Democratizing Error-Efficient Computing

Research Statement by Radha Venkatagiri

I am a **computer architect** and I build efficient computing systems that conserve resources (e.g., time, energy, cost) by allowing the system to make controlled errors. I work on *Error-Efficient Computing* with emphases on *Low-Cost Hardware Resiliency* and *Approximate Computing*. My work has multi-disciplinary implications in core CS/CE (programming languages, software engineering, hardware design, systems) & emerging application areas (AI, robotics, edge computing).

1 Motivation and Research Vision

We face an urgent need to improve compute efficiency due to the following trends: (1) diminishing benefits (in performance and power) of CMOS technology scaling, (2) increased computational demand due to the explosive growth of data and (3) increasing susceptibility of hardware to errors – from faulty devices to malicious attacks. The intersection of these trends makes it **extremely expensive to guarantee perfect functionality across billions of devices**. Fortunately, the workloads that are driving this demand for computing (learning, recognition, mining and search, among others) also provide opportunities since their primary goal is not a precise, numerically correct answer. Rather, their 'correctness' lies in producing results that are good enough, or of sufficient quality, to produce an acceptable user experience.

The paradigm of *Error-Efficient Computing* exploits this forgiving nature of applications by *allowing the computing system to make controlled errors*. Such systems can be considered as being **error-efficient: they only prevent as many errors as they need to** for an acceptable user experience. The definition of what constitutes an error varies across system components and which errors are acceptable depends on the application and/or user expectations. Allowing the system to make errors (when behaving correctly would be too expensive) can conserve resources. The resources conserved could be time, energy, bandwidth or more abstract quantities such as reliable operation, manufacturing costs, etc. Some examples of error-efficient techniques include (1) *approximate computing* which deliberately introduces errors in computation for improved performance or energy, and (2) *ultra-low cost hardware resiliency* which allows some unintentional hardware errors to escape as user-tolerable output corruptions (rather than incurring high overheads to prevent all errors).

The overarching vision of my research is to (1) enable both novice programmers and expert designers to adopt error-efficiency as a first-class design principle, while operating within (2) a computing stack that allows a principled, unified and yet, customizable way of computing efficiently with heterogeneous sources of errors.

2 Summary Of Contributions and Impact

Error-efficient computing can revolutionize the way we design hardware and software to exploit significant new opportunities of compute efficiency. Despite its promise, the adoption of error-efficiency has been thwarted by the undue burden placed on programmers to have a deep understanding of how errors (perturbations in program execution) affect program output. This in turn limits these techniques to the few application domains for which such expertise exists. My dissertation addresses these limitations by **developing methodologies that enable systematic, principled and scalable application of error-efficiency without the need for programmer expertise**. My key contributions are:

(1) Developed **automated error analysis** techniques and tools to determine the impact of billions of errors on program output with high speed [8] and accuracy. These tools [9, 11, 12] are the first-of-their-kind to quantify output quality for all errors (for a given hardware error model) in program instruction and data (Section 3.1). The application error profiles they generate are used for customized error-efficiency (Section 3.2). Two tools have been publicly released [2, 4].

(2) Built a framework called *Minotaur* [6] that **systematically integrates (hardware) error analyses into the software development workflow**. The novel insight is that testing for software bugs is similar to analyzing (hardware) errors. Minotaur uses software testing techniques to improve the speed (up to 55x) and **scalability** of error analyses (Section 3.3).

(3) Demonstrated the **effectiveness and resiliency of error-efficiency** techniques in edge computing domains. This was the first study (in collaboration with IBM research) to show the **interaction of software approximations and system resiliency** in a state-of-the-art vision analytics workflow on-board Unmanned Aerial Vehicles (UAVs) [13] (Section 3.4).

Overall, my dissertation **broadens the reach of error-efficient computing to novice programmers and new domains**. Thus far, my work has had the following **impact**:

- Minotaur [6] laid the foundation for a new software engineering discipline for hardware errors.
- My work [13] on UAV vision analytics was nominated for the 2019 Pat Goldberg Memorial Best Paper Award; this is an annual award given to the best CS papers originating from IBM Research.
- The IBM collaboration [13] enabled key accomplishments in IBM's DARPA PERFECT [1] project, and IBM is using concepts from it to improve the efficiency of Autonomous Vehicle workloads in another ongoing DARPA project.
- My work on automated error analysis was selected twice by industry experts for TECHCON [7, 10], which showcases the best of SRC [3] sponsored research. Intel and ARM have expressed interest in using the tools I've developed.

