Alisha Mehta
Professor Quon
ECE 4300

ECE 4300 Final Exam Code

Section 1: Introduction to Security Exploits Hierarchical Memory Pipelining and Out of Order Execution

Modern computer systems rely on increasingly complex processor designs to deliver high performance, efficiency and scalability. However, these design choices also introduce vulnerabilities that attackers can exploit at the hardware level. Security exploits such as Rowhammer and Spectre demonstrate how flaws in memory and execution mechanisms can be leveraged to corrupt data or leak sensitive information, challenging long-standing assumptions about processor reliability. At the same time, understanding the memory hierarchy, including cache associativity, size, and placement policies, is critical for balancing speed, cost, and efficiency in system design. Finally, performance enhancements such as pipelining and dynamic scheduling algorithms like Tomasulo's show how instruction-level parallelism can be exploited to maximize throughput, while still facing hazards and design trade-offs.

This paper explores these interconnected topics by examining security exploits, memory hierarchy design, and pipelining techniques, highlighting both their benefits and limitations in shaping modern processor architecture.

Section 2: Security Exploits

Security exploits exist in memory due to security vulnerabilities which come from the way a processor's internal components and operations are designed and implemented. This allows attackers to extract sensitive information or manipulate system behavior. Two examples of these attacks are Spectre and Rowhammer.

The rowhammer attack refers to a vulnerability in DRAM chips where repeatedly accessing (reading or writing to) a specific row in DRAM can cause data corruption in physically adjacent memory rows without directly accessing them [2]. A disturbance error is caused when a DRAM row is activated (opened) and then precharged (closed) many times in rapid succession, the resulting voltage fluctuations on its wordline stress inter-cell coupling effects. This may lead to a security vulnerability where a system is hijacked and memory that belongs to other programs is corrupted.

Some solutions are outlined such as: creating better circuits, ECC (error correction code) and refreshing all rows repeatedly. But there are shortcomings of all three solutions.

Creating better chips by improving intercell isolation. Although, this problem could resurface with future process technology upgrades as cells continue to shrink and become more vulnerable.

The second solution ECCs are not failsafe against disturbance errors because these errors often lead to multi-bit errors within 64-bit words, which SECDED cannot correct or even detect. This solution is also very costly so it is rarely used in consumer grade systems.

The last proposed solution to retire cells to refresh all cells repeatedly and retire cells by taking out victim cells have their limitations as it significantly degrades performance and energy efficiency. Shortening the refresh interval from the default 64 ms to 8.2 ms would increase the time spent performing refreshes from 1.4–4.5% to an unacceptable 11.0–35.0% [2].

The paper also discusses PARA (Probabilistic Adjacent Row Activation) which is the proposed low-overhead solution. Simulations show that PARA has a small performance impact, with an average degradation of only 0.197% in instruction throughput [2].

PARA works whenever a row is opened and then closed, the controller probabilistically refreshes one of its adjacent rows with a very low probability. If the aggressor row is toggled many times, this statistically ensures that its adjacent rows will eventually be refreshed.

The paper also highlights some advantages and shortcomings of PARA. Advantages of PARA are: it is stateless (doesn't require expensive hardware) and it offers strong statistical guarantee against disturbance errors. But the second advantage can also serve as a disadvantage as it does not give absolute certainty as there's still a minute chance that an adjacent row might not be refreshed in time, especially under an adversarial access pattern. Arguably, in comparison to other system components the failure rate probability is extremely low (e.g., 9.4 x 10^-14 for Nth=100K over one year with p=0.001) [2].

The Spectre attack is a class of microarchitectural attacks that exploit speculative execution in modern processors to leak confidential information via the side channel. It accomplishes this attack and causes several side effects such as: induces erroneous operations, bypasses security mechanisms and affects a wide range of processes [3].

Solutions to the Spectre attacks discussed by the paper are countermeasures and not definitive solutions. Preventing speculative execution with lfence instructions is considered an effective countermeasure against Spectre attacks because it enforces strict control flow. However, this approach significantly reduces processor performance. To fully secure conditional branches, lfence would need to be inserted on both outcomes, causing dramatic slowdowns. Another limitation is that all vulnerable software must be instrumented with these instructions. This creates challenges for legacy applications that cannot easily be modified.

Section 3: Hierarchical Memory

Hierarchical memory is the organization of computer memory into multiple levels based on speed, size, and cost. At the top are the fastest but smallest memories, such as CPU registers and cache. Main memory (RAM) provides larger capacity but is slower and more costly per bit than storage. Secondary and tertiary storage, like SSDs, are the slowest but offer massive capacity at low cost. This structure takes advantage of the principle of locality, keeping frequently used data in faster memory while less-used data is stored in slower memory.

