DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2020

Effective Task Scheduling of In-Memory Databases on a Sub-NUMA Processor Topology

HADAR GREINSMARK

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Effective Task Scheduling of In-Memory Databases on a Sub-NUMA Processor Topology

Hadar Greinsmark
hadarg@kth.se

Master's Programme in Computer Science
Master of Science in Engineering in Computer Science

Supervisor:
Hamid Reza Faragardi*

Industry supervisors:
Norman May †, Roman Dementiev ‡,
and Thomas Willhalm ‡

Examiner:
Elena Troubitsyna*

* KTH Royal Institute of Technology, † SAP, ‡ Intel

February 24, 2020

Walldorf, Germany / Stockholm, Sweden

ABSTRACT

Query throughput of in-memory databases greatly depends on how fast data can be accessed in DRAM. With a growing number of cores on processors, the expectation that all attached DRAM on a processor is equally accessible by every core is harder to meet. Therefore, Intel has introduced a mode coined Sub-NUMA Clustering (SNC) on Skylake, which subdivides cores and memories into multiple sub-domains for improved core-to-memory access within each sub-domain on the processor. Other models offer similar modes. Usage of SNC poses challenges on how to balance database workloads between domains when an in-memory database share data between workers on cores in different sub-domains. In this thesis, we verify SNC's impact on memory latency and bandwidth primarily on Intel Skylake. Furthermore, we test the in-memory database SAP HANA when using SNC on single-row and analytical workloads. We conclude that two equally sized analytical workloads put on separate sub-domains and fully isolated from each other, only would increase query throughput up to 3%. Also, as memory bandwidth is divided into sub-domains rather than aggregated on the entire processor, analytical workloads that are sensitive to bandwidth drastically decrease query throughput if data are not evenly partitioned between sub-domains.

SAMMANFATTNING *(SWEDISH)*

Genomströmning av databasförfrågningar till minnesdatabaser beror i hög grad på hur snabbt åtkomligt data liggandes i DRAM är. Då antal kärnor på processorer hela tiden ökar, blir det allt svårare uppehålla förväntningen av att alla DRAM-enheter är lika snabbt tillgängliga för alla kärnor. Intel har därför introducerat en ny inställning vid namn ”Sub-NUMA Clustering” förkortat SNC. Denna inställning är tillgänglig på Skylake-processorer och delar upp kärnor och minnesenheter i subdomäner, med syftet att förbättra åtkomsttiden av minnesförfrågningar från en kärna till minne i samma subdomän. Även andra modeller erbjuder liknande typ av inställningsmöjlighet. Med separata subdomäner måste arbetet i en databas på något sätt koordineras mellan subdomänerna, vilket försvåras om minnet i en minnesdatabas delas av arbeten i olika subdomäner samtidigt. Vi vill därför i den här uppsatsen först verifiera hur SNC förändrar åtkomsthastigheten till minnet och minnesgenomströmningen. Framförallt tittar vi på Skylake-arkitekturen. Vi testar också att köra minnesdatabasen SAP HANA med SNC-inställningen när denna söker efter enskilda tabellrader, eller gör statistiska dataanalyser. Vi drar slutsatsen att om två helt isolerade statistiska dataanalyser kör samtidigt på olika subdomäner, leder detta bara upp till 3% ökad genomströmning av databasförfrågningar. Då minnet är delat i två subdomäner kan inte minne från olika DRAM-enheter sammanflätas för högre minnesgenomströmning, vilket framförallt saktar ner dataanalyser som behöver skanna stor mängd data som inte är jämnt fördelat mellan subdomänerna.

TABLE OF CONTENTS

DEFINITIONS

bus	a communication system for transfers within a processor
COD	Cluster-on-Die; a mode first introduced by Intel on their Haswell processors
core	a set of hardware threads sharing *L1* and *L2* caches
CPU	Central Processing Unit; in this thesis interchangeable with *hardware thread*
directory	centralized tracking of cache lines; see section 2.2
DRAM	Dynamic Random Access Memory
hardware thread	processing unit; in this thesis interchangeable with *CPU*
L1/L2	Level 1 and 2 Cache; memory caches local to each core
L3	Level 3 Cache; memory caches shared between a set of cores
local access	a memory request within a *domain*
NUMA	Non-Uniform Memory Access; see section 2.3
NUMA domain	a set of *CPUs* and memory address regions within a *NUMA* topology; see section 2.3
remote access	memory request with originating core and target memory in different *domains*
silicon die	a circuit manufactured on a single integrated block
SNC	Sub-NUMA Clustering; Intel's replacement of *COD* on Skylake
socket interconnect	interconnection between two or more *sockets* on a machine; implemented by QPI and UPI on Intel processors
SQL	a query language for clients to send commands to databases
sub-domain	a *NUMA domain* in COD/SNC mode that is a subdivision of a *domain* when this mode is disabled

1 INTRODUCTION

Database performance is heavily dependent on how fast the database can access large amounts of data. Among many different vendors of relational SQL databases, SAP HANA is a database developed by SAP, powered by an in-memory approach to make data access fast. By compressing the data and using computers with large DRAM memories, entire datasets can reside in memory, avoiding access to slow discs. Tables are stored column-wise, rather than row-wise as in traditional databases, making compression easier when similar data are co-located. These approaches are beneficial for analytical workloads that generally deal with aggregations in a columnar fashion. Fast analytics is the key selling point for SAP HANA.

The performance of an in-memory database depends heavily on how fast the database can access its data in DRAM. Access time not only depends on the type of memory, but also the caching strategy used in the processor, as well as how far a memory access request must travel inside the processor to the target memory. Processor manufacturers continually offer processors with more cores, and memories are getting bigger and cheaper. Rather than having one central processor attached to one memory region, hardware nowadays is more cluster-oriented with co-located cores and memories working either in isolation or coordination. Because of this, offering equally fast access to data in all memory regions from every core in a processor is getting increasingly difficult. To expose these variations to applications, Intel introduced an optional mode in their Haswell generation coined *Cluster-on-Die* (COD) that later got replaced by *Sub-NUMA Clustering* (SNC) in the Skylake generation [1]. Similar technologies are found by other vendors [2] [3]. These features subdivide the *processor* into *sub-domains*, with an aim to make core-to-memory access faster within each sub-domain of the processor. Access across different domains is still accessible, but at a potentially higher latency. Sub-domains are particularly interesting as the current trend indicates a further increase in the number of cores on processors, something that inevitably makes access times further dependent on a core's placement within the processor.

Technologies like COD/SNC and alike are intuitively suited for applications where processes use their own memory rather than sharing it between them. Typically, this is the case for virtual machines (VM), where VMs rarely access each other's memory. VMware is one VM platform that can take advantage of Intel's COD mode [4]. In contrast, in-memory databases use a large memory region shared between simultaneously executing query tasks. Execution and memory cannot be as easily isolated from each other as in VMs. Nevertheless, databases such as SAP HANA utilize some task scheduling practices on machines with multiple processors, where tasks scanning a table column are preferably executed on the processor, in which the DRAM device of the column's memory is attached to for improved locality. However, it may be allowed to execute elsewhere for load balancing reasons. This makes memory scanning faster and SAP HANA scale better to more processors on a machine [5].

Research question. Can the potentially improved memory latency and bandwidth with sub-NUMA domains utilize locality-aware scheduling practices, in order to increase query throughput with SAP HANA? Memory latency and bandwidth between sub-domains within a processor versus between processors are expected to be greatly different. The task scheduler may also more actively need to rebalance tasks between sub-domains. In this thesis, we would like to explore these potential problems and improvements.

Limitations. Our focus is on read-only queries, as the key selling point for SAP HANA is fast analytics rather than small update transactions. Database joins and mixed workloads executed in parallel are good candidates for future investigation. Furthermore, we only look at Intel's implementation. This is because Intel where the first one offering something like sub-domains, and because SAP HANA is mainly built for and used with Intel processors. For many users, the necessary hardware is already in use, and it is only a matter of changing the configuration.

Research methodology. To answer our question, we first verify the impact of COD and SNC on memory latency and bandwidth when core and DRAM are of a memory request are located in the same

versus different sub-domains. This will give us an understanding of how well the underlying hardware is performing and removes any complexities introduced when adding the database on top. As our target database queries heavily relies on memory latency and throughput, this will hopefully help explaining our subsequent database experiments. In the second round of experiments, we introduce the database in our tests and measure the change of query throughput on single- and multi-row queries in different scheduler configurations and during skewed workloads. These two types of queries are simple to test and should correlate with memory latency and bandwidth respectively.

Outline. In chapter 2, we look at how COD and SNC are supposed to improve memory latency at the hardware level. Also, we explain how SAP HANA is accessing column data in memory as well as how tasks accessing the data are currently scheduled on machines with multiple processors. A brief introduction to related research is given in chapter 3; some that is already done on SAP HANA, as well as similar challenges on multi-machine setups. We begin our experiments with verifying the COD and SNC improvements in chapter 4, and conclude that the COD mode performs below our expectations for memory accesses across sub-domains. SNC gives however a slight improvement. In chapter 5, we therefore continue our testing with SAP HANA only with SNC on single- and multi-row aggregation workloads. Afterwards we discuss our findings in chapter 6. Based on our experiments, we conclude in chapter 7 that even if SNC gives slight improvements in latency and bandwidth, it is not justifiable to use with SAP HANA at this point, when considering the extra cost of load balancing between sub-domains.

