Relational databases on many-core processors

Xuntao Cheng

Supervisor: Associate Professor Bingsheng He

Contents

- Background
  - Relational databases
  - Many-core processors
- Related work
- Hash join - A Study of Main-Memory Hash Joins on Many-core Processor: A Case with Intel Knights Landing Architecture
- Hash table - Deployment of hash tables on many-core architectures with die-stacked High Bandwidth Memory
- Query processing - Many-core needs fine-grained scheduling: A case study of query processing on Intel Xeon Phi processors

Relational databases

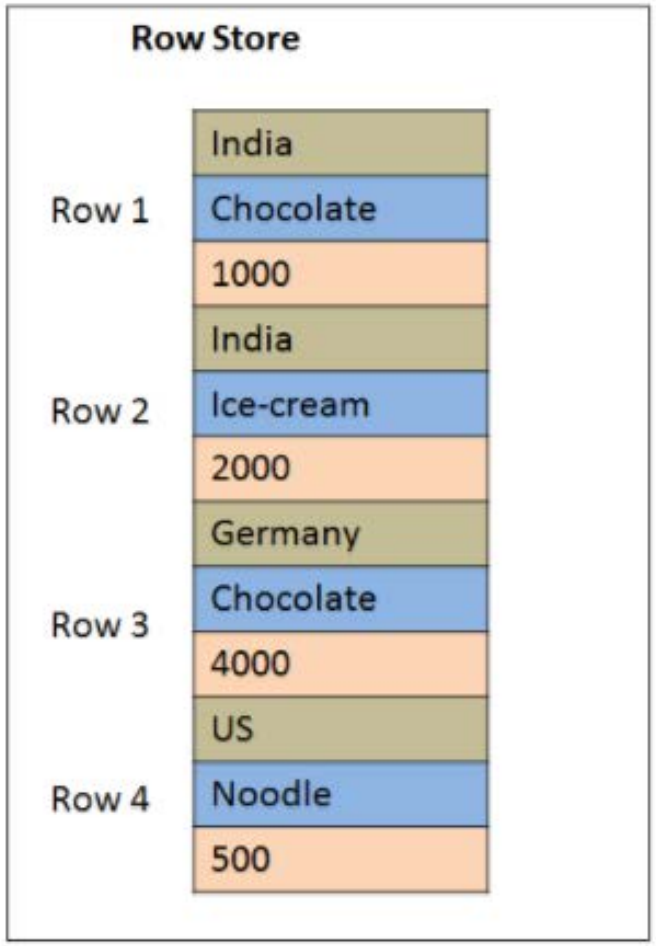
- Invented by Edgar Codd of IBM in 1970.
- Organizes data into tables (or relations):
  - Rows in the table are records.
  - Columns are attributes.
  - Each row has an unique identifier, called a key.
- Mostly use SQL (Structed Query Language) for querying and maintaining the database.
  - E.g., “select * from student where age>18”
- Common operations:
  - Selection, Sort, Join, Group-by, Aggression …

Storage layout

- Row-store: stores records in the sequence of rows
- Column-store: stores records in the sequence of columns

Table

	Country	Product	Sales
Row 1	India	Chocolate	1000
Row 2	India	Ice-cream	2000
Row 3	Germany	Chocolate	4000
Row 4	US	Noodle	500



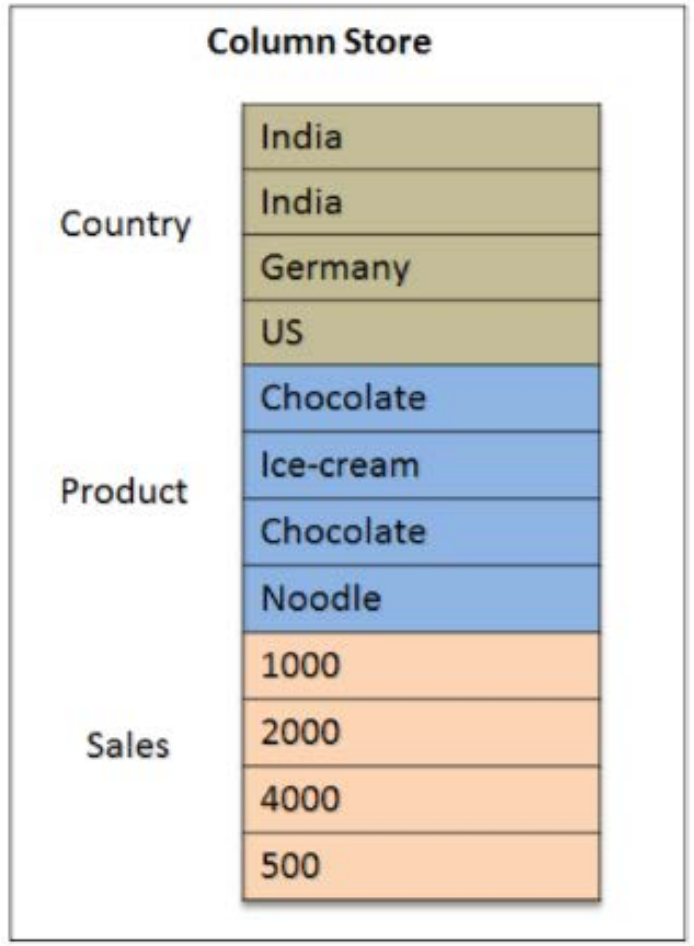


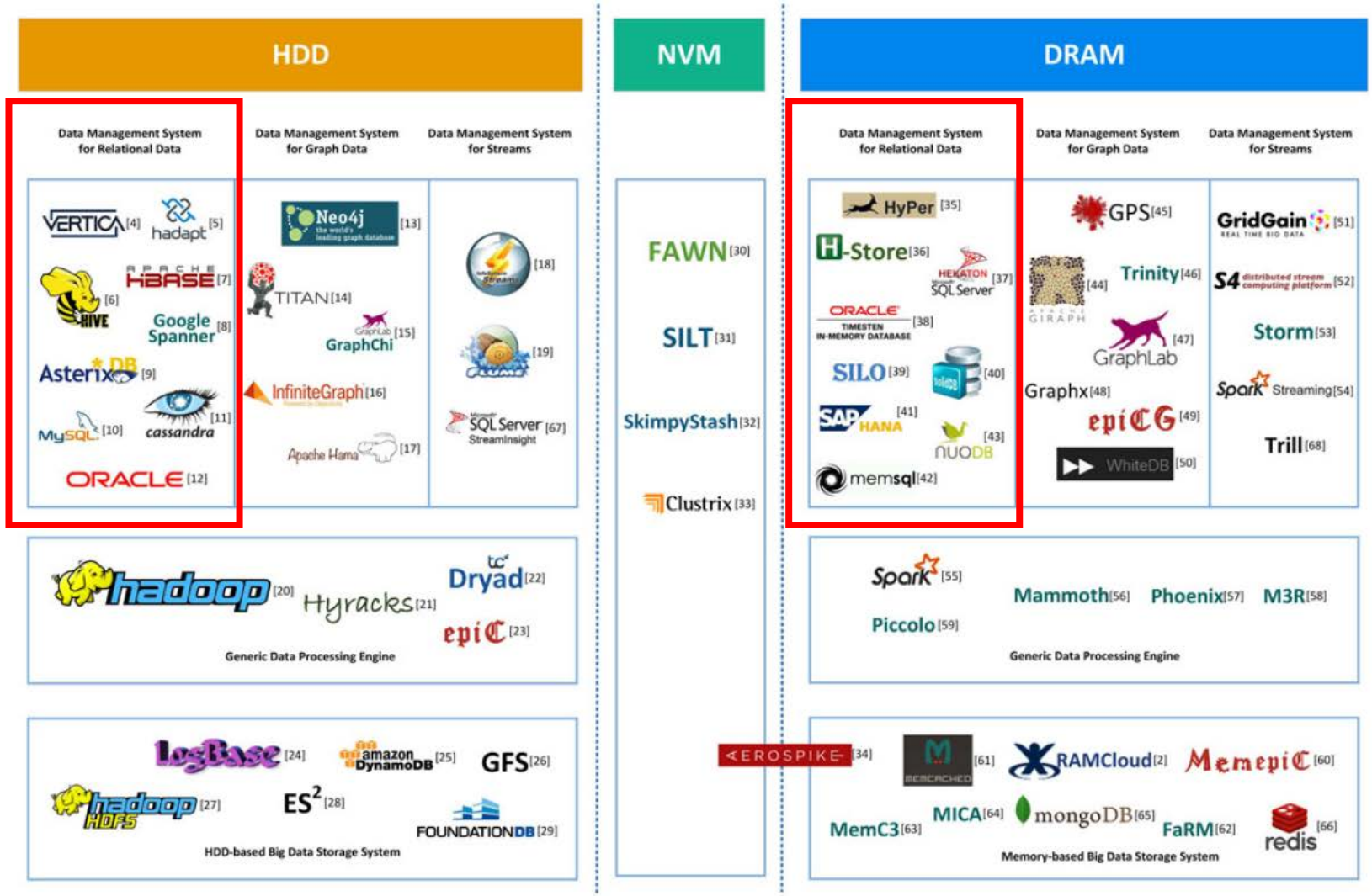
Figure from saphanatutorial.com

Some impacts of the storage layout

- Column-store makes compression easier.
- Column-store enables sequential memory accesses on columns.
- Operations on columns in a column-store can be executed in parallel naturally.
- Row-store is better, when:
  - A query only accesses a few records.
  - A query needs to access complete records, or most attributes.

Some other research issues

- Online Analytical Processing: to answer queries
- Online Transaction Processing: to update a database
- Scalability
- Query response time
- Throughput
- Data storage
- Security
- Visualization
- Interaction
- …



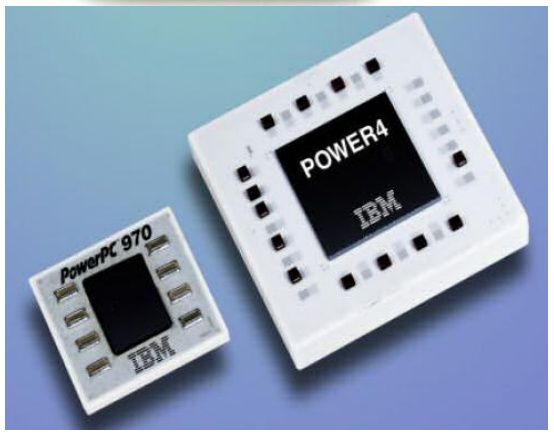
H. Zhang, G. Chen, B. C. Ooi, K. L. Tan and M. Zhang, "In-Memory Big Data Management and Processing: A Survey," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920-1948, July 1 2015.

