Leveraging Modern Hardware in SAP HANA
Present and Future

Dr.-Ing. Ismail Oukid (SAP)
January 21, 2019

PUBLIC
Agenda

Present

- Single Instruction Multiple Data (SIMD)
- Hardware Transactional Memory (HTM)
- Non-Volatile Memory (NVM)

Future

- Graphic-Processing Units (GPUs)
- Field-Programmable Gate Arrays (FPGAs)
- Software-Defined Memory Coherency
SIMD Single Instruction Multiple Data
Single Instruction Multiple Data (SIMD)

Scalar processing
- traditional mode
- one instruction produces one result

SIMD processing
- with Intel® SSE, AVX
- one instruction produces multiple results

SOURCE
Scalar OP
DEST
X
Y
XopY

SOURCE
SIMD OP
DEST
127
X4 X3 X2 X1
Y4 Y3 Y2 Y1
X4opY4 X3opY3 X2opY2 X1opY1
Use-Case: Pack Integers in Bit-Fields

**Example**: Packed 17-bit fields

Integers in the range \([0, 100000]\) need only **17 bits**

**Idea**: Store only 17 bits (saving 15 bits per value)
Decompress Unaligned Bit Fields (Example: Packed 17-Bit Fields)

1. Load a 128-bit segment of input data into SSE register

2. Shuffle compressed values to target “32-bit segment”

3. Shift values to align them to 32-bit boundary

4. Store uncompressed values
Search Unaligned Bit Fields (Example: Packed 17-Bit Fields)

1. Load a 128-bit segment of input data into SSE register

2. Shuffle compressed values to target “32-bit segment”

3. Parallel compare to shifted search range

4. Store result of search as bit-vector

```
F E D C B A 9 8 7 6 5 4 3 2 1 0
...
110300 65536 1772 2702 2 42
```

```
1772 2702 2 42
```

```
_mm_shuffle_epi8
```

```
2702 2702 2702 2702
```

```
_mm_shuffle_epi8
```

```
0 1 0 0
```
Full-table Scan is 1.63x faster with SSE

Source: Willhalm et al. SIMD-scan: ultra fast in-memory table scan using on-chip vector processing units. PVLDB 2009
Problem: No “vector-vector shift” in SSE

Hint: How do you implement a fast multiplication by “2”?
Solution: Use multiplication for shifting

Multiply to shift left

Shift right
## Unpacking of Bit-Fields with Intel® AVX2

<table>
<thead>
<tr>
<th>Pseudo Code</th>
<th>SSE 4.1</th>
<th>AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector load v from input array</td>
<td>movdqu x8(%r10,%rcx,1),%xmm6</td>
<td>vmovdqu xmm8, xmmword ptr[rax+rcx*1+0x11]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vinserti128 ymm9, ymm8, xmmword ptr [rax+rcx*1+0x19], 0x01</td>
</tr>
<tr>
<td>byte shuffle v</td>
<td>pshufb %xmm1,%xmm6</td>
<td>vpshufb ymm10, ymm9, ymm1</td>
</tr>
<tr>
<td>vector shift v</td>
<td>pmulld %xmm2,%xmm6</td>
<td>vpsrlvd ymm11, ymm10, ymm0</td>
</tr>
<tr>
<td></td>
<td>psrld $0xe,%xmm6</td>
<td></td>
</tr>
<tr>
<td>vector and v</td>
<td>pand %xmm0,%xmm6</td>
<td>vpand ymm12, ymm11, ymm2</td>
</tr>
<tr>
<td>vector store v in output array</td>
<td>movdqa %xmm6,0x10(%r8)</td>
<td>vmovdqu ymmword ptr[r8+0x20], ymm12</td>
</tr>
</tbody>
</table>

Double the number of data elements vs. Intel® SSE

Implementation takes advantage of new variable shift instruction
Intel® AVX2 Unpacking - Performance