2 BACKGROUND

In order to understand how COD and SNC are aiming to reduce access time to data within sub-domains, we need to explain common processor microarchitecture and how they are implemented on Haswell and Skylake. We also introduce caching strategies to the reader, as different strategies is impacting the latency and changes slightly when using COD and SNC. On the software side, we look at how these varying access times are exposed in Linux with libnuma, and how it is used by the SAP HANA database to schedule tasks for efficient column access.

2.1 Processor Microarchitecture

A processor's external interface and internal design are critical for serving many memory requests fast. Figure 1 shows a simplified processor microarchitecture that is similar to what is found on Intel Skylake. Haswell's microarchitecture is also similar to this figure, except for a slightly different caching logic.

Externally, an array of DRAM though *memory channels* is attached to the processor. If a machine has multiple processors, memory requests may require communication across processors through a *socket interconnect* in order to access DRAM attached to another processor, which is also used to maintain *cache coherence* between the processors. Common interconnect protocols are QPI [6] for Intel processor models such as Haswell, and the faster UPI [1] for newer Intel generations like Skylake. Other vendors offer similar protocols for their processors. Communication across an interconnect introduces additional latency and has a limited bandwidth, but allows for more attached DRAM and an increase in processing power within a machine.

Internally in a processor, *cores* and external interfaces communicate through a *bus*. Processor logic closer to each other on the bus communicate faster, and does not traffic other paths of the bus [7]. Each core has one or more *hardware threads* – in this thesis equivalently referred to as CPUs – as well as cache coherence logic. External interfacing with DRAM is managed by one or more *memory controllers* that each control its array of memory channels. A continuous memory region in the physical memory address space is often *interleaved* between the channels at a 256-byte granularity, in order to use the bandwidth of both channels in parallel [7]. On processors with more than one memory controller, interleaving can also be done across controllers.

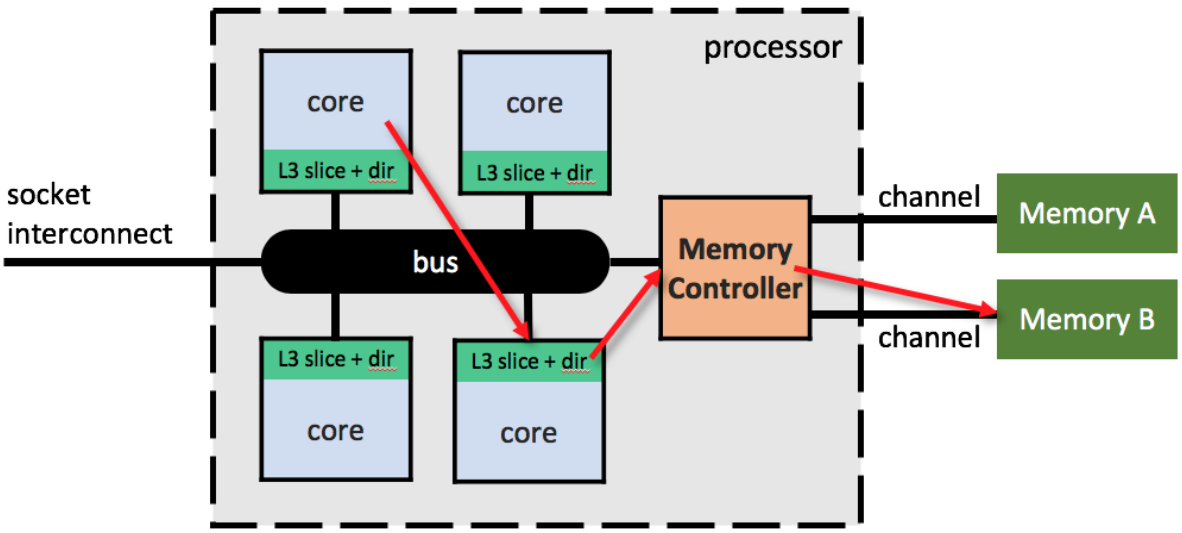

Figure 1: Simplified microarchitecture of a processor and its attached memory. The arrows annotate the route of a sample DRAM request from a CPU, to its address-mapped cache directory, to target memory controller on a cache miss, to target interleaved memory channel. In this figure, only one memory controller and socket interconnect is available.

2.2 Directory-based Cache Coherency

Both cores and memory controllers may be accompanied by cache coherence logic. Cores both on Haswell and Skylake microarchitectures implement local L1 and L2 caches at each core, but also a *slice* of the L3 cache that is shared between cores in a *domain*. Each memory request a CPU is doing has to be checked against the cache logic and follow the given *cache coherence protocol*.

Directory-based cache coherency is one protocol used on Haswell when COD is enabled [8], and the only protocol available on Skylake [1]. A *directory* centralizes tracking of cache lines for a particular set of physical memory addresses that are mapped to it. Each request to a memory address not cached locally in L1 or L2 needs to be looked up in its responsible directory. The directory checks if the address is present in the corresponding L3 slice or any other cache on any processor. If not, the request is forwarded to the memory controller with a channel to target DRAM. See the annotated path in figure 1.

Haswell with COD uses for each memory controller one directory that is responsible for its addresses. To reduce congestion at these points, Skylake instead has as many directories as cores, co-located along the cores. A hash function map addresses to directories for an even load.

For the in-memory database queries we are looking at, caching data could make frequently accessed rows form indexed columns quickly available. But it also has limited benefit when scanning columns greatly bigger than the cache, which is often the case in SAP HANA [9].

