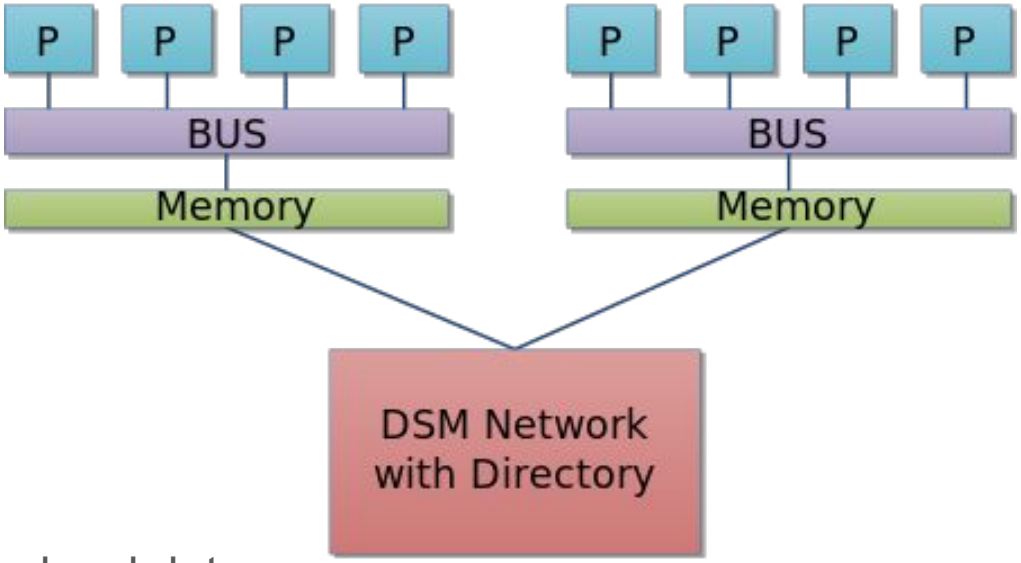
Morsel-Driven Parallelism

Jason Banks

Problem

- Increasing multi-core parallelism > increasing single-thread performance
- Plan-driven parallelism/volcano-style parallelism
  - Bottlenecks and load balancing issues; not flexible
  - Parallelism hidden from operators; not optimal partitioning results due to lack of locality awareness.
  - Partitioning is static; load imbalances
  - Not NUMA aware; memory controllers decentralized to chips.
    - Main memory databases; many-core allows parallelization, but decentralization increases costs.