3 Brief Overview of Dissertation Work

3.1 Tool-Suite for Automated Application-Level Error Analysis

For any error-efficiency solution, we first need a fundamental understanding of how errors in a program affect its resultant output quality. Techniques today rely on program/domain experts for this knowledge which is severely limiting. To mitigate this limitation, I developed a suite of automated application-level error analysis tools [9, 11, 12] that can *automatically extract the error characteristics of applications.* In this work, an error is defined as a perturbation in program state (instructions or data) caused by an underlying fault in hardware. Using a hybrid technique [5, 8] of program analysis and some error injections, these tools comprehensively analyze billions of possible errors that can impact a program's execution with high accuracy and speed. They impose minimum burden on the programmer and only require the user to provide an unmodified program, a (domain-specific) quality metric and, optionally, a quality threshold. The error analyses techniques developed in this line of work are general and can be used to analyze different computations and error models. To the best of my knowledge, these are the **first application-level error analysis tools that solve the significant research challenge of meeting all five requirements of automation, accuracy, comprehensiveness, speed and generality.** The output of automated error analyses are *comprehensive application error profiles* that list the errors that can affect the program's execution along with the corresponding output quality expected for each of the errors. The application error profiles can then be used by programmers or systems to understand the application's error characteristics.

Automated Analysis of Errors in Program Instructions (Compute): **Approxilyzer** [11] was the first automated error analysis tool to quantify the impact, of all errors (for a given error model) in program instructions, on the program's output quality with high accuracy and speed. The error model used in Approxilyzer is single-bit transient errors in register operands of dynamic instructions. Approxilyzer's output is the application's comprehensive *instruction error profile.* In order to enable researchers to easily use and extend Approxilyzer concepts, I built a fully open-source error analysis toolkit called **gem5-Approxilyzer** [9]. gem5-Approxilyzer uses the open-source gem5 simulator and is designed to enable researchers to adapt and extend it to different instruction set architectures (ISAs).

Automated Analysis of Errors in Program Data (Storage): Errors in instructions and data propagate differently through the program and, hence, require different analysis techniques. I built **Winnow** [12], to automatically analyze and quantify the impact of errors in program data on output quality. Winnow accurately and comprehensively analyzes all errors (for a given error model) in program data accessed at different points in the program's execution. The error model used in Winnow is multi-bit (1-bit, 2-bit, 4-bit and 8-bit) transient errors in system memory. The output of Winnow is the application's comprehensive *data error profile* which lists the output quality produced in the presence of virtually all data errors that can impact a program execution.

3.2 Automated Error Analysis to Customized Error-Efficiency

An application's error profile can be used to understand its error characteristics, which can inform customized error-efficiency. For example, I demonstrated how an application's instruction error profile is used for error-efficiency solutions targeted towards **ultra-low cost resiliency to hardware errors** [11]. I showed that large resiliency overhead reductions (up to 55%) are possible if the user is willing to tolerate a very small loss in output accuracy (1%) while still providing high (99%) resiliency coverage [11]. In another example, I used the application's error profile to identify promising subsets of instructions and/or data for **approximate computing** across different quality thresholds and approximation targets. I demonstrated the automatic identification and mapping of non-critical (approximable) data to be stored in approximate DRAM with low refresh rates (which saves energy but incurs modest errors), without programmer intervention [12]; prior techniques required the programmer to identify non-critical data with annotations. The error profiles of applications can also be used to gain *insights* that can motivate further research. For example, my work showed that error profiles for the same application, compiled to different ISAs (SPARC vs. x86), can vary widely [9]; motivating not only the need for customized error-efficiency for different architectures, but also the benefits of error analysis at the machine-code level (as opposed to at the intermediate representation or source-code level).

3.3 Leveraging Software Testing Techniques for Efficient and Scalable Error Analyses

I built a framework called **Minotaur** [6] that significantly improves the speed and scalability (across multiple inputs and workloads) of error analysis techniques. Minotaur used the insight that *analyzing an application for (hardware) errors has many conceptual similarities to analyzing it for software bugs*; therefore, adapting techniques from the rich software testing literature can lead to principled and significant improvements in error analyses. Minotaur was the first work to suggest **leveraging software testing methodologies for comprehensive error analysis**; thereby, laying the foundation for a principled integration of hardware error analysis into the software development work-flow.

Minotaur identified and adapted four concepts from software testing to: (a) introduce the concept of input quality criteria for error analysis and suggest a simple but effective criterion (statement coverage at the object-code level); (b) develop a methodology to create high quality (fast) minimized inputs from (slow) standard benchmark inputs; (c)

prioritize the analysis of specific program locations for a given error analysis objective and terminate analysis early when the objective is met; and (d) show scalable error analysis over multiple inputs by progressively prioritizing analysis over fast (but potentially inaccurate) inputs first. Minotaur improved the speed of comprehensive error analysis by 4x, on average, over state-of-the-art tools like Approxilyzer. These gains went up to 39x (or 55x) when the error analysis was targeted to specific techniques like hardware resiliency (or approximate computing).

