Joint Laboratory for Petascale Computation

Charm++ on NUMA Platforms: the impact of SMP Optimizations and a NUMA-aware Load Balancer

Laércio Pilla (UFRGS/Brazil - INRIA)

Christiane Pousa (INRIA)

Daniel Cordeiro (USP/Brazil - INRIA)

Jean-François Méhaut (INRIA)

Outline

- **Introduction**

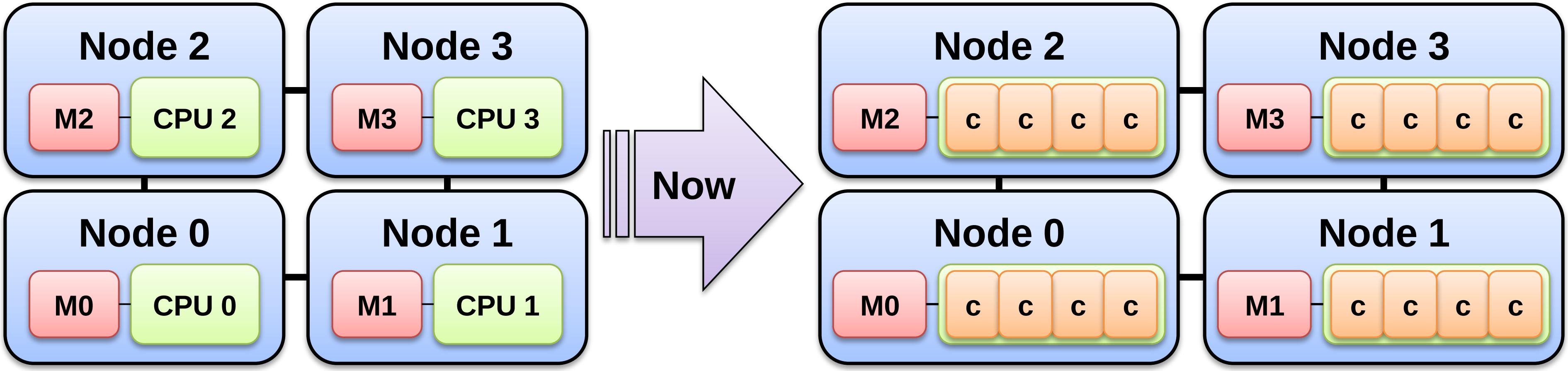
- Performance Evaluation of SMP Optimizations of Charm++ on NUMA Machines

- NUMA-aware Load Balancer on Charm++

- Conclusion and Future Work

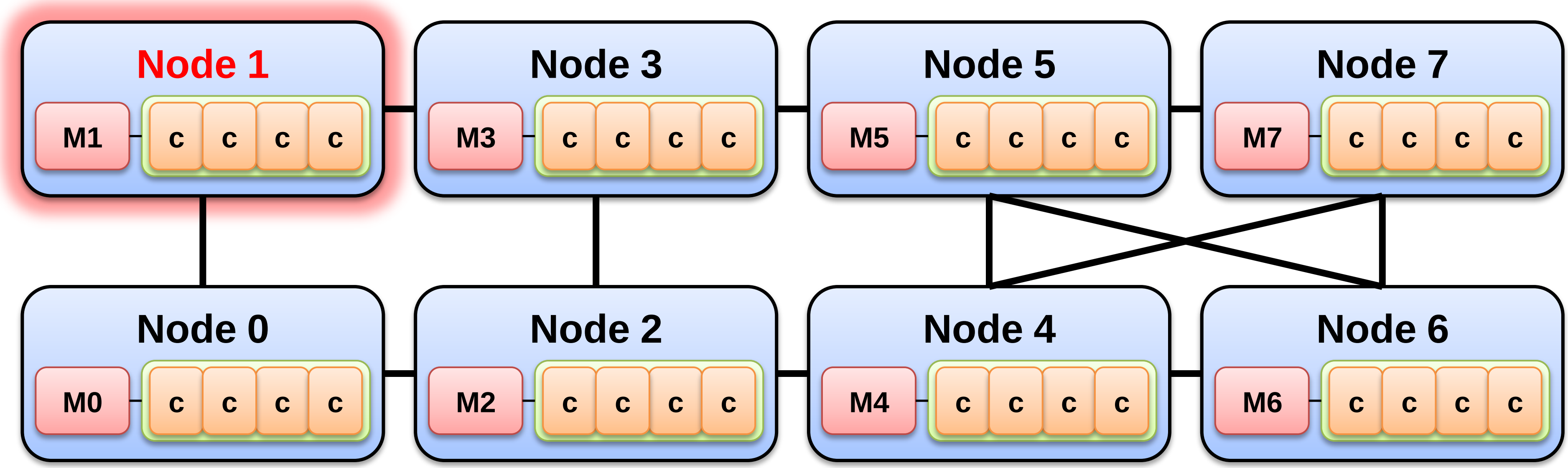
Motivation for NUMA platforms

- The number of cores per processor is increasing
  - **Hierarchical shared memory** multiprocessors
  - ccNUMA is coming back (**NUMA factor**)