2.3 Non-Uniform Memory Access

Non-Uniform Memory Access (NUMA) is a design for exposing the relative memory access times from different CPUs to different memory address regions on a machine. Linux's Memory Management system represents these relative access times as *NUMA domains*, which each have a set of CPUs and memory address regions associated with it [10]. See figure 2 for a simplified representation. Every domain is given a relative distance to other domains. Access from a CPU to memory in a domain with a lower relative distance is expected to have lower latency and higher bandwidth. This topology can in Linux be used by the OS and its applications to place a given process and the memory it is accessing closer to each other. Placement can be done manually using the libnuma library [11], or automatically using the NUMA-balancing service, which it is doing by maintaining heuristics on memory accesses [12].

```
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.

As a NUMA topology expose relative variations in different memories' access time, it is therefore often used in multi-processor setups like the one shown in figure 3. As the socket interconnect introduces latency and limits the bandwidth to a relatively high degree compared to memory accessed within a processor, placing associated processes and memory on the same processor improves access times [13].

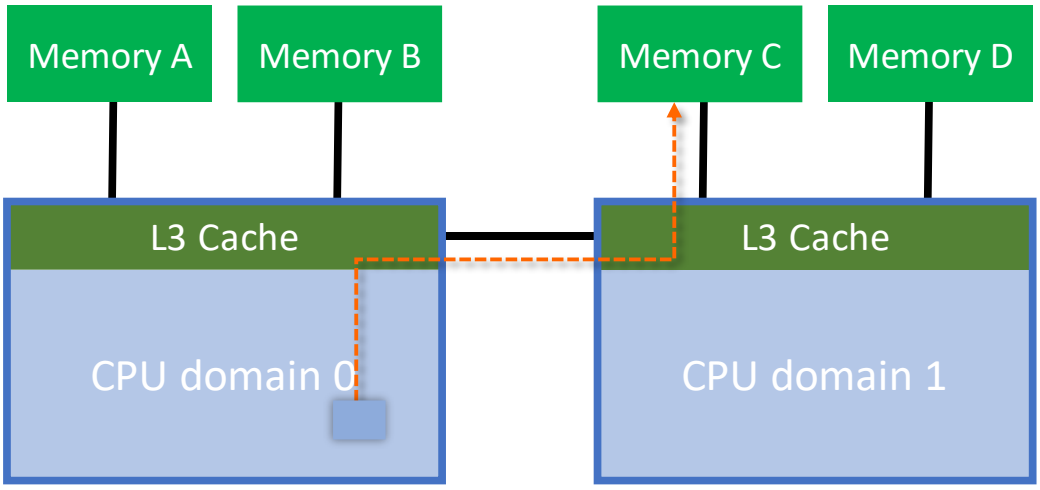

Figure 3: Example of a NUMA topology of a multi-processor setup, here consisting of two interconnected processors each with two DRAMs attached. The displayed memory request originates from the processor of Domain 0, and targets Memory C that is attached to processor of Domain 1.

2.4 NUMA Within a Processor

The latency of an access from a core to DRAM depends on how far the memory request needs to travel within the processor. Larger server processors nowadays provide dozens of cores and more than one memory controller on a processor. Thus, access latency increasingly depends on how far apart the processor logic needed to serve the request are located. To expose these latency differences within a processor, Intel introduced a new feature coined *Cluster-on-Die* (COD) in the Haswell microarchitecture available on models with more than one memory controller [1]. Enabling COD in BIOS divides the cores and memory controllers into two subdivided NUMA domains on the same silicon die within the processor. As the cache directory for local memory accesses is located at the memory controller of the target memory, the request is contained within a subset of the silicon die.

Experiments by Molka [8] on a 12-core Haswell Intel Xeon E5-2680 v3 processors in a multi-processor setup have shown around a 7% latency reduction (table III), and 3.7% increased bandwidth (table VII and VIII) for requests locally within a sub-domain. Remotely reading shared memory on neighboring sub-domain seems to, depending on the caching state, require a broadcast to other processors, which more than doubles the latency and reduces bandwidth by 42% compared to a local sub-domain access (table V and VIII). The Haswell processor used in these experiments has two buses that cores and memory controllers are connected to unevenly. The buses are bridged to each other by queues. As the two exposed sub-domains are of even size, these domains does not map to the two unevenly sized buses, which requires some local requests within a sub-domain to pass the queue. In this thesis we perform similar experiments on a Haswell processor where the two sub-domains map strictly to separate buses.

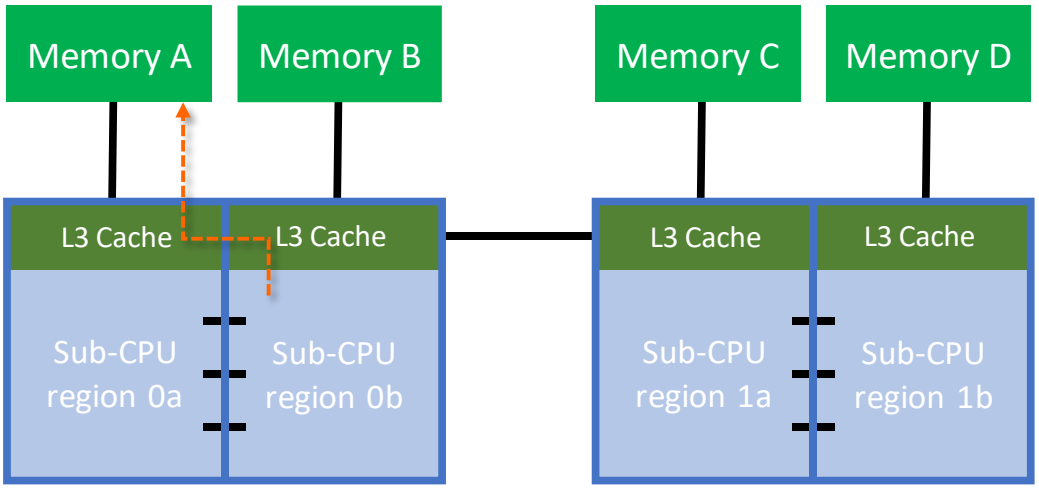

Figure 4: Two processors, each processor split into two sub-domains. In this picture, a core in CPU region # 1 is accessing memory in its neighboring sub-region.

With the Intel Skylake microarchitecture, COD was replaced by *Sub-NUMA Clustering* (SNC) [1] [14]. In contrast to Haswell that locate cache directories at the memory controllers, Skylake distributes the directories and slices of L3 cache each used by addresses mapped to its directory, along the cores as seen in figure 5. Every local and remote memory address is mapped to one of the directories based on an unpublished hash function. When SNC is enabled, this mapping is reconfigured so that addresses to **Memory A** (see figure 5) only maps to directories on the left portion of the processor, whereas addresses to **Memory B** only maps to the right portion. With memory is not interleaved, then this essentially creates two sub-NUMA domains, with faster access to memory within a domain, as the requesting core, directory and memory are closer to each other. On the contrary, only one memory controller participates in the request which might limit the throughput. Addresses from other processors are still mapped across all directory and L3 slices. Attempts to reverse engineer the unpublished hash function have resulted in proposed slice-aware memory management, where cores and directory used are placed even closer to each other [15]. Intel promises a lower latency to neighbor sub-domain and improved utilization of the L3 cache as those cache lines are never duplicated across the sub-domains, but which in return might increase L3 latency in some cases. (see section 9.2 in Reference Manual [14]).

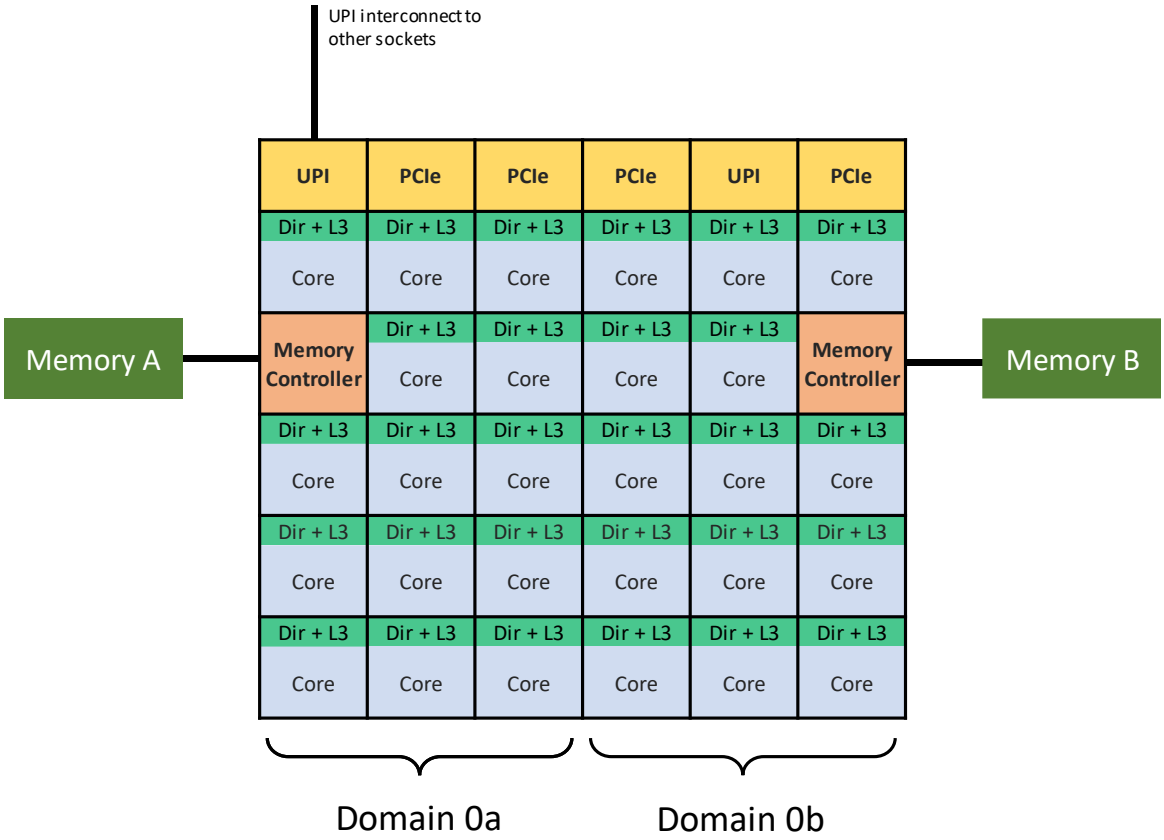

Figure 5: Skylake processor microarchitecture for Intel Xeon Platinum 8180 [1]. Each core has its own part of the distributed directory (Dir) and a slice of the L3 cache. Every core and external interface is connected to a Manhattan-style bus network.

Intel also offers the Knights Landing microarchitecture that targets high-performance computing environments, having up to 72 cores and a total of 10 memory controllers of different types, but with no shared L3 cache [16]. Like Skylake, the processor can with SNC be divided into sub-domains, with the possibility of up to four sub-domains. Additionally, Knights Landing has an extra mode, *quadrant mode*, where addresses are remapped to directories the same way as in SNC, for reduced directory-to-memory latency, but with no exposure of sub-NUMA domains in the OS (see figure 6 in [16]).

Intel processors have continuously increased the number of cores on a single silicon die. As this means larger dies and increased production costs, AMD has opted for a multi-die approach on their EPYC processor, targeting server environments [3]. The EPYC processor is divided into four tightly linked silicon dies within the same processor. The latency between dies is slightly higher due to these links, which is exposed in the NUMA topology as one domain per die. In a multi-processor setup, dies on different processors can connect directly to each other without having to pass through a centralized socket interconnect on the processors. AMD is marketing this as Infinity Fabric. Similarly, ARM's newly introduced Neoverse microarchitecture targeting server environments lets the manufacturer decide if cores on one processor should be split into multiple dies or not [2]. Also, Neoverse does not provide an L3 cache per default.

2.5 Column Access in SAP HANA

As mentioned in the introduction, SAP HANA stores each column separate from each other for efficient compression and fast scanning. For large datasets where many queries are highly selective, there is also the possibility of adding indexes on a column. The primary key is indexed per-default. Highly selective accesses on indexed columns only touch a few elements using binary search and is therefore mainly sensitive to memory latency. Accesses to non-indexed columns result in column scans that are more sensitive to memory bandwidth. As columns are compressed and made available in DRAM, column

scanning is often faster than index lookups on non-trivial queries. Indexes also require maintenance during table modifications.

SAP HANA gives an administrator the opportunity to hand-tune which NUMA domain a table is placed on using the SQL statement `ALTER TABLE "TBL_X" NUMA NODE ('Y')`, upon which table `TBL_X` is placed on domain `Y`. There is also the opportunity for fine granular placement of each column or a range of rows in a table [17].

2.6 Task Scheduling in SAP HANA

Each client session connected to the database gets assigned its own OS thread used for connection management which also may generate execution plans for queries [5]. The execution plan and tasks themselves dynamically generate a task graph, where a given task is executable after any children finished executing. A task gets pushed into one of the task queues that are present in every NUMA domain. Preference for a specific domain is settable during task creation and used when the task access memory mainly in that domain. In SAP HANA, preference is used for column scanning, but not on indexed columns. Worker threads in each domain pull tasks from the local queue for execution.

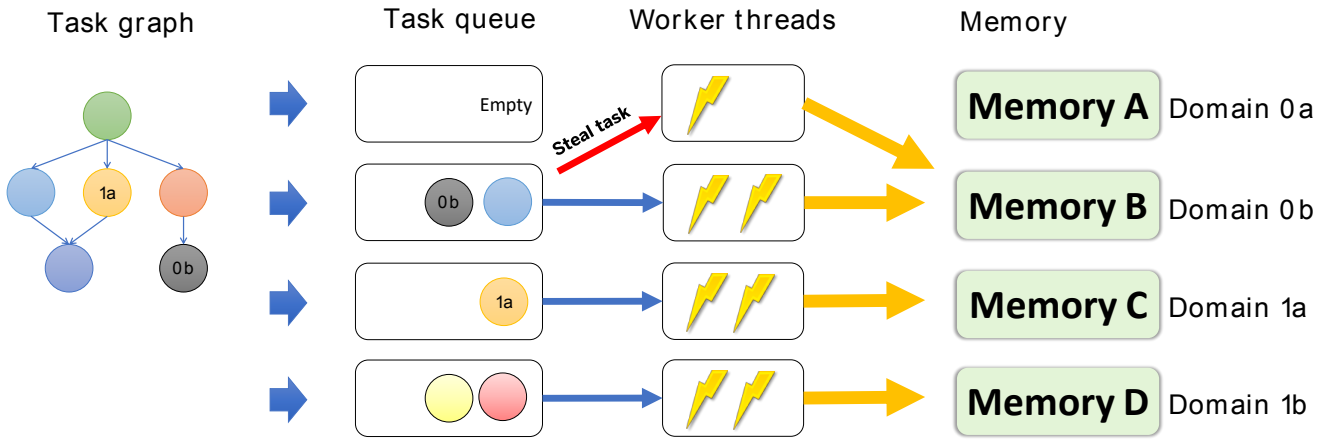

Figure 6: SAP HANA task execution flow. Executable tasks in the task graph are pushed to the task's preferred domain, or any queue if a preference is not set. Worker threads primarily pull tasks from its local queue or from a neighbor queue if the local one is empty. If a preferred task is stolen, the worker thread might have to access memory remotely, as seen in domain 0a in this figure.

To balance tasks among domains, SAP HANA allows worker threads in a domain with empty task queues to steal tasks from queues in other domains, that is, pull tasks from a queue on a remote domain. Task stealing is a heavily researched load balancing technique [18] and widely used in programming languages such as Go [19]. Internally in SAP HANA, stealing can be turned off, be allowed only between close neighbors in the NUMA-topology, or across any domain. Neighbor-stealing is the default setting in SAP HANA, as it prevents costly stealing operations from putting excessive load on socket interconnects [5]. There is also the possibility of bulk stealing by stealing multiple tasks at a time.

SAP HANA can, in addition to stealing, give the OS scheduler mandate to move worker threads between domains, as the OS is sometimes better at balancing the workload between CPUs than SAP HANA's built-in task scheduler. The mandate to move worker threads is given by different binding policies of worker threads:

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

3 RELATED WORK

Previous research on task stealing for generic workloads has investigated remote task stealing in a hierarchical way on multi-machine setups. In "HotSLAW" [20], the locality of the task to be stolen is represented using a locality-hierarchy where the stealer attempts to steal from the local region first and moves up the hierarchy to steal tasks from a less local region. The levels of the locality hierarchy can be a shared L2 cache, a NUMA domain, another processor, or another sever. For each hierarchy, the stealer randomly picks victims to steal from and only moves upwards in the hierarchy if all random attempts failed. HotSLAW uses Partitioned Global Address Space (PGAS) for giving a task access to remote memory on any server. Memory in PGAS is kept consistent across servers using software. Our scheme is slightly different in that we only use processors on the same machine. Also, memory consistency is in our case managed by the hardware at the microarchitecture level, rather than in software. We expect this to make the latency and throughput in our hierarchy different.

Research has also been done in trying to make databases distributed across multiple machines run faster using Remote Direct Memory Access (RDMA) [21]. Some technologies use traditional underlying IP based networking, and some use special protocols like InfiniBand. Recently, these types of high performance networking hardware have become more cost-competitive. The networking hardware can provide bandwidth between machines at levels similar to what is achieved in memory bandwidth within a machine. By not having to go through the CPU or OS during memory accesses across machines, RDMA can also reduce the latency to memory on a remote machine. In [21] it is shown that the transfer of 1 KB data using RDMA can finish in about the same time as a local memory request. It is mentioned, though, that on smaller data requests such as hash table lookups is hard to make RDMA as fast as a local memory access, as the network latency, and not the bandwidth, in this case is the more dominant factor. Also, work stealing is mentioned as a relevant technique for load balancing across machines. RDMA would be an interesting technology to test, but it requires additional hardware, and the higher latency would likely require modifications in how, for example, random access on columns are made. In the case of COD and SNC, the latency difference of accessing a small memory piece between local and neighbor sub-domain on the same processor is likely at a more similar scale than between local and remote machine.

