The Impact of Process Placement and Oversubscription on Application Performance: A Case Study for Exascale Computing

Florian Wende, Thomas Steinke, Alexander Reinefeld

Zuse Institute Berlin

EASC2015: EXASCALE APPLICATIONS AND SOFTWARE CONFERENCE 2015, 21-23 April 2015

Our Initial Motivation for this Work

- **How to cope with an increasing failure rate on exascale systems?**
  - Cannot expect all components to survive a single program run.
  - Checkpoint/Restart (C/R) is one means to cope with it.
  - We implemented erasure-coded memory C/R in the DFG project FFMK "*Fast and Fault-tolerant Microkernel based System*"

- **Q1 (Process Placement): Where to restart previously crashed processes?**
  - Does process placement matter at all?

- **Q2 (Oversubscription): Do we need exclusive resources after the restart?**
  - If yes: reserve an "emergency allocation"
  - If no: oversubscribe

Broader Question (not just specific to C/R)

- Does oversubscription work for HPC?
  - For almost all applications, some resources will be underutilized, no matter how well balanced the system is.
    - memory wall
    - (MPI) communication overhead
    - imbalanced computation

- From a system provider's view, oversubscription
  - may provide better utilization
  - may save energy

- How from the user's view?

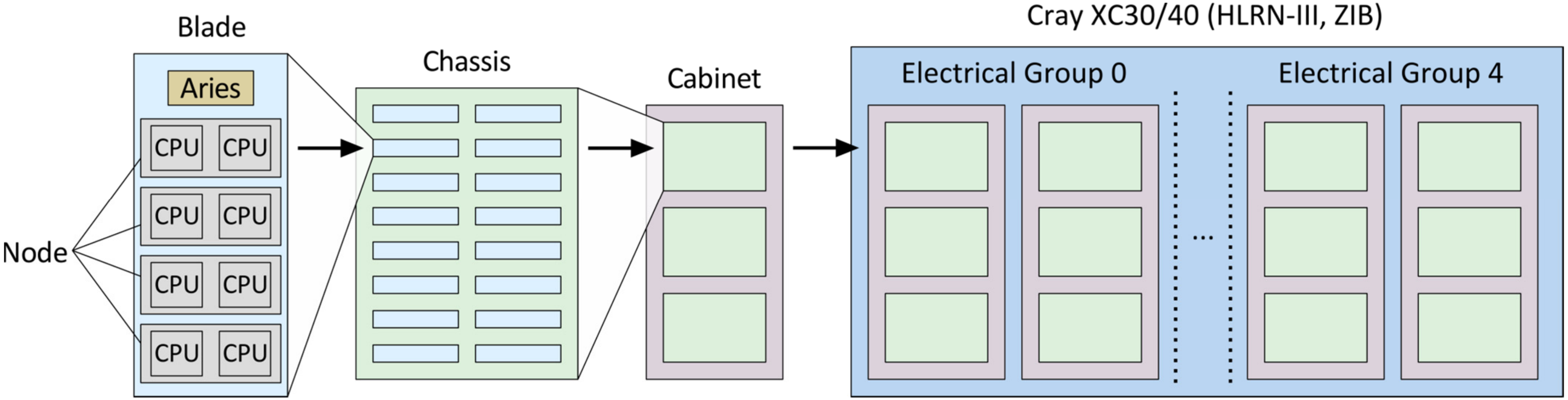
2 Target Systems, 3 HPC Legacy Codes

Cray XC40

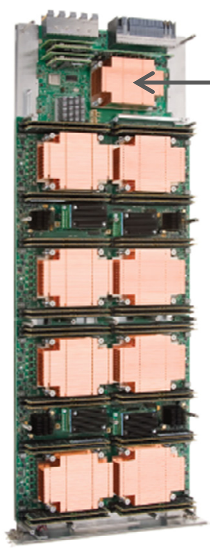
IB Cluster

Cray XC40 "Konrad" @ Zuse Institute Berlin

Cray XC40 Network Topology



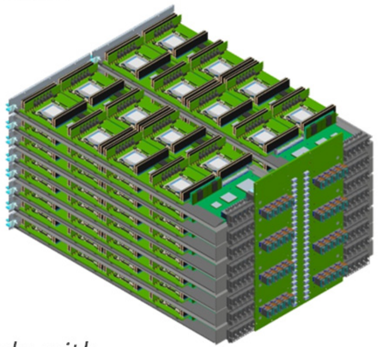
Blade with 4 Nodes



Aries router

Node with two 12 core HSW, IVB

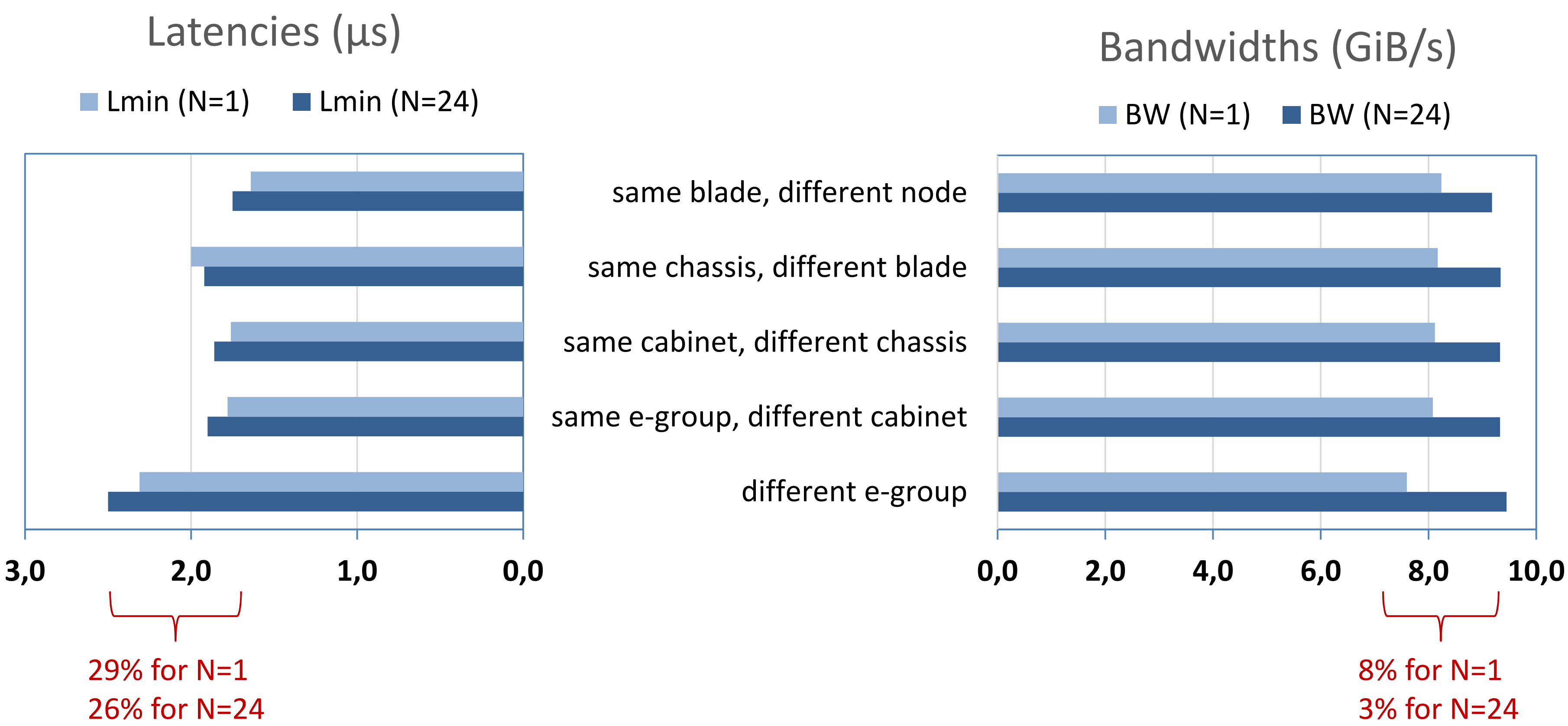
Chassis with 16 Blades



Electrical Group (2 cabinets)

Cray XC40 Network Characteristics

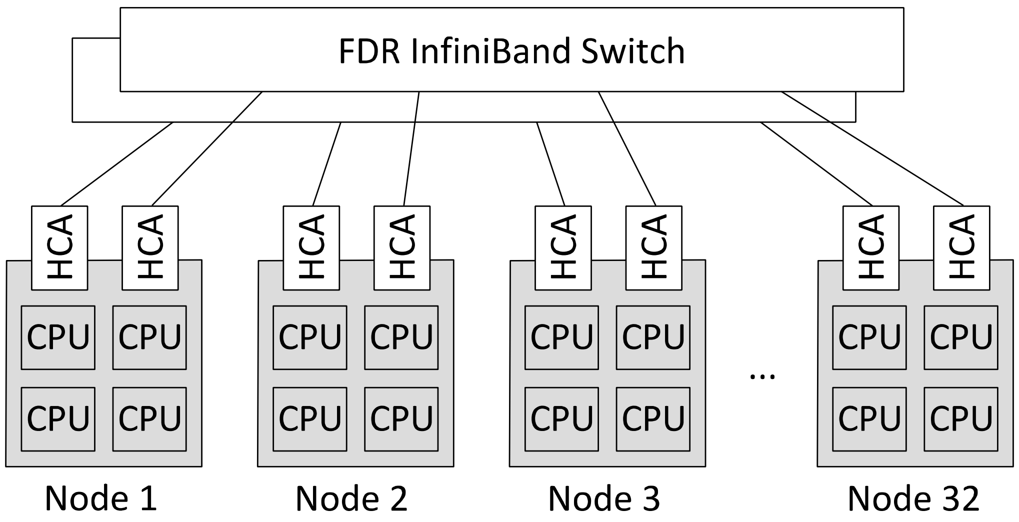
Latency and per-link bandwidth for *N* pairs of MPI processes



Intel MPI pingpong benchmark 4.0: -multi 0 -map n:2 -off_cache -1 -msglog 26:28

InfiniBand Cluster

- 32 Xeon IVB quad-socket nodes
  - 40 CPU cores per node (80 with hyperthreading)
  - Dual port **FDR** InfiniBand adapters (HCA)
    - All nodes connected to 2 IB FDR switches
    - Flat network: latencies down to 1.1µs, bandwidths up to 9 GiB/s saturated



similar results as Cray XC40 (see paper)

Applications

We selected 3 HPC legacy applications with different characteristics:

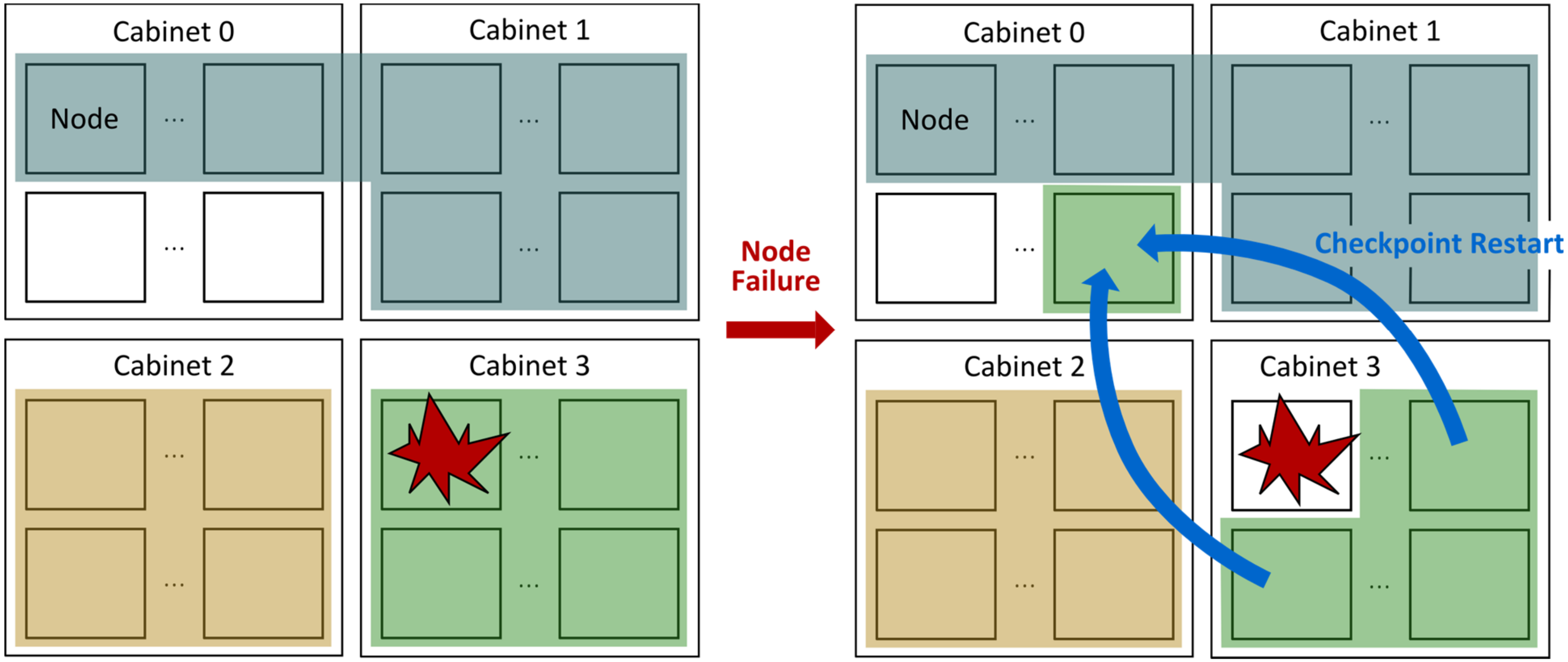
- CP2K
  - atomistic and molecular simulations (uses density functional theory)
- MOM5
  - numerical ocean model based on the hydrostatic primitive equations
- BQCD
  - simulates QCD with the Hybrid Monte-Carlo algorithm

