Improving Virtual Machine Scheduling in NUMA Multicore Systems

Jia Rao, Xiaobo Zhou
University of Colorado, Colorado Springs

Kun Wang, Cheng-Zhong Xu
Wayne State University

http://cs.uccs.edu/~jrao/

University of Colorado
Colorado Springs

Multicore Systems

- Fundamental platform for datacenters and HPC
  - power-efficiency & parallelism
- Performance degradation and unpredictability
  - contention on shared resources: last-level cache, memory controller ...
- NUMA architecture further complicates scheduling
  - low NUMA factor, remote access is not the only/main concern
- Virtualization
  - app or OS-level optimizations ineffective due to inaccurate virtual topology

Multicore Systems

- Fundamental platform for datacenters and HPC
    - power-efficiency & parallelism
- Performance degradation and unpredictability
    - co

Objective:
Improve performance and reduce variability

- NUMA architecture further complicates scheduling
    - low NUMA factor, remote access is not the only/main concern
- Virtualization
    - app or OS-level optimizations ineffective due to inaccurate virtual topology

UCCS University of Colorado
Colorado Springs

Multicore Systems

- Fundamental platform for datacenters and HPC
  - power-efficiency & parallelism
- Performance degradation and unpredictability

Objective:
Improve performance and reduce variability

  - co

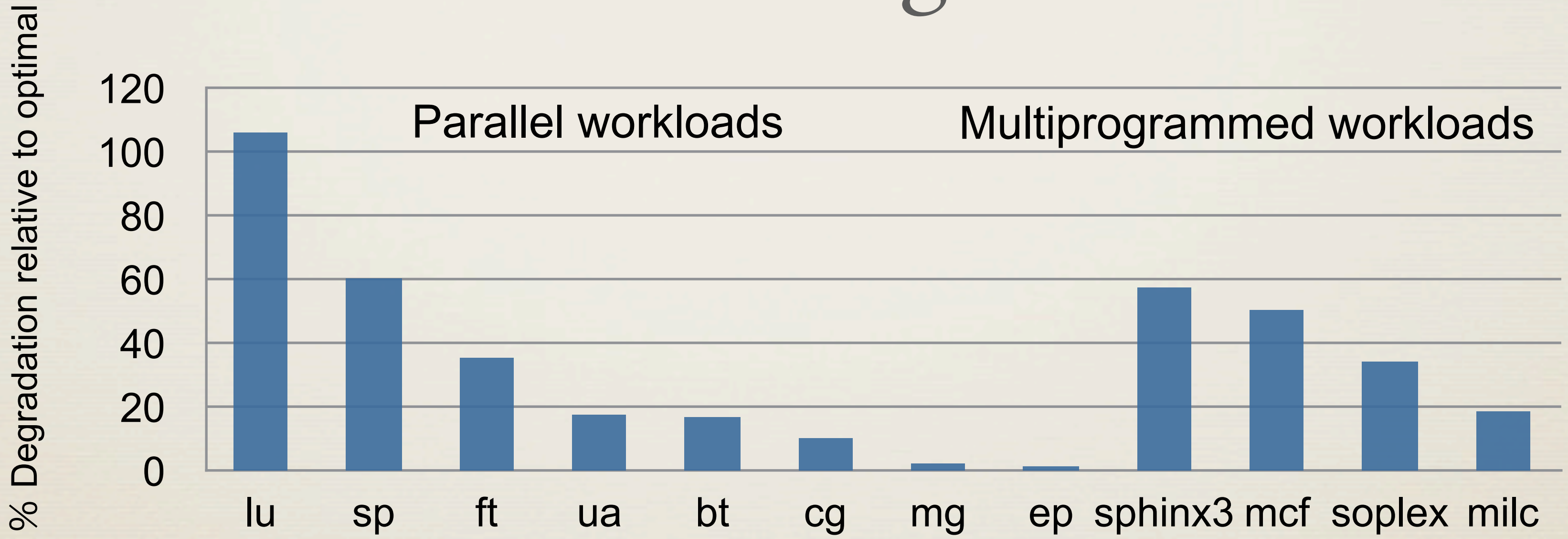
- NU ... ling

Approach:
Add NUMA and contention awareness to virtual machine scheduling

  - lov
- Virtualization
  - app or OS-level optimizations ineffective due to inaccurate virtual topology

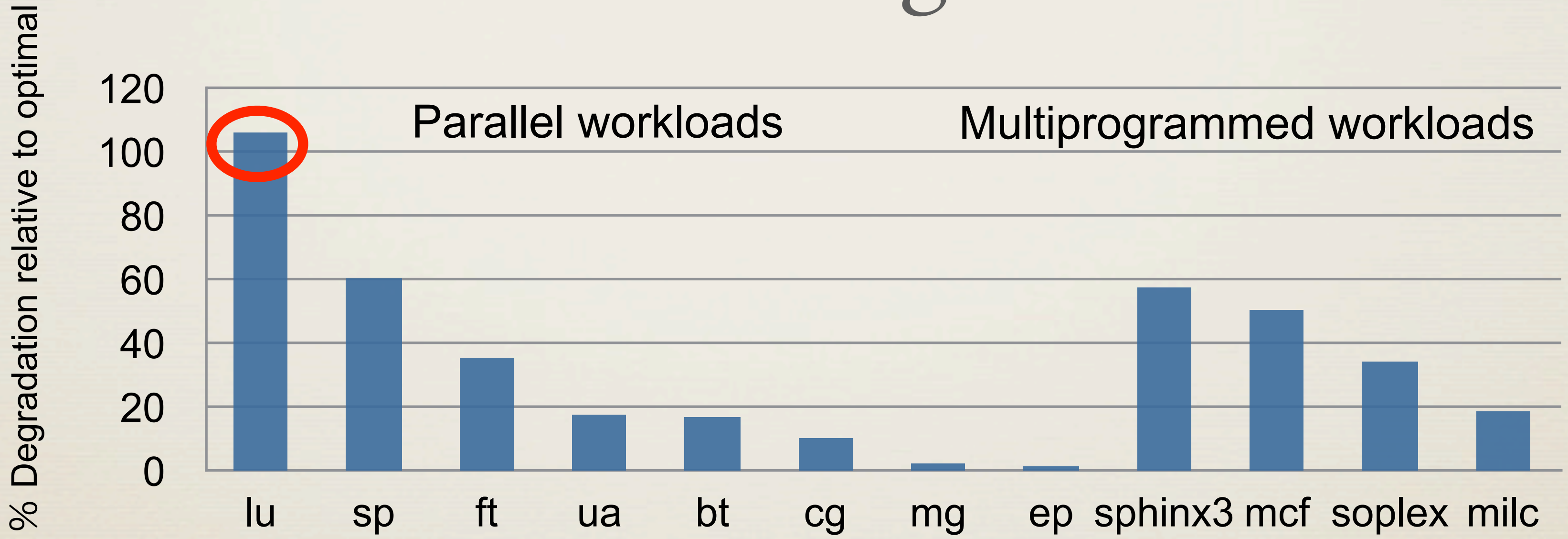
Motivation

- Enumerate different thread-to-core assignments
  - 4 threads on two-socket Intel Westmere NUMA machine
- Calculate the worst to best degradation



