Authentication of Sub-NUMA Clustering effect on Intel Skylake for Memory Latency and Bandwidth

Srikanta Kumar Mohapatra [a], Sankararao Majji [b], Prathipati Ratna Kumar [c], Ravula Arun Kumar [d] and Santoshachandra Rao Karanam [e]

[a] Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, srikanta.mohapatra@chitkara.edu.in
[b] Assistant Professor, GRIET, Hyderabad, sankar2367@gmail.com
[c] Assistant Professor, CSE Department, Koneru Lakshmaiah Education Foundation, Hyderabad, rk30111972@klh.edu.in
[d] Assistant Professor, Vardhaman college of Engineering, Hyderabad India, arunravula12@gmail.com
[e] Assistant Professor, Department of IT, Anurag University, Hyderabad, kschandra.rao@gmail.com

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 10 May 2021

Abstract: The search efficiency of in-memory databases depends significantly on how quickly DRAM data can be obtained. With an increasing multiple core on processors, it is more difficult to satisfy the requirement that all attached DRAM on a processor is equally accessible to any core. Intel has therefore implemented a Sub-NUMA Clustering (SNC) mode on Skylake, which subdivides cores and memories into various sub for enhanced core-to-memory access within each processor sub-domain. Similar modes are given by other models. When an in-memory database shares data between staff on cores in separate sub-domains, the use of SNC creates problems in how to manage database workloads among domains. In this research, we verify the effect of SNC specifically on Intel Skylake on memory latency and bandwidth. We conclude that two similarly broad analytical workloads are focused on different and fully independent sub-domains just up to 3 percent will improve query throughput and be totally isolated from each other. Often, as memory bandwidth is split into sub-domains rather than aggregated across the entire processor, bandwidth-sensitive analytical workloads significantly reduce application performance if data is not split equally among sub-domains.

1. Introduction

Data output depends heavily on how easily vast volumes of data can be accessed by the database. SAP HANA is a database built by SAP among many various vendors of relational SQL databases, driven by an in-memory approach to make interprocess communication fast. Full datasets may remain in memory by compressing the data and utilizing machines with large DRAM memories, preventing access to sluggish disks. As in conventional databases, columns are processed column-wise, rather than row-wise, allowing aggregation simpler when co-locating similar data. For analytical workloads that typically deal with agglomerations in a columnar format, these methods are advantageous. The main selling point for SAP HANA is quick metrics.

The efficiency of an in-memory database is highly dependent on how easily the database can reach its DRAM data. Access time will depend not only on the memory type, as well as on the encryption technique used in the processor, as well as how far a request for memory access has to travel to the targeted memory within the processor. Processor manufacturers are increasingly selling more cores to processors, and memories are becoming larger and cheaper. Hardware has become more cluster-oriented with co-located cores and memories, operating either in isolate or collaboration, rather than making one central processor attached to one memory area. Just because of that, it is becoming extremely difficult to provide equally quick access to data from any core in a processor in all memory regions. In order to reveal these variants to applications, Intel implemented an alternative mode coined by Cluster-on-Die (COD) in its Haswell generation that was later replaced in the Skylake generation by Sub-NUMA Clustering (SNC)[1]. Certain vendors considered similar technologies [2] [3]. In order to make core-to-memory access faster within each sub-domain of the processor, such functions subdivide the processor into sub-domains. There is still open connectivity across multiple domains, but at a potentially higher latency. In specific, sub-domains are interesting as the latest trend suggests a further rise in the number of processor cores, which eventually makes access times more dependent on the location of a core within the processor.
For implementations where processes use their own memory rather than exchanging it among them, technologies such as COD/SNC and such are intuitively appropriate. This is generally the case for Virtual Machines (VMs), where VMs seldom access the memory of each other. VMware is one forum for VMs that can take advantage of the COD mode of Intel [4]. In-memory databases, by comparison, use a memory interface region shared between query tasks performed simultaneously. As in VMs, execution and memory cannot be separated from each other

as quickly. However, databases such as SAP HANA use certain task allocation methods on multi-processor computers, although functions checking a table segment are preferably performed on the processor to which the column's memory DRAM unit is connected for improved location. For load balancing purposes, however, it may be permitted to operate elsewhere. This makes it quicker to search memory and SAP HANA scales easier for more processors on a machine [5].

2. RELATED WORK:

Earlier studies on task theft have studied remote task stealing on multi-machine setups in a hierarchical way for generic workloads. The location of the task to be stolen is represented in "HotSLAW"[6] using a locality hierarchy where the stealer first tries to loot from the local area and travels up the hierarchy to steal tasks from a less local zone. Locality hierarchy levels may be a shared L2 cache, a NUMA domain, another processor, or a separate server. The stealer randomly selects victims for each hierarchy to loot from and only progresses upwards in the hierarchy after all random attempts have failed. HotSLAW uses Partitioned Global Address Space (PGAS) to give remote memory on any server access to a mission. Memory in PGAS is kept consistent across software-use servers. Our device varies slightly in that we only use processors on the same computer.