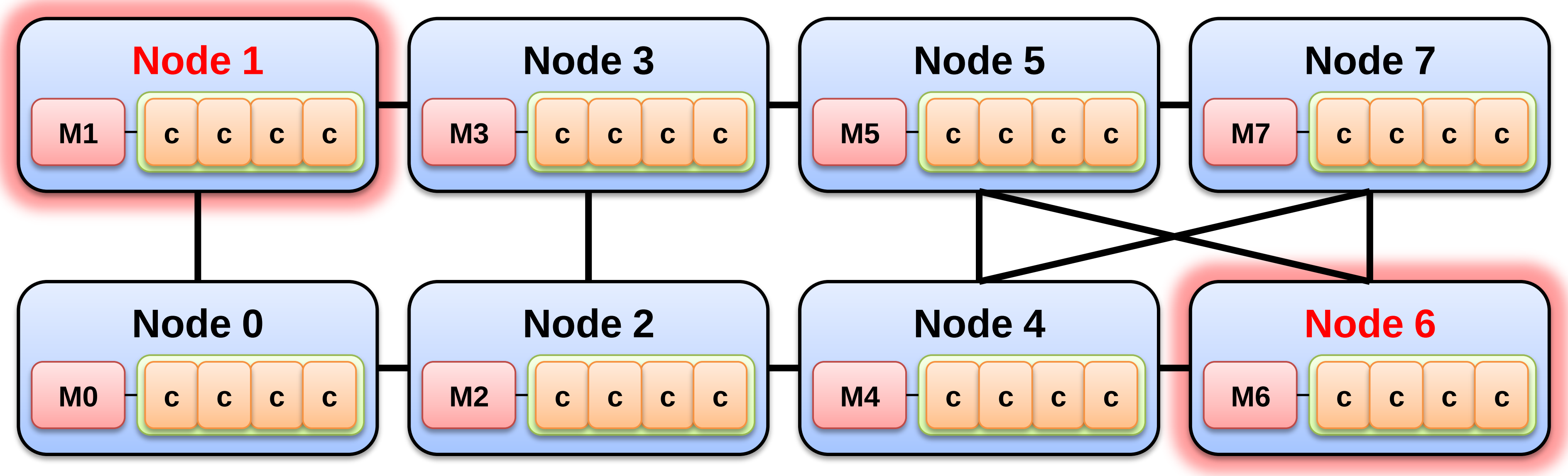
NUMA problems

- Remote access
  - Optimization of **latency**



NUMA problems

- Remote access
  - Optimization of **latency**



Node 1
M1 c c c c

Node 3
M3 c c c c

Node 5
M5 c c c c

Node 7
M7 c c c c

Node 0
M0 c c c c

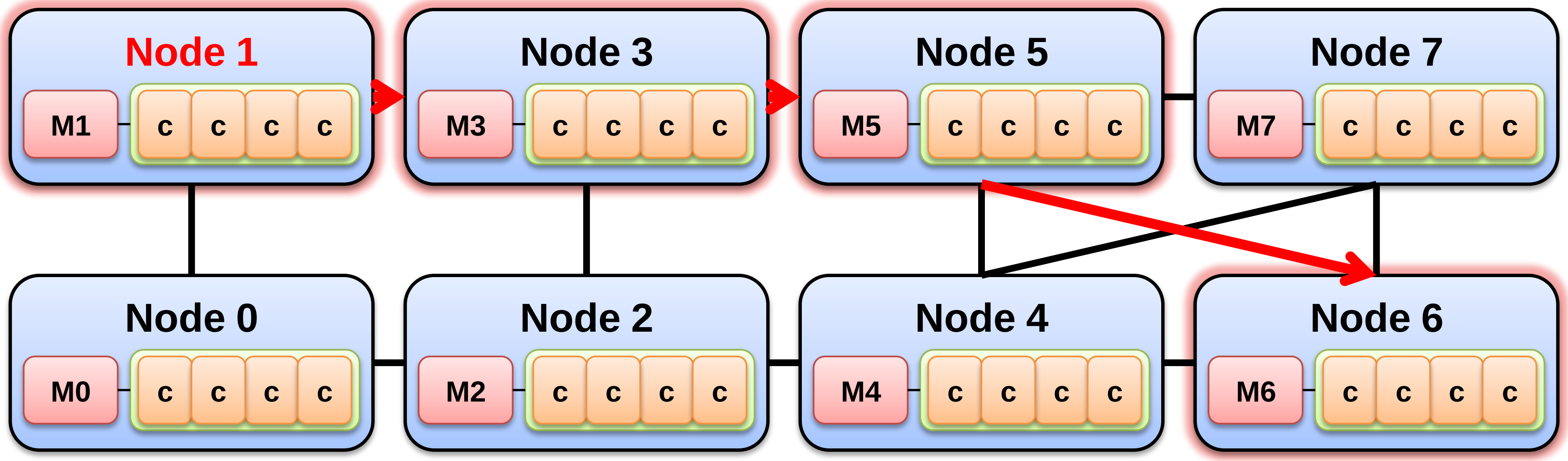
Node 2
M2 c c c c

Node 4
M4 c c c c

Node 6
M6 c c c c

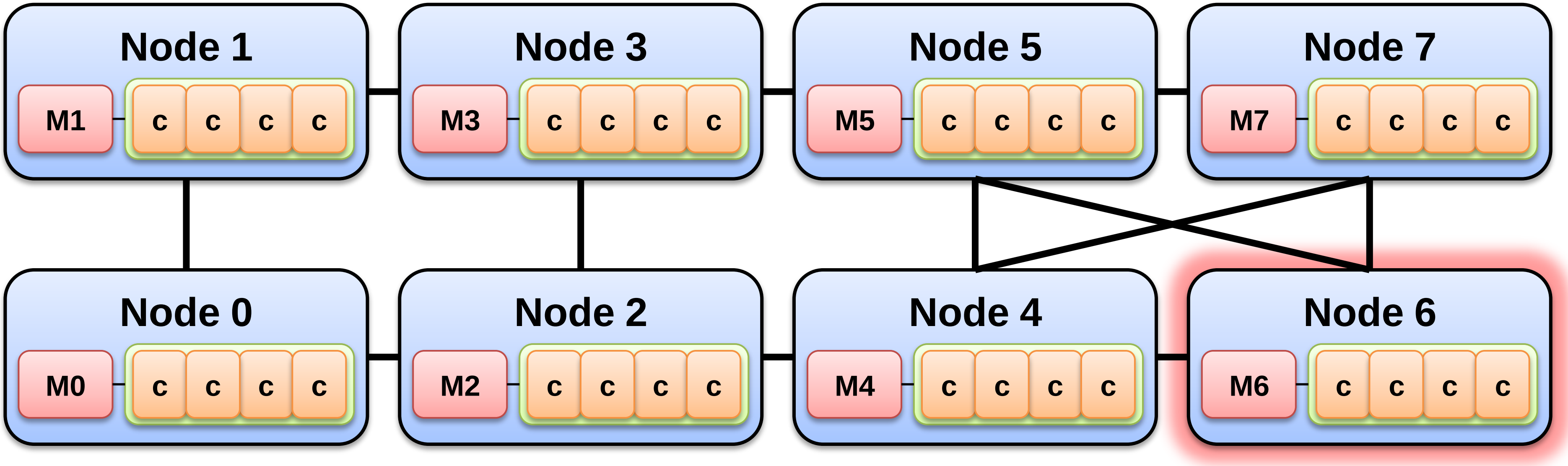
NUMA problems

- Remote access
    - Optimization of **latency**



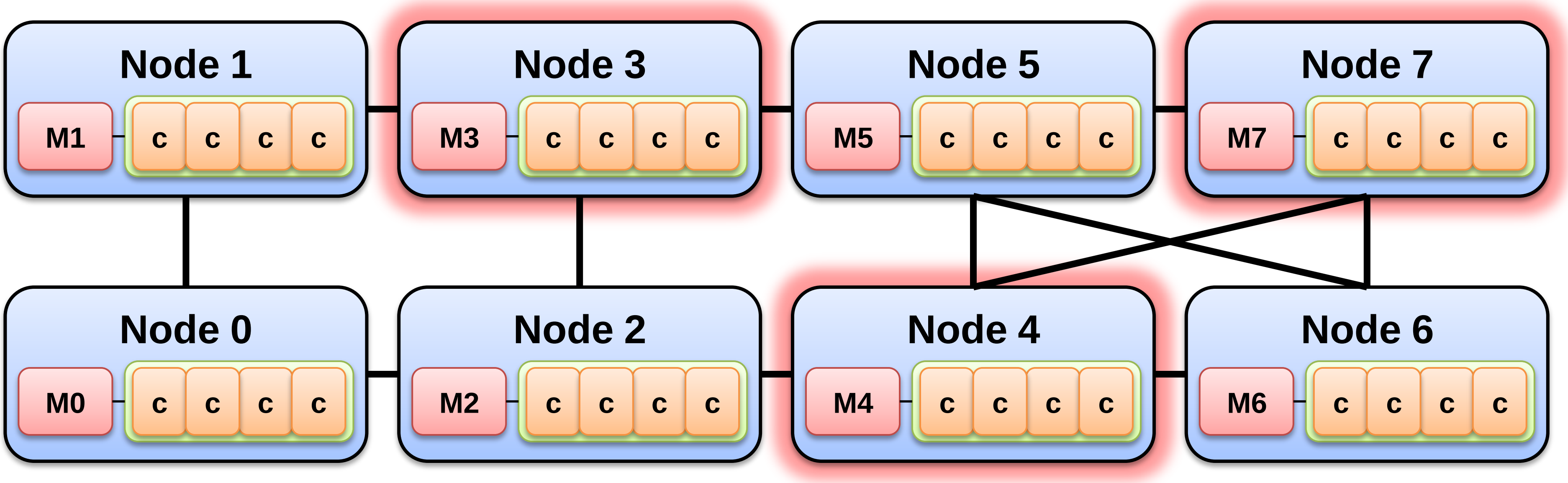
NUMA problems

- Memory contention
  - Optimization of **bandwidth**



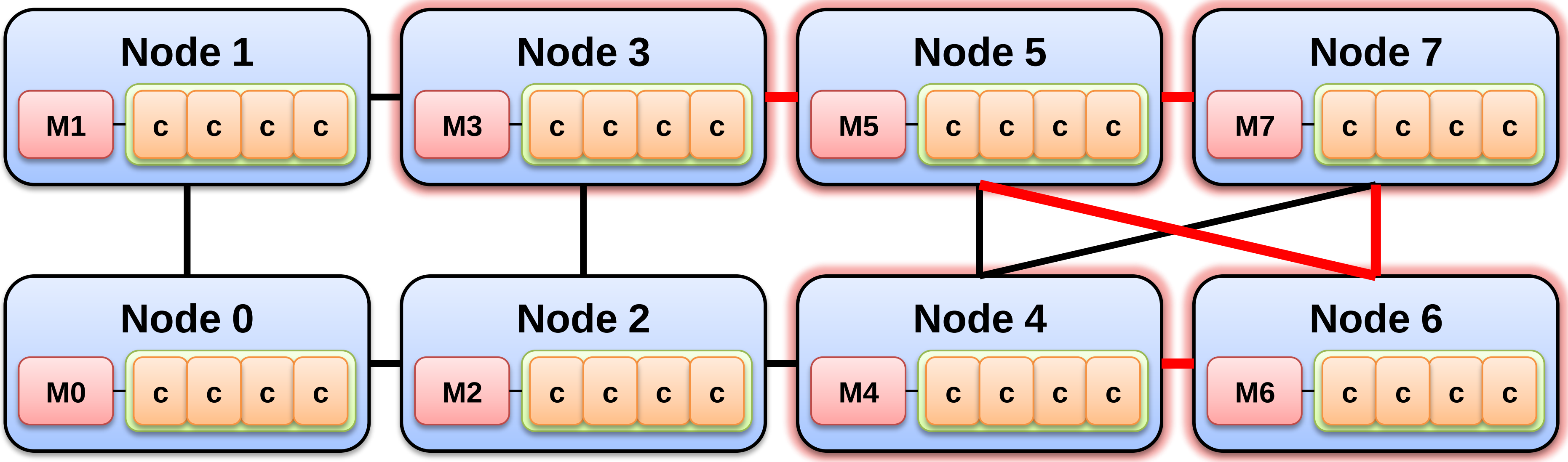
NUMA problems

- Memory contention
  - Optimization of **bandwidth**



NUMA problems

- Memory contention
  - Optimization of **bandwidth**



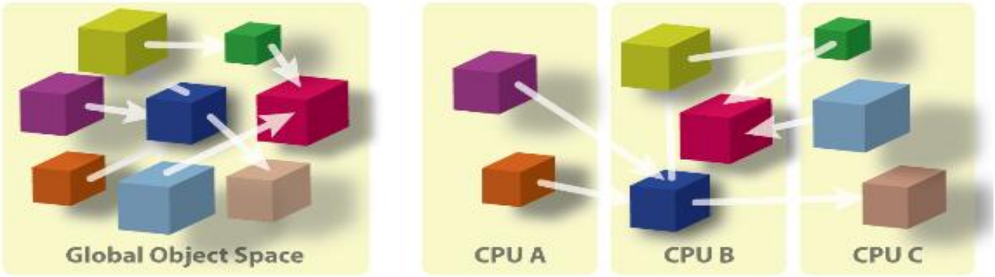
NUMA problems

On NUMA machines, data distribution matters!

