White Paper

How the Meltdown and Spectre bugs work and what you can do to prevent a performance plummet

Programs that do a lot of I/O are likely to be the worst hit by the patches designed to fix the Meltdown and Spectre bugs. Here we explain why that is and what you can do about it.

Ellexus is a team of experts in I/O performance. The software company has a range of products to find and fix programs that will be affected by the patches for these attacks.

For the high performance computing (HPC) community, a drop in performance of 30% would mean disaster, but it doesn't need to be that bad if the proper approach is taken and our best practices are followed.

Contents

ELLEXUS LTD: THE I/O PROFILING COMPANY

Ellexus make tools to help you take control of the way big compute accesses big data. Our technology and best practices allow you to identify problem workloads, profile file and network I/O and optimise complex applications for high-performance computing and the cloud.

From managing single file dependencies to profiling an entire compute cluster with many thousands of cores, we've got the tools to make sure that your applications are fast, agile and ready to face the future.

For more information on Meltdown and Spectre, as well as the available patches, visit:

https://googleprojectzero.blogspot.co.uk/2018/01/reading-privileged-memory-with-side.html

Meltdown and Spectre: The facts

How do the Meltdown and Spectre vulnerabilities work?

Meltdown exploits caching and speculative execution to allow programs to read information that should only be accessible to the operating system. Spectre allows programs to read data from each other's memory when both programs are running on a single processor. This is a problem for systems with multiple users on one machine, but can also be an issue for single user systems. Do you want an arbitrary website running javascript to have your bank details, email and social media?

Processor performance: Speculation

Although programs are written in relatively high-level languages these days, they all turn into a series of machine instructions before execution. Older processors used to execute these one at a time, but the performance increases we've seen over the last decade depend on clever optimisations such as speculative execution or out of order execution. Speculative execution is when the path of the program is not yet known, but the processor makes a good guess (a branch prediction) and starts executing the next bit of code early. When it gets it right, it will have executed two pieces of code at once and is ahead. When it gets it wrong, the speculative results are discarded and the processor starts to execute the right path or code branch instead. A good step-by-step guide to branch prediction can be found on the Raspberry Pi blog: https://www.raspberrypi.org/blog/why-raspberry-pi-isnt-vulnerable-to-spectre-or-meltdown/

Memory performance: Caching

Most of the optimisations over the last few years have been to make up for processors increasing in performance far more quickly than memory access. In fact, it's not just memory; all layers of storage have got proportionally slower as compute clock cycles get faster and more productive. 25 years ago, computers could access memory several times for each instruction executed, but laptops today can now execute many thousands of instructions for every memory access even though the memory has got a lot faster. As each processor generation passes, it takes more and more clock cycles to fetch data from networked storage, local disk or memory.

Caching does a lot to relieve this performance gap by storing data at each layer in a small amount of more expensive, faster storage on the CPU. Data that has been recently used is stored there as well as data that might be used soon. This is known as pre-fetching. Both Meltdown and Spectre exploit caching by timing how long it takes to access data from memory so you can tell if it came from RAM or the cache. By timing how long it takes to access the memory, the attacker can make deductions about whether the data came from memory or the cache.

How the Spectre and Meltdown attacks use caching and speculation

In theory, accessing restricted memory during speculative execution isn't a problem because either the code won't be reached by the true path of executing and the results discarded, or a fault will be raised and the results of any subsequent calculation discarded. In practice, the acceleration of memory accesses using a cache means that there could be side effects to the speculative execution that are not removed when the speculative results are discarded. Spectre and Meltdown are known as side channel attacks because they use a vulnerability in the physical implementation of a theoretically secure system.

Normally, when a restricted area of memory is accessed a fault occurs and execution is halted. However, during speculative execution faults are delayed and execution may continue past the restricted memory access. If the next instructions fetch non-restricted data based on the results of the restricted read then the cache will have changed and the contents will depend on restricted data. When the fault is raised, the results of the speculative execution will be discarded, but the changes to the cache won't be undone. One can then use the time taken to access the non-restricted memory to determine what is already in the cache. From here you can deduce something about the restricted data that was read during the speculative execution. This allows data to be read out bit by bit from forbidden areas of memory that could contain passwords and other sensitive operating system data.

What is the difference between Meltdown and Spectre?

Meltdown works by running an attack program that reads privileged data from kernel address space – the data that should only be accessible to the operating system. Spectre works by tricking a victim program into reading data it wouldn't normally access and leaking data to a second attack program running on the same machine. Meltdown is simpler to implement, but can be prevented by the KAISER patch. Spectre is CPU specific, much harder to implement, but currently has no patch.

Protecting against the Meltdown attacks

The attacks are not easy to implement, but they are also hard to protect against and leave no trace. At this stage no-one can be sure who has used them and how successful they were.

In time, code will appear that allows non-expert users to exploit the vulnerabilities and eventually that will be known to virus checkers, but at the moment we are operating in the dark and already code that exploits the attacks are available online. The fix for Meltdown is based on a patch set called KAISER (https://lwn.net/Articles/738975/) and Windows has already released a patch of that name. The solution is to prevent any access of the kernel memory while a program is running by keeping user and kernel memory completely separate. Instead of using a single lookup table to store where data is in memory, two tables are set up and maintained: one for the user program that contains only locations for user memory, and one for the operating system with locations of both user and kernel memory. Unfortunately, this means that switching between user and kernel modes becomes more expensive because the memory lookup table has its own cache (TLB) that needs to be flushed each time.