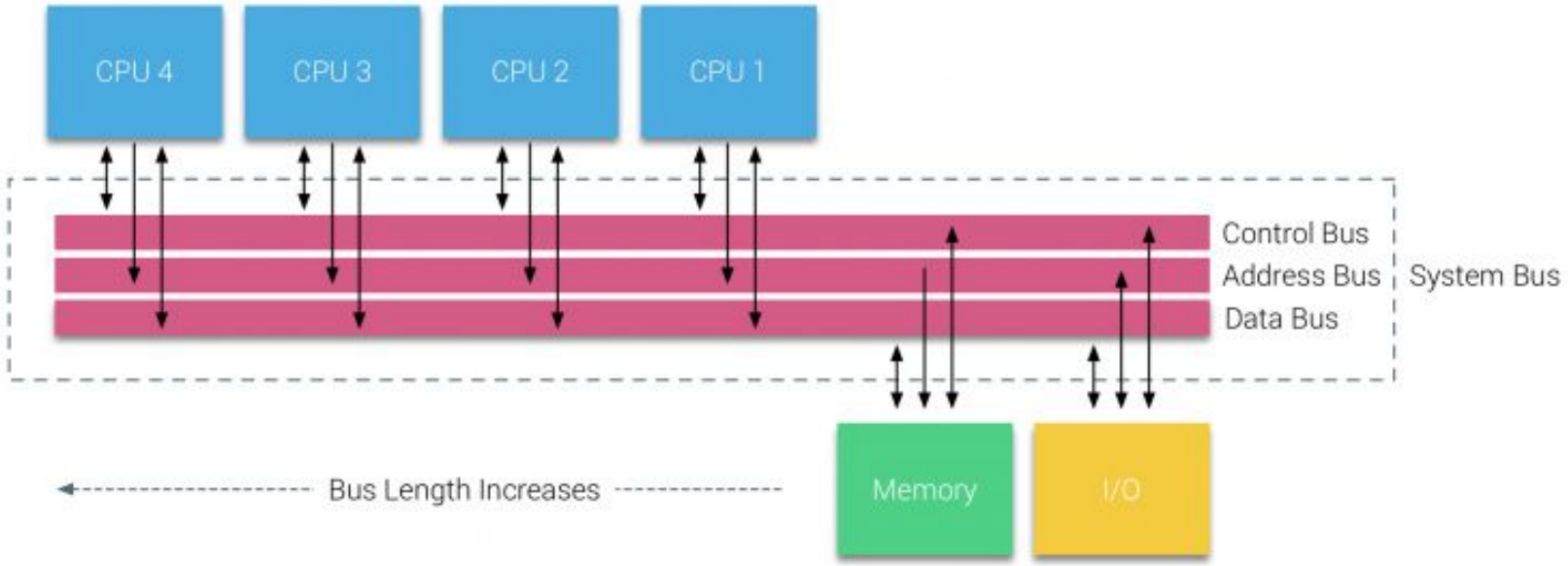
NUMA-Awareness



- Non-Uniform Memory Access
- Mem controllers → chips.
- Access between cores and main memory not uniform.
- Access costs varied dependent on chip location (decentralized)
- Memory hierarchies need to be accounted for s.t. threads work on local data
  - Local data accessed faster than non-local data
  - Strategies: faster memory access > faster processors; limit memory accesses

Example

- Processors accessing non-local data increases latency
- Bus length increases with more processors
  - Scalability problems; increased latency; memory hierarchy is important.



Adaptive morsel-driven query execution

- Parallel query evaluation framework
- Task distribution at runtime; fully elastic
- Input processed by constant-sized work units
- NUMA-aware
- Morsel-wise scheduling for operators; hash-table sharing.
- Worker threads are equal to hardware threads.

Features

- Morsels (small fragments of input data) scheduled to threads.
  - Reduces load imbalances, workers finish on time.
  - Allocation of memory to decentralized chips; reduces access time; query cancelling
- Constant sized; facilitate work-stealing and preemption.
- Pipeless dependent
  - Volcano: oblivious to parallelism

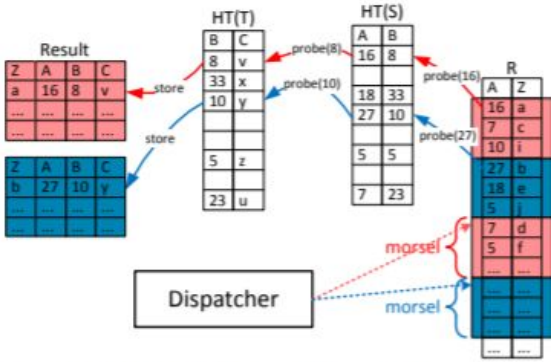


Figure 1: Idea of morsel-driven parallelism: $R \bowtie_A S \bowtie_B T$

Parallel Processing

- Hash table split to two pipelines. Thread local storage; avoid synch
- Scan and filter input T; build hash table HT(T) (Build and Probe Phases)