... all compiled with MPI (latest compilers and optimized libraries)

PROCESS PLACEMENT

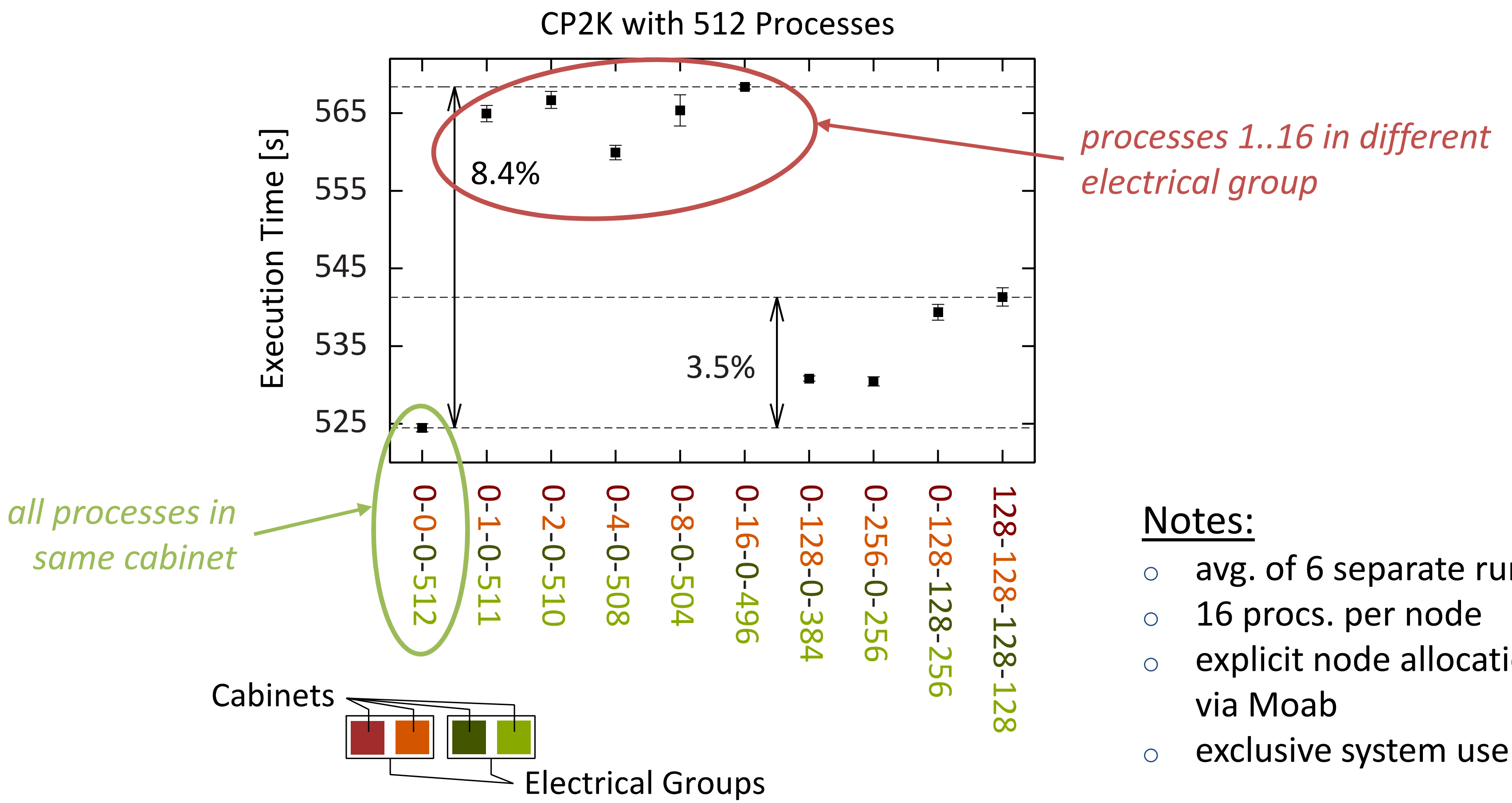
Process Placement

Does it matter where to restart a crashed process?



