Research Statement
Ashish Venkat

My research explores novel computer architecture and compiler techniques to design machines that are faster, energy-efficient, and more secure. In recent years, the computing landscape has witnessed a shift towards hardware heterogeneity in response to the rapidly evolving software, changing market risks, and technological limitations. My goal is to empower the hardware/software interface with diverse capabilities to enable the seamless adoption of heterogeneous hardware, without breaking the traditional models of programming.

My dissertation research enables programs to cross a heretofore forbidden boundary – the Instruction Set Architecture (ISA). Although the advantages of single-ISA heterogeneity have been well established, there was no evidence, or even speculation, that extending that heterogeneity to the ISAs offered by the processors could be profitable. My research is the first to challenge the assumption that that boundary is necessary. By pulling down this boundary wall, my research has unlocked several previously unexplored hardware designs that offer greater performance, energy efficiency, and security. First, I established the viability of multi-ISA heterogeneity by developing a low-cost cross-ISA process migration infrastructure [1] that is orders of magnitude faster than prior art. Second, I showcased the performance and energy savings potential of multi-ISA heterogeneity via a massive design space exploration process [2]. Third, I developed a security defense that employs ISA diversification to thwart many evasive variants of the Return-Oriented Programming (ROP) attack [3, 4]. Fourth, I significantly alleviated the complexity concerns of multi-ISA heterogeneity by proposing hardware and software techniques that recreate and in many cases, supersede the gains of multi-ISA heterogeneity using a single composite-ISA derived from a large superset [5].

The infrastructure development for my dissertation work spanned multiple levels – compiler backend and toolchain, dynamic binary translation and runtime management, ISA design, processor front end and microcode design, and processor microarchitecture development. A significant portion of the compiler and runtime infrastructure was ported and transferred to the Cloud Platforms division of the IBM Haifa Research Lab in 2016. Furthermore, my dissertation research has sparked interest in multiple computer architecture and security research groups across the world. I have been invited to give talks at Intel, IBM, AMD, Qualcomm, and Technion as part of their distinguished lecture/seminar series. I also have been a regular invited guest at Qualcomm’s annual Security Summit.

In future work, I will continue to pursue cross-disciplinary research that will enable hardware design with a security focus, apply novel machine learning-based techniques to take advantage of hardware and ISA heterogeneity, and provide efficient hardware/runtime support for new computing paradigms and emerging architectures. I present my dissertation research and future work in greater detail below.

Dissertation Research
Cross-ISA Process Migration

Modern architectures allow us to instantly configure the frequency, voltage, cache size, and other microarchitectural parameters to increase efficiency. Yet our ISA choice is typically constrained by a decision made when our phone, laptop, or server was purchased. And yet, the choice of ISA can have a significant impact on execution efficiency. My research allows us to now make that choice, not just individually for each program, but every few milliseconds within the execution of a single program. However, process migration across heterogeneous ISAs is a non-trivial problem. This is because the runtime program state of an application is always kept in ISA-specific form, potentially requiring expensive state transformation at the time of migration.

In 2012, my colleagues at UCSD and I proposed novel compiler and runtime mechanisms that allow for seamless and instantaneous cross-ISA process migration at less than 5% degradation in overall performance [1]. The key components of the migration infrastructure being (1) a multi-ISA compilation framework that emits a symmetrical fat binary containing multiple ISA-specific text sections, a common stack frame organization, and a common ISA-agnostic data section, and (2) a migration runtime that performs dynamic binary translation until execution reaches a compiler-marked equivalence point, at which program state (registers and stack objects) can be safely transformed to a different ISA. We showed that careful and consistent multi-ISA compilation can enable faster and more frequent migrations by significantly minimizing the amount of runtime program state to be transformed, while simultaneously enabling most, if not all ISA-specific transformations. Furthermore, due to the relatively high frequency of compiler-marked equivalence points, we found that binary translation for cross-ISA migration calls for a different modus operandi – minimize the translation time rather than optimizing the translation itself. Overall, this work crosses a critical threshold by allowing processes to migrate across ISAs potentially every timer interrupt, while advancing prior state-of-the-art by orders of magnitude.