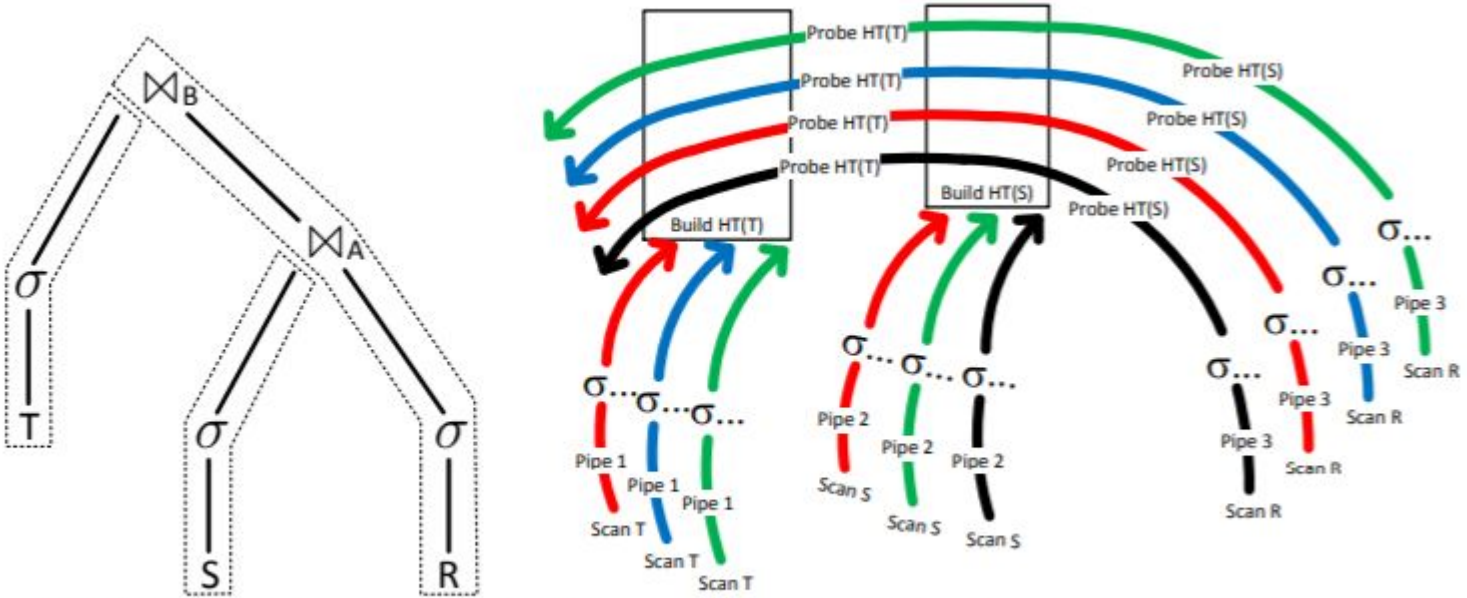


Figure 2: Parallellizing the three pipelines of the sample query plan: (left) algebraic evaluation plan; (right) three- respectively four-way parallel processing of each pipeline

Features

- Adaptive dispatcher
  - This allows adjusting of resources and reaction to execution speed during runtime.
  - Reallocates morsels and tasks if one finishes early (work-stealing)
  - Reduces/increases parallelism on threads
- NUMA-Aware; awareness of data locality
  - Process T morsel-wise; store NUMA-locally.
  - Pointers to hash table

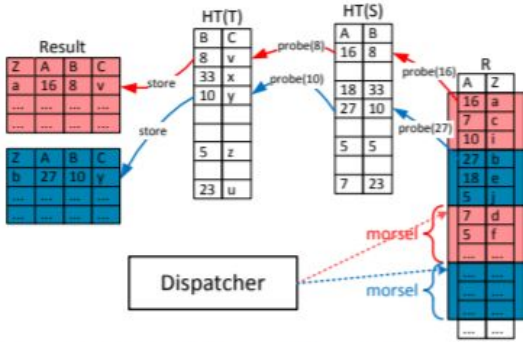


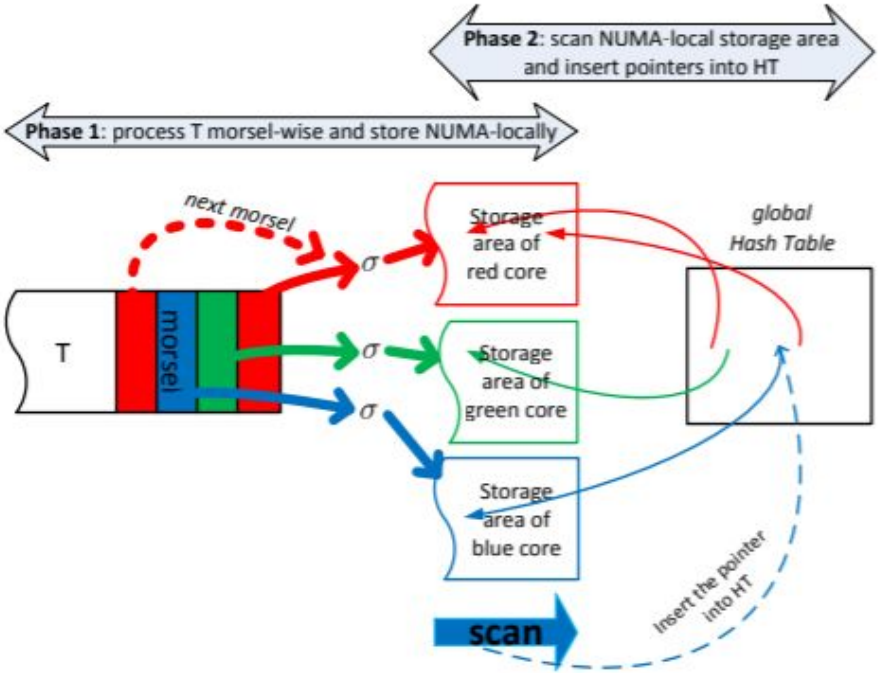
Figure 1: Idea of morsel-driven parallelism: $R \bowtie_A S \bowtie_B T$