Process Placement: **CP2K on Cray XC40**

- CP2K setup: H_20-1024 with 5 MD steps
- Placement across 4 cabinets is (color)encoded into string C1-C2-C3-C4

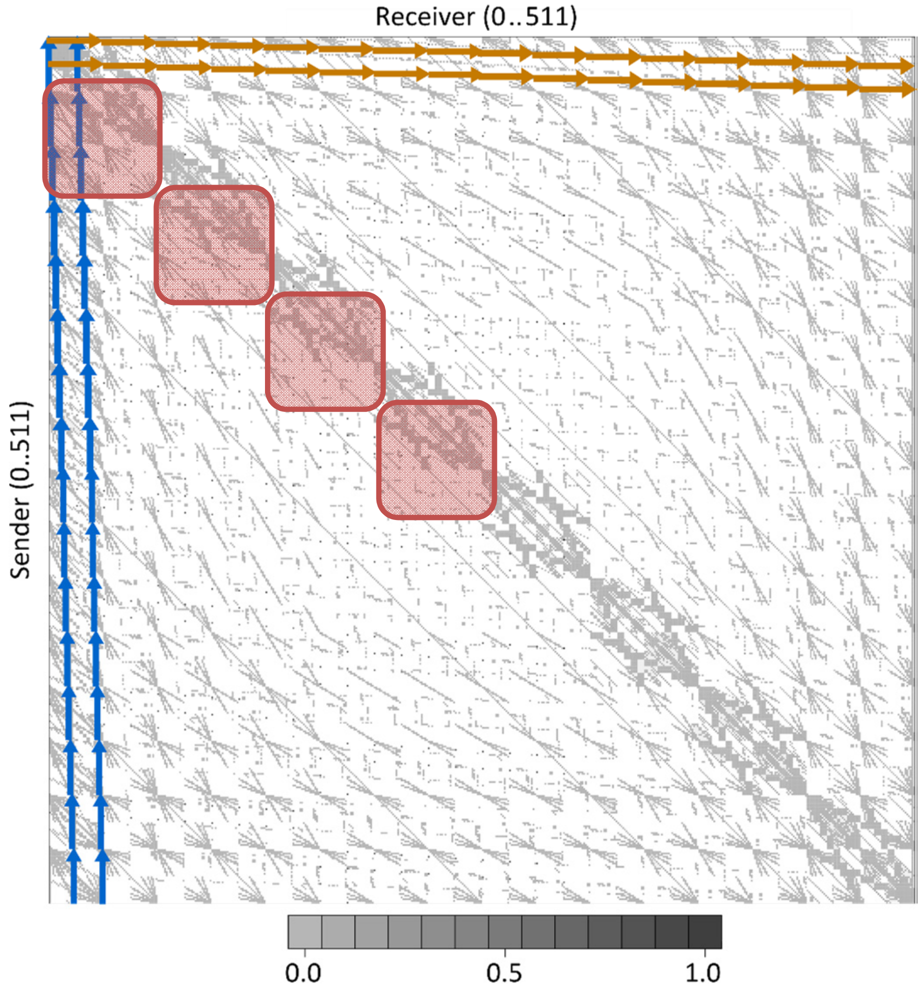


Notes:
- avg. of 6 separate runs
- 16 procs. per node
- explicit node allocation via Moab
- exclusive system use

Process Placement: **CP2K on Cray XC40**

- Communication matrix for H_2O-1024, 512 MPI processes
  - Some MPI ranks are src./dest. of **gather** and **scatter** operations → Placing them far away from other processes may cause performance decrease
  - **Intra-group** and **nearest neighbor** communication

Normalized communication costs (512 MPI processes)



<u>Notes:</u>

- *tracing* experiment with CrayPAT
- some comm. paths pruned away

Process Placement: **Summary**

- Process placement is almost irrelevant: **3 … 8%**
  - Same for all codes (see paper)
  - Same for all architectures: Cray XC40, IB cluster
    - Perhaps not true for systems with “island concept”?

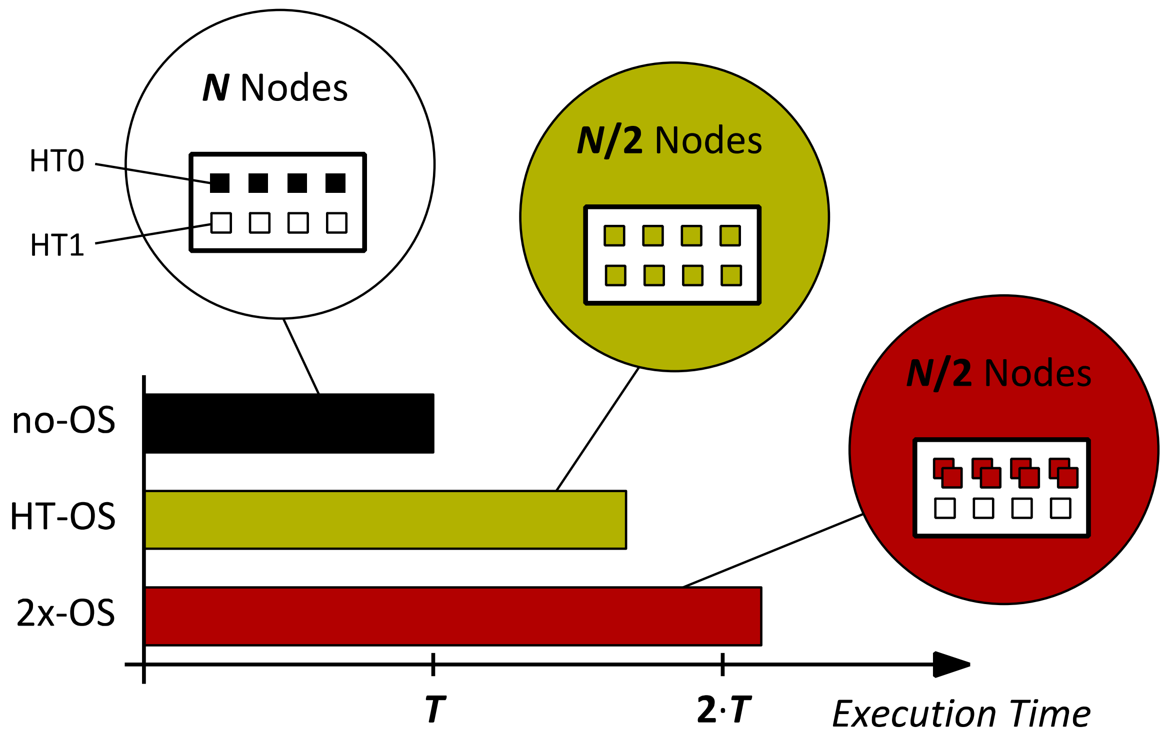
- Worst case (8%) when placing src/dest of collective operations far away from other processes
  - need to identify processes with collective operations and re-map at restart

