Exploring Data Security on Microprocessor Hardware

Meet Udeshi
14D070007
Supervisor: Prof. Virendra Singh

CADSL - IIT Bombay

June 16, 2019

Introduction

- Multi-thread, multi-core hardware focus on sharing computing resources among different programs.
- In cloud computing, technologies like virtual machines and virtual environments are allowing multiple different programs to share the same computing resources.
- Programs running on current hardware inevitably leave fingerprints of their execution in power traces, EMI spectrum, caches, etc.[1]
- Attackers with the right knowledge and tools can leverage hardware implementation flaws in the design of these shared resources to extract data from a victim process via undetectable side-channels.

---

[1][Chenglu Jin, Side Channel Attacks]

Contents

Side Channels

Shared resources of the processor can leak information about the tasks being performed in it. Measuring cache hit/miss, time of execution, power consumption, EMI spikes can determine what part of code is running or what data is being processed.
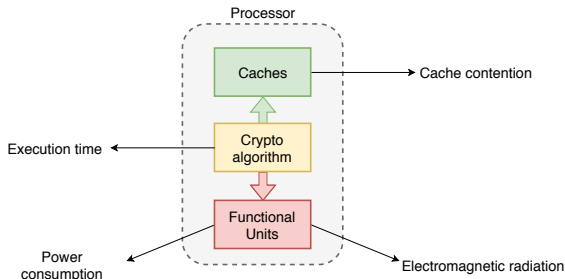

Figure: A number of side channels which are capable of leaking data.

Data dependent execution

Fast exponentiation algorithm used in RSA[2]

```
while ( key > 0 ) {
    e = key % 2;

    Square ( ) ;
    Reduce ( ) ;
    if ( e == 1) {
        Multiply ( ) ;
        Reduce ( ) ;
    }

    key >>= 1;
}
```

S-box access used in AES[3]

```
for ( i =0; i <9; i ++) {
    x = k ;
    y = k ^ n ;

    e = {S [ x [ 1 ] ] ^ 1 ,
         S [ x [ 2 ] ] ,
         S [ x [ 3 ] ] ,
         S [ x [ 0 ] ] } ;
}
// Just for last round
y [ 0 ] = S [ y [ 0 ] ] ^ S [ y [ 1 ] ] ^
         S [ y [ 2 ] ] ^ S [ y [ 3 ] ] ^
         x [ 0 ] ;
```

[2][C. Percival, Cache missing for fun and profit]
[3][D. J. Bernstien, Cache-timing attacks on AES]

Cache Side Channel

1. Determine the cache-line accessed by victim using contention in shared cache.
2. In set-associative cache design, that reveals a few bits of the address.
3. For publicly available encryption libraries (OpenSSL), determine memory region of victim accesses.
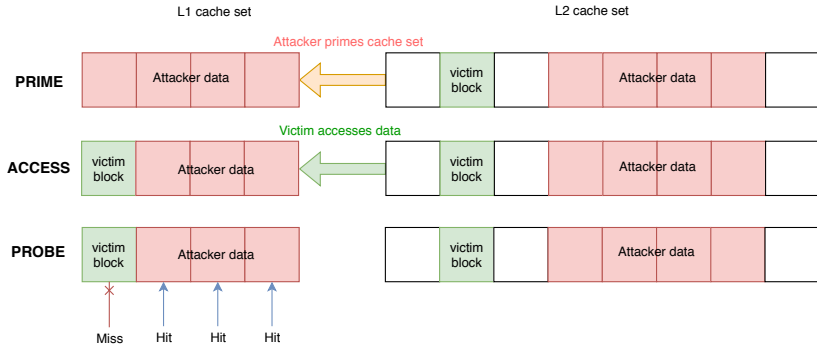4. Using both these information, we can pinpoint exact address accessed by victim.