Motivation

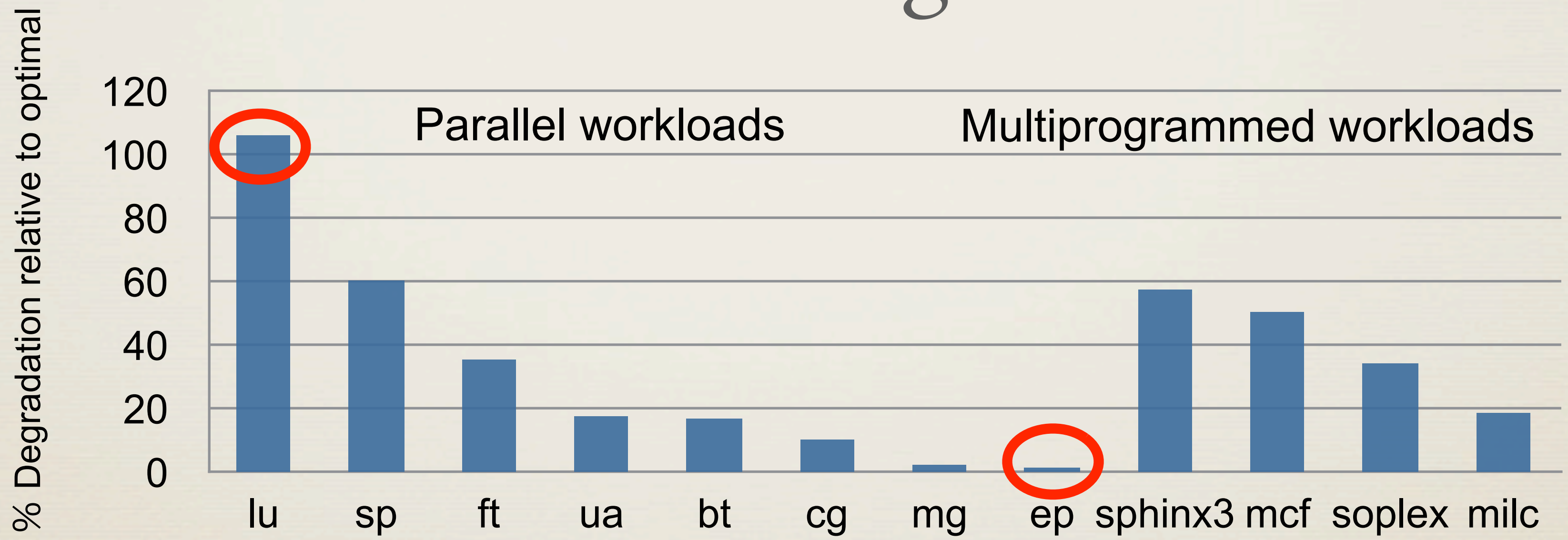
- Enumerate different thread-to-core assignments
  - 4 threads on two-socket Intel Westmere NUMA machine
- Calculate the worst to best degradation



scheduling plays an important role
for NUMA-sensitive workloads

Motivation

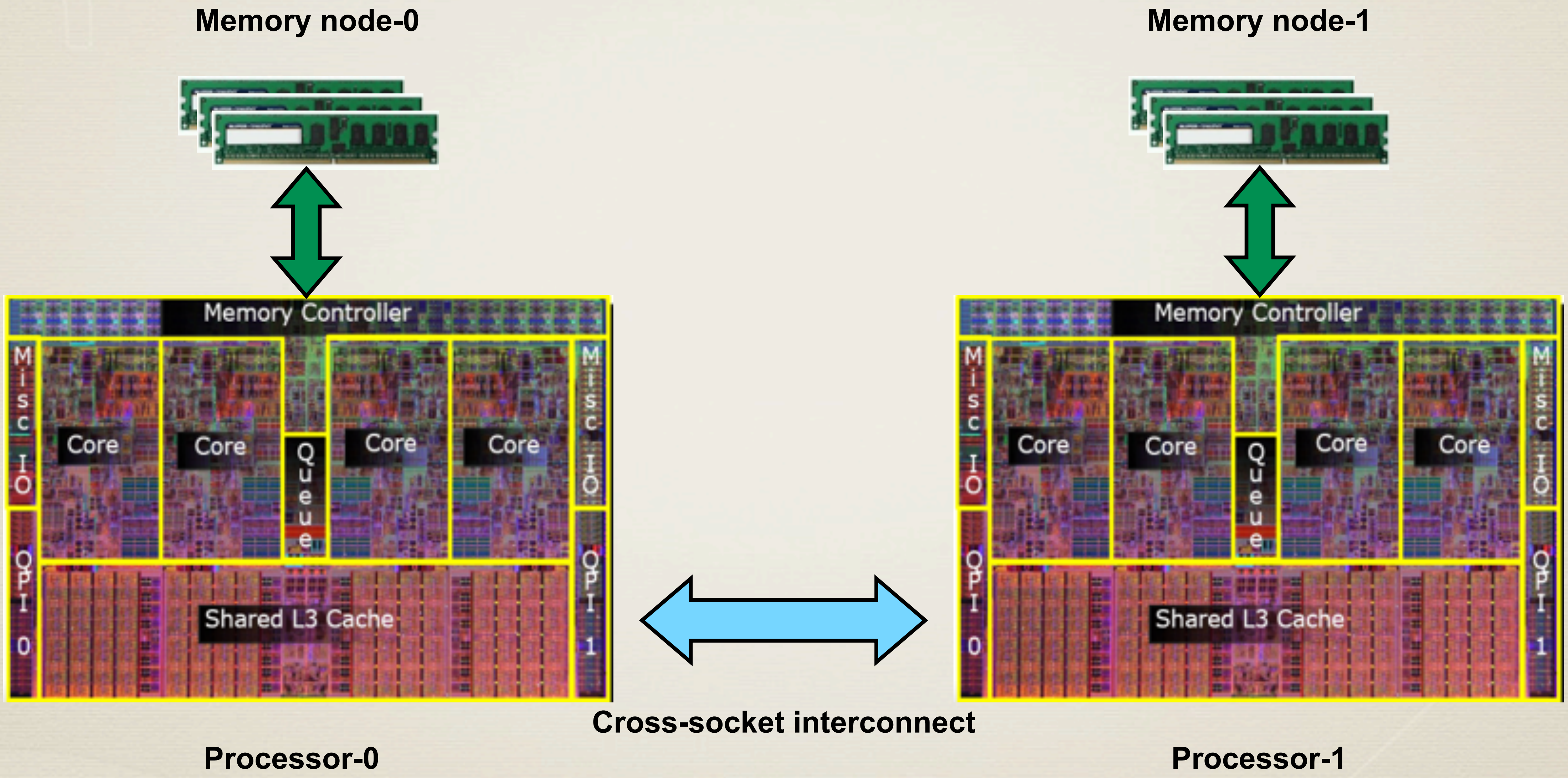
- Enumerate different thread-to-core assignments
  - 4 threads on two-socket Intel Westmere NUMA machine
- Calculate the worst to best degradation



