Single-node optimization: still important

Bettina Krammer
Université de Versailles St-Quentin-en-Yvelines
Exascale Computing Research Center
bettina.krammer@uvsq.fr

Quick Tour in 15 Minutes

- This presentation is not a tutorial on single-node optimization techniques, tools nor performance modelling
  - would need a lot more time for that
  - G. Wellein, G. Hager, J. Treibig, *Node-Level Performance Engineering*, Course, LRC, Munich, 6-7 Dec 2012
- Many HPC applications achieve <10% of peak performance
  - High-level optimizations: algorithms, parallel implementation, …
    - Application specialists (domain experts – not computer scientists)
    - Long-term vision: software often in use for decades
  - Low-level optimizations: close to hardware, micro-architectural details, …
    - Performance engineers (computer scientists)
    - Short-term vision: hardware changes frequently
    - Importance of single-node optimization: developers still missing out on many opportunities there

Optimization: Scalability + Performance

1. **Load balancing and avoiding excessive communication should be addressed first**
   - Across nodes
   - Inside node: trend towards hybrid programm
     - Static partitioning: MPI + OpenMP threads
     - Dynamic load-bal hybrid MPI + ta
     - P GPU, MIC,..
2. **Sing ext**
   - E ftwar
     …
     rogramming
   - Redu n (memory,
   - Performance m can guide the way to
3. **Finally, optimizations for massive par**
   - Communication/computation overlap
   - MPI fine-tuning
   - …

Callgrind
Kcachegrind
MAQAO
Likwid
PAPI
Threadspotter
VTune/Amplifier XE
…

Vampir
Score-P
Periscope
TAU
Scalasca
…
Paraver

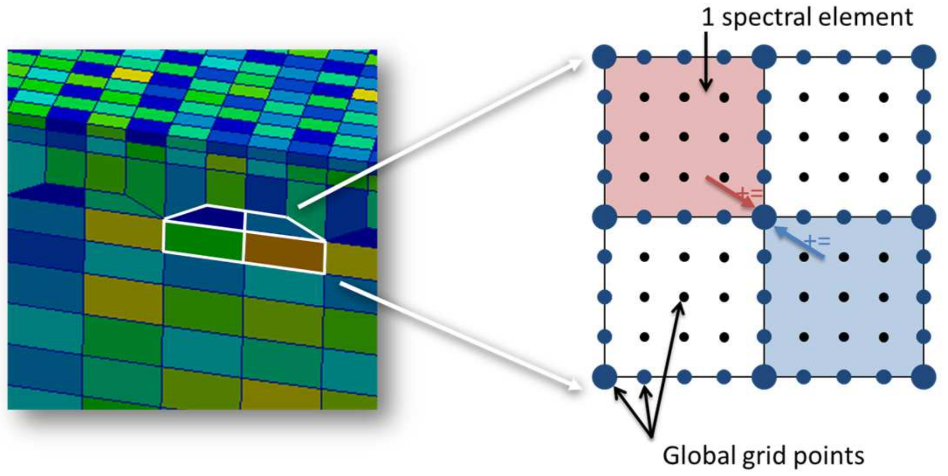
With the aid of (integrated) single-and multi-node tools

Optimization: Scalability + Performance

1. **Load balancing and avoiding excessive communication should be addressed first**
   - Across nodes
   - Inside node: trend towards hybrid programming
     - Static partitioning: MPI + OpenMP threads
     - Dynamic load-balancing with hybrid MPI + task-based models
     - Potentially CPU + accelerator (GPU, MIC,…)
2. **Single-node execution comes next**
   - Exploit underlying hardware and software stack optimally
     - Memory & cache hierarchy, NUMA, …
     - Arithmetic units, vector units, …
     - Compiler, node-level parallel programming and runtime,…
   - Reduce resource consumption (memory footprint, …)
   - Performance model can guide the way to optimal node performance
3. **Finally, optimizations for massive parallelism**
   - Communication/computation overlap
   - MPI fine-tuning
   - …

Hybrid MPI + X

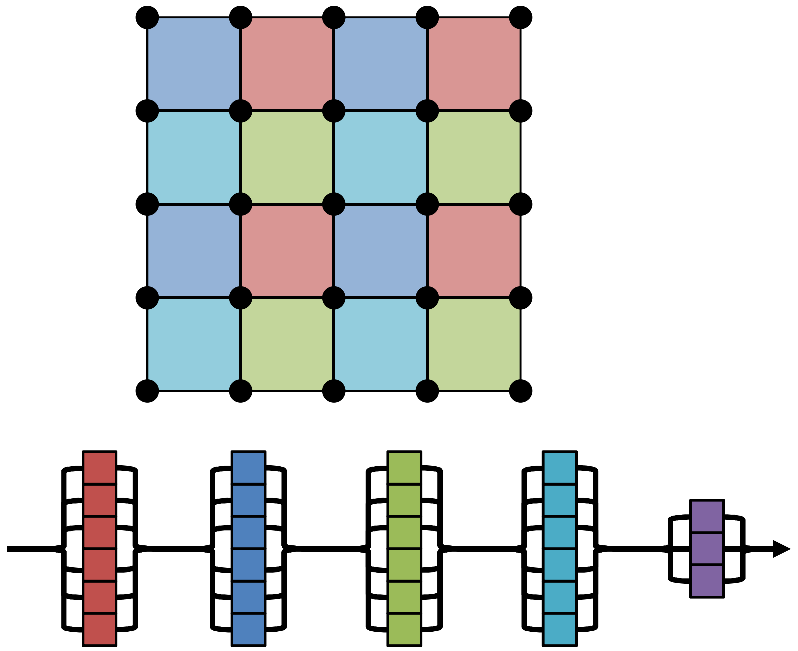
- Commonly used: Hybrid MPI + OpenMP threads
- Current and future work: hybrid MPI + task-based models for dynamic load-balancing inside node
  - First step: elastic forces kernel from specfem3d implemented with cilk vs OpenMP (no MPI yet)
  - Ongoing and future work: hybrid MPI + cilk vs hybrid MPI + OpenMP