In Fall 2016, I was invited by the Cloud Platforms division of the IBM Haifa Research lab to transfer our multi-ISA compilation and migration runtime infrastructure to enable cross-ISA container migration [6] across POWER, Intel, and ARM servers in IBM’s datacenter. A significant portion of the container migration infrastructure will be shortly integrated into the open source CRIU (Checkpoint/Restore in User Space) framework. In addition to industry transfer, this project has further inspired independent spin-off research projects that have since explored OS support for heterogeneous-ISA multicore [7] and cross-ISA memory consistency model translation [8] respectively.
Harnessing ISA Diversity: Potential for Greater Performance and Efficiency

By decoupling the execution binary from historical ISA choices, our cross-ISA migration research has unlocked several new multi-ISA architectures that have since shown promising potential. My ISCA 2014 work shows that multi-ISA heterogeneous architectures can outperform single-ISA heterogeneous architectures by an average of 21% and save 23% in energy at no loss in performance. These results were evidenced by a massive design space exploration effort that encompassed 72 software workloads, 600 single core configurations, and 128 billion distinct heterogeneous multicore configurations that harness the diversity offered by three modern ISAs: (1) ARM’s ultra-low power Thumb, (2) the traditionally RISC Alpha, and (3) the high-performance CISC x86-64. The three ISAs offer diversity across several different axes including, but not limited to, code density, decode and instruction complexity, register pressure, and specialized support.

The design space exploration revealed two key insights. First, different applications exhibit a natural affinity for one ISA or another, and that affinity can change as the application progresses into a different execution phase. For example, in a homogeneous-ISA setting, Thumb is not a serious candidate, because it performs so poorly for certain codes; however, as part of a multi-ISA solution, it shines for certain code regions. As a result, ISA-heterogeneity consistently offers superior performance and energy savings, even in scenarios where hardware heterogeneity provides diminishing returns. Second, the ISA has a significant influence on microarchitectural design choices that enable efficient transistor investment on the available silicon real estate, calling for a tighter ISA-microarchitecture co-design. These were both previously unknown results.

This project involved a significant effort in terms of infrastructure development. I ported and integrated three different compiler backends into a unified multi-ISA compiler framework, developed six different binary translators and a stack transformer, and undertook a large design space exploration effort. However, as a researcher and an architect, I not only found the insights from the design space exploration to be incredibly stimulating, but as a graduate student, I found the overall across-the-stack learning experience to be immensely satisfying. Moreover, inferences from this work have spurred follow-up work that show promising potential. These include exploiting server-level ISA heterogeneity [6, 9, 10] that could boost energy savings by as much as 66%.

HIPStR: Thwarting ROP Attacks via Systematic ISA and Program State Randomization

Buffer overflow vulnerabilities form a major class of security exposures that plague the Internet today. These vulnerabilities have been systematically exploited by code reuse attacks such as Return-Oriented Programming (ROP) to perform arbitrary malicious computation without injecting malicious code [11, 12]. ROP attacks hinge on the attacker being able to chain together short code snippets in the program (called gadgets) that end with a return instruction, by overflowing the stack with a carefully constructed sequence of return addresses, and other malicious data.

My ASPLOS 2016 [4] work leveraged our existing cross-ISA process migration infrastructure to introduce HIPStR (Heterogeneous-ISA Program State Relocation), a security defense that has the potential to radically transform the attack landscape of state-of-the-art Return-Oriented Programming. The primary motivation for HIPStR was the fact that ROP attacks thrive on two fundamental properties. First, the knowledge of the underlying ISA is critical to construct a successful exploit. Second, any program including a ROP program requires some amount of program state (in the form of registers and stack objects) to perform computation. Owing to its unique ability to perform seamless and instantaneous cross-ISA process migration, HIPStR significantly inhibits notorious JIT-based ROP attacks [13] by forcing the attacker to chain ROP gadgets across different ISAs. In addition, HIPStR employs dynamic binary translation to continuously randomize the register and stack state to an extent that brute force attacks [14] are rendered practically infeasible on current, or even distant future microprocessors. Overall, HIPStR offers a formidable defense against several variants of ROP, with an attack surface reduction of as much as 99.7%.

The continuous dynamic program state randomization part of the defense unleashes the full power of binary translation, significantly boosting the entropy of the system. My colleagues at Intel and I first formulated it as a security feature for Intel’s Processor Binary Translation (Transmeta) engine. In 2015, we obtained a United States Patent Grant for the invention [3].