Charm++ Parallel Programming System

- Platform independent
  - Both **shared and distributed memory**

- Architecture abstraction
  - Programmer productivity



From charm++ site: http://charm.cs.uiuc.edu/research/charm/

Charm++ Parallel Programming System

- **Communications** originally implemented with **message passing**
  - Even on SMP machines
- Currently, uses **optimizations for SMP** systems
  - Chao Mei et al., “Optimizing a parallel runtime system for multicore clusters: a case study”, in *TG ‘10*

Charm++ & NUMA

- How these **optimizations work on NUMA** machines?

- How can we use **knowledge about the NUMA system** to improve performance on Charm++?

Charm++ & NUMA

- How these optimizations work on NUMA machines?
  - **Our evaluation**
- How can we use knowledge about the NUMA system to improve performance on Charm++?
  - **NUMA-aware load balancer**

Outline

- Introduction
- **Performance Evaluation of SMP Optimizations of Charm++ on NUMA Machines**
- NUMA-aware Load Balancer on Charm++
- Conclusion and Future Work

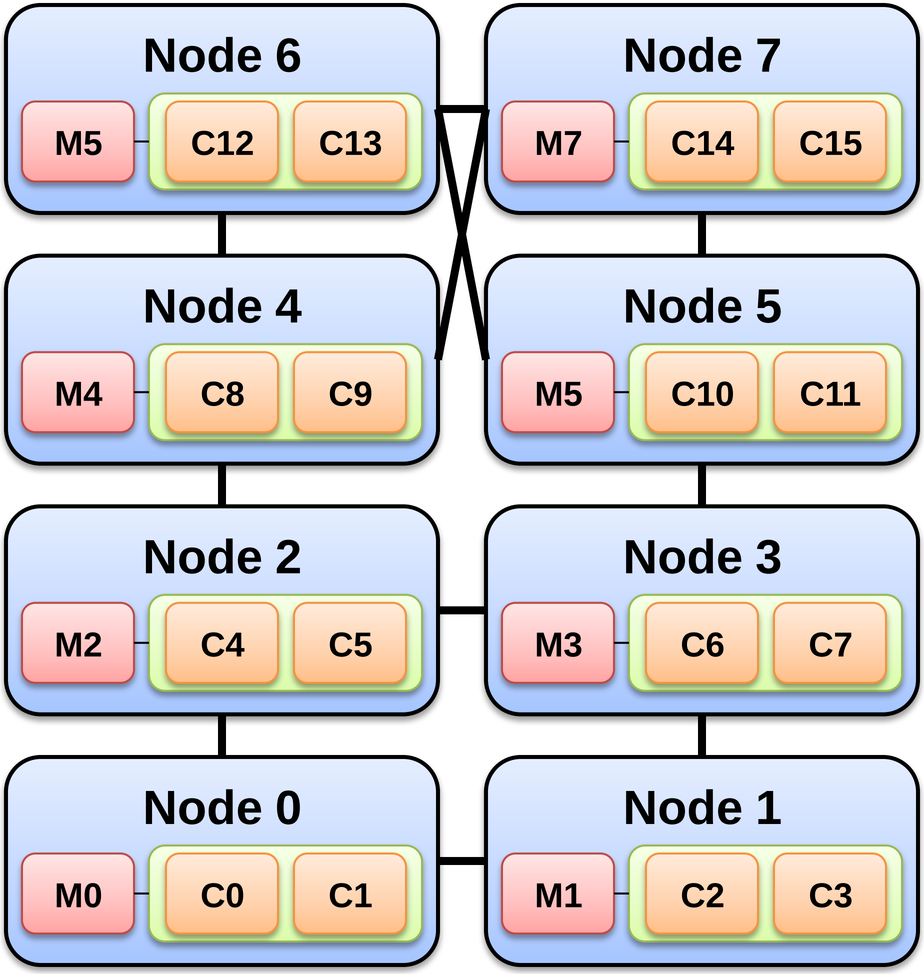
Evaluation of SMP optimizations

- Different Charm++ versions
  - With optimizations
  - Without optimizations

- Different architecture compilations (flavors)
  - net-linux: distributed memory
  - multicore: shared memory

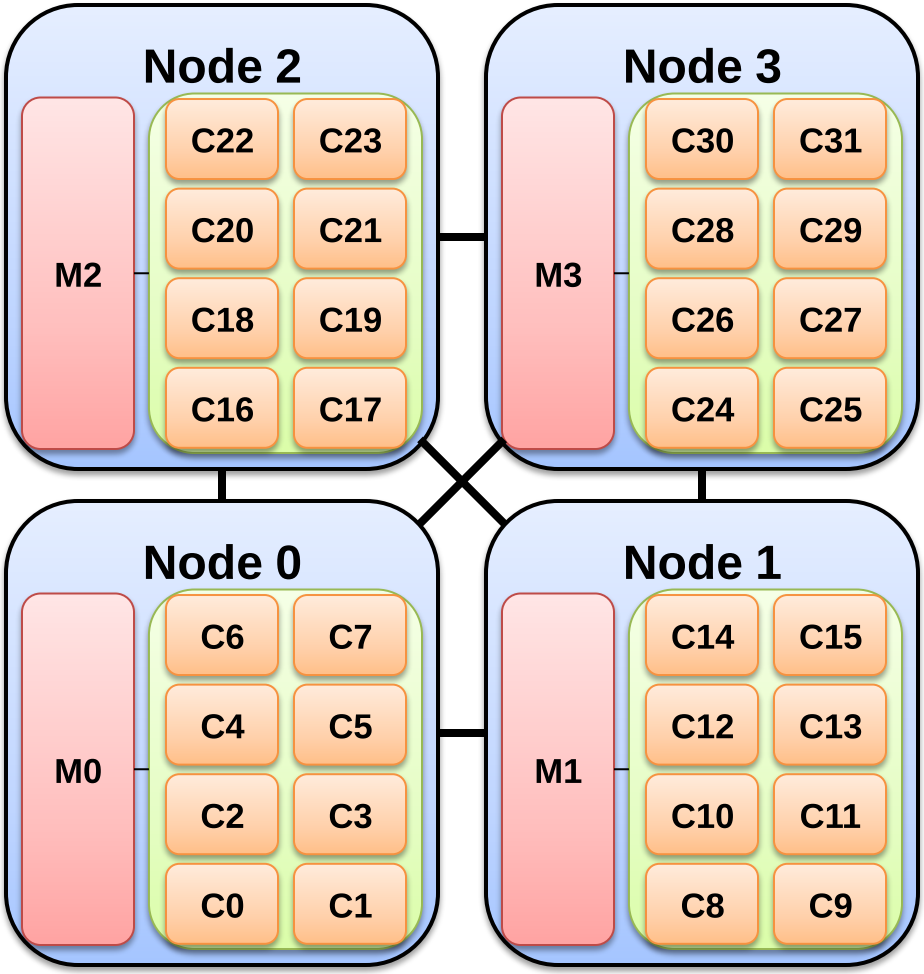
NUMA machines

- AMD Opteron
- **8 nodes x 2 cores**
  - @ 2.2GHz
- 2 MB L2 cache
- 32 GB main memory
- Low latency for local memory access
- Crossbar
- **NUMA factor: 1.2 – 1.5**
- Linux 2.6.32.6



NUMA machines

- Intel Xeon X7560
- **4 nodes x 8 cores**
    - @ 2.27 GHz
- 24 MB shared L3 cache
- 64 GB main memory
- QuickPath
- **NUMA factor: 2 - 2.6**
- Linux 2.6.32



