NUMA Support for Charm++

Does memory affinity matter?

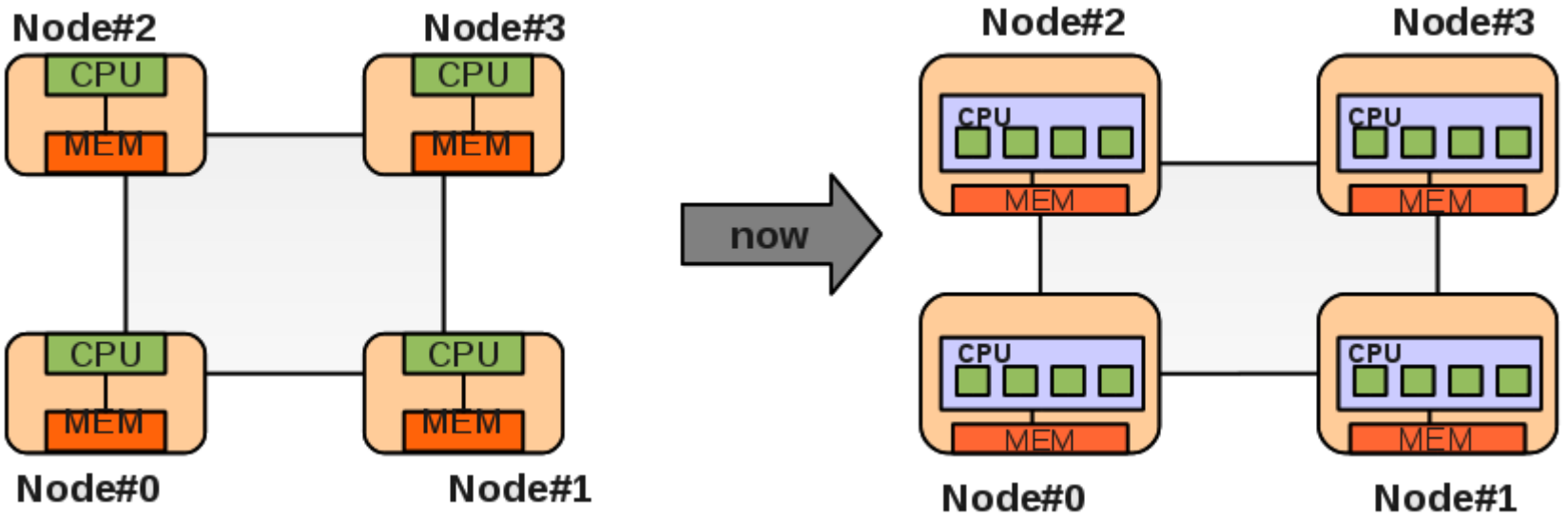
Christiane Pousa Ribeiro
Maxime Martinasso
Jean-François Méhaut

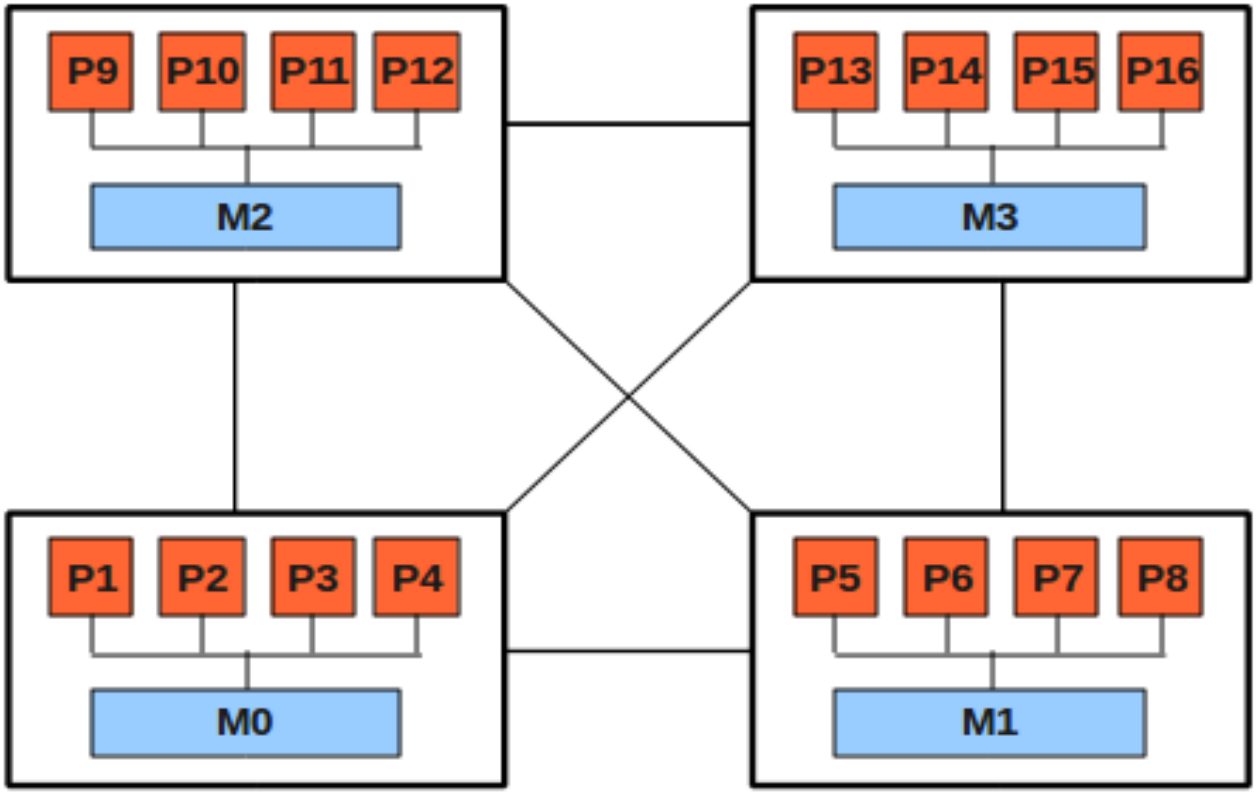
Outline

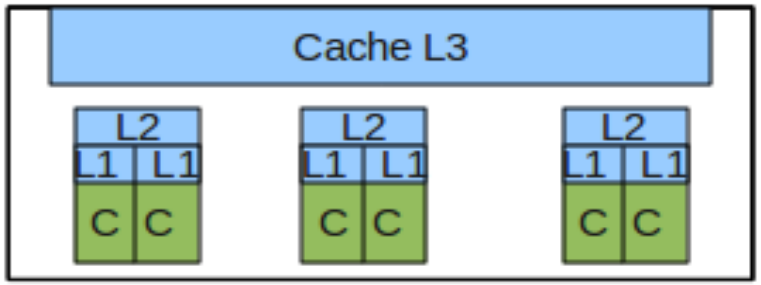
- Introduction
  - Motivation
  - NUMA Problem
- Support NUMA for Charm++
- First Results
- Conclusion and Future work

Motivation for NUMA Platforms

- The number of cores per processor is increasing
  - Hierarchical shared memory multiprocessors
  - cc-NUMA is **coming back** (NUMA factor)
  - AMD hypertransport and Intel QuickPath