Previous investigations of column and placement of scanning tasks in SAP HANA on multi-processor setups have been done by Psaroudakis [5]. In these experiments, a higher CPU load is achieved by increasing the number of rows selected by query predicates. This increases work in the more CPU-intensive materialization phase. On a balanced workload with highly selective predicates, results showed a decrease in throughput when these data intensive tasks are allowed to be stolen by remote processors. However, on less selective predicates, thus increasingly CPU-intensive, stealing gave benefits. The experiments where only done with each NUMA domain representing one processor. We extend this by looking at the impact when using sub-domains using COD and SNC.

In section 2.4, we mentioned that AMD's EPYC and potentially ARM's Neoverse processor, depending on manufacturer, are processors made out of multiple silicon dies, which can be exposed as sub-NUMA domains and be used for application optimizations. The hop across dies add latency and is likely higher than sub-domains on Haswell and Skylake. This should make improved locality with sub-domains more relevant. If one could achieve high memory throughput across dies, this would be something of interest for scanning heavy database queries. Though in this thesis, we only look at Intel's COD and SNC offering. Intel is also the first vendor offering something like sub-domains.

To the author's knowledge, no previous work has investigated task stealing within a database on sub-domains within a processor.

4 LATENCY AND BANDWIDTH WITH SUB-NUMA

To answer the first part of our research question, we first observe the change in memory latency and bandwidth between different sub-domains when using COD and SNC mode. In this chapter, we are testing both modes, and are then evaluating the potential for improving the performance of SAP HANA.

4.1 Experimental Setup

For evaluating the COD mode, we are using with two Broadwell 22-core Intel Xeon E5-2699 v4 processors interconnected with QPI with a total 66 GB of DDR4 memory. Broadwell is the successor of Haswell, and is using a shrunken transistor size. Other than that, Broadwell mostly provides the same microarchitecture as Haswell. Unlike experiments in [8] previously mentioned, we use a processor with more cores, where the cores are symmetrically distributed on the two buses, avoiding local accesses within a sub-domain to pass through a bus queue.

For SNC, we test two Skylake 28-core Intel Xeon Platinum 8180 processors, each having 56 hardware threads with a total of 28 MiB L2 cache and 38.5 MiB of L3 cache for each processor. This processor provides two memory controllers with three memory channels each [22]. The processors are interconnected by two UPI interconnects located closer to one of the sub-domains (see figure 5). In our setup, every memory channel is connected to a 16 GB DDR4 2666 memory device, giving a total memory of 197 GB. Installed is a security updated BIOS version[1] with SUSE Linux Enterprise Server 15 as OS.

To measure memory latency and bandwidth across the NUMA topology we use the *Memory Latency Checker* (MLC) tool provided by Intel [23], where one requesting CPU and its target memory is bound to different NUMA domains during sampling. MLC measures memory latency by traversing a linked list fragmented in a memory region too big to fit in cache and calculates the average time it takes to access each element in the list. To observe any latency differences within a NUMA domain, we sample the latency for each hardware thread targeting memory in each NUMA domain individually. Memory bandwidth is measured by calculating the average amount of throughput when letting all hardware threads from a domain simultaneously request memory in another domain, each sequentially scanning memory regions 16 bytes at a time. We first test with COD and SNC modes disabled, and then do the same test when these modes are enabled. Each sample is running for 10 seconds, targeting a memory region of up to 400 MB. During testing, the MLC process is given the highest scheduling priority in Linux, in order to avoid disturbances.

4.2 Results from Intel Broadwell

In table 1 and table 2, we present the average latency and its span observed from different hardware threads within each domain to memory located in different domains, as measured using MLC.

With COD, we see an average decrease of 7% and an average increase of 94% in latency to local versus neighboring sub-domain respectively. There is also a substantial increase in latency to remote processor.

Table 1: Memory access latency in nanoseconds for hardware threads in each domain. COD disabled.

Mem	**Domain 0**			**Domain 1**		
CPU	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

[1] Exact BIOS version: SE5C620.86B.0D.01.0395.022720191340

Table 2: Memory access latency in nanoseconds for hardware threads in each domain. 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

Table 3 and table 4 shows our measured MLC bandwidth. As memory is not interleaved between memory controllers when COD is enabled, each sub-domain of the processor should roughly have half the available bandwidth available. This can be seen when comparing the tables. Bandwidth per memory controller when COD is enabled is improved by 2.1-3.7% versus dropping by 38-51% for local and neighbor sub-domain. We can also see that the sum of throughput to both sub-domains on remote processor decreases by 3-8% per memory controller. Bandwidth to remote processor seems to differ slightly depending on which sub-domains on each processor is used, which is reasonable considering that the QPI interconnect is connected to the bus used by one sub-domain.

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: Bandwidth between sub-domains in MB/s. 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

4.3 Results from Intel Skylake

Table 5 and table 6 shows MLC latency measurements for our Skylake machine. We see an average 6.4% reduction in local latency when enabling SNC. This is reasonable as the CPU, target cache directory, and target memory controller are on average closer to each other. Latency to neighbor sub-domain increases slightly compared to local access without SNC, but the latency is stable when looking at the fastest and slowest hardware thread at the neighbor. The upper bound of the latency throughput within a processor is about the same both when SNC is disabled and enabled, whereas the lower bound is slightly improved with SNC. Latency to the remote processor is on average mostly unaffected. Average latency to both local and neighbor sub-domain within a processor decreases by 2.4 nanoseconds. This should mean that applications that are not NUMA-aware and randomly accesses memory on either sub-domain should see lower latency. Note that our previous measurements on older BIOS versions have had 7 nanoseconds faster local latency with SNC is disabled than the measurements shown here, and have therefore seen a lower latency improvement 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 in nanoseconds for hardware threads in each domain. 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 shows MLC bandwidth measurements on our Skylake machine. Using SNC mode should – as with COD – give us roughly half the bandwidth per sub-domain, which we also can see in the tables. Bandwidth per memory controller is on average increasing by 4.2% and 4.4% for local and neighbor sub-domain. Interesting to note here is that there is a higher increase in bandwidth to neighbor sub-domain, and therefore higher bandwidth than to local sub-domain. Bandwidth to remote processor neither increases – as the case with COD – nor decreases.

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

4.4 Evaluation

Our measured latencies and bandwidths for COD is about the same as measured by Molka [8]. Using our Broadwell processor with symmetrical sub-domains and shrink in transistor size did not seem to change the behavior of heavily increased latency and decreased bandwidth to neighbor sub-domain. The slightly improved latency and bandwidth for local sub-domain accesses seems to make COD a good fit for workloads isolated from each other such as VMs. However, as the bandwidth to neighbor sub-domain drops heavily, COD does not seem to be suitable for workloads that need to read a a large amount of data from memory regions shared between each other.

With SNC, latency and bandwidth to neighbor sub-domain are closer to what was measured on local accesses than a remote processor access. This is more reasonable than what was observed with COD, and is likely due to the different caching strategy used in Skylake. In contrast to COD, memory access to another processor seems to be mostly unaffected. Our measured bandwidth to remote processor showed about the same bandwidth with and without SNC. The UPI interconnect likely is the bottleneck here, and might have reached full capacity in both modes. As mentioned in section 2.4, memory from remote processors is mapped across all directory and L3 slices on the local processor, and a CPU in one sub-domain uses the entire L3 cache available on its processor. L3 caching of remote memory in SNC mode therefore have the same behavior as without SNC. Because of the similarities in latency, bandwidth, and the management of L3 caching of remote memory, it is reasonable to assume that the performance of larger multi-processor setups should not be affected much by the usage of SNC. As the security updated BIOS on Skylake seems to have changed some of the performance characteristics, it could be worthwhile to in future see how the baseline and SNC is comparing on the newer Intel Cascade Lake microarchitecture.

The drastically worse latency and bandwidth to neighbor sub-domain seems to make COD a bad fit for workloads accessing lots of shared memory such as SAP HANA. Because of this, we dismiss further experiments of the COD mode on Broadwell. On SNC, the more reasonable latency and bandwidth to neighbor sub-domain makes SNC a better candidate for further experiments on SAP HANA. Skylake also is the newer generation, and other processor manufacturers have introduced microarchitectures similar to it, making it reasonable to guess that future processors will use microarchitectures similar to Skylake. In the next chapter, we therefore test our Skylake machine with applying our SAP HANA workload.

5 RUNNING SAP HANA WITH SUB-NUMA

In the following chapter, we look at how different task stealing and worker binding practices affect the query latency and throughput with sub-NUMA when the query tasks are accessing columns in memory. On Skylake, we last chapter saw a reduced memory latency within a sub-domain and a reasonably higher memory latency to neighbor sub-domain, as well as an increase in bandwidth to local and neighbor sub-domain.

To answer our research question, whether query throughput can be improved by the usage of sub-domains, we have to look at the scheduling tradeoff between increased locality and load balancing between sub-domains. As SAP HANA uses task stealing for load balancing, we would like to test whether stealing across neighboring sub-domains yield the desired load balancing, or if the cost of stealing and letting a task access target memory on a neighbor sub-domain slows down query throughput. This scenario is illustrated in figure 3. Worker binding in SAP HANA gives the OS a reduced mandate to move tasks and memory between NUMA domains closer to each other. Therefore, we would also like to test whether the OS can help in adjusting the tradeoffs between locality and load balancing across sub-domains.

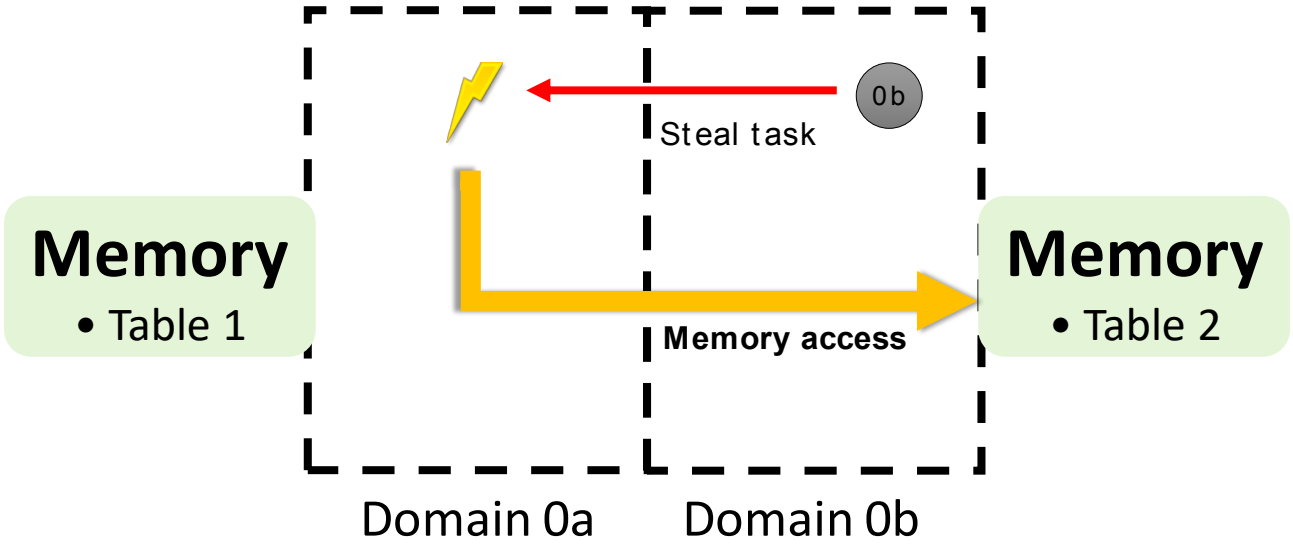

Figure 7: Stealing of a task that prefers domain 0b to neighbor sub-domain 0b.

5.1 Experimental Setup

To observe the query throughput of SNC with different scheduling practices, we create different scenarios by using the ability in SAP HANA to explicitly place table columns on specific NUMA domains, as mentioned in section 2.6. Also, we restrict all worker threads to particular CPUs. Although fixing each scenario like this is a bit artificial, it hopefully more clearly exposes the different scheduling characteristics. We also use SAP HANA's internal task scheduler settings, as previously mentioned, to change the stealing and binding behavior. As latency and bandwidth between processors did not change much with SNC, and as previous research on SAP HANA [5] saw high costs of stealing data intensive tasks across processors, we only look at the impact on load balancing within a processor. Nevertheless, multi-processor setups might still be interesting to research in future, considering the more complicated NUMA topology that is introduced with SNC.

Following our decision to limit the investigation on random row access and simple aggregations, we use tables with the schema seen in figure 8, having one indexed column `C0` for random row access queries, and six non-indexed columns `C1`-`C6` for aggregations which will require column scans. The tables contain 50 million rows, where the primary key and integer column `C0` is incremented starting from 1, and column `C1`-`C6` consist of random integers between 1 and 50 million, with possible duplicates. Randomness makes the column harder to compress. The columns use 32-bit signed `INTEGER`, but as

our random numbers are small enough not to use all these bits, the column-wise storage enables for compression. With additional overhead by SAP HANA for efficient lookup, the unique column `C0` uses about 513 MB, and column `C1`-`C6` around 276 MB each when loaded in memory. These column sizes are big enough to not fit in Skylake's caches.