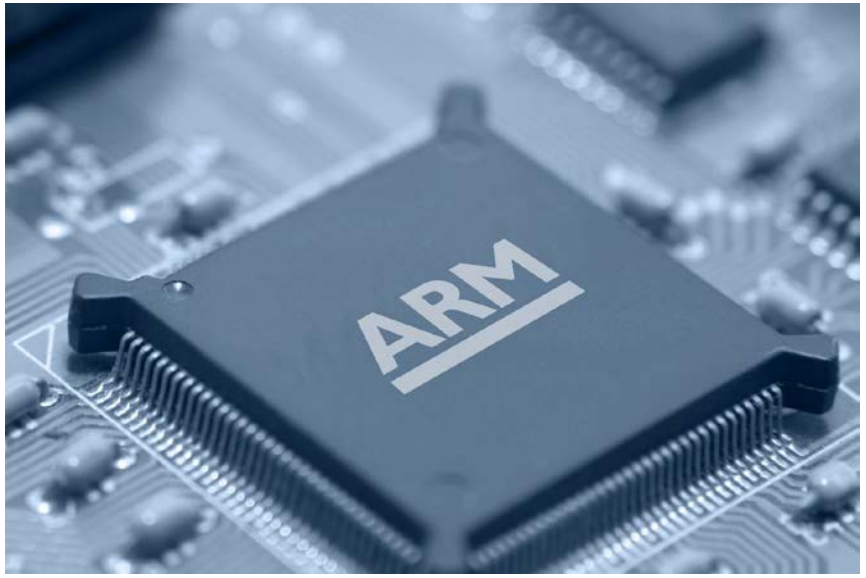
Processors

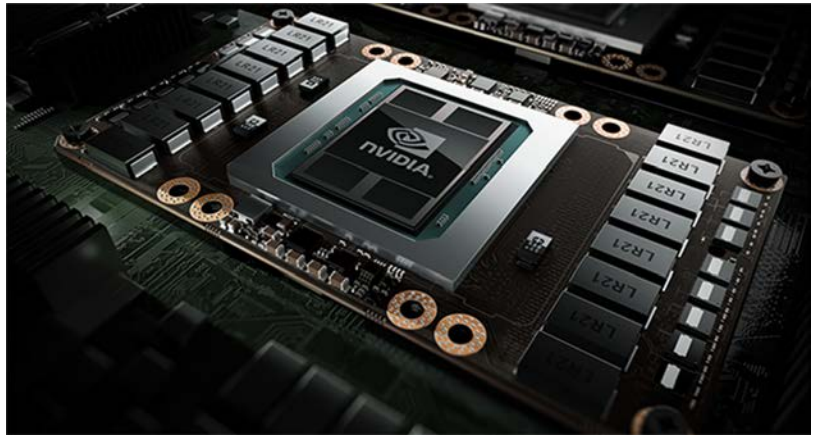

intel
Intel® Xeon®
Processor
E3-1200 V6


AMD
OPTERON

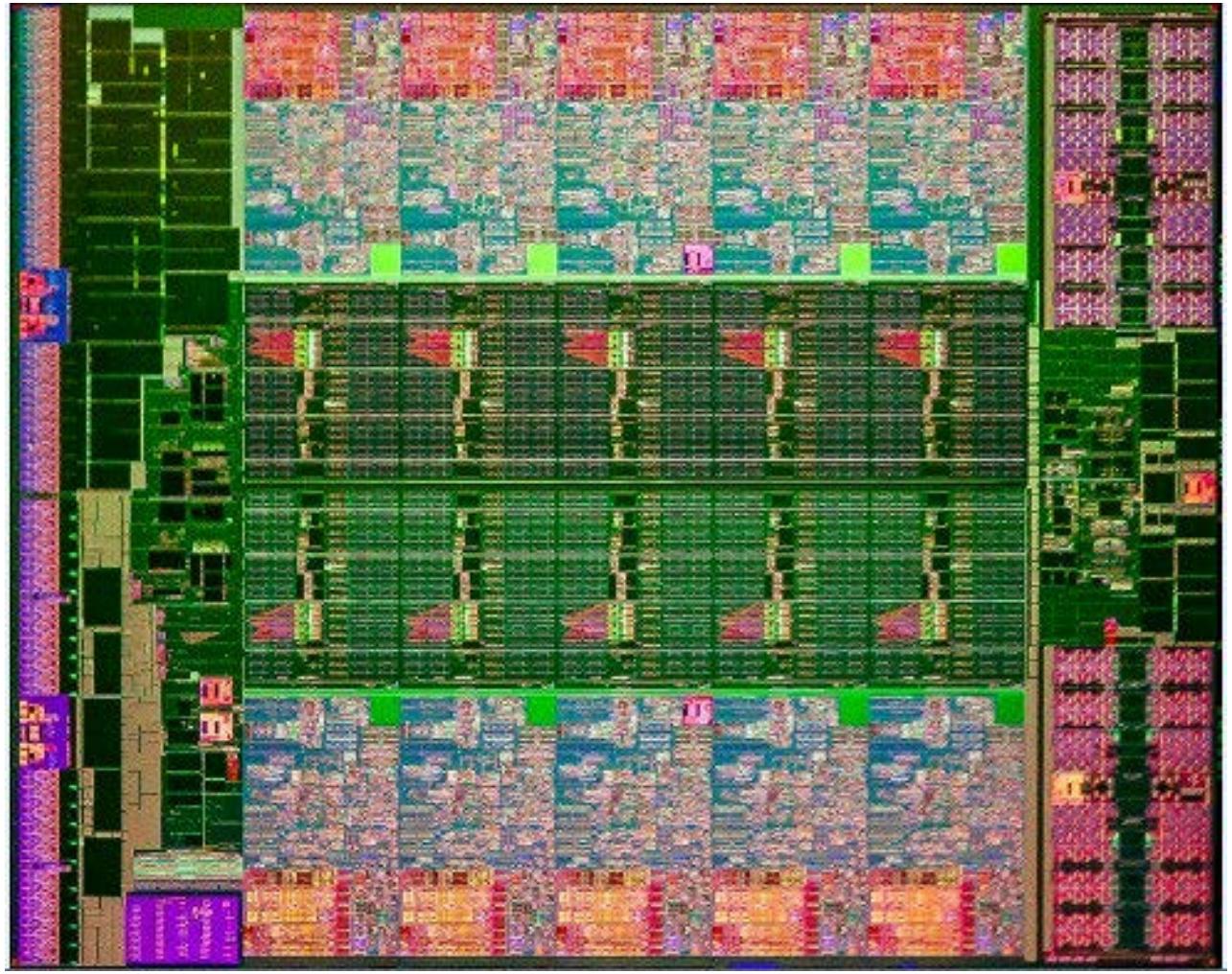

intel
i386
A80386DX-16
SX213
L9311389
INTEL © '85


POWER4
IBM
PowerPC 970
IBM

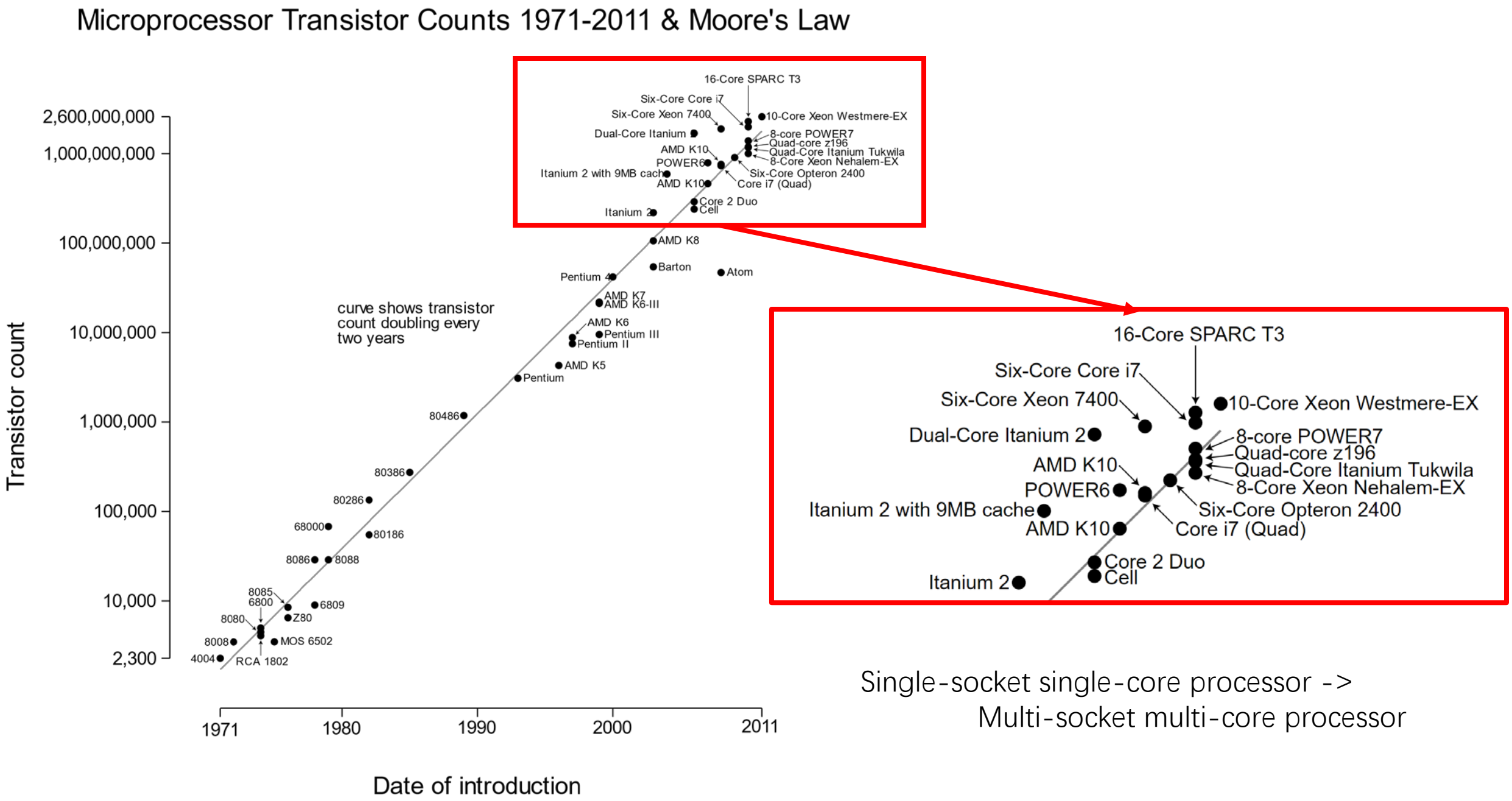

ARM


NVIDIA

CPU



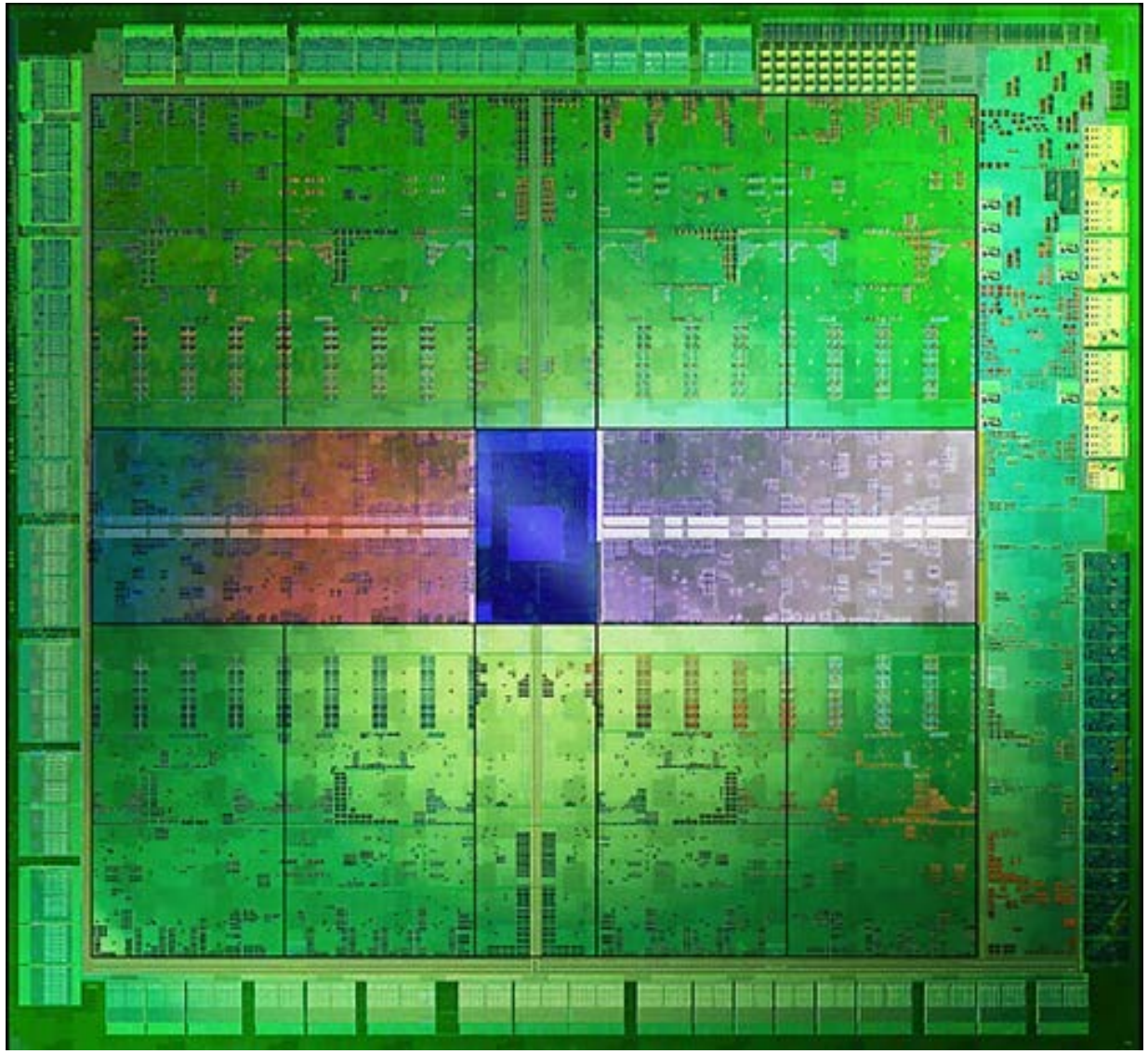
Xeon E5-2600 CPU
Ten-core



Single-socket single-core processor ->
Multi-socket multi-core processor

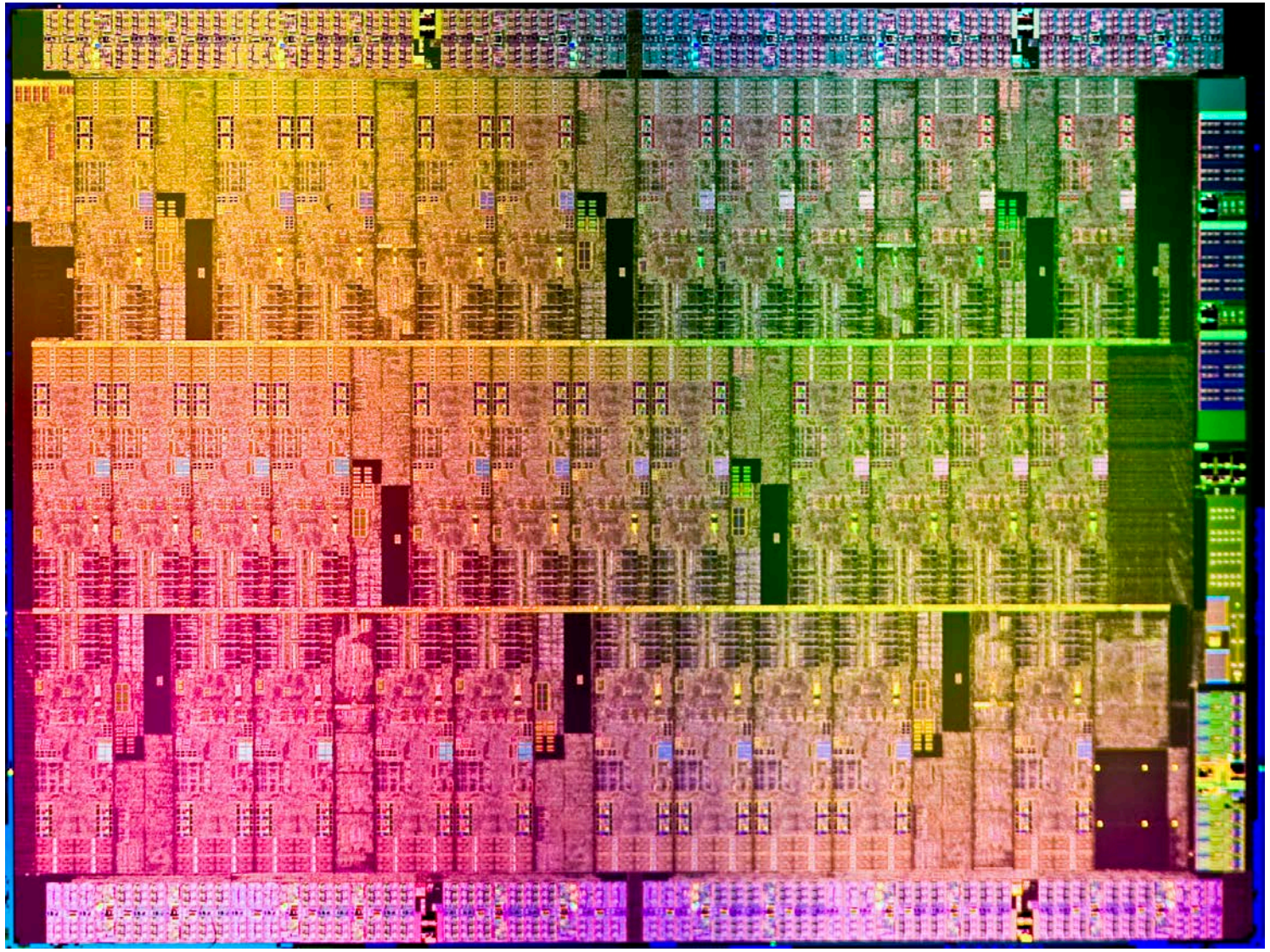
https://en.wikipedia.org/wiki/Moore%27s_law