Research has also been done to try to use Remote Direct Memory Access (RDMA)[7] to make databases spread through several machines run faster. Some technologies use standard IP-based underlying networking, and some use special protocols such as InfiniBand. These types of high-performance hardware for networking have recently become more cost-competitive. At levels close to what is achieved in memory bandwidth inside a computer, the networking hardware may provide bandwidth between machines. RDMA will also reduce latency to memory on a remote computer by not needing to go through the CPU or OS during memory accesses across computers. It is shown in [8] that the transfer of 1 KB of data using RDMA will end at about the same time as a request for local memory. It is noted, however, that it is difficult to render RDMA as fast as local memory access on smaller data requests such as hash table lookups, as the network latency, and not the bandwidth, is the most dominant factor in this case.

Psaroudakis has conducted previous column investigations and placement of scanning tasks in SAP HANA on multi-processor configurations [9]. A higher Processor load is achieved in these simulations by increasing the number of rows selected by question constants. In the more CPU-intensive process of materialization, this increases work. Result is a reduction in throughput on a balanced workload with highly selective predicates when these data-intensive tasks are permitted to be captured by remote processors. Nevertheless, on less selective predicates, thus increasingly CPU-intensive, robbery offered advantages. The simulations were only performed with one processor representing each NUMA domain. When using sub-domains using COD and SNC, we expand this by looking at the effect.

Depending on the manufacturer, AMD's EPYC and potentially ARM's Neoverse processor are processors made from multiple silicon dies, which can be exposed as sub-NUMA domains and used for optimization of applications. The hop across dies adds latency and is definitely higher on Haswell and Skylake than sub-domains. This should increase the importance of enhanced locations with sub-domains. This would be something of interest for scanning heavy database queries if one could achieve high memory throughput across dies. Even though we just look at Intel's COD and SNC offerings in this thesis. Intel is also the first provider to sell sub-domains of anything like that.
No previous study has examined task stealing inside a database on sub-domains within a processor, to the knowledge of the author.

3. Methods of analysis

To address our query, we first check the effect of COD and SNC on memory latency and bandwidth when the core and DRAM are situated in the same versus specific sub of a memory demand. This will give us an idea of what the underlying hardware works and eliminates any complications introduced when the database is placed on top. Since our target database queries depend heavily on memory latency and throughput, our subsequent database experiments will hopefully help to clarify this. We implement the database in our tests in the second round of experiments and study the effect of query throughput on single- and multi-row queries in various scheduler architectures and during slanted workforces. These two kinds of queries are easy to test and should be correlated with both memory latency and bandwidth.

4. Microarchitecture of Processors

The external interface and internal architecture of a processor are important for quickly serving several memory requests. A generalized processor microarchitecture that is identical to what is found on the Intel Skylake is shown in Figure 1. With the exception of a completely different caching logic, Haswell's microarchitecture is also identical to this figure. An array of DRAMs is physically mounted to the processor through memory channels. Memory requests can involve communication across processors via a socket interlink if a machine has multiple processors in able to reach DRAM connected to some other processor, which is often used to preserve cache coherence seen between processors. QPI [10] for Intel processor models including such Haswell and the faster UPI [11] for recent Intel generations such as Skylake are standard interconnect protocols. For their processors, other vendors give similar protocols. Communication over an interconnect provides extra latency and has a small bandwidth, but allows more DRAM attached and processing capacity within a system to be increased.
Internally, cores and external interfaces interact via a bus within a processor. The logic of the processor, closer to each other on the bus, interacts faster and does not transfer other bus paths [12].
Every core has one or more threads of hardware, comparably referring to as CPUs in this thesis, and perhaps even cache alignment logic. DRAM external interfacing is controlled by one or more memory controllers, each controlling its memory channel array. In the virtual memory address space, a continuous memory area is always interlocked with a 256-byte granularity between the systems in
order to use the bandwidth of both channels in parallel [13]. Output may also be done through controllers on processors with even more than one memory controller.

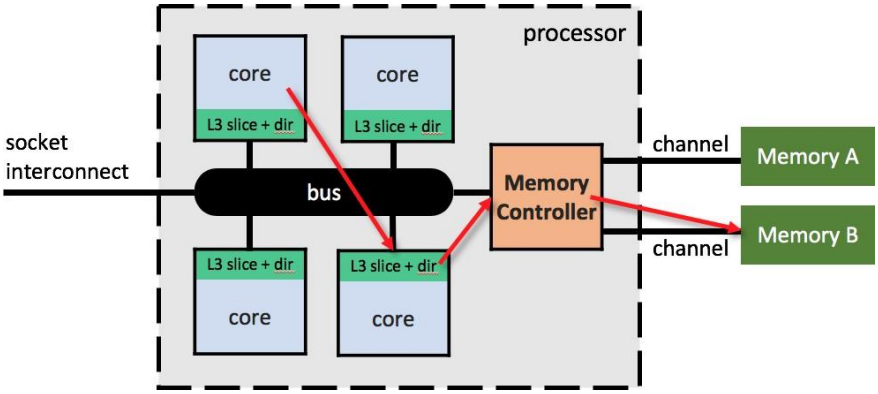

Figure 1: Generalized processor microarchitecture and its connected memory.