Composite-ISA Cores: Enabling Multi-ISA Heterogeneity using a Single ISA

Despite their potential for greater performance, energy efficiency, and security, the deployment of heterogeneous-ISAs on a single chip is non-trivial due to a number of practical concerns. First, heterogeneous-ISA multicores incur particular overheads owing to fat binaries and a complex process migration scheme that includes binary translation and stack transformation. Second, integration of multiple vendor-specific commercial ISAs on a single chip is fraught with significant licensing, legal, and verification costs and barriers.

In our recent ISCA submission [5], we significantly alleviate these concerns by embarking on yet another design space exploration to identify composite-ISA architectures that can recreate the effects of multi-ISA heterogeneity using a single composite-ISA. We derive composite-ISAs by leveraging a large superset ISA that resembles the Intel x86 and offers customization along five different axes of diversity: (1) register depth (8 to 32 programmable registers), (2) register width (32 vs 64-bit), (3) instruction complexity (1:1 vs 1:n micro-op encoding), (4) predication (full vs partial), and (5) specialized support (vector vs scalar). In this way, we can achieve far greater ISA diversity, enabling the full performance and energy benefits of a multi-ISA design, without the issues of multi-vendor licensing, fat binaries, and complex migration schemes.

Due to the constraint of a single ISA, composite-ISA designs can never match all advantages of distinct ISAs (e.g., code compression of Thumb). However, the greater flexibility and composability of our design offer substantial new ISA-affinity
advantages. Composite-ISA heterogeneous architectures match and in many cases, surpass the efficiency gains of multi-ISA heterogeneous architectures, and further enhance existing gains due to hardware heterogeneity by an average of 19% in performance and 31% in energy savings. Furthermore, owing to the overlapping nature of the feature sets that make up the composite ISAs, the overall cost of migration drastically drops to just 0.06%. By combining the ISA-affinity advantages of multi-ISA heterogeneity and the simplicity of single-ISA heterogeneity, this research brings the best of both worlds.

The infrastructure development for this project involved superset ISA design, full RTL design of the Intel x86 decoder with customizations and associated power/area analysis, compiler backend development for efficient utilization of the component ISA features, Intel front end simulation infrastructure development (e.g., micro-op cache and micro-op fusion), and runtime development for software emulation of missing features (e.g., during a feature downgrade from 64 to 32 registers). Furthermore, the design space exploration (of about 500 trillion heterogeneous multicore designs) involved a massive simulation effort of over 100,000 core hours on a large 2 petaflop supercomputer.

**Future Work**

My dissertation research has demonstrated that substantial benefits arise by strengthening the hardware/software interface, specifically the ISA and the runtime system, with diverse capabilities. The expanding gap between the increasingly diverse software and the underlying heterogeneous hardware offers significant additional potential that is yet to be harnessed. The research that stems out of this endeavor will not only enable greater levels of performance, energy efficiency, and security in state-of-the-art computing systems, but will further drive emerging technologies.

My research will continue to explore more underexploited levers to transform the architectural landscape for greater gains in efficiency and security. Some of those opportunities include (1) the internal ISA-translation feature of modern CISC engines that can allow the processor to radically change the energy and security characteristics of running code without changing user-visible execution, (2) the universal availability (in virtually all execution environments) of machine-readable program text that can be fed into powerful NLP-based techniques to model dynamic execution behavior, (3) underexploited architectural innovations in modern Intel processors that can accelerate deeply embedded software layers of data analytics engines, and (4) disruptive binary translation techniques that can enable platform-independence for even evolving architectural paradigms, such as quantum computing, that are still very much in their infancy. I describe these in greater detail below.

**Mobilizing the Micro-Ops: Context-Sensitive Decoding for Enhanced Security and Efficiency**

Modern processors employ translated ISAs, as the Intel and AMD x86 processors and many ARM processors typically feature translation from the native instruction set into internal micro-ops that enter the pipeline for execution. These architectures enjoy the dual benefits of a versatile backward-compatible CISC front-end and a simple cost-effective RISC back-end. However, in such architectures, the translation is static, changing once per generation. Instead, I propose that translation be dynamic, potentially changing frequently within the execution of a single program. In this way, for example, a performance-optimized code region can become energy-optimized, or a legacy insecure executable can instantly and seamlessly feature several security properties, without recompilation or binary translation.