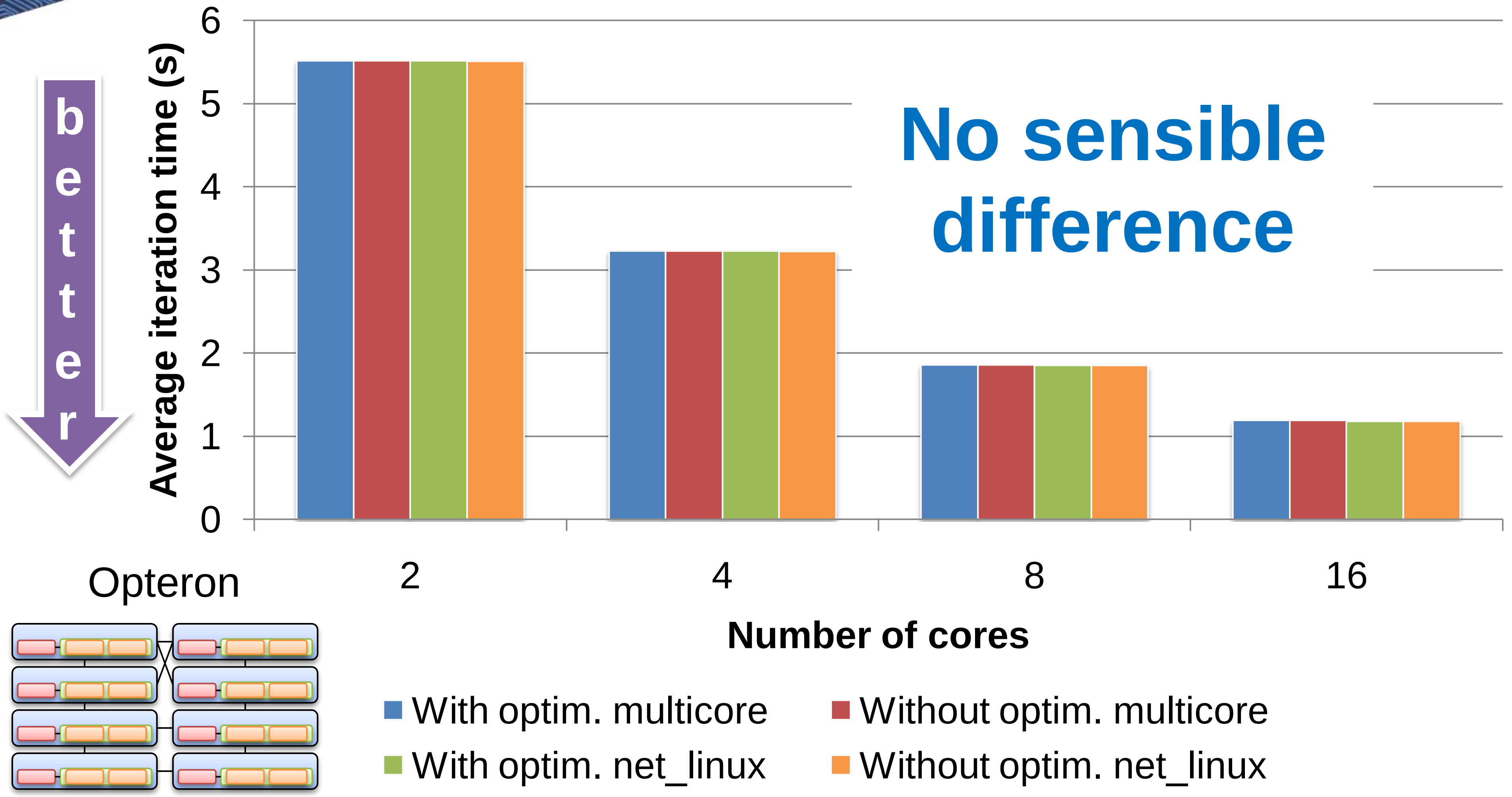
Experimental setup

- Exclusive access to the machines
- Minimum of 10 executions
- Low standard deviation (< 5%)
- Different numbers of cores

Benchmark: Jacobi2D

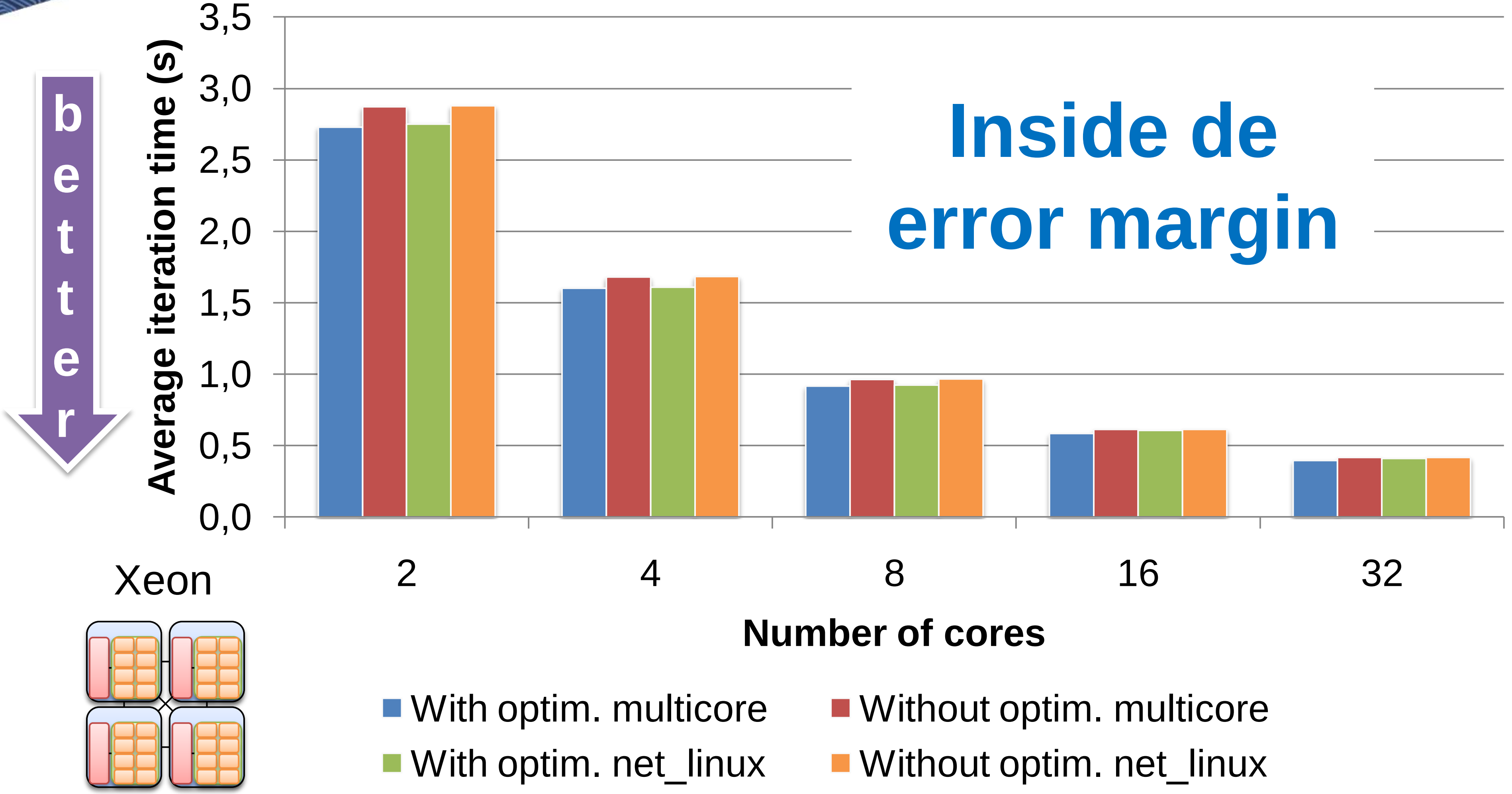
- Iterative benchmark
- Computations over 2D matrix
  - Communications with 4 neighbors
- Stencil (CPU bound)
  - Imbalanced