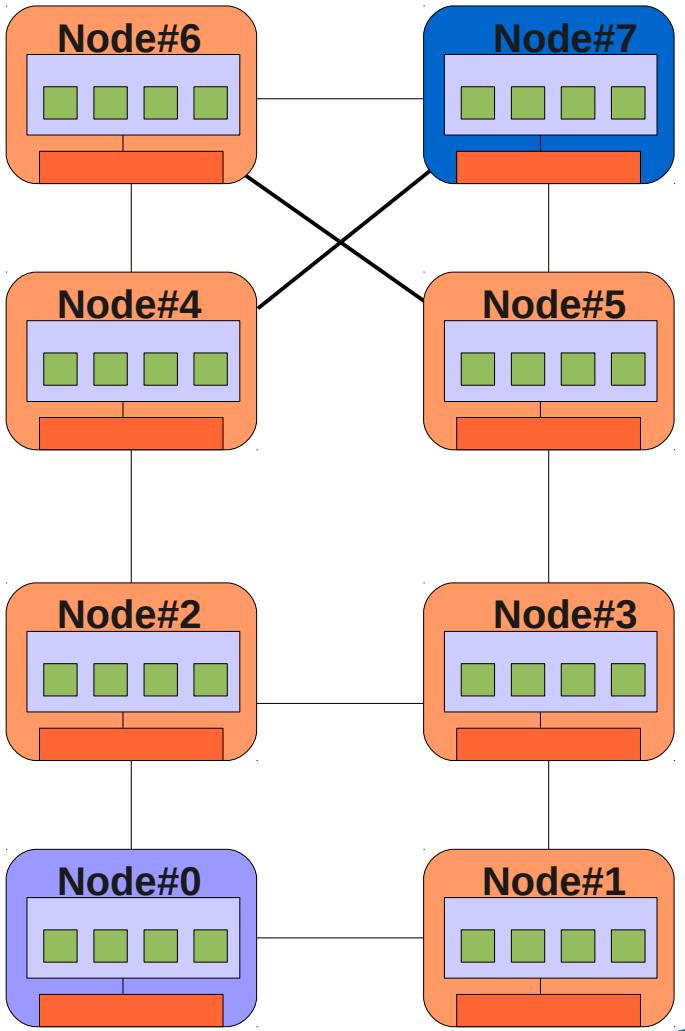



P9
P10
P11
P12
M2
P13
P14
P15
P16
M3
P1
P2
P3
P4
M0
P5
P6
P7
P8
M1


Cache L3
L2
L1
L1
C
C
L2
L1
L1
C
C
L2
L1
L1
C
C

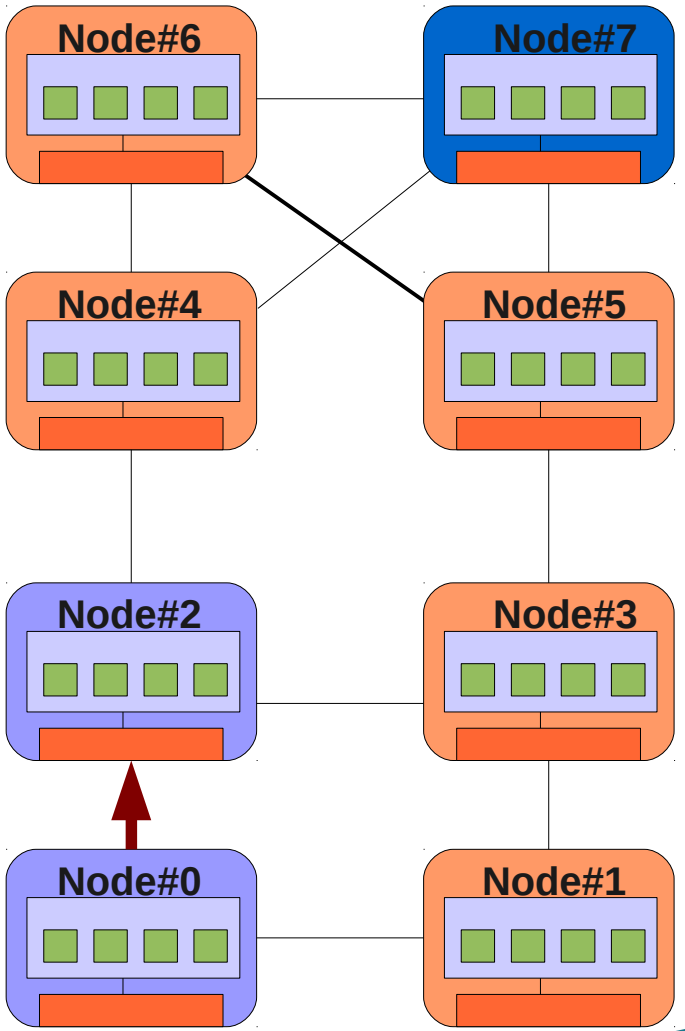
NUMA Problem

- **Remote access** and Memory contention
- Optimizes:
  - **Latency**
  - Bandwidth
- Assure memory affinity



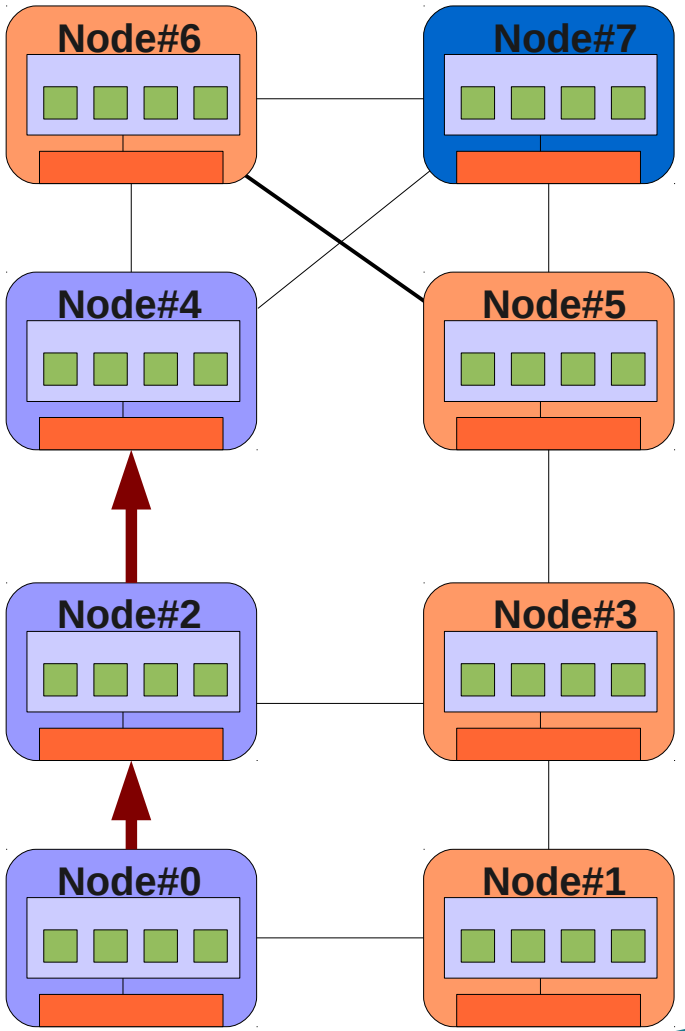
NUMA Problem

- **Remote access** and Memory contention
- Optimizes:
  - **Latency**
  - Bandwidth
- Assure memory affinity



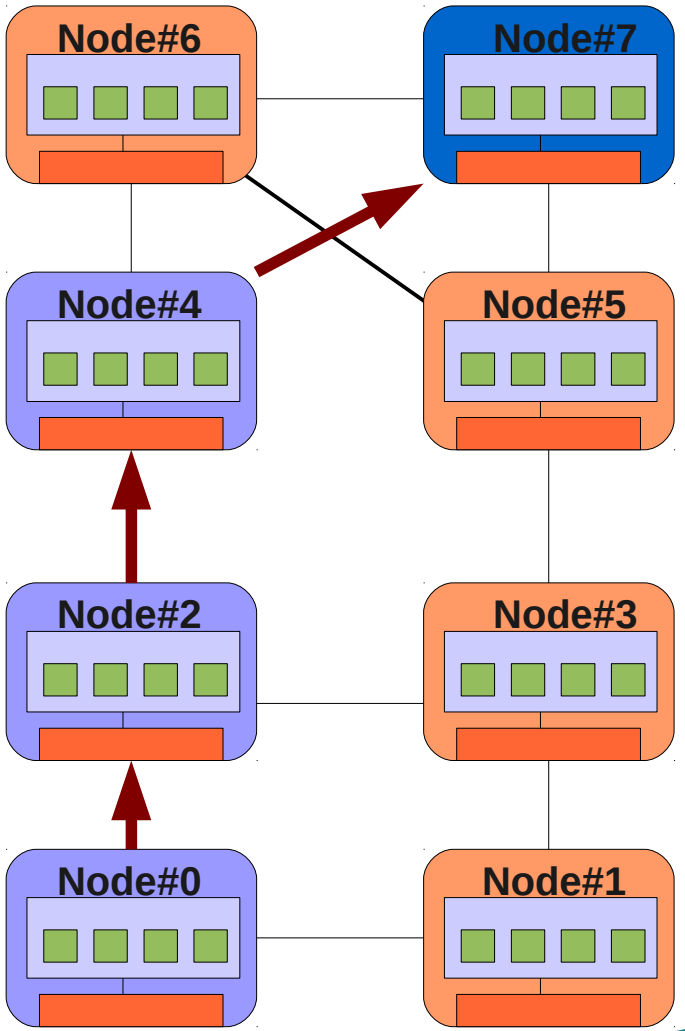
NUMA Problem

- **Remote access** and Memory contention
- Optimizes:
  - **Latency**
  - Bandwidth
- Assure memory affinity



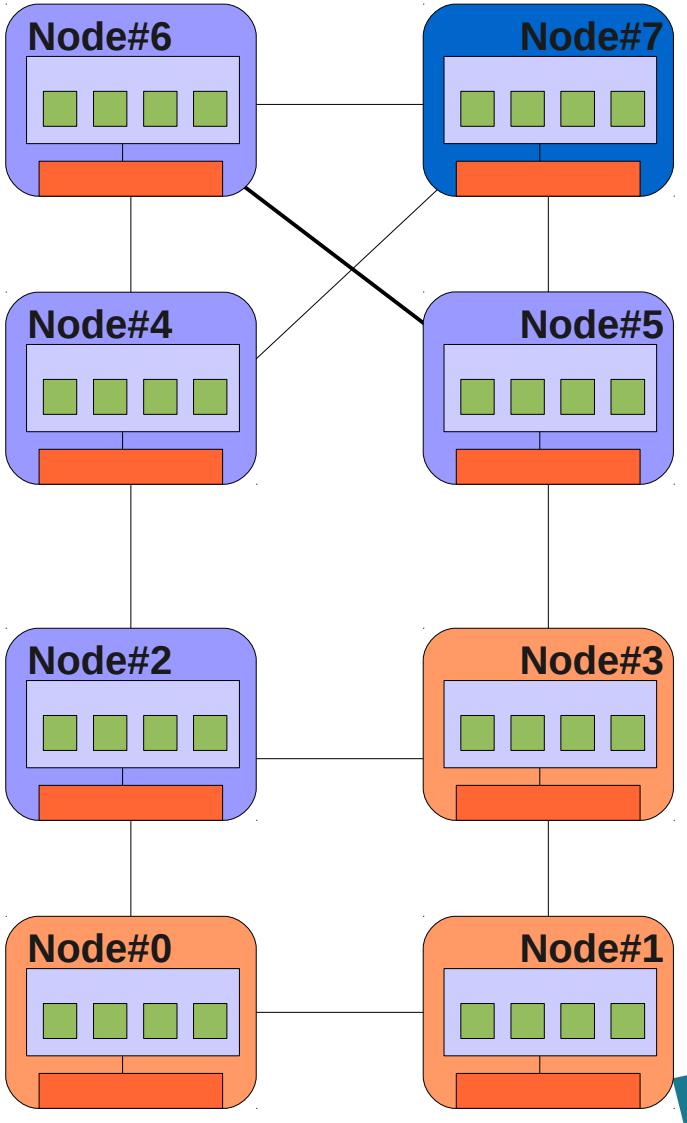
NUMA Problem

- **Remote access** and Memory contention
- Optimizes:
  - **Latency**
  - Bandwidth
- Assure memory affinity