OVERSUBSCRIPTION

Oversubscription Setups

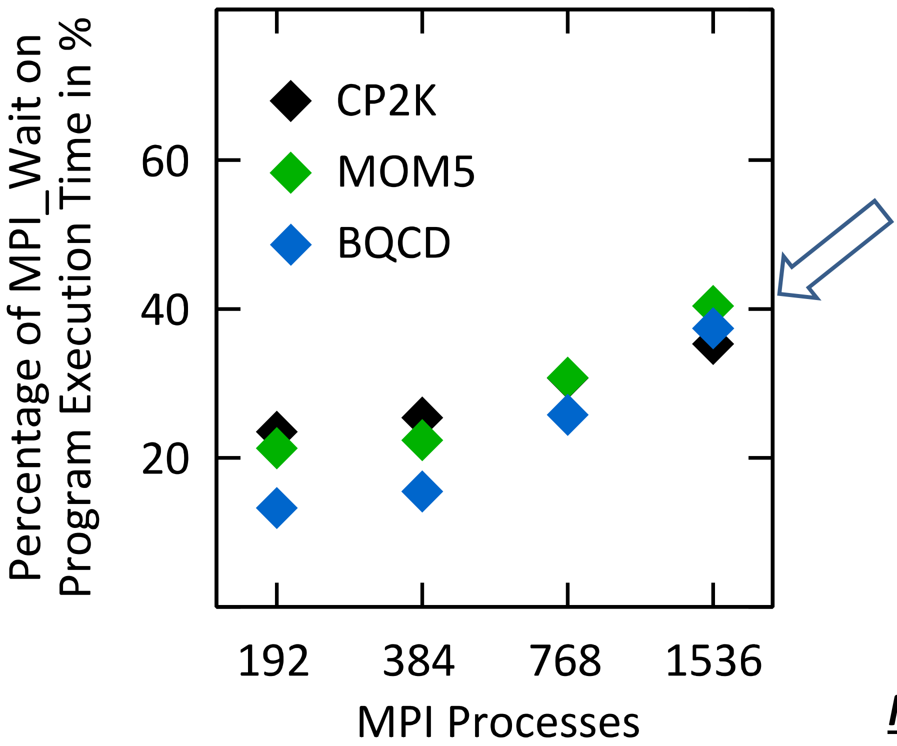
- **no-OS:** 1 process per core on HT0 (hyperthread 0)
- **HT-OS:** 2 processes per core on HT0 & HT1 (scheduled by CPU)
- **2x-OS:** 2 processes per core, both on HT0 (scheduled by operating system)

Note:
*HT-OS and 2x-OS require only half of the compute nodes **N** for a given number of processes (compared to no-OS)*



Percentage of MPI_Wait

MPI is dominated by MPI_Wait for CP2K, MOM5, BQCD



Strong scaling to larger process counts increases the fraction of MPI on program execution time because:

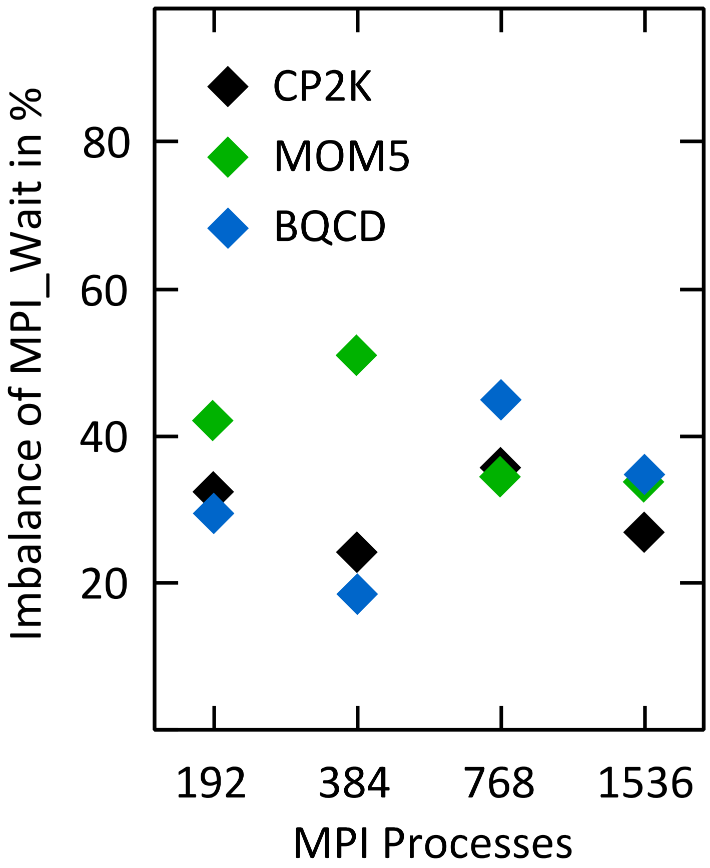
- wait times increase
- imbalances increase
- CPU utilization decreases

Note:

- *24 MPI processes per node*
- *Sampling experiment with CrayPAT*
- *CP2K: H_2O-1024, 5 MD steps*
- *MOM5: Baltic Sea, 1 month*
- *BQCD: MPP benchmark, 48x48x48x80 lattice*

