Intel Cache Monitoring: Present and Future

About this document

This document represents the result of in investigation on whether it would be possible to more extensively exploit the Platform Shared Resource Monitoring (PSR) capabilities of recent Intel x86 server chips. Examples of such features are the Cache Monitoring Technology (CMT) and the Memory Bandwidth Monitoring (MBM).

More specifically, it focuses on Cache Monitoring Technology, support for which has recently been introduced in Xen by Intel, trying to figure out whether it can be used for high level load balancing, such as libxl automatic domain placement, and/or within Xen vCPU scheduler(s).

Note that, although the document only speaks about CMT, most of the considerations apply (or can easily be extended) to MBM as well.

The fact that, currently, support is provided for monitoring L3 cache only, somewhat limits the benefits of more extensively exploiting such technology, which is exactly the purpose here. Nevertheless, some improvements are possible already, and if at some point support for monitoring other cache layers will be available, this can be the basic building block for taking advantage of that too.

Terminology

In the remainder of the document, the term core, processor and (physical) CPU (abbreviated with pCPU), are used interchangeably, for referring to a logical processor. So, for instance, a server with 2 sockets, each one containing 4 cores and with hyperthreading will be referred to as a system with 16 pCPUs.

The Cache Monitoring Technology (CMT)

Cache Monitoring Technology is about the hardware making cache utilization information available to the Operating System (OS) or Hypervisor (VMM), so that it can make better decisions on workload scheduling.

Official documentation from Intel about CMT is available at the following URLs:

- Benefits of Cache Monitoring
- Intel CMT: Software Visible Interfaces
- Intel CMT: Usage Models and Data
- Intel CMT: Software Support and Tools

The family of chips that first includes CMT is this:

- https://software.intel.com/en-us/articles/intel-xeon-e5-2600-v3-product-family

Developers' documentation is available, as usual, in Intel's SDM, volume 3B, section 17.14

Materials about how CMT is currently supported in Xen can be found here and here.

Current status

Intel itself did the work of upstreaming CMT support for Xen. That happened by mean of the following changesets:

- *x86: expose CMT L3 event mask to user space* : 877eda3223161b995feacce8d2356ced1f627fa8
- *tools: CMDs and APIs for Cache Monitoring Technology* : 747187995dd8cb28dcac1db8851d60e54f85f8e4
- *xsm: add CMT related xsm policies* : edc3103ef384277d05a2d4a1f3aebd555add8d11
- *x86: add CMT related MSRs in allowed list* : 758b3b4ac2c7967d80c952da943a1ebddc66d2a2
- *x86: enable CMT for each domain RMID* : 494005637de52e52e228c04d170497b2e6950b53
- *x86: collect global CMT information* : 78ec83170a25b7b7cfd9b5f0324bacdb8bed10bf
- *x86: dynamically attach/detach CMT service for a guest* : c80c2b4bf07212485a9dcb27134f659c741155f5
- *x86: detect and initialize Cache Monitoring Technology feature* : 021871770023700a30aa7e196cf7355b1ea4c075
- *libxc: provide interface for generic resource access* : fc265934d83be3d6da2647dce470424170fb96e9
- *xsm: add resource operation related xsm policy* : 2a5e086e0bd6729b4a25536b9f978dedf3be52de
- *x86: add generic resource (e.g. MSR) access hypercall* : 443035c40ab6a0566133a55090532740c52d61d3

Quickly summarizing, it is possible, from Dom0, to start and stop monitoring the L3 cache occupancy of a domain, by doing something like this:

```
[root@redbrick ~]# xl psr-cmt-attach 7
```

Results can be seen as follows:

```
[root@redbrick ~]# xl psr-cmt-show cache_occupancy
Total RMID: 71
Name                                        ID        Socket 0        Socket 1        Socke
Total L3 Cache Size                                   46080 KB        46080 KB        46080
wheezy64                                     7          432 KB            0 KB         2016
```

What happens in Xen is that an RMID is assigned to the domain, and that RMID is loaded in a specific register, during a context switch, when one of the domain's vCPU starts executing. Then, when such a vCPU is switched out, another RMID is loaded, and that could be the RMID assigned to another domain being monitored or the special RMID 0, used to represent the 'nothing to monitor' situation.