Jacobi2D on Opteron Machine



No sensible difference

Jacobi2D on Xeon Machine

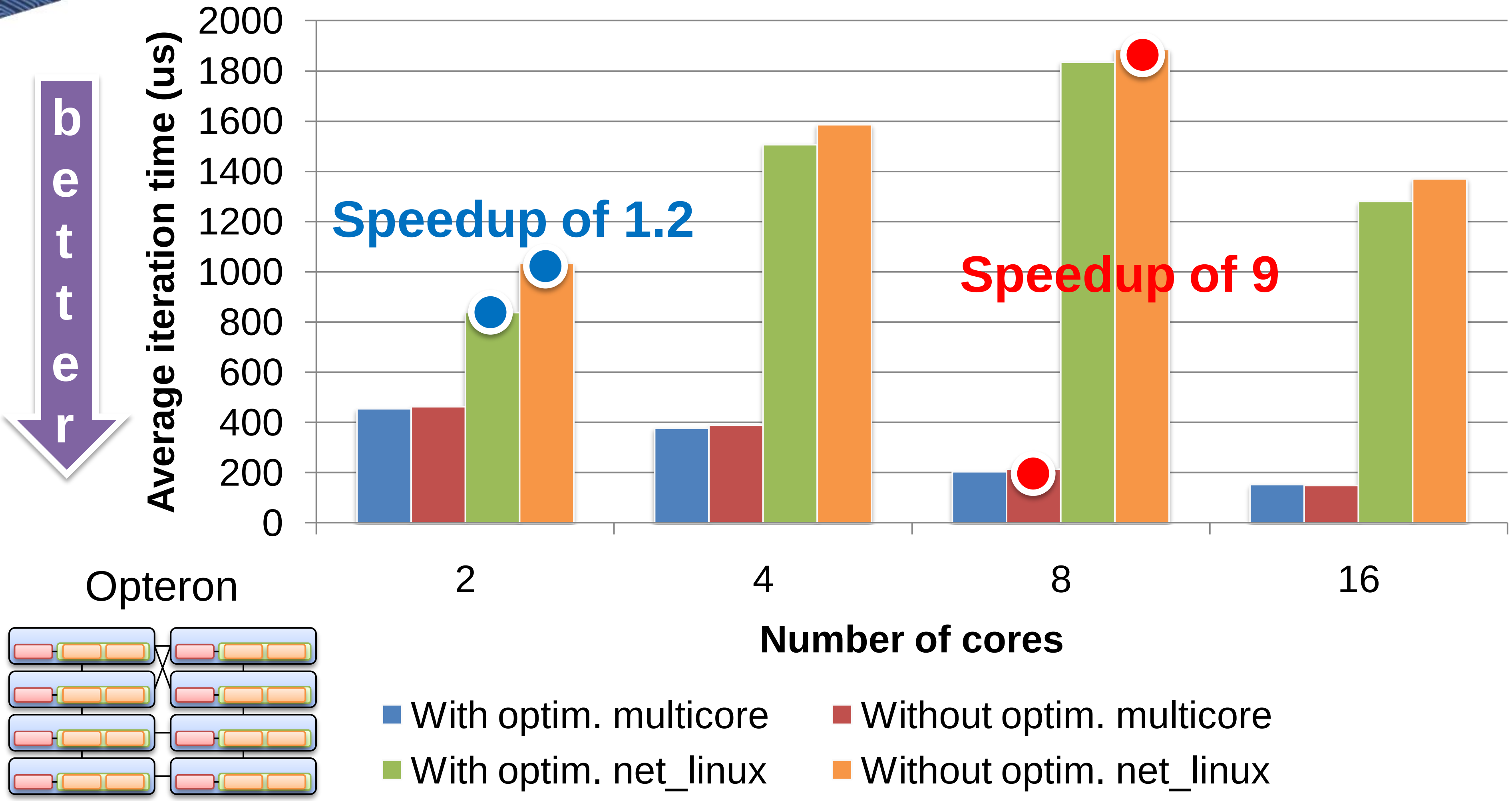


Inside de error margin

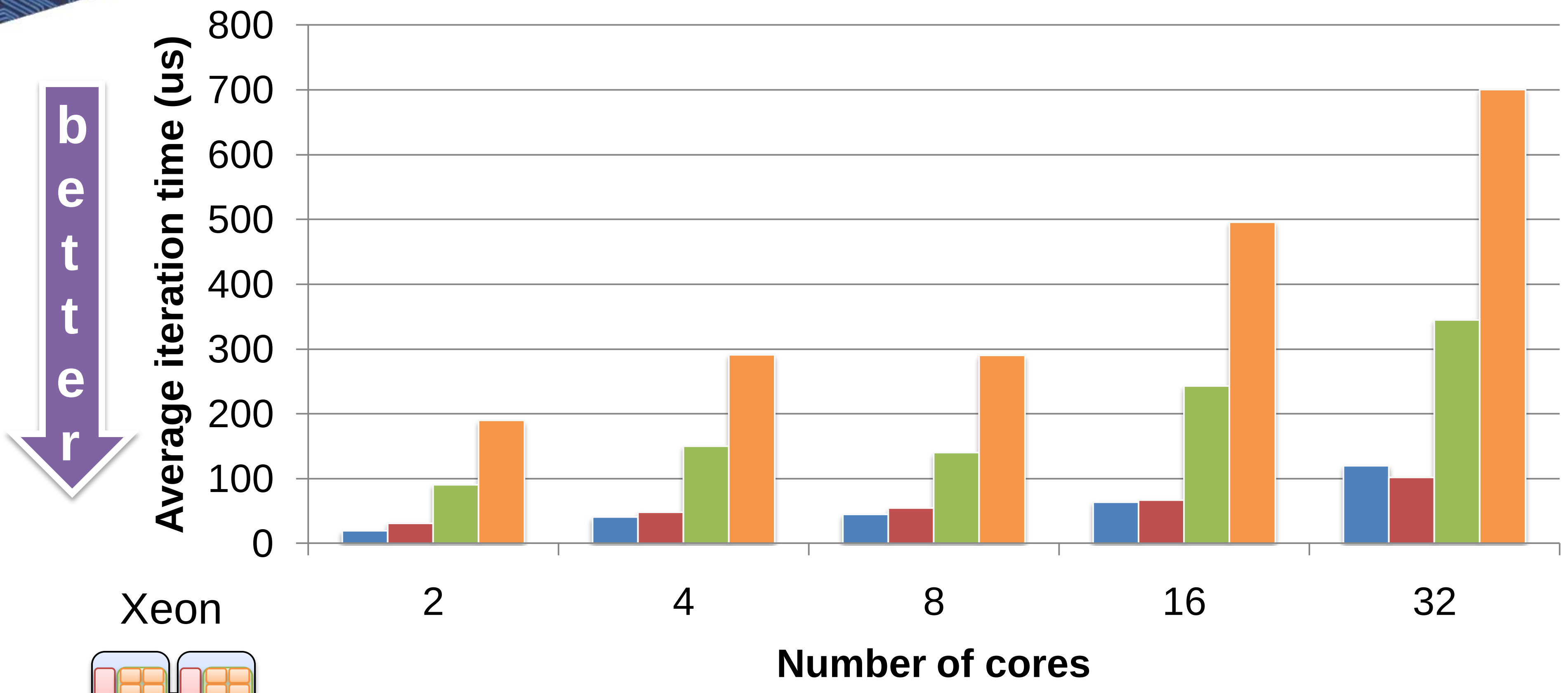
Benchmark: kNeighbor

- Synthetic benchmark
  - Completely communication bound
- Each chare communicates with *k* neighbors
  - *k* = 3
  - Message size = 1024 B

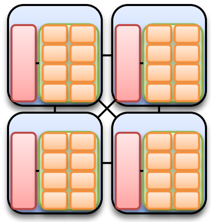
kNeighbor on Opteron Machine



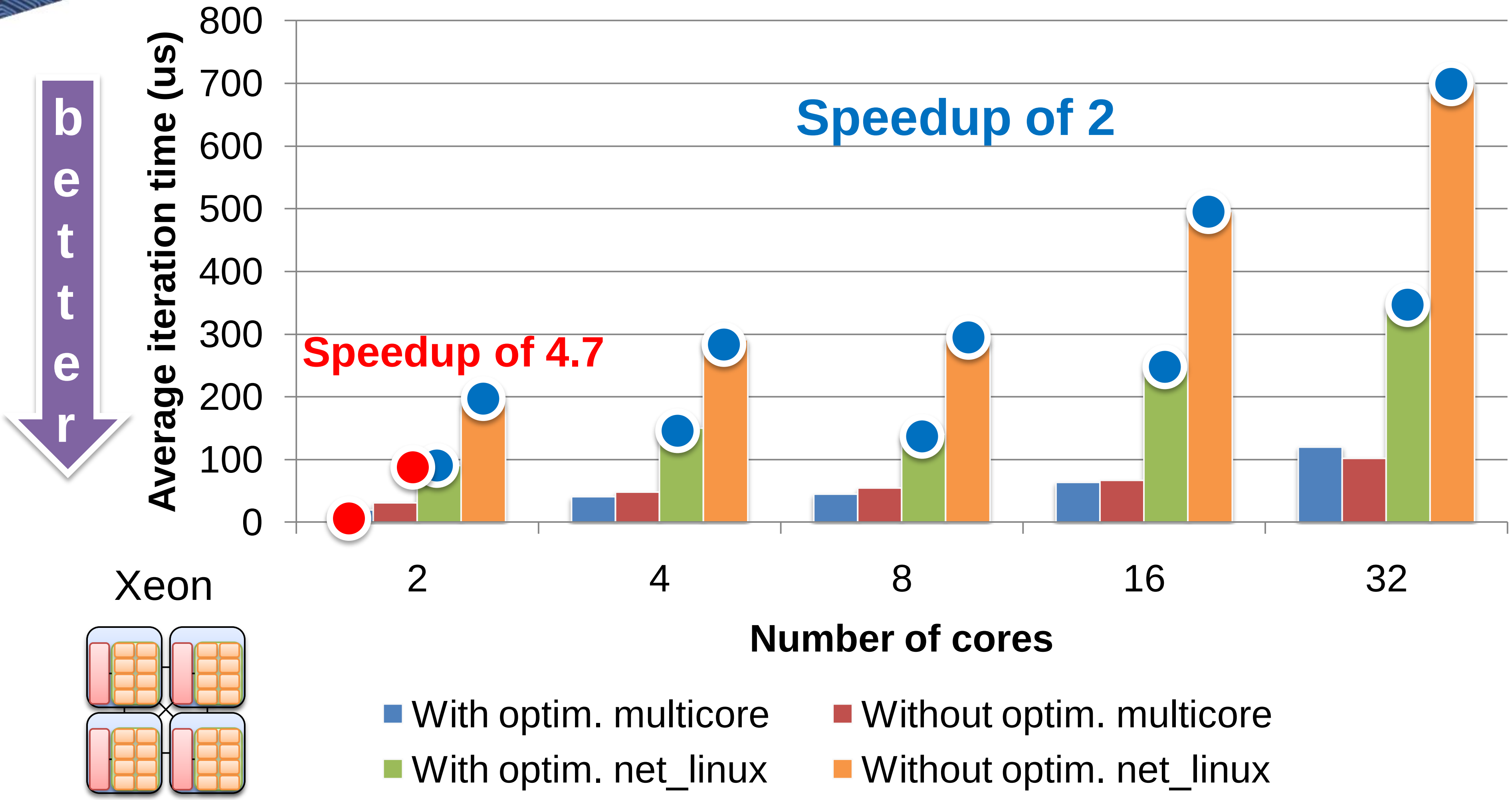
kNeighbor on Xeon Machine



Xeon



kNeighbor on Xeon Machine



Speedup of 2

Speedup of 4.7

Partial conclusions

- Times can be have a **50% difference between Charm++ versions**
- Times **90% smaller when using multicore** instead of net-linux
- Impact proportional to the amount of communications