Imbalance of MPI_Wait

Imbalance estimates the fraction of cores not used for computation

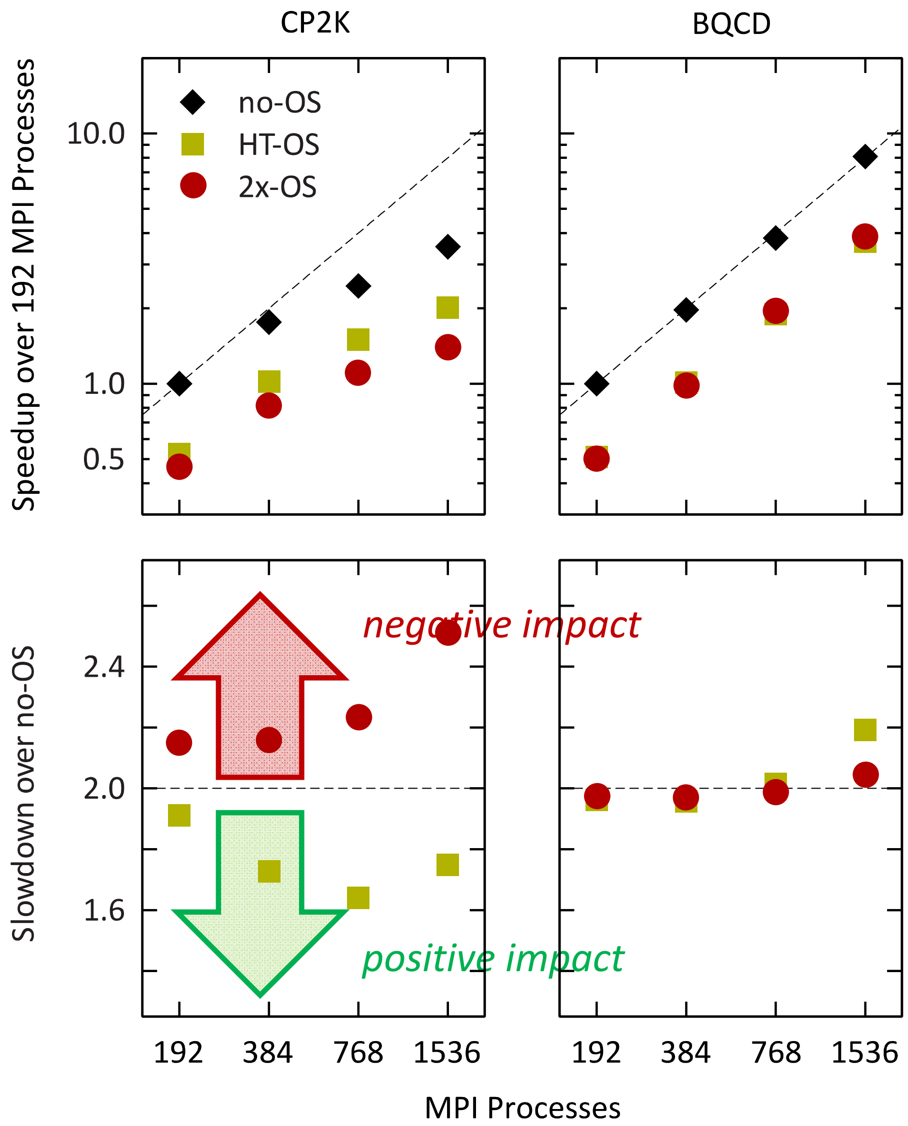


- imbalance $_{\text{(CrayPAT)}} = (X_{avg} – X_{min}) / X_{max}$

- stragglers (i.e. slow processes) have a huge impact on imbalance

Results

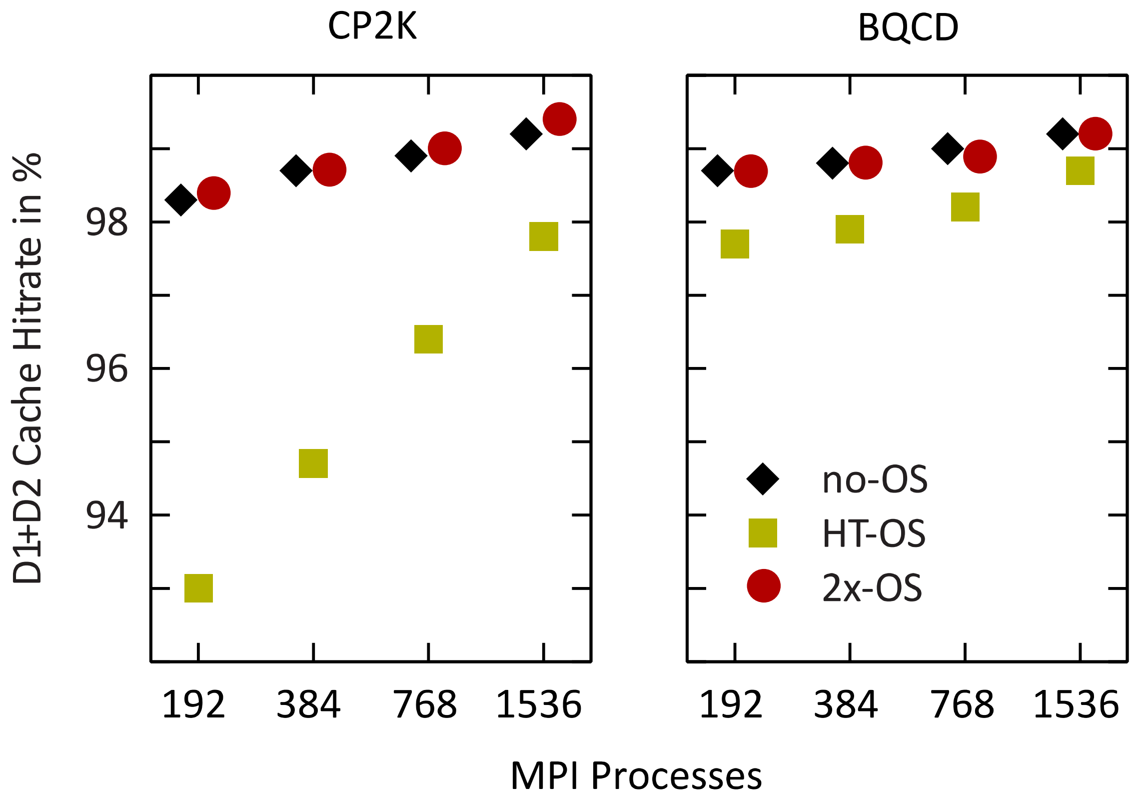
- Impact of Hyper-Threading oversubscription (HT-OS) and 2-fold oversubscription (2x-OS) on program runtime
  - no-OS: 24 p.p.n
  - HT-OS, 2x-OS: 48 p.p.n
  - HT-OS and 2x-OS need only half of the nodes
    - increased shared memory MPI communication
    - cache sharing



2x-OS seems not to work, but HT-OS does!