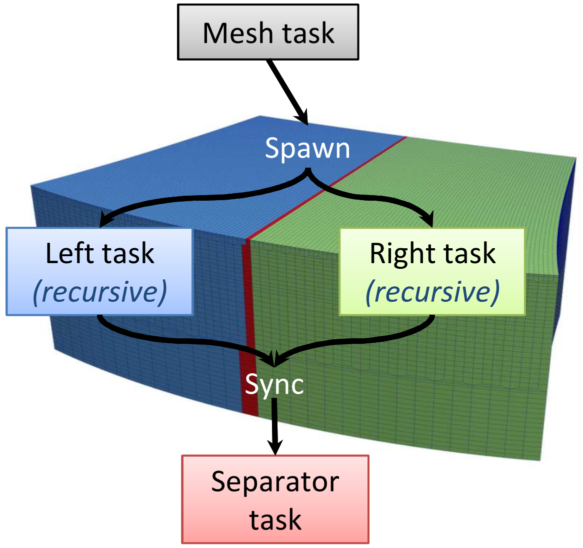


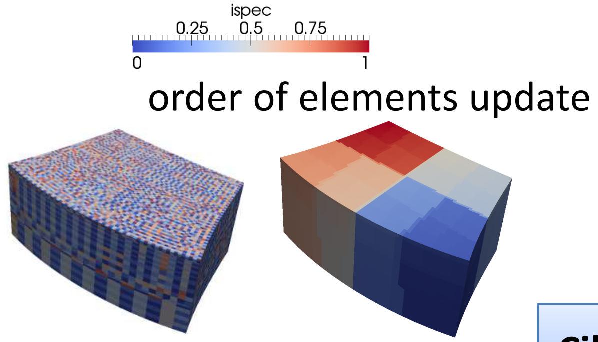
Gather displacements from mesh points → Compute forces for element → Accumulate forces back to mesh

Concurrent writes to global mesh by different spectral elements

OpenMP vs Cilk







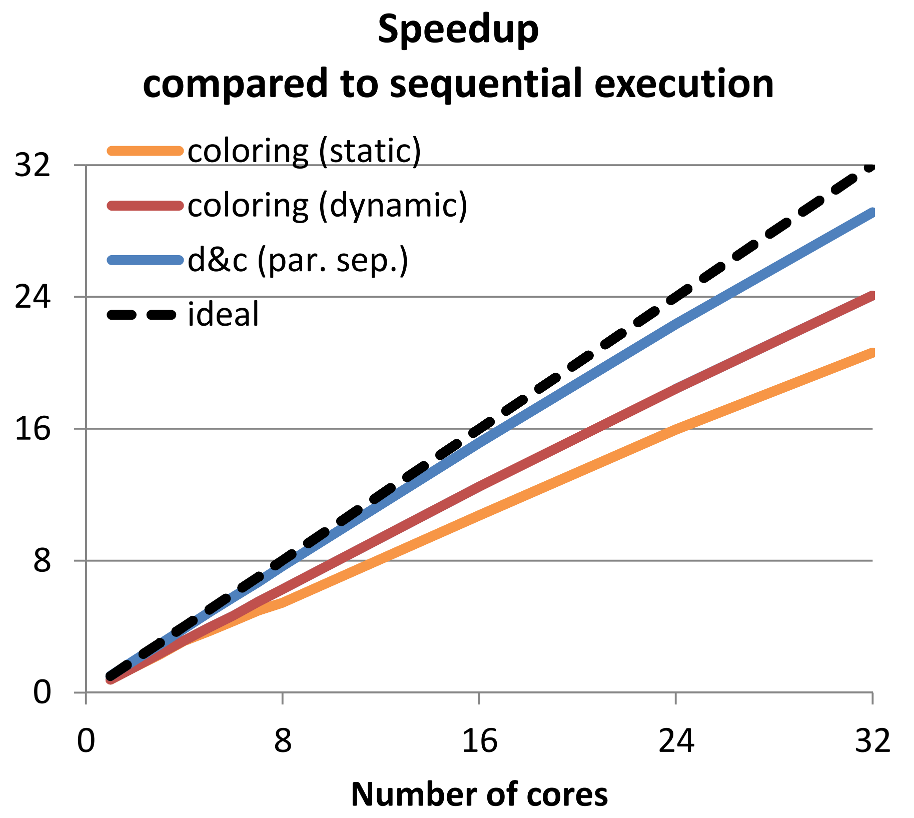
OpenMP with mesh coloring

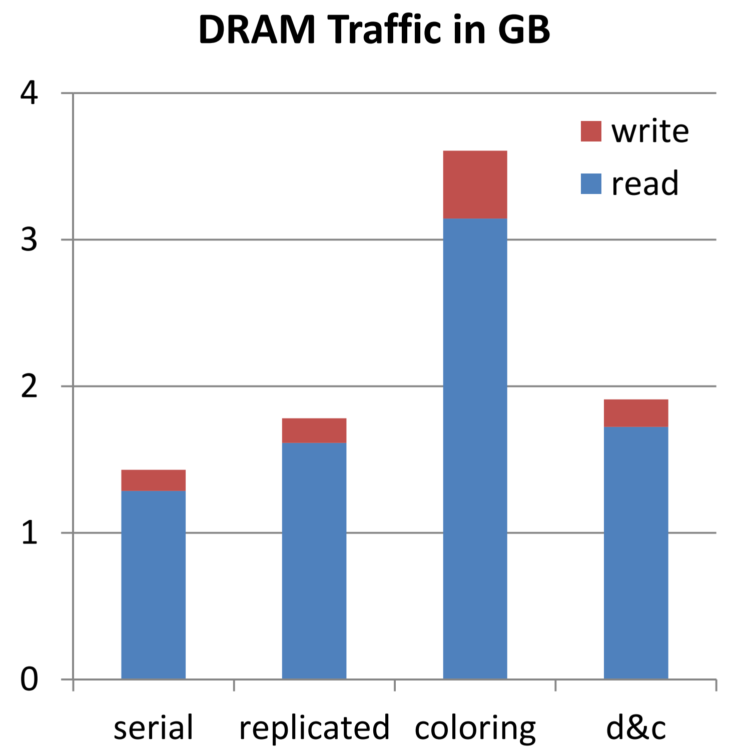
Coloring

D&C

Cilk with Divide & Conquer

Overall Performance

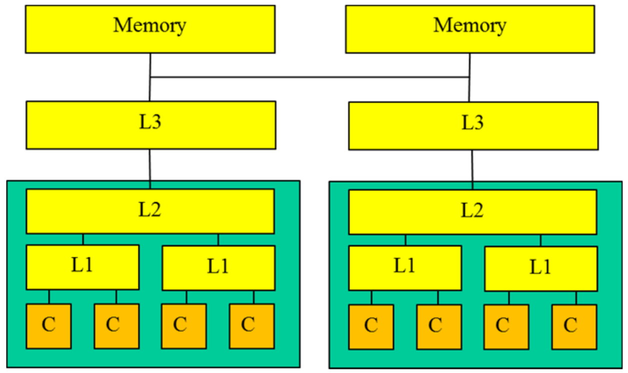




Cilk D&C parallel kernel is **1.2x faster** than the dynamic OpenMP coloring kernel, with a **1.9x DRAM traffic reduction;** better data locality with cilk.
Future work: hybrid MPI + cilk