Prime+Probe

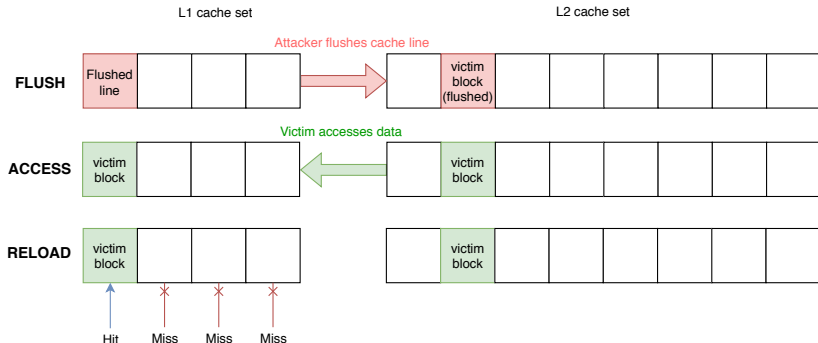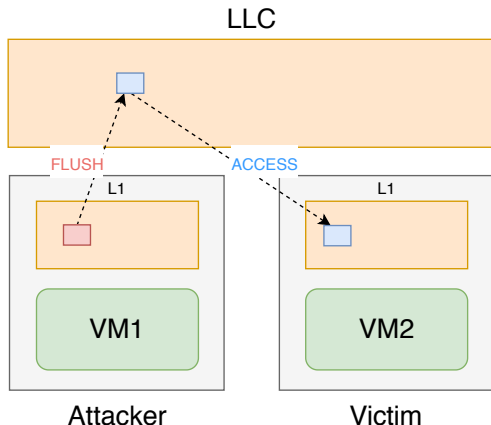
Figure: Working of Prime+Probe attack

Flush+Reload

Figure: Working of Flush+Reload attack

Flush+Reload on LLC

Figure: Flush+Reload Attack via LLC

- Virtual Machine environments are extensively used in cloud-computing
- VMs share a common host OS which perform page de-duplication.
- This allows VMs to share memory pages
- Enables hardware based side channels[4].

---
[4][Wait a minute! A fast, Cross-VM attack on AES - RAID'14]

Cache side channels on GPGPU

- Nvidia GPGPUs recently started to support concurrent kernel execution at SM level
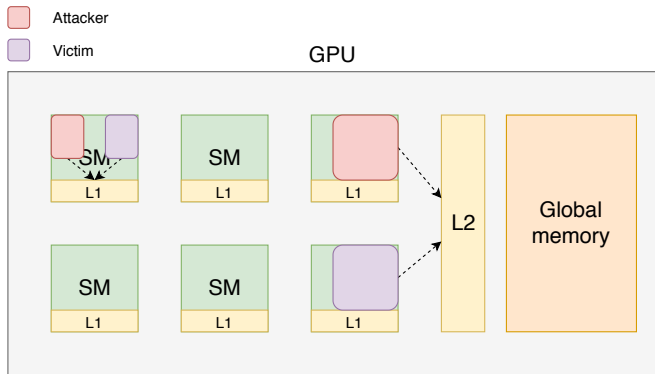- In this shared context, side channels can be exploited.

Figure: Memory layout of GPGPUs

GPGPU covert channel

Naghibijouybari et al[5] achieve communication speed of over 4Mbps using a combination of L1 cache contention and SFU contention as covert channels. They use the inherent parallelism in GPGPUs to open parallel communication channels on each SM.
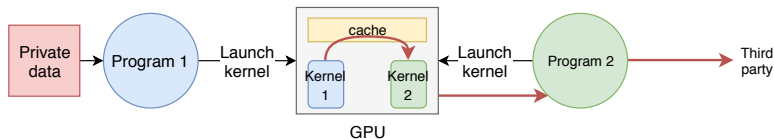

Figure: Covert channel implementation on GPGPU

[5][Constructing and characterizing covert channels on GPGPUs - MICRO-50]

Reverse engineer cache parameters

- Knowing cache parameters is necessary for mounting any kind of side channel attacks on caches.
- Implemented CPU benchmarking based on the GPU microbenchmarking by Wong et al[6].
- Stride access pattern over an array triggers a predictable number of cache-misses.

Listing 1: Generate array of given size with fixed stride pattern

```
size_t* array; // malloc beforehand
size_t t;

for (int i=0; i<array_size; i++) {
    t = i + STRIDE:
    if (t >= array_size) t %= STRIDE;
    array[i] = (size_t)array + sizeof(size_t)*t;
}
```

---

[6][Demystifying GPU microarchitecture through microbenchmark - ISPASS]

Reverse engineering cache parameters

For measuring timing of the array access, `rdtsc` instruction is used to get a reading of the Time Stamp Counter before and after accessing the array.