The arrows annotate the route of a CPU sample DRAM request to its register cache directory, to the target cache memory controller, to the alternating memory channel point. Only one memory controller and socket interconnect are accessible in this figure.

Cache Coherence Directory-based:

Cache coherence logic can accompany each cores and memory controllers. For each core, cores on both Haswell and Skylake microarchitectures incorporate local L1 and L2 caches, as well as a slice of the L3 cache that is shared in a domain among cores. It is important to check any memory request a CPU makes against the cache logic and implement the cache coherence protocol provided.
Directory-based cache coherence is the only protocol available on Skylake [14] that is used on Haswell when COD is enabled [15]. For a specific set of physical main memory that are assigned to it, a directory consolidates cache line tracking. Each application for a memory address that is not cached locally in L1 or L2 must be glanced in the directory that is liable for it. The path checks whether the address is present on any processor in the corresponding L3 slice or any other cache. If not, the request will be redirected to a memory controller with a DRAM targeting channel.

Through COD, Haswell uses single directory for each memory controller that is concerned for its addresses. Rather, Skylake has as many directories as cores, co-located along the cores, to minimize latency at these stages. A hash function maps directory addresses for an even load.
Caching data will render regularly accessed rows from indexed columns easily accessible for the in-memory database queries we are looking at. However, it also has a small advantage when scanning columns far larger than the cache, as is often the case with SAP HANA [16]. If we will consider the factors like reliability, confidentiality, and Quality of Service etc. of IOT then this clustering architecture can be effectively implemented [17].

Non-Uniform Accessing Memory

Non-Uniform Memory Access (NUMA) is a design for exposing different memory address regions on a computer to relative memory access times at various CPUs. These relative access times are interpreted by the Linux Memory Management system as NUMA domains, each of which has a set of CPUs and memory address previously been associated with it. For a simple description, see figure 2. A significant disruption to other domains is given to any domain. It is anticipated that access from a CPU to memory in a domain with a lower relative distance would have lower latency and greater bandwidth. In Linux, the OS and its implementations can use this configuration to position a given process and the memory that it accesses closer to each other. The placement can be performed manually using the library libnuma, or automatically using the NUMA-balancing service, which is performed by preserving memory access heuristics.

```
struct NumaDomain {
Cpu [] cpus;
MemChunk[]  regions;
}
Struct NumaDistance  {
NumaDomain* cpu;
NumaDomain* mem;
int weight;
}
```

Figure 2: Simplified data structures in C of how Linux Memory Management system represents a NUMA topology.

Since the NUMA topology shows relative differences in the access time of various memories, it is therefore frequently used in multi-processor configurations such as the one shown in Figure 3. Since the socket interconnect imposes latency and restricts the bandwidth to a comparatively high degree compared to memory viewed within a device, it increases access times by putting similar processes and memory on the same processor.

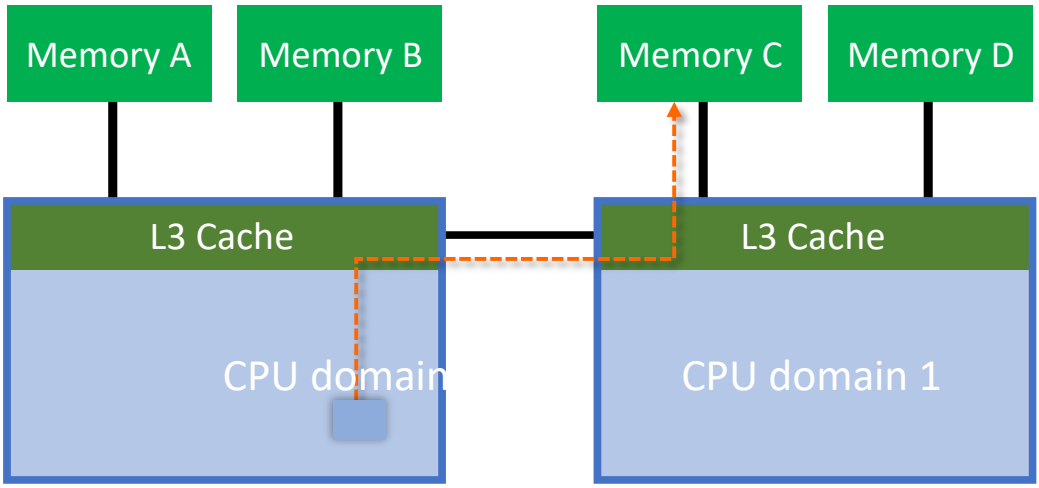

Figure 3: Example of a NUMA topology of a multi-processor setup, here consisting of two interconnected processors each with two DRAMs attached. The displayed memory request originates from the processor of Domain0, and targets Memory C that is attached to processor of Domain1.