Hitting the Memory Wall

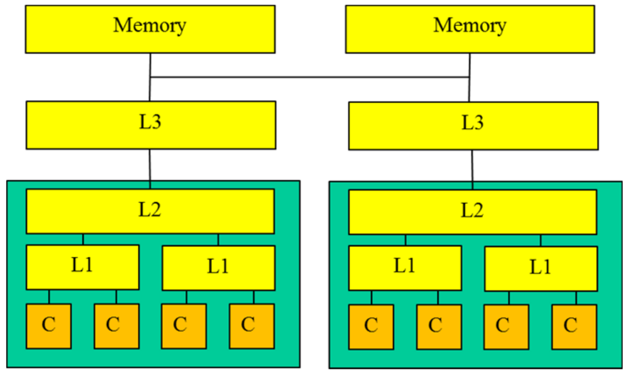
- Know the phases of your code when it is compute- or memory-bound
  - Different optimization strategies for each case
- Memory bandwidth limiting factor for many HPC applications



- Memory latency: alleviated by cache hierarchy
- Temporal/spatial locality – huge impact on perf. !
  - May require data or loop restructuring
  - Prefetching, cache blocking, …
  - Avoid strided or irregular access
    - Use contiguous buffers
    - Domain decomposition: cut multi-dimensional arrays along the slowest axes (depending on programming language – C/C++ or Fortran)

Data Locality and Process/Thread Affinity

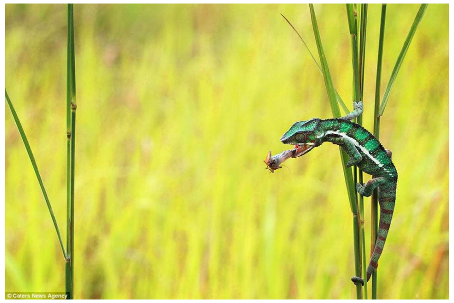
- Beware the NUMA effects
  - Access to remote memory is possible but more costly than to local memory
  - First touch policy (Linux): memory pages mapped into local memory of core first touching variable (write - not allocate!) if enough memory available



- Example: MPI code on *Curie* supercomputer (RTM seismic application mini-app) on up to 5000 cores
  - Default BullMPI: load balanced, no problem
  - But observed huge difference in compute time per core with other MPIs (Open MPI, Mpich, Intel MPI): up to 1.4 x
  - Reason: array allocated and initialized before MPI_Init, resulting in half of the processes accessing non-local memory

Data Locality and Process/Thread Affinity

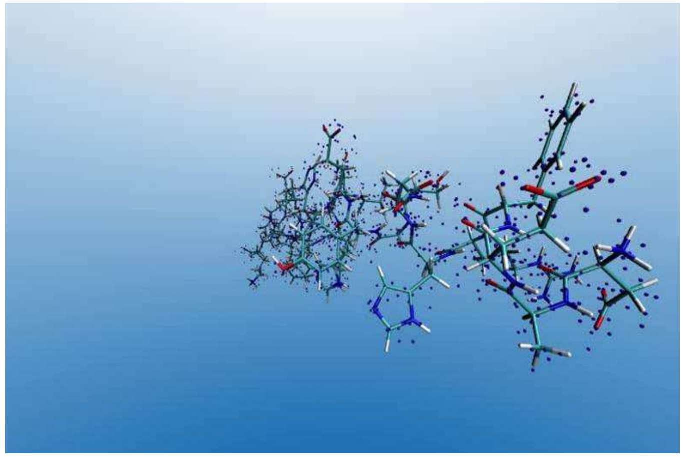
- Ensuring data locality is beneficial in **any** programming model
  - MPI, OpenMP, CnC, …
  - E.g. hybrid MPI+OpenMP: at least one MPI process per NUMA domain
- Pinning processes/threads to cores should be enforced by user or runtime
  - Runtime configuration parameters (depending on MPI, OpenMP,… implementation)
  - Special tools, e.g. Likwid-pin
  - OS dependent commands
  - Runtimes aren't clever enough yet



Loop transformations

- Unrolling, interchange, fusion, fission, …
- Impact of compiler
  - Optimization flags (e.g. -O3, -fast) deliver good results in many cases but can sometimes be outperformed
  - Huge search domain for finding optimum compiler flags combination
    - Machine learning
- Very important: vectorization
  - Compilers may fail: user intervention needed (pragmas, SSE/AVX, code restructuring, …)
  - Can speed-up loop by factor of 2-4 x

Example: QMC=Chem



- Quantum Monte Carlo (QMC) simulation for complex molecular processes, taking place e.g. at Alzheimer's disease
- Low memory footprint but compute-intensive
- Communication only during initialization and finalization, otherwise independent processes
- Fault-tolerant
- Almost ideal scaling with number of cores

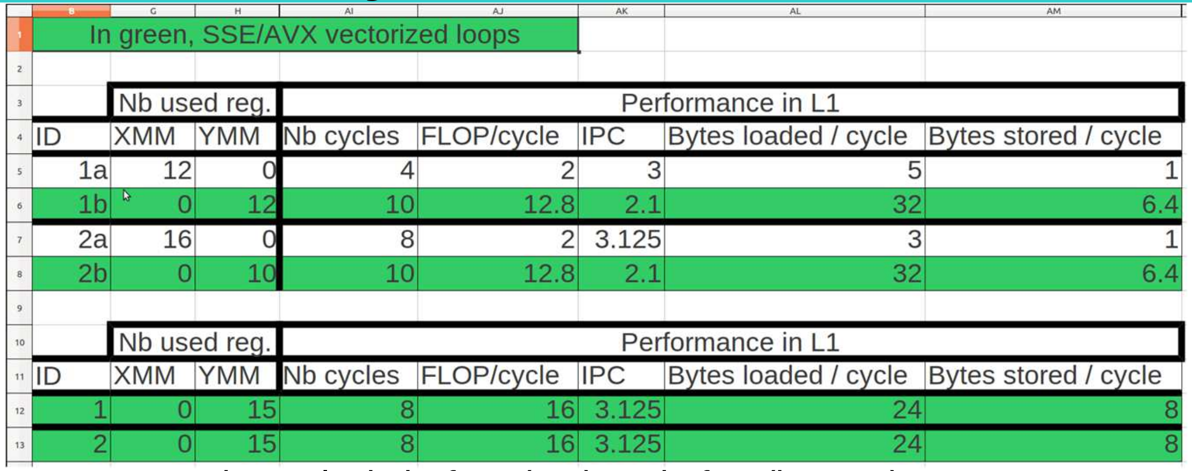
→ Single-core performance determines overall performance

- Bottlenecks identified:
- A matrix inversion, via the Intel MKL library (O(N^3))
- Matrix-matrix products using a sparse-dense implementation (O(N^2))

Dense x Sparse Matrix multiplication Static Analysis

```
!DIR$ VECTOR ALWAYS
!DIR$ VECTOR ALIGNED
do j=1,LDC
 C1(j,i)=C1(j,i)+(A(j,k_vec(1))*d11 &
                + A(j,k_vec(2))*d21 &
                + A(j,k_vec(3))*d31 &
                + A(j,k_vec(4))*d41)

 C2(j,i)=C2(j,i)+(A(j,k_vec(1))*d12 &
                + A(j,k_vec(2))*d22 &
                + A(j,k_vec(3))*d32 &
                + A(j,k_vec(4))*d42)
enddo
```



