Research Statement

David Schall
Technical University of Munich (TUM)
david.schall@tum.de

We are at a critical moment for computer architecture. Moore's Law has slowed, AI has created unprecedented computing demands, and underlying this is the pressing need for sustainable solutions, driven by the urgency of climate change. Together, these challenges have exposed the limits of incremental improvement. As CPUs ceased delivering needed performance gains, industry and academia embraced heterogeneity—building specialized accelerators for specific workloads. While pragmatic, this approach is shortsighted. Accelerators address immediate performance needs but sacrifice long-term sustainability: software evolves rapidly while silicon remains fixed, and research shows that manufacturing costs often exceed lifetime operational savings, resulting in a net loss in environmental footprint [4, 6]. The most sustainable platform, therefore, remains the general-purpose CPU.

This observation drives my research: *How can we redesign the CPU to enable better adaptation to applications, achieving accelerator-level performance while maintaining the generality required for long-term sustainability?* I argue that a key bottleneck preventing this adaptation is the rigid hardware-software interface. While today's software is written at a highly abstract level—allowing programmers and compilers to express and extract rich semantic intent—this information is lost when compiled to low-level instructions. Although modern hardware expends vast resources attempting to rediscover this intent, this approach is fundamentally limited and wasteful: discarding information only to expend energy and silicon attempting to reconstruct it.

My vision is an *Application-Directed Microarchitecture* that bridges the semantic gap with a secondary interface, allowing software to communicate high-level program semantics directly to hardware and enabling processors to tailor execution precisely to each application while preserving generality. In this statement, I outline my research methodology and draw on past experience that has led to this vision, describe my current work in building the necessary cross-stack foundation, and detail my future research agenda to achieve it.

Research Methodology

My research philosophy is grounded in first-principles thinking. Every project begins with a deep analysis and detailed characterization to drill down to the root cause of a problem. As an architecture researcher, I believe in viewing computer architecture through the eyes of a software developer; *What* does the application want to achieve, and *how* can the hardware architecture best support it.

I advocate for ambitious research goals that may challenge the current state of the art. As academic researchers, we should not fall into the trap of merely optimizing mechanisms for marginal percentage gains—that is the role of industry. Our role is to think radically, questioning the abstractions and interfaces that seem set in stone. Just because a mechanism is standard today does not mean it will remain so forever.

However, while the high-level vision must be ambitious, execution must remain grounded with tangible steps and milestones. I apply this principle in my own research and in mentoring students: starting with well-scoped projects that build deep expertise, then progressively tackling more ambitious cross-stack challenges. My research trajectory reflects this: my past work focused on gaining deep microarchitectural understanding, while my current work broadens my cross-stack expertise—building the foundation necessary to realize an Application-Directed Microarchitecture.

Past and Current Research

Microarchitecture and Data Center Workloads

My doctoral research focused on microarchitectural innovations for data center workloads. I analyzed how workload behavior has evolved to identify how hardware must adapt.

Context switching and cold state. In *Jukebox* [10] and *Ignite* [11], I characterized serverless workloads. These represent a key trend toward cloud-native applications but are characterized by extremely short execution times and high context switching rates. This stands in sharp contrast to traditional long-running workloads, preventing the CPU from ”warming up” its predictors and caches. We developed lightweight mechanisms allowing the CPU to preserve learned information between context switches, significantly improving efficiency. This work received an Honorable Mention at the IEEE MICRO Top Picks 2022 [3].

Branch prediction. In a second line of work, I addressed branch prediction for data center workloads with massive instruction working sets [2, 7]. In *The Last-level Branch Predictor* [12, 13], we showed that although larger predictors offer higher accuracy, they remain constrained by a fundamental latency trade-off. By taking an application-centric view, we found that predictor metadata exhibits locality rooted in the application's high-level control flow. This insight enabled the development of a hierarchical predictor organization that breaks the latency barrier by decoupling prediction latency from capacity.

These projects demonstrated to me how critical high-level semantic information is to CPU performance. They demonstrated both the cost of losing this information and the potential benefits of exploiting application-level insights in microarchitectural designs.

Broadening Horizons: System and Cross-Stack Research

For my postdoc, I expanded into systems-oriented research, which has been both successful and formative: I have published at top venues in new areas and led successful grant proposals. More importantly, working at the intersection of hardware, operating systems, and applications taught me the cross-stack perspective and confidence necessary to pursue my vision of redefining the hardware-software interface.