This project aims at unlocking the full potential of translated ISAs via context-sensitive decoding, a technique that allows native instructions to be translated into a different set of custom micro-ops based on their current execution context. This presents operating systems, runtime systems, and antivirus programs with the unique opportunity of triggering different custom translation modes, at the microsecond granularity, by simply configuring a set of model-specific registers (MSRs) and/or via the already well-established Intel microcode update procedure [15, 16, 17], with no significant changes to the hardware. Furthermore, due to its potential low performance overhead and non-intrusive nature, context-sensitive decoding could have potential applications that span several disciplines of computer science – computer security (dynamic information flow tracking, side channel obfuscation, malware detection, bounds checking, and capability-based access control checks), programming languages (dynamic instrumentation, software fault isolation, and code specialization for performance/energy optimization), software engineering (fine-grained watchpoint configuration and program verification support), and performance analysis (runtime profiling and phase tracking with highly configurable performance counters). I present a specific security application from preliminary research below.

Cache-based side channel attacks typically involve probing one or more cache lines of a co-located victim in order to capture its memory access patterns that could potentially reveal secret information. For example, an attacker who intends to break a cryptographic algorithm could compute one or more bits of a secret key by capturing cache access patterns of key-dependent loads and branches. The goal of an obfuscation-based side-channel defense is to provide an illusion of a modified architectural state by obfuscating the micro-architectural characteristics alone. This specific application of context-sensitive decoding dynamically injects decoy micro-ops into execution that obfuscate a victim’s control path and/or access to sensitive data structures in an attacker-oblivious way (e.g., by prefetching all cache blocks that contain T-tables of AES and the multiply functions of RSA). Our ongoing research shows that this defense completely thwarts instruction and data cache-based side channel attacks on both RSA and AES, while outperforming state-of-the-art software solutions [18] by 20X. Furthermore, owing to its proximity and unfettered access to hardware structures, a context-sensitive decoding-based defense has the potential to defend against several other side channel attacks such as branch prediction, timing, and power analyses.
NLP-Inspired Text and Sequence Modeling of Machine Code

Natural Language Processing (NLP) has become an integral part of the computing universe, with applications ranging from intelligent personal assistants such as Apple’s Siri and Google Now, to large-scale question answering systems such as IBM’s Watson. Extensive research in NLP, linguistics, and machine learning has enabled AI machines to infer contextual and semantic information from not just written documents and speech, but even abstract data forms like music. In software engineering, NLP-inspired source code analysis exploits well-structured syntactic dependencies in high-level programming languages to enable automatic source code plagiarism detection, software modularization, documentation, and bug localization [19, 20, 21, 22, 23]. I propose exploring state-of-the-art NLP-inspired text and sequence modeling techniques to extract semantic information from machine code and further predict dynamic power, performance, and security characteristics of a running application.

The key to this exploration is the availability and ubiquity of a large corpus of machine executable code running on the world’s computing infrastructure everyday, and the fact that the ISAs that encode such machine code offer compact opcode vocabularies and well-defined notions of data and control dependences. Furthermore, the classical control-flow execution model allows us to draw a striking analogy between machine executable code and text documents written in natural language. For example, basic blocks of a program can be modeled as paragraphs and execution phases as chapters of the document. However, unlike natural language documents, machine code is characterized by high determinism and repeatability, which potentially open up new avenues to explore in terms of both topic and sequence modeling research.

NLP-inspired techniques could bring several new advantages to modern runtime analysis frameworks. First, the ability to establish code similarity could enable automatic phase characterization, verify particular security and privacy properties of co-located applications, and further assist the runtime in making global scheduling and co-scheduling decisions. Second, by inferring program semantics on-the-fly, we could take significant burden off the programmer to explicitly specify them in source code using a complex mix of heterogeneous programming models. Third, semantic clustering of machine code could help identify candidate regions that can be offloaded to custom accelerators. Finally, by combining the inferred contextual information from machine code with the massive database of power/performance statistics from my dissertation research, we could significantly expand the capabilities of state-of-the-art learning-based cross-platform performance prediction mechanisms.