L1D + L2D Cache Hit Rate

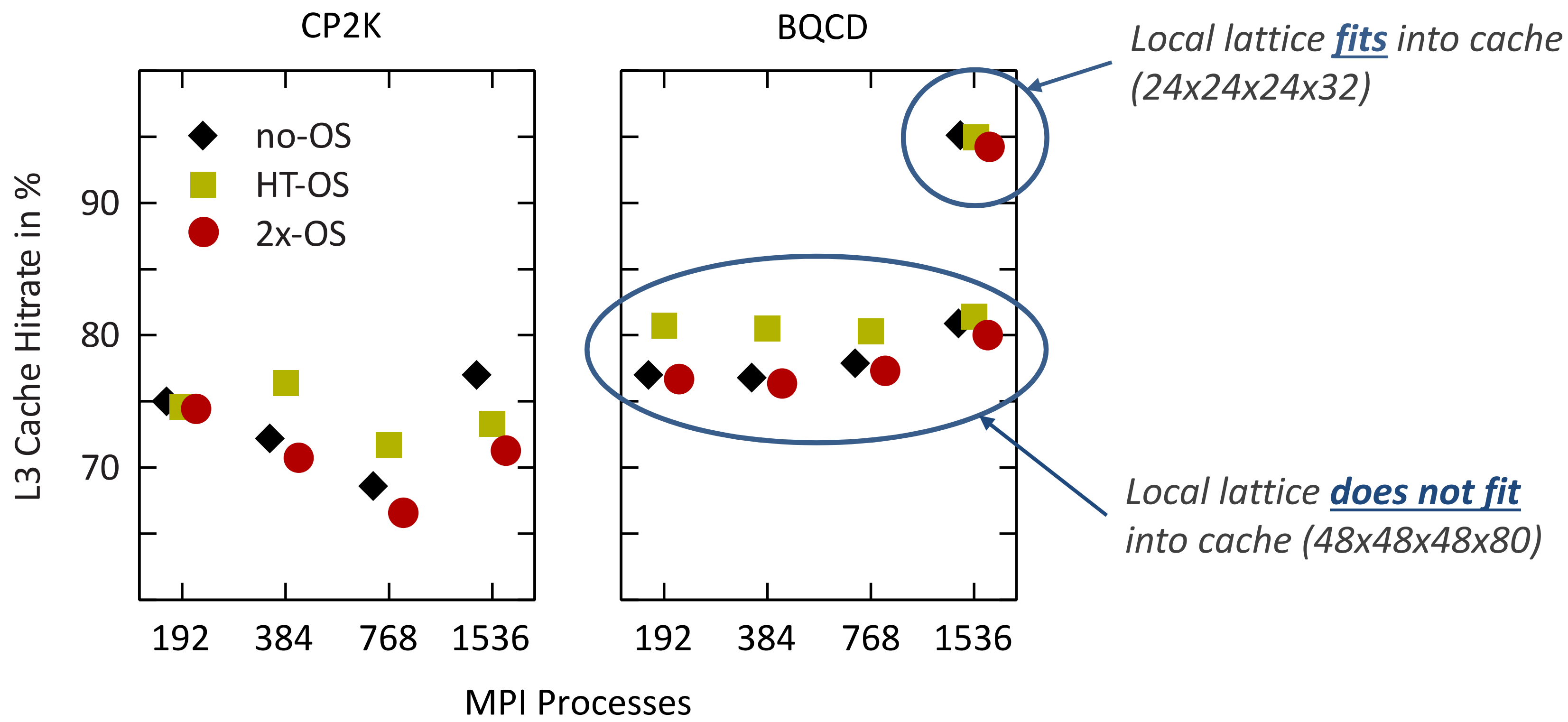
- Lower L1+L2 hit rates for **HT-OS**: processes on HT0 and HT1 are interleaved → mutual cache pollution (not so for 2x-OS with coarse-grained schedules)



measured with CrayPAT
(PAPI performance counters)

L3 Hit Rate

- HT-OS seems to improve caching, 2x-OS does not



measured with CrayPAT
(PAPI performance counters)

Oversubscribing 1 or 2 Applications

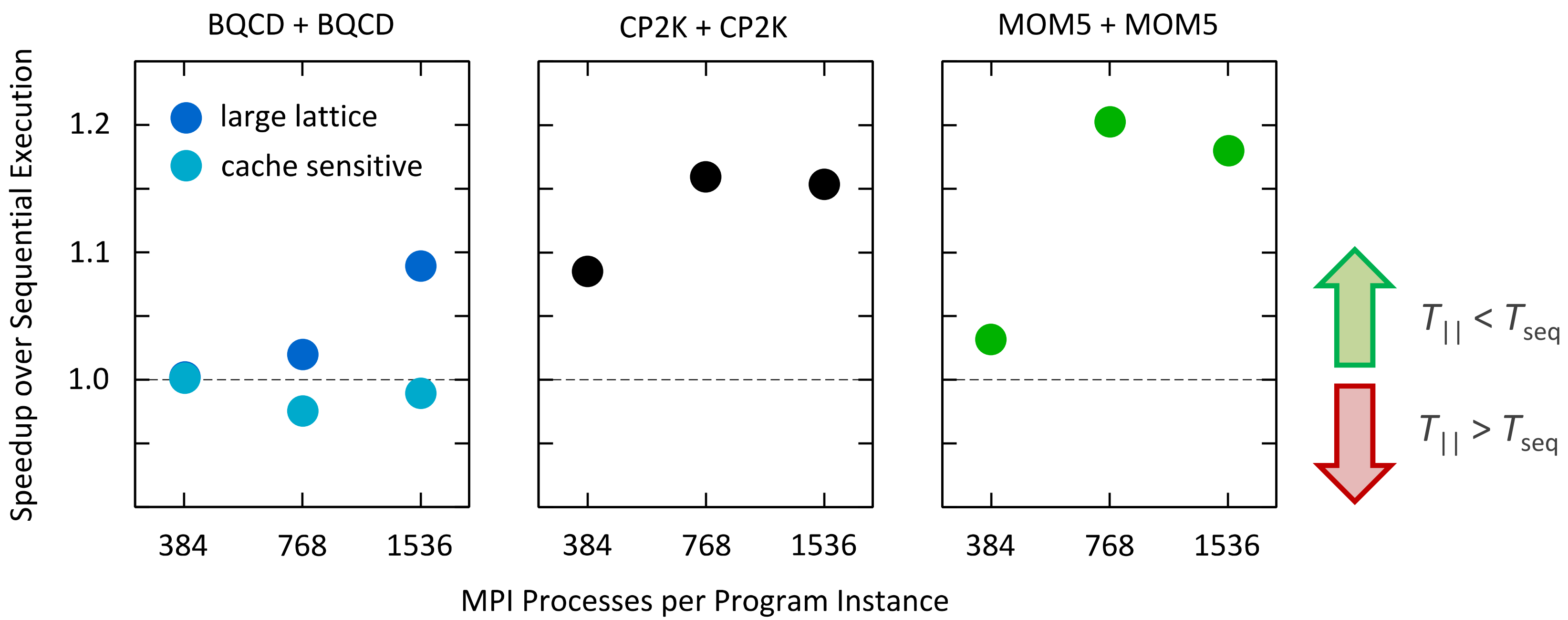
- Above results for HT-OS are with ***one*** application (i.e. $24 \cdot N$ processes on only $N/2$ instead of N nodes)
  - CP2K: 1.6x – 1.9x slowdown (good)
  - MOM5: 1.6x – 2.0x slowdown (good)
  - BQCD: 2.0x – 2.2x slowdown (bad)