Listing 2: Timing measurement of stride access over the entire array

```
long start = __rdtsc();
size_t* next_ptr = &array[0];
for(int i=0; i<MAX_ITERS; i++) {
    next_ptr = *((size_t**)next_ptr);
}
long time = __rdtsc() - start;
```

Result Plots

Setup: GEM5 x86 atomic CPU. L1 data cache 1KB, 2-way, 64B line size.
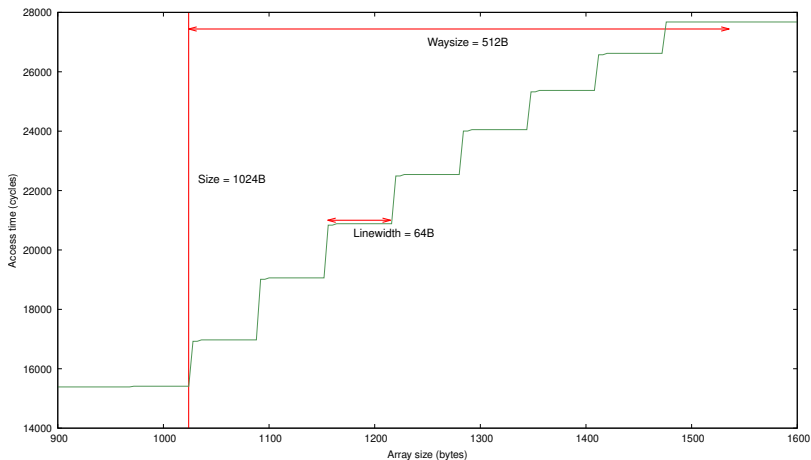

Figure: Latency vs Array Size plot

Result Plots

Setup: GEM5 x86 atomic CPU. L1 data cache 16KB, 4-way, 64B line size.
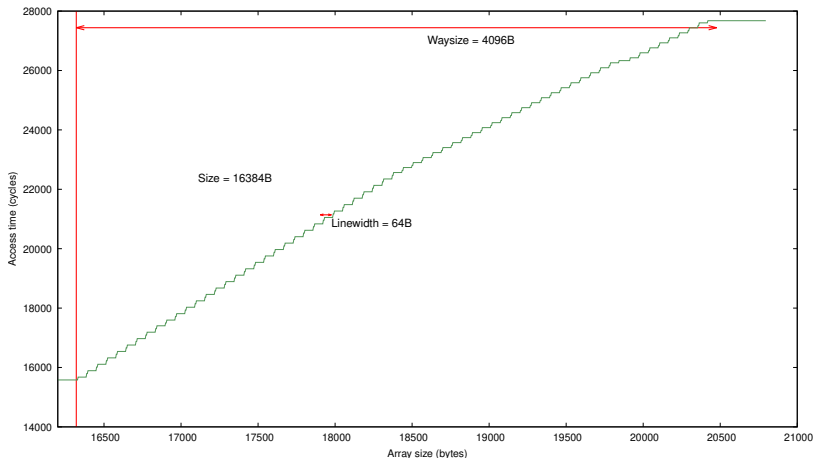

Figure: Latency vs Array Size plot

Result Plots

Setup: Intel Skylake x86_64 i5-6500. L1 data cache 32KB, 8-way, 64B line size.
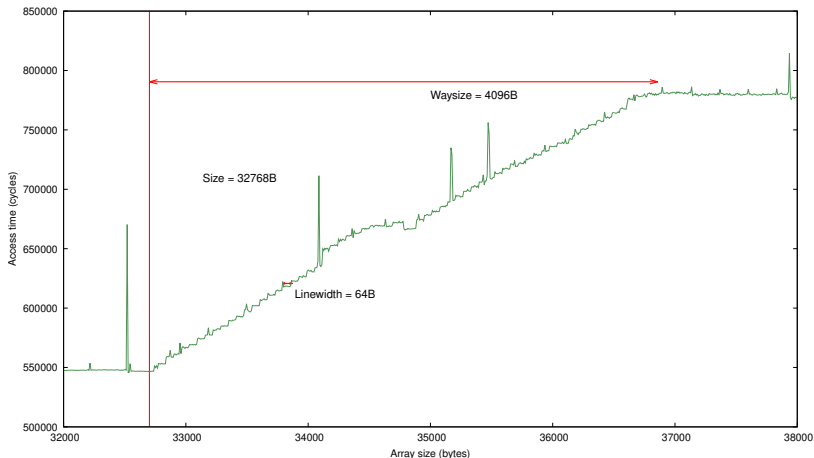