Outline

- Introduction
- Performance Evaluation of SMP Optimizations of Charm++ on NUMA Machines
- **NUMA-aware Load Balancer on Charm++**
- Conclusion and Future Work

NUMA-aware Load Balancer

- Use **knowledge about the system**
  - **NUMA-factor** among nodes
    - Collected through libarchtopo
  - Communication history
  - No knowledge about the chare’s memory

- Improve performance by **reducing** communication **latency**
- Avoid too many chare migrations

NUMA-aware Load Balancer

```
Calculate processors’ load
Sort chares by decreasing load

While there are migratable chares
   Pick most loaded chare $k$
   Compute $W(k,i)$ for all processors $i$
   Migrate k for the processor with smaller $W(k,i)$
```

NUMA-aware Load Balancer

$$
\begin{aligned}
W(k,i) = \quad & L(i) + \\
& \alpha*(\\
& \quad - M(k,i) \\
& \quad + \Sigma_{j=1..N,\ j!=i}\,(M(k,j)*NF(j,i)) \\
&)
\end{aligned}
$$

NUMA-aware Load Balancer

Load on candidate processor (core)

$$\begin{aligned} W(k,i) = {} & \boxed{L(i)} + \\ & \boxed{\alpha}*(\quad \text{Communication weight (constant)} \\ & \qquad - M(k,i) \\ & \qquad + \Sigma_{j=1..N,\, j!=i}\,(M(k,j)*NF(j,i)) \\ &) \end{aligned}$$

NUMA-aware Load Balancer

$$
\begin{aligned}
W(k,i) = \quad & L(i) + \\
& \alpha*(\\
& \quad - \boxed{M(k,i)} \\
& \quad + \Sigma_{j=1..N,\ j!=i} (M(k,j)*NF(j,i)) \\
&)
\end{aligned}
$$

Intra-core communications (extended for intra-NUMA node)

NUMA-aware Load Balancer

$$
\begin{aligned}
W(k,i) = \ & L(i) + \\
& \alpha*(\\
& \quad - M(k,i) \\
& \quad + \boxed{\Sigma_{j=1..N,\ j!=i}\ (M(k,j)}\boxed{*NF(j,i))} \\
&)
\end{aligned}
$$

NUMA factor

Inter-core communications (extended for inter-NUMA node)

Load Balancer Evaluation

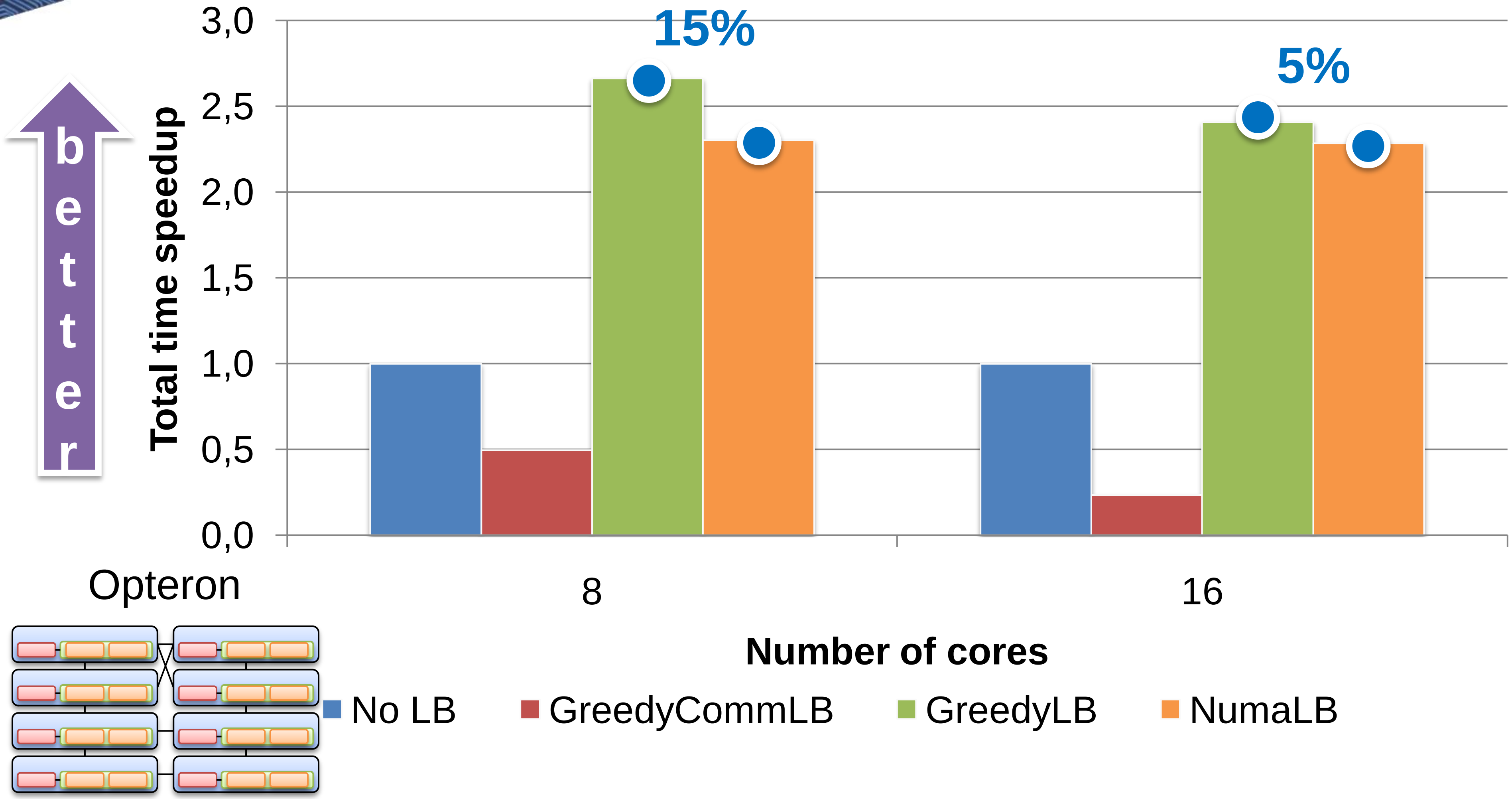
- Benchmarks
  - Imbalance
  - Jacobi2D
  - Poisson3D

- Comparison with different load balancers
  - GreedyLB
  - GreedyCommLB

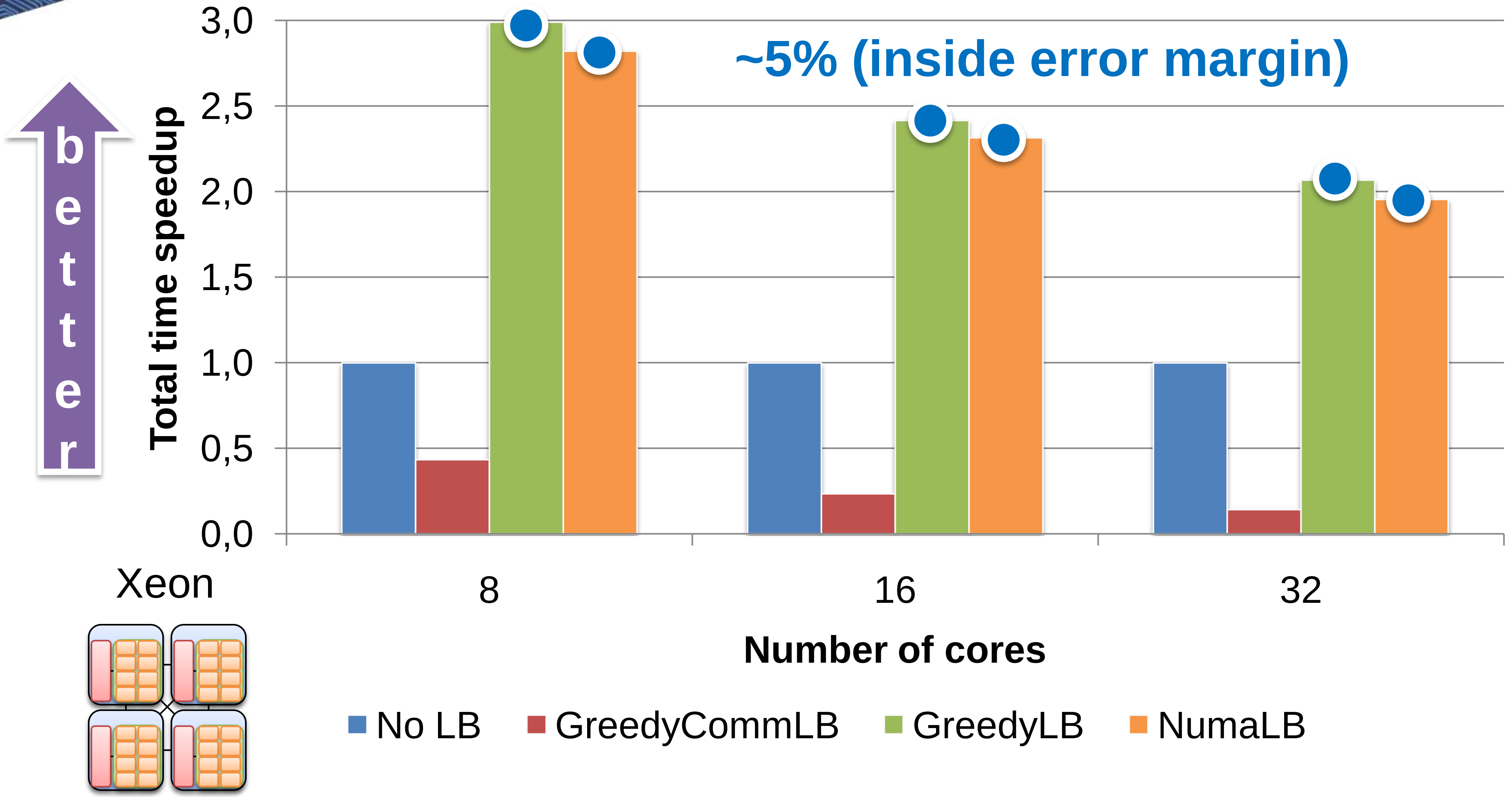
Benchmark: Imbalance

- By Isaac Dooley
  - Based on Fractography3D
- Iterative benchmark
  - Imbalance increases with computations
- Computations over 2D array of chares
  - Communications with 4 neighbors