```
CREATE COLUMN TABLE "TBL_X" (
       "C0" INTEGER,
       "C1" INTEGER,
       "C2" INTEGER,
       "C3" INTEGER,
       "C4" INTEGER,
       "C5" INTEGER,
       "C6" INTEGER,
       PRIMARY KEY ("C0")
)
```

Figure 8: SQL schema used for all benchmarks.

Our database client is written in C++ for giving a low footprint and uses the common ODBC driver interface for sending SQL queries to SAP HANA. The client starts 56 connections in separate OS threads, each waiting for a query to finish before sending the next one in a sequential manner. The database has 56 hardware threads available, so this will hopefully give enough parallel load. Before tests are started, every connection sends a prepared SQL statement [24]. Each query then just has to transmit any parameters that replace "?" placeholders in the prepared SQL statement, and SAP HANA only need to parse the SQL query once. The client never fetches the result from the database after the query has finished executing, in order to avoid as much additional latency in data transmission as possible. Nevertheless, as our query results are a few lines at most, this should not make a big difference. An additional thread monitors the execution by summing the query throughput for each second for all connections periodically by calling `sleep()`. The timing is expected to be slightly inaccurate as it allows the timer to drift slightly, but avoids expensive process interrupts to affect our experiments. The client process is bound to the neighboring processor in order to not interfere with SAP HANA. Local and remote throughput, as well as the average amount of instructions and cycles for each domain is also monitored by using Intel's PCM tool [25].

We start with testing a single-row access query seen in figure 9 to see the impact on latency sensitive workloads. Then we put most of our effort into trying a more complicated bandwidth-dependent aggregation query thaw will also introduce skewing, shown in figure 10. For a fair baseline, corresponding tests when SNC is disabled are designed to utilize the hardware similarly to when SNC is enabled.

Random access workload. Our single-row query is shown in figure 9, and is comparing the placeholder "?" with a random number between 1 and 50 million against the primary key on the indexed column `C0` so that any row is equally likely to be selected. Because of the index available on column `C0`, SAP HANA will use binary search, resulting in many random accesses. The resulting value from column `C1` is cross referenced in constant time from the index on column `C0`. As this query workload is rather lightweight, we only use one table copy, `TBL_0`, and place it in domain 0a as we do not need to balance workload across sub-domains.