Figure: Latency vs Array Size plot

Mitigations against cache side channel

- Change the implementation of each encryption algorithm to avoid leaks.
- Only possible for known attacks and unavoidable for any software to not leave fingerprint in shared resources.
- Change hardware design of caches so that one process doesn't affect other processes via its cache accesses in a predictable way.
- Intentionally pollute cache to add noise in side channel measurements.

Improved cache designs
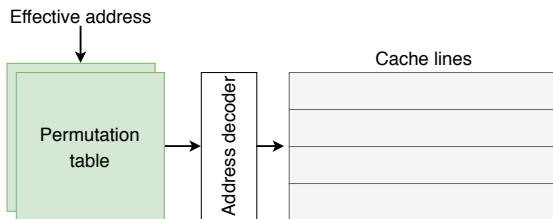

Figure: Partition-Locked cache - single cache line

Figure: Random-permutation cache

Context Sensitive Decoding

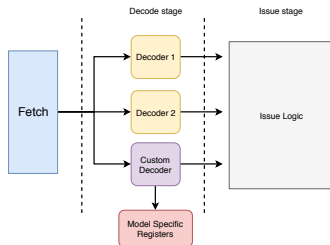Taram et al[7] implement a custom decoder which can insert decoy instructions to improve security.

Figure: Custom decoder for Context Sensitive Decoding

Decoy instructions pollute the cache by running decoy loads and disrupt attackers attempting side channel attacks on the cache.

---

[7][Mobilizing the Micro-Ops: Exploiting Context Sensitive Decoding for Security and Energy

Disruptive Prefetching

- Fuchs et al[8] introduce additional steps to the pre-fetchers to increase the randomness in the loaded memory address.
- Randomise the pattern sequence and degree to intentionally pollute the cache with unnecessary data.
- Slightly degrades the performance of non-malicious programs, but terribly disrupt side channel.
- Prime+Probe attack will not know whether the victim or the pre-fetcher evicted its block.
- Flush+Reload would get false cache hits which were not caused by the victim.

---

[8][Disruptive prefetching: impact on side-channel attacks and cache designs - SYSTOR'15]
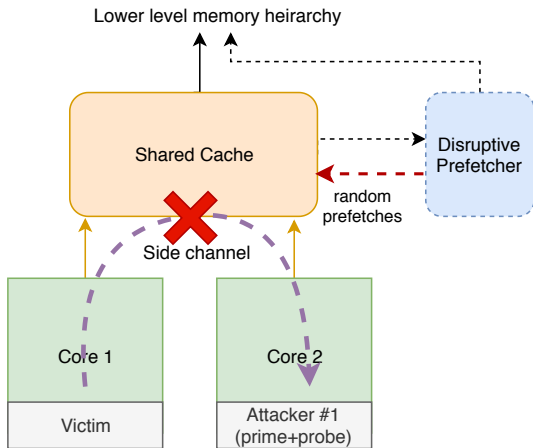
Disruptive Prefetching

Figure: Disruptive Prefetching

Disabling Prefetcher to avoid Cache Pollution

- Cache side-channels work by extracting information about cache accesses of victim process
- Any other process or hardware block which accesses the cache will increase noise in the side-channel
- We propose to disable the prefetcher by not allowing it to learn the stride patterns
- This can be done by running a third process in parallel which makes random loads at PC which alias with victim process' load PCs