Bit-field unpacking runs up to 1.6x faster on average with Intel® AVX2

Source: Willhalm et al. Vectorizing Database Column Scans with Complex Predicates. ADMS@VLDB 2014.
Intel® Advanced Vector Extensions 512

Expands register size to 512 bits

New mask registers:
- Results of comparisons
- Masking operations

New instructions:
- `vpcompressd` – extract selected DWORDs
- `Scatter` – distribute values
Using Mask Registers for Predication

if (v5<v6) {v1 += v3;}

- Instructions operate only on selected elements
- Enables vectorization of “branchy” code, e.g. processing of NULL values
HTM Hardware Transactional Memory
Locks are blocking like traffic lights

Serialize execution only when necessary

Picture idea from Dave Boutcher
Intel® Transactional Synchronization Extensions

Transactionally execute lock-protected critical sections

Execute without acquiring lock
- Expose hidden concurrency

Hardware manages transactional updates – All or None
- Other threads can’t observe intermediate transactional updates
- If lock elision cannot succeed, restart execution & acquire lock
Intel® Transactional Synchronization Extensions

No Serialization and no communication if no data conflicts

Transactions keep read and write sets in L1 cache

Lock remains free throughout

Hash Table

Acquire
Critical section
Release

Acquire
Critical section
Release

A
B
Initial Analysis: B+Tree

Intel TSX provides significant gains with no application changes

- Outperforms RW lock on read-only queries
- Significant gains with increasing inserts (6x for 50%)

Source: HPCA 2014: *Improving In-Memory Database Index Performance with Intel® Transactional Synchronization Extensions*. Tomas Karnagel et al.
Up to 2x Performance Boost in OLTP When Running SAP HANA with TSX

Figure 1. Upgrading to the Intel® Xeon® processor E7 v3 family and SAP HANA® SPS 09 (S-OLTP stress test lab results) provides incremental performance gains.¹


Disclaimer: Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system.
Analysis: TSX Aborts in Delta Storage Index

Capacity Aborts
- Algorithmic level
  - Node/Leaf Search Scan
  - Causes random lookups
- Cache Associativity Limits
  - Aborts typically before cache size limits
  - Hyper-threads share the L1 cache
- Dictionary contributes to larger footprint

Data Conflicts
- Single dictionary
- Global memory allocator

Analysis leading to improvements in future HW generations

Source: Karnagel et al. Improving In-Memory Database Index Performance with Intel® Transactional Synchronization Extensions. In HPCA 2014.
NVM Non-Volatile Memory
Non-Volatile Memory

NVM is a merging point between memory and storage

NVM writes slower than reads

Limited write endurance

Access Latency in Cycles for a 4 GHz Processor [1]

Non-Volatile Memory

Scale-up systems are constrained by the scalability limits of DRAM
→ NVM as potential remedy

Opportunities:

• Increased scalability
  o Larger memory modules means more memory available per server
• Significant cost savings
  o NVM will be cheaper than DRAM
• Improved recovery times

Challenges:

• Higher (than DRAM) latency impacting performance
• New technology, standards still evolving…
  o Means slow, phased implementation with increased complexity and uncertain timelines
Non-Volatile Memory Adoption in SAP HANA

- HANA is architected as a twin store
  - A large, read-only main part
  - A smaller, mutable delta part that is periodically merged into the main part

- Current NVM adoption in SAP HANA
  - The main part is persisted in and directly accessed from NVM

- Pros:
  - Faster restart time
  - Reduced TCO
  - Larger main memory
  - Write latency and endurance do not matter, only read latency does

- Cons:
  - Performance penalty due to the higher latency of NVM
Leveraging Intel DC PM for SAP HANA: Performance analysis

Promising results from a prototype using HW emulation:

- Significant improvements in restart time → >100x improvement measured
- Acceptable performance impact of higher latencies (0-15%)
Why is Restart Time Important?

