NUMA vs PostgreSQL

Andres Freund
PostgreSQL Developer & Committer
Email: andres@anarazel.de
Email: andres.freund@microsoft.com

https://anarazel.de/talks/2024-10-23-pgconf-eu-numa-vs-postgresql/numa-vs-postgresql.pdf

Microsoft

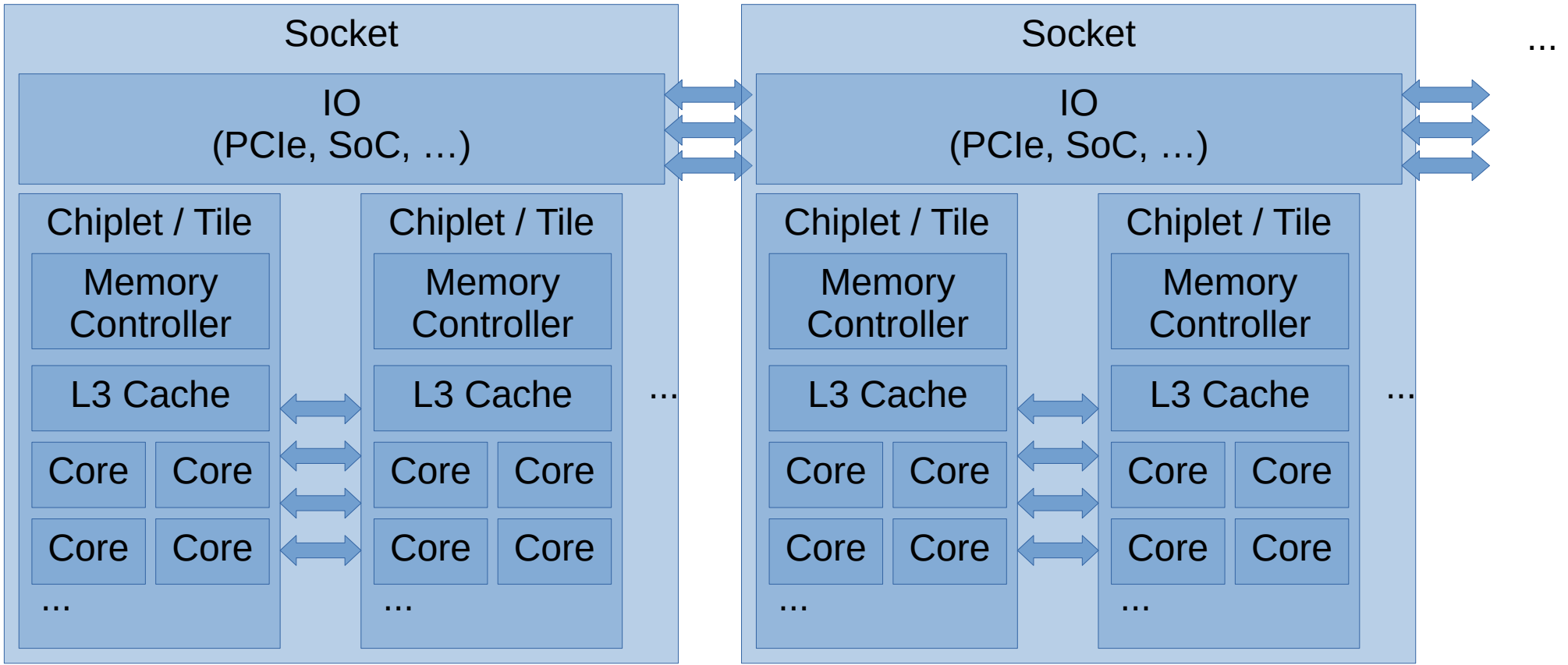
What are we doing here?

- NUMA aware postgres has been discussed a lot – but without concrete projects being identified
- I tend to waste a lot of time with low level hardware stuff
- Don’t have cycles to implement all the fixes
- Tried to **prototype** changes, everything very hacky
- **NOT** claiming **any** identified projects as my own

Why should we work on this?

- “Moore’s law is dead”
  - everyone is moving to “chiplet” style hardware architectures
  - core counts are increasing
- Throughput has improved, latency has effectively gotten **worse**
  - same or worse absolute time, faster clock speeds
  - cross-chiplet / socket latencies have increased