Parallel Operators

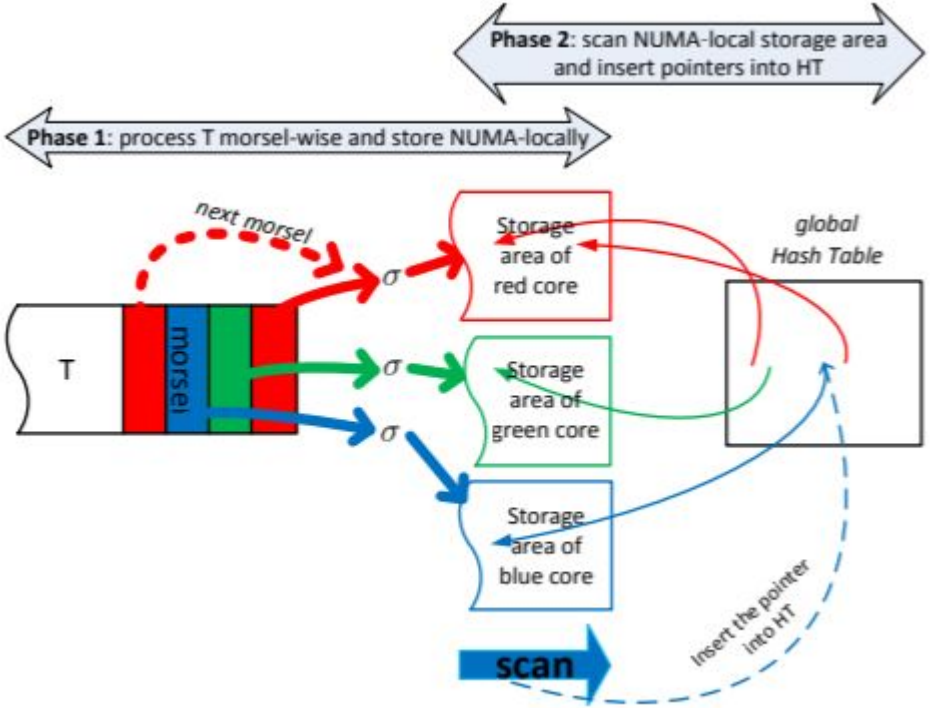
- Need to accept tuples in parallel for pipeline parallelization
- Awareness of parallelism
- Dispatcher awareness of data
  - locality of morsels
  - Data and computation locations
  - Most execute on NUMA-local memory; faster accesses vs. non-local memory.
  - Operator pipelines for workers, still elastic

Hash Join

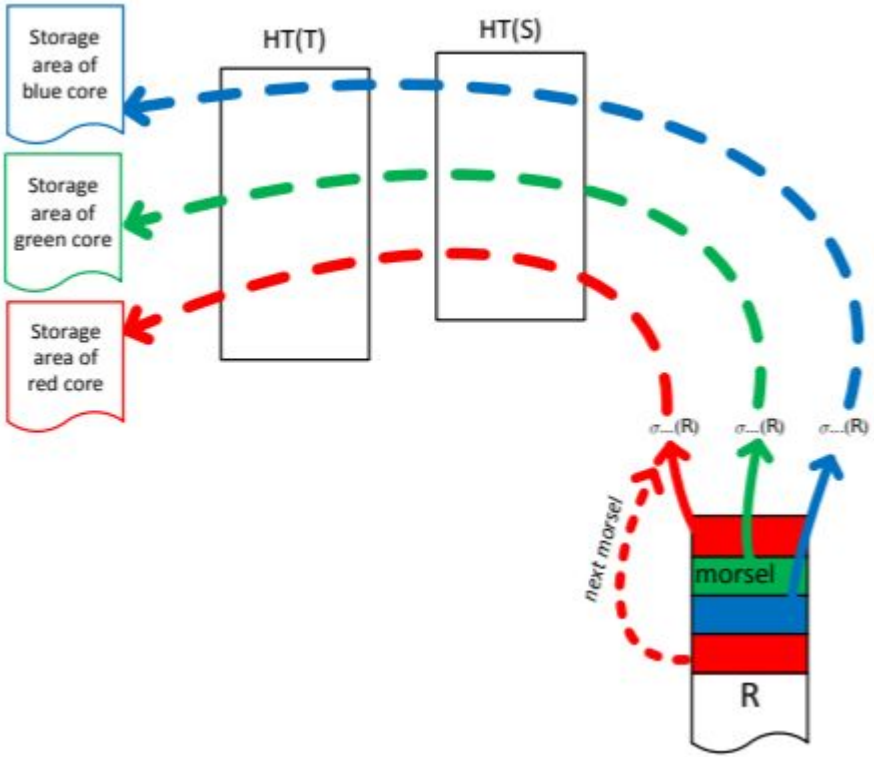
- Two tables (HT(T);HT(S))
- Build phase
  - Materialize input tuples to thread local storage
  - Input size known, empty table of input size constructed
- Probe phase
  - Inserts pointers to tuples
  - Check matches
  - Insert only; lookups after all inserts
  - Low synchronization costs



Build Phase


Phase 2: scan NUMA-local storage area and insert pointers into HT
Phase 1: process T morsel-wise and store NUMA-locally
next morsel
T
morsel
σ
σ
σ
Storage area of red core
Storage area of green core
Storage area of blue core
global Hash Table
scan
Insert the pointer into HT