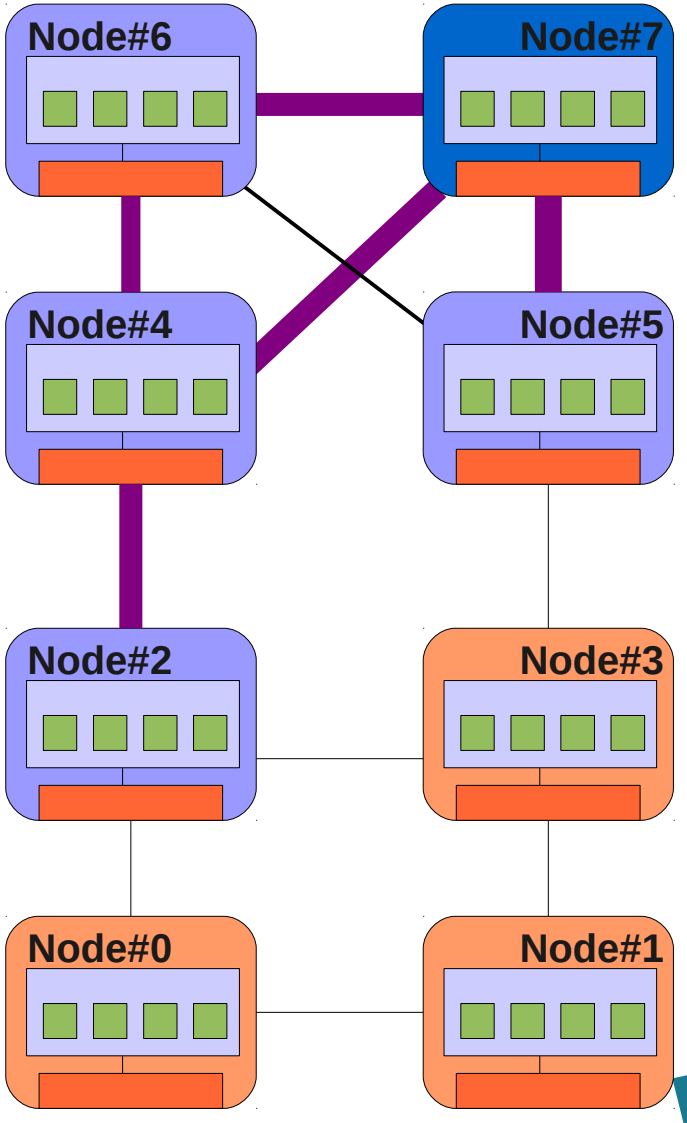
NUMA Problem

- Remote access and **Memory contention**
- Optimizes:
  - Latency
  - **Bandwidth**
- Assure memory affinity



NUMA Problem

- Remote access and **Memory contention**
- Optimizes:
  - Latency
  - **Bandwidth**
- Assure memory affinity



NUMA Problem

- Memory access types:
  - Read and write
  - **Different costs**
- Write operations are more expensive
  - Special memory policies
- On NUMA, data distribution matters!

NUMA support on Operating Systems

- Operating systems have some support for NUMA machines
- Physical memory allocation:
  - First-touch, next-touch
- Libraries and tools to distribute data

Memory Affinity on Linux

- The actual support for NUMA on Linux:
  - Physical memory allocation:
    - First-touch: first memoy access
  - NUMA API: developers do all!
    - System call to bind memory pages
    - **Numactl**, user-level tool to bind memory and to pin threads
    - **Libnuma** an interface to place memory pages on physical memory

Charm++ Parallel Programming System

- Portability over different platforms
  - Shared memory
  - Distributed memory
- Architecture abstraction => programmer productivity
- Virtualization and transparence

Charm++ Parallel Programming System

- Data management:
  - Stack and Heap
- Memory allocation based on malloc
- Isomalloc:
  - based on mmap system call
  - allows threads migration
- What about physical memory?

NUMA Support on Charm++

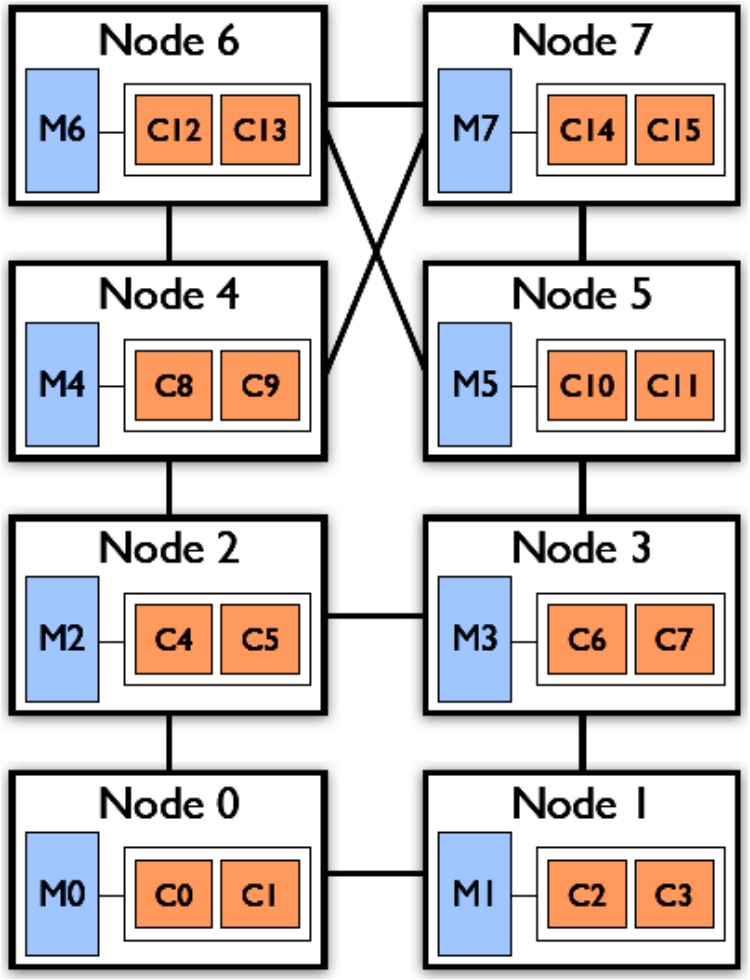
- Our approach
  - Study the impact of memory affinity on charm++
  - Bind virtual memory pages to memory banks
- Based on three parts:
  - +maffinity option
  - Interleaved heap
  - NUMA-aware memory allocator

Impact of Memory Affinity on charm++

- Study the impact of memory affinity
  - different memory **allocators** and memory **policies**
- Memory allocators
  - ptmalloc and NUMA-aware tcmalloc
- Memory policies
  - First-touch, bind and interleaved
- NUMA machine: AMD Opteron

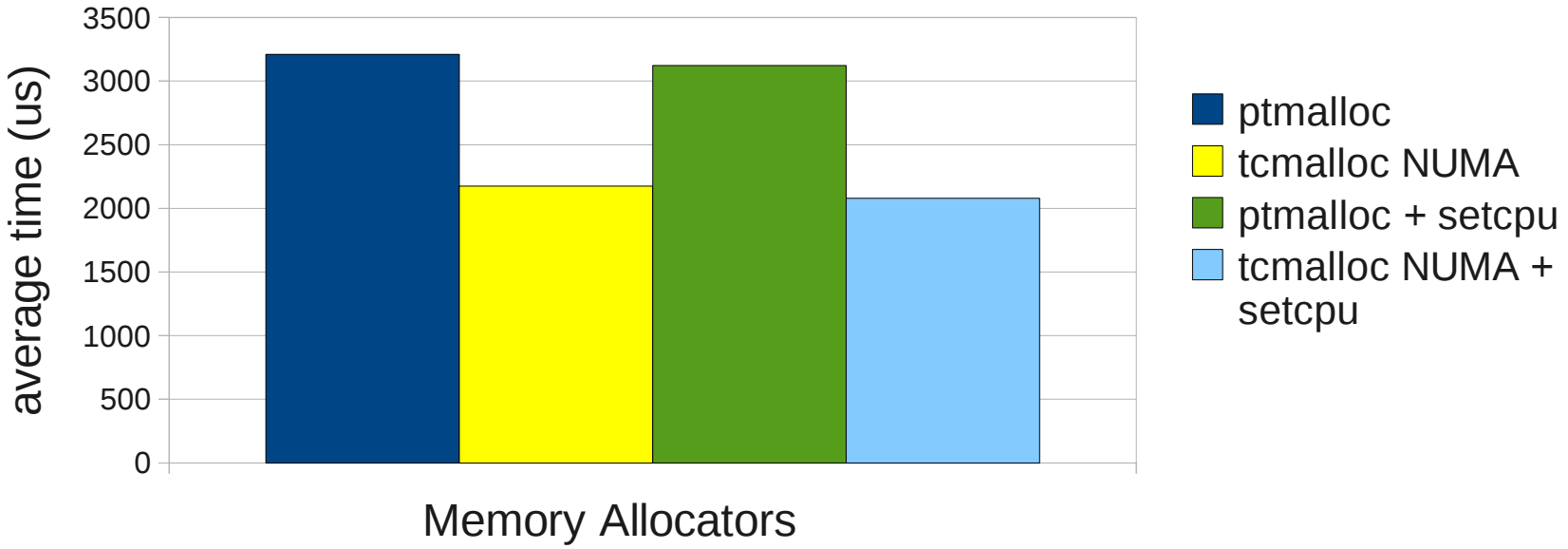
AMD Opteron

- NUMA machine
  - AMD Opteron
  - 8 (2 cores) x 2.2GHz processors
  - Cache L2 (2Mbytes)
  - Main memory 32Gbytes
  - Low latency for local memory access
  - Numa factor: 1.2 – 1.5
  - Linux 2.6.32.6