In green, SSE/AVX vectorized loops

	Nb used reg.		Performance in L1				
ID	XMM	YMM	Nb cycles	FLOP/cycle	IPC	Bytes loaded / cycle	Bytes stored / cycle
1a	12	0	4	2	3	5	1
1b	0	12	10	12.8	2.1	32	6.4
2a	16	0	8	2	3.125	3	1
2b	0	10	10	12.8	2.1	32	6.4

	Nb used reg.		Performance in L1				
ID	XMM	YMM	Nb cycles	FLOP/cycle	IPC	Bytes loaded / cycle	Bytes stored / cycle
1	0	15	8	16	3.125	24	8
2	0	15	8	16	3.125	24	8

MAQAO Static analysis before (top) and after (bottom) optimization

- Examine two hottest loops with MAQAO → obtain theoretically perfect efficiency
  - FLOP/cycle: from 12.8 to 16 (AVX, 32 bits elements, perfect ADD / MUL balance)
  - Loop count (LDC) always a multiple of 8: replace loop count with its hard coded value to allow the compiler to factor loads
- Grand Challenge run on *Curie* end of 2011:
  - **~960 TeraFlops** (mixed precision SP and DP, 200 GigaFlops per node, 4800 nodes à 16 Xeon E5 cores = 76 800 cores)
  - **~38%** peak core performance

Know what's optimal

- White box / black box approaches for performance modelling, e.g.
  - **Roofline model:** S.W.Williams, A.Waterman, D. A. Patterson. *Roofline: An insightful visual performance model for floating-point programs and multicore architectures.* Tech. Rep. UCB/EECS-2008-134, EECS Department, University of California, Berkeley, 2008. http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html
  - **ECM (Execution – Cache – Memory) model:** J. Treibig, G. Hager*: Introducing a Performance Model for Bandwidth-Limited Loop Kernels. In Proc.* of Workshop "Memory issues on Multi- and Manycore Platforms" at PPAM 2009, Wroclaw, Poland, September 13-16, 2009. DOI: 10.1007/978-3-642-14390-8_64
  - **Capacity model:** Dave Kuck, *Computational Capacity-Based Codesign of Computer Systems*, in High-Performance Scientific Computing, Berry, Gallivan, Gallopoulos, Grama, Philippe, Saad, Saied (eds.), Springer 2012. http://dx.doi.org/10.1007/978-1-4471-2437-5_2
- Tools support, e.g.
  - **MAQAO**: performance prediction for loops (vectorization, unrolling factor,…), assuming all operands are in L1
  - **Threadspotter**: cache usage, predictions for different architectures

Know what's optimal

White box approach

- Analyze requirements of algorithm (data volume, arithmetic instructions) and of actual implementation (data access in caches/memory, communication volume, FLOPS,…)
  - Optimize implementation if needed
- Analyze hardware characteristics (caches/memory/interconnect bandwidth and latency, arithmetic units, SIMD, …)
  - Micro-benchmarks
- Derive performance model and compare against benchmarks
  - total execution time **T** per time step composed of time for computation, for communication, for I/O (big issue in some domains such as climate modelling,…), for dealing with boundary conditions,…
- Refine model if needed

Summary

Optimization needs to address Scalability + Performance

1. Load balancing and severe communication issues
2. Single-node execution
   - Huge performance gains possible
3. Fine-tuning for massive parallelism

Taking into account interaction of

- Application and algorithms
- Software stack (compiler, parallel programming models and runtimes, OS,…)
- Underlying hardware (micro-architecture, node topology, system topology,…)

- Performance modelling and performance analysis tools
  - Help understand issues and bottlenecks
  - Guide way to optimum performance

References

- *Scalable and composable shared memory parallelism with tasks for multicore and manycore*, Marc Tchiboukdjian, Thomas Guillet. Teratec Forum 2012, Atelier Exascale.

- *Hybrid Programming with Task-based Models,* Bettina Krammer, Rosa M. Badia, Christian Terboven. BoF SC'12.

- *Quantum Monte Carlo for large chemical systems: Implementing efficient strategies for petascale platforms and beyond*, Anthony Scemama, Michel Caffarel, Emmanuel Oseret, William Jalby. CoRR abs/1209.6630 (2012)

- *Evaluation of the Coarray Fortran Programming Model on the Example of a Lattice Boltzmann Code*. Klaus Sembritzki, Georg Hager, Bettina Krammer, Jan Treibig, Gerhard Wellein. PGAS'12.

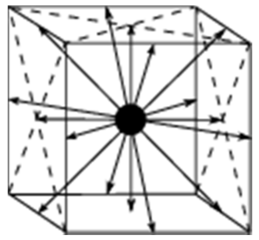
THANKS

QUESTIONS ??

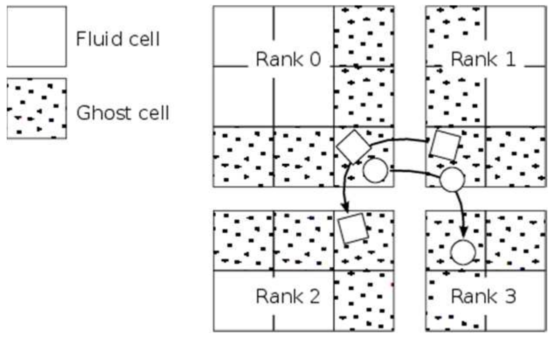
Backup

Example: Lattice Boltzmann solver (LBM)

- CFD method (simulating streaming and collision of particles)
- 19-point stencil, 1D/2D/3D domain decomposition, ghost cells
- Fortran + MPI compared against Coarray Fortran
  - Cray XE6 (AMD 6172 2.2 Ghz)
  - Intel Westmere cluster (X5650, 2.67 Ghz)



D3Q19



- Performance model:

  total execution time **T** per time step is composed of
  - **Time for computation (→ memory stream benchmark)**
  - **Time for communication (→ ringshift benchmark, inter/intra-node, neighbors/network topology)**
  - Time for boundary conditions (→ not taken into account, complexity only N^2)
- Performance metrics: Lattice cell updates per sec [LUPS/s]

  P processors, **N** lattice cells in each dimension: **(P * N^3) / T**

LBM single-node performance model

- LBM memory bound
- Memory bandwidth per node:
  - Stream benchmark mimicking 19-point stencil
  - Saturated with 24 procs per node
    - 24 physical cores/node on XE6
    - 12 physical / 24 virtual cores/node on Intel