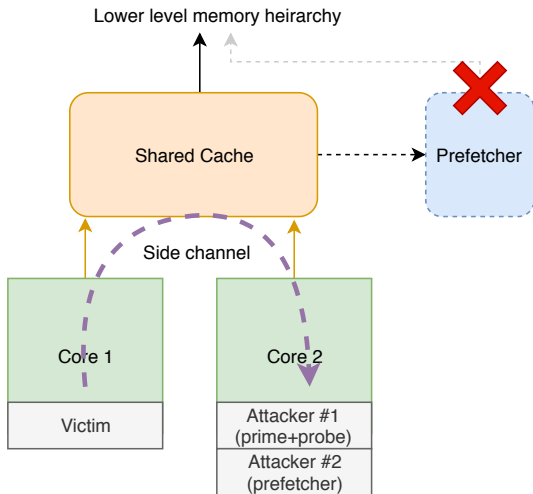# Disabling Prefetcher to avoid Cache Pollution

Figure: Prevent prefetcher from issuing memory accesses

Prefetcher attack implementation

- Place load instructions at multiple PC address to fill up the prefetcher table
- Randomize stride for every load address
- Restrict load address to single cache line by keeping stride less than 64 bytes
- Ensure cache miss every time by flushing before load using `clflush`
- Use `nop` instructions to align loads to desired PC address

- Drawback: targets whole prefetcher and not few entries relevant to victim. Makes the attack slower and allows prefetcher to train on victim's accesses.

Assembly for Full Attacker

Listing 3: Assembly showing load misses at different PCs

```
00000000000006ca <attack>:
 6ce:   8b 58 36          mov     0x36(rax),ebx
 6d1:   90                nop
 6d2:   0f ae 78 36       clflush 0x36(rax)
 6d6:   8b 58 08          mov     0x8(rax),ebx
 6d9:   90                nop
 6da:   0f ae 78 08       clflush 0x8(rax)
 6de:   8b 58 3f          mov     0x3f(rax),ebx
 6e1:   90                nop
 6e2:   0f ae 78 3f       clflush 0x3f(rax)
 6e6:   8b 58 38          mov     0x38(rax),ebx
 6e9:   90                nop
 6ea:   0f ae 78 38       clflush 0x38(rax)
 6ee:   8b 58 20          mov     0x20(rax),ebx
 6f1:   90                nop
```

Targeted Attacker

- Simulate victim process with random inputs and look at memory access patterns.
- Identify load addresses of the victim which have high probability of generating prefetches. These PCs will be target of the attack.
- Load instructions in the attacker are placed at PCs which alias with the target PCs of the victim. Remaining gaps are filled with nop.
- This attacker runs faster and is able to prevent the victim from successfully training the prefetcher.

Assembly for Targeted Attacker

Listing 4: Attacker targeting specific PC addresses

```
00000000000006ca <attack>:
 6da:   8b 58 0f               mov    0xf(rax),ebx
 6dd:   0f ae 78 0f            clflush 0xf(rax)
 6e1:   90                     nop
 6e2:   90                     nop
 6e3:   90                     nop
 6e4:   8b 58 3c               mov    0x3c(rax),ebx
 6e7:   0f ae 78 3c            clflush 0x3c(rax)
 6eb:   90                     nop
 6ec:   90                     nop
     <nop slide> ...
 6f7:   90                     nop
 6f8:   8b 58 2f               mov    0x2f(rax),ebx
 6fb:   0f ae 78 2f            clflush 0x2f(rax)
 6ff:   90                     nop
```

Simulator Setup

Victim process runs on core1 and attacker runs on core2. Number of prefetches issued are measured for every 1,000,000 instructions of victim process.

Simulator	gem5 X86
Cores	2
L1 Icache	32K 8-way
L1 Dcache	32K 8-way
L2 cache	256K 16-way shared between cores
L2 prefetcher	Stride 64-entry 4-way, confthresh 4

Results for benchmarks

The drawback of the full attacker is apparent here where it is not able to fully reduce the number of prefetches to 0.
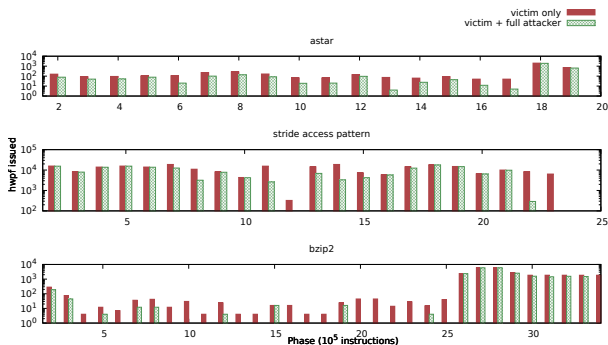