Different Memory Allocators

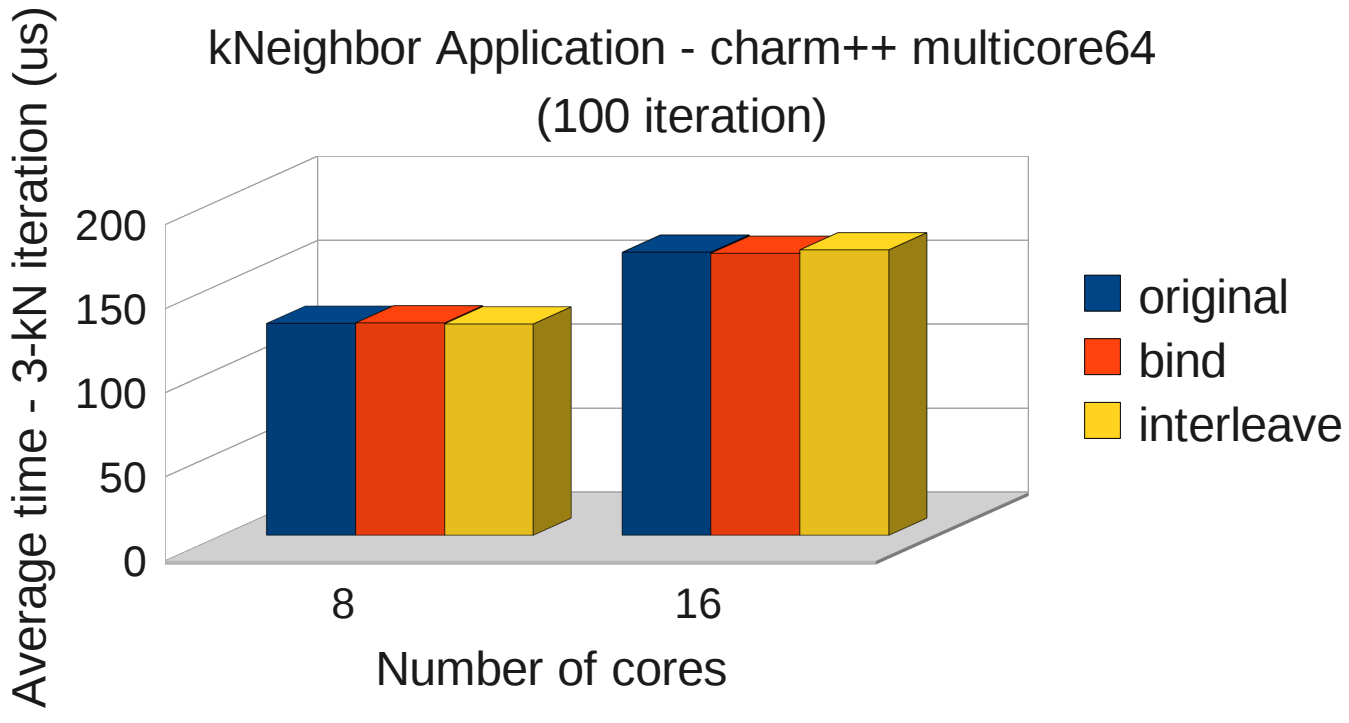
kNeighbor Application - charm++ multicore64



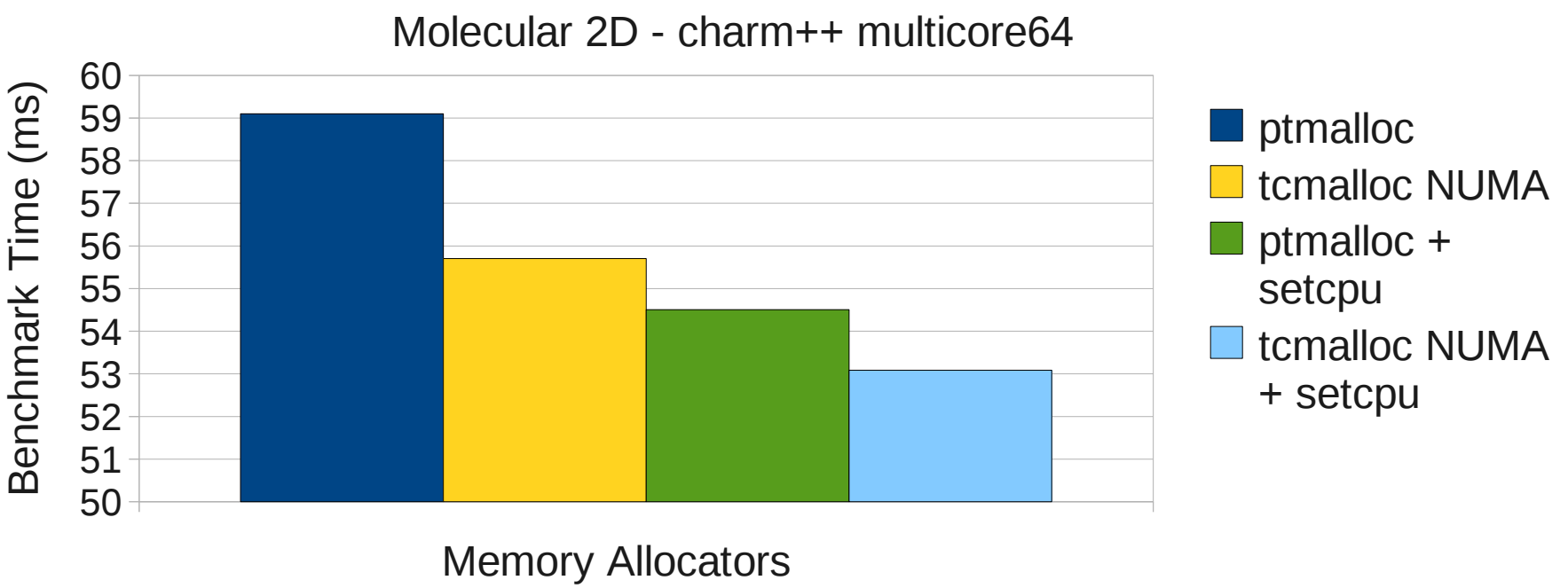
numactl

kNeighbor Application - charm++ multicore64

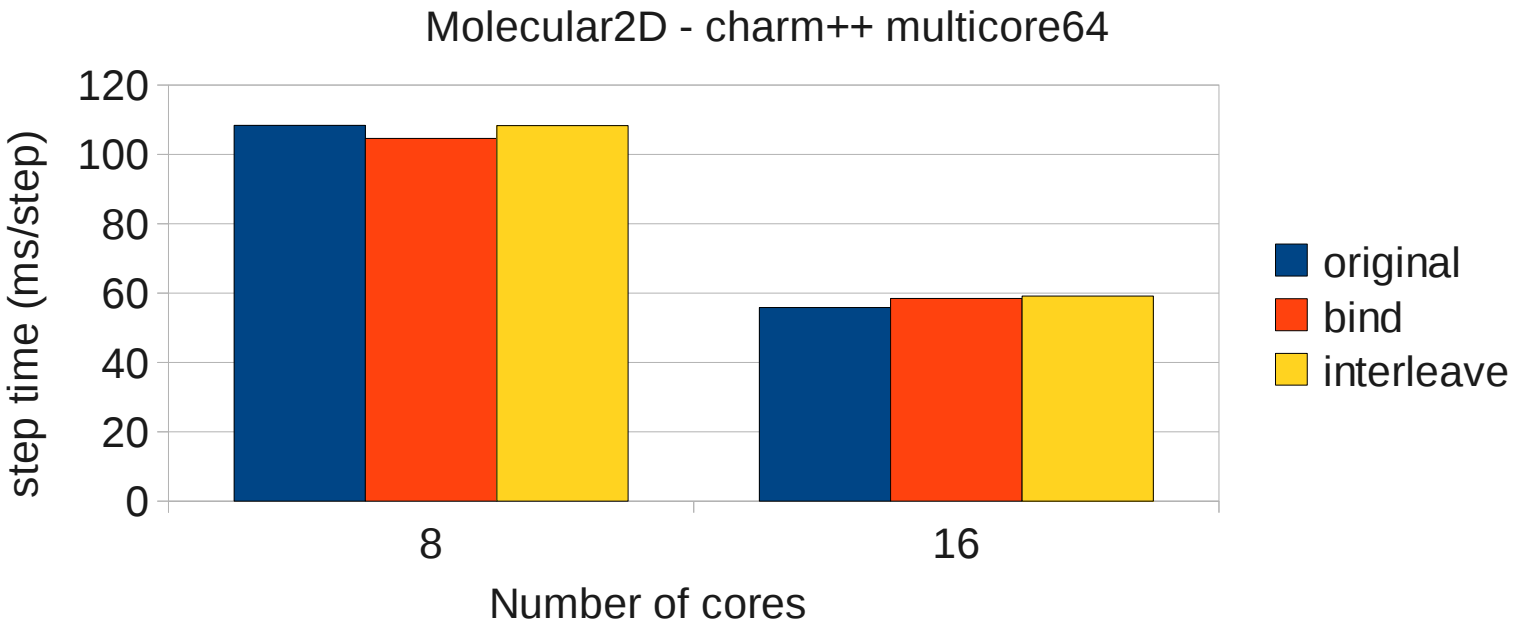
(100 iteration)