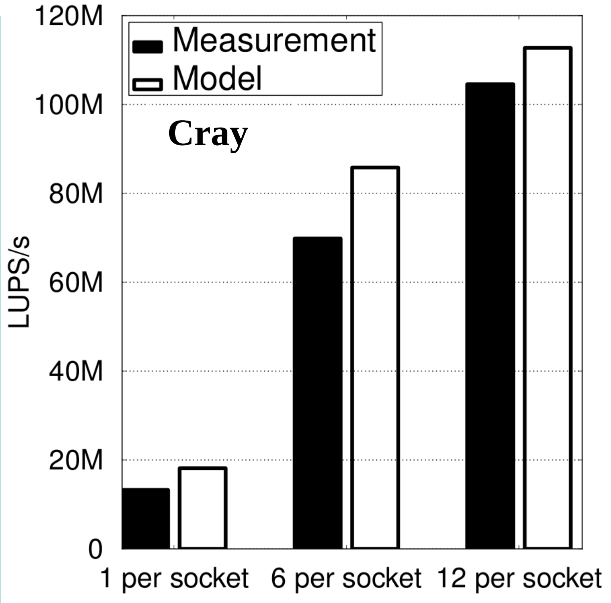
```
double precision :: a(n,19), b(n,19)
for i = 1.. n
    for l = 1..19
        a(i,l) = b(i,l)
    end for
end for
```

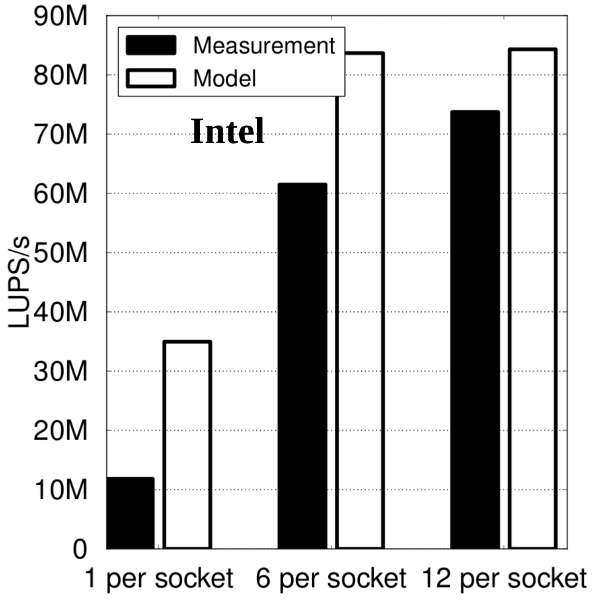
Streams / Proc	Procs / Node	XE6, Mem. BW [GB/s]	Intel, Mem. BW [GB/s]
2	2	18.5	29.6
2 * 19	2	8.4	16.1
2	12	51.9	40.1
2 * 19	12	39.3	38.3
2	24	54.1	41.1
2 * 19	**24**	**51.9**	**38.9**

Domain size: 110 x 110 x 110 per rank (400 MB per rank)

Measurements and model prediction for LBM LUPS/s per node

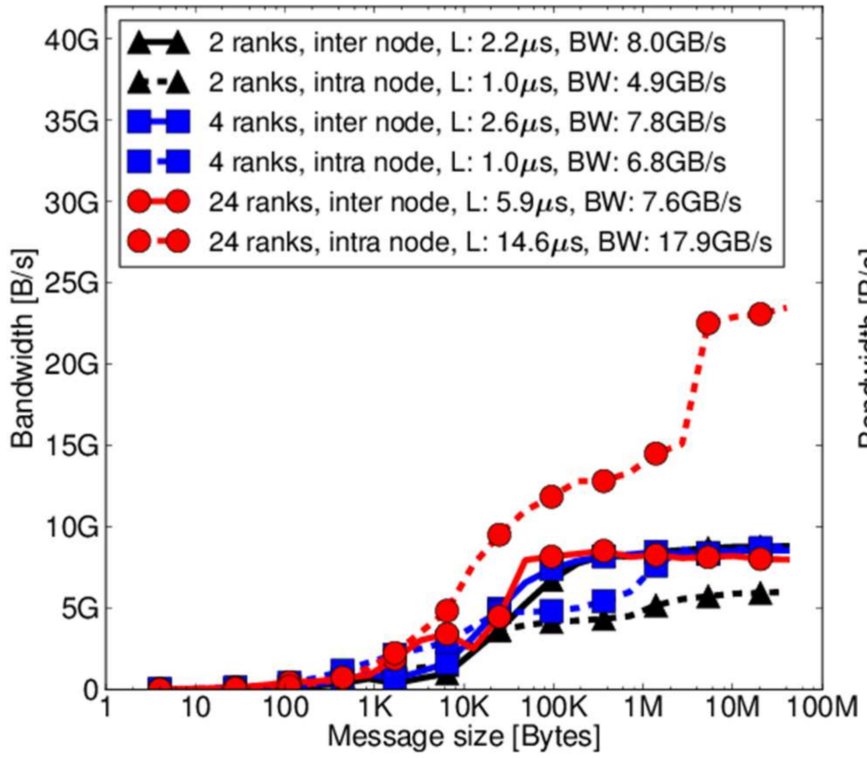
- Time for computation based on stream benchmark
- MPI intra-node communication takes place but communication time is subtracted from total time for LUPS/s calculation



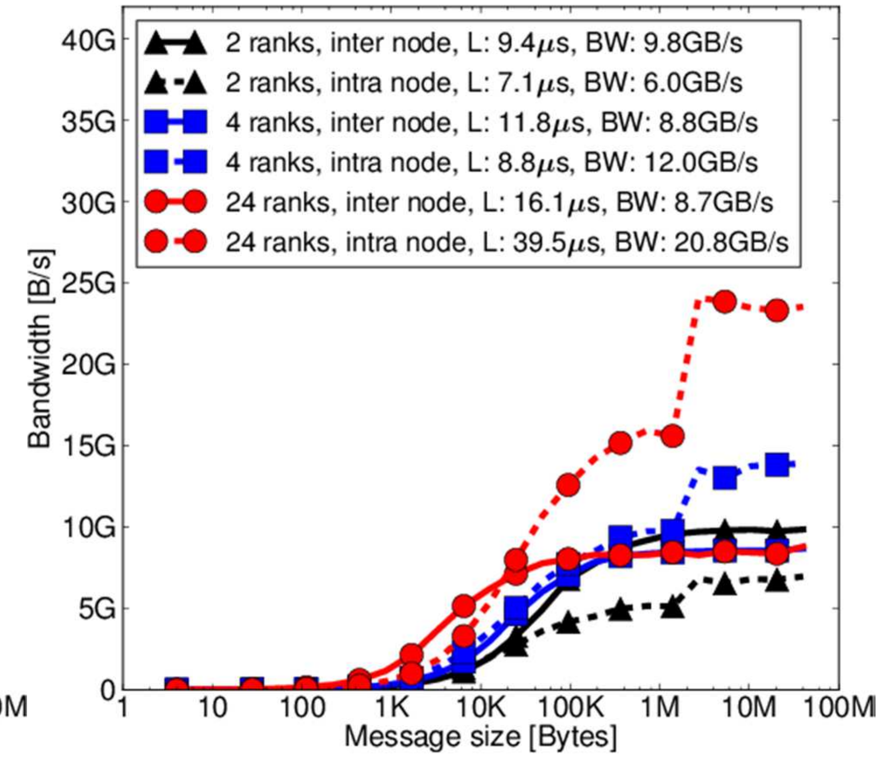


Ringshift benchmark (Cray)