GPU



The die shot of the NVIDIA GeForce GTX 680
1536-core

Xeon Phi

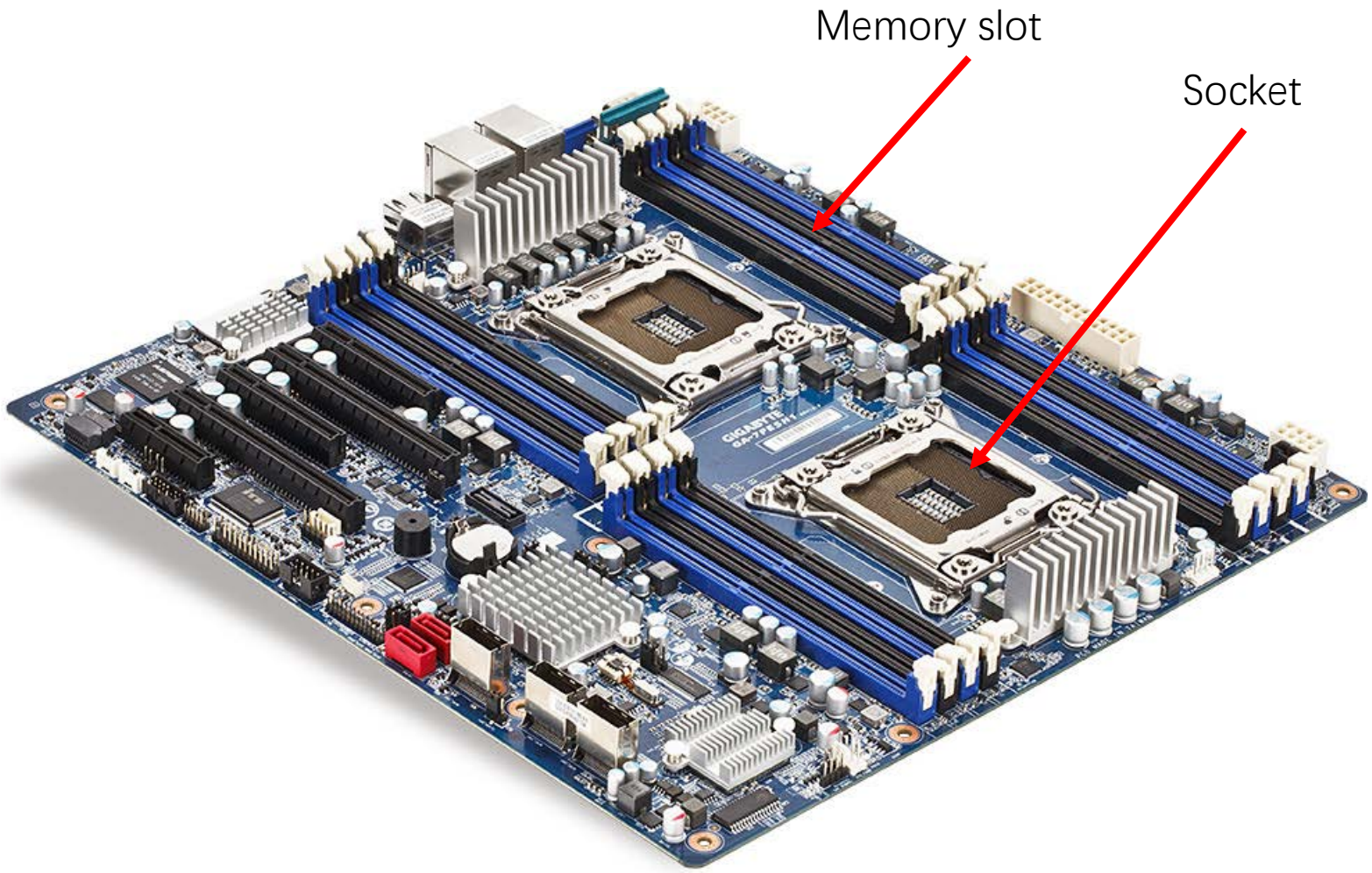


Intel Xeon Phi
72-cores

Sockets

A socket is
a connector on the motherboard that houses a CPU.

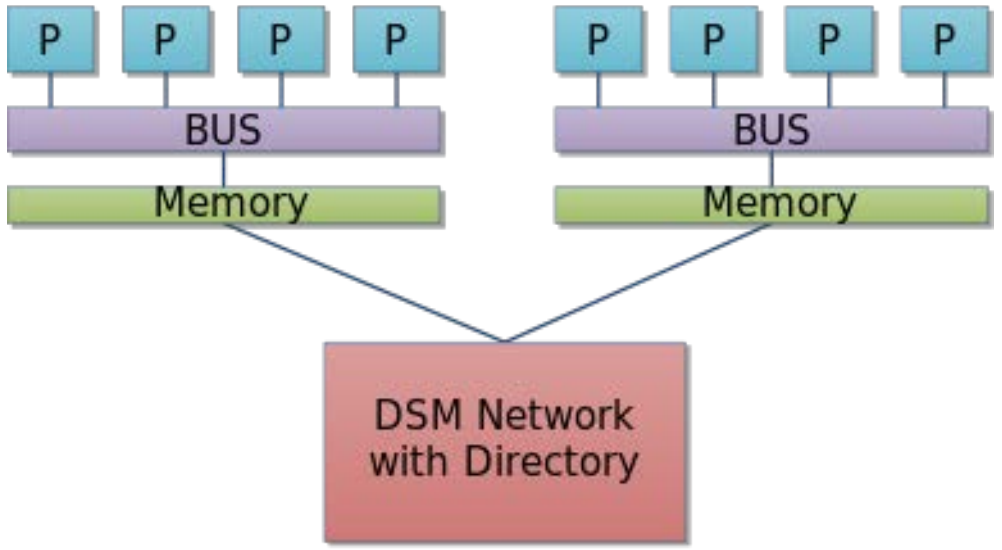
It forms the electrical interface and the contact with the CPU.



Gigabyte GA-7PESH1
A dual-socket motherboard

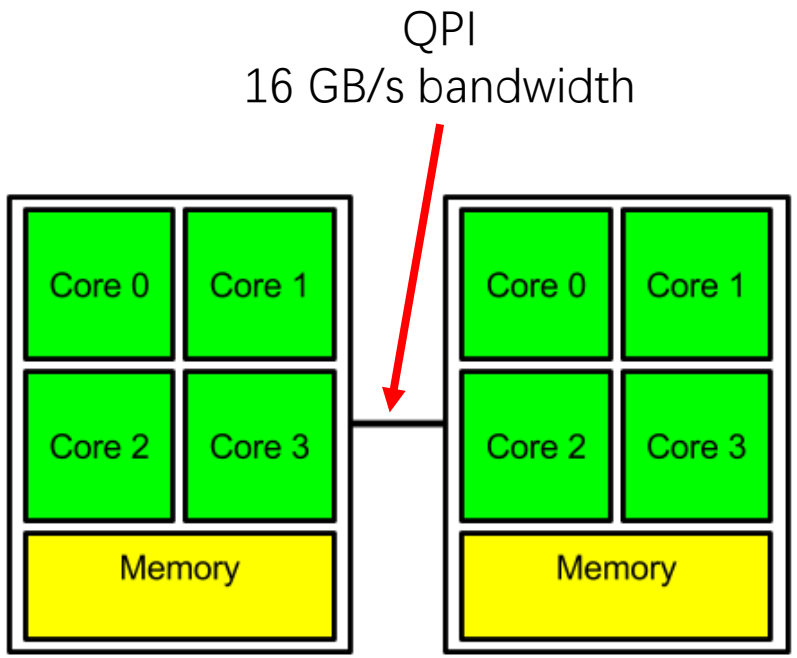
Non-uniform memory access (NUMA)

- In multi-processing, there are multiple processors and memories.
- The time of memory accesses depend on the memory location relative to the processor.

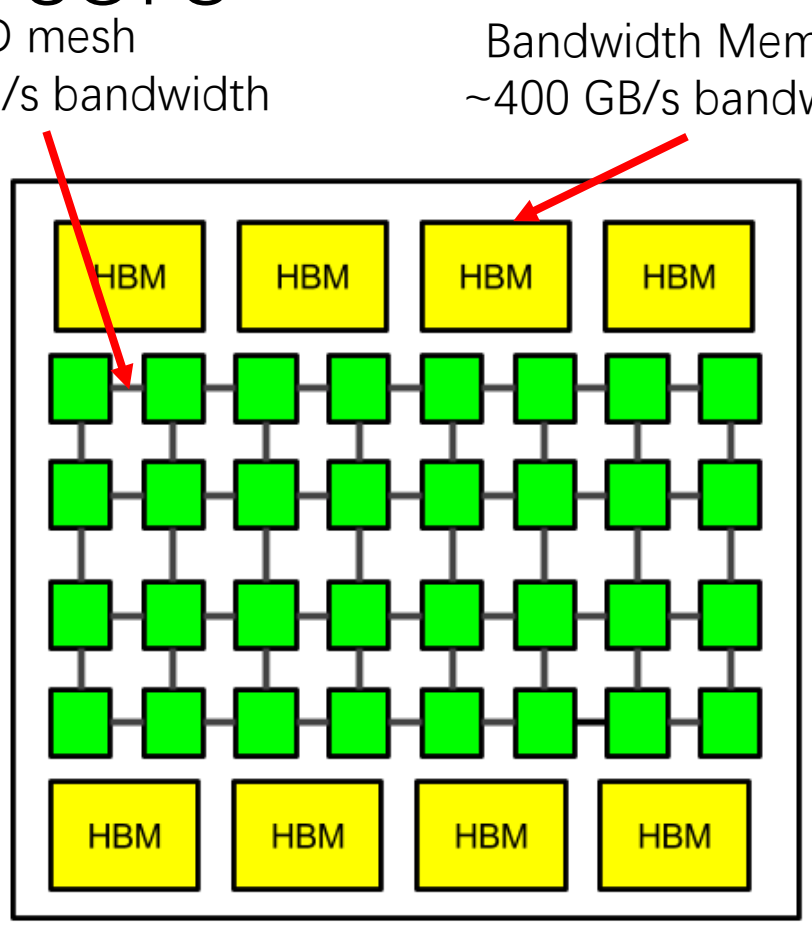


A NUMA architecture

Multi-socket vs. Many-core



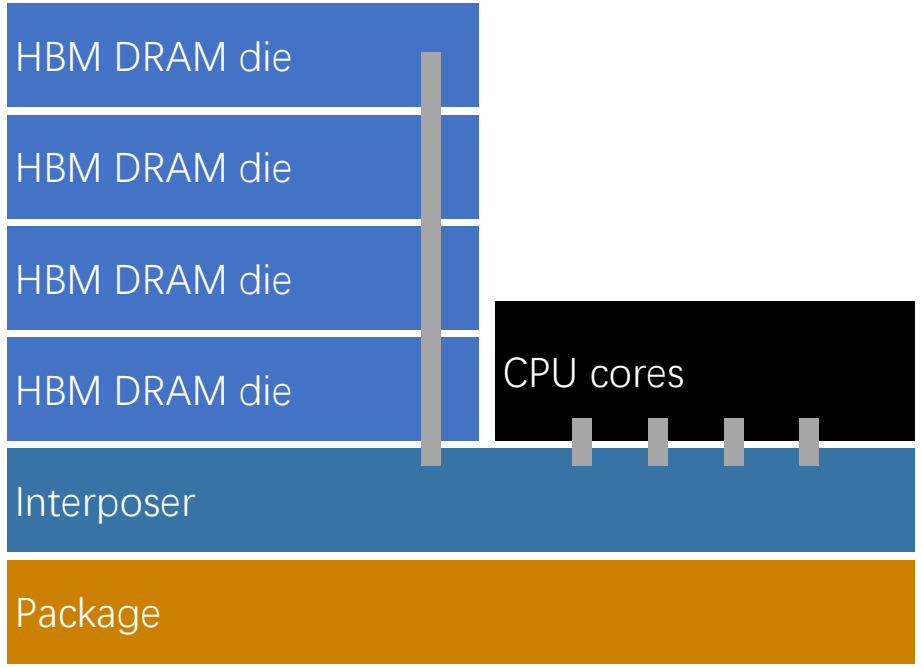
Multi-socket multi-core CPUs



Many-core CPU
Intel Xeon Phi Knights Landing
(KNL)