Different Memory Allocators



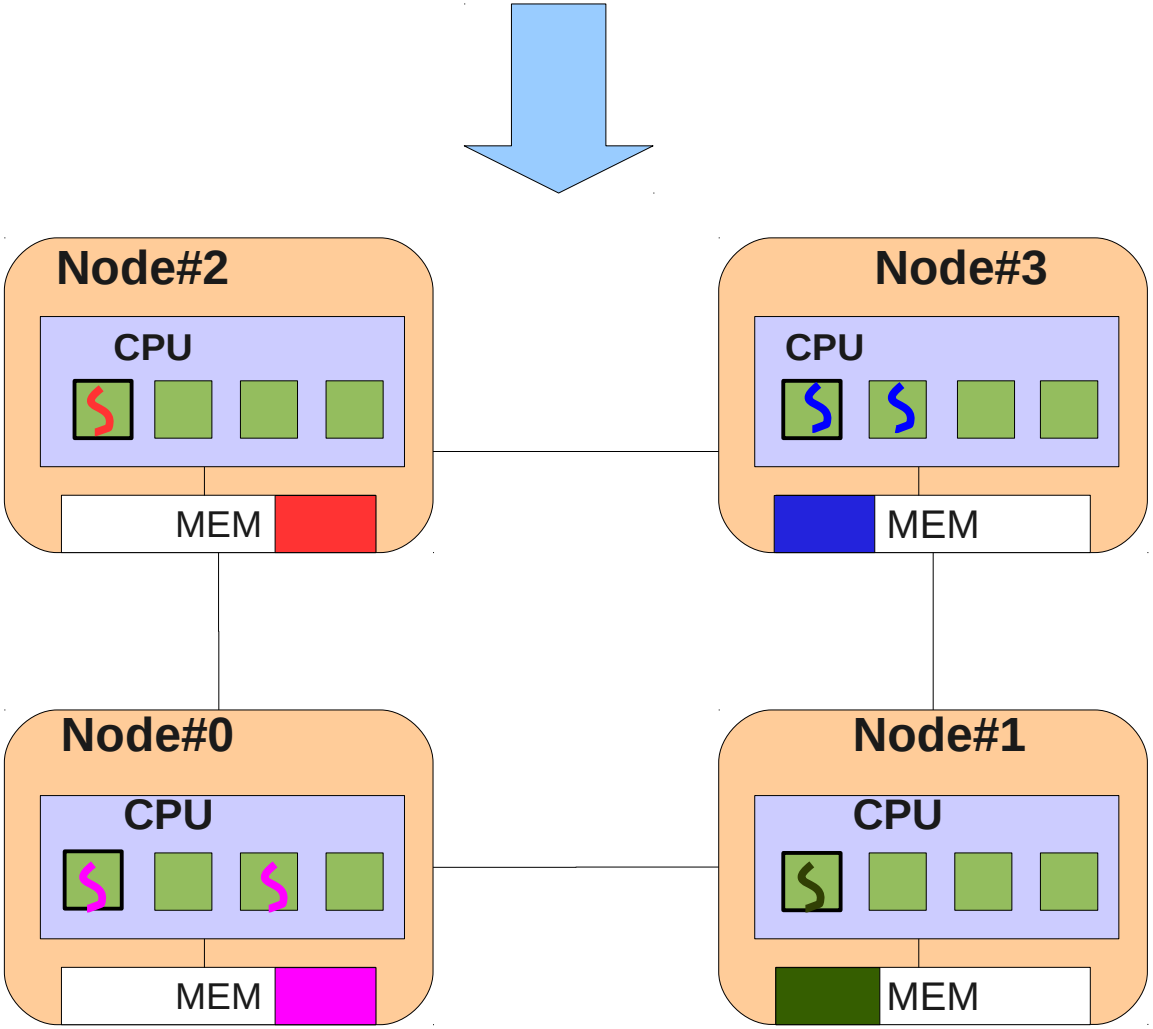
numactl



+maffinity option

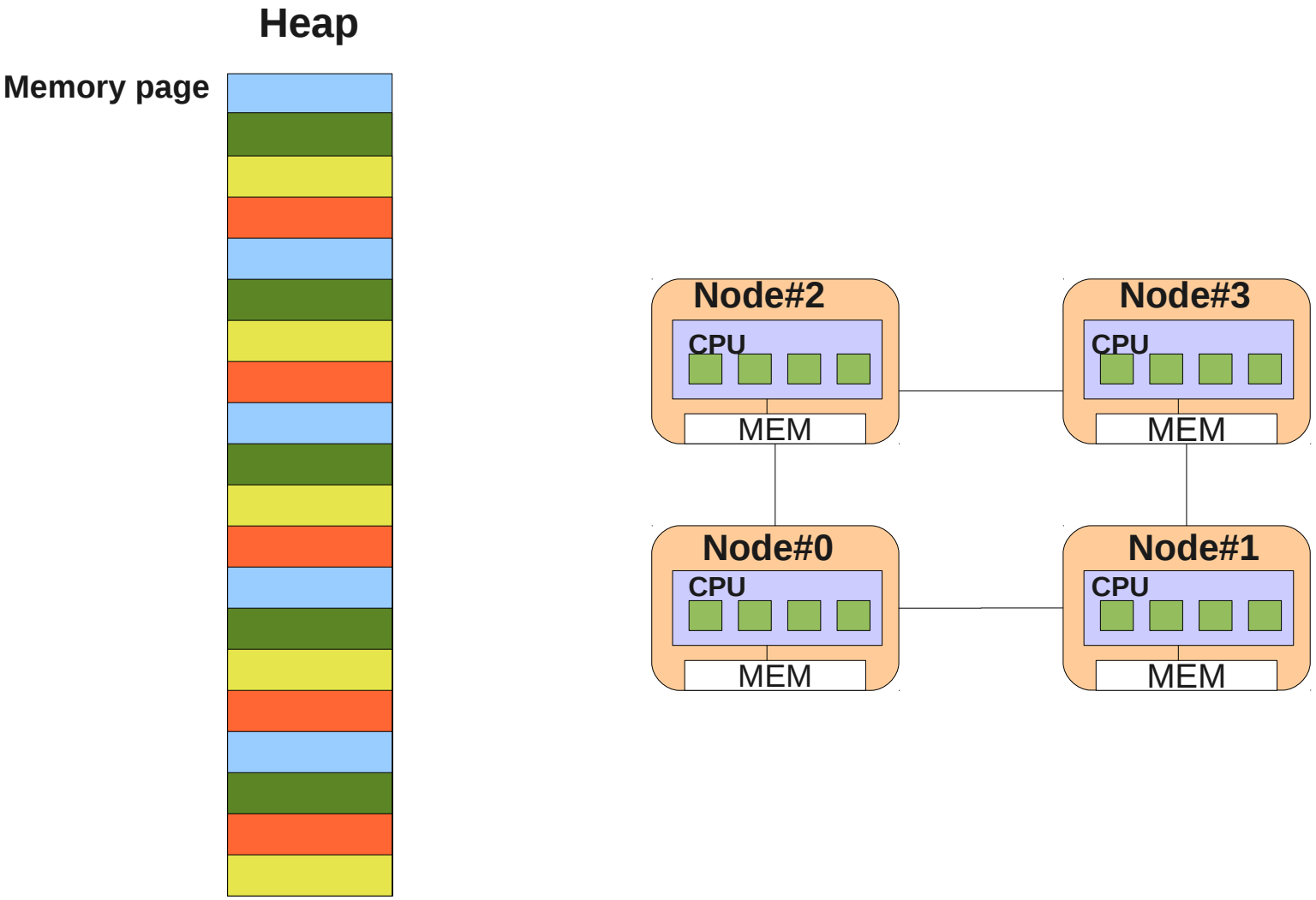
- set memory affinity for processes or threads
- Based on Linux NUMA system call
  - Set the process/thread memory policy
  - Bind, preferred and interleave are used in our implementation
- Must be used with +setcpuaffinity option

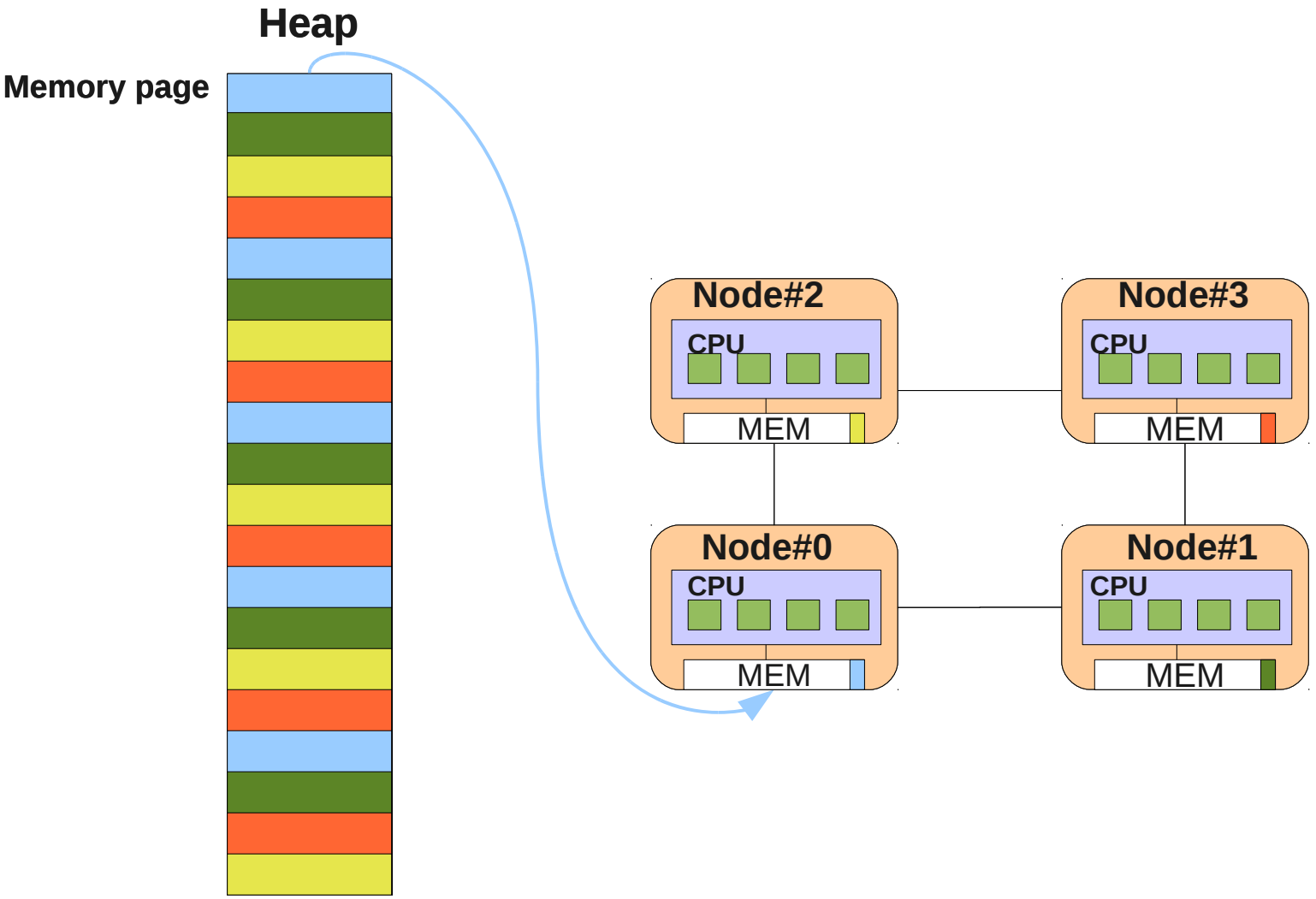
./charmrun **prog** +p6 +setcpuaffinity +coremap 0,2,4,8,12,13 **+maffinity +nodemap 0,0,1,2,3,3 +mempol preferred**