MPI, Cray



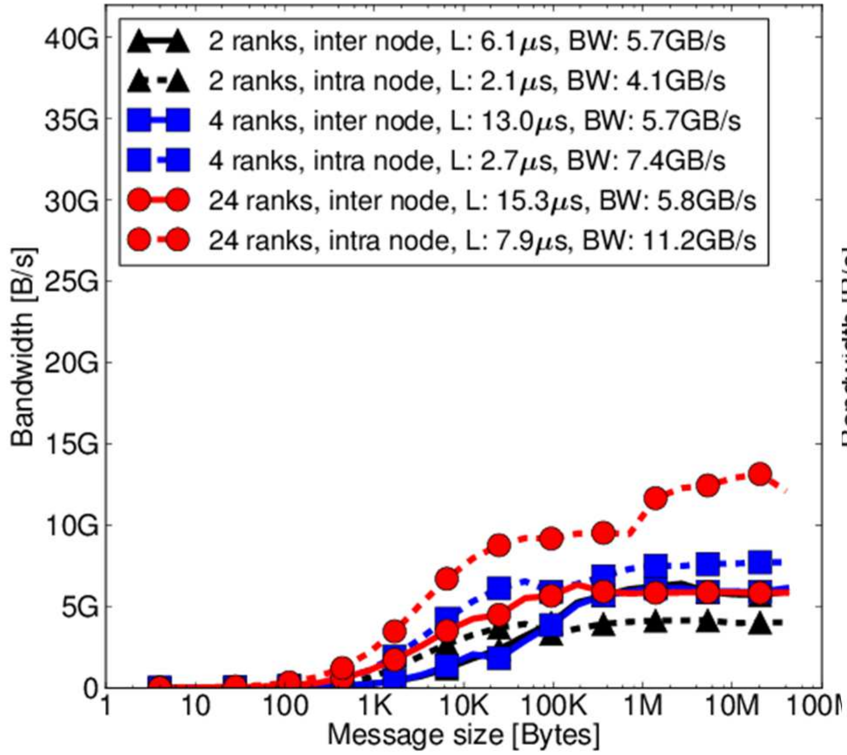
CAF, Cray



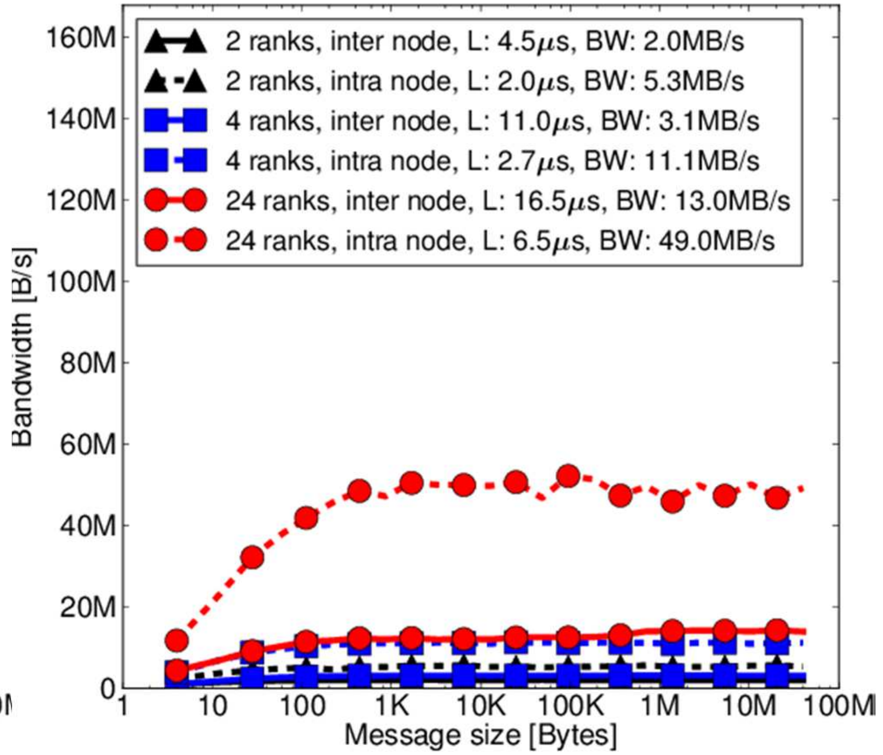
- Similar performance for MPI and CaF
- Higher latency and bandwidth for CaF than for MPI
- Inter- and intra-node bandwidth and latency differs!

Ringshift benchmark (Intel)

MPI, Intel



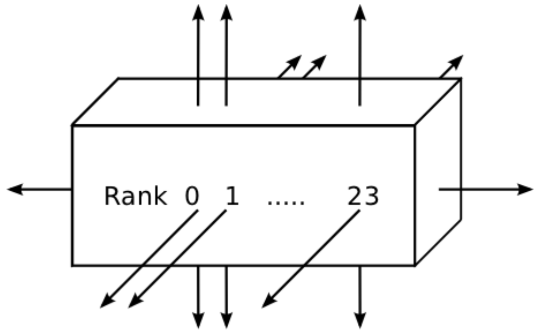
CAF, Intel



- CaF not competitive with MPI on Intel: orders of magnitude difference

LBM: Performance model

P	total number of processes
p	number of processes per node
$L_{e/a}$	inter/intra node communication latency, measured with ringshift benchmark
$B_{e/a}$	inter/intra node communication bandwidth, measured with ringshift benchmark
M	memory bandwidth, measured with 2x19-memory-streams benchmark
N	number of lattice cells in each dimension of the subdomain stored by a process



Inter- and intra-node neighbors (24 procs per node)