3.4 Error-Efficiency for Emerging Domains

Error-efficient computing can enable emerging domains, such as edge-computing (UAVs, connected cars, industrial robotics, etc.), to meet strict performance and energy requirements. However, the strict reliability requirements of some mission-critical (high impact of failure/disruption) applications in this domain have traditionally not allowed inexact computations. For example, edge computing platforms on UAVs (Unmanned Aerial Vehicles), that are engaged in rescue and recovery operations, must not only meet strict real-time performance, energy and bandwidth requirements, they must also be resilient while operating in harsh environments subject to sharp variations in temperature, altitude and weather conditions, and tolerate glitches in input and output. Hence, any error-efficiency solutions applied to such systems (to improve, say, performance and energy) must not degrade the system resiliency.

In a collaborative study with *IBM Research* [13], I demonstrated the **resiliency and effectiveness of error-efficiency techniques for edge-computing applications** used in UAVs (the concepts are relevant to other applications as well). I studied a state-of-the-art Video Summarization (VS) application (developed at IBM Research) that constitutes key end-to-end video and image analytics performed aboard UAVs. *In this first-of-a-kind work that studies the effect of approximations on system performance, energy and resiliency*, I showed that software approximations yield significant energy and performance (up to 68%) benefits for the end-to-end VS work-flow without degrading the overall resiliency of the system.

4 Future Work

While my dissertation work shows the promise of automated error analysis techniques that can potentially be integrated within a software development framework, there is still a long way to go towards the research vision of automated full-stack methodologies that can provide optimal system-wide error-efficiency solutions for emerging applications. In this section, I will describe some of the future work I envisage towards that vision.

4.1 [Short-term] Error-Efficiency at Scale for Emerging Cognitive Applications

My research thus far has shown fast, comprehensive and automated error analyses for single-threaded CPU workloads from different domains (image processing, data mining, scientific computing, etc.); precisely analyzing these workloads in their entirety was a challenge before my work. *Next, I would like to perform scalable and efficient automated error analysis for large multi-threaded applications*. I am especially interested in analyzing cognitive applications – such as **image processing, autonomous driving, virtual/augmented reality and robotics** – that aim to extract context/insight from vast quantities of data. These applications heavily use sensory signals (image, audio, etc.) that can inherently tolerate inaccuracies in data and/or computation and hence, are natural candidates for error-efficient computing. Error-efficiency techniques applied to these domains today are largely empirical and focused on single/few techniques which is sub-optimal and may leave additional sources of efficiency untapped. An optimal solution that considers different combinations of error-efficiency techniques (customized to system/user specifications) is an intractable optimization problem today; automatic, scalable and comprehensive error analysis of these applications can provide insights and be the first step towards a systematic methodology to extract maximum resource efficiency for emerging domains.

4.2 [Short-Term] Error-Efficiency for Performance, Energy and Beyond

The output of automated error analysis (for a given application) is a *comprehensive application error profile* that lists each error (for the error model under consideration) that can perturb the program's execution along with its corresponding impact on output quality. My dissertation work shows a few examples (targeted to approximate computing and hardware resiliency) of how these comprehensive error profiles can enable customized error-efficiency, but I believe that this is just a beginning. I intend to explore **other use-cases for application error profiles**, such as, **mixed dynamic precision** or criticality testing for **emerging storage solutions** like non-volatile memories (NVMs).

Error-efficiency techniques have largely been focused on trading functional correctness, in the form of output inaccuracies, to improve power/energy or performance. But the promise of error-efficiency and automated error analysis can potentially be extended to other domains that deal with anomalies in computing behavior; for instance, efficient protection from **fault attacks**, analyzing interference behaviour of **secure/private data** being computed on erroneous substrates, using application error profiles to assist with debugging during **hardware validation**, etc. I plan to collaborate with experts in these fields to explore these possibilities.

4.3 [Longer-Term] Error-Efficiency Across the Computing Stack

An ideal error-efficient solution should take any application, along with user inputs (like quality of service guarantees or optional domain-specific information) and map it to optimal combinations of hardware/software error-efficiency

techniques. To realize this, we need *integrated, full-stack solutions* that treat **error-efficiency as a first class metric at each layer of the compute stack**. My work thus far has been at the application layer of the stack and I have shown how error-efficiency can be integrated into the software development workflow. Other researchers have made progress at different layers of the stack – hardware/software error-efficiency techniques, schedulers, optimizers, compilers, programming languages, etc. However, we are missing the right abstractions to enable integration across the different layers.