NUMA Within a Processor:

The latency of an entry from a core to DRAM depends on how far inside the processor the memory request needs to go. Today's world, larger server processors have hundreds of cores on a processor and more than one memory controller. Access latency therefore depends progressively on how far apart the processor logic necessary to satisfy the application is located. Intel implemented a new coined Cluster-on-Die (COD) feature in the Haswell microarchitecture accessible on designs with more than one memory controller to reveal these latency differences within a processor.

Enabling COD in the BIOS separates the cores and memory controllers on the same silicon die inside the processor into two subdivided NUMA domains. Since the specific memory access cache directory is situated on the target memory's memory controller, the request is stored within a subsection of the silicon die.

In a multi-processor configuration, tests by Molka on a 12-core Haswell Intel Xeon E5-2680 v3 processor showed a reduction in latency of about 7 percent (Table III) and a 3.7 percent improvement in bandwidth (Tables VII and VIII) for requests dynamically within a sub-domain. Obviously, it depends on the caching state, remote reading of shared memory on the adjacent sub-domain appears to entail a broadcast to other processors, which more than doubles the latency and decreases bandwidth by 42% opposed to a local sub-domain access (table V and VIII). The Haswell processor used in these simulations has two buses that are unevenly attached to cores and memory controllers. There are queues linking the buses to each other. As the two sub-domains revealed are of similar size, these interfaces do not map to the two unevenly scaled buses, causing the queue to pass any local requests within a sub-domain. Similar experiments are performed in this study on a Haswell processor where even the two sub-domains map exclusively to different buses.

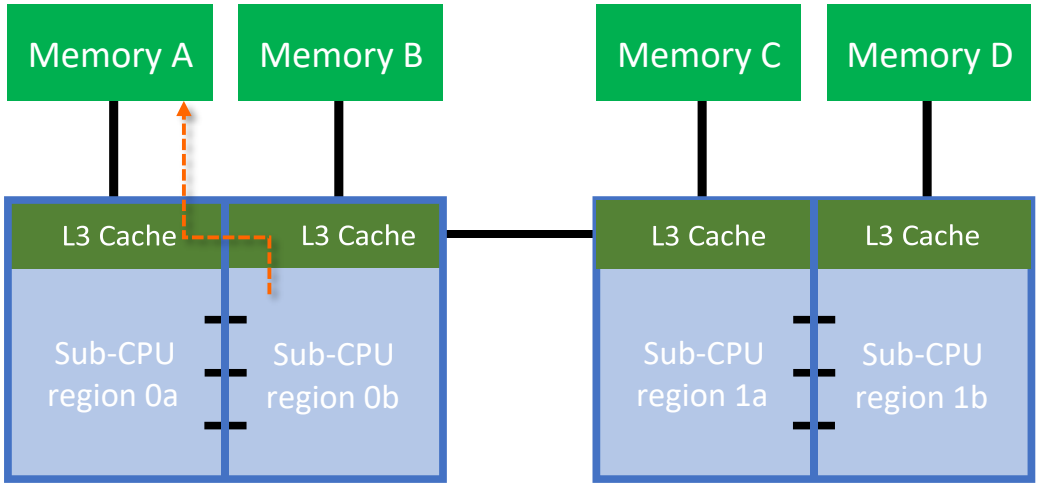

Figure 4: Two processors, each processor split into two sub-domains. In this picture, a core in CPU region # 1 is accessing memory in its neighboring sub-region.

COD was replaced by Sub-NUMA Clustering (SNC) with the Intel Skylake microarchitecture. In comparison to Haswell that locate cache directories at the memory controllers, Skylake allocates the directories and slices of L3 cache each used by keys mapped to its directory, across the cores as shown in figure 5. Any local and remote memory address is assigned to one of the directories based on an undocumented hash function. This mapping is reconfigured when SNC is allowed so that addresses to Memory A (see Figure 5) are only mapped to directories on the left side of the processor, while addresses to Memory B are only mapped to right section. This effectively creates two sub-NUMA domains with memory is not interleaved, with quicker access to memory within such a domain, as the requesting center, directory and memory are closer to each other. On the opposite, only one memory controller is involved in the request, which may restrict the performance. Both directories and L3 slices are also mapped with addresses from those other processors. Efforts to try to reverse engineer the unpublished hash function have resulted in the suggested management of slice-aware memory, where the cores and directories used are much closer to each other. As these cache lines are never modified across the sub-domains, Intel offers a lower latency to the neighboring sub-domain and better usage of the L3 cache, but in addition, in some cases, will increase L3 latency.