Imbalance on Opteron Machine



Imbalance on Xeon Machine

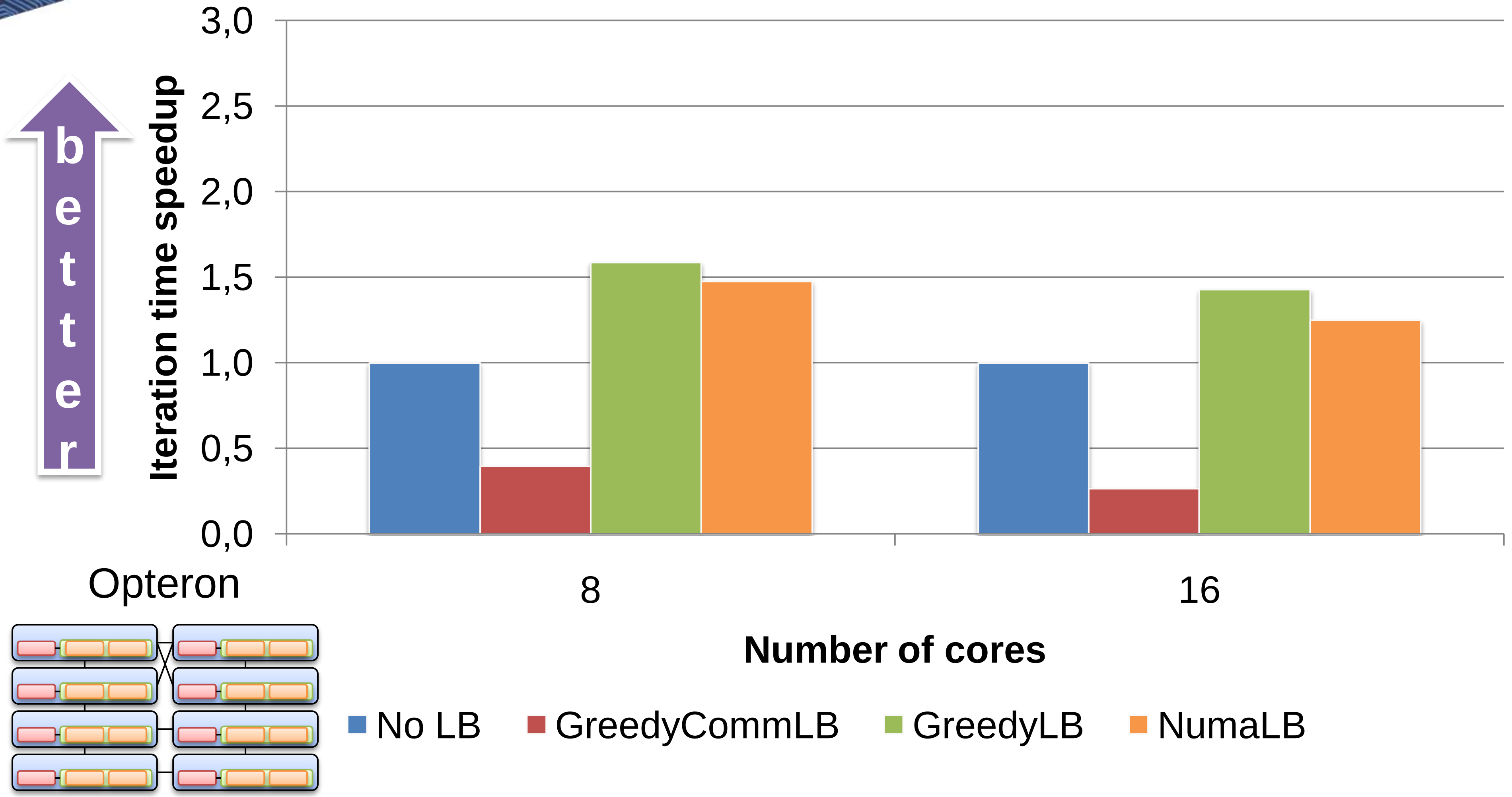


~5% (inside error margin)

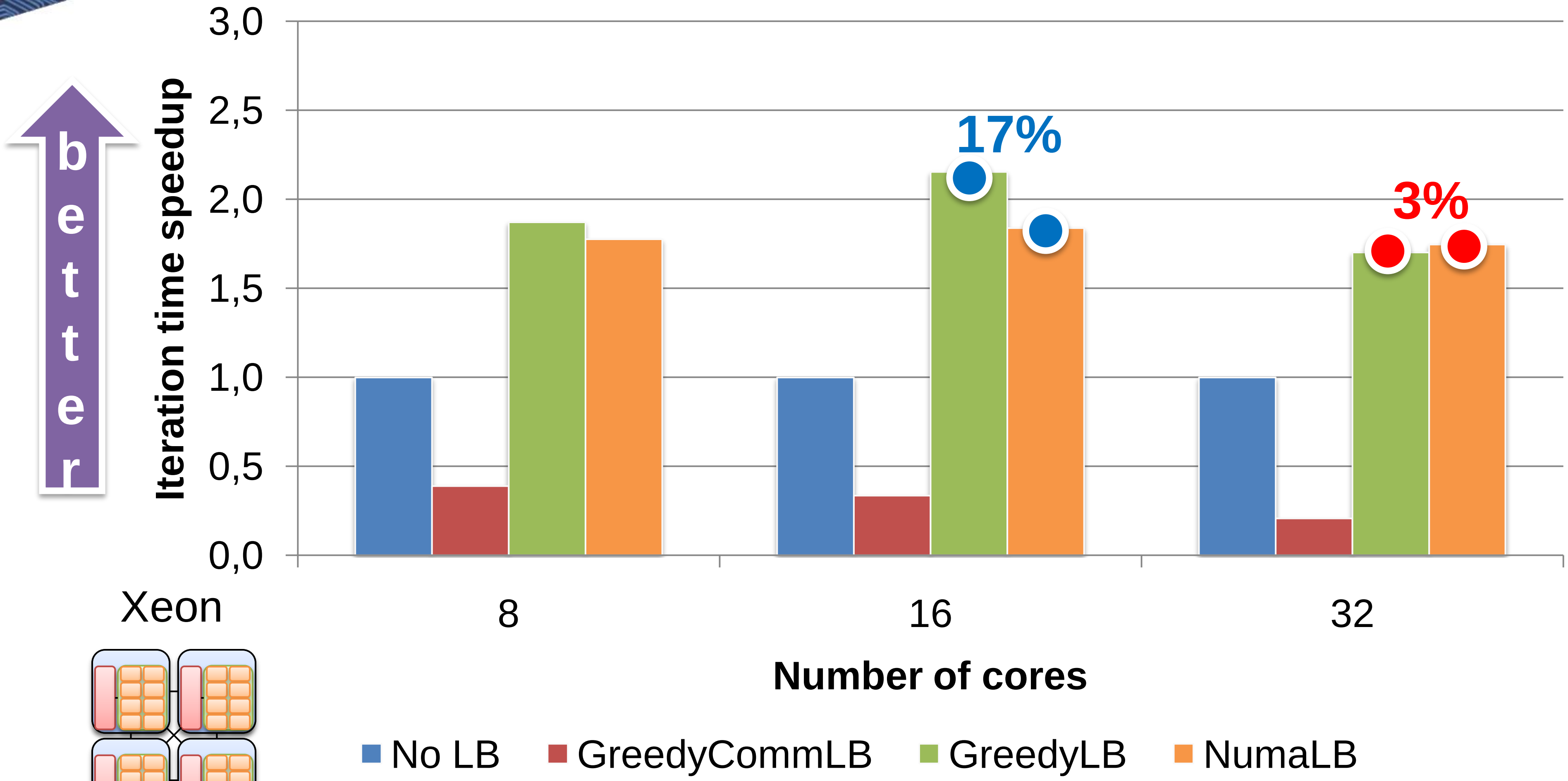
Benchmark: Jacobi2D

- Iterative benchmark
- Computations over 2D matrix
  - Communications with 4 neighbors
- Stencil (CPU bound)
  - Imbalaced