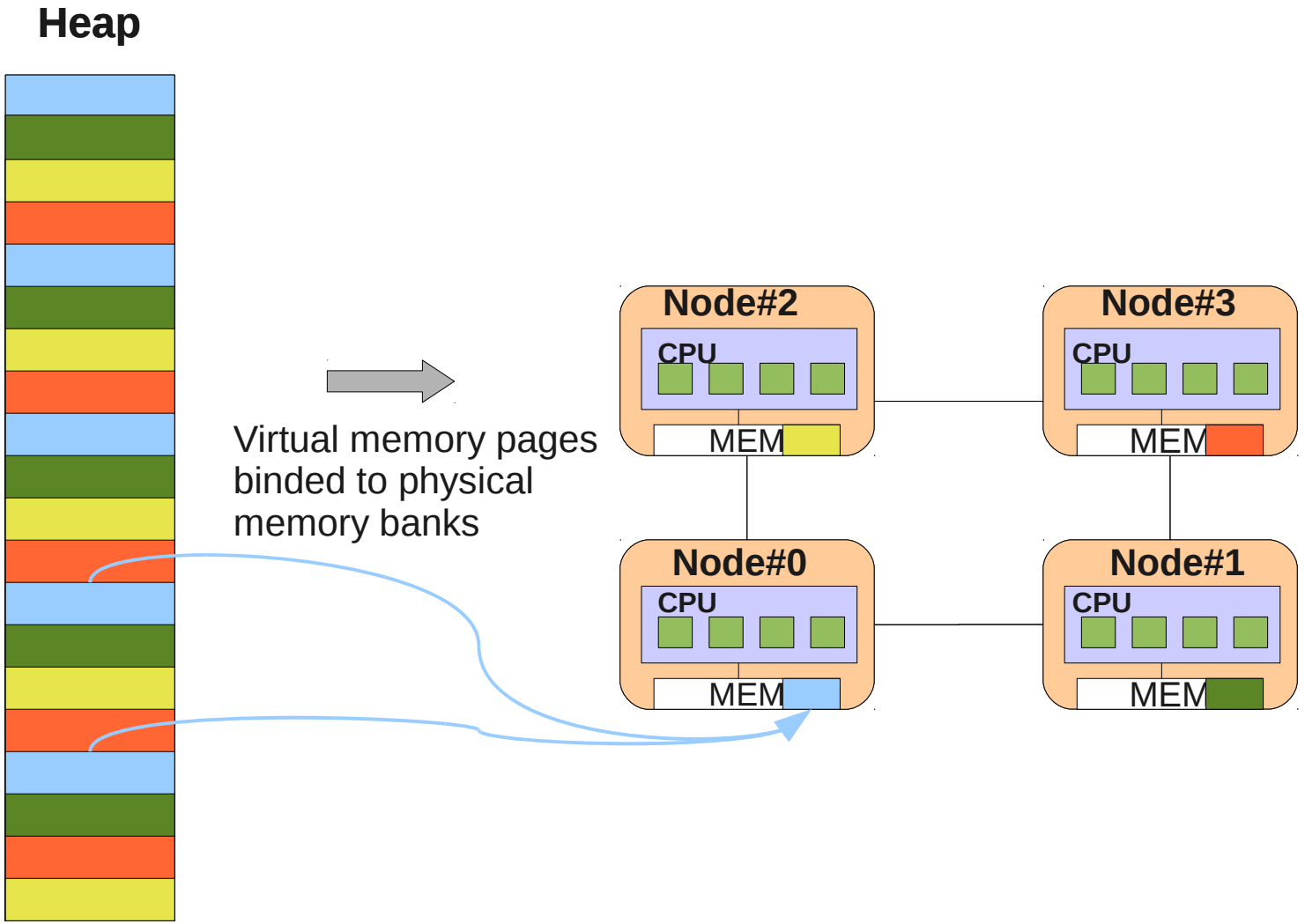


Interleaved Heap

- Based on mbind Linux system call
- Spread data over the NUMA nodes
- The objective is to reduce memory contention by optimizing bandwidth
- One mbind per mmap


Heap
Memory page
Node#2
CPU
MEM
Node#3
CPU
MEM
Node#0
CPU
MEM
Node#1
CPU
MEM


Heap
Memory page
Node#2
CPU
MEM
Node#3
CPU
MEM
Node#0
CPU
MEM
Node#1
CPU
MEM

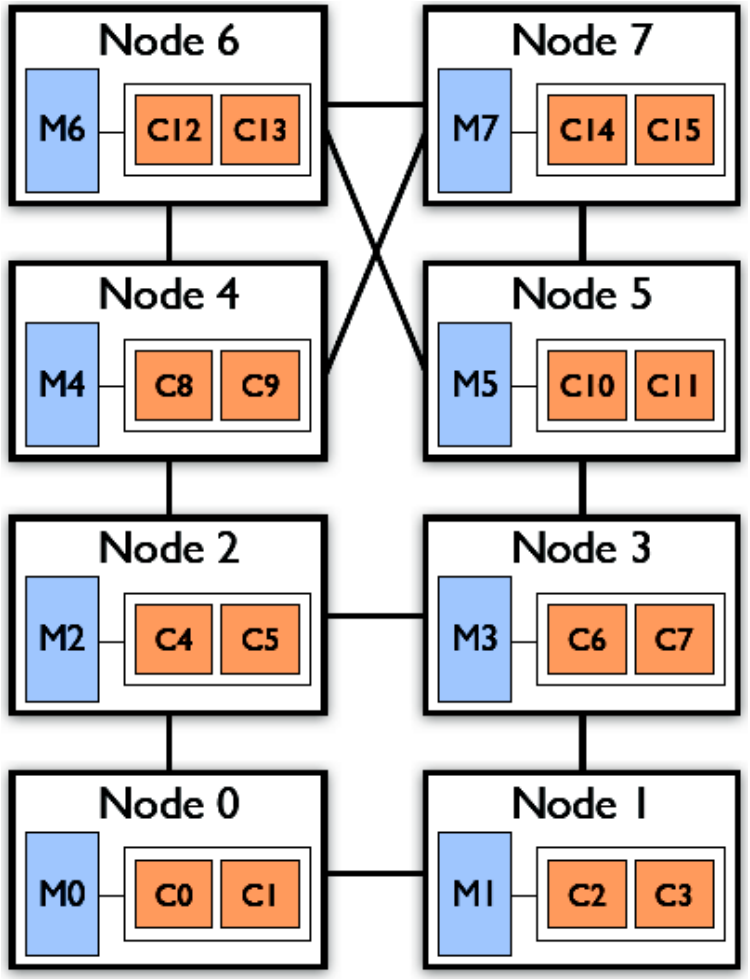

Heap
Virtual memory pages
binded to physical
memory banks
Node#2
CPU
MEM
Node#3
CPU
MEM
Node#0
CPU
MEM
Node#1
CPU
MEM

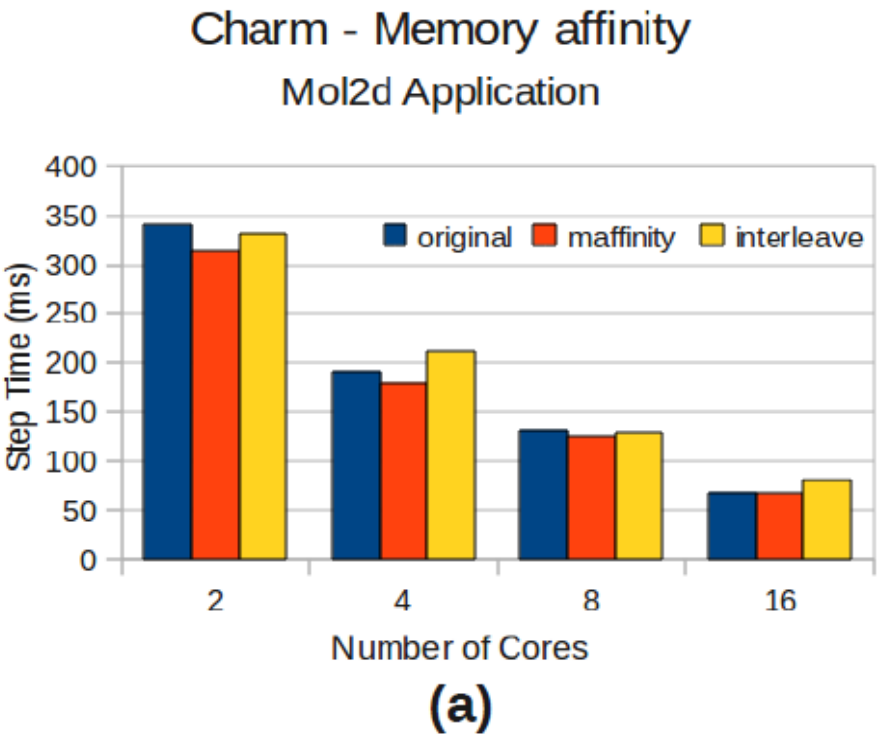
First Results

- Charm++ version:
  - 6.1.3
  - net-linux-amd64
- Applications:
  - Molecular2D
  - Kneighbor (1000 iterations - msg 1024)