Figure: Number of prefetches issued on different benchmarks

Results for AES victim

The targeted attacker is able to successfully reduce the prefetches to nearly 0.
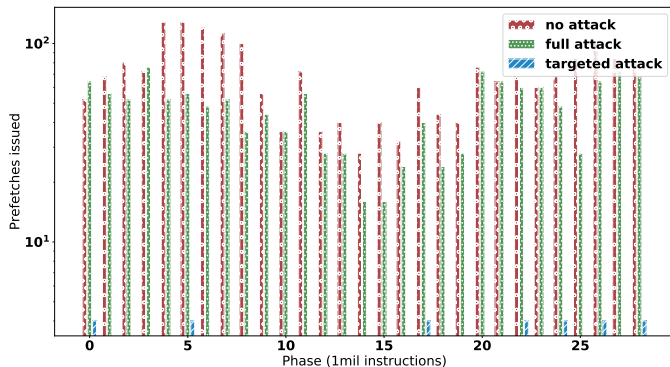

Figure: Number of prefetches issued under three execution conditions: no attacker, full attacker, targeted attacker

Results for AES victim

This figure explains how the attacker is able to reduce the number of prefetches by decreasing confidence of the prefetcher. It does not indicate how the targeted attacker is able to perform better than full attacker.
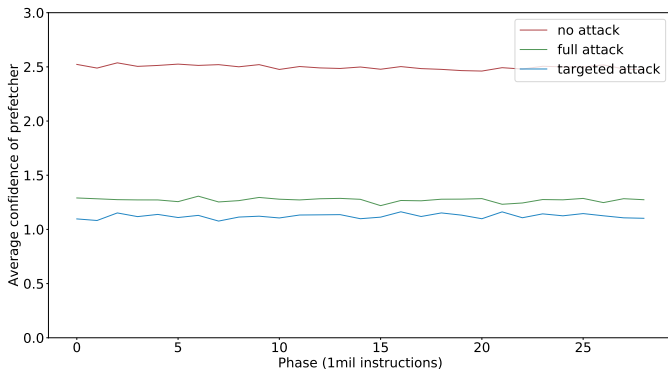

Figure: Average confidence of prefetcher entries

Results for AES victim

This figure shows the major reduction in prefetcher table hits for targeted
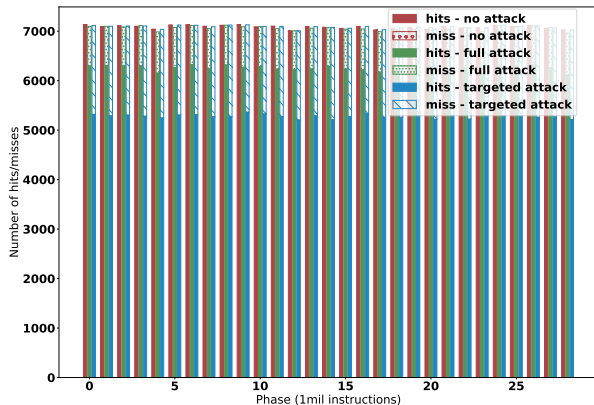attacker which explains why it can perform better.

Figure: Hits/Misses comparision for prefetcher table under three execution
conditions

Results for GHB Prefetcher

The stride prefetcher was replaced with a GHB prefetcher implementation using Delta Correlating Prediction Tables (DCPT).
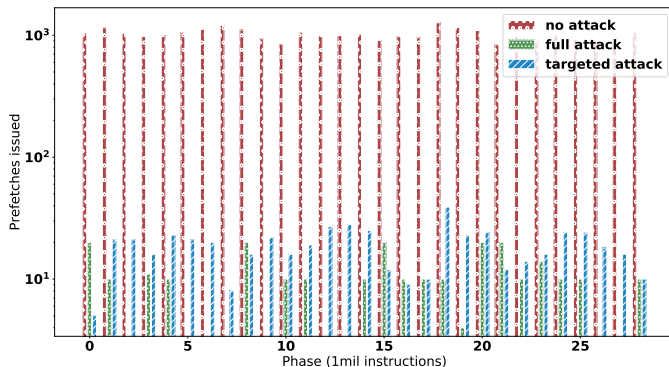