  } *with only half of the nodes*

- Does it also work with ***two*** applications?
  - 2 instances of the *same* application
    - e.g. parameter study
  - 2 *different* applications
    - should be beneficial when resource demands of the jobs are orthogonal

Oversubscription: Same Application Twice

- How friendly are the applications for that scenario?
  - Place application side by side to itself
    - Execution times T_1 and T_2 (single instance has execution time T)
    - Two times the same application profile / characteristics / bottlenecks

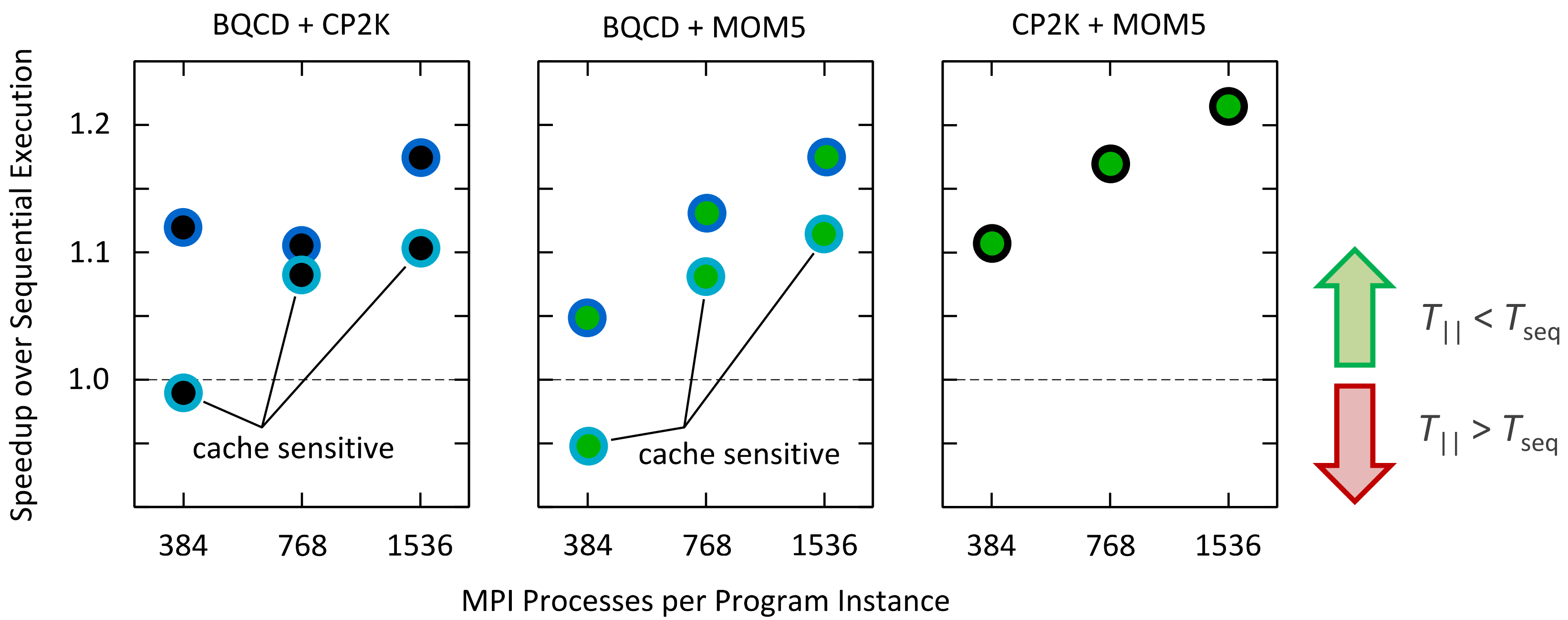


T_{seq} **= 2 · T** *: sequential execution time*

$T_{||}$ **= max(** T_1**,**T_2 **)** *: concurrent execution time*

Oversubscription: Two Different Applications

- Place *different* applications side by side
  - Input setups have been adapted so that executions overlap > 95% of time
  - Execution on XC40 via ALPS_APP_PE environment variable + MPI communicator splitting (no additional overhead)



Summary

- **Process Placement** has little effect on overall performance
  - just 3 … 8%

- **2x-OS** Oversubscription doesn't work
  - coarse time-slice granularity (~8 ms)
  - long sched_latency (CPU must save large state)

- **HT-OS** Oversubscription works surprisingly well
  - Oversubscribing on half of the nodes needs just 1.6 … 2x more time
  - Works for both cases:
    - 2 instances of the *same application*
      - parameter studies
    - 2 *different applications* side by side
      - for all combinations: BQCD+CP2K, BQCD+MOM5, CP2K+MOM5
      - but difficult scheduling

for details see our paper

Disclaimer
- *just 2 Xeon architectures*
- *just 3 apps.*
- *memory may be the limiting factor*