Die-stacked high-bandwidth memory

- Stacked or interposed on the same package with the CPU
- Up to 512 GB/s memory bandwidth
  - >5X higher than conventional DDR main memory
- 16 GB memory capacity
  - Constrained by area and power limits

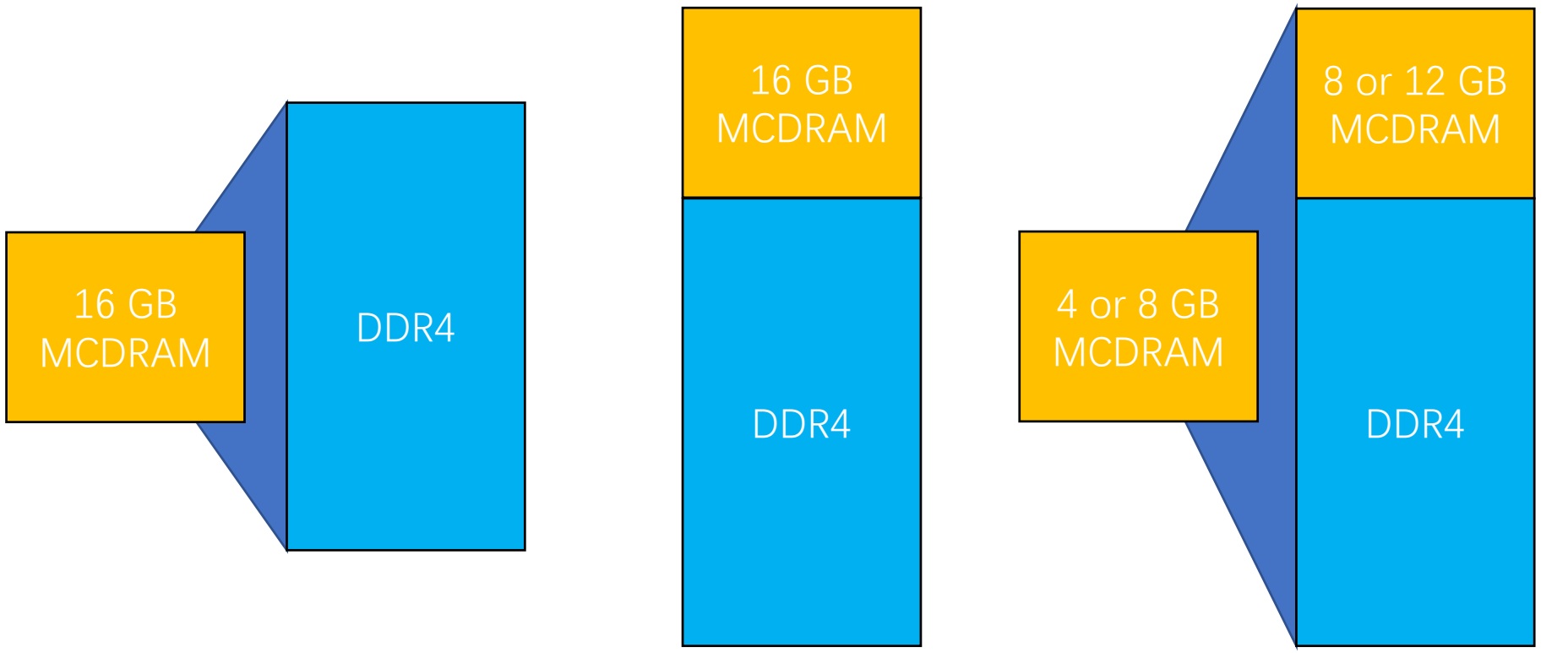


Interposing HBM dies on Intel KNL

What it looks like for real.

Intel Xeon Phi Knights Landing

Multi-channel DRAM (MCDRAM)



Cache mode
Direct-mapped cache

Flat mode
Part of the memory space

Hybrid mode
Part cache, part memory

Trends in Computer Architectures

- Processing Units
  - Multi-core & many-core processors
  - General purpose processors
  - Special purpose ASICs
- Non-uniform Memory Acce
  - Local vs. remote
  - Interconnect
- Die-stacked DRAM
  - 3D stacking
  - 2.5D interposing

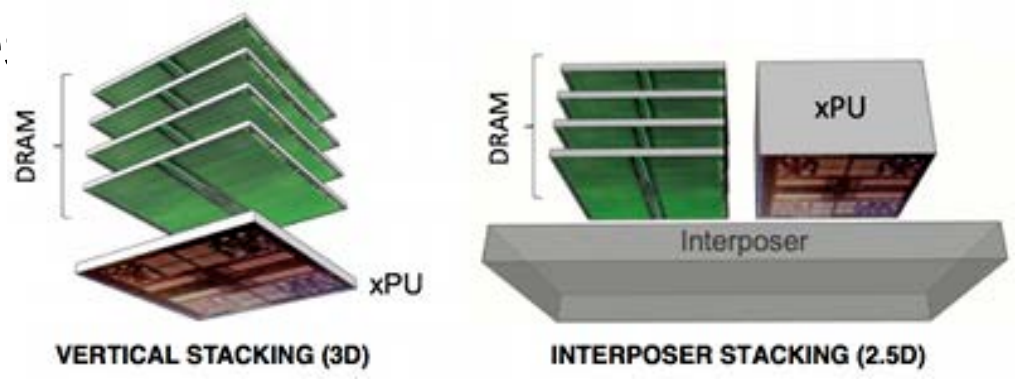


Figure from 3DInCities.com

Research Questions

- Efficiency of the state-of-the-art software optimizations
  - Hardware-conscious tuning [Balkesen et al, ICDE'13]
  - NUMA-aware optimizations [Schuh et al, SIGMOD'16]
  - Vectorization [Polychroniou et al, SIGMOD'15], ···
- Exploiting new hardware for performance improvements
  - High-bandwidth MCDRAM
  - 2D mesh
  - Many cores
  - Configurable modes
- Impacts of optimizations on different algorithms
  - Simple hash join (SHJ)
  - Partitioned hash join (PHJ)

Relational Joins on Graphics Processors

- A join is a frequently used means to combine columns for one or more tables by using values common to each.
- This paper presented novel join implementations on GPUs.
  - Design a set of data-parallel primitives: split, sort, etc.
  - Combine primitives to form join algorithms: indexed, non-indexed nested loop, sort-merge and hash joins.
- The performance is 2-7 times higher than the optimized CPU-based counterparts.

Bingsheng He, Ke Yang, Rui Fang, Mian Lu, Naga Govindaraju, Qiong Luo, and Pedro Sander. 2008. Relational joins on graphics processors. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (SIGMOD '08). ACM, New York, NY, USA, 511-524.

Revisiting co-processing for hash joins on the coupled CPU-GPU architecture

- Coupled CPU-GPU architectures share the same memory, saving the costs of data transfer between the CPU and the GPU.
- This paper proposed a ***fine-grained*** scheduling approach to divide the work between two processors.

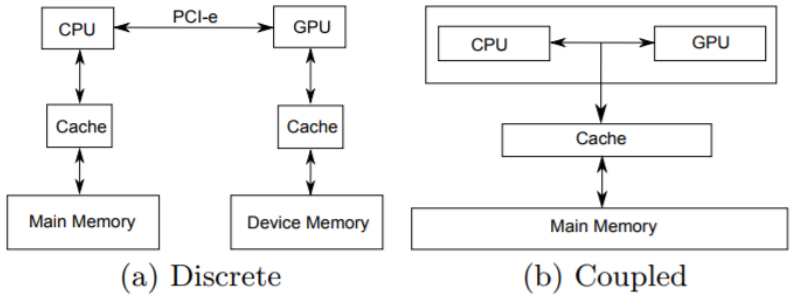


Figure 1: An overview of discrete and coupled CPU-GPU architectures

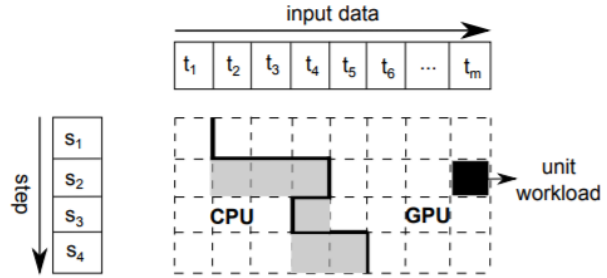


Figure 2: Fine-grained co-processing algorithm on a series of steps.

Jiong He, Mian Lu, and Bingsheng He. 2013. Revisiting co-processing for hash joins on the coupled CPU-GPU architecture. *Proc. VLDB Endow.* 6, 10 (August 2013), 889-900.

Rethinking SIMD Vectorization for In-Memory Databases

- Intel Xeon Phi has so many cores with 512-bit Vector Processing Units (VPU) on each core. SIMD can boost the performance of query processing greatly.
- However, it is challenging to vectorize computations with data races.
- This paper proposed a novel approach to address this using gather/scatter SIMD operations.
- They vectorized many database operators on Xeon Phi.

Orestis Polychroniou, Arun Raghavan, and Kenneth A. Ross. 2015. Rethinking SIMD Vectorization for In-Memory Databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (SIGMOD '15). ACM, New York, NY, USA, 1493-1508.

Adaptive work placement for query processing on heterogeneous computing resources

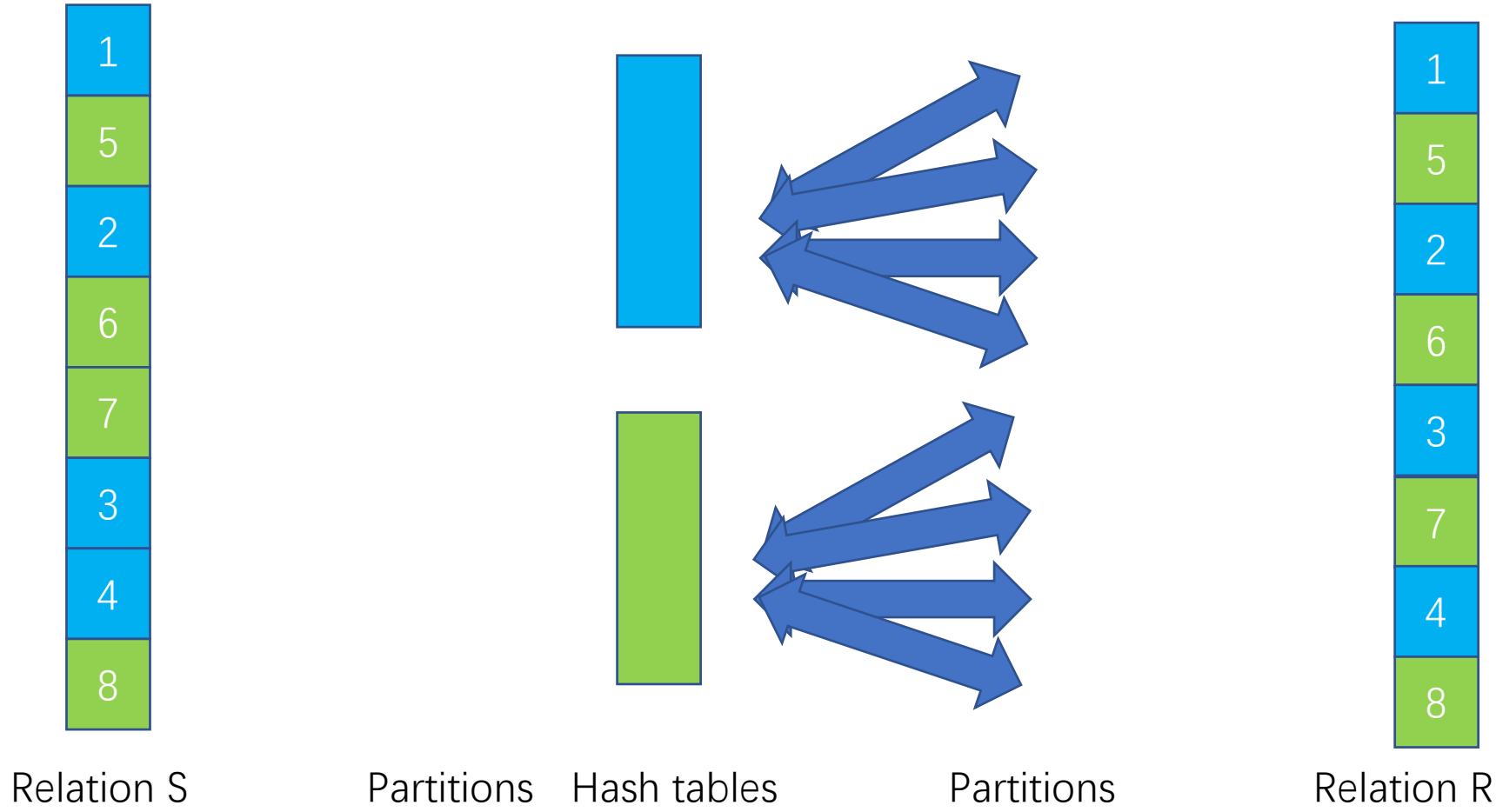
- Homogenous multi-core systems -> Heterogenous systems (CPU + GPU + FPGA + Xeon Phi …)
- The challenge is to place the right work on the right computing unit.
- This paper proposed an adaptive placement approach that addresses
  - Inaccurate cardinality estimation
  - Inaccurate runtime estimation
  - Influence of intermediate result location

Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. 2017. Adaptive work placement for query processing on heterogeneous computing resources. *Proc. VLDB Endow.* 10, 7 (March 2017), 733-744.