Probe Phase


Storage area of blue core
Storage area of green core
Storage area of red core
HT(T)
HT(S)
σ...(R)
σ...(R)
σ...(R)
next morsel
morsel
R

Grouping/Aggregation

- Performance dependent on number of distinct keys; cache misses

Figure 8: Parallel aggregation

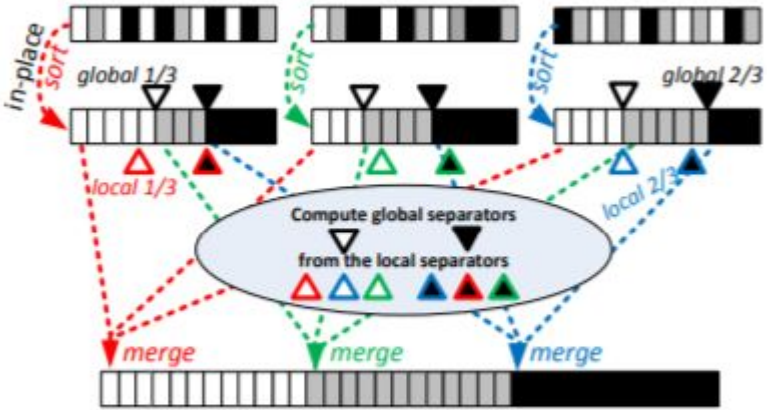


Figure 9: Parallel merge sort

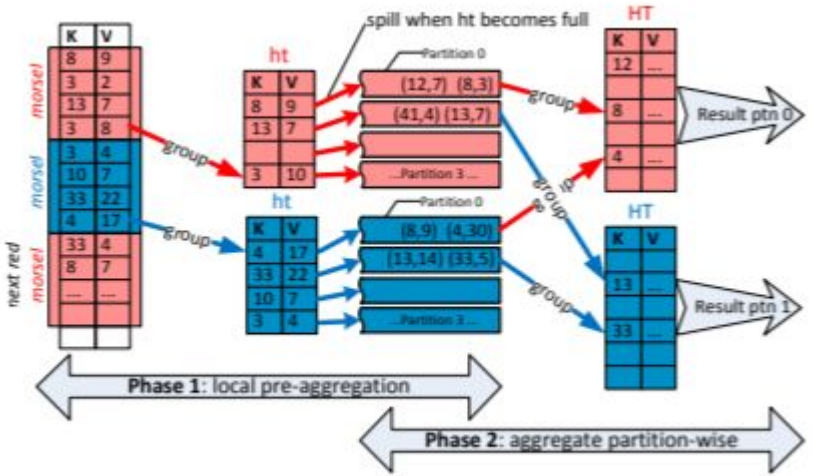


Figure 8: Parallel aggregation

Dispatcher

- Preserve NUMA locality
- Full elasticity
  - One morsel at a time
  - Priority scheduling of queries
- Load balancing
  - Prevent fast cores from waiting for slow cores.