```
SELECT "C1" FROM "TBL_0" WHERE "C0" = ?
```

Figure 9: SQL query for random access. Parameter "?" is randomly picked within the range 1 to 50 million.

We try to run this query both with and without stealing, each time running three times for 100 seconds after a warmup phase. Worker threads are bound to both sub-domains, and NUMA balancing is turned

on and off to see if Linux manage to relocate the workload better. However, NUMA balancing is in general recommended to be turned off when running SAP HANA, in order to not interfere with the built-in NUMA placement strategies. We only use the default *opportunistic binding* policy, since index lookups in SAP HANA does not use a preferred domain anyways, because the placement is less important for tasks not having a large memory throughput.

Aggregation workload. As SNC introduces a more complex NUMA topology, we need to somehow balance workload in our tests to utilize all resources on the machine. However, a perfectly balanced workload is unrealistic, and we therefore also want to observe the sensitivity of workload skewing. Our way of introducing skewing is to place identical tables on each sub-domain, and letting each query simultaneously access these to a varying degree. SAP HANA's scheduler places all preferred tasks targeting a given table on the same domain, regardless of whether the tasks access the table's columns placed in other domains. By having two distinct tables, our workaround makes it possible for the scheduler to always place a task for scanning a column on the same domain as the column. With SNC, we place table `TBL_0` on sub-domain 0a and the copy `TBL_1` on sub-domain 0b. Without SNC, both tables are placed in the same single domain for a fair baseline comparison.

The used queries, as seen in figure 10, use our table arrangement by accessing each table with `SELECT` statements, merged together using a `UNION ALL` statement. Skew is adjusted by varying the number of columns having predicates in the statement. Columns `C1`-`C6` have no index, so the predicate's less-than operator triggers a scan of the entire column. Scans on both tables are combined in the same query using `UNION ALL` as a way to easily synchronize the scans of the two tables. An OS thread in our client would then always have to wait for both table scans to finish, before executing a new query. Internally in SAP HANA, result rows for each `SELECT` statement is uniquely identified, and after that, aggregated by a `COUNT()` function as a next step. `COUNT()` avoids large result tables from being buffered in the database while waiting for some client to potentially fetch the result. We also adjust the row selectivity of each column predicate to by replacing "`range_pred`" in the query with 10 000, 100 000, or 1 000 000. This changes how large the result gets. The row selectivity of each `UNION ALL` block changes slightly when rebalancing the number of predicates, as we use an `OR` operator and more column predicates more likely select the same rows. However, as the row selectivity on each column is 2% of our 50 million rows at max, the impact in total selectivity is small.