Figure: Prefetches issued with DCPT Prefetcher

Hypothetical ROB based Side Channel

- Reorder buffer is used by Out-of-Order cores for in-order retirement.
- In SMT cores, fairness policy ensures that roughly equal instructions are retired for all threads to maintain equal IPC.
- Simple implementation is to share retire pipeline width equally among threads.
- But when one thread is stalled, other thread will use full pipeline width to retire.
- IPC of other threads will see a slight increase.
- Data-dependent branch misses and cache misses can be infered from IPC information.
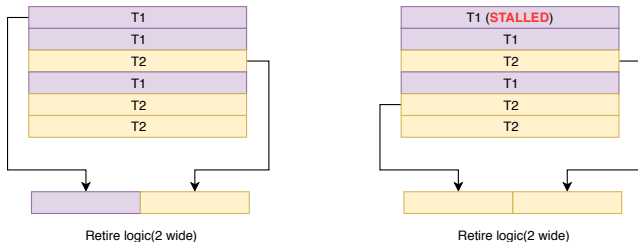
Hypothetical ROB based Side Channel

Figure: Reorder buffer for SMT. When T1 is stalled T2 retires twice as many instructions.

Conclusion

- Modern hardware features have introduced leakages in the processor which is a security concern.
- GPGPUs are being targeted as a new domain of security leaks.
- Hardware design needs to take into account these kind of security leaks.
- Attacks to disable prefetcher are possible and may help enhance side-channels

Future Scope

- Test prefetch attacker with Disruptive Prefetcher implementation.
- Create an attacker tailored for GHB instead of reusing stride attacker.
- Create a viable attack which can test and exploit ROB side-channel

The End

For Further Reading I

📄 Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel , D. Page, Department of Computer Science, University of Bristol

📄 Cache Missing for Fun and Profit, Colin Percival

📄 Cache-timing attacks on AES, D.J. Bernstien

📄 Side Channel Attacks, Chenglu Jin, Department of Electrical & Computer Engineering, University of Connecticut

📄 Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, Berk Sunar, Wait a minute! A fast, Cross-VM attack on AES

📄 D. Page, *Partitioned Cache Architecture as a Side-Channel Defence Mechanism*

📄 Z. Wang and R. B. Lee, New Cache Designs for Thwarting Software Cache-based Side Channel Attacks, ISCA'07

For Further Reading II

📄 Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In Proceedings of the 41st Annual International Symposium on Microarchitecture, MICRO 2008

📄 Adi Fuchs and Ruby B. Lee. 2015. Disruptive prefetching: impact on side-channel attacks and cache designs. In Proceedings of the 8th ACM International Systems and Storage Conference (SYSTOR '15)

📄 Hoda Naghibijouybari, Khaled N. Khasawneh, and Nael Abu-Ghazaleh. Constructing and characterizing covert channels on GPGPUs - MICRO'50

📄 Henry Wong, M. M. Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. 2010. Demystifying GPU microarchitecture through microbenchmark - ISPASS'10

For Further Reading III

📄 Mohammadkazem Taram, Ashish Venkat, Dean Tullsen, Mobilizing the Micro-Ops: Exploiting Context Sensitive Decoding for Security and Energy Efficiency - ISCA'18

📄 J. W. C. Fu, J. H. Patel and B. L. Janssens, *Stride Directed Prefetching In Scalar Processors*, [1992] Proceedings the 25th Annual International Symposium on Microarchitecture MICRO 25

📄 The gem5 Simulator. Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. May 2011, ACM SIGARCH Computer Architecture News.

📄 http://www.gem5.org/docs/html/stride_8cc_source.html

For Further Reading IV

https://github.com/gem5/gem5/blob/master/src/mem/cache/prefetch/del

Storage efficient hardware prefetching using delta-correlating prediction tables, M Grannaes, M Jahre, L Natvig - Journal of Instruction-Level Parallelism, 2011