Main limitations of current approach

The existing CMT support in Xen suffers from some limitations, some imputable to hardware, some to software, some to both.

Per-domain monitoring

Xen scheduler's schedule vCPUs, not domains. So, if wanting to use results coming out of CMT for scheduling, the monitoring would have to happen on a per-vCPU basis, rather than on a per-domain one.

This is mostly matter of how CMT support has been implemented, and that can be changed toward per-vCPU monitoring pretty easily, at least from a purely software perspective. That would mean, potentially, more overhead (RMID being switched more often than now, and each RMID switch means writing a Model Specific Register). The biggest hurdle toward achieving per-vCPU CMT support, however, most likely is the fact that the number of available RMID is limited (see below).

RMIDs are limited

And that is an hard limit, imposed, per the best of author's understanding, by the underlying hardware.

For example, a test/development box at hand, with an Intel(R) Xeon(R) CPU E7-8890 v3 as CPU, had 144 pCPUs and only 71 RMIDs available.

This means, for instance, that, if we would like to turn CMT support into being to per-vCPU, we will not be able to monitor more vCPUs than 1/2 the number of the host's pCPUs.

Only L3 monitoring

For now, it is only possible to monitor L3 cache occupancy. That is, as far as the author of this document understands, a limitation currently imposed by hardware. However, making it possible to monitor other cache layer, e.g., L2, is something Intel is planning to introduce, still to the best of the document's author's knowledge.

In any case, the fact that CMT is limited to L3 for now, makes it less appealing than one may think to be used from within the Xen scheduler, to make cache aware scheduling decisions. In fact, L3 is, in most architectures, the so called LLC (Last Level Cache), shared by all the cores of a socket. Furthermore, a socket, most of the times, coincides with a NUMA node.

Therefore, although knowing about L3 cache occupancy enables making better informed decisions when it comes at load balancing among NUMA nodes, moving one (or more) vCPUs of a domain to a different socket will most likely have some rather bad implications, for instance if the domain has it's memory on the NUMA node where it is running (and both the domain creation logic and the Xen scheduler have heuristics in place for make this happen!). So, although more cache efficient, this would lead to a lot of remote memory traffic, which is certainly undesirable.

Not all the domains are always allocated and configured in such a way to achieve as much locality of memory accesses as possible, so, theoretically, there is some room for this feature to be useful. However, practically speaking, until we will have L2 monitoring the additional complexity and the overhead introduced would most likely outweighs the benefits.

RMIDs reuse

It is unclear to the author of this document what behavior would be considered to be the proper one when an RMID is "reused", or "recycled". What that means is what should happen if a domain is assigned an RMID and then, at some point, the domain is detached from CMT, so the RMID is freed and, when another domain is attached, that same RMID is used.

This is exactly what happens in the current implementation. Result looks as follows:

```
[root@redbrick ~]# xl psr-cmt-attach 0
[root@redbrick ~]# xl psr-cmt-attach 1
Total RMID: 71
Name                                        ID        Socket 0        Socket 1        Socket
Total L3 Cache Size                                   46080 KB        46080 KB        46080
Domain-0                                     0         6768 KB           0 KB            0
wheezy64                                     1            0 KB         144 KB          144
```

Let's assume that RMID 1 (RMID 0 is reserved) is used for Domain-0 and RMID 2 is used for wheezy64. Then:

```
[root@redbrick ~]# xl psr-cmt-detach 0
[root@redbrick ~]# xl psr-cmt-detach 1
```

So now both RMID 1 and 2 are free to be reused. Now, let's issue the following commands:

```
[root@redbrick ~]# xl psr-cmt-attach 1
[root@redbrick ~]# xl psr-cmt-attach 0
```

