Liu Duo, born in 1980. PhD, professor. His main research interests include intelligent Internet of things systems and edge computing, big data processing and advanced storage, embedded smart terminals, energy-efficient computer systems, machine learning and system optimization.