NUMA

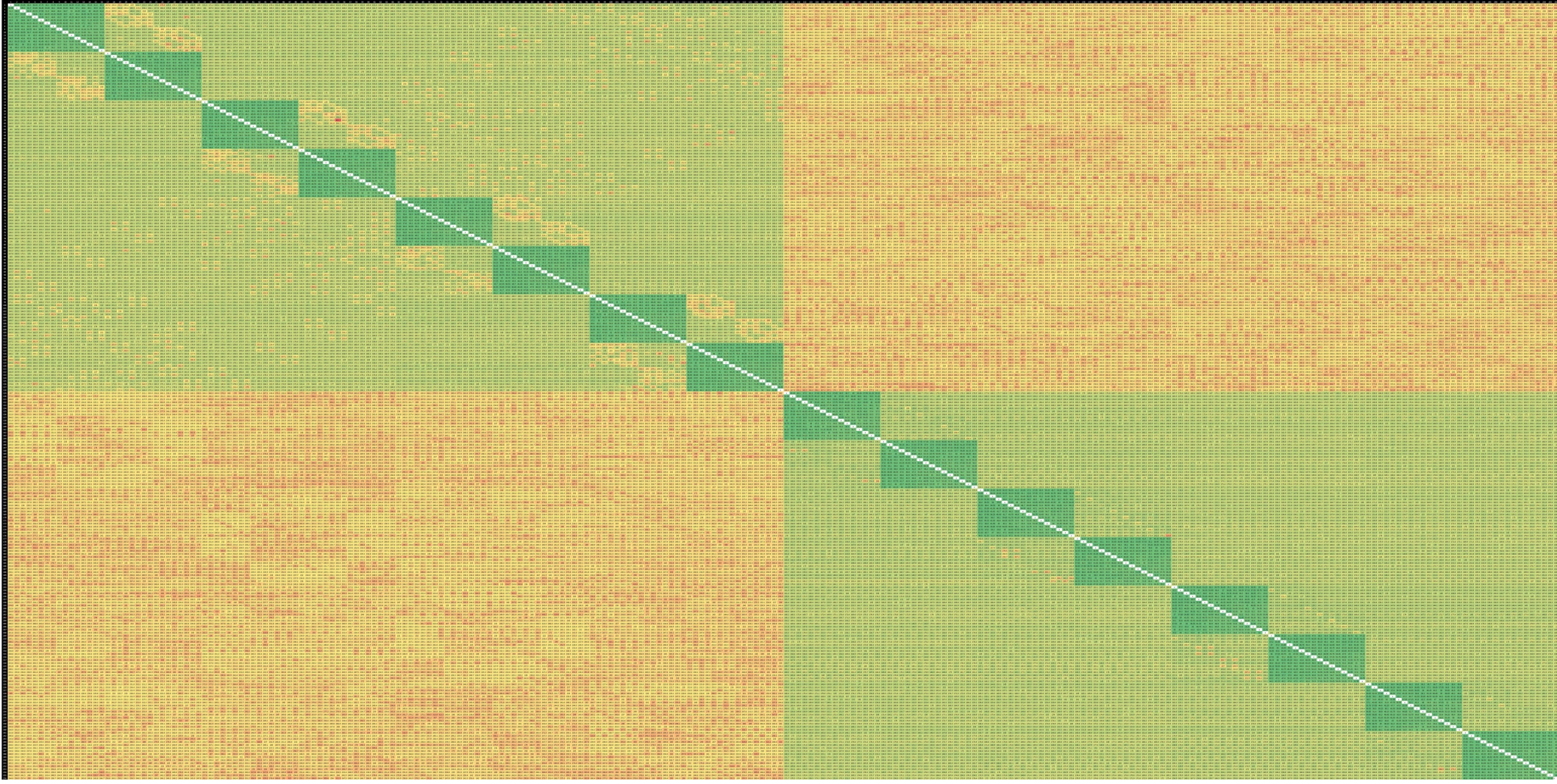


VERY APPROXIMATE – DETAILS VARY

Microsoft

Impact

- Increased Latency
  - base memory latency: ~ 80-140ns
  - cross socket: + ~~ 80-100ns
  - cross tile: + ~~ 30ns
  - Biggest issue: Contended Lock
  - Also bad: Latency sensitive data like hashtables
- Decreased Throughput
- Cached in L1-3 – there is no perf difference for cached accesses



https://github.com/nviennot/core-to-core-latency
https://chipsandcheese.com/p/amds-turin-5th-gen-epyc-launched

Microsoft

“Official” vs “Inofficial”

- official
  - NUMA visible to OS / applications
  - can be addressed using NUMA aware code
- inofficial
  - some latency difference without being visible
  - throughput less affected
  - can be addressed by making code more scalable in generic ways

Numa on Linux

- default allocation policy: local node
- allocation on first use (not mmap()/malloc()!)
  - pg_prewarm() etc will lead to unbalanced memory!
- NUMA balancing tries to move memory around
  - `/proc/sys/kernel/numa_balancing`

Problem #1 – Visibility

- Currently no postgres level insights available
- Minimum: add NUMA information to pg_buffercache
- Also important:
  - NUMA information for other parts of shared memory
  - NUMA information for memory context stats
  - NUMA information for dynamic shared memory
- Maybe: NUMA information for EXPLAIN (BUFFERS, …)?
- Maybe: Functions to pin backends?