Distributed cache-coherent systems (CXL). For data center applications, a key determining factor for performance is efficient data management and communication across distributed nodes. Therefore, we conduct research in a hardware-accelerated, cache-coherent shared memory abstractions via CXL. We recently published works addressing how heterogeneous hosts—Arm or x86 CPUs with different cache coherence protocols and memory consistency models—can share a common memory pool [9, 8], eliminating the need for explicit software data movement. As part of this project, I led a successful grant application to the german science funding agency DFG.

User-managed page-fault handling. Inspired by User-space Interrupts in modern CPUs, I am leading a project to enable hardware-accelerated page-fault mechanisms managed entirely in user space for Database Management Systems. In this project, we combine my microarchitectural knowledge with the OS, database, and kernel teams to develop a truly cross-stack, application-centric solution.

Dynamic Quantum Circuits. Leading a grant proposal for the Bavarian State Science Foundation, we are developing a project with the quantum team at the chair to apply conventional microarchitecture techniques, such as speculative execution or predication, to dynamic quantum circuits, where high readout delays create feedback loops that hurt throughput and accuracy.

Research Vision: Application-Directed Microarchitecture

My past and current research has formed my central conviction: the rigid hardware-software interface is the key bottleneck for performance, as it strangles CPU adaptability. While my microarchitecture work revealed how hardware struggles to rediscover lost semantic information, and my systems work demonstrated the power of cross-stack co-design, these experiences collectively pointed to the same fundamental problem—the ISA provides no mechanism for communicating high-level program semantics to hardware.

My vision is to bridge this semantic gap by augmenting the ISA with a secondary interface—an additional channel through which software and compilers can communicate application intent and high-level semantic information to the hardware. Critically, this channel is optional, orthogonal, and not required for correctness. The hardware can ignore it entirely and still execute code faithfully. This ensures that different processor designs, with different power and area constraints, can exploit this information in different ways while preserving the ISA's generality. This interface is comparable to software pragmas or compiler hints, but rather than stopping at the compiler, this information is generated by the compiler itself and passed all the way to the microarchitecture. For example, the CPU currently expends significant resources attempting to dynamically learn program characteristics—such as a branch's behavior, or the high-level data structure a memory region represents. Using the orthogonal interface, the compiler could pass concise annotations detailing this semantic intent, such as: *"This branch is loop-invariant and has a known reconvergence point at X,"* or *"This memory range holds temporary, low-reuse data."*

These semantic hints, which are entirely optional and irrelevant to the functional ISA, allow the hardware to make targeted microarchitectural decisions without costly dynamic training. For instance, a loop-invariant branch hint can enable highly aggressive speculation, while the reconvergence point allows for selective squashing during misprediction recovery. Conversely, tagging a memory range as temporal data can inform the memory subsystem to *not* cache this region, preventing it from polluting the caches and preserving capacity for high-reuse data.

Future Research Agenda

My research experience across two diverse groups has positioned me well to pursue end-to-end hardware-software co-design. Expertise in microarchitecture, combined with experience in operating systems, compilers, and distributed systems, enables me to work across research silos, which is essential for realizing my cross-stack vision. In the near term, I will focus on three research directions that explore different facets of the semantic interface, as well as a consolidating project addressing its cross-stack realization.

1. Semantic Interface for Speculative Execution and Recovery. Modern CPUs pay a heavy penalty for speculation errors, often discarding the work of hundreds of instructions by flushing the entire pipeline. While prior work has shown that performance and efficiency can be saved by only selectively squashing and re-executing affected instructions [1, 5], these techniques remain largely unadopted due to high implementation complexity and limited coverage. The complexity stems from the hardware's dynamic need to rediscover control-flow boundaries and reconvergence points—information that is readily available to the compiler. I plan to use the orthogonal interface to allow the compiler to mark independent code regions, convey reconvergence points, and provide branch confidence hints. This high-level semantic knowledge eliminates the need for expensive dynamic hardware learning, overcoming the complexity barrier and enabling highly efficient, high-coverage selective squashing mechanisms.

2. Co-designed Hardware Prediction and JIT Compilation. A significant fraction of today's chip area is devoted to predictors that learn complex patterns in control flow and data access.This project explores how to shift part of this learning burden to the software stack. By using the semantic interface, the compiler can extract high-level program patterns and communicate them directly to the hardware. This reduces the need for long dynamic training phases and frees up hardware resources, enabling simpler and more effective prediction mechanisms. Furthermore, this project will explore the opportunities for a *bidirectional* interface, allowing the hardware to provide fine-grained performance feedback to the JIT compiler. For instance, the hardware could mark a consistently hard-to-predict branch, prompting the JIT to transform it into simpler operations (e.g., conditional moves) or reorder code to simplify prediction complexity, creating a truly collaborative execution environment between the hardware and the software runtime.