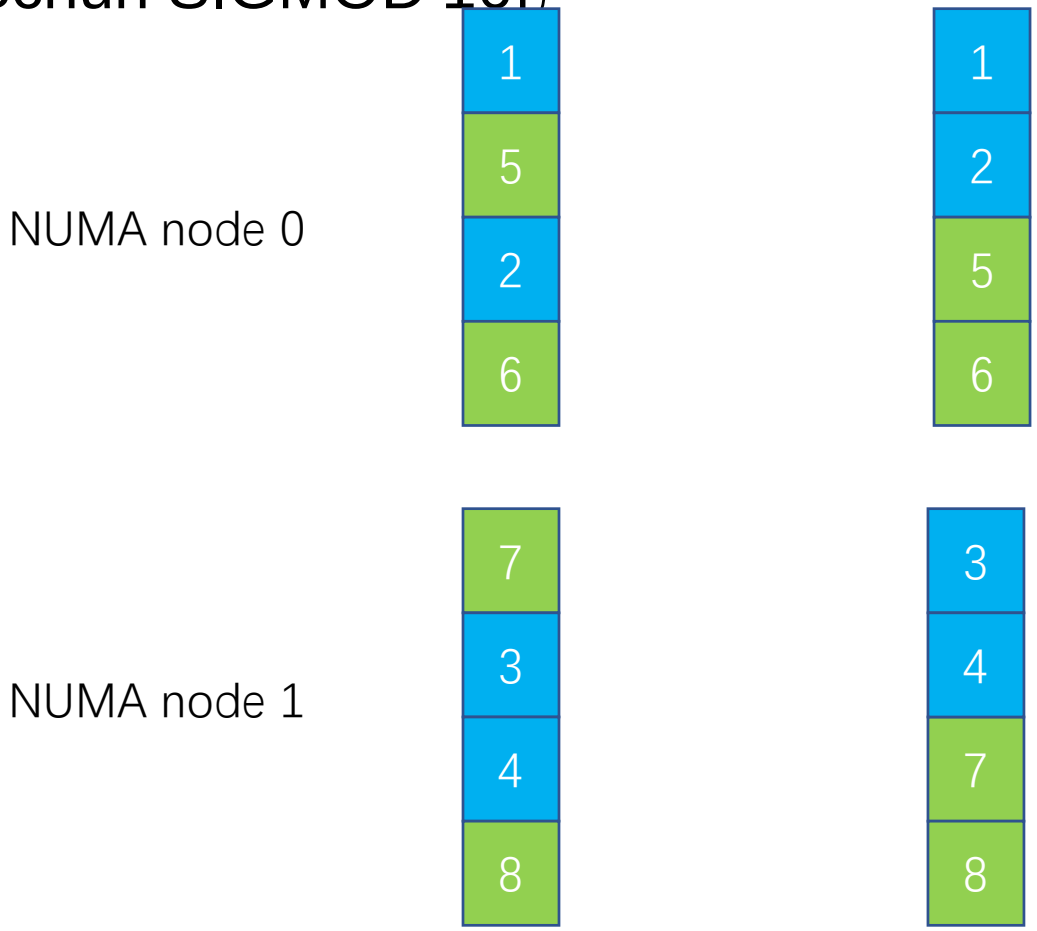
A Study of Main-Memory Hash Joins on Many-core Processor: A Case with Intel Knights Landing Architecture

Xuntao Cheng, Bingsheng He, Xiaoli Du, Chiew Tong Lau

Overview of Partitioned Hash Join (R⋈S)

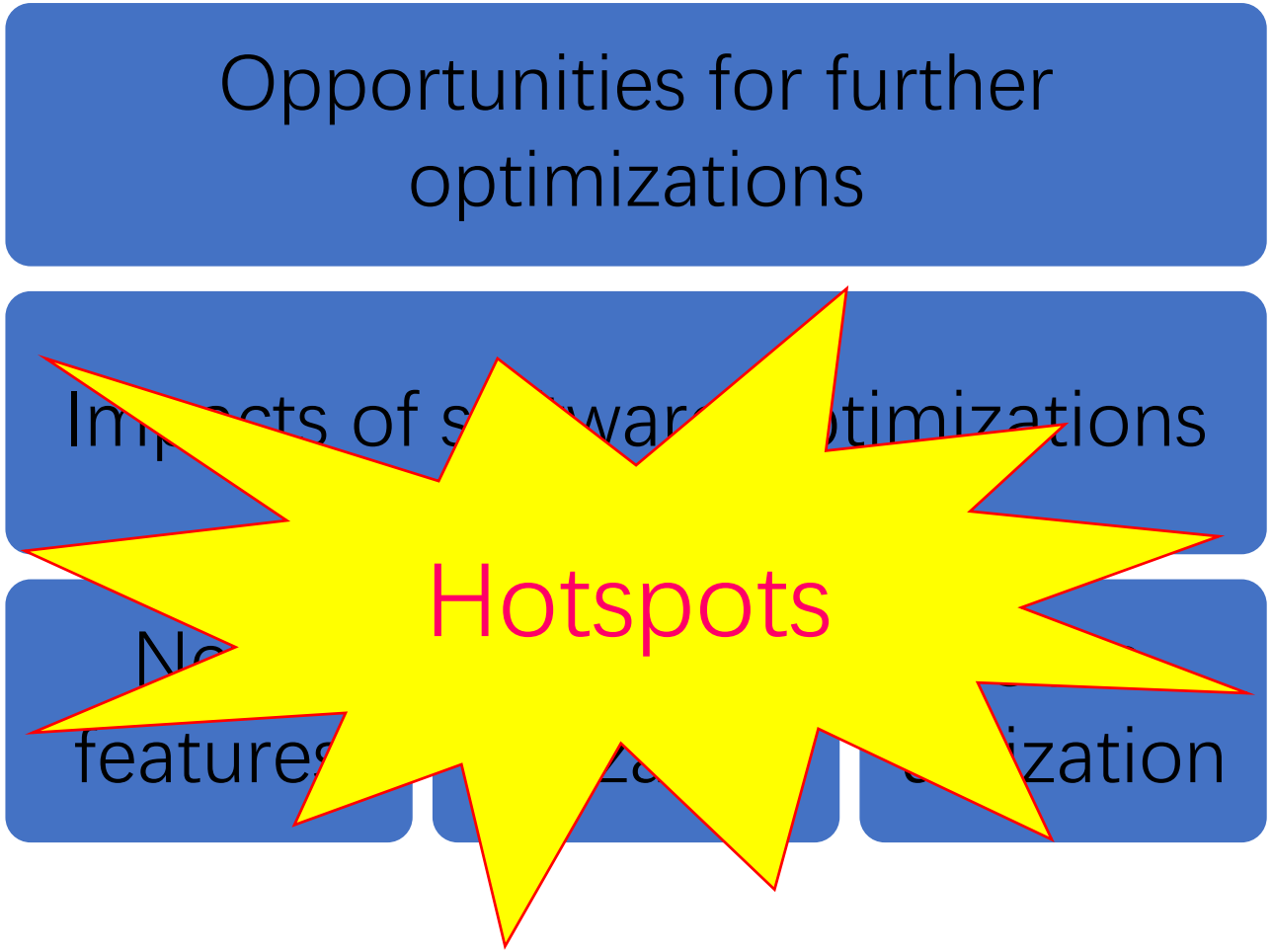


NUMA-aware Partitioning (Chunked Parallel Radix Join, CPRA, **[Schuh SIGMOD'16]**)

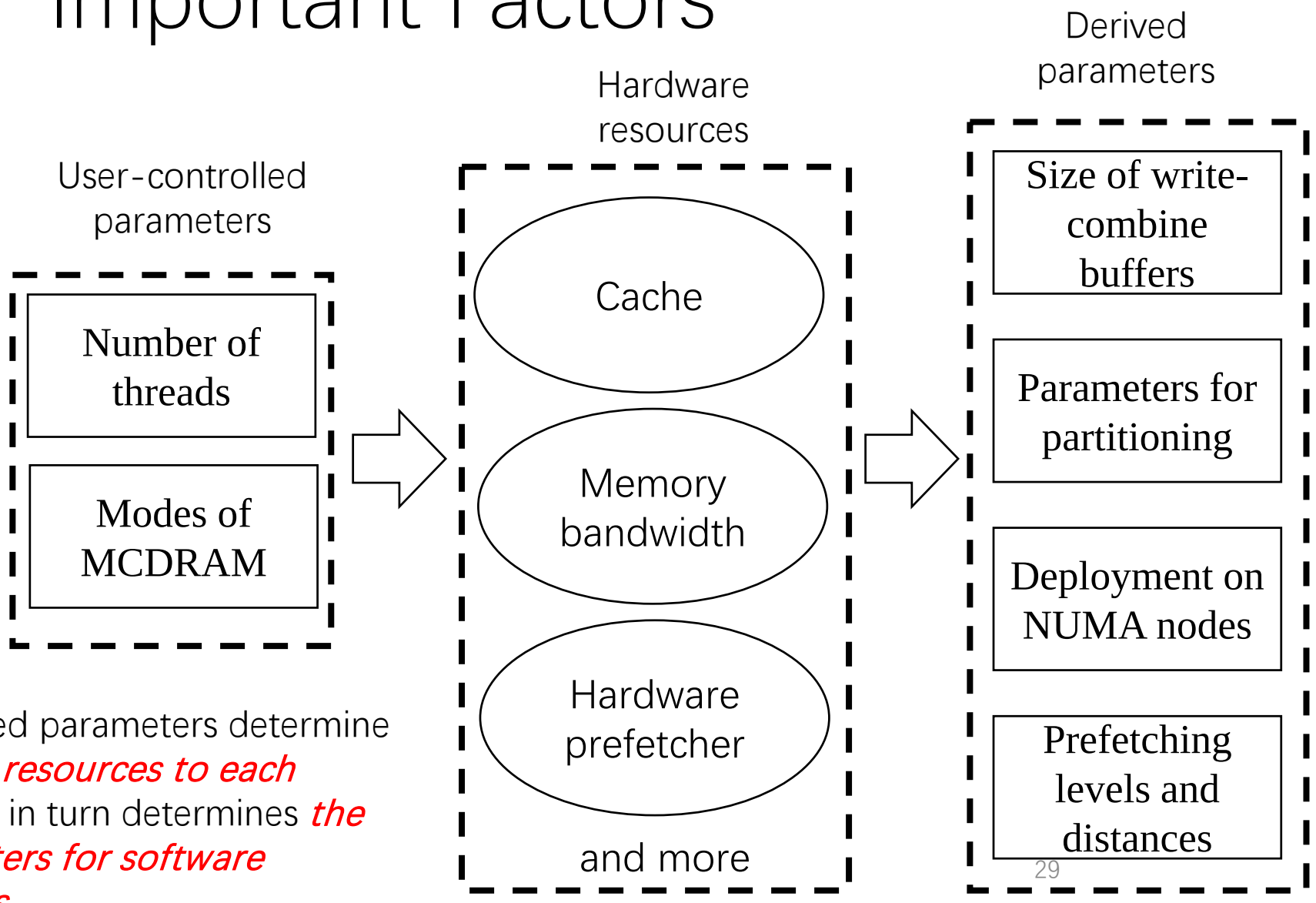


Sequential NUMA reads are much faster than random NUMA writes.

Goals and Approaches



Important Factors



User-controlled parameters determine *the available resources to each thread*, which in turn determines *the best parameters for software optimizations*.

Experimental Setup

- Processors
  - 1x Intel Xeon Phi 5110P of the Knights Corner architecture (KNC)
  - 1x Intel Xeon Phi 7210 of the Knights Landing architecture (KNL)
  - 4x Intel Xeon X7560 CPU

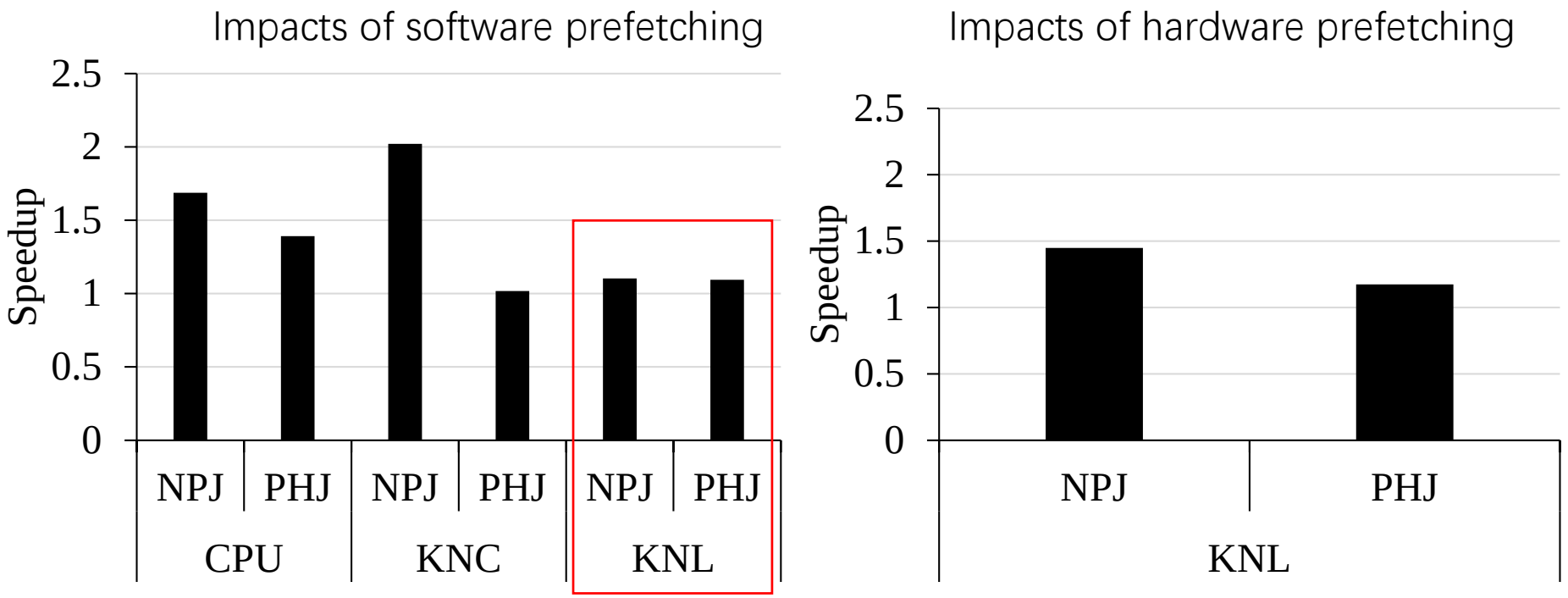
- Workload
  - Equi-join query
  - Tuple size: 4 B key, 4 B payload
  - Size: 128 million to 1024 million tuples per relation