Associativity in computer architecture refers to how memory blocks map to cache locations. It defines how many possible places in the cache a given block of main memory can be stored. A direct-mapped cache allows each block to go to only one location, while a fully associative cache lets a block go anywhere, and a set-associative cache strikes a balance by grouping cache into sets with multiple possible ways. Higher associativity reduces cache conflicts but increases hardware cost and access time, while lower associativity is cheaper and faster but more prone to misses.

Cache size is a critical design factor in the memory hierarchy, balancing speed, cost, and hit rate. A larger cache can store more data, which generally increases the cache hit rate and reduces the need to access slower main memory. However, larger caches also have longer access times and higher power consumption, which can reduce overall performance gains.

Smaller caches are faster and cheaper but suffer from more cache misses, forcing frequent memory accesses. Designers therefore choose cache sizes at each level (L1 small and fast, L2 moderate, L3 large) to optimize the trade-off between speed and capacity within the hierarchy.

Section 4: Pipelining

In the video *Dynamic Scheduling Using Tomasulo's Algorithm*, Dr. Juurlink discusses Tomasulo's algorithm which encompasses the unique pipeline. Dynamic scheduling happens when functions are stalling (for example, subtract stalls in the decode stage) but since it does not depend on any other instruction it can execute out of order. Out of order execution would look like the subtract executing simultaneously while add double stalls. Robert Tomasulo's algorithm was created to achieve high floating point performance. The IBM 360/91 processor had only 4 double precision registers which caused limitations on compiler scheduling. He achieved this by implementing register renaming in hardware. The algorithm has buffer stations and gets informed so that the stalled instruction can proceed which are broadcasted in CDBs (Common Data Busses). Tomasulo has several different pipeline phases:

1. IF Stage: where data is fetched in a FIFO queue of pending instructions
2. Issue: next instruction is fetched form queue, if there is a free matching reservation station, then the instruction is placed there
    a. If there is no matching reservation then there is a structural hazard and the execution has to stop
3. Eexecute: instructions executed if all operands are available (no RAW)
4. Write result: data written and reservation station marked free

Section 5: Conclusion

The study of processor architecture reveals a constant tension between performance, efficiency, and security. Section 2 highlighted how security exploits like Rowhammer and Spectre arise directly from design optimizations. While speculative execution improves instruction throughput, it inadvertently opens side channels, and while high-density DRAM reduces cost per bit, it introduces disturbance errors. The proposed defenses—such as ECC, refresh mechanisms, or instruction fencing—demonstrate that security countermeasures often come at significant performance or cost penalties, underscoring the incomplete nature of current solutions.

Section 3 examined the memory hierarchy, where cache associativity and size are carefully tuned to exploit locality and reduce latency. Although these designs improve performance, they also increase complexity and power consumption. Furthermore, hierarchical memory systems remain vulnerable to both software inefficiencies and hardware-level exploits, showing that faster memory does not guarantee safer or more reliable memory.

Section 4 analyzed pipelining and dynamic scheduling, which aim to maximize throughput by overlapping instructions and exploiting out-of-order execution. While these techniques significantly boost performance, they also introduce new hazards and complexity in hardware, making correctness and predictability harder to guarantee. Tomasulo's algorithm, for example, resolved some register bottlenecks but further expanded the surface for subtle timing- and state-related vulnerabilities.

Overall, these examples reveal that many architectural designs prioritize performance first, with security and reliability treated as afterthoughts. Future improvements must focus on designing processors with security, predictability, and energy efficiency as first-class goals, rather than retrofitting countermeasures onto inherently insecure mechanisms. Promising directions include hardware-software co-design for security, novel memory technologies less prone to disturbance errors, and simplified but secure execution pipelines that balance performance with verifiability. As computing systems continue to expand into critical infrastructure, addressing these fundamental trade-offs will remain central to the evolution of computer architecture.

Bibliography

[1] Prof. Dr. B. H. Juurlink, “2 1 3 Dynamic Scheduling Using Tomasulo’s Algorithm.”

[2] P. Kocher *et al.*, “Spectre attacks,” *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, Jun. 2020, doi: https://doi.org/10.1145/3399742.

[3] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, Minneapolis, MN, USA, 2014, pp. 361–372. doi: 10.1145/2678373.2665726.