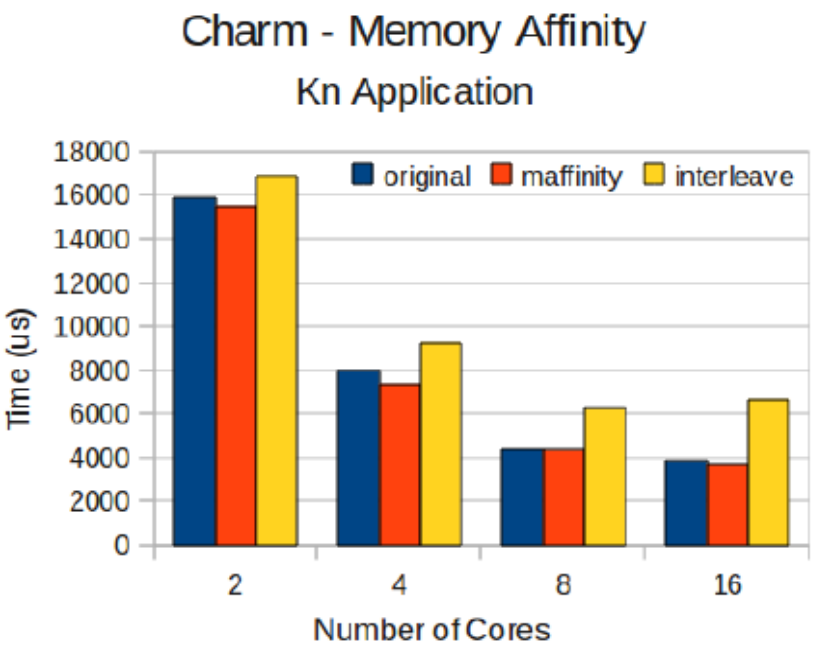
First Results

- NUMA machine
  - AMD Opteron
  - 8 (2 cores) x 2.2GHz processors
  - Cache L2 shared (2Mbytes)
  - Main memory 32Gbytes
  - Low latency for local memory access
  - Numa factor: 1.2 – 1.5
  - Linux 2.6.32.6




Charm - Memory affinity
Mol2d Application
Step Time (ms)
400
350
300
250
200
150
100
50
0
original
maffinity
interleave
2
4
8
16
Number of Cores

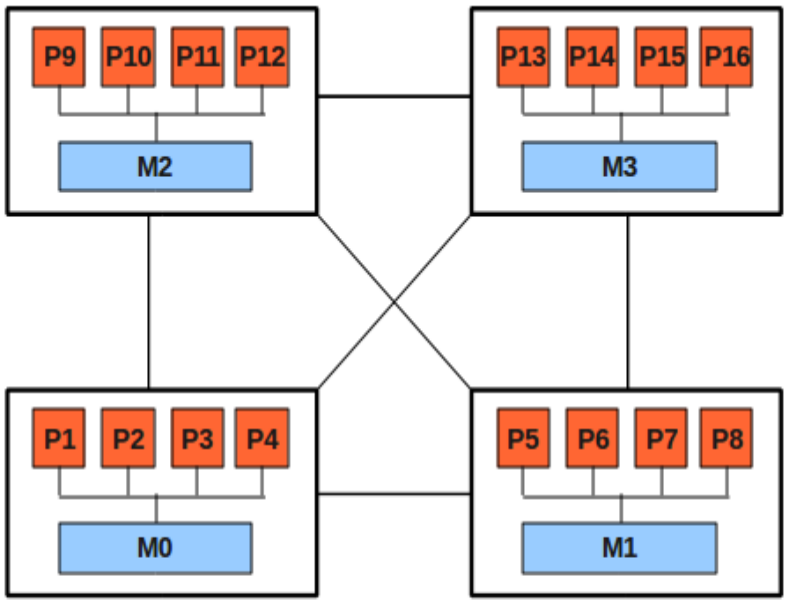
(a)

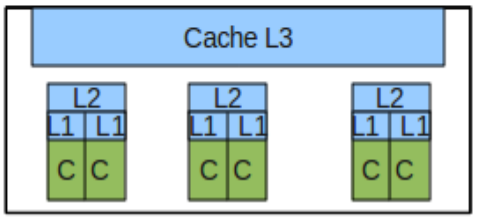

Charm - Memory Affinity
Kn Application
Time (us)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
original
maffinity
interleave
2
4
8
16
Number of Cores

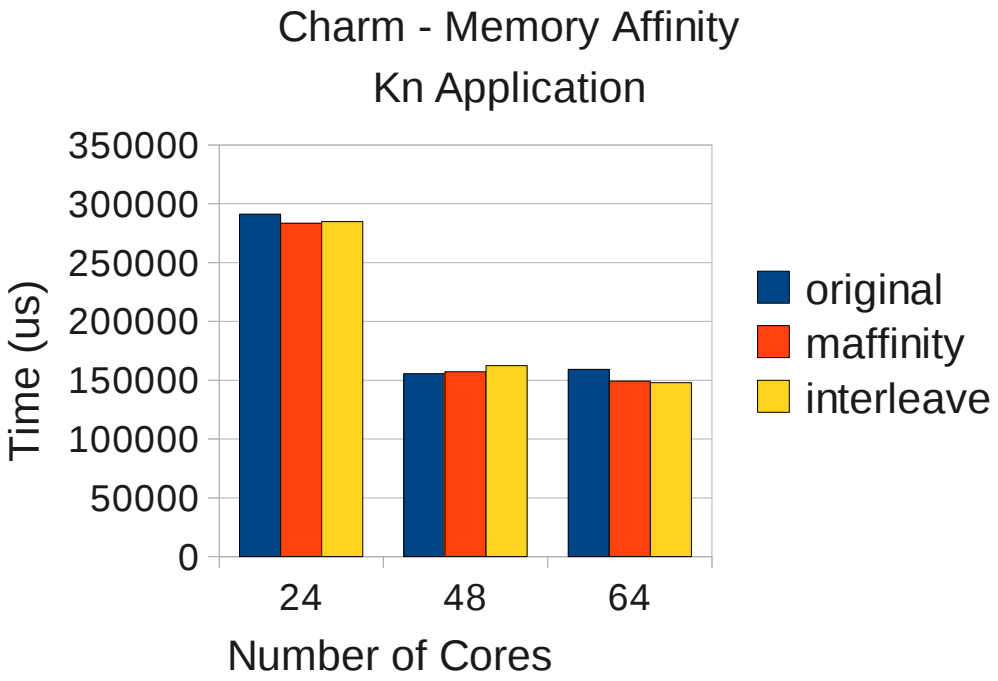
(b)

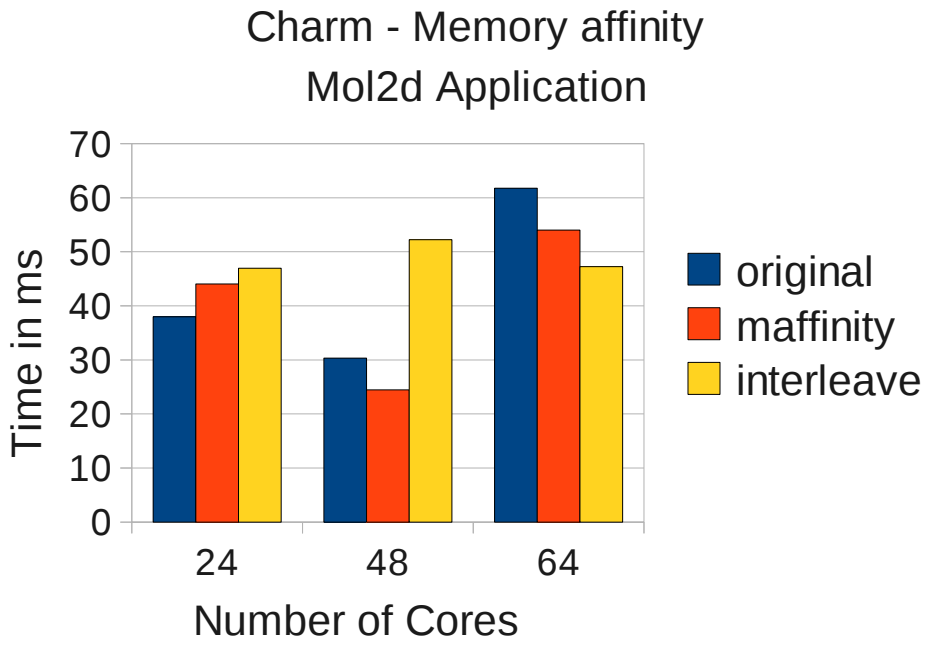
Intel Xeon

- NUMA machine
  - Intel EM64T
  - 4 (24 cores) x 2.66GHz processors
  - Shared cache L3 (16MB)
  - Main memory 192Gbytes
  - High latency for local memory access
  - Numa factor: 1.2 - 5
  - Linux 2.6.27






Charm - Memory Affinity
Kn Application
350000
300000
250000
200000
150000
100000
50000
0
Time (us)
24
48
64
Number of Cores
original
maffinity
interleave


Charm - Memory affinity
Mol2d Application
70
60
50
40
30
20
10
0
Time in ms
24
48
64
Number of Cores
original
maffinity
interleave