Intel also presents the microarchitecture of Knights Landing that targets high computing environments, with up to 72 cores and a total of 10 different kinds of memory controllers, but without a shared L3 cache. Like Skylake, with SNC, the processor can be split into sub-domains, with up to four sub-domains possible. Knights Landing also has an optional quadrant mode, which addresses are forwarded to directories in the same way as in SNC, for minimized directory-to-memory latency, but without sub-NUMA domain exposure in the OS.

On a single silicon die, Intel processors have steadily increased the number of cores. As this implies larger dies and increased development costs, AMD has opted for a multi-die strategy targeting server environments on its EPYC processor. Inside the same processor, the EPYC processor is split into four closely connected silicon dies. Because of these relations, which are revealed in the NUMA topology as one domain per die, the latency among

dies is slightly higher. In a multi-processor configuration, dies can link directly to each other on different processors without having to move through a standardized socket interconnect on the processors. This is being sold by AMD as Infinity Fabric. Similarly, the newly released Neoverse microarchitecture targeting server environments from ARM allows the designer to determine whether or not cores should be divided into several dies on one processor. Neoverse also does not, by default, have an L3 cache.

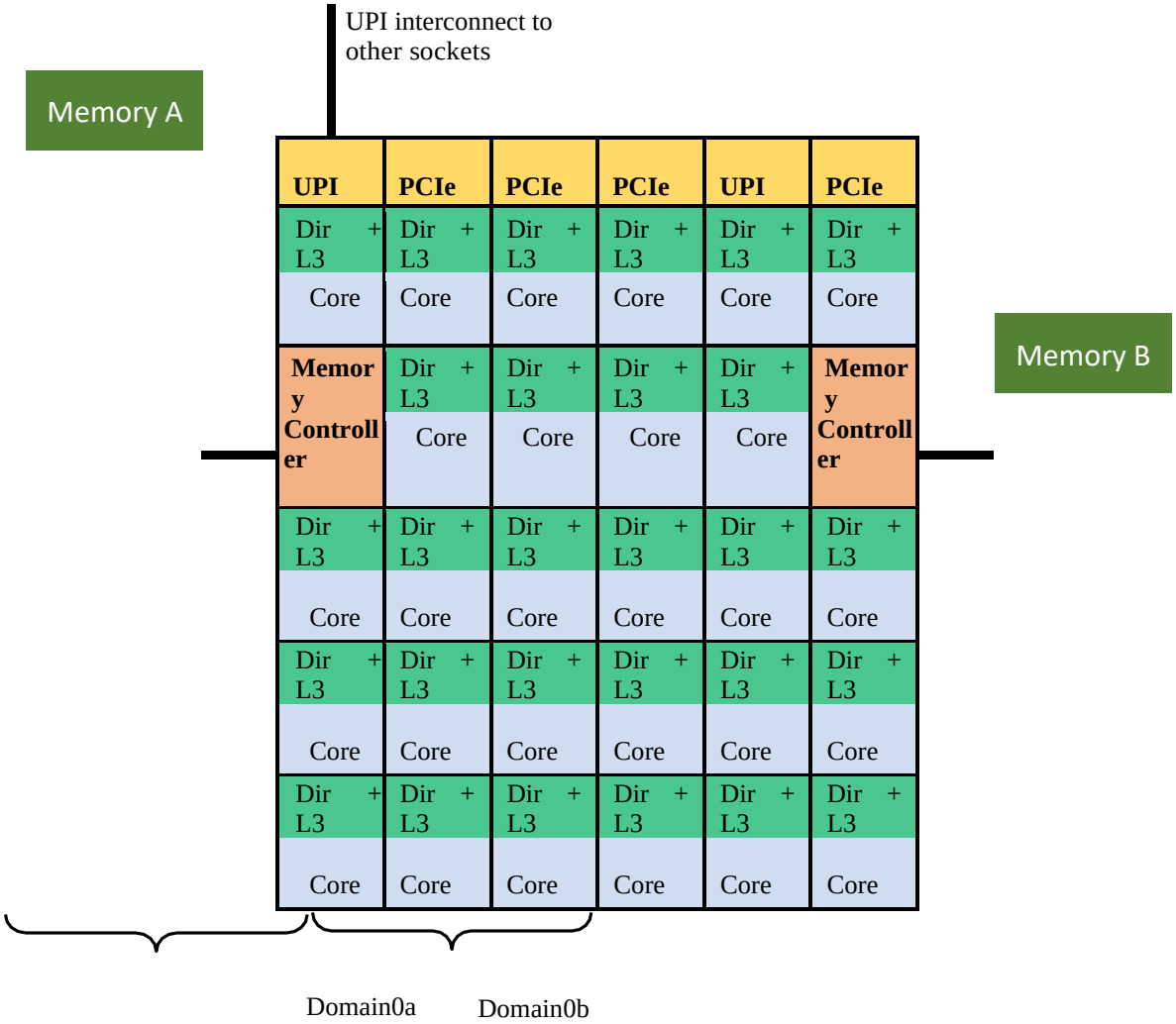

Figure 5: Skylake processor microarchitecture for Intel Xeon Platinum 8180 [1]. Each core has its own part of the distributed directory (Dir) and a slice of the L3 cache. Every core and external interface are connected to a Manhattan-style bus network.