Hardware Acceleration of Data Analytics Engines

Modern Intel processors sport several architectural techniques that were seemingly invented to deal with the specific nuances of the x86 ISA. This project aims to bring several such mundane “general purpose” techniques to light as acceleration mechanisms for new execution paradigms in data analytics. For example, the Intel front end already features a suite of parallel speculative instruction length decoders to efficiently parse variable-length x86 instruction records. The architectural mechanisms from this feature can be leveraged to accelerate JSON parsing, which is by far, the biggest bottleneck for most data analytics applications. In fact, parsing raw JSON data makes up more than 80% of the total query processing time, even for complex queries that involve joins and aggregations [24]. This is due to the fact that JSON records typically feature deeply nested and variable-length arrays and dictionaries, which makes software parsing extremely cumbersome. However, the parallel speculative decoding scheme of x86 is a natural fit for parsing JSON records.

Software parsers that exploit SIMD parallelism and coarse-grained speculation already showcase an order of magnitude improvement in performance. The inherently parallel and speculative nature of hardware accelerators would not only further amplify these gains in performance, but potentially open up additional pockets of opportunities for optimizing other deeply embedded software layers of data analytics engines, including query evaluation and type conversion. This allows, for example, pipelining speculative parsing of JSON records with type conversion of raw string objects to their respective native binary representations. Type conversion itself can benefit from the overloaded nature of Intel’s micro-op translation mechanisms and seamlessly support multiple levels of the type hierarchy. At a much broader level, by trading off transistors for specialized acceleration units, we could potentially influence several architectural design choices for such dedicated server SKUs.

Binary Translation for Emerging Architectures

Quantum computing has been showcased as a promising computing paradigm that can provide dramatic speedups for certain well-known classically intractable problems. However, with quantum computer architecture still in its infancy, quantum computers will most likely be used as co-processors alongside classical computers for the foreseeable future. This is due to three primary reasons. First, it is still unclear if there is a substantial performance advantage to developing quantum algorithms to perform a majority of today’s general-purpose computation [25]. Second, quantum computing is not without resource constraints, calling for hybrid classical/quantum algorithms that leverage classical techniques to perform efficient computation on a minimal set of quantum resources [26, 27]. Third and most importantly, the fragility of quantum data requires interfacing with a fast and error-correcting cryogenic control computer that generates an appropriate set of pulses and signals to enable fault-tolerant computation. The control computer itself is typically driven by a large interfacing classical computer that streams in physical quantum gate sequences, and processes intermediate results as required [28, 29].

The physical quantum gate sequences and error correction mechanisms are highly target-specific, and could significantly diverge based on the underlying qubit topology of the quantum machine. This is further exacerbated by the fact that different vendors such as IBM, Microsoft, Intel, and Google have already started to diverge in terms of their custom implementations of the quantum hardware/software stack, a trend that is going to invariably result in platform-specific quantum executables that
contain physical gate sequences tailored to a specific platform. I propose a binary translation infrastructure that can transform a quantum application compiled to one platform to seamlessly execute on another. Since many quantum applications typically intersperse classical code with quantum code, this research will not only leverage existing mechanisms from conventional binary translation, it will also propose several new techniques and transformations that are unique to quantum computing. This should be seen as the first step in creating a platform-independent programming model for quantum computing.

This research will address several interesting challenges. First, translation of quantum gate operations could involve multiple transformations between the source and the target physical Quantum Abstract Machines (QAM) [30] via a logical QAM. This is akin to most modern binary translators that leverage a suitable intermediate representation to assist cross-ISA translation. The complexity of such transformations, however, is a function of the divergence between the different QAMs with respect to the number and type of quantum resources they sport. Second, the resulting physical gate sequences should be carefully scheduled and optimized for parallelism by appropriately commuting gate operations and measurements [30, 31]. Finally, the applied transformations should preserve and/or enhance existing fault-tolerance levels in order to seamlessly cope with the differences in error correction mechanisms.

In summary, my research will leverage my across-the-stack experience in Computer Architecture and Compilers to develop secure, efficient, and robust hardware and software systems with the potential to impact several disciplines of Computer Science, and further lay strong foundations to enable future technologies.
References