3. Machine Learning for Semantic Extraction. The compiler is deterministic but often lacks high-level program understanding, while ML models excel at extracting semantic intent but fail to produce reliably bug-free low-level output. The proposed semantic interface provides a unique opportunity to unify these strengths. This creates a powerful **division of concerns**: since the interface is optional and orthogonal to correctness, the ML model is relieved of the burden of functional accuracy, allowing it to focus entirely on learning and injecting high-level semantic intent, while the compiler remains solely responsible for guaranteeing deterministic correctness via the functional ISA.

4. Interface Design and Cross-Stack Realization. While the preceding projects address *what* semantic information can be conveyed, this project addresses *how* to design an interface that is expressive, extensible, and efficient across the entire stack. The project tackles interrelated questions across three perspectives. From a design perspective: What principles guide a robust semantic interface that remains flexible as new semantic categories emerge? From an implementation perspective: How does semantic information integrate into the microarchitecture alongside instruction and data flows? Does it require dedicated caching mechanisms? Should metadata reside in instruction streams, memory-mapped regions, or separate channels? From a systems perspective: How do we define clean APIs spanning from high-level languages through compilers and operating systems down to hardware?

Answering these questions requires expertise across the full stack and will synthesize insights from the preceding projects to produce a concrete foundation for an Application-Directed Microarchitecture.

References

[1] Ahmed S. Al-Zawawi, Vimal K. Reddy, Eric Rotenberg, and Haitham H. Akkary. Transparent control independence (tci). In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, page 448–459, New York, NY, USA, 2007. Association for Computing Machinery.

[2] Grant Ayers, Nayana Prasad Nagendra, David I. August, Hyoun Kyu Cho, Svilen Kanev, Christos Kozyrakis, Trivikram Krishnamurthy, Heiner Litz, Tipp Moseley, and Parthasarathy Ranganathan. AsmDB: understanding and mitigating front-end stalls in warehouse-scale computers. pages 462–473. ACM, 2019.

[3] Christopher Batten and Jae W. Lee. Special Issue on Top Picks From the 2022 Computer Architecture Conferences . *IEEE Micro*, 43(04):6–10, July 2023.

[4] Lieven Eeckhout. Focal: A first-order carbon model to assess processor sustainability. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 401–415, New York, NY, USA, 2024. Association for Computing Machinery.

[5] Stijn Eyerman, Wim Heirman, Sam Van Den Steen, and Ibrahim Hur. Enabling branch-mispredict level parallelism by selectively flushing instructions. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 767–778, New York, NY, USA, 2021. Association for Computing Machinery.

[6] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing carbon: The elusive environmental footprint of computing. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 854–867, 2021.

[7] Tanvir Ahmed Khan, Muhammed Ugur, Krishnendra Nathella, Dam Sunwoo, Heiner Litz, Daniel A. Jiménez, and Baris Kasikci. Whisper: Profile-Guided Branch Misprediction Elimination for Data Center Applications. pages 19–34. IEEE, 2022.

[8] Anatole Lefort, Julian Pritzi, Nicolò Carpentieri, David Schall, Simon Dittrich, Soham Chakraborty, Nicolai Oswald, and Pramod Bhatotia. vCXLGen: Automated Synthesis and Verification of CXL Bridges for Heterogeneous Architectures. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2026)*, 2026.

[9] Anatole Lefort, David Schall, Nicolò Carpentieri, Julian Pritzi, Soham Chakraborty, Nicolai Oswald, and Pramod Bhatotia. C^3: CXL Coherence Controllers for Heterogeneous Architectures. In *Proceedings of the 32nd IEEE International Symposium on High-Performance Computer Architecture (HPCA-32)*, 2026.

[10] David Schall, Artemiy Margaritov, Dmitrii Ustiugov, Andreas Sandberg, and Boris Grot. Lukewarm serverless functions: characterization and optimization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 757–770, New York, NY, USA, 2022. Association for Computing Machinery.

[11] David Schall, Andreas Sandberg, and Boris Grot. Warming up a cold front-end with ignite. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 254–267, New York, NY, USA, 2023. Association for Computing Machinery.

[12] David Schall, Andreas Sandberg, and Boris Grot. The last-level branch predictor. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, MICRO '24, pages 464–479. IEEE, 2024.

[13] David Schall, Mária Ďuračková, and Boris Grot. The last-level branch predictor revisited. In *Proceedings of the 32nd IEEE International Symposium on High-Performance Computer Architecture (HPCA-32)*, 2026.