Hotspots in PHJ

Hotspots	Time Consumption	Implementation
Probing hash tables	14.49%	SIMD gather
Resolving hashing conflicts	13.96%	SIMD gather/scatter
Loading keys and payloads	10.81%	SIMD sequential reads

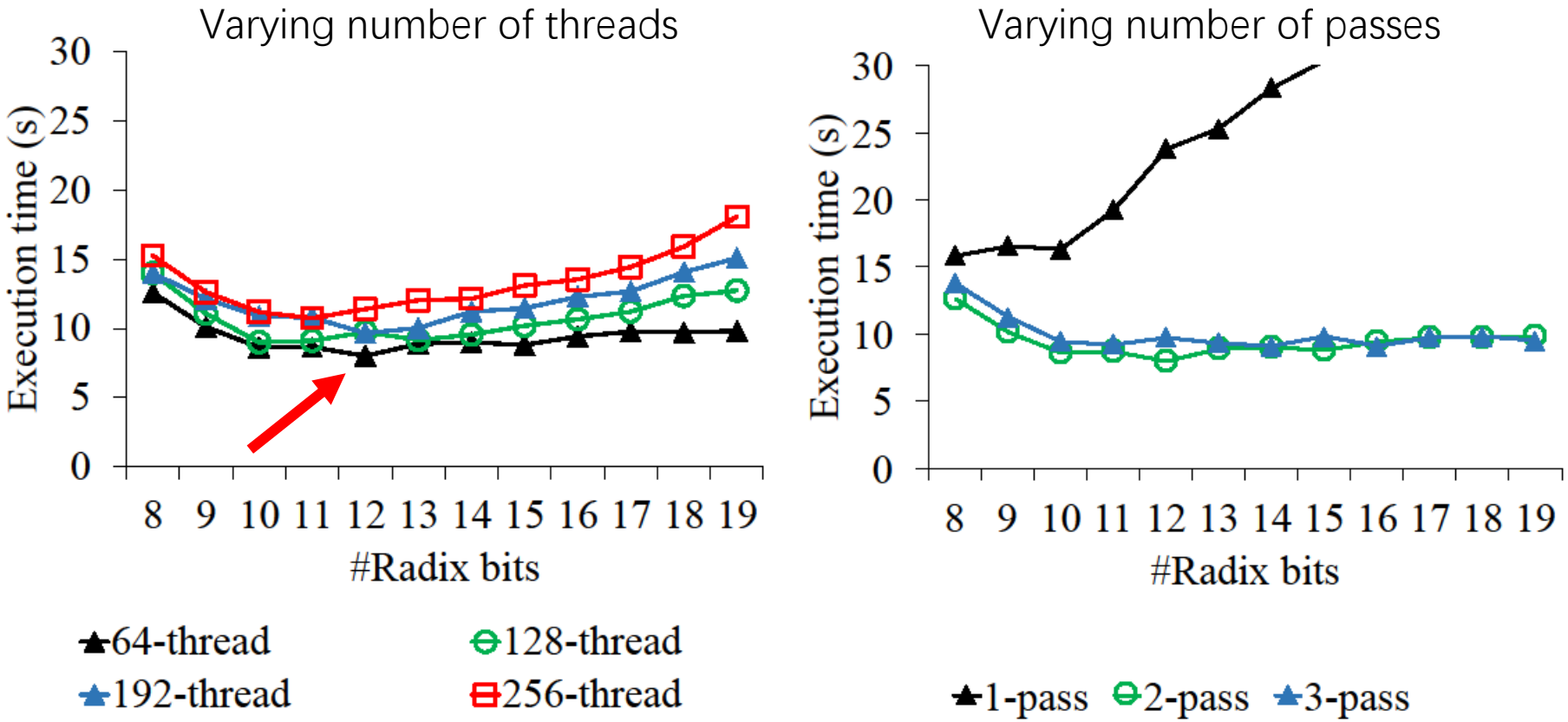
- Hotspots in PHJ are all memory-intensive
- Top 2 hotspots are random memory accesses

Impacts of Prefetching



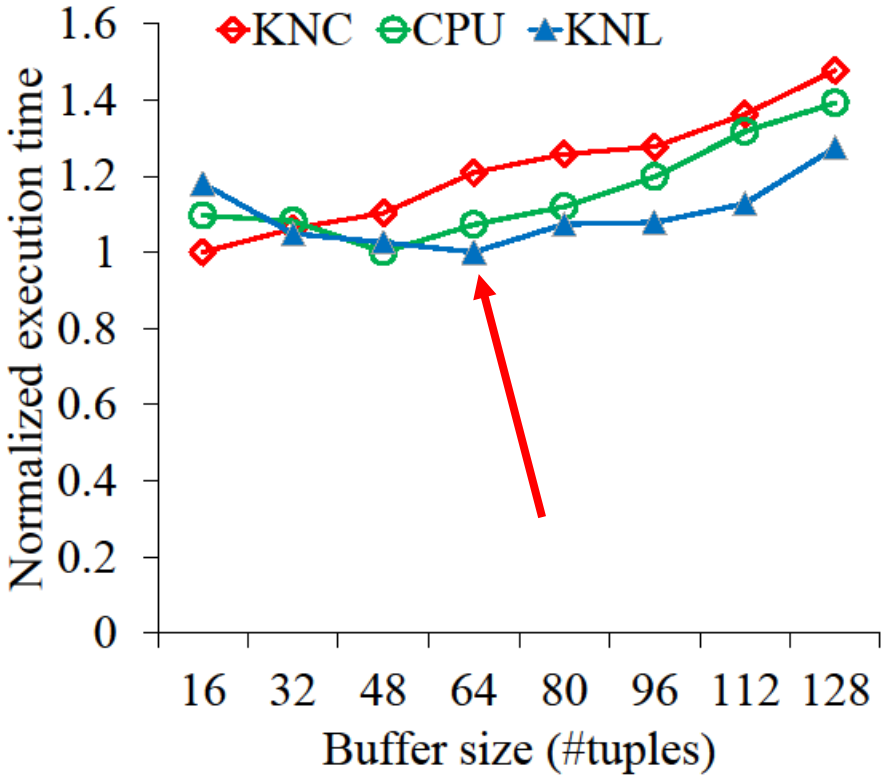
- Software prefetching techniques are not impactful on KNL, compared with other processors.
- Hardware prefetching achieves a higher speedup than software ones.

Threads, # Partitions & # Passes



- The 64-thread configuration performs the best.
- Less threads -> Larger cache per thread -> Larger fanout -> Less passes of partitioning

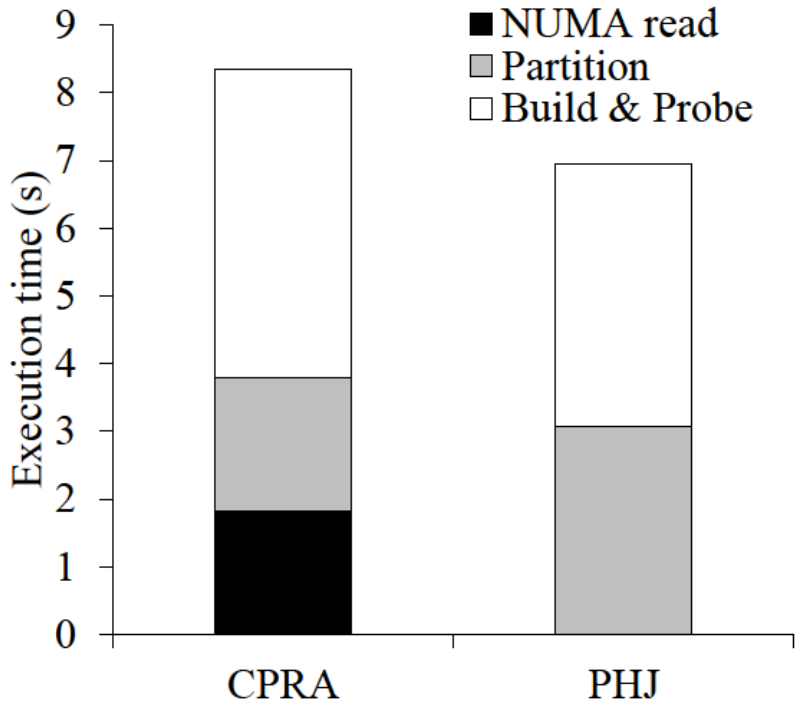
Size of Write-combine Buffers



KNL has a larger optimal buffer size than those on other processors.

Less threads -> Larger cache per thread

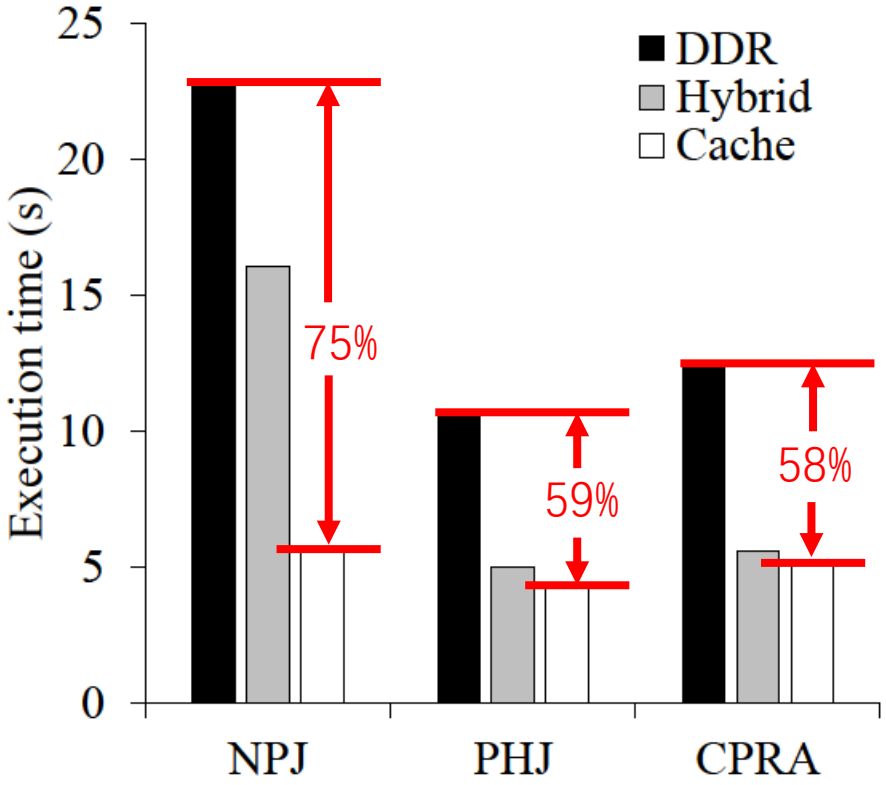
PHJ vs. CPRA (NUMA-aware opti.)



Existing NUMA-aware optimizations are not helpful on KNL.

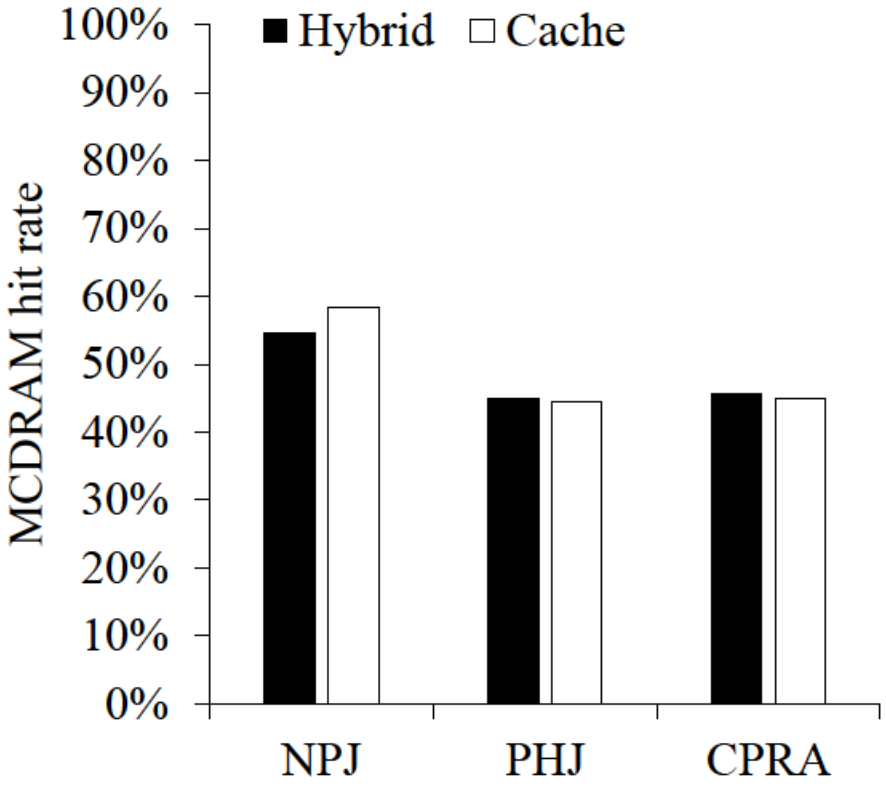
Please refer to the paper for detailed evaluations on the NUMA architecture.