Time for one time step

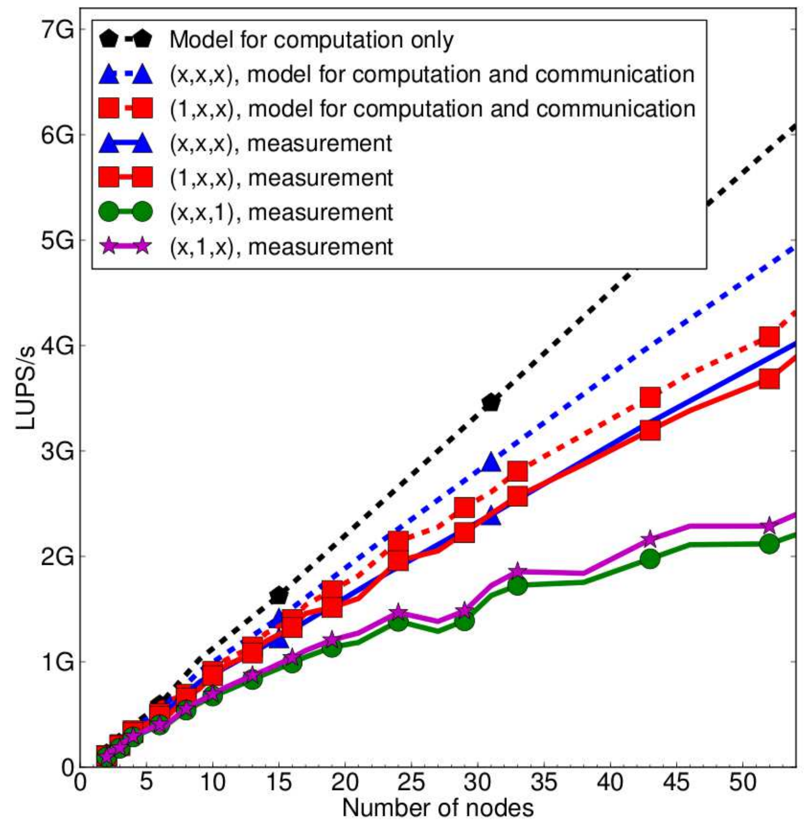
$$\begin{aligned} T_{\text{step}} &= T_{\text{comm}}^{\text{inter}} + T_{\text{comm}}^{\text{intra}} + T_{\text{calc}} \\ &= 3 \cdot L_e + \frac{5 \cdot 8\,\text{Bytes} \cdot N^2}{(B_e/2)/(4 \cdot p + 2)} \\ &\quad + 3 \cdot L_a + \frac{5 \cdot 8\,\text{Bytes} \cdot N^2}{(B_a/2)/(2 \cdot (p-1))} \\ &\quad + \frac{3 \cdot 19 \cdot 8\,\text{Bytes} \cdot N^3}{M} \end{aligned}$$

Performance (lattice updates per sec):

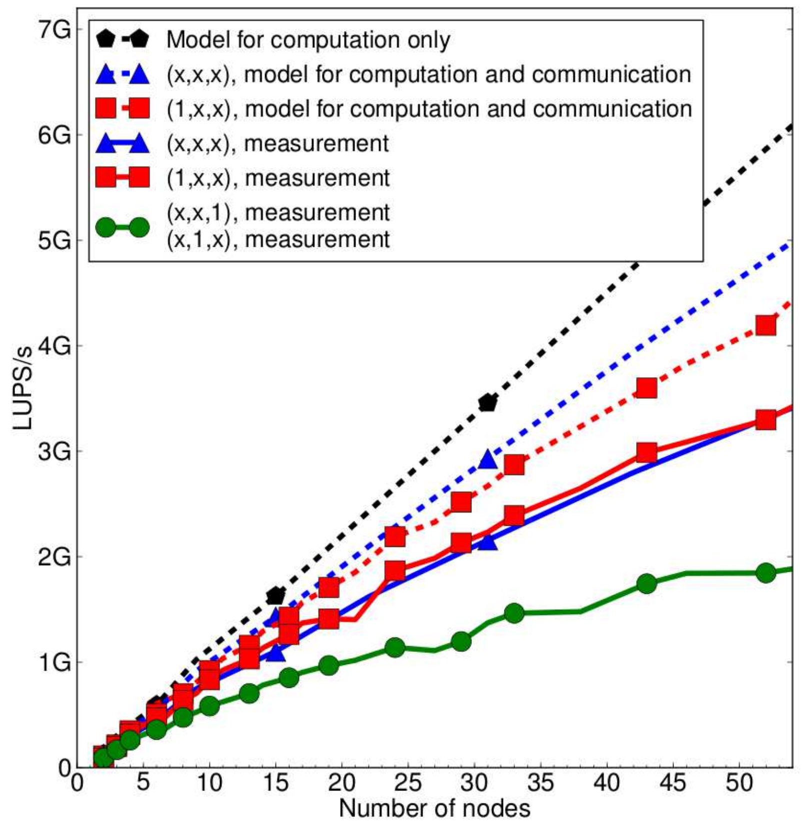
$$S[\text{LUPS/s}] = P \cdot \frac{N^3}{T_{\text{step}}}$$