Access column in SAP HANA

As described in the introduction, for efficient compression and quick scanning, SAP HANA stores each column separate from each other. The option of applying indexes to a column is also possible for large datasets where many queries are highly selective. Per-default, the primary key is indexed. Extremely selective access on indexed columns uses binary search to touch just a few elements and is thus primarily susceptible to memory latency. Column scans that are more susceptible to memory bandwidth result from access to non-indexed columns. Since columns in DRAM are compressed and rendered usable, columns.

In SAP HANA, Job Scheduling

Each database-connected client session is allocated its own OS thread used for link management, which may also generate query execution plans. A task graph is dynamically generated by the implementation plan and tasks themselves, where a given task is executable after all children have finished executing. One of the work queues that are current in every NUMA domain is forced into a task. During task formation, the preference for a particular domain is set and used when the task primarily accesses the memory in that domain. Preference is used in SAP HANA for column screening, but not on indexed columns. For each domain, worker threads pull tasks out of the local queue for implementation.

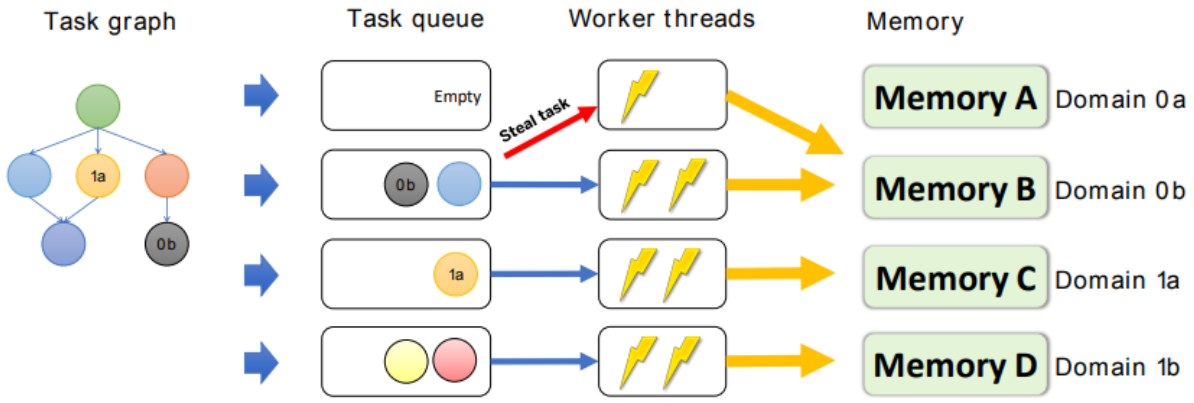

Figure 6: SAP HANA task execution flow.

Executable tasks in the task graph are pushed to the task's preferred domain, or any queue if a preference is not set. Worker threads primarily pull tasks from its local queue or from a neighbor queue if the local one is empty. If a preferred task is stolen, the worker thread might have to access memory remotely, as seen in domain 0a in this figure.

SAP HANA enables worker threads in a domain with unused task queues to steal tasks from queues in other domains in order to balance tasks between domains, that is, grab tasks from a remote domain queue. Task theft is a highly studied technique for load balancing and is commonly used in programming languages like Go. Stealing can be switched off internally in SAP HANA, enabled only between near neighbors in NUMA-topology, or across any domain. The default setting in SAP HANA is neighbor-stealing, as it avoids expensive stolen activities from placing unnecessary load on socket interconnects. The probability of bulk theft by stealing several tasks at a time is also present.

- *no binding*: workers are freely movable by the OS
- *preferred binding*: bind to the preferred domain only when working on a task with preference
- *opportunisticbinding*(default): In addition to *preferred binding*, also bind the task's direct children
- *bind always:* Prohibit the OS from moving around workers.

5. Results

Using Broadwell Processor:

The average lag and its duration observed from various hardware threads within each domain to memory located in distinct domains, as calculated using MLC, are presented in table 1 and table 2. With COD, we see an average 7 percent decrease and an average 94 percent rise in local versus neighboring sub-domain latency, respectively. There is also a significant rise in remote processor latency.