Workaround #1 - Visibility

- OS level stats, via /proc/$pid/numa_maps

Problem #2 – Imbalance

- Allocation on first use: pg_prewarm(), CREATE INDEX, COPY lead to memory on one node being overused
- numactl –interleave=all => also interleaves malloc() style memory allocations
- Secondary issue: First use of memory **much** slower
  - Forcing pre-allocation with MAP_POPULATE triggers memory to be allocated on postmaster's node

Problem #2 – Imbalance

- Workload: #ncpu concurrent sequential scans of independent tables, fits in s_b, --interleave=all, prewarmed
- default:
  - latency average: 382.700 ms
  - latency stddev: 68.596 ms
- interleave=all:
  - latency average: 352.581 ms
  - latency stddev: **7.276** ms

Problem #2 – Imbalance

- Workload: CPU intensive parallel seqscan
- --interleave=all: 1679.224ms
- --interleave=all + numa_set_localalloc(): 1597.208ms

Solution #2 – Imbalance

- Use libnuma to explicitly spread shared_buffers across nodes
- Use libnuma to set default policy for memory allocations to local
- Configuration needed?
- Portability?

Problem #3

- Workload: #ncpu concurrent sequential scans of independent tables, fits in s_b, --interleave=all
- Zen 4 laptop (7840U)
  - “naturally filled”: avg 559.658ms
  - “prewarmed”: avg 539.189ms (3.8% faster)
- 2x Xeon Gold 6442Y
  - “naturally filled”: avg 413.757ms
  - “prewarmed”: avg 375.201ms (10.2% faster)

2x Xeon Gold 6442Y

“naturally filled”

```
   931,652,902,170      dTLB-loads
        28,666,216      dTLB-load-misses                #    0.00% of all dTLB cache accesses
     1,264,689,154      LLC-loads
     1,144,084,854      LLC-load-misses                 #   90.46% of all LL-cache accesses
   790,249,841,952      cycles
 2,865,494,242,052      instructions                    #    3.63  insn per cycle
```

“prewarmed”

```
   931,414,147,893      dTLB-loads
         8,868,946      dTLB-load-misses                #    0.00% of all dTLB cache accesses
       654,284,801      LLC-loads
       534,562,990      LLC-load-misses                 #   81.70% of all LL-cache accesses
   723,453,968,846      cycles
 2,864,166,151,433      instructions                    #    3.96  insn per cycle
```

Microsoft

Problem #3 – Buffer Replacement

- clocksweep in Buffer ID order → victim buffer IDs often have “sequential chunks”
- concurrent clocksweep ➡ concurrent scans are less often consecutive
- ➡ less dense buffer accesses ➡ more TLB misses
- ➡ fewer reads can be combined into shorter readv() vectors ➡ slower reads

Solution #3 – Buffer Replacement

- Partition freelist & clock sweep by the number of cores
- Partition boundaries at huge_page_size boundaries
- Occasionally balance between freelist & clock sweeps if one backend / core is busier
- Co-locate BufferDesc and buffer data
  - huge_page_size=2MB➡256MB on one node

    (((2 * 1024 * 1024) / 64) * 8192) / (1024 * 1024) = 256

Problem #4 – Buffer Lock Contention

- SELECT abalance, bbalance
  FROM pgbench_accounts
  JOIN pgbench_branches USING (bid)
  WHERE aid = :aid;
  10 statements pipelined
- Patch to avoid needing to re-find btree root page applied
- Pinned to 1-4 NUMA nodes

#Nodes	1	2	3	4
TPS:	131,912	167,361	94,236	62,357
Sep DBs	131,915	256,811	378,540	515,292

Solution #4 – “Fast Path Buffer Locks”

- Hotly accessed, rarely modified pages are often the worst contended
- Mark buffer as super-locked ➡ no need to pin, lock
- Super-locked page get pinned & locked in per-backend state
- To exclusively lock, all backend-local locks need to be re-acquired
- Hard part: When to acquire super-locks

Outlook – PG Optimizations

- Read-mostly and frequently changing data on same cacheline
  - example: TransamVariablesData, quick fix: 50% increased throughput with lots of subxids
- Procarray: “too dense”, pad and have per-numa node freelists?
- Use huge pages more selectively (e.g. not procarray)

Outlook

- IO: Faster to do IO on NUMA node that has PCIe device attached
- CXL: Memory via PCIe (slower, cheaper, more)
  - + ~~ 200ns latency
  - Secondary bufferpool?
- CXL: Loan Memory from other nodes
  - + ~~ 350ns latency

Add-On: Profiling

- perf c2c can be helpful
  https://anarazel.de/talks/2024-05-29-pgconf-dev-c2c/postgres-perf-c2c.pdf
- Perf events
  E.g. on Intel HW:

```
perf stat --per-node -a -e
        mem_load_l3_miss_retired.remote_dram,
        mem_load_l3_miss_retired.remote_fwd,
        mem_load_l3_miss_retired.remote_hitm,
        mem_load_l3_miss_retired.local_dram,
        uncore_imc/cas_count_read/,
        uncore_imc/cas_count_write/
        -r 0
        sleep 1
```

NUMA vs PostgreSQL

Andres Freund
PostgreSQL Developer & Committer
Email: andres@anarazel.de
Email: andres.freund@microsoft.com

https://anarazel.de/talks/2024-10-23-pgconf-eu-numa-vs-postgresql/numa-vs-postgresql.pdf

Microsoft