LBM: Cray

MPI, Cray



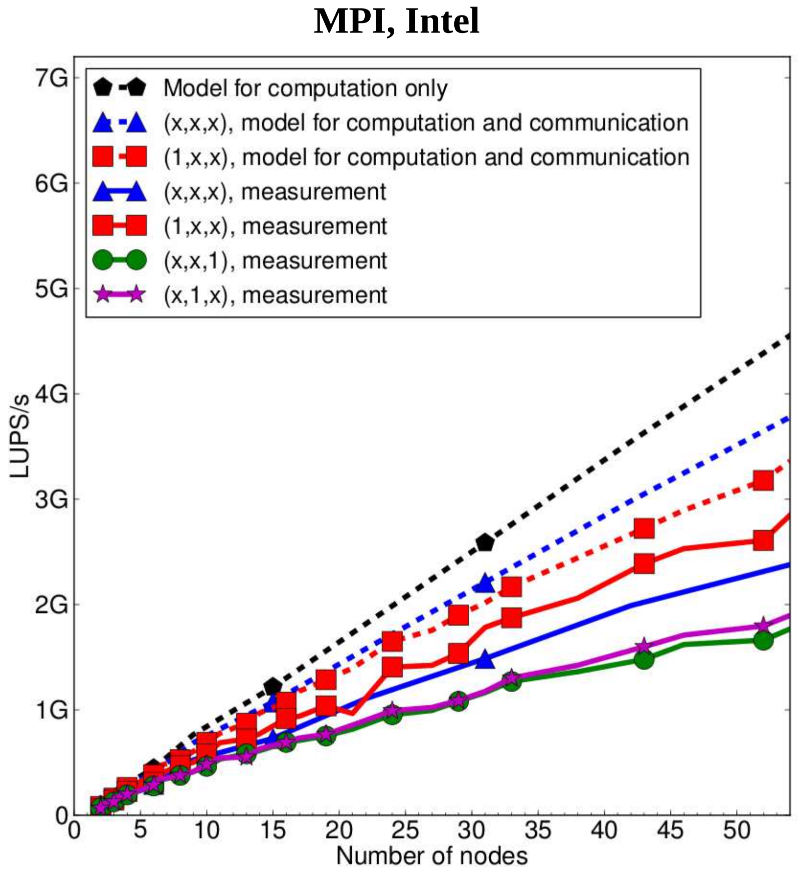
CAF, Cray

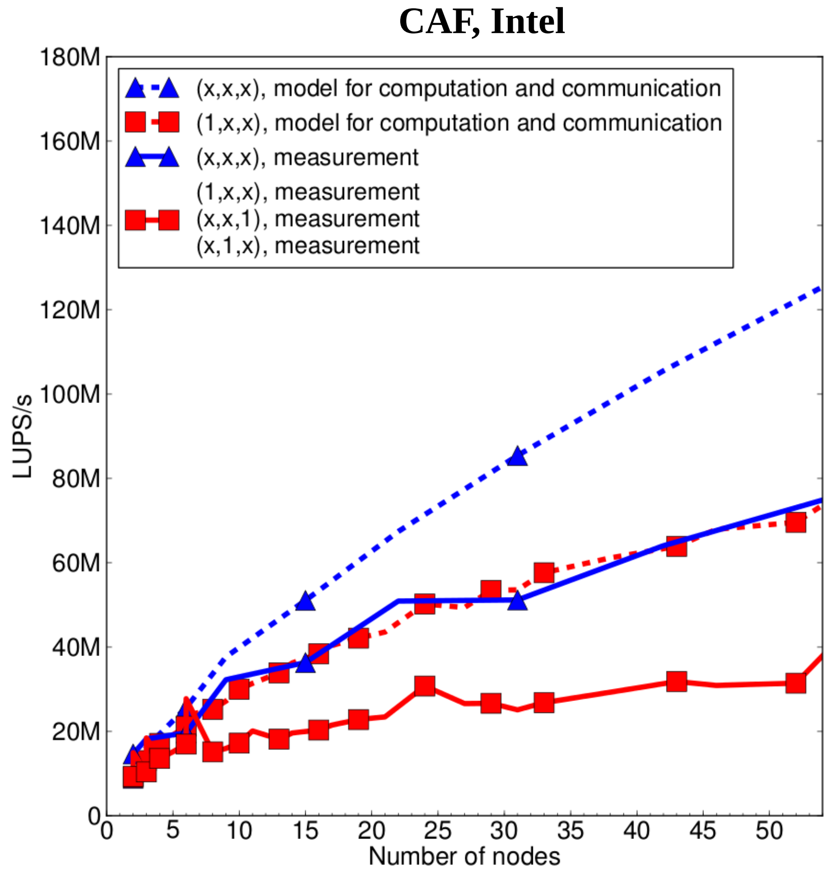


Strong scaling (350^3 lattice cells ~13 GB), 2D/3D domain decomposition

- CaF competitive with MPI
- 3D decomposition and 2D (cutting along the slowest axes) perform equally
- Performance model simple (memory and inter/intra-node ringshift benchmark) but reasonably close

LBM: Intel

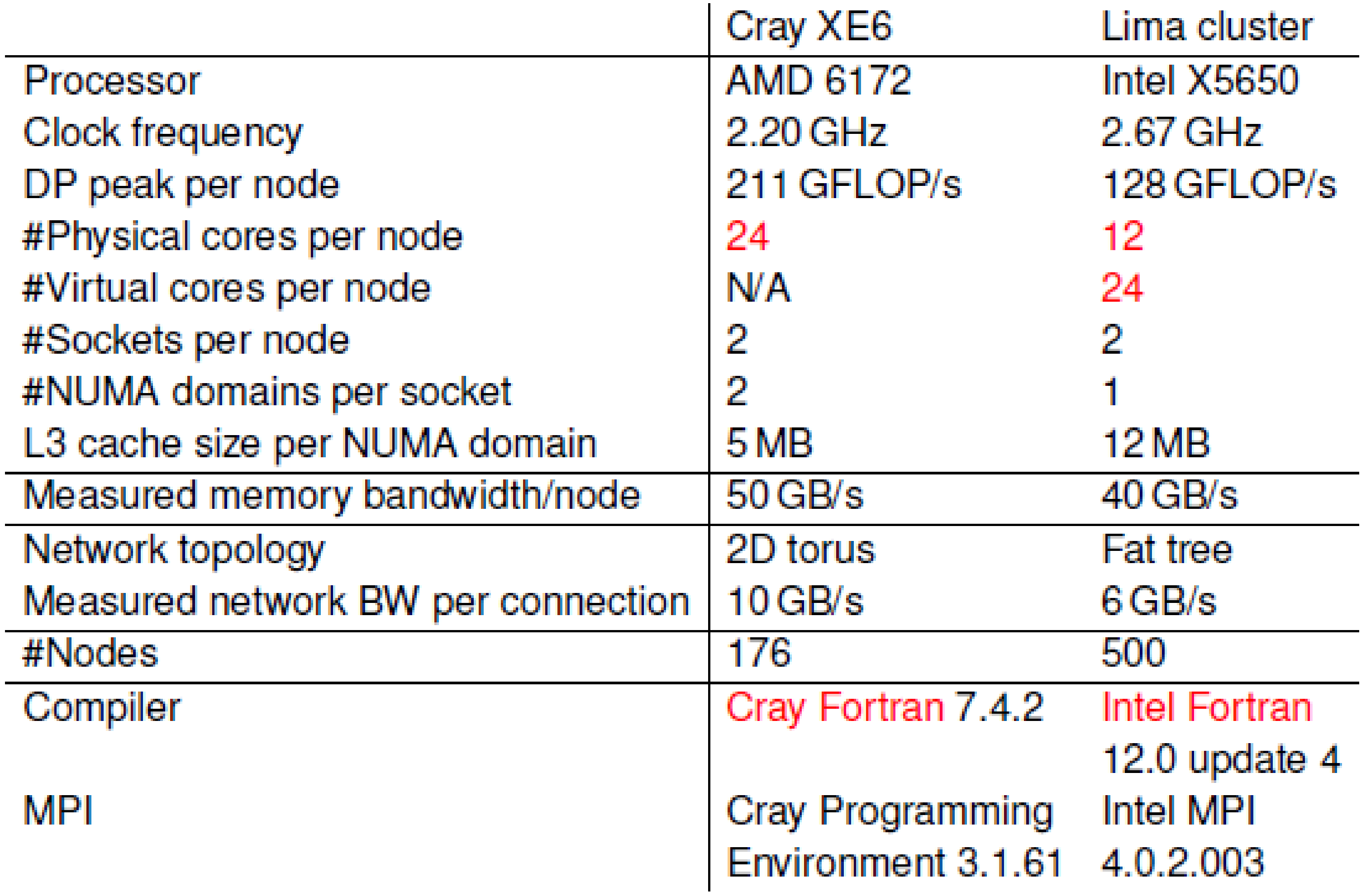




Strong scaling, 2D/3D domain decomposition, on the Intel Westmere Cluster

- CaF disappointing

LBM – Test systems



	Cray XE6	Lima cluster
Processor	AMD 6172	Intel X5650
Clock frequency	2.20 GHz	2.67 GHz
DP peak per node	211 GFLOP/s	128 GFLOP/s
#Physical cores per node	24	12
#Virtual cores per node	N/A	24
#Sockets per node	2	2
#NUMA domains per socket	2	1
L3 cache size per NUMA domain	5 MB	12 MB
Measured memory bandwidth/node	50 GB/s	40 GB/s
Network topology	2D torus	Fat tree
Measured network BW per connection	10 GB/s	6 GB/s
#Nodes	176	500
Compiler	Cray Fortran 7.4.2	Intel Fortran 12.0 update 4
MPI	Cray Programming Environment 3.1.61	Intel MPI 4.0.2.003