Table 1: In nanoseconds, memory access latency for hardware threads in each domain. Disabled COD.

Mem / *CPU*	**Domain 0**			**Domain 1**		
	min	avg	max	min	avg	max
Domain 0	-2,8	79,3	+2,3	-3,8	134,4	+5,0

Domain 1	-3,6	134,2	+5,7	-3,4	79,5	+4,0

Table 2: memory access latency with COD enabled.

Mem / *CPU*	**Domain 0a**			**Domain 0b**			**Domain 1a**			**Domain 1b**		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Domain 0a	-1,5	74,0	+2,1	-0,8	156,6	+2,4	-0,4	195,4	+1,0	-2,1	204,9	+0,6
Domain 0b	-2,3	152,7	+2,2	-1,3	75,2	+1,4	-1,7	196,7	+3,4	-2,3	204,7	+1,0
Domain 1a	-1,0	196,3	+2,1	-1,2	205,3	+0,2	-1,3	74,1	+1,0	-1,8	156,2	+2,0
Domain 1b	-2,6	197,9	+2,5	-2,2	205,4	+1,5	-1,9	152,0	+2,4	-1,5	75,1	+1,6

When COD is activated, bandwidth per memory controller is enhanced by 2.1-3.7 percent versus falling by 38-51 percent for local and neighbor sub-domain. We can also see that on a remote processor, the throughput number of both sub-domains decreases by 3-8 percent per memory controller.

Table 3 Bandwidth between domains in MB/s. COD disabled.

Mem / *CPU*	**Domain 0**	**Domain 1**
Domain 0	62 134	30 471
Domain 1	30 155	61 502

Table 4 Bandwidths with COD enabled

Mem / *CPU*	**Domain 0a**	**Domain 0b**	**Domain 1a**	**Domain 1b**
Domain 0a	32 210	15 322	12 742	12 274
Domain 0b	18 335	31 749	15 013	14 503
Domain 1a	12 754	12 296	31 751	15 268
Domain 1b	14 850	14 392	18 808	31 750

Using Skylake Machine:

Table 5 and table 6 display our Skylake machine's MLC latency measurements. We see an insignificant difference in local latency of 6.4 percent when SNC is activated. As the CPU, target cache directory, and specified memory controller are closer to each other on average, this is fair. Notice that our earlier observations on older BIOS versions have been disabled with SNC with 7 nanoseconds faster local latency than the measurements shown here, and have thus seen a lower improved performance in latency with SNC.

Table 5: Memory access latency in nanoseconds for hardware threads in each domain. SNC disabled.

Mem / *CPU*	**Domain 0**			**Domain 1**		
	min	avg	max	min	avg	max
Domain 0	-3,7	80,8	+1,6	-3,6	138,9	+3,8
Domain 1	-3,9	139,7	+3,8	-3,2	79,9	+1,9

Table 6: Memory access latency with SNC enabled.

Mem / *CPU*	**Domain 0a**			**Domain 0b**			**Domain 1a**			**Domain 1b**		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Domain 0a	-1,2	74,2	+3,1	-0,1	81,5	+0,3	-2,1	132,0	+3,0	-1,4	142,1	+1,6
Domain 0b	-0,2	82,0	+0,2	-3,5	76,4	+5,0	-1,9	135,6	+2,0	-1,5	144,5	+1,6
Domain 1a	-2,1	132,4	+3,0	-1,4	142,0	+1,5	-0,7	73,6	+6,0	-0,1	81,5	+0,1
Domain 1b	-1,9	136,0	+2,1	-1,4	144,4	+1,7	-0,1	81,5	+0,1	-3,6	76,6	+5,0

Table 7 and table 8 demonstrate the calculation of MLC bandwidth on our Skylake computer. As with COD, utilizing SNC mode should give us approximately half the bandwidth per sub-domain, which we can notice in the tables as well. On average, bandwidth per memory controller increases for local and neighboring sub domains by 4.2 percent and 4.4 percent. It is important to note here that the bandwidth of the neighboring sub-domain is higher and thus greater than that of the local sub-domain.

Table 7: Bandwidth between domains in MB/s. SNC disabled.

Mem / *CPU*	**Domain 0**	**Domain 1**
Domain 0	111 083,3	34 451,2
Domain 1	34 454,9	111 618,7

Table 8: Bandwidth between sub-domains in MB/s. SNC enabled.

Mem / *CPU*	**Domain 0a**	**Domain 0b**	**Domain 1a**	**Domain 1b**
Domain 0a	58 087,1	58 122,9	34 254,3	34 238,9
Domain 0b	58 144,8	58 012,9	34 265,9	34 235,1
Domain 1a	34 287,6	34 248,1	58 064,0	58 146,7
Domain 1b	34 288,3	34 253,6	58 145,0	58 006,9

Our observed COD latencies and bandwidths are nearly similar to those measured by Molka. It did not seem to alter the behavior of substantially increased latency and decreased bandwidth to neighbor sub-domain by using our Broadwell processor with symmetrical sub-domains and shrinking in transistor capacity. With SNC, adjacent sub-domain latency and bandwidth are closer to what has been calculated on local access than access to a remote processor. This is more rational than what has been found with COD, and is possibly due to the different Skylake caching strategy. Memory access to that other processor appears to be largely unchanged, in comparison to COD.