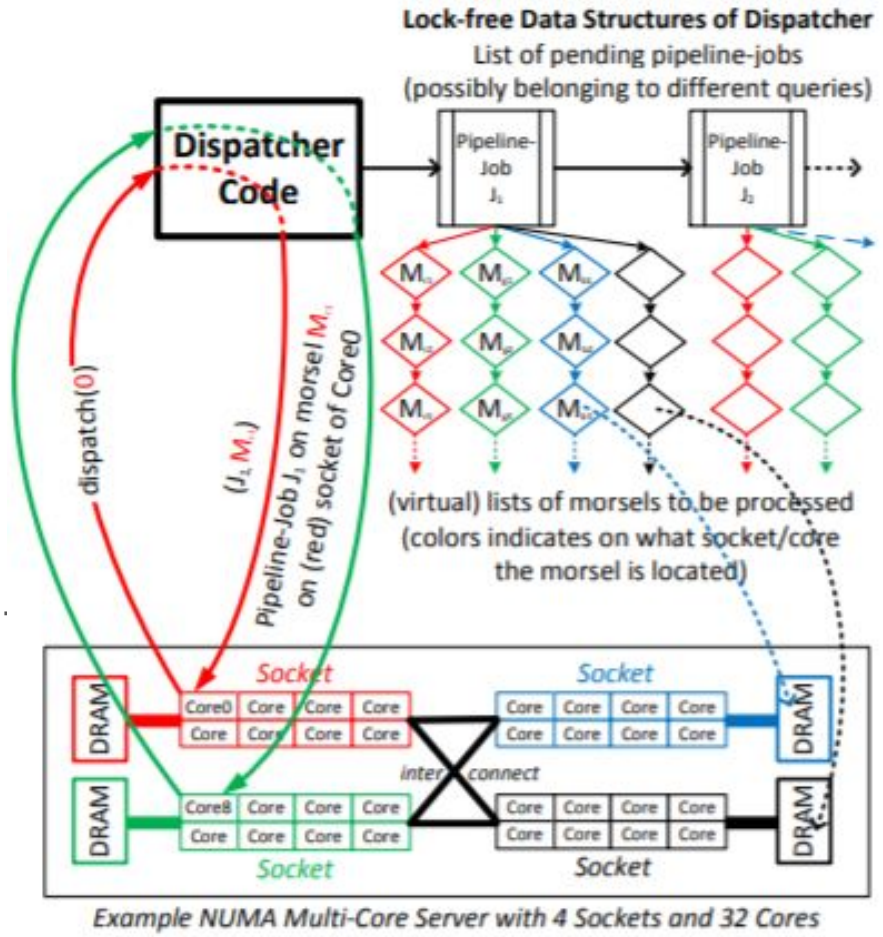


Figure 5: Dispatcher assigns pipeline-jobs on morsels to threads depending on the core

Dispatcher

- Assigns resources to parallel pipelines (tasks → worker threads)
- Aware of data locality of NUMA-local morsels; allow local execution.
- Cores are in different clusters with their own local memory regions.
- Can still access memory in other clusters.
- Scheduling mechanism; allows flexible parallel execution (elasticity).

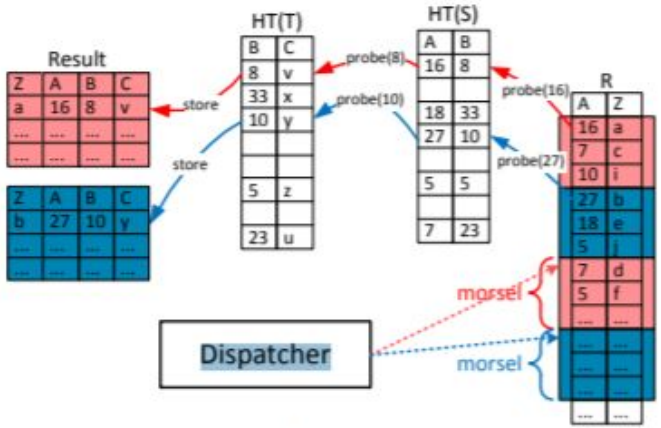


Figure 1: Idea of morsel-driven parallelism: $R \bowtie_A S \bowtie_B T$

Implementation

- HyPer
  - Main memory column database
  - No intra-query parallelism
  - JIT (Just-In-Time) Approach
    - Running during execution, schedule is morsel-wise.
- QEProject
  - Executable pipelines to dispatcher
  - Observes data dependencies
  - Temporary storage area for each thread/core

Implementation (Vectorwise)

- TPC-H
  - Vectorwise; competitor system
    - Not NUMA-aware
  - HyPer achieves 30x speed up
  - NUMA awareness important in some cases
  - Non-adaptivity reduces in most
  - All perform better than Vectorwise

system	geo. mean	sum	scal.
HyPer	0.45s	15.3s	28.1×
Vectorwise	2.84s	93.4s	9.3×
Vectorwise, full-disclosure settings	1.19s	41.2s	8.4×

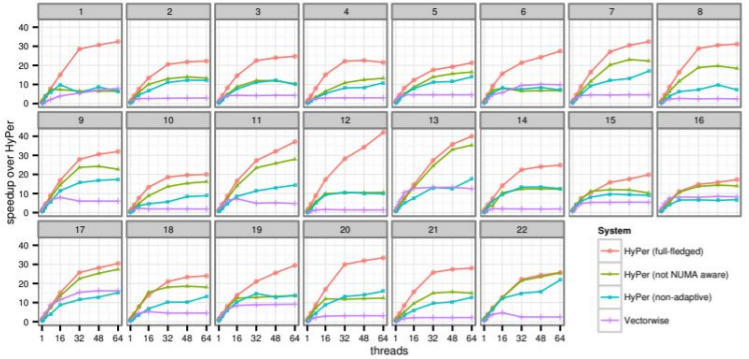


Figure 11: TPC-H scalability on Nehalem EX (cores 1-32 are "real", cores 33-64 are "virtual")