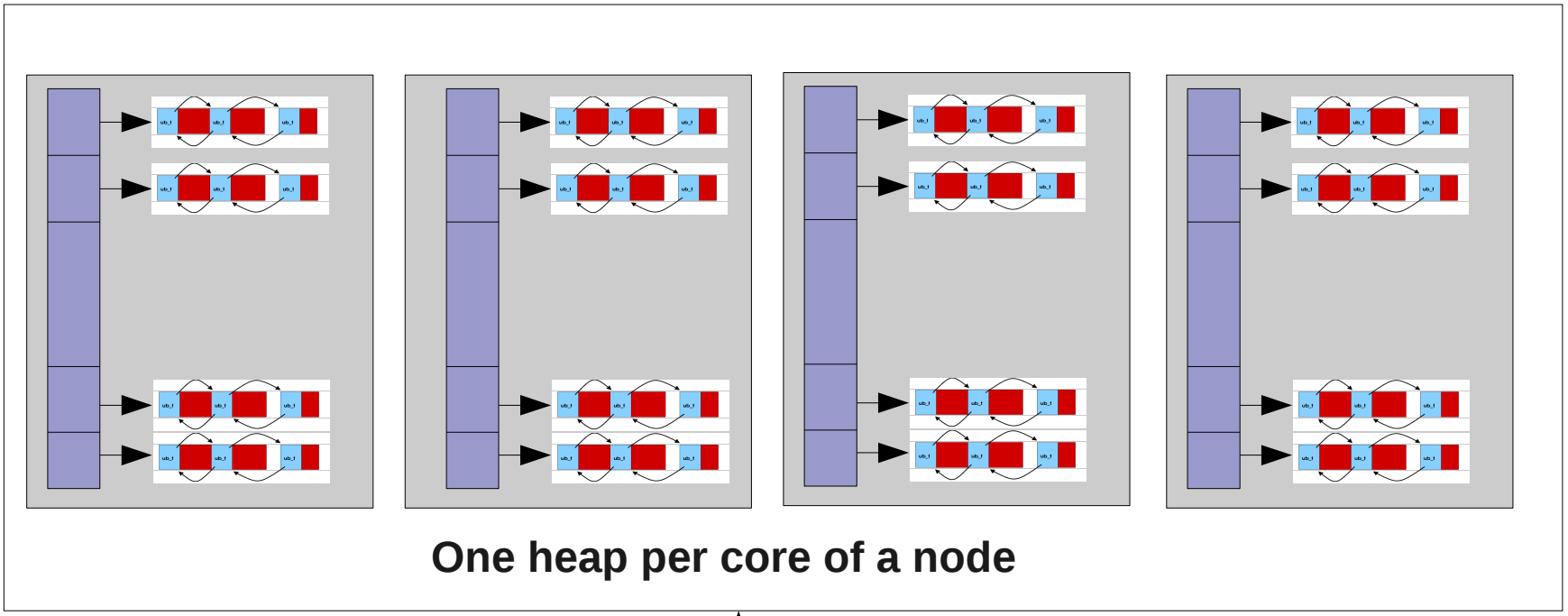
HeapAlloc

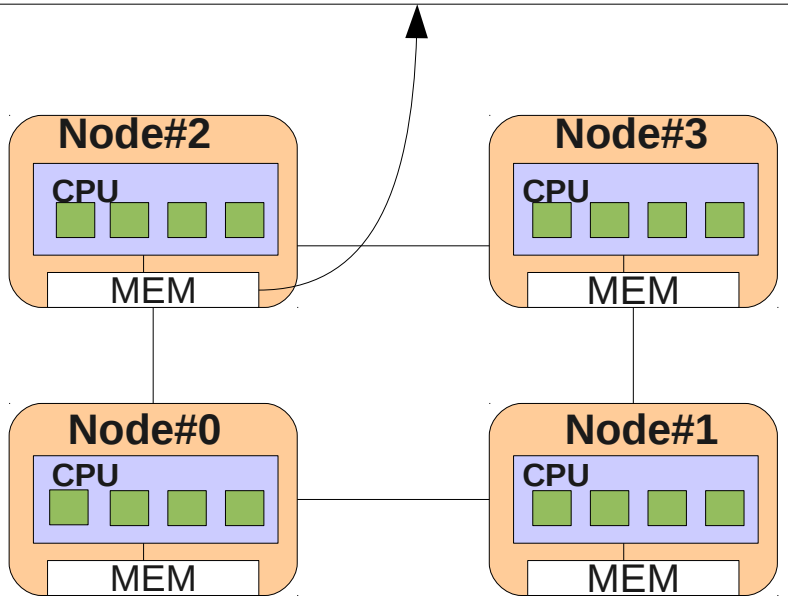
- NUMA-aware memory allocator
- Reduces lock contention and optimizes data locality
- Several memory policies: applied considering the access mode (read, write or read/write)

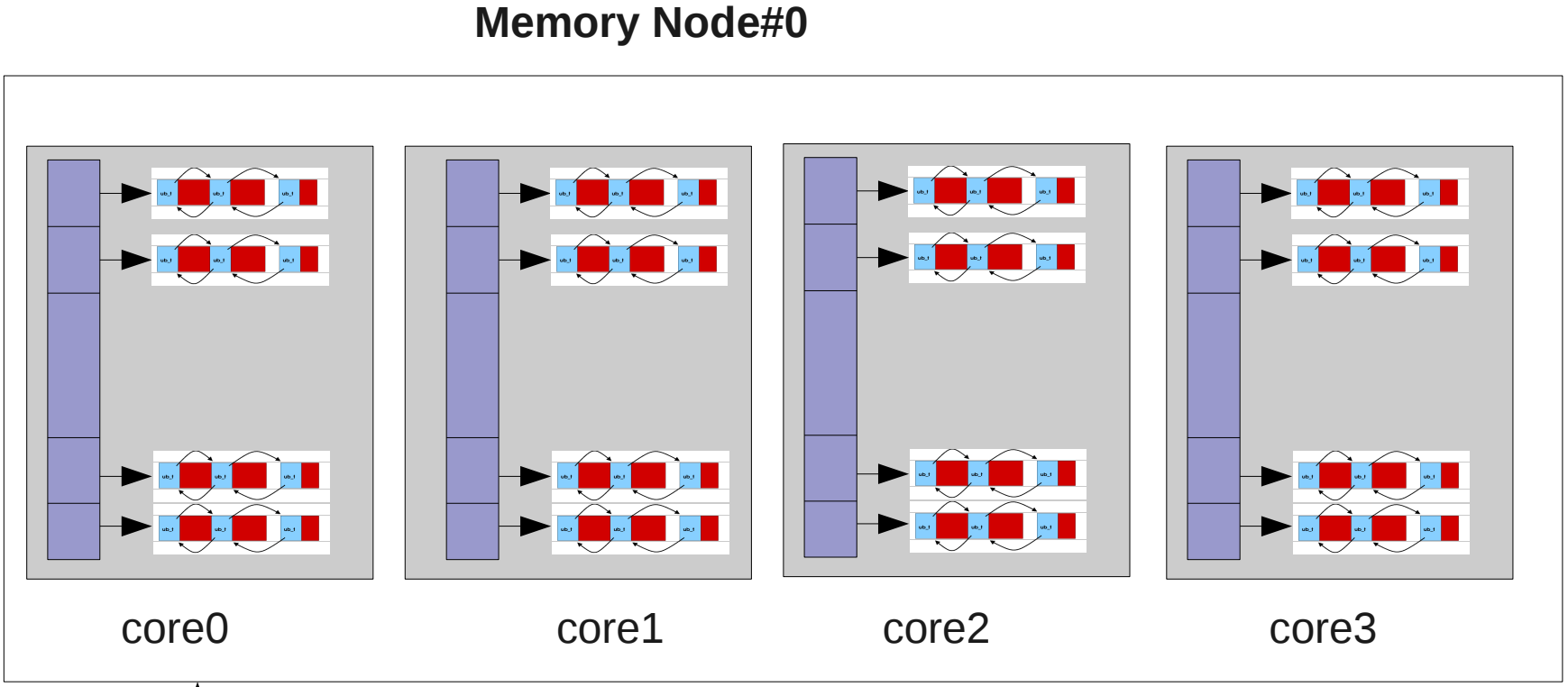
HeapAlloc

- Default memory policy is bind
- High-level interface: glibc compatible, any modifications in source code
- Low-level interface: allows developers to manage their heaps

Memory Node#2

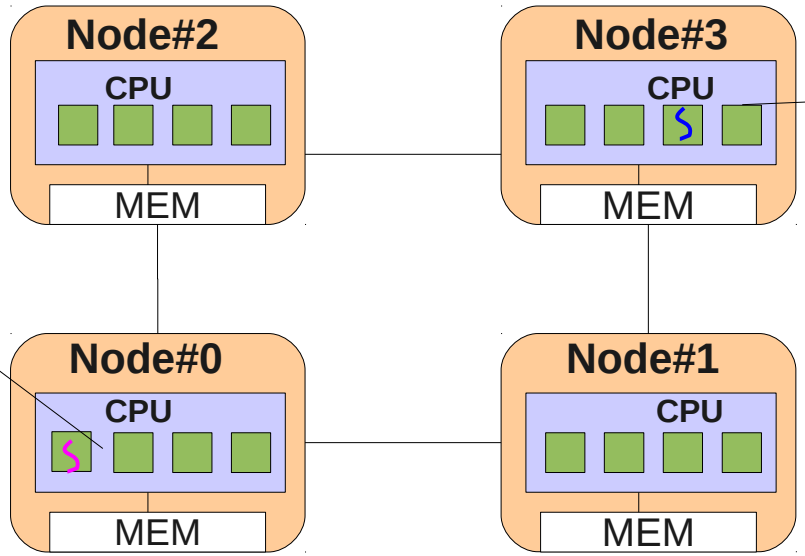






Memory is allocated from heap 'core0'

Thread running on node#0 calls **malloc**



Thread running on node#3 calls **free** for memory allocated by thread

Memory is returned to heap 'core0'

Conclusions

- Charm++ performance on NUMA can be improved
  - Tcmalloc NUMA-aware
  - +maffinity
  - Interleaved Heap
- Proposal of an optimized memory allocator for NUMA machines

Future Work

- Conclude the integration of HeapAlloc in charm++
- Study the impact of different memory allocators on charm++
- What about several memory policies?
  - Bind, interleave, next-touch, skew_mapp .....

Questions?

pousa@imag.fr