scheduling plays an important role for NUMA-sensitive workloads

some workloads are NUMA insensitive

The NUMA Architecture



Memory node-0

Memory node-1

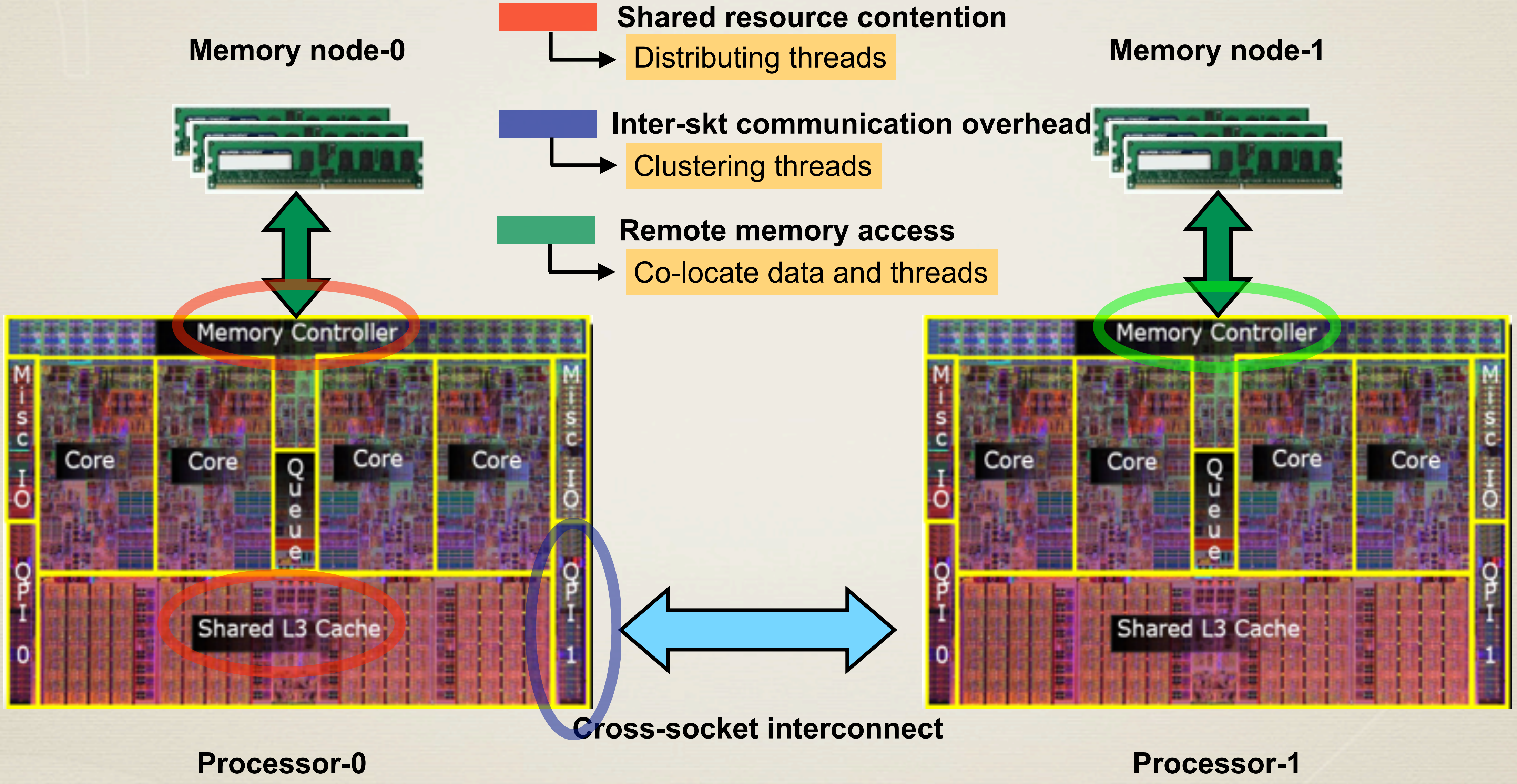
Cross-socket interconnect

Processor-0

Processor-1

Two-socket Intel Nehalem NUMA machine

The NUMA Architecture



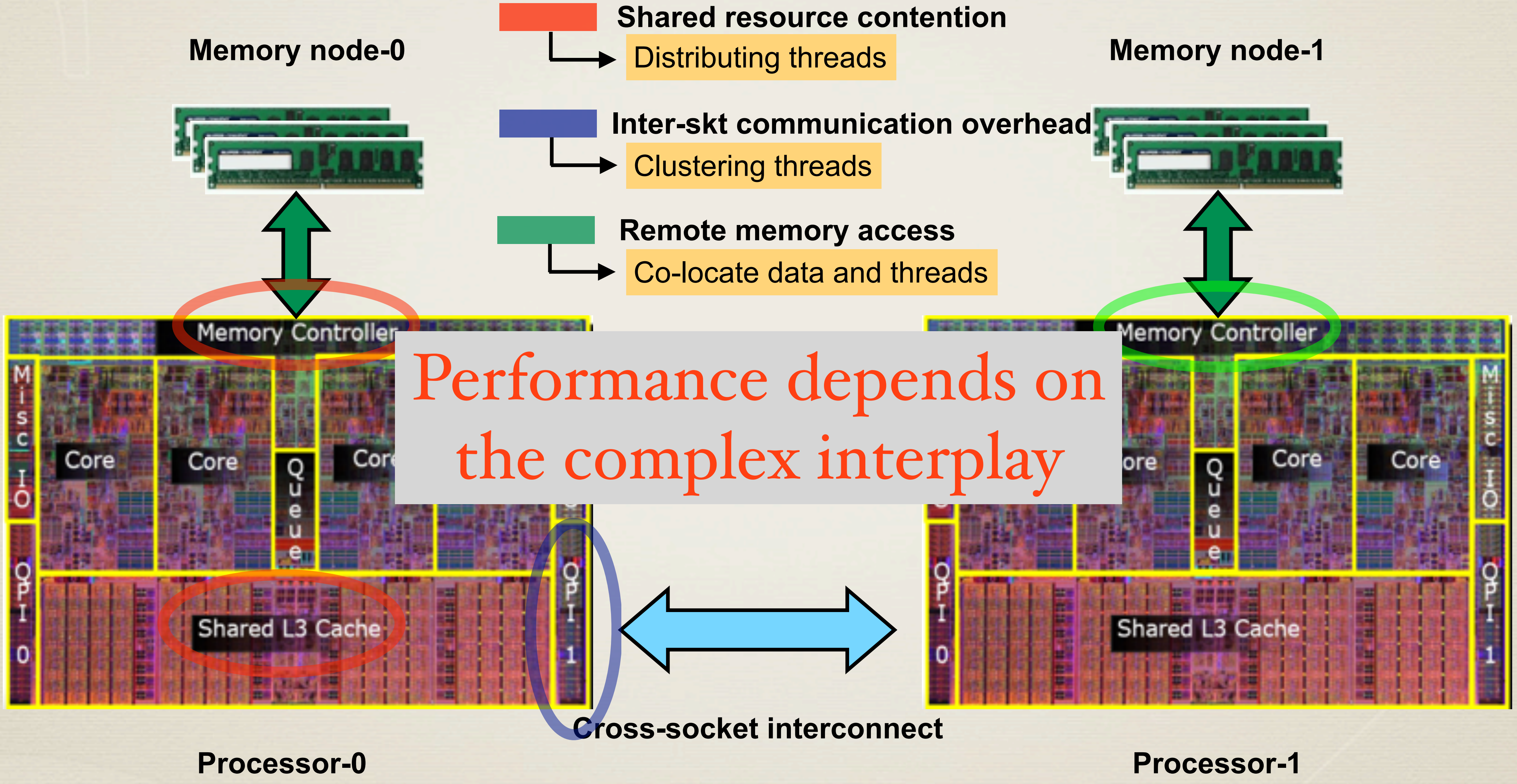
🟥 **Shared resource contention**
→ Distributing threads