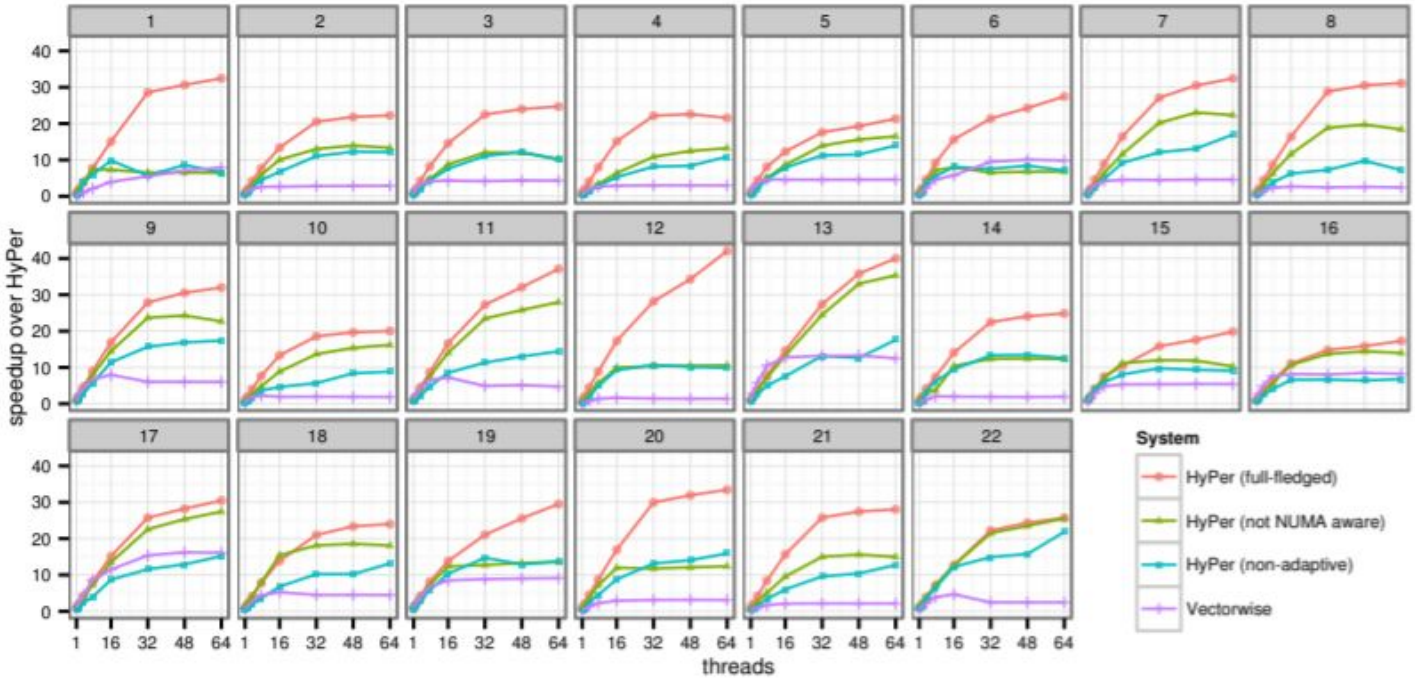


Figure 11: TPC-H scalability on Nehalem EX (cores 1-32 are "real", cores 33-64 are "virtual")

Implementation (NUMA awareness)

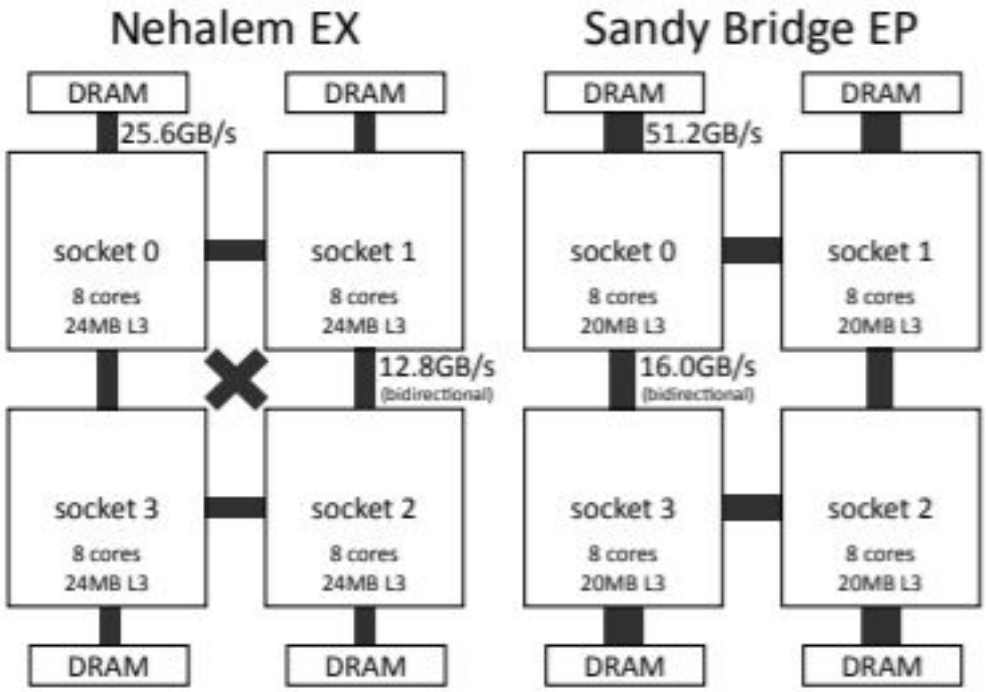


Figure 10: NUMA topologies, theoretical bandwidth

Implementation (NUMA awareness)