Logging, warehousing, processing information: lifeline of companies

Information availability depends on database availability (9's)

Minimize restart time to improve database availability

<table>
<thead>
<tr>
<th>Availability</th>
<th>Annual Downtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>97%</td>
<td>11 days</td>
</tr>
<tr>
<td>98%</td>
<td>7 days</td>
</tr>
<tr>
<td>99%</td>
<td>3 days 15 hrs</td>
</tr>
<tr>
<td>99.9%</td>
<td>8 hrs 48 min</td>
</tr>
<tr>
<td>99.99%</td>
<td>53 min</td>
</tr>
<tr>
<td>99.999%</td>
<td>5 min</td>
</tr>
<tr>
<td>99.9999%</td>
<td>32 sec</td>
</tr>
</tbody>
</table>

- Each restart for an IMDB can take up to 1 hour to load TBs of data to memory.
- Dell study shows millions of dollars lost per hour due to downtime**
- Existing HA solutions increase the price exponentially for every nine

World’s first Intel Persistent Memory Demo at Sapphire 2017
Next step: Mutable data structures on NVM
NVM Programming Challenges

• Database developers are used to:
  • Ordering operations at the logical level (e.g., write undo log, then update primary data)
  • Fully controlling when data is made persistent (e.g., log durability must precede data durability)

• NVM invalidates these assumptions:
  • Little control over when data is made persistent
  • Writes need to be ordered at the system level
  • New failure scenarios

How to persist data in NVM?
Example: Array Append Operation

```c
void push_back(int val){
    m_array[m_size] = val;
    sfence();
    clwb(&m_array[m_size]);
    sfence();
    m_size++;
    sfence();
    clwb(&m_size);
    sfence();
}
```

Pros:
- Easy to use and to reason about

Cons:
- programmer must reason about the application state
  → Harder to use and error prone

### What is in NVM?

<table>
<thead>
<tr>
<th>m_size</th>
<th>m_array</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2017</td>
</tr>
<tr>
<td>1</td>
<td>2017</td>
</tr>
</tbody>
</table>

Pros:
- Low-level optimizations possible

Cons:
- Overhead due to systematic logging
- Low-level optimizations not possible
NVM Performance Implications

- Random memory access patterns significantly affected
- Sequential memory access patterns hardly affected

Need to keep DRAM next to NVM
NVM-Based Data Structures: State-of-the-Art

Literature focuses mostly on tree-based data structures
→ Failure-atomic updates
→ Reduce NVM writes
FP Tree Design Goals

- Persistence
- Fast Recovery
- Near DRAM-Performance
- High scalability

1. Selective Persistence

Volatile (DRAM)

2 5 7

1 2

3 4 5

6 7

Persistent (NVM)

L1 <- L2 <- L3 <- L4 <- L5 <- L6 <- L7

Inner nodes in DRAM for better performance

Leaves in NVM to ensure durability

Inner nodes rebuilt from leaves upon recovery in $O(#\text{entries})$

Recovery is up to 100x faster than a full rebuild
2. Unsorted Leaves

Counter

1. 3 4 7 14
   a b c

2. 3 4 7 14
   a b c

3. 3 4 5 7 14
   a d b c

4. 4 4 5 7 14
   a d b c

Sorted leaf

Unsorted leaf

Examining the implementation:

- Potential corruption!
- Many writes!

But...linear search instead of binary search
3. Fingerprinting

A fingerprint is a 1-byte hash of a key

Fingerprints act as a filter to limit the number of key probes
3. Fingerprinting

Expected number of probed keys is **one** for leaf sizes up to **64**
Hardware Transactional Memory (HTM)

Transactions keep read and write sets in L1 cache

HTM and NVM are apparently incompatible

FLUSH → Abort!
4. Selective Concurrency

Transient – Hardware Transactional Memory

Persistent – Fine-grained locking
4. Selective Concurrency: Insertion

Selective Concurrency solves the incompatibility of HTM and NVM
The FPTree is competitive with a Transient B+-Tree  
Only ~3% of data in DRAM
FPFTree Performance Evaluation: Multi-Threaded