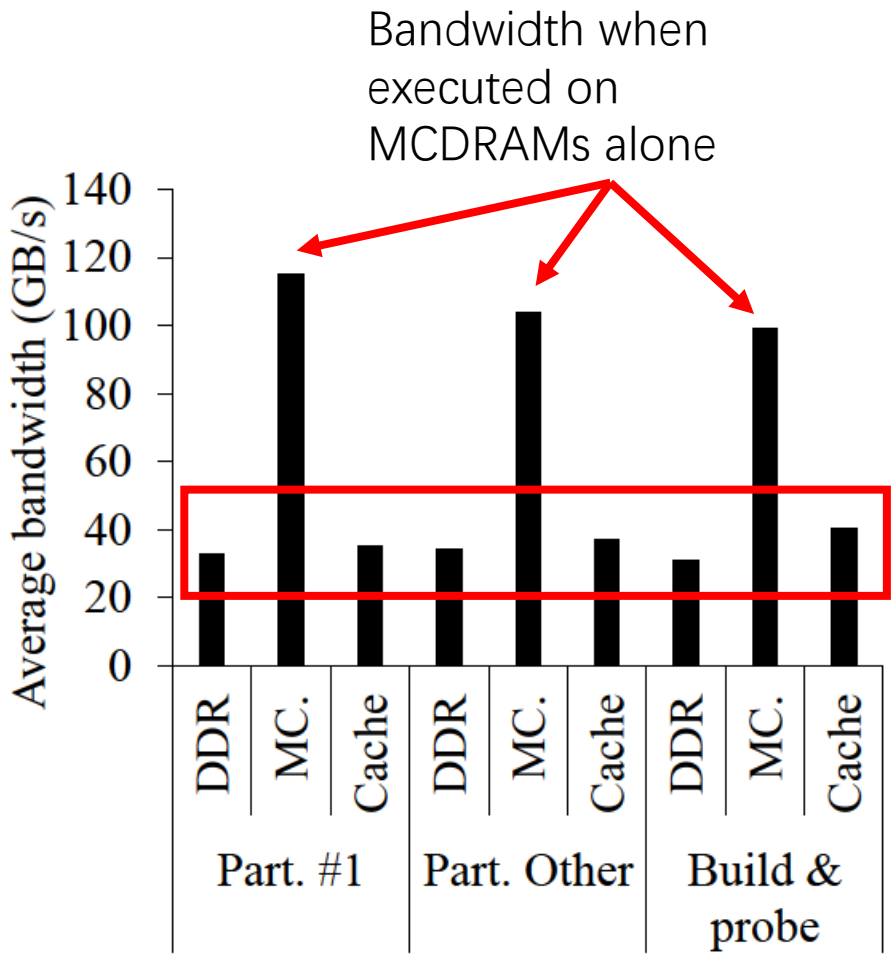
Impacts of the different Modes of the MCDRAM



Significant reduction in execution time!

MCDRAM's Utilization





- Low cache hit rates.
- Low bandwidth utilization.

Additional Results

- The best performing configuration of partitioned hash joins does not yield the highest memory bandwidth utilization.
- The cache mode of the MCDRAM has similar L1 cache hit rate but lower L2 cache hit rate than DDR only.
- Scattering threads among cores outperforms other threads' affinities.
- NUMA costs are significantly lower on KNL compared with multi-socket processors.
- There are new SIMD intrinsics available in AVX-512 that can assist random memory accesses.
- More results are in the paper.

Conclusions

- Hardware-conscious optimizations are very impactful on the performance of hash joins.
- We have experimentally revisited the state-of-the-art hash join algorithms and existing software optimizations on KNL.
- Our findings lead to new opportunities for database algorithms:
  - High-bandwidth MCDRAM + Large-capacity DDR4
  - New SIMD instructions
  - Many hardware threads

Deployment of hash tables on many-core architectures with die-stacked High Bandwidth Memory

Xuntao Cheng, Bingsheng He, Eric Lo, Chiew Tong Lau

An new NUMA architecture

- High intra-node bandwidth
- HBM
  - Higher bandwidth
  - Lower capacity
- Main memory (DDR4)
  - Lower bandwidth
  - Higher capacity

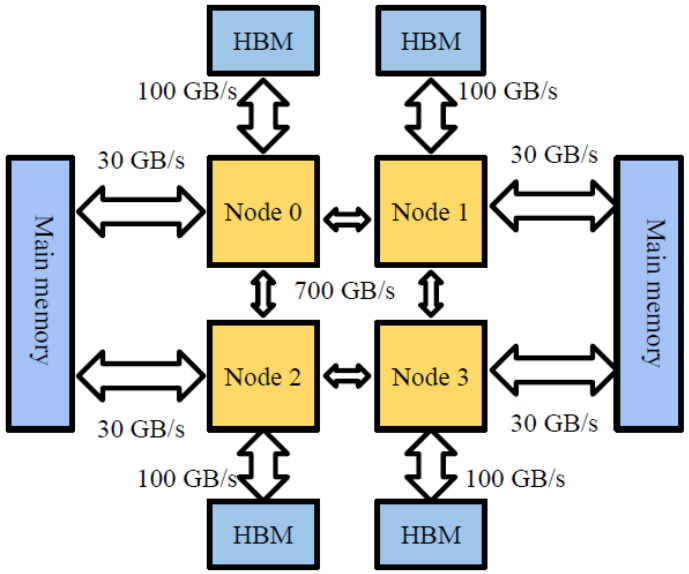
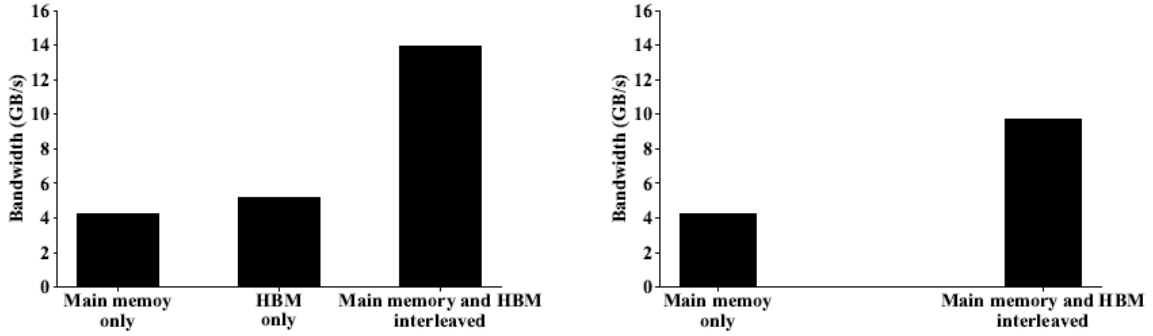


Figure 1: NUMA topology and peak bandwidth on the Intel KNL many-core processor with die-stacked HBMs.

Potentials in memory bandwidth

- HBM has a slightly higher bandwidth than the main memory for random memory accesses
- The highest bandwidth is achieved when using both memories in parallel.



(a) 4 GB (b) 64 GB

Figure 2: Peak memory bandwidth accessing a data structure placed in different ways

Problem

- Hash table is an important data structure in relational databases.
  - Used for hash joins, lookups, etc.

- We want to deploy a hash table on the many-core architecture with die-stacked HBM while
  - Maximizing the exploitation of memory resources of all NUMA nodes,
  - Minimizing workload imbalance.

Challenges

- Memory accesses to hash tables are hard to predict, and may happen between any core and any NUMA node (NUMA).
  - Unknown memory access costs.
- Existing NUMA-aware optimizations treat all NUMA nodes equally.
  - They are designed for a single type of memory.
- Existing NUMA-aware optimizations do not consider thread-level workload imbalance on many-core processors.
  - Thread-level imbalance can lead to stragglers easily.

Workflow

- Estimate the memory access cost first.
- Distribute the estimated costs evenly across all NUMA nodes by
  - Placing the hash table.
  - Placing the threads.
- This procedure can be applied to both the building and probing of a hash table.

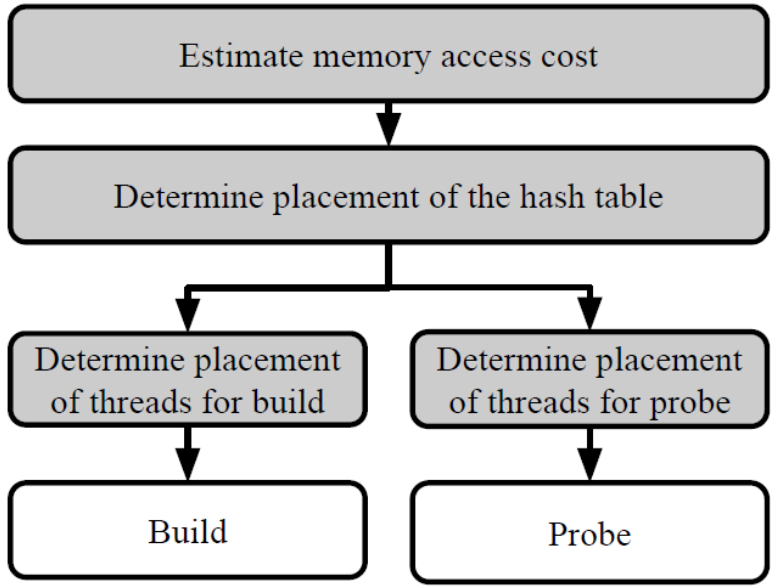


Figure 3: Workflow of the proposed deployment algorithm

Formulations

- Placement of the hash table -> Partition problem
- Placement of threads -> College entrance problem

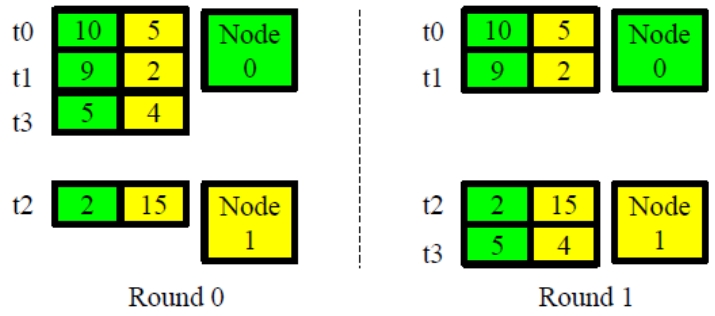


Figure 5: An example of placing four threads to two NUMA nodes using the thread placement algorithm.

Hash join variants

- Two camps of hash joins
  - Simple hash join (SHJ)
  - Partitioned hash join (PHJ)
- Three memory modes
  - DDR-only
  - HBM as a cache
  - All nodes managed by the NUMA utility of Linux
  - All nodes managed by our proposed algorithms

Table 1: List of hash join variants

Variant	HBM mode	Sources
Simple hash joins		
SHJ (DDR)	DDR only	[3, 19, 6]
SHJ (Cache)	HBM configured as a cache	[3, 19, 6]
SHJ (NUMA)	All nodes managed by the NUMA utility	[3, 19, 6]
HSHJ	All nodes managed by the proposed optimizations	This paper
Partitioned hash joins		
PHJ (DDR)	DDR only	[3, 19, 6]
PHJ (Cache)	HBM configured as a cache	[3, 19, 6]
PHJ (NUMA)	All nodes managed by the NUMA utility	[3, 19, 6]

Overall performance comparison

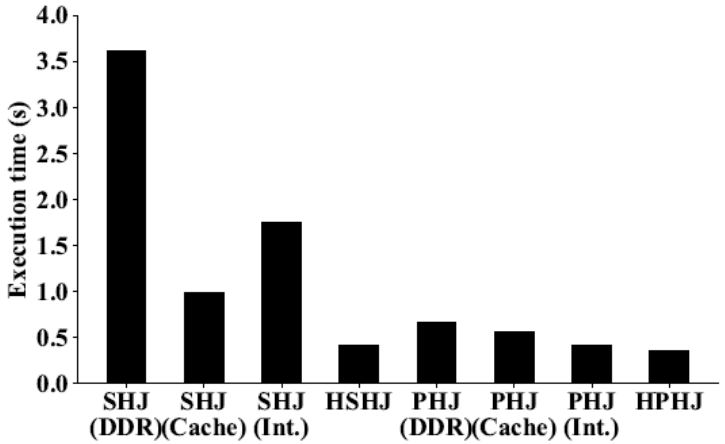


Figure 7: Execution time of hash join variants processing the small-size workload

Our proposed algorithms improve the performance of SHJ and PHJ by about 20% and 3 times.