Conclusion and Future scope:

In this thesis, when allowing COD and SNC modes on Intel, we checked the change in memory latency and bandwidth between different cores and memory devices. For any remote access, COD did not display attractive latency and bandwidth, partially due to the cache policy being used. SNC provided the OS and its applications with better latency outcomes and a more comprehensive exposure of the processor topology. Bandwidth to the local and neighbor sub-domain with SNC mode both increased slightly at about the same level.
This is possibly due to the nature of the test question. The bigger problem with sub-domain scans is that there is no interleaved memory and no aggregated throughput available. It renders an application unable to use the total bandwidth available in a processor without load balancing. While our attempts to balance scanning, workloads did not resolve the interleaving of hardware with SNC disabled, it might be worthwhile to look at whether software-based interleaving can be achieved relatively quickly in the future, so that random access workloads can enjoy lower latencies at the same time. It will also be of interest to test the sub-domain solutions of other processor vendors to see whether query throughput with sub-domains can be enhanced.

Assuming that there will be even more cores for future processors, placing both cores on one die will be more difficult. Therefore, it will also be important to test processors in the future with multiple dies. Even further analysis at the stealing mechanisms in a multi-machine RDMA setup will also be of interest. It is logical to assume, with even more cores, that cores will inevitably have to be spread through many machines.

References

1. Intel, "Intel Xeon Processor Scalable Family Technical Overview," 14 Sep 2017. [Online]. Available: https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family- technical-overview.
2. T.P Morgan, "The Next Platform," Feb 2019 [Online]. Available: https://www.nextplatform.com/2019/02/20/arm-goes-to-war-in-the-datacenter-with-aries- designs/.
3. TIRIAS Research, "AMD Optimizes EPYC Memory with NUMA," Mar 2018. [Online]. Available: https://www.amd.com/system/files/2018-03/AMD-Optimizes-EPYC-Memory- With-NUMA.pdf.
4. VMware, "Intel Cluster-on-Die (COD) Technology, and VMware vSphere 5.5 U3b and 6.x," [Online]. Available: https://kb.vmware.com/s/article/2142499.
5. I. Psaroudakis, T. Scheuer, N. May, A. Sellamiand A. Ailamaki, “Scaling Up Concurrent Main- Memory Column-Store Scans: Towards Adaptive NUMA-aware Data and Task Placement," in VLDB Endowment, 2015.
6. Intel, "An Introduction to the Intel Quick Path Interconnect," Jan 2009. [Online]. Available: https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html.
7. Intel, "How Memory Is Accessed," [Online]. Available: https://software.intel.com/en- us/articles/how-memory-is-accessed. [Accessed 6 Sep2019].
8. D. Molka, D. Hackenberg, R. Schöneand W.E. Nagel,"Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture," in Proceedings of the 44th International Conference on Parallel Processing (ICPP’15),2015.
9. S. Noll, J. Teubner, N. May and A. Bohm, "Accelerating Concurrent Workloads with CPU Cache Partitioning,"2018.
10. The Linux Foundation, "What is NUMA?" 8 Jul 2019. [Online]. Available: https: //www. kernel. org/doc /html/v4.18/vm/numa.html.
11. A. Kleen, "NUMA policy library," [Online]. Available:http://man7.org/linux/man- pages/man3 /numa.3.html. [Accessed 27 Aug2019].
12. SUSE, "Automatic Non-Uniform Memory Access (NUMA) Balancing," [Online]. Available: https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/cha.tuning.numactl.html. [Accessed 27 Jul2019].
13. U. Drepper, "What Every Programmer Should Know About Memory," 21 Nov 2007. [Online]. Available: https: //people.freebsd.org/~lstewart/articles/cpumemory.pdf.
14. Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual," 2019. [Online]. Available: https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures- optimization-manual.pdf.
15. A. Farshin, A. Roozbeh, M.J.Gerald Q and D.Kostić,"Make the Most out of Last Level Cache
16. inIntelProcessors,"inProceedingsoftheFourteenthEuroSysConference2019, Dresden,2019.
17. R.D. Blumofe, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall and Y. Zhou, "Cilk: an efficient multithreaded runtime system," Sigplan Notices, vol. 30, no. 8, pp. 207-216,1995.
18. P. Datta, and B. Sharma. "A survey on IOT architectures, protocols, security and smart city based applications." In 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1-5. IEEE, 2017.