Which means that RMID 1 is now assigned to wheezy64, and RMID 2 is given to Domain-0. Here's the effect:

```
[root@redbrick ~]# xl psr-cmt-show cache_occupancy
Total RMID: 71
Name                                    ID        Socket 0        Socket 1        Socke
Total L3 Cache Size                               46080 KB        46080 KB        46080
Domain-0                                 0          216 KB          144 KB          144
wheezy64                                 1         7416 KB            0 KB         1872
```

It looks quite likely that the 144KB occupancy on sockets 1 and 2, now being accounted to Domain-0, is really what has been allocated by domain wheezy64, before the RMID "switch". The same applies to the 7416KB on socket 0 now accounted to wheezy64, i.e., most of this is not accurate and was allocated there by Domain-0.

This is only a simple example, others have been performed, restricting the affinity of the various domains involved in order to control on what socket cache load were to be expected, and all confirm the above reasoning.

It is rather easy to appreciate that any kind of 'flushing' mechanism, to be triggered when reusing an RMID (if anything like that even exists!) would impact system performance (e.g., it is not an option in hot paths), but the situation outlined above needs to be fixed, before the mechanism could be considered usable and reliable enough to do anything on top of it.

RMID association asynchronousness

Associating an RMID to a domain, right now, happens upon toolstack request, from Dom0, and it will be effective as soon as the next context switches involving the vCPUs of the domain happen.

The worst case scenario is for an (new) RMID being assigned to a domain an instant after all its vCPU started running on some pCPUs. After a while (e.g., at the end of their timeslice), they will be de-scheduled and context switched in back when their turn come. It is only during this last context switch that we load the RMID in the special register, i.e., we have lost a full instance of execution and hence of cache load monitoring. This is not a too big issue for the current use case, but it might be when thinking about using CMT from within the Xen scheduler.

Detaching case is similar (actually, probably worse). In fact, let's assume that just an instant after all the vCPUs of a domain with an RMID attached started executing, the domain is detached from CMT and another domain is attached, using the RMID which just got freed. What happens is that the new domain will be accounted for the cache load generated by a full instance of execution of the vCPUs of the old domain. Again, this is probably tolerable for the current use case, but cause problems if wanting to use the CMT feature more extensively.

Potential improvements and/or alternative approaches

Per-vCPU cache monitoring

This means being able to tell how much of the L3 is being used by each vCPU. Monitoring the cache occupancy of a specific domain, would still be possible, just by summing up the contributions from all the domain's vCPUs.

Benefits / Drawbacks From the scheduler perspective, this would be best possible situation, provided the information is precise enough, and is available for all (or at least a relevant, for some definition of relevant, subset of) vCPUs.

For instance, Credit1 already includes a function called __csched_vcpu_is_cache_hot(). That is called when, during load balancing, we consider whether or not to "steal" a vCPU from a remote pCPU. That may be good for load balancing, nevertheless, we want to leave where they are the ones that have most of their data in the remote pCPU cache. Such function right now tries to infer whether a vCPU is really "cache hot" by looking at how long ago it run; with something like per-vCPU CMT, it could use actual cache occupancy samples.

However, implementing per-vCPU cache monitoring would entail the following:

- on large servers, running large (and/or many) guests, we may run out of RMIDs
- more frequent RMID switching on the pCPUs, which (it being an MSR write) means increased overhead
- rather frequent sampling of the cache occupancy value, which (it being an MSR read) means increased overhead

Moreover, right now that only L3 monitoring is supported, the information would not be useful for deciding whether or not to migrate a vCPU between pCPUs belonging to the same socket (in general, between pCPUs insisting on the same LLC).

It therefore seems that, enabling per-vCPU monitoring and using the information it could provide from withing the scheduler would require overcoming most of the current (both hardware and software) limitations of CMT support.