The FPFTree scales nearly linearly
GPU Graphic Processing Unit
## GPUs: The New Supercomputers

<table>
<thead>
<tr>
<th></th>
<th>Nvidia V100 (2017)</th>
<th>IBM ASCI White (2000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Processor Cores</td>
<td>3584</td>
<td>8192 (512 nodes x 16 IBM Power3)</td>
</tr>
<tr>
<td>Double-Precision Performance</td>
<td>7.5 TeraFLOPS</td>
<td>7.2 TeraFLOPS</td>
</tr>
<tr>
<td>NVIDIA NVLink™ v2 Interconnect Bandwidth</td>
<td>2x150 GB/s</td>
<td>N/A</td>
</tr>
<tr>
<td>PCIe x16 Interconnect Bandwidth</td>
<td>2x16 GB/s</td>
<td>N/A</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>16 GB</td>
<td>6 TB DRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Power 3 with up to 16 MB L2 cache)</td>
</tr>
<tr>
<td>Max. overall data transfer speed</td>
<td>900 GB/s</td>
<td>?</td>
</tr>
<tr>
<td>Weight</td>
<td>450 gramm</td>
<td>106 tons</td>
</tr>
<tr>
<td>Energy consumption</td>
<td>300W</td>
<td>3 MW</td>
</tr>
</tbody>
</table>

---

## GPUs and DBMSs: A Story of Negative Research Results

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory</strong></td>
<td></td>
</tr>
<tr>
<td>• &lt; 1GB of memory too small for DBMS workload</td>
<td>• 16 GB of memory maybe enough for hot tables</td>
</tr>
<tr>
<td>• Allocation: host vs. device memory</td>
<td>• Memory allocation unchanged</td>
</tr>
<tr>
<td><strong>Bandwidth</strong></td>
<td></td>
</tr>
<tr>
<td>• Good Bandwidth to GDRAM, but</td>
<td>• PCIe v4 with 2x1.9 GB/s per lane still too slow</td>
</tr>
<tr>
<td>• PCIe v2 with 2x0.5 GB/s per lane as bottleneck</td>
<td>• Nvidia NVLink v2 with up to 2x150 GB/s!</td>
</tr>
<tr>
<td><strong>CPU ↔ GPU</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Programming Model</strong></td>
<td></td>
</tr>
<tr>
<td>• Hard to debug and profile</td>
<td>• Improved tool support, e.g. debugging and profiling</td>
</tr>
<tr>
<td></td>
<td>• Matured libraries</td>
</tr>
</tbody>
</table>

### Need to revisit negative GPU research results
Query Processing on GPUs

Query Engine
- GPU memory still tiny
- Research not promising:
  - Significant programming effort
  - Code duplication for relatively small gain
  - OLTP on the GPU?
  - Most DBMS code not GPU-friendly
- But promising examples in streaming and analytic scenarios, e.g. MapD, Kinetica.

Geospatial Processing
- Classical expensive UDFs, often natural mapping to GPU operations
- Geo-Clustering
- Typically Graphical User Interfaces

https://devportal.yaas.io/services/earthobservationanalysis/latest/
Beyond Query Processing

Cardinality Estimation
- Small data that rarely change, e.g. samples, sketches, histograms
- Asynchronous processing often acceptable
- Some methods are very compute intensive and GPU-friendly, e.g. Maximum Entropy, Kernel Density Models

Machine Learning Extension in the DBMS Kernel
- HANA has Machine Learning / Data Mining capabilities built in
  - Predictive Analytics Library (Clustering, etc.)
  - Tensor-Flow Integration