We are also seeing a trend towards specialization and heterogeneity at different scales. This has been a boon for error-efficiency techniques that have leveraged hardware/software co-design to extract maximum compute efficiency for specialized domains. However, it has made the task of developing **general full-stack error-efficiency methodologies**, that still retain the **flexibility to specialize/customize at different layers**, very challenging. Going forward, I am interested in exploring the following fundamental questions that need to be answered for error-efficiency solutions across the compute stack: What are the right metrics to quantify an application's error characteristics? What is the right abstraction for **programming languages** to express error-specifications in inputs, return values, instructions and data? How can **compilers** encode/translate dynamic error-efficiency opportunities of applications? Can the heterogeneous error characteristics of **hardware** be abstracted as standardized knobs that can be tuned at the direction of **software**? Answers to these questions will require a **cross-disciplinary** collaborative effort that I look forward to initiating.

4.4 [Longer-Term] System Wide Error-Efficiency over Heterogeneous Sources of Errors

As mentioned in the beginning of this statement, part of my research vision is to develop a *principled and unified methodology for analyzing different sources of errors*. This is an important but extremely challenging problem since modern workloads can encounter a wide variety of **heterogeneous components** that have **varied sources of errors**, each with a *different error model* and no established methodology for translation between them. It is unclear how different error-efficiency techniques across a range of devices will interact, combine and complement or negate each other to provide end-to-end application and/or system benefits. **System wide solutions** that can holistically optimize for error effects arising from heterogeneous compute, data and networks will require a *systematic methodology for incremental and compositional error-efficiency analyses* over a range of workloads, devices and error models. Conceptual intuition gained by my dissertation work makes me optimistic that, albeit hard, this is a problem that can be solved. For example, my work provides an intuition for how the problem of finding the errors in compute that lead to output corruptions can be tackled by systematically performing incremental analysis over increasingly complex error models. Formalizing these concepts will be crucial for establishing a systematic methodology for system-wide error-efficiency.

Summary of Intended Future Collaborations: As detailed above, I intend to explore novel solutions through collaborations with researchers from both *academia and industry* working in areas such as Computer Architecture and Systems, Hardware/Software Reliability, Security/Privacy, Approximate Computing, Programming Languages, Compilers, Software Testing, Hardware Validation, Edge Computing, Artificial Intelligence and Robotics, among others.

Funding: My dissertation work was funded through NSF, DARPA and SRC (JUMP) grants. I plan to rely on government (NSF, DARPA) and industry sources (Intel, Nvidia, Google, Facebook, etc.) for funding as an independent PI.

References

[1] Darpa's power efficiency revolution for embedded computing technologies (PERFECT) program.

[2] gem5-approxilyzer. `https://github.com/rsimgroup/gem5-approxilyzer`.

[3] Semiconductor research corporation. `https://www.src.org`.

[4] Approxilyzer. `https://cs.illinois.edu/approxilyzer`.

[5] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. In *Architectural Support for Programming Languages & Operating Systems (ASPLOS)*, 2012.

[6] A. Mahmoud, R. Venkatagiri, K. Ahmed, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve. Minotaur: Adapting software testing techniques for hardware errors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[7] A. Mahmoud, R. Venkatagiri, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve. Harnessing software testing techniques for hardware resiliency analysis. In *SRC TECHCON*, 2018.

[8] S. K. Sastry Hari, R. Venkatagiri, S. V. Adve, and H. Naeimi. Ganges: Gang error simulation for hardware resiliency evaluation. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, 2014.

[9] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. Adve. gem5-Approxilyzer: An open-source tool for application-level soft error analysis. In *International Conference on Dependable Systems and Networks (DSN)*, 2019.

[10] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve. gem5-Approxilyzer: Towards general-purpose, comprehensive and automated soft error analysis. In *SRC TECHCON*, 2019.

[11] R. Venkatagiri, A. Mahmoud, S. K. S. Hari, and S. V. Adve. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. *International Symposium on Microarchitecture (MICRO)*, 2016.

[12] R. Venkatagiri, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve. Winnow: Automated application-level error analysis for data. In *Under submission*.

[13] R. Venkatagiri, K. Swaminathan, C. Lin, L. Wang, A. Buyuktosunoglu, P. Bose, and S. Adve. Impact of software approximations on the resiliency of a video summarization system. In *International Conference on Dependable Systems and Networks (DSN)*, 2018.