Per-pCPU cache monitoring

This means being able to tell how much of the L3 is being used by activities running on each core that insists it. These activities are typically vCPUs running on the various logical processors but, depending on the actual implementation, it may or not include the effects on cache occupancy of hypervisor code execution.

Implementing this would require reserving an RMID for each logical processor, and keep it in the appropriate special register until the activity we are interested in monitoring is running on it. Again, due to the “L3 only” limitation, this is not so useful to know when inside the scheduler. However, it could be a sensible and useful information for in-toolstack(s) balancing and placement algorithms, as well as for the user, to decide how to manually place domains and set o tweak things like vCPUs hard and soft affinities.

Benefits / Drawbacks How much of the LLC is being used by each core is a fair measure of the load on a core. As such, it can be reported all the way up to the user, as we currently do with per-domain cache occupancy.

If such information is available for all cores, by summing up all their contributions, and subtracting it from the size of such LLC, we obtain how much free space we have in each L3. This is an information that, for example, the automatic NUMA placement algorithm that we have in libxl can leverage, to have a better idea of on what pCPUs/NUMA node would be best to place a new domain.

In this form of “how much free cache there is on each L3 (on each socket)”, it is possible for the Xen scheduler too to make a somewhat good use of it, although, of course, only when the decisions involve more than one socket. Also, the overhead being introduced by a similar CMT configuration, is certainly not concerning.

If the association of RMIDs to the cores has to be dynamic, i.e., if it has to be possible for an user to attach, detach and re-attach a core to CMT, the issue described in the previous section about RMID reuse is very much relevant for this use case too. If it is static (e.g., established at boot and never changed), that is less of a concern.

If the information about the amount of free L3 has to be used from within the scheduler, that requires some more MSR manipulations, as we not only need to load the RMIDs, we also need to sample the cache occupancy value from time to time. That still looks feasible, but it is worth mentioning and considering (e.g., what periodicity should be used, etc.).

The biggest drawback is probably the fact that, if enabling this per-core CMT setup, monitoring specific domains, i.e., what we support right now, would be really difficult. In principle, it just requires that:

- when a core is executing the vCPU of a non-monitored domain, the RMID of the core is used
- when the vCPU of a monitored domain is context switched in, the RMID of the domain is used
- the cache load generated while the RMID of the domain is used must be recorded and stored or accumulated somewhere

- when the reading the cache occupancy of a core, the accumulated value of cache occupancy of domains that run on the core itself should be taken into account
- when a domain stop being monitored, its cache occupancy value should be somehow incorporated back in the core's cache occupancy
- when a domain is destroyed, its cache occupancy value should be discarded.

Currently, point 3 can't be achieved, due to the lack of precision of the measurements, deriving from the asynchronous (with respect to scheduling) nature of RMIDs association, as described in previous sections. Points below it, are also quite difficult to implement, at least as far as reporting the value up to toolstack is concerned, given how cache occupancy values read is implemented (i.e., as a generic 'MSR read' platform hypercall).

Therefore, if wanting to go for the per-core CMT configuration, sane alternatives seem to be to just disallow specific domain monitoring, to avoid it screwing up the per-core monitoring, or the exact opposite. That is just allow it and let it temporarily (i.e., as long as a domain is monitored plus, likely, some settle time after that) screw up the per-core monitoring, of course warning the user about it.

Conclusion

In extreme summary, the outcome of this investigation is that CMT (and other PSR provided facilities by Intel) is a nice to have and useful feature, with the potential of being much more useful, but that depends on the evolution of both hardware and software support for it.

Implementing per-core CMT setup looks to be a reasonably low hanging fruit, although some design decisions needs to be taken. For this reason, an RFC patch series drafting an implementation of that is provided, together with this document.

Please, see the changelogs of the individual patches and find there more opportunities to discuss the design of such a setup, if that is something considered worth having in Xen.

Find the series in this git branch also:

- git://xenbits.xen.org/people/dariof/xen.git wip/sched/icachemon