GPU application domain is expanding

Source: http://dl.acm.org/citation.cfm?id=2749438
Pipelined Compression


TPC-H `I_orderkey` compression rates with NVCOMP:
- RLE + Byte-packing = 3.2x
- RLE + Delta + RLE + Byte-packing = 10.6x → 2x faster data load
- TPC-H Q4 and Q21 ~10x and ~6x faster on GPU than on CPU (details in presentation video)
FPGA Field-Programmable Gate Array
Broad Architecture Possibilities

**FPGA as Coprocessor**
- On-chip
- Connected via PCIe
- Connected via UPI
- Connected via Network

**Networking**
- FPGA on NIC
- NIC on FPGA
- FPGA on Network Switch

**Storage**
- FPGA on SSD
  - e.g., Samsung’s KV SSD: [https://www.samsung.com/semiconductor/global.semi.static/Samsung_Key_Value_SSD_enables_High_Performance_Scaling-0.pdf](https://www.samsung.com/semiconductor/global.semi.static/Samsung_Key_Value_SSD_enables_High_Performance_Scaling-0.pdf)
  - FPGA as storage node*
  - FPGA as memory node*

*FPGAs may have ARM cores

Major Roadblock: Programmability
Our Pathfinding Efforts so Far Using OpenCL

Three ways an FPGA as coprocessor can be beneficial

- Accelerate an existing workload
- Offload operations to FPGA to relieve CPU resources without loss in performance
- Given a time budget, do something better than the CPU, e.g. better compression ratio

String Predicate Evaluation on FPGA

- Create a configurable select engine that is faster than a plain CPU implementation
- 80% of SAP HANA columns contain strings → Focus on string operators e.g. SQL LIKE

String Compression on FPGA

- HANA uses extensively compression
- Goal: either accelerate or improve the compression ratio without loss in speed

FPGAs have huge potential, but compelling use cases are hard to find
The Intel / Altera OpenCL Experience

The Good
- The Compiler (Quartus) gives useful warnings (although misses important ones)
- Compilation reports help identify bottlenecks
- Intel / Altera are very responsive to our feedback

The Bad
- Data has to be packed and streamed chunk by chunk to the FPGA
- Clear separation between host and device code
- Limited streaming capabilities (pipes not adequate)
- Limited debugging possibilities on the device (mainly printf)
- Obscure language semantics (e.g. how to organize load/store units, etc.)
- **No definitive how-to guide.** Knowledge only acquired through (a lot of) experience.

Hardware Limitations
- Host must actively transport data to device, thereby consuming CPU resources
- Not possible to create more than one pipe per direction
Desired Properties

- Unified host-device virtual memory
- Direct, zero-copy access to host memory from the FPGA with NUMA-like latencies
- Software-tunable hardware prefetching from host to FPGA
- Programming language with device code similar to host code (e.g. CUDA, SYCL?)
- Tool supportability, industry-wide standardization, compiler maturity
SDMC Software-Defined Memory Coherency
Motivation

Figure 4: Throughput of different atomic operations on a single memory location.

Source: Tudor et al. *Everything you always wanted to know about synchronization but were afraid to ask*. SOSP’13

Atomic Instructions Do Not Scale
Why is CC so Expensive?

HW always tracks coherency state
  • Not always necessary, e.g. read-only

HW enforces coherency on cache line basis
  • HW efficiently handles CC for cache lines, but size of “piece” of data varies

Example: HPE UV300
  - Cache-coherent domain is huge (4-32 sockets and up to 48 TB DRAM)
  - Increased memory latencies due to snooping
  - Atomic instructions prohibitively expensive at the slightest contention
Key Idea: Split the System Into Smaller Cache-Coherent Domains

Domain granularity is tunable

- HW manages coherency within a domain, SW manages coherency across domains
- No or partial hardware coherency across domains → expected latency improvement

Scale up system with scale out semantics
Prototype based on HPE Superdome Flex*

HPE announced a prototype for Software-Defined Scalable Memory based on Superflex Dome

A step towards “real” memory-centric computing with fabric-attached memory (FAM)


Gen-Z – Towards Software-Composable Hardware

- Pool of media modules (DRAM, NVM, GPU, FPGA, etc.) and SoCs
- Remote resources exposed/accessible as local resources
Conclusion

- **SIMD and HTM**
  - Well established technologies with proven benefits. Still evolving, thereby unlocking novel use cases.