Impact on workload balancing

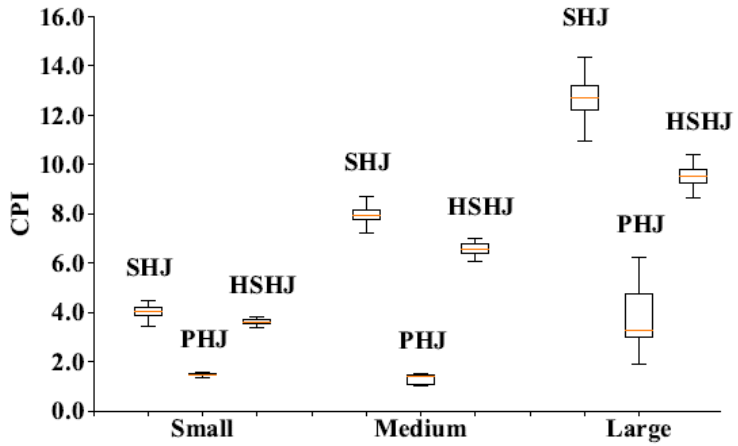


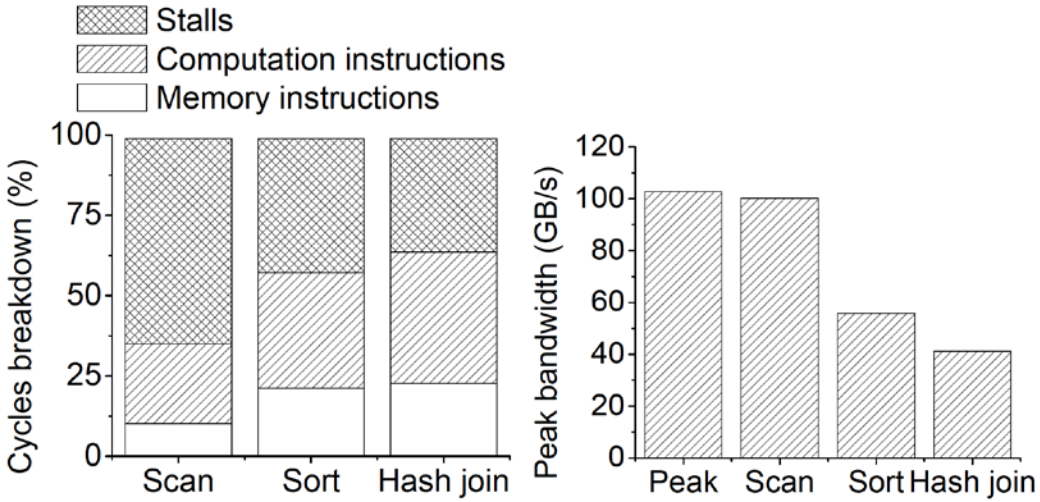
Figure 11: Distribution of CPI per core while executing different hash join variants.

- Partitioned hash joins have lower CPIs in general.
- Our optimized SHJ reduces the CPIs of the baseline SHJ.

Many-core needs fine-grained scheduling: A case study of query processing on Intel Xeon Phi processors

Xuntao Cheng, Bingsheng He, Mian Lu, Chiew Tong Lau

Observations



(a) Breakdown of cycles (b) Bandwidth

- Databases contain complex operators which suffer from memory stalls on many-core processors.
- Sort and hash join have poor memory bandwidth utilization.

Idea

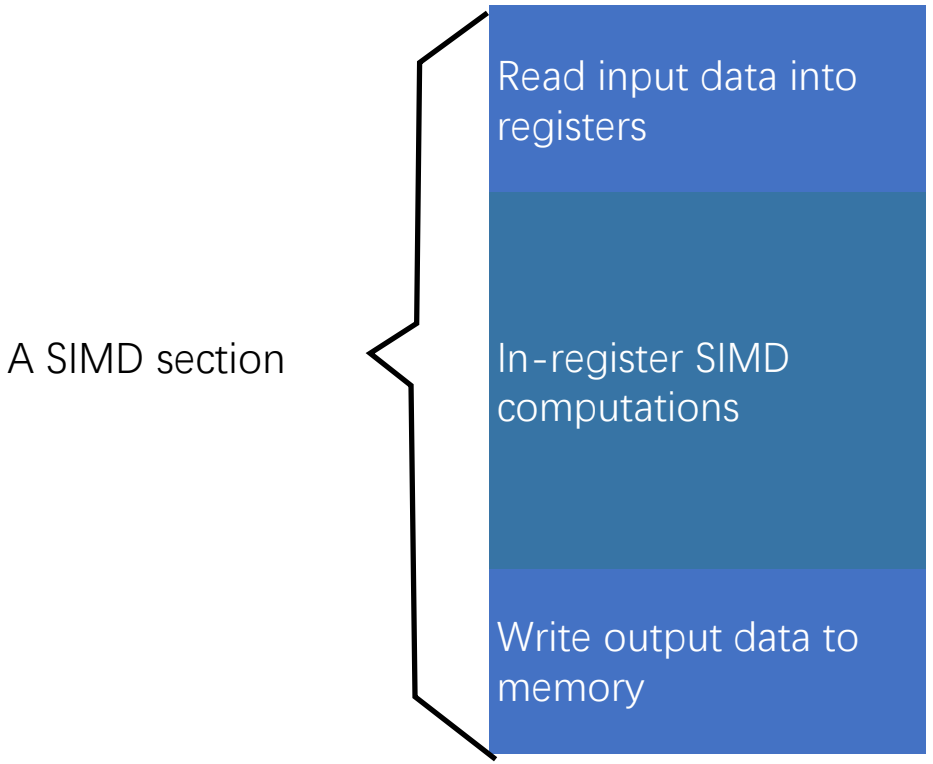
- Concurrent executions of operators with complimentary resource requirements can utilize the hardware better. However, complex operators tend to have similar performance requirements.
- Thus, we try decompose complex operators into fine-grained phases
  - Each phase is chosen to be either compute-bound or memory bound.
  - Among each core's hyper threads, we mix such phases to increase the overall IPC.

Challenges

- Decomposition causes overhead in runtime.
  - Phases decomposed from the same operators have dependencies, and have to be executed in serial.
  - There are potential context switching overhead between them in runtime.
- The search for good matches for concurrent executions takes time.
  - The search space can be very large while processing multiple queries.

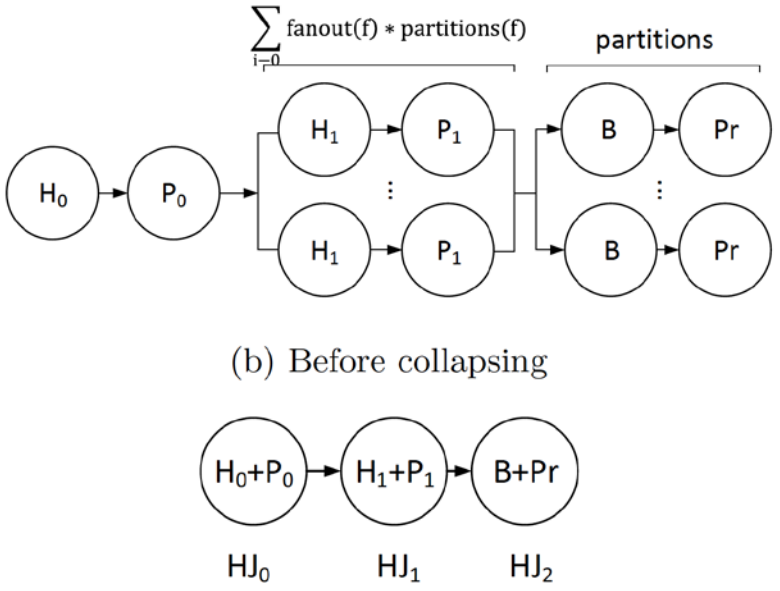
SIMD section

- All vectorized SIMD computations work on a small number of dedicated SIMD registers.
- The structure of codes determines resource requirements.
- There is no benefit in breaking such sections.



Candidate phase graph

- Each operator consists of several SIMD sections.
- An entire operator is formulated to be a DAG of such sections where each section is a vertex.
- The decomposition of operators is translated into a graph problem where we merge vertices with similar resource requirements together.

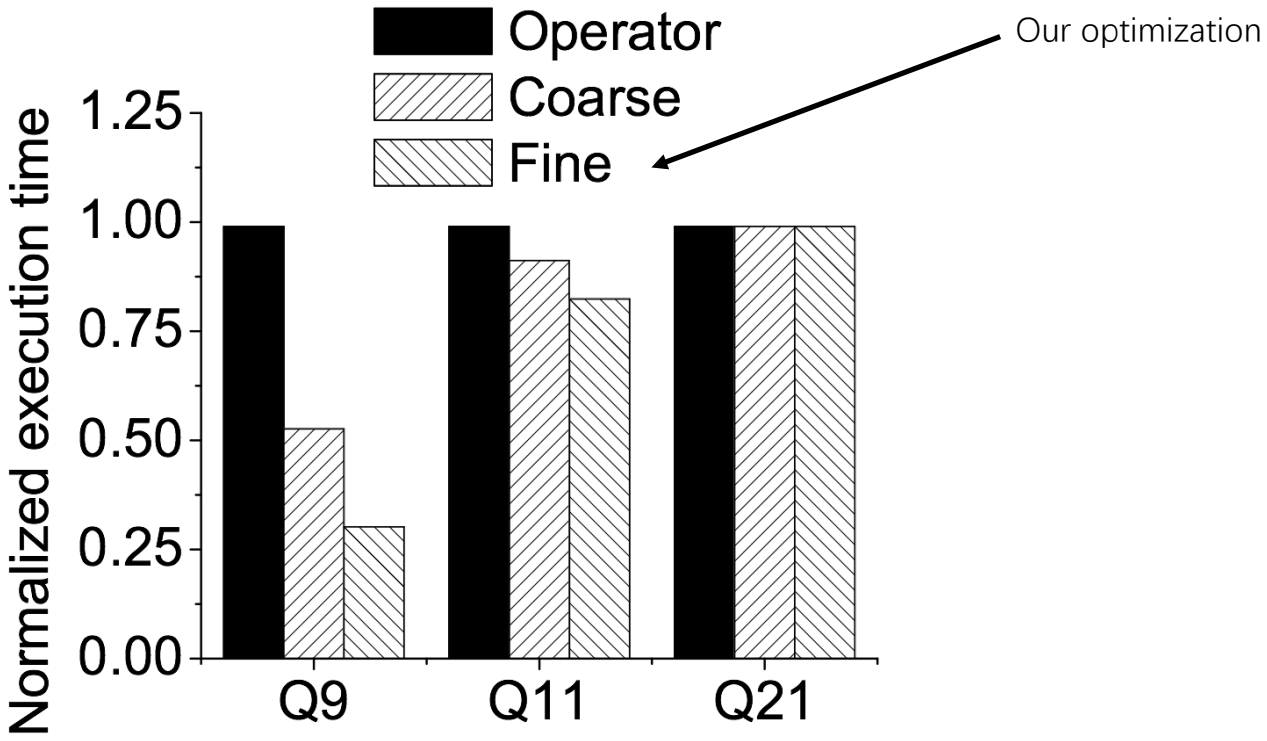


(b) Before collapsing

(c) After collapsing

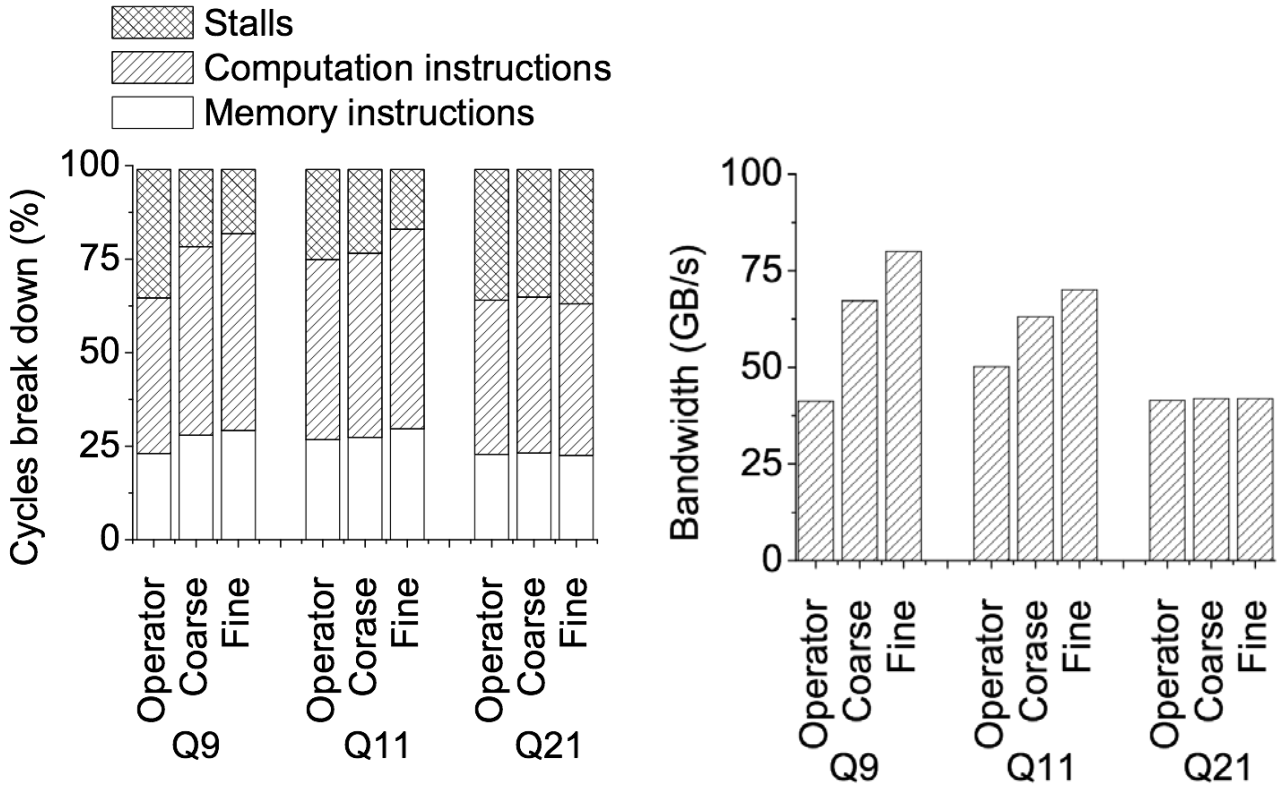
Figure 4: Candidate phase graph of hash join

Single query evaluation



More operators in a query -> more phases for concurrent executions -> higher performance impacts of our optimizations.

Improved resource utilizations



Our optimizations have improved the resource utilization of both the pipelines and the memory bandwidth.

Thanks.