- NUMA-aware processing; lower latency and higher bandwidth
- Reducing of remote accesses in joins
- Alternatives
  - OS Default (OS placement)
    - Memory controller becomes a bottleneck
  - Interleaved (robin round allocation)
    - Reasonable not optimal

TPC-H:

	Nehalem EX		Sandy Bridge EP	
	geo. mean	max	geo. mean	max
OS default	1.57×	4.95×	2.40×	5.81×
interleaved	1.07×	1.24×	1.58×	5.01×

	bandwidth [GB/s]		latency [ns]	
	local	mix	local	mix
Nehalem EX	93	60	161	186
Sandy Bridge EP	121	41	101	257

Implementation (Elasticity)

- Parallelization shown to be fully elastic.
- TPC-H queries
- Threads can switch between queries dynamically.
  - Succeeded in reassigning threads in runtime and prioritizing.
- VS Static assignment in Volcano:
  - 2nd test: input/threads = morsel size.
  - Not too different w since query at once
  - Another single-threaded process reduces performance by 36.8%(vs 4.7%)

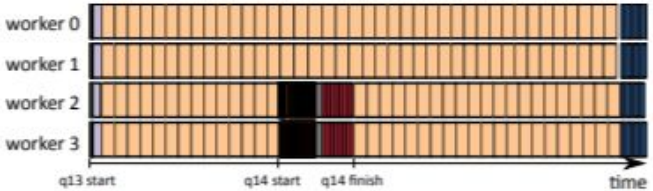


Figure 13: Illustration of morsel-wise processing and elasticity

SSB #	time [s]	scal. [×]	read [GB/s]	write [GB/s]	remote [%]	QPI [%]
1.1	0.10	33.0	35.8	0.4	18	29
1.2	0.04	41.7	85.6	0.1	1	44
1.3	0.04	42.6	85.6	0.1	1	44
2.1	0.11	44.2	25.6	0.7	13	17
2.2	0.15	45.1	37.2	0.1	2	19
2.3	0.06	36.3	43.8	0.1	3	25
3.1	0.29	30.7	24.8	1.0	37	21
3.2	0.09	38.3	37.3	0.4	7	22
3.3	0.06	40.7	51.0	0.1	2	27
3.4	0.06	40.5	51.9	0.1	2	28
4.1	0.26	36.5	43.4	0.3	34	34
4.2	0.23	35.1	43.3	0.3	28	33
4.3	0.12	44.2	39.1	0.3	5	22

Table 3: Star Schema Benchmark (scale 50) on Nehalem EX

Implementation (Star Schema)

- Data warehousing scenario
- Speed up of 40x
- Large fact table, NUMA-local
- Faster aggregation
- Better scalability vs TPC-H
  - TPC-H more complex
  - Complex joins and aggregations
  - Single table

SSB #	time [s]	scal. [×]	read [GB/s]	write [GB/s]	remote [%]	QPI [%]
1.1	0.10	33.0	35.8	0.4	18	29
1.2	0.04	41.7	85.6	0.1	1	44
1.3	0.04	42.6	85.6	0.1	1	44
2.1	0.11	44.2	25.6	0.7	13	17
2.2	0.15	45.1	37.2	0.1	2	19
2.3	0.06	36.3	43.8	0.1	3	25
3.1	0.29	30.7	24.8	1.0	37	21
3.2	0.09	38.3	37.3	0.4	7	22
3.3	0.06	40.7	51.0	0.1	2	27
3.4	0.06	40.5	51.9	0.1	2	28
4.1	0.26	36.5	43.4	0.3	34	34
4.2	0.23	35.1	43.3	0.3	28	33
4.3	0.12	44.2	39.1	0.3	5	22

Table 3: Star Schema Benchmark (scale 50) on Nehalem EX

Conclusion

- NUMA-awareness and runtime adaptivity is important; allows load balancing, elasticity and reduces latency in parallelism.
- Oblivious operators are not always the best solution. (parallel-aware)
- Splitting task and morsels carefully is important for analytical query performance.
- Possible to further reduce NUMA remote accesses.
- Morsel-driven parallelism performs better when handling multiple threads and queries where speed and workload vary throughout runtime.

Bibliography

- https://frankdenneman.nl/2016/07/07/numa-deep-dive-part-1-uma-numa/
- https://whatis.techtarget.com/definition/NUMA-non-uniform-memory-access#:~:text=NUMA%20(non%2Duniform%20memory%20access)%20is%20a%20method%20of,symmetric%20multiprocessing%20(%20SMP%20)%20system.