The impact on performance

Why does the patch affect I/O?

The fix sounds simple, but it has performance impacts. For example, if a program needs to access data on disk it will need to make a system call to the operating system. The processor must switch from user mode into kernel mode and it must now flush the memory lookup table cache each time. Nearly all system calls with be affected, including I/O calls as well as other calls to the operating system often used indirectly in I/O intensive programs such as calls to get the time of day. This means that I/O intensive applications such as those using a database, e-commerce application or HPC programs will be worst affected.

Worst I/O patterns for performance

The programs that see the biggest avoidable performance impact will be those that do more I/O calls that necessary. Some reads and writes are buffered in user space and passed onto the operating system, but plenty of programs do small I/O that makes a system call and an expensive memory context switch each time (see case study). Programs do small I/O due to legacy code, third party libraries or a misunderstanding over the implementation of the I/O calls. Unless you have profiled the I/O there is no way of knowing.

Meta-data operations can also be expensive. These include opening and closing files or checking file existence, permissions or other attributes such as size. Each will cause a context switch and many can be avoided.

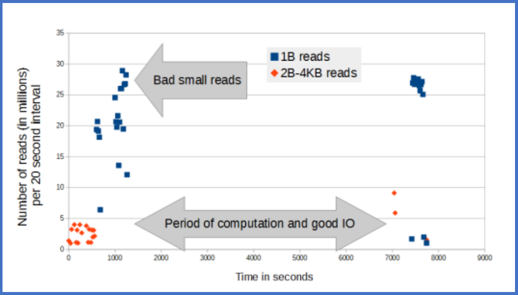
What does this mean for HPC and other I/O intensive applications?

The HPC sector includes scientific applications such as weather forecasts, Formula 1 car design, genome mapping and oil and gas exploration. All these applications need a lot of computation on a lot of data. The impact of getting that wrong is huge. It's very easy for applications sharing datasets to overload the file system and prevent other applications from working, but bad I/O can also affect each program in isolation, even before the patches for the attacks make that worse.

Estimates of the performance impact of applying the KAISER patches have been quoted at about 10-30% for I/O intensive applications. As I/O experts, we think this sounds about right. In the short term, the Meltdown fix may actually improve performance for the file system by slowing down the compute of I/O intensive applications, but this is at the expense of a drop in application throughput. If your applications are affected by a slow down of 30% this will be noticeable straight away.

Case study: Small I/O discovered in a popular genome pipeline

Ellexus profiled one of the genome pipelines at the Sanger Institute with Mistral to look for such I/O patterns. The pipeline had been optimised in some areas, but our profiling showed that there was still potential for it to be improved.



The figure above shows the alerts generated when there were more than a million small reads in a 20s period. At the start there were up to a million 1 byte reads per second. These small reads harm the computational performance and create sub-optimal I/O patterns on the shared storage.

Can you afford to lose a third of your compute real estate?

The longer-term solution is to find and fix the applications affected because this will make both the compute and the storage faster. The patterns that will be most affected by the Meltdown patch have a lot in common with some of the patterns that harm shared storage and overall application performance. By addressing this issue now, many applications will be faster than before the patch was applied, but only if the I/O patterns are improved.

How to act now

Time to profile I/O

Ellexus provides a suite of tools to locate bottlenecks and applications with bad I/O on large distributed systems, cloud infrastructure and super computer clusters. Once applications with bad I/O patterns have been located, our tools will indicate the potential performance increases as well as pointers on how to achieve them. Often the optimisation is as simple as changing an environment variable, changing a single line in a script (see case study) or changing a simple I/O call to read more than one byte at a time. In some cases, the candidates for optimisation will be obvious – a workflow that clearly stresses the file system every time it is run, for example, or one that runs for significantly longer than a typical task.

Three steps to improve performance

In others it may be necessary to perform an initial high-level analysis of each job. Follow three steps to optimise application I/O and mitigate the impact of the KAISER patch:

1. Profile all your applications with Mistral to look for the worst I/O patterns.

Mistral, our I/O profiling tool, is lightweight enough to run at scale. In this case Mistral would be set up to record relatively detailed information on the type of I/O that workflows are performing over time. It would look for factors such as how many meta data operations are being performed, the number of small I/O and so on.

2. Deal with the worst applications, delving into detail with Breeze

Once the candidate workflows have been identified they can be analysed in detail with Breeze. As a first step, the Breeze trace can be run through our Healthcheck tool that identifies common issues such as an application that has a high ratio of file opens to writes or a badly configured $PATH causing the file system to be trawled every time a workflow uses "grep".

3. Put in place longer-term I/O quality assurance

Implement the Ellexus tools across your systems to get the most from the compute and storage and to prevent problems reoccurring.

By following these simple steps and our best practices guidance it is easy to find and fix the biggest issues quickly and give you more time to optimise for the best performance possible.

Case study: Remove unnecessary system calls with Breeze

Ahmed Ismail, software engineer at software company Mentor Graphics, on how he optimised his software build

"Breeze showed me that I just needed to do the below change in a few places:

```
MODELTECH_DIR := $(shell cd
$(MTI_ROOT); pwd)/modeltech
```

"After rerunning I observed about 87% improvement in incremental build (16 sec to 2 sec)!"

Get in touch to learn more about how the I/O profiling tools from Ellexus can improve system performance by up to 30%.

info@ellexus.com