- **Non-Volatile Memory**
  - Programming model and toolchain available (PMDK)
  - Hardware availability is imminent!

- **GPUs**
  - Need to revisit negative results in light of new hardware advancements
  - GPU application domain is expanding

- **FPGAs**
  - Programmability is a major issue, OpenCL toolchain not mature
  - The potential is huge: Smart NICs & Storage, encryption, compression, QP, storage/memory node, etc.

- **Software-Defined Memory Coherency**
  - A merging point between scale-up and scale-out systems
  - The first step towards software-composable hardware
Thank you.

Contact information:
Dr.-Ing. Ismail Oukid
Ismail.Oukid@sap.com

We are hiring!
- Full-time positions
- Student jobs (master/bachelor theses, internships, working student)

Email us at: students-hana@sap.com
**Benchmark Details**

SAP BW Enhanced Mixed Load (BW-EML) Standard Application Benchmark with a total of 1,000,000,000 records. Dell PowerEdge R930, 4 processors / 72 cores / 144 threads, Intel Xeon Processor E7-8890 v3, 2.50 GHz, 64 KB L1 cache and 256 KB L2 cache per core, 45 MB L3 cache per processor, 1536 GB main memory. Certification #: 2015015

SAP BW Enhanced Mixed Load (BW-EML) Standard Application Benchmark with a total of 1,000,000,000 records. HP DL580 G7, 4 processor / 40 cores / 80 threads, Intel Xeon Processor E7-4870, 2.40 GHz, 64 KB L1 cache and 256 KB L2 cache per core, 30 MB L3 cache per processor, 512 GB main memory. Certification #: 2013027

SAP BW Advanced Mixed Load (BW-AML) Standard Application Benchmark, 2B initial records. Fujitsu PRIMERGY RX4770 M3, 4 processors / 96 cores / 192 threads, Intel Xeon Processor E7-8890 v4, 2.20 GHz, 64 KB L1 cache and 256 KB L2 cache per core, 60 MB L3 cache per processor, 1024 GB main memory. Certification #: 2017012

SAP BW Advanced Mixed Load (BW-AML) Standard Application Benchmark, 2B initial records. Fujitsu PRIMERGY RX4770 M2, 4 processors / 72 cores / 144 threads, Intel Xeon Processor E7-8890 v3, 2.50 GHz, 64 KB L1 cache and 256 KB L2 cache per core, 45 MB L3 cache per processor, 1536 GB main memory. Certification #: 2016049

SAP* BW edition for SAP HANA* Standard Application Benchmark* @ 1.3 billion (1.3B) initial records result published at [http://global.sap.com/solutions/benchmark](http://global.sap.com/solutions/benchmark) as of 11 July 2017 Huawei FusionServer RH5885H V3, 4 processor / 96 cores / 192 threads, Intel Xeon Processor E7-8890 v4, 2.20 GHz, 64 KB L1 cache and 256 KB L2 cache per core, 60 MB L3 cache per processor, 2048 GB main memory. Certification #: 2017004

**SAP** BW edition for **SAP** HANA** Standard Application Benchmark** @ 1.3 billion (1.3B) initial records result published at [http://global.sap.com/solutions/benchmark](http://global.sap.com/solutions/benchmark) as of 11 July 2017. 4x Intel® Xeon® Platinum 8180 processor (112 cores/224 threads) on HPE CS500 (DL560 Gen10) with 3072 GB total memory on **SUSE** Linux Enterprise Server 12 using SAP HANA 1.0, SAP NetWeaver 7.50. Benchmark: SAP BW for **SAP** HANA @ 1.3B initial records, Source: Certification #: 2017025: