CWI Centrum Wiskunde & Informatica

NUMA obliviousness through memory mapping

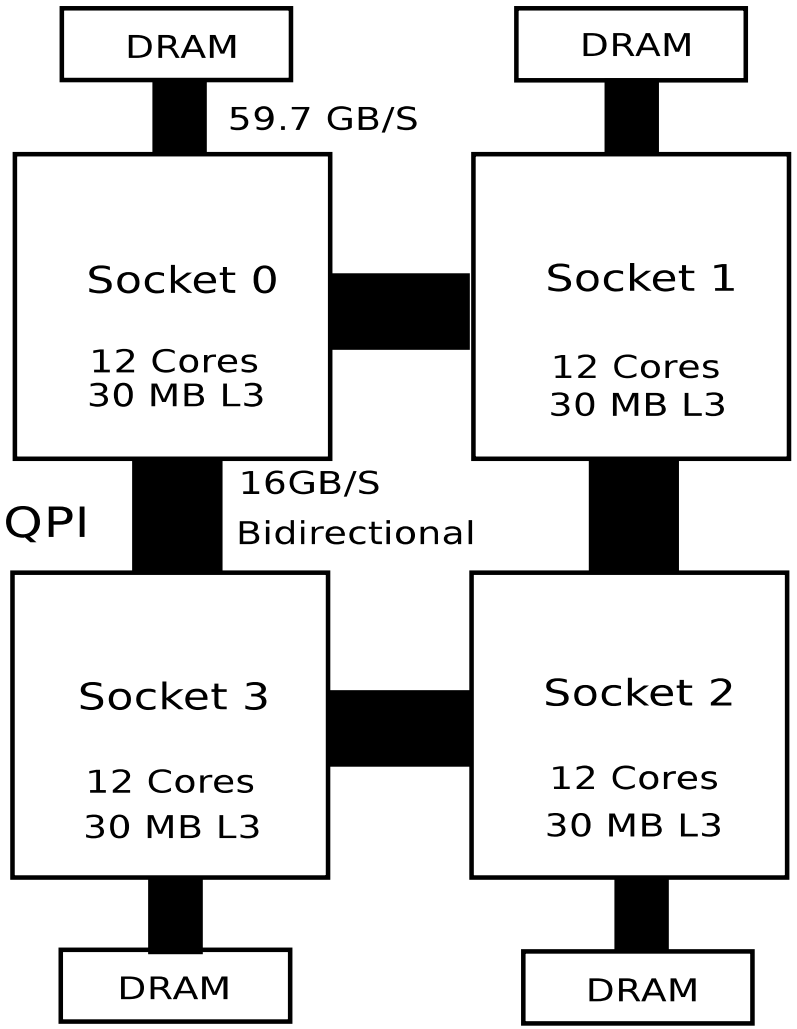
Mrunal Gawade **Martin Kersten**
CWI, Amsterdam

DaMoN 2015 (1st June 2015)
Melbourne, Australia

COMMIT/

monetdb

NUMA architecture



Intel Xeon E5-4657L v2 @2.40GHz

Memory mapping

What is it?

Operating system maps disk files to memory

E.g. Executable file mapping

How is it done?

System call – mmap(), munmap()

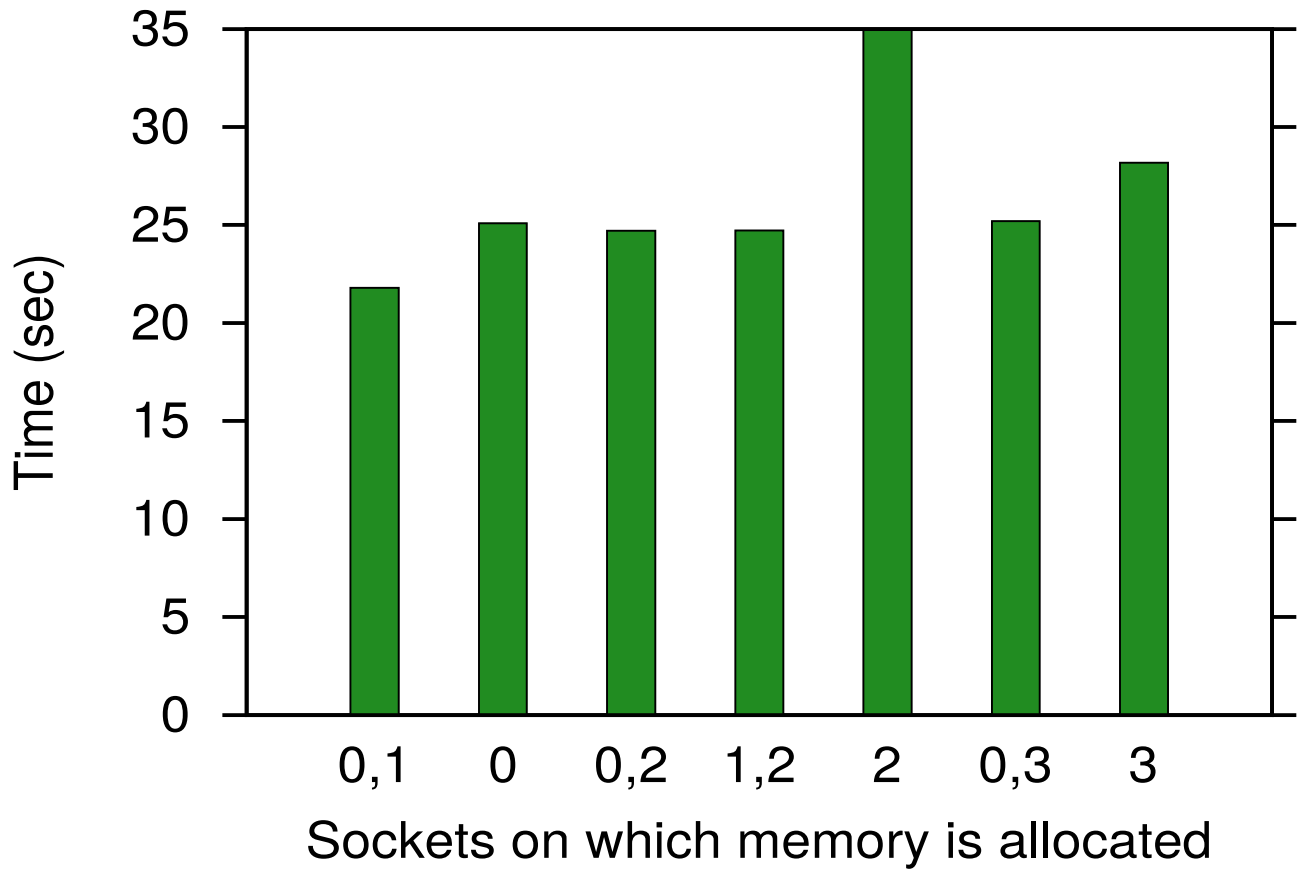
Relevance for the Database world?

In memory columnar storage disk files mapped to memory

Motivation

Memory mapped columnar storage and NUMA effects, in analytic workload

TPC-H Q1 … (4 sockets, 100GB, MonetDB)



numactl -N 0,1 -m “Varied between sockets 0-3” “Database server process”

Contributions

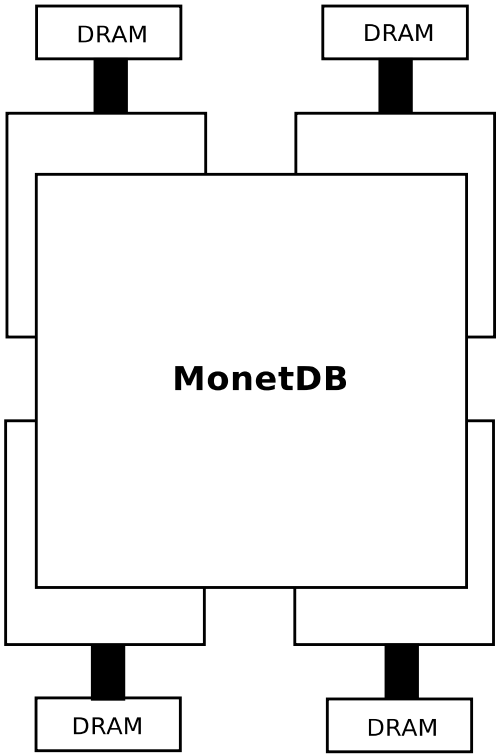
- NUMA oblivious (shared-everything) is relatively good compared to NUMA aware (shared-nothing). (using SQL workload)
- Effect of memory mapping on NUMA obliviousness insights. (using micro-benchmarks)
- Distributed database system using multi-sockets (shared-nothing) reduces remote memory accesses.

NUMA oblivious vs NUMA aware plans

- **NUMA_Obliv- (shared everything)**

  Default parallel plans in MonetDB

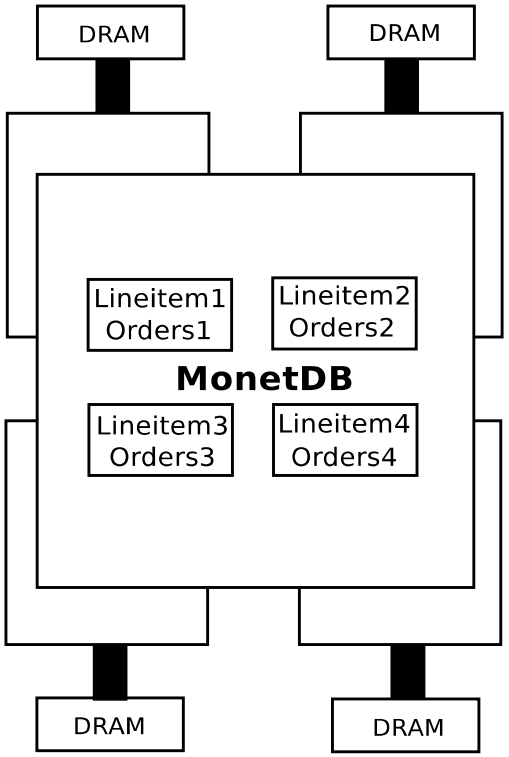
  Only “Lineitem” table is sliced



- **NUMA_Shard- (Variation of NUMA_Obliv)**

  Shard aware plans in MonetDB

  “Lineitem” and “Orders” table sharded in 4 pieces (orderkey) and sliced

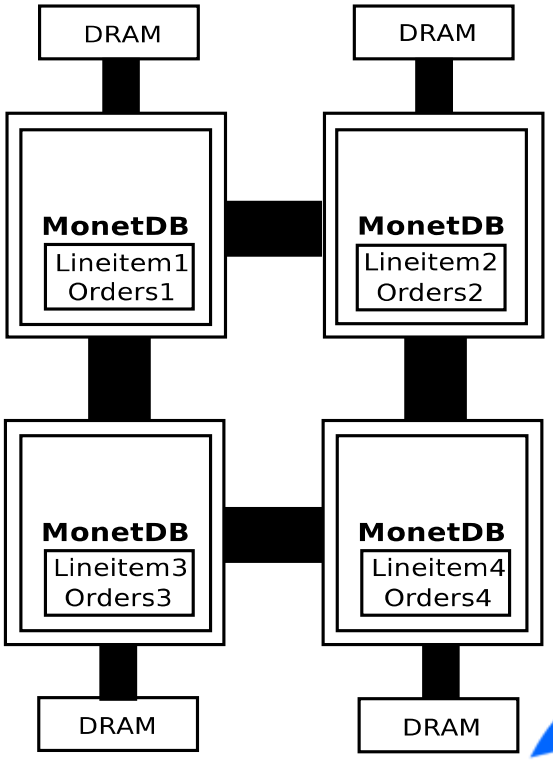


- **NUMA_Distr- (shared nothing)**

  Socket aware plans in MonetDB

  “Lineitem” and “Orders” table sharded in 4 pieces(orderkey), and sliced

  Dimension tables replicated

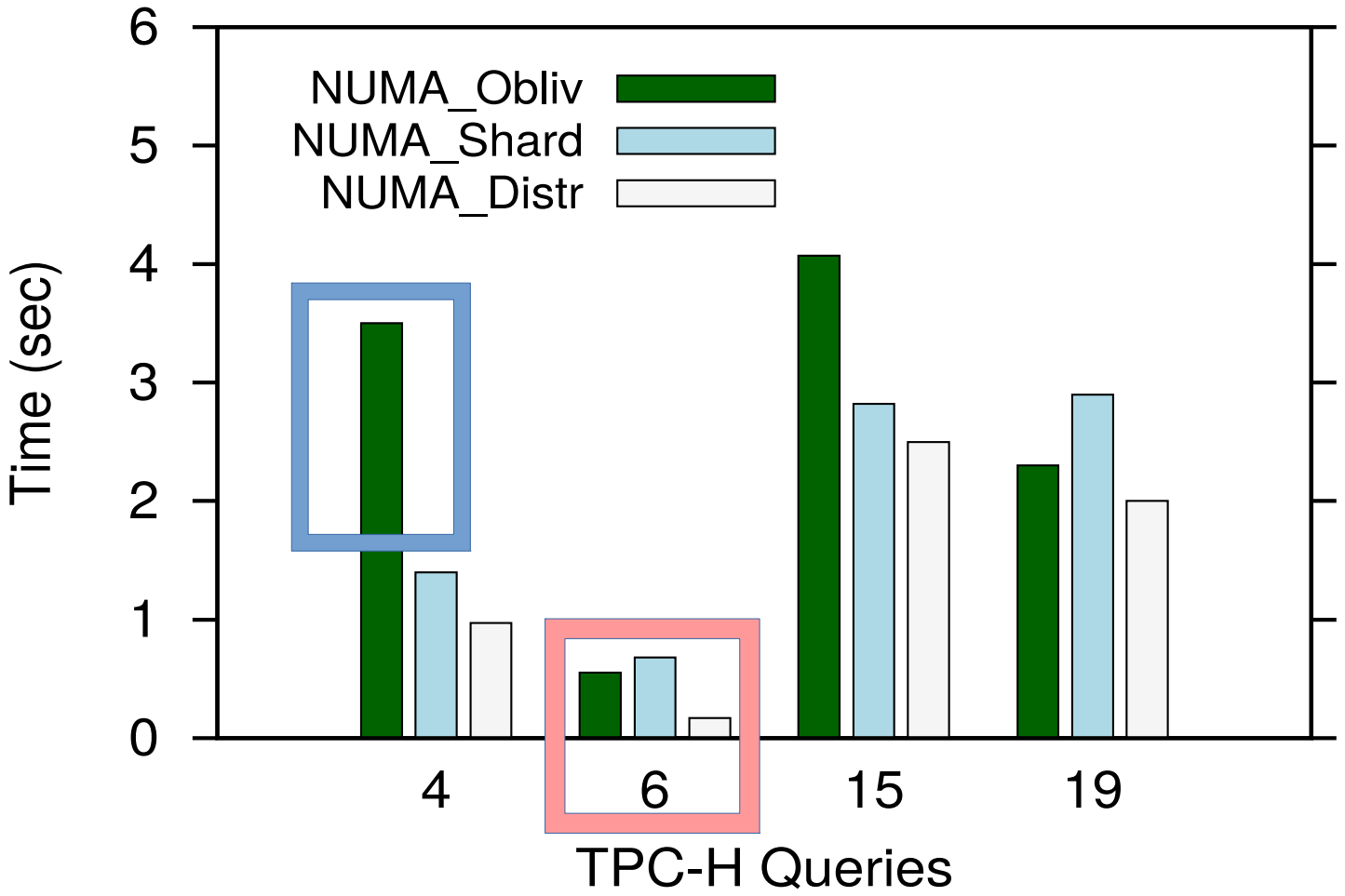


monetdb

System configuration

- Intel Xeon E5-4657L v2 @2.40GHz, 4 sockets, 12 cores per socket (total 96 threads with Hyper-threading)
- Cache - L1=32KB, L2 =256KB, shared L3=30MB.
- 1TB four channel DDR3 memory, (256 GB memory / socket).
- O.S. - Fedora 20 Data-set- TPC-H 100GB
- **Tools** – numactl, Intel PCM, Linux Perf
- **MonetDB** open-source system with memory mapped columnar storage

TPC-H performance



NUMA_Shard is a variation of NUMA_Obliv with sharded & partitioned “orders” table.

Micro-experiments on modified Q6

Why Q6? - **select count(*) from lineitem where l_quantity > 24000000;**

- Selection on “lineitem” table
- Easily parallelizable
- NUMA effects analysis is easy (read only query)

Process and memory affinity

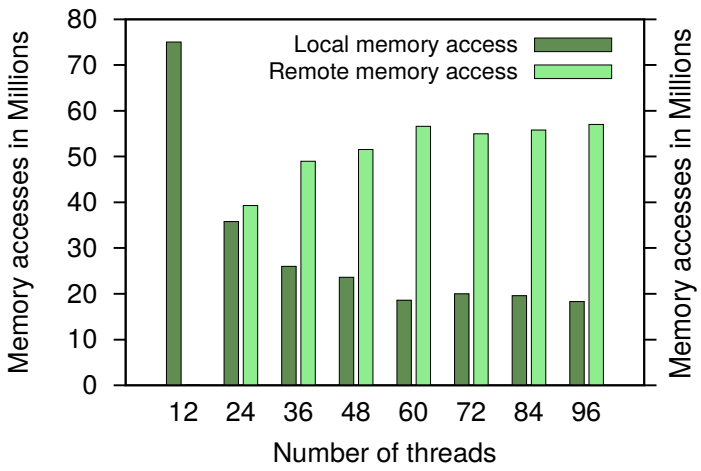
Example:

numactl -C 0-11,12-23,24-35 -m 0,1,2 "Database Server"

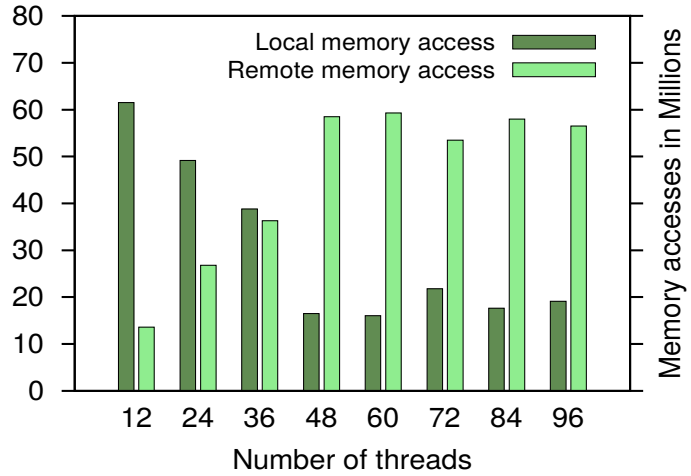
	Socket 0	Socket 1	Socket 2	Socket 3
cores	0-11	12-23	24-35	36-47
cores	48-59	60-71	72-83	84-95

NUMA_Obliv
Micro-experiments on Q6

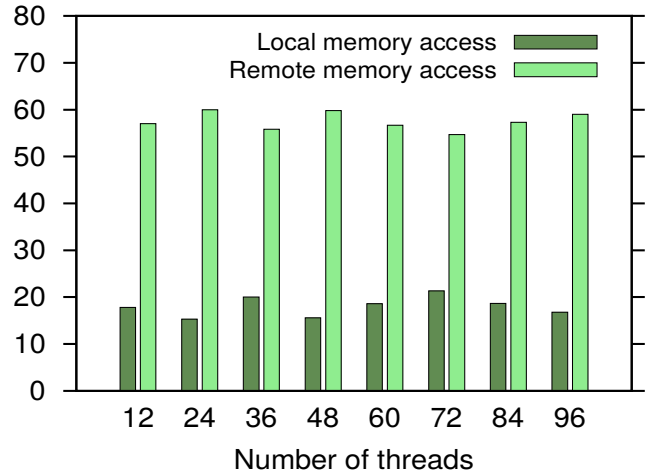
Local vs Remote memory access



PMA= yes, BCC=yes



PMA= no, BCC=yes

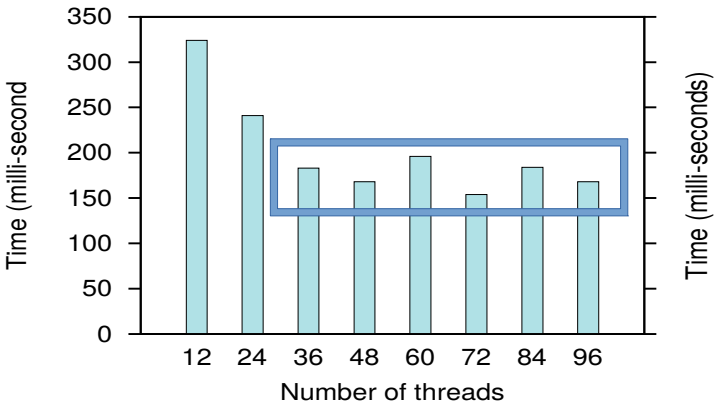


PMA= no, BCC=no

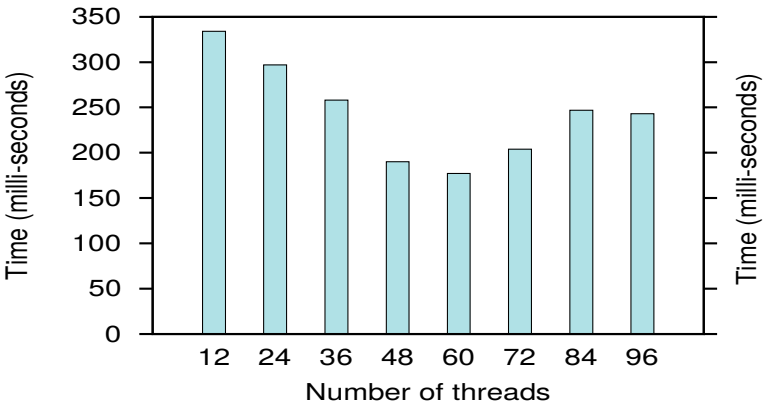
Process and memory affinity = PMA
Buffer cache cleared = BCC (echo 3 | sudo /usr/bin/tee /proc/sys/vm/drop caches)

monetdb

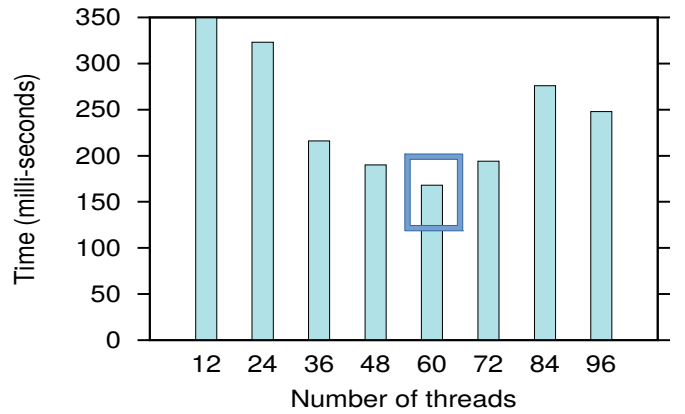
Execution time (Robustness)



PMA= yes, BCC=yes
Most robust



PMA= no, BCC=yes
Less robust



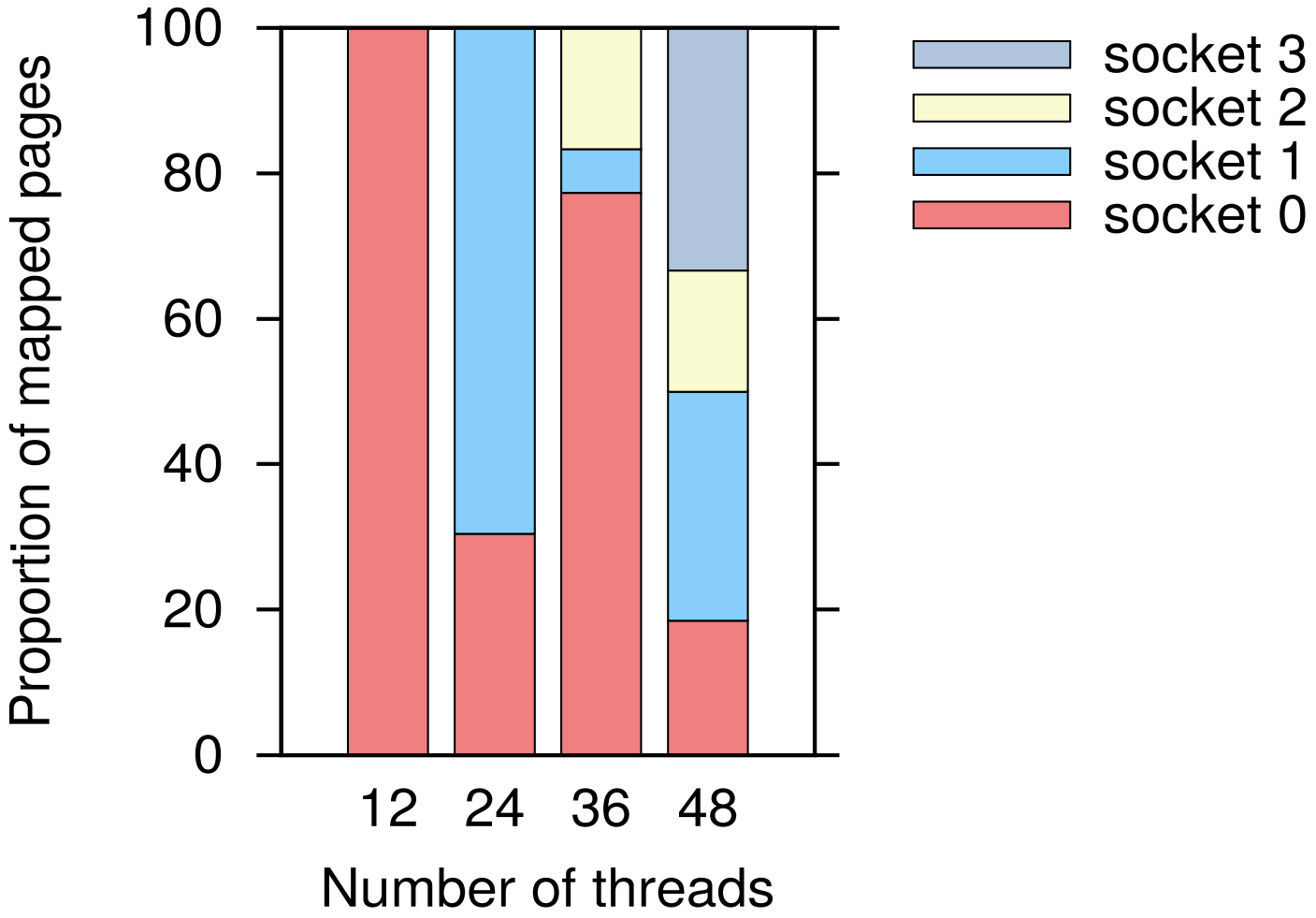
PMA= no, BCC=no
Least robust

Process and memory affinity = PMA
Buffer cache cleared = BCC (echo 3 | sudo /usr/bin/tee /proc/sys/vm/drop caches)

monetdb

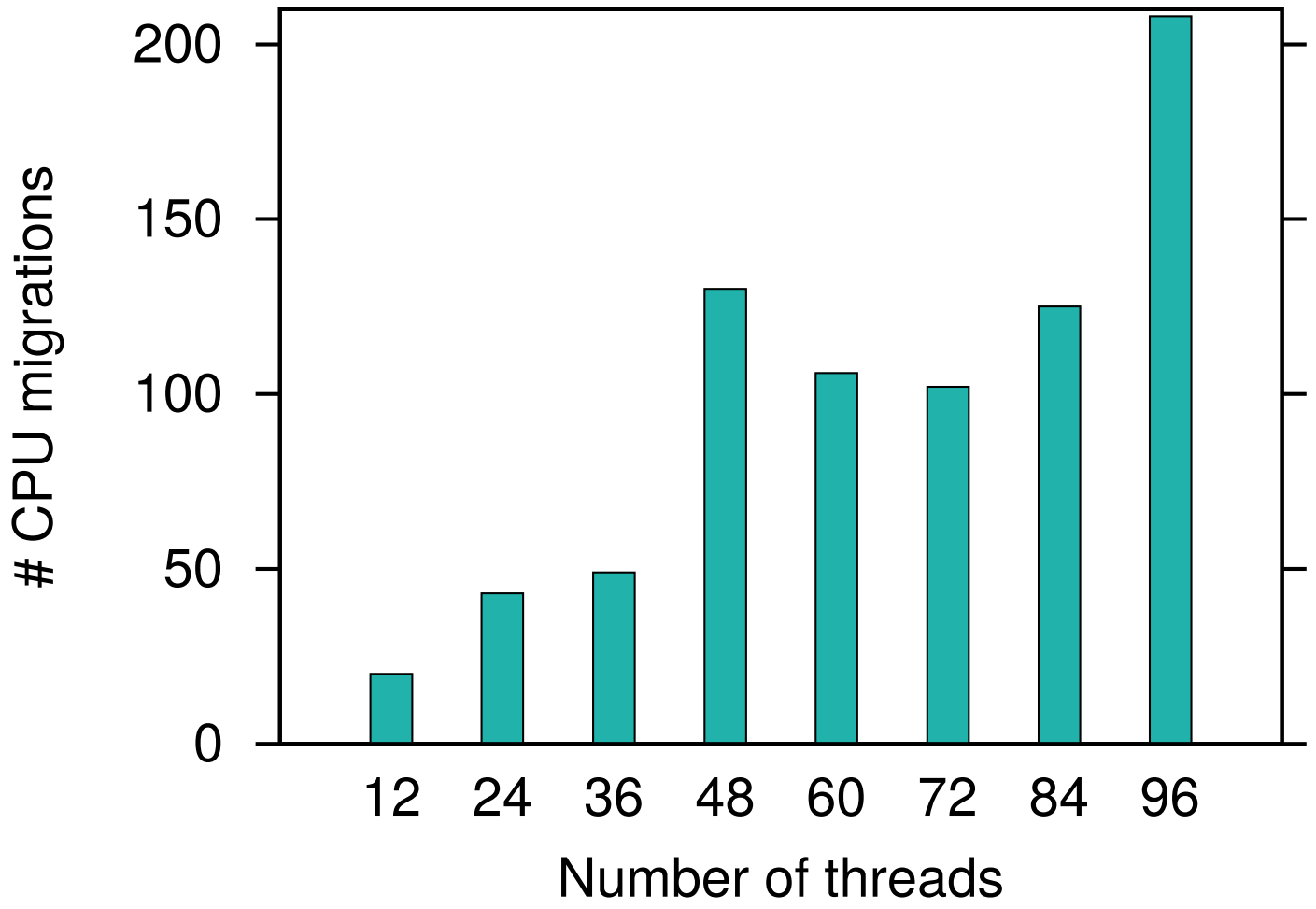
Increase in threads = more remote accesses?

Distribution of mapped pages

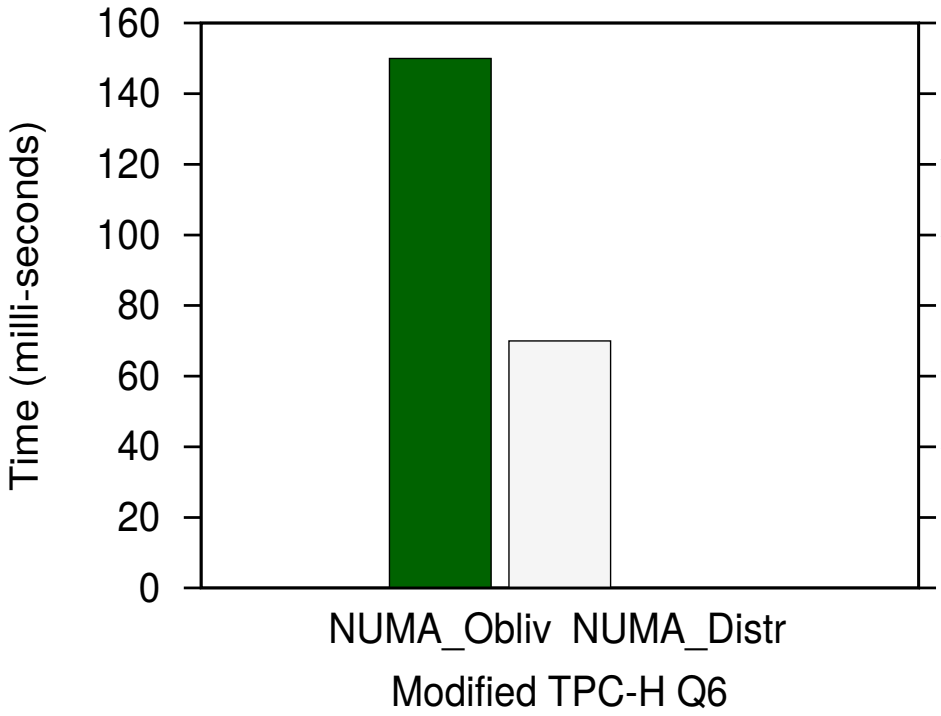


/proc/process id/numa maps

CPU migrations



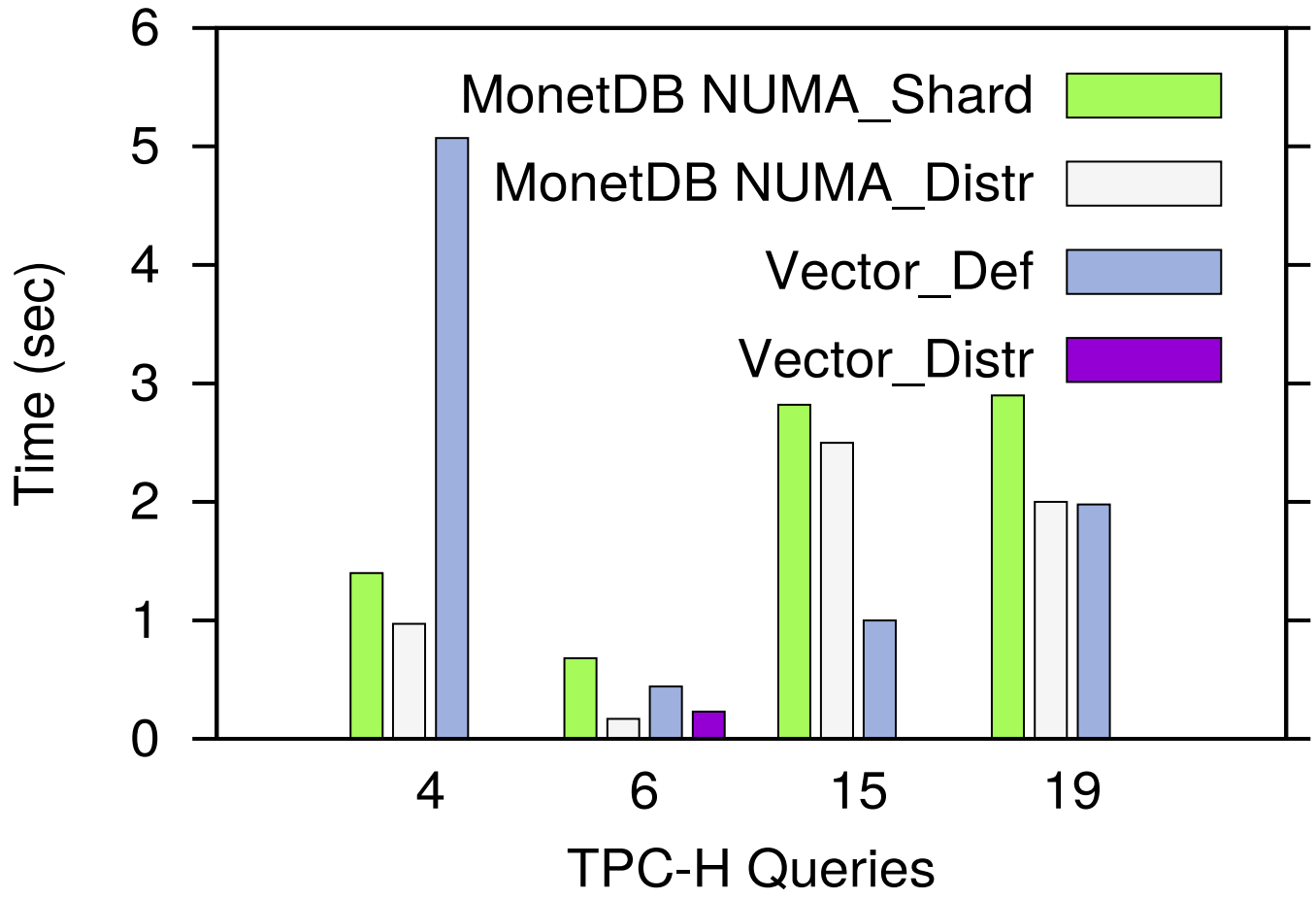
Why remote accesses are bad?



	#Local Access	# Remote Access
NUMA_Obliv	69 Million (M)	136 M
NUMA_Distr	196 M	9 M

NUMA_Distr to minimize remote accesses ?

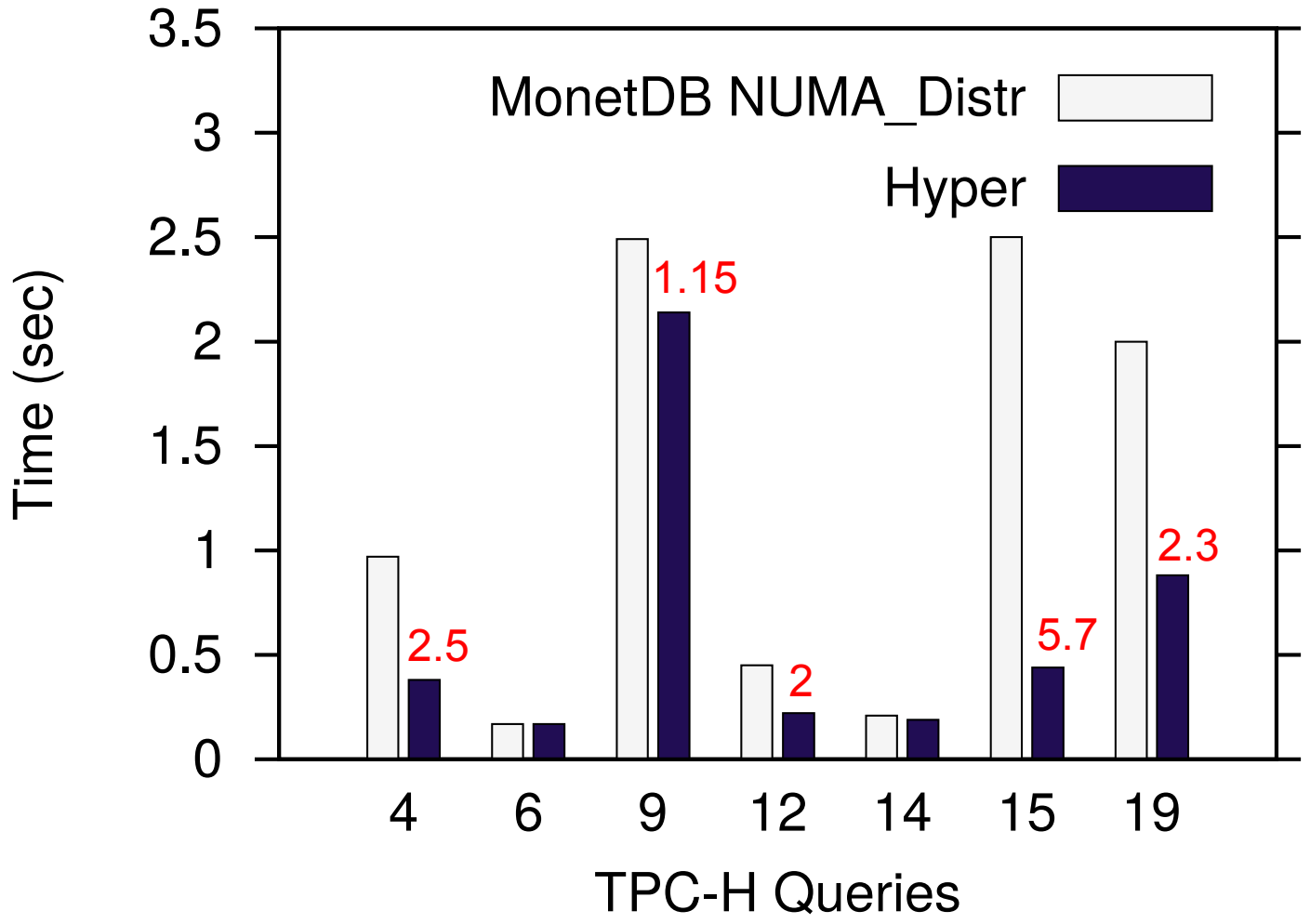
Comparison with Vectorwise



Vectorwise has no NUMA awareness and also uses a dedicated buffer manager

monetdb

Comparison with Hyper



The RED numbers indicate speed-up of Hyper over MonetDB NUMA_Distr plans.

Hyper generates NUMA aware, LLVM JIT compiled fused operator pipeline plans.

monetdb

Conclusion

- NUMA obliviousness fares reasonably to NUMA awareness.
- Process and memory affinity helps NUMA oblivious plans to perform robustly.
- Simple distributed shared nothing database configuration can compete with the state of the art database.

Thank you