```
(SELECT COUNT(*) FROM "TBL_0" WHERE
       "C1" < range_pred OR
       "C2" < range_pred OR
       "C3" < range_pred OR
       "C4" < range_pred
       ...
) UNION ALL
(SELECT COUNT(*) FROM "TBL_1" WHERE
       "C5" < range_pred OR
       "C6" < range_pred
       ...
)
```

Figure 10: SQL query for the scanning of two tables. The workload is skewed by adjusting the number of column predicates on each table, here marked as "…". Total amount of column predicates is always six for an easy comparison. "range_pred" is replaced before the client sends the query. In this example, two columns will be scanned on TBL_0 and four columns on TBL_1.

Each of our 56 client connections parallelly fires the exact same query 512 times sequentially, thus executing 28 672 queries in total. SAP HANA is set not to cache any query results, so the result of a query sent multiple times is always re-calculated. Within a processor, we test binding worker threads to either one of the sub-domains, both sub-domains, or half of the cores in each sub-domain being randomly

selected. By only using half the cores, one can verify if the workload is not compute-bound and it is comparable to only using one sub-domain in terms of computing resources. We continue to enable and disable stealing, and are also trying out different binding policies. NUMA balancing is turned off in every test case for less interference.

5.2 Results Using Random Access Workloads

In figure 11 we see the results of the random-access query when we use SNC and disabling and enabling stealing and NUMA balancing. With SNC, query throughput 1.2-3.6% lower in all configurations. The query seems to perform better without using SNC mode. When obtaining these measurements, it is observed that the CPUs of the clients are, in general, not overloaded.

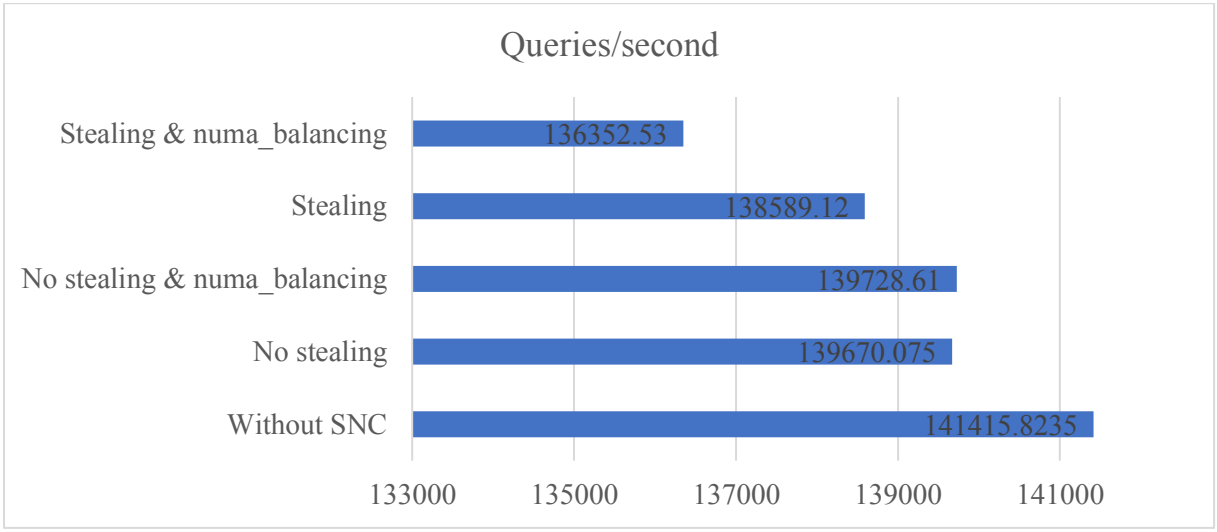

Figure 11: Query throughput when executing the query in Figure 9.

5.3 Results Using Aggregation Workloads

In the figures 12, 13, and 14, we show the baseline without SNC as well as results with SNC when rebinding worker threads in different locations, in combination with swapping stealing mode if the worker binding allows for it. In each configuration, co-located bars show a varying level of skewed predicates, from all predicates against `"TBL_0"` on the left, incrementally moving predicates to `"TBL_1"` until all predicates are against TBL_1. Only results with the *opportunistic binding* are shown in this thesis, as other binding strategies showed either similar or worse throughput. The leftmost diagrams show the query throughput and standard deviation as measured by our client monitor. The rightmost diagrams show their total memory throughput of requests. Each bar is chunked by domain, targeting memory locally in 0a, locally in 0b, remotely from 0a, and remotely from 0b, from top to bottom and as measured by PCM. During testing, we observe that some of the 56 connections get stalled so that some are finishing much earlier than others. Query throughput is nevertheless persistently kept at a similar rate, as the standard deviations show.

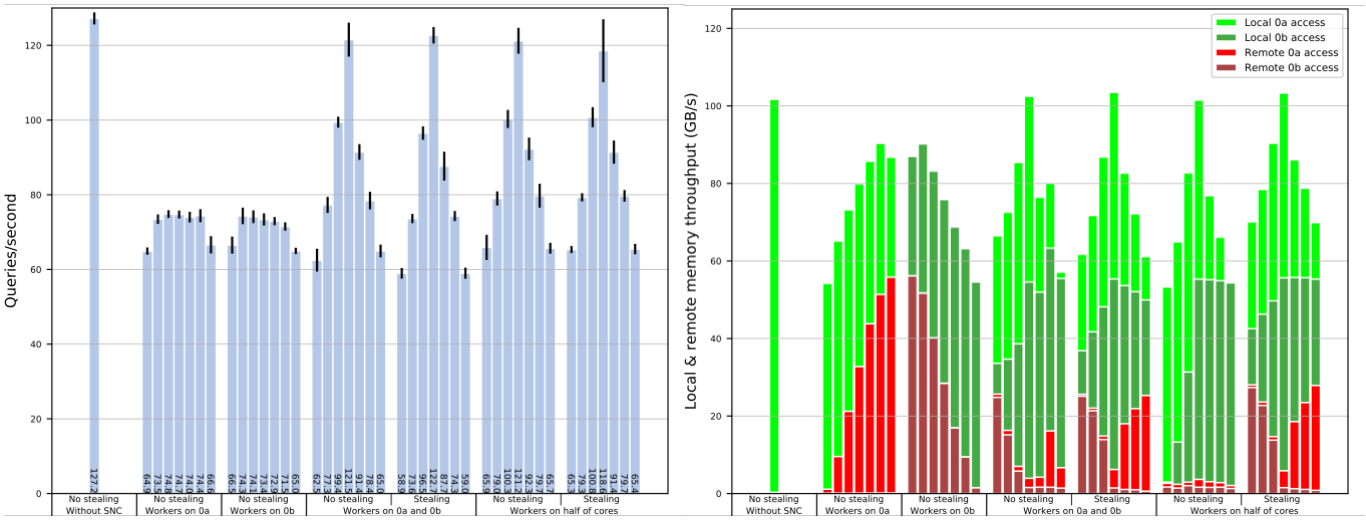

Figure 12: Aggregation workload with range_pred set to 10 000.

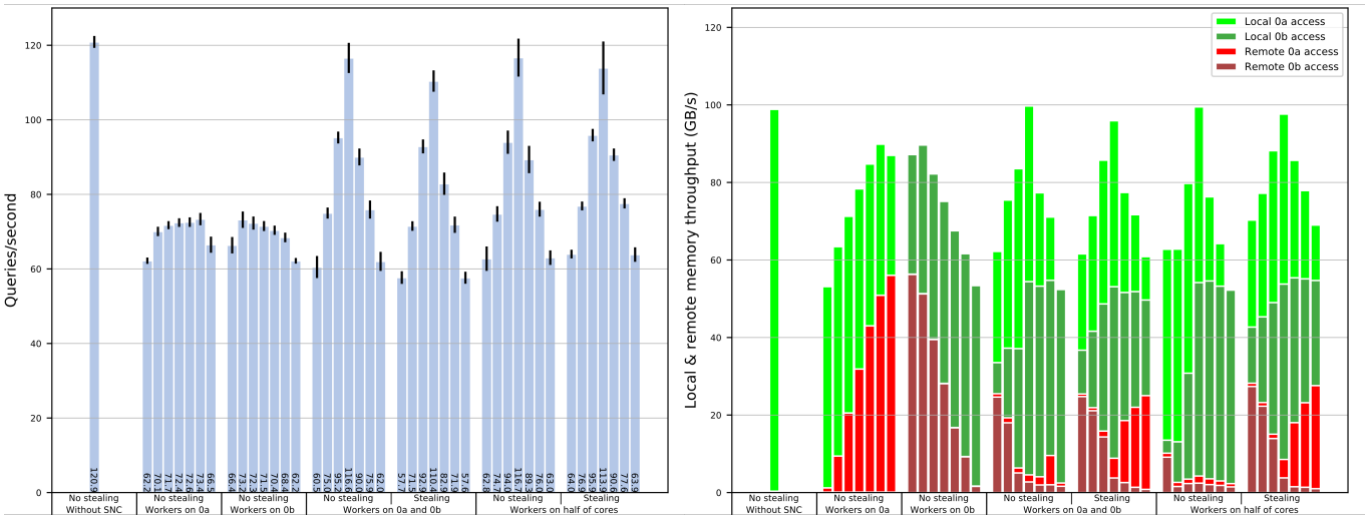

Figure 13: Aggregation workload with range_pred set to 100 000.

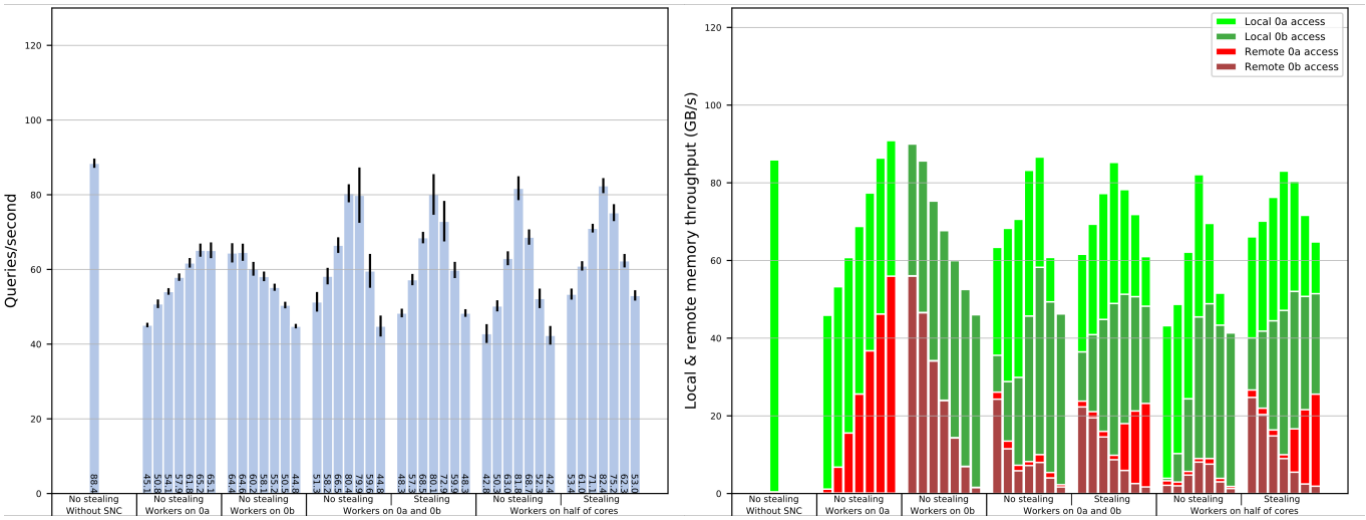

Figure 14: Aggregation workload with range_pred set to 1 000 000.

5.4 Evaluation

For evaluating our workloads, we mainly try to evaluate our SNC results by looking at the ratio of query throughput against our baseline when SNC is disabled. We also check if this ratio is similar to the memory latency and throughput measured in chapter 4.

No speedup on random access workloads. Our experiments with random access workloads did not show any improvements when using SNC, regardless of if stealing or NUMA balancing was enabled. NUMA balancing's impact could maybe have been greater if it would have been set to be more aggressive, or if the test were to run for a longer period. Stealing did also not give any benefits. This is reasonable as our simple query is not computationally heavy. As SAP HANA does not use domain preference on non-scanning tasks, it could be that tasks are not placed in the same sub-domain as the index. This experiment did not manage to show that reduced memory latency using SNC gives increased query throughput, but at least showed that SNC does not reduce the throughput of these types of queries to a high degree. As SAP HANA is not targeting random access workloads, the lower query throughput could be permitted, if other target workloads perform better.

Speedup on aggregation workloads using one sub-domain. The query throughput ratio between when predicates act only on the table at the same sub-domain as where worker threads are kept, and when SNC is disabled, is between 50.6-51.5% regardless of selectivity and stealing policy. It is reasonable that this ratio is near 50%, as when worker threads and target columns are only acting on one sub-domain, only half of the available cores and memory channels on the processor is used. Our ratio is higher than 50%, indicating a higher query throughput per core and memory channel with SNC. Memory throughput to where target columns reside also correlate with this ratio on the same tests. Similarly, local memory bandwidth of a sub-domain versus when SNC is disabled, has a ratio of 51.9-52.3%, as measured in section 4.3. So, it is likely that the increased hardware utilization is due to the slightly increased memory bandwidth with SNC.

Further speedup scanning neighbor sub-domains. Interesting to note is that query throughput is higher when worker threads scan columns on neighbor sub-domain rather than its own sub-domain. This difference gets bigger as the selectivity increases. When "`range_pred`" is 10 000, 100 000 versus 1 000 000, query throughput increases by 2.6%, 6.8% versus 44%, and aggregated memory throughput on the machine by 59%, 63% and 96% respectively. Likely this is because the higher selectivity requires a larger result table that does not fit in L3 cache. When column scans and creation of a result table are done in separate sub-domains, both sub-domains' memory channels are utilized. Additionally, as memory accesses to one sub-domain never pollute L3 caches in neighbor sub-domains, scanning memory in one sub-domain will not evict the L3 caches in the neighboring sub-domain.

Slowdown on a balanced workload. Our balanced query with three predicates each on the two tables gives an even scanning load to both sub-domains. When worker threads are bound to both sub-domains, a query throughput of 90.5-96.6% and memory throughput of 96.2-102.5% compared to without SNC is achieved, regardless of worker threads using all or half of cores; i.e. a slightly worse performance on a balanced workload with SNC. In the cases when worker threads are bound only to one of the sub-domains but are evenly scanning columns on both sub-domains, this ratio decreases to 57.6-65.9% and 75.1-80.7% respectively, which is considerably lower. This is odd, as we have the same number of cores, and remote access to neighbor sub-domain does not pollute each other's L3 caches.

Sensitive to skew. Tests when "`range_pred`" is 10 000 and 100 000 with a 2/6 versus 4/6 predicate skew between sub-domains leads to a 20.5% and 21.5% reduction in query throughput respectively. With stealing, these sensitivities are relatively unaffected. This is not very surprising, as the same tests with half the cores show similar results; i.e., the workload is not bound to compute resources, so balancing tasks does not help. For a "`range_pred`" of 1 000 000, some benefits of stealing are seen, particularly when only half of the cores are used, at which memory throughput increases as well. The higher selectivity of rows, in this case, seems to benefit from the load balancing of tasks.

6 DISCUSSION

Workload of memory access versus CPU independent. We observed that skewed workloads decrease throughput to a high degree. With a more complex NUMA topology, this becomes a greater challenge. One solution to this would be to partition data more evenly across sub-domains, rather than just at a columnar granularity. Psaroudakis is envisioning an adaptive partitioning scheme were a partition's size and NUMA domain is adjusted to rebalance CPU and memory bandwidth utilization across domains (see chapter 7 in [5]). However, our case is a little different. Sub-domains in SNC enjoy almost equally high memory bandwidth to a neighbor as local sub-domain, which is shown in section 4.3. A scanning task's CPU versus memory access workload would therefore independently be rebalanced across neighboring sub-domains; no need to move both to the same sub-domain. This is confirmed by the tests of SAP HANA in section 4.3, that showed no slowdown when scanning columns on neighbor sub-domain. Instead, this scenario showed to be a load balancing practice by itself in the tests, because of the non-competing L3 caching and memory bandwidth, between a task and its column memory request.