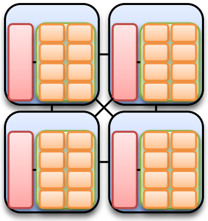
Jacobi2D on Opteron Machine



Jacobi2D on Xeon Machine



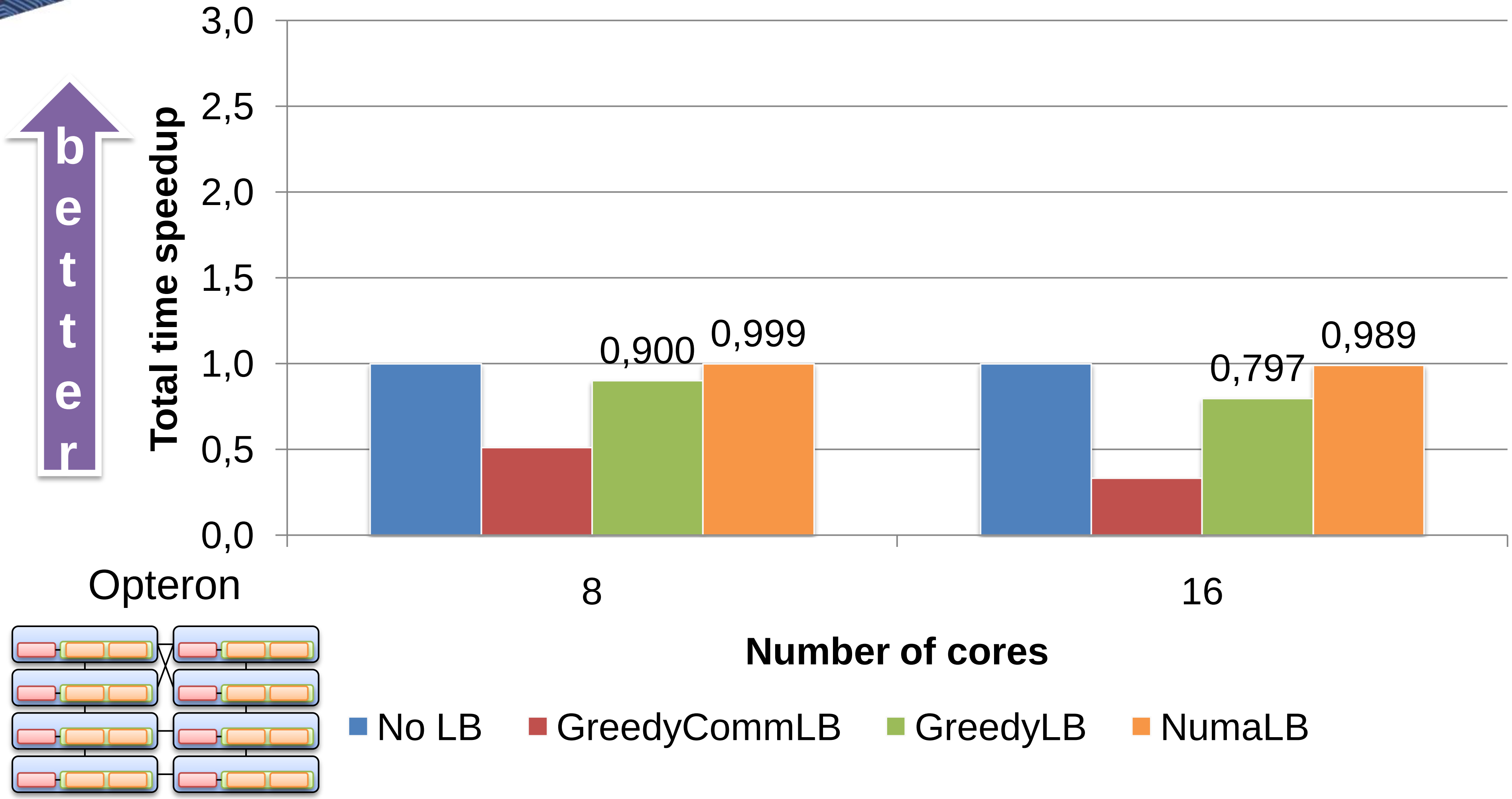
Xeon



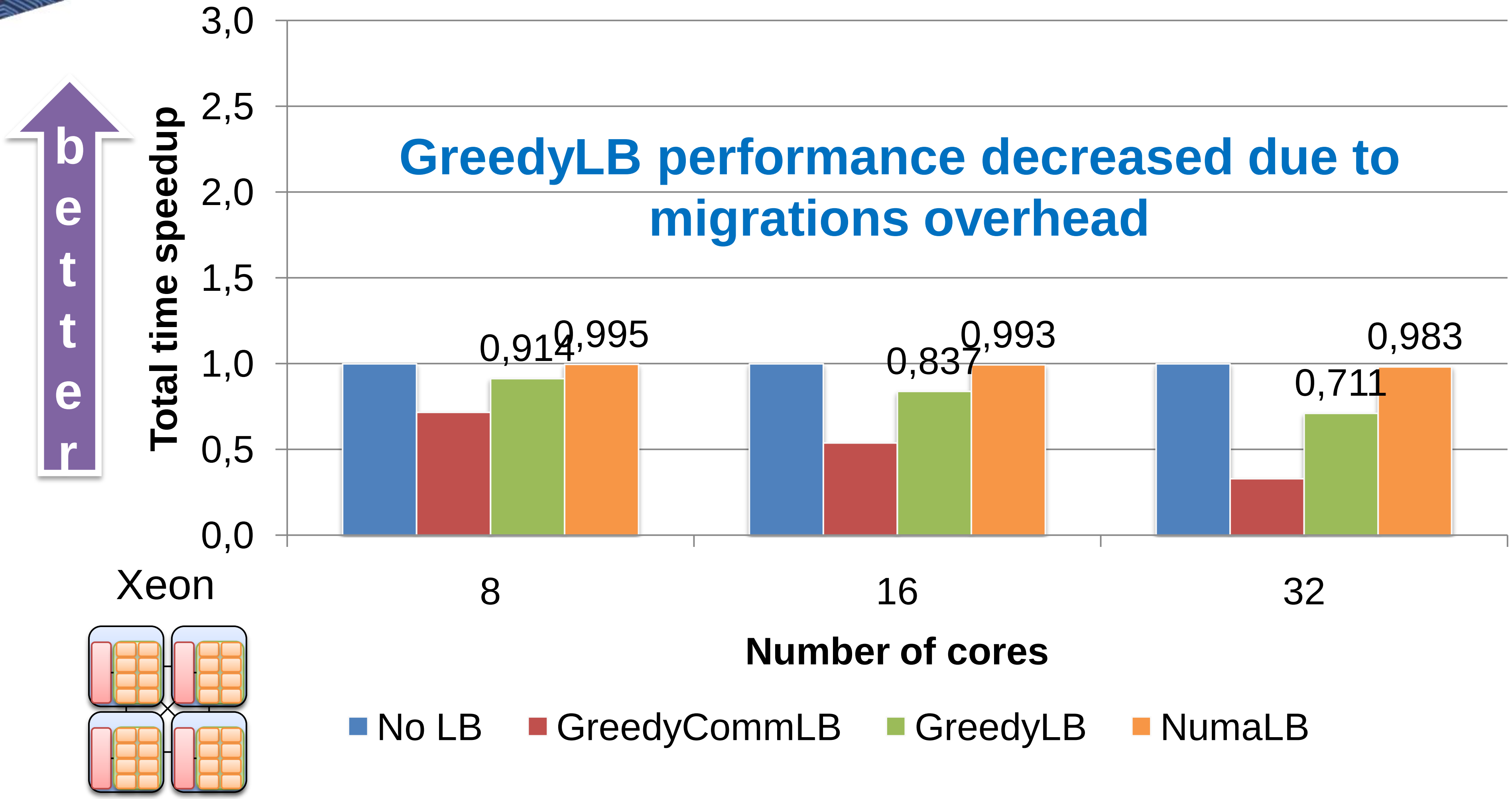
Benchmark: Poisson3D

- By Xavier Besseron and Thierry Gautier
- Solves the Poisson equation on a 3D domain
  - Parallelized by domain decomposition
- Well balanced

Poisson3D on Opteron Machine



Poisson3D on Xeon Machine



GreedyLB performance decreased due to migrations overhead

Outline

- Introduction
- Performance Evaluation of SMP Optimizations of Charm++ on NUMA Machines
- NUMA-aware Load Balancer on Charm++
- **Conclusion and Future Work**

Conclusion

- SMP **optimizations do affect the performance on NUMA** machines
  - Up to 50% between versions and 90% between architecture-specific compilations
- Gains with NUMA LB
  - **Speedups** of up to **2.8** (compared to no LB)
    - Performance near GreedyLB
  - Avoid migrations

Future Work

- Evolution of NUMA-aware LB
  - Consider **topology**
    - Number of hops
    - Cache hierarchy
  - **Memory** per chare
  - Improve NUMA information discovery
    - Initialization overheads
- Run experiments with **communication intensive benchmarks**
- Interface for a memory LB?

Charm++ on NUMA Platforms: the impact of SMP Optimizations and a NUMA-aware Load Balancer

Thank you

CAPES

.inf INSTITUTO DE INFORMÁTICA UFRGS

USP

GRENOBLE UNIVERSITÉ

INRIA

LIG