🟦 **Inter-skt communication overhead**
→ Clustering threads

🟩 **Remote memory access**
→ Co-locate data and threads

Two-socket Intel Nehalem NUMA machine

The NUMA Architecture



🟥 **Shared resource contention**
→ Distributing threads

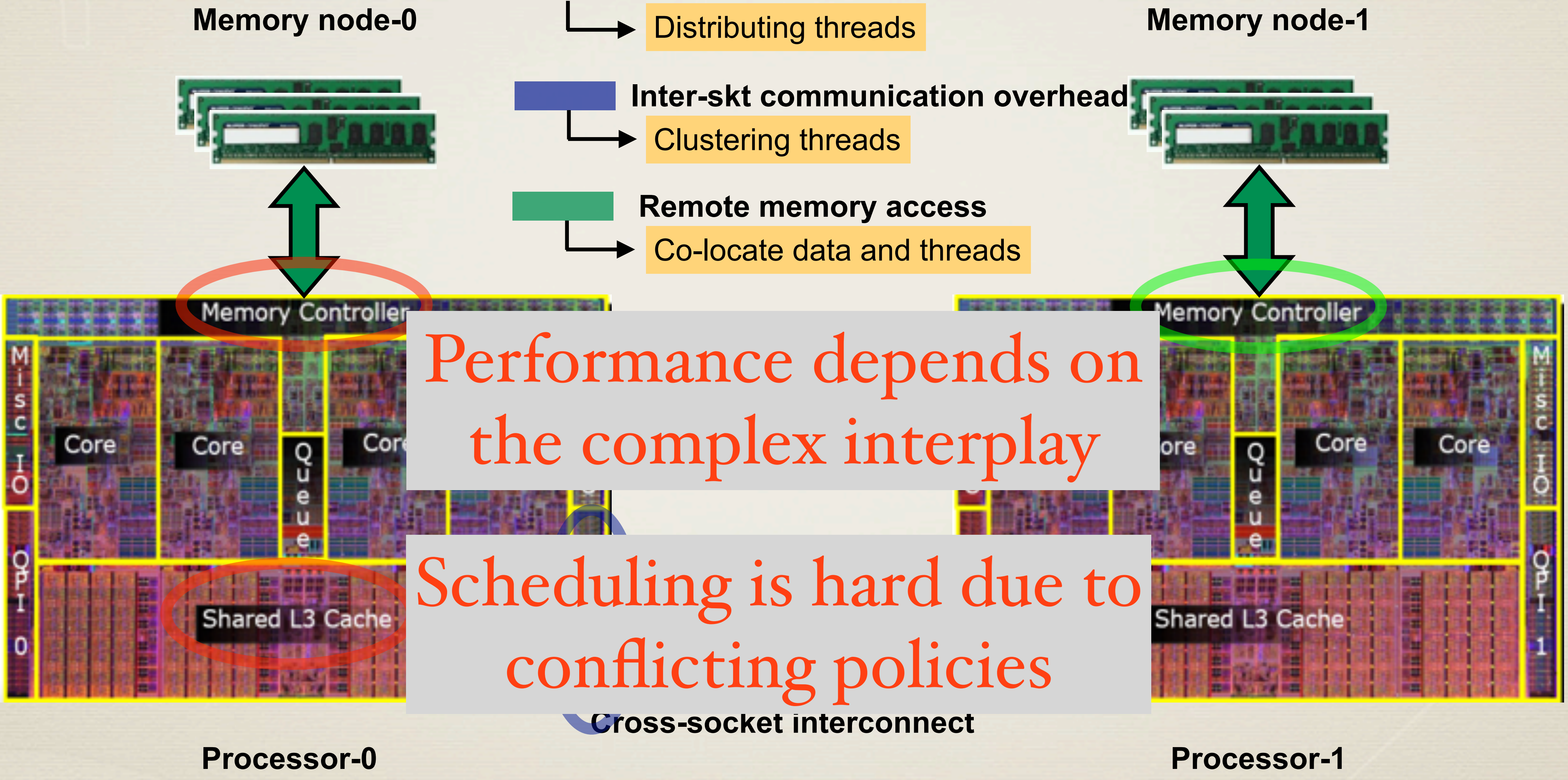
🟦 **Inter-skt communication overhead**
→ Clustering threads

🟩 **Remote memory access**
→ Co-locate data and threads

Performance depends on the complex interplay

Two-socket Intel Nehalem NUMA machine

The NUMA Architecture



🟥 **Shared resource contention**
→ Distributing threads

🟦 **Inter-skt communication overhead**
→ Clustering threads

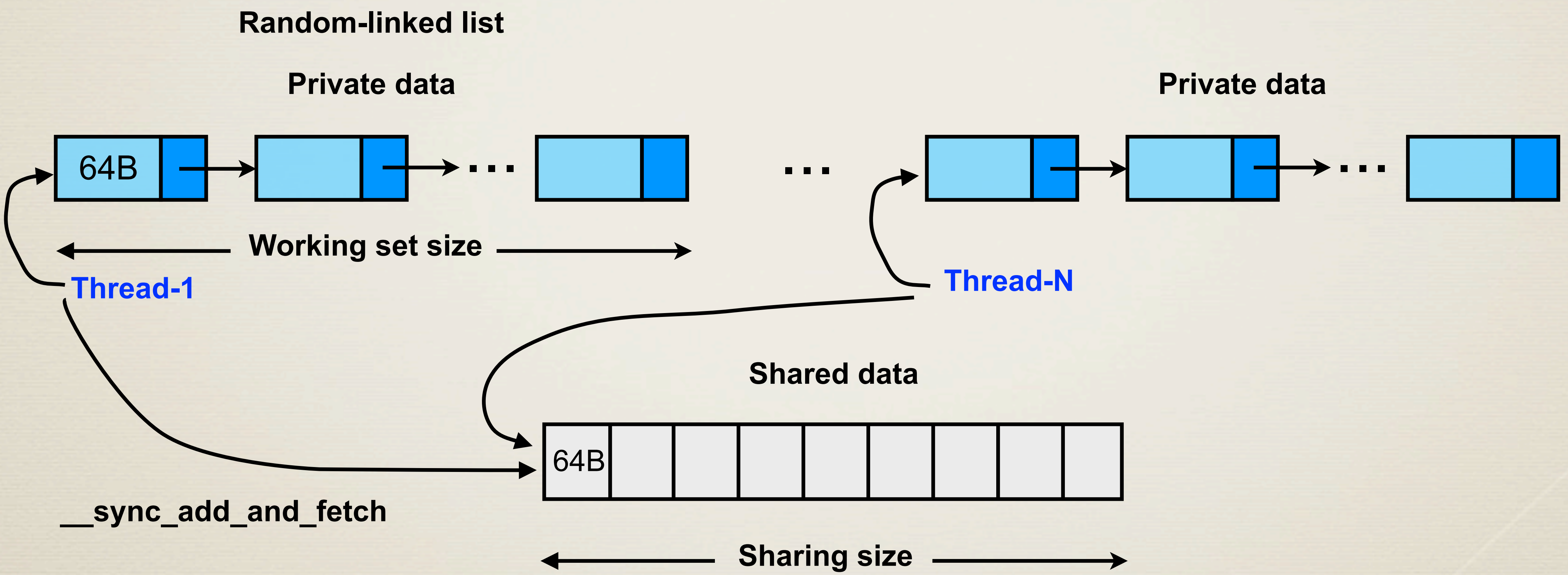
🟩 **Remote memory access**
→ Co-locate data and threads

Performance depends on the complex interplay

Scheduling is hard due to conflicting policies

Two-socket Intel Nehalem NUMA machine

Micro-benchmark



Random-linked list

Private data

Private data

64B

Working set size

Thread-1

Thread-N

Shared data

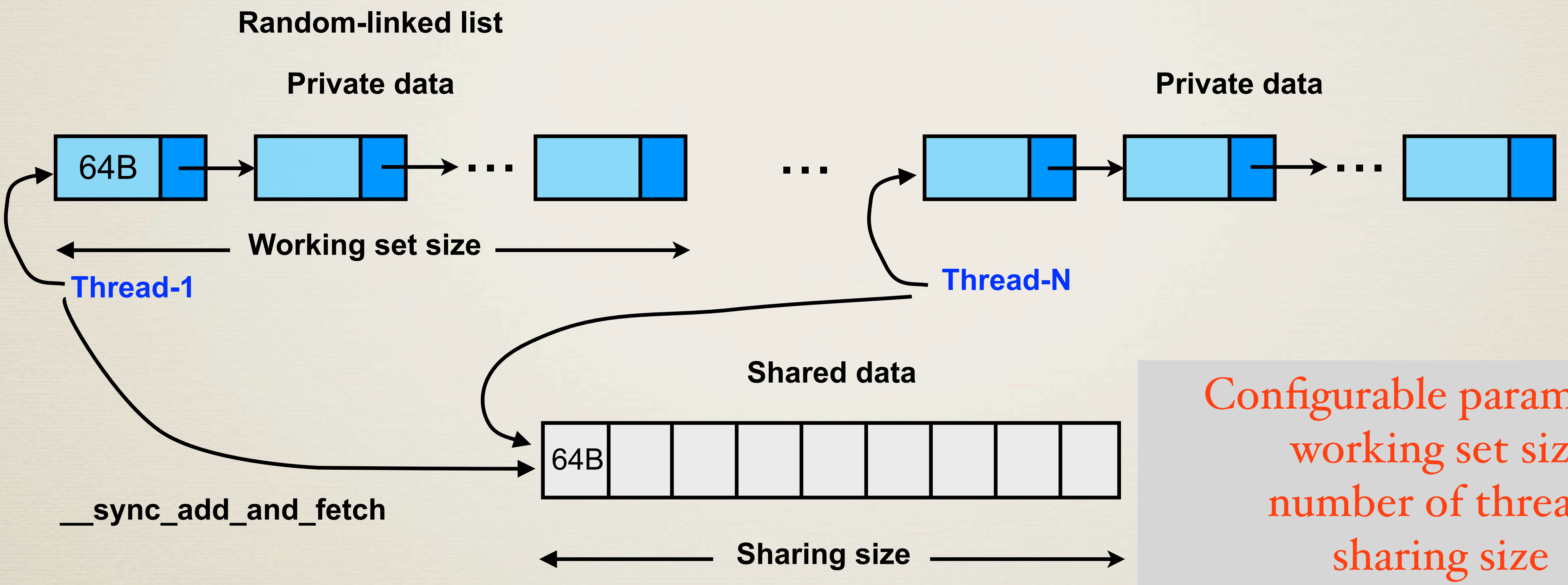
64B

__sync_add_and_fetch

Sharing size

University of Colorado
Colorado Springs

Micro-benchmark



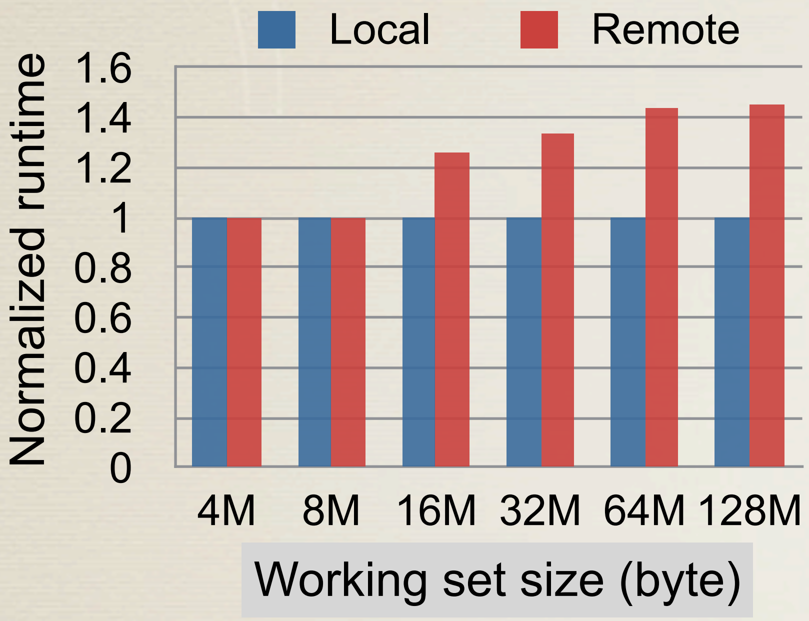
Configurable parameters:
working set size
number of threads
sharing size

UCCS University of Colorado Colorado Springs

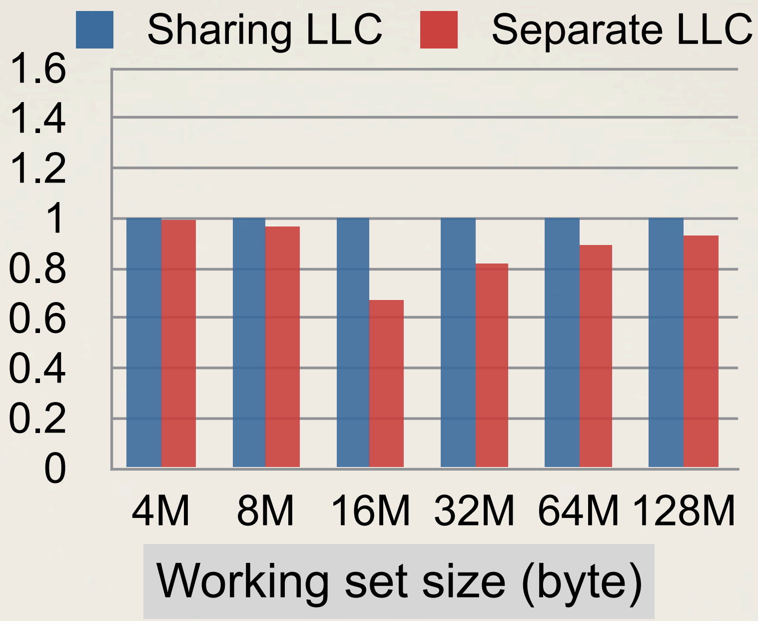
Experiment Testbed

	Intel Xeon E5620
Number of cores	4 cores (2 sockets)
Clock frequency	2.40 GHz
L1 Cache	32KB ICache, 32KB DCache
L2 Cache	256KB unified
L3 Cache	12MB unified, inclusive, shared by 4 cores
IMC	32GB/s bandwidth, 2 memory nodes, each with 8GB
QPI	5.86GT/s, 2 links

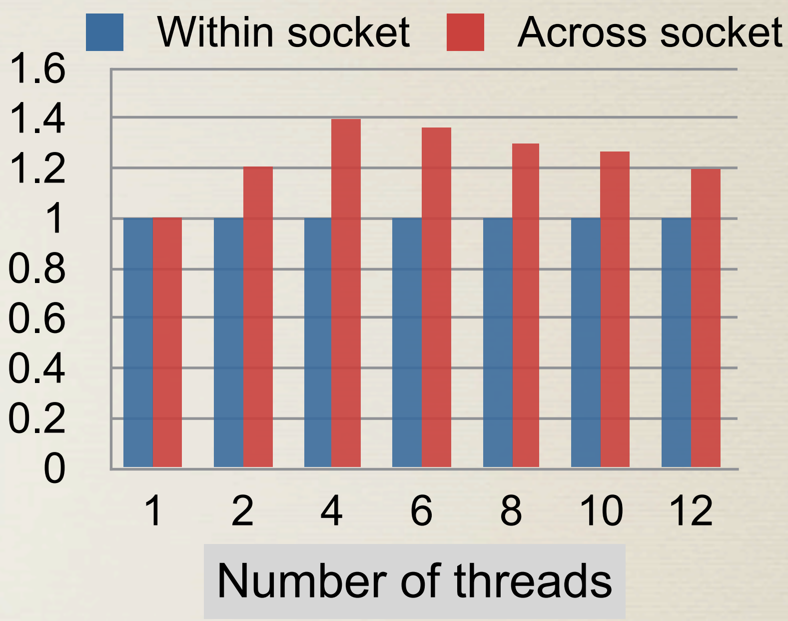
Single Factor on Performance


(a) Data locality

1 thread, sharing disabled

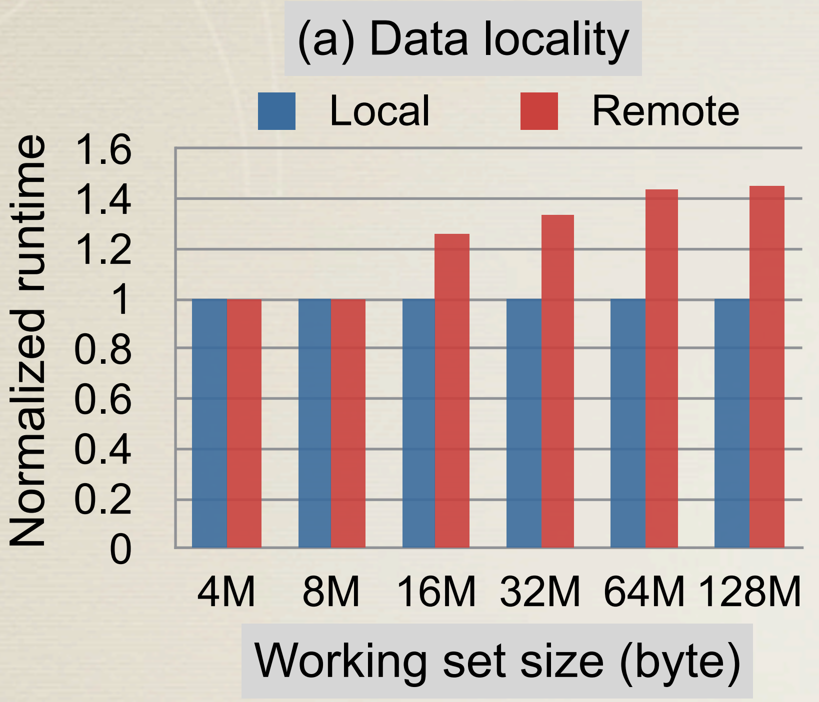

(b) LLC contention

4 thread, sharing disabled, co-located data with thread


(a) Sharing overhead

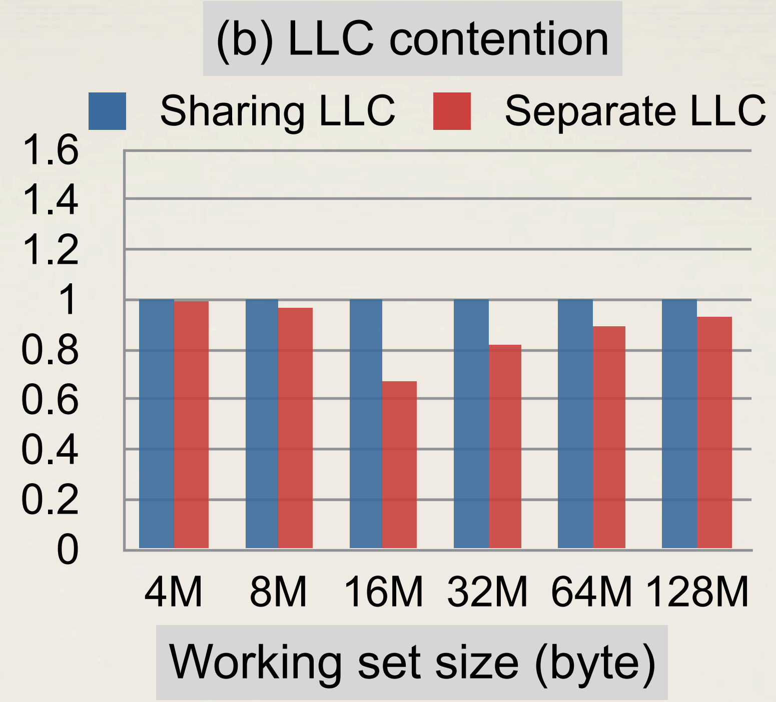
private data disabled, sharing 1 cacheline

Single Factor on Performance



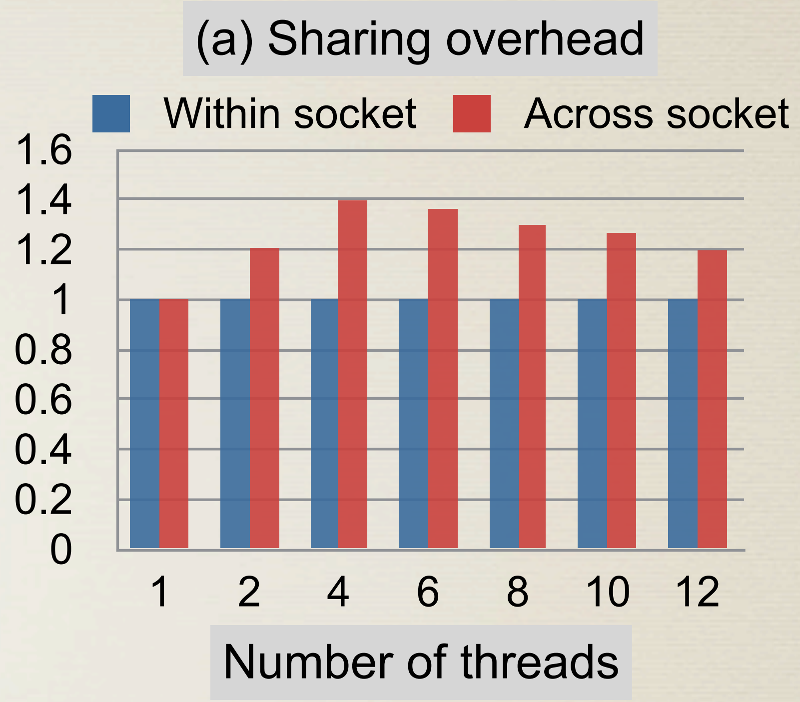
(a) Data locality

1 thread, sharing disabled



(b) LLC contention

4 thread, sharing disabled, co-located data with thread



(a) Sharing overhead

private data disabled, sharing 1 cacheline

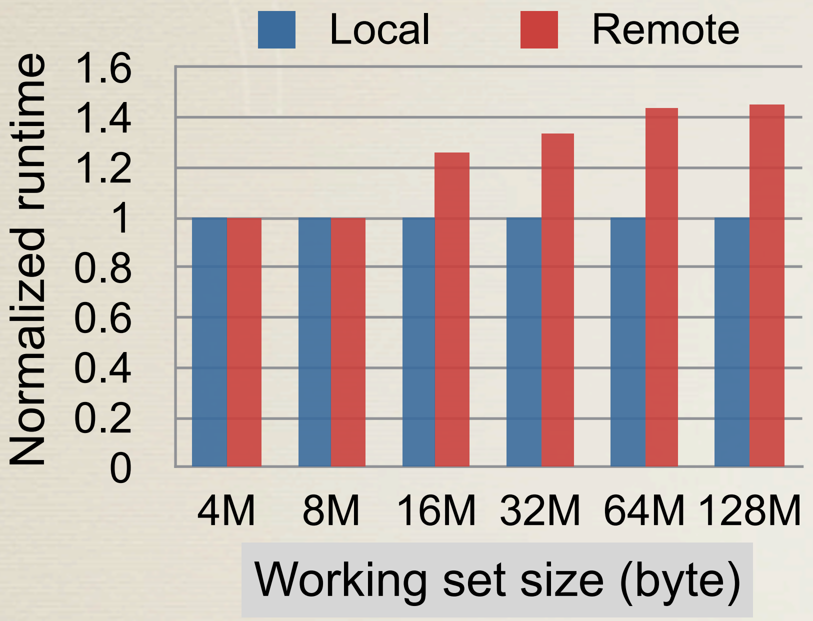
1. When WSS is smaller than LLC, remote access does not hurt performance

2. When WSS is beyond LCC capacity, the larger the WSS, the larger impact remote penalty hits performance

University of Colorado
Colorado Springs

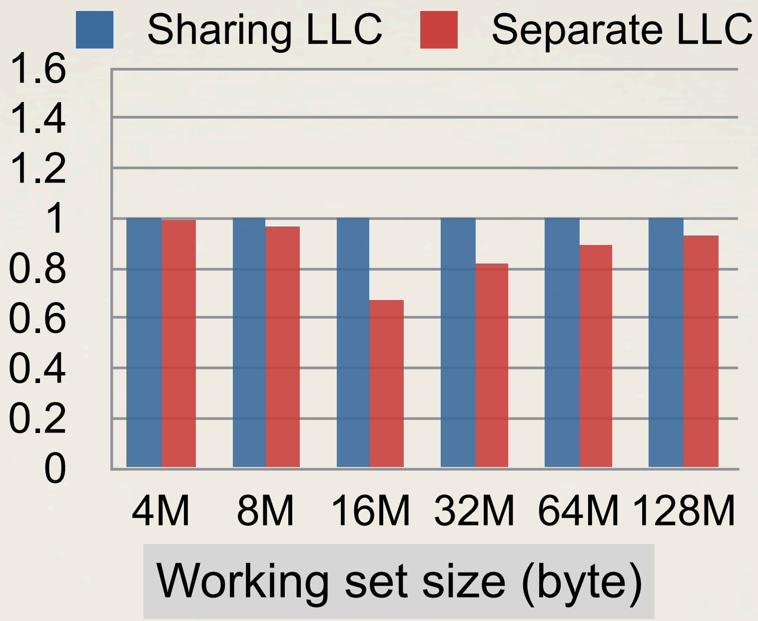
Single Factor on Performance

(a) Data locality



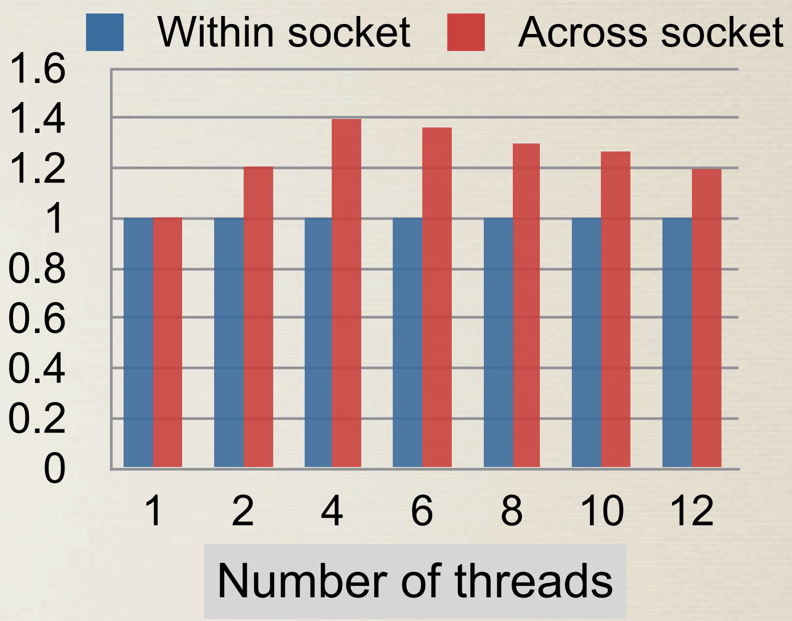
1 thread, sharing disabled

(b) LLC contention



4 thread, sharing disabled, co-located data with thread

(a) Sharing overhead



private data disabled, sharing 1 cacheline

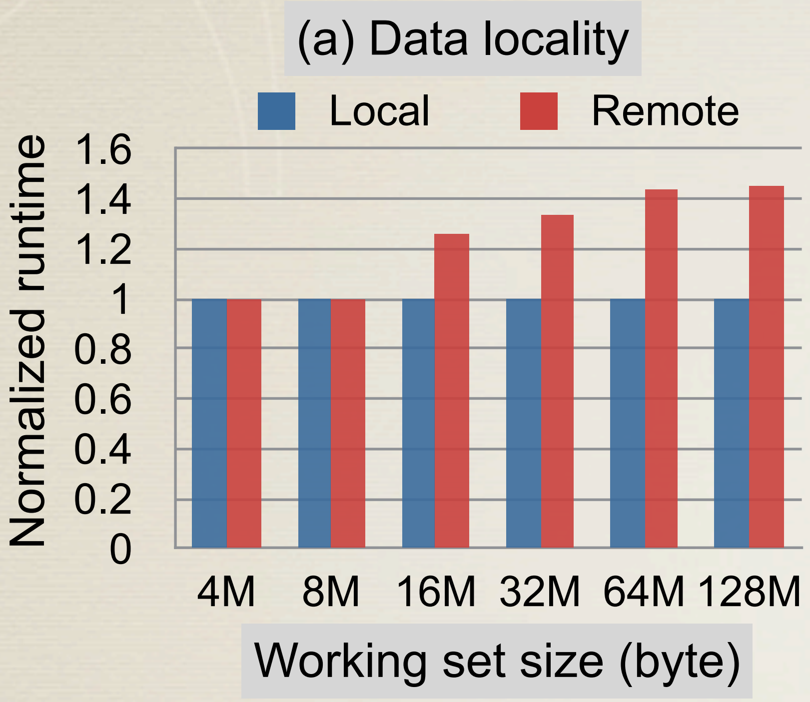
1. When WSS is smaller than LLC, remote access does not hurt performance

2. When WSS is beyond LCC capacity, the larger the WSS, the larger impact remote penalty hits performance

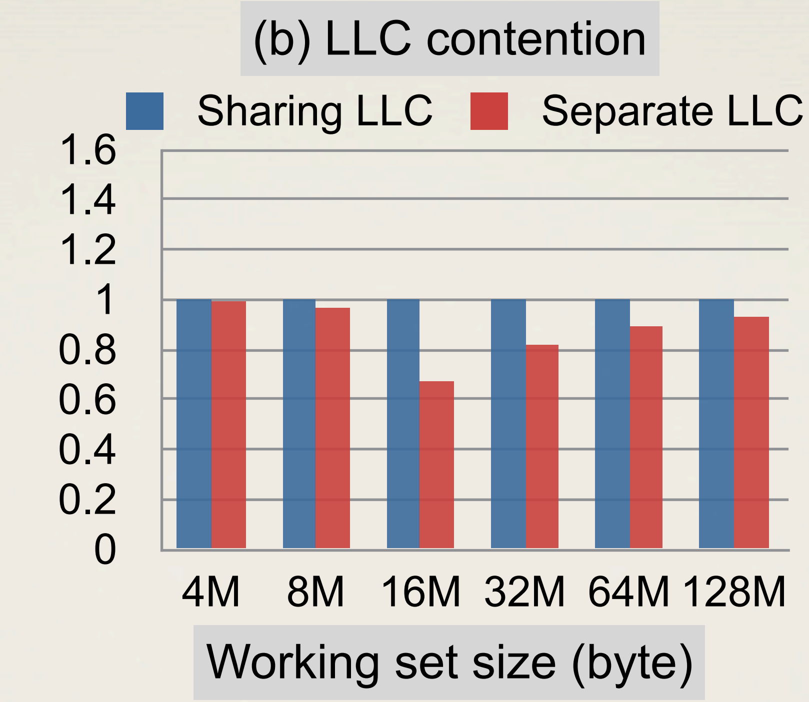
1. Scheduling does not affect performance when WSS fits in LLC

2. As WSS increases, LLC contention has diminishing effect on performance

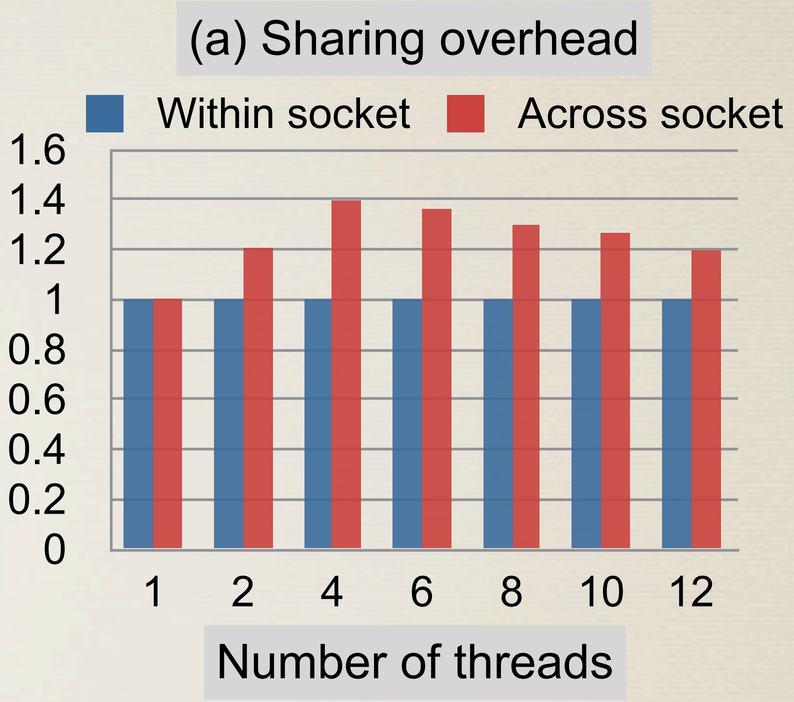
Single Factor on Performance


(a) Data locality

1 thread, sharing disabled


(b) LLC contention

4 thread, sharing disabled, co-located data with thread


(a) Sharing overhead

private data disabled, sharing 1 cacheline

1. When WSS is smaller than LLC, remote access does not hurt performance

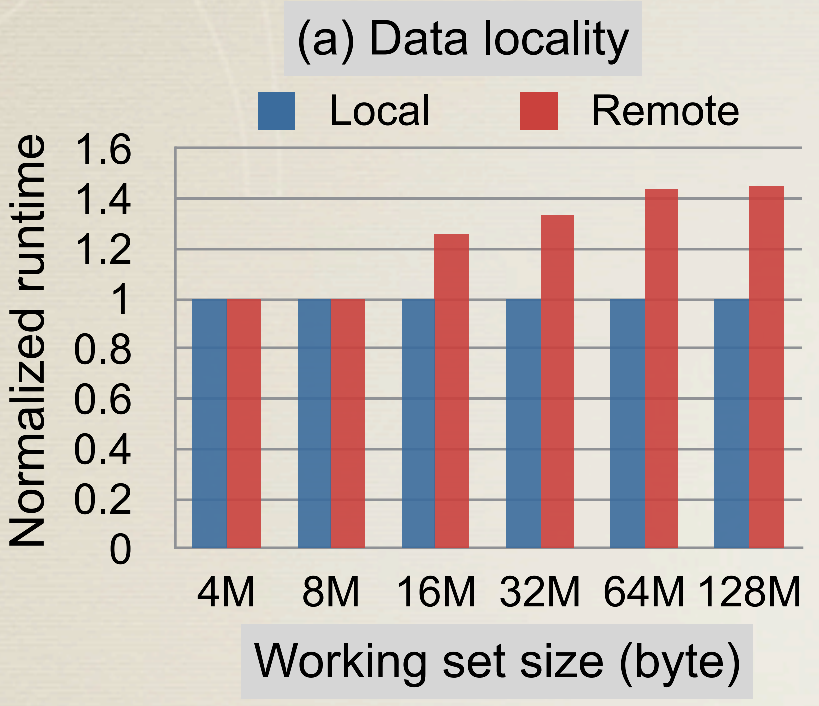
2. When WSS is beyond LCC capacity, the larger the WSS, the larger impact remote penalty hits performance

1. Scheduling does not affect performance when WSS fits in LLC

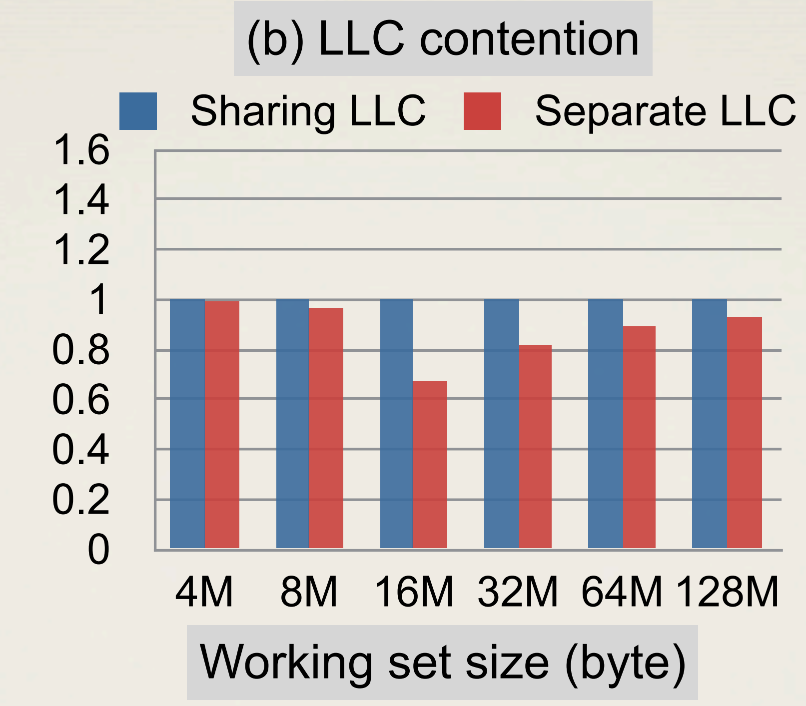
2. As WSS increases, LLC contention has diminishing effect on performance

1. Inter-socket sharing overhead increases initially but decreases as more threads are used

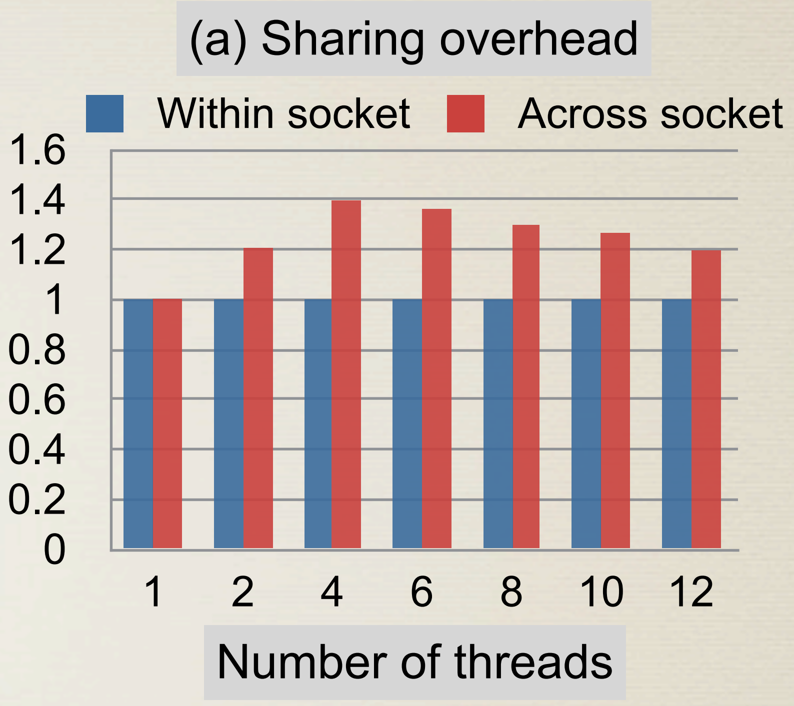
Single Factor on Performance



1 thread, sharing disabled



4 thread, sharing disabled, co-located data with thread



private data disabled, sharing 1 cacheline

1. When WSS is smaller than LLC, remote access does not hurt performance