Potential benefits of interleaving. Continuing with the premise that bandwidth to local and neighbor sub-domain is equal, it should make sense to evenly partition data structures that are sequentially scanned, but not randomly accessed. For example, libnuma offers the possibility of interleaving memory allocations across NUMA domains, at page granularity [11]. A page is normally 4 KB. This software interleaving becomes similar to hardware interleaving like the one on Skylake without SNC, where interleaving is done on up to a 4-cache-line granularity. Four cache lines take 256 bytes. It is even more similar to what is done in quadrant mode on Intel's Knights Landing mentioned in section 2.4 in that memory is interleaved, but cache directories and its memory are closer to each other, while the requesting core can be in any of the sub-domains. It is unlikely that a software-based interleaving would beat a hardware-based one. In our previously mentioned tests, when placing worker threads on a single sub-domain that scan columns evenly, throughput drastically dropped. Some type of synchronization issue between the scanning tasks could have been a potential reason, but would have to be verified. However, if it would be possible to make software-based interleaved fast, interleaving, and non-interleaving of different data structures could be mixed on the same system.

Limited potential for improvement. We saw up to a 3.0% increase in query throughput per sub-domain when SAP HANA is running only on one sub-domain. If we reason that two fully loaded SAP HANA instances would run completely isolated from each other on separate sub-domains, SNC would also only give a 3.0% improvement. This does not seem to be very fruitful, especially if accounting for some overhead of synchronizing a single SAP HANA instance across sub-domains. Workloads with more random accesses such as index lookups might be able to achieve better performance. Our tests on single-row queries did not show any improved throughput. However, we saw with SNC an average latency reduction of 6.4% in section 4.3, so the potential is higher for improvements of single-row access patterns. SAP HANA is mainly targeting analytical customer workloads, so this improvement is not as relevant in these scenarios, but could be beneficial for other types of workloads.

Statistical validity. The measurements of latency and bandwidth using COD and SNC have been stable between multiple test runs. This is reasonable as these tests are rather simplistic. Our span in latency given between the slowest and fastest core is also reasonable, as it is expected that latency to different cores depends on where the core is placed. Measurements on the SAP HANA database is of a more dynamic nature. In our aggregation workloads, we give the standard deviation on an interval of one second in our figures. These are generally low, except for when the scanning is equally balanced between sub-domains. However, as we ran 28 672 queries in our tests with query throughput mostly below 120 queries/s, a lower deviation is seen on a larger interval. No standard deviation is given on our memory throughput, as these metrics are expected to correlate heavily with query throughput.

Sustainability and ethics. A great benefit of sub-domains is that it is already available on many of Intel's many-core processors that are used with SAP HANA. If using sub-domains would give the desired performance improvement, there are in many cases no extra cost in investing in new hardware,

as it is only a matter of configuring the BIOS. Improved usage of existing hardware would also reduce the need for larger machines, thus reducing the demand for consuming more resources and making analytics more cost efficient. This would also make large-scale analytics more democratic as it would reduce the necessity of having larger machines at hand. Also, by not using more customized hardware such as RDMA, the technology is easier to use for other actors than just major cloud providers. However, to heavily optimize the database to Intel processors, could also create a lock-in effect and make it harder to use processors from other vendors. Other vendors we have mentioned, like AMD and ARM, have similar technologies available, but one would expect the need for additional fine-tuning to see a potential performance benefit on those processors.

Limitations and future work. For our experiments, we limited ourselves to only simple random access and aggregation queries. Common join queries would also be of interest, but as we did not get good results on our simpler queries, likely more complicated queries would perform worse, considering the more complicated NUMA topology. In this thesis, we tested SAP HANA's scheduler against Skylake. Due to observed changes in latency for security updated BIOS versions, it would be relevant to also test against the newer Cascade Lake architecture. It would also be interesting to have a look at the other processor vendors offering similar sub-domain technologies. Broadwell and Skylake have all their cores on the same silicon die, and our latency measurements did show a fairly small difference in memory latency between cores. Both Infinity Fabric by AMD and ARM Neoverse, depending on the manufacturer, have cores distributed on different silicon dies within a processor. As this increases latency for accesses across dies, awareness of these dies in the form of sub-NUMA domains would be increasingly relevant for task scheduling decisions. Assuming that future processors will have even more cores, it will be harder to put all cores on one die. It would, therefore, also be relevant to evaluate processors with multiple dies in the future. It would also be of interest to have a further look on the stealing mechanisms on a multi-machine RDMA setup. With even more cores, it is reasonable to think that cores eventually would have to be distributed across multiple machines.

7 CONCLUSIONS

In this thesis, we verified the change in memory latency and bandwidth between different cores and memory devices, when enabling COD and SNC modes on Intel. COD did not show attractive latency and bandwidth for any remote access, partly due to the cache policy being used. SNC gave better latency results, and a more detailed exposure of the processor topology to the OS and its applications. This should give potential benefits for workloads with many random accesses to memory not shared between tasks on different sub-domains. Bandwidth with SNC mode to local and neighbor sub-domain both slightly increased at about the same level. Our experiments with SAP HANA using SNC did not manage to show any increase in throughput of random-access queries. Some improvement of aggregation queries per sub-domain was seen with SAP HANA, but not when concurrently putting workload on multiple sub-domains. Likely because of the design of the test query.

The greater challenge with scans on sub-domains is that memory is not interleaved and aggregated throughput not available. Without load balancing, it makes an application unable to use the total bandwidth available in a processor. Although our efforts to balance scanning workloads did not beat the hardware interleaving with SNC disabled, it could be worthwhile to in future look into if software-based interleaving can be made reasonably fast, so random access workloads simultaneously can enjoy lower latencies. Also, it would be of interest to test other processor vendor's sub-domain solutions to see if query throughput can be improved with sub-domains. In the case of SNC with SAP HANA, the potential room for improvement in query throughput is likely not big enough to motivate more complex scheduling for balancing the load between sub-NUMA domains, considering processor sizes with SNC currently available.

8 BIBLIOGRAPHY

[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. Sellami and 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 QuickPath 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 Sep 2019].

[8] D. Molka, D. Hackenberg, R. Schöne and 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. Böhm, "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 Aug 2019].

[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 Jul 2019].

[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 in Intel Processors," in *Proceedings of the Fourteenth EuroSys Conference 2019*, Dresden, 2019.

[16] A. . Sodani, R. . Gramunt, J. . Corbal, H.-S. . Kim, K. N. Vinod, S. . Chinthamani, S. R. Hutsell, R. . Agarwal and Y.-C. . Liu, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro,* vol. 36, no. 2, pp. 34-46, 2016.

[17] SAP, "SQL Statements to Apply NUMA Location Preferences," [Online]. Available: https://help.sap.com/viewer/6b94445c94ae495c83a19646e7c3fd56/2.0.04/en-US/dca7da8d692548d183605921bf20ba71.html. [Accessed 8 Jun 2019].

[18] 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.

[19] D. Vyukov, "Scalable Go Scheduler Design Doc," 2 May 2012. [Online]. Available: https://docs.google.com/document/d/1TTj4T2JO42uD5ID9e89oa0sLKhJYD0Y_kqxDv3I3XMw/edit?usp=sharing.

[20] S.-J. Min, C. Iancu and K. Yelick, "Hierarchical work stealing on manycore clusters," in *Fifth Conference on Partitioned Global Address Space Programming Models (PGAS11)*, 2011.

[21] C. Binnig, A. Crotty, A. Galakatos, T. Kraska and E. Zamanian, "The End of Slow Networks: It's Time for a Redesign," in *Proceedings of the VLDB Endowment, Vol. 9, No. 7*, 2016.

[22] WikiChip, 13 Feb 2019. [Online]. Available: https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server).

[23] Intel, "Intel Memory Latency Checker v3.6," 2018. [Online]. Available: https://software.intel.com/en-us/articles/intelr-memory-latency-checker.

[24] SAP, "Prepared Statement Caching," 16 Aug 2019. [Online]. Available: https://help.sap.com/viewer/0eec0d68141541d1b07893a39944924e/2.0.04/en-US/78f2163887814223858e4369d18e2847.html.

[25] Intel, "Intel Performance Counter Monitor - A Better Way to Measure CPU Utilization," 16 Aug 2019. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor.

TRITA -EECS-EX-2020:43