Paper title: A Comparison of Software and Hardware Techniques for x86 virtualization
Paper authors: Keith Adams, Ole Agesen
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
The paper talks about the various methods for virtualization - software-based techniques like trap-and-emulate virtualization, and simple and adaptive binary translation, along with the more recent hardware support introduced by Intel and AMD. This introduction of hardware-backed VMMs is expected to improve performance, but does not always do so. The paper provides a comprehensive analysis of both domains, along with an in-depth reasoning behind each observation.

The **RISC vs. CISC debate** serves as a motivation towards the ideology that hardware might not always outperform software, which is what this paper discovers. **picoJava microprocessor** was built to redo the software JVM, but it was underperforming as compared to the modern JVM. **Transmeta's Crusoe** built a CPU compatible with x86 architecture using a VLIW core running a large software which incorporated a binary translator. Paravirtualization, which effectively means modifying an operating system for effective virtualization, is a concept that was brought to life by open-source operating systems like Linux. This kind of virtualization was effective, but was restrictive to the remaining operating systems, along with the addition of the requirement of a standard interface between the OS and the hardwa re.

What are the key contributions of the paper?
The paper provides a comprehensive comparison of VMware Workstation's software VMM to the more recent hardware-assisted VMMs.

The x86 architecture was not virtualizable using the classical trap-and-emulate technique, due to the two main obstacles - first, the guest VM is able to notice whenever it is deprivileged and second, whenever privileged instructions run at the user-level, no trap happens as any user-mode *popf* flag blocks any attempt to modify the IF flag. For this reason, a VMM that leverages binary translation is used. VMware Workstation's software VMM uses a binary translator that is dynamic, on-demand and outputs a safe subset of the full x86 instruction set. It works at system-level as the rules are based on the x86 ISA and is adaptive in nature. It also uses the notable chaining optimization to allow a compiled code fragment to proceed to the next CCF without calling the translation cache (TC). The translator works on the assumption that the guest code is perfectly optimized by the developers and runs it as it is. For memory accesses, the VMM prefers the segmentation mechanism instead of performing address checking. While a simple binary translator-based VMM outperforms a classical VMM, it is unable to eliminate traps from non-privileged instructions. For this, an adaptive binary translator is used, which allows guest instructions to start in an innocent state, and then adapts based on various events such as change in frequency of the traps, accesses to page tables, devices or the VMM's address range.

AMD and Intel provided several hardware extensions like the VMCB, which is an in-memory data structure. It also provides the *guest* mode, which allows the direct execution of the guest code. Vmrun is a new instruction which allows transfer from host to guest mode. The additions allow the VMM to reduce the frequency of exits.

To achieve native speeds for the guests, both software and hardware-backed VMMs implement several optimizations and deal with several trade-offs. Overall, binary translation is able to eliminate most traps, achieve good emulation speed and avoid callout costs. However, the hardware VMM is able to surpass the former in preserving code density by skipping translation, achieves precision in exceptions, and is able to execute system calls without the VMM's mediation.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The experiments on the various benchmarks highlight the performance of the two VMMs. For low computations, both software and hardware-backed VMMs show good performance. For the Apache workload on Windows OS, and the 2D PassMark workload, the hardware-assisted VMM outperforms the software VMM. But in every other case, the software VMM is able to perform better than its hardware counterpart. This is because the hardware VMM is unable to effectively adjust to the frequently exiting workloads. In terms of overhead as well, the binary translator is much more efficient compared to the hardware VMM. Thus, the paper demonstrates, through a series of benchmarks, that while the hardware extensions do provide some optimization for the x86 architecture, the software VMMs far outperform their hardware-assisted counterparts.

Write down one question you may want to ask the authors.
What were some challenges faced in performing an apple-to-apple comparison of the software and hardware-assisted VMMs, especially when creating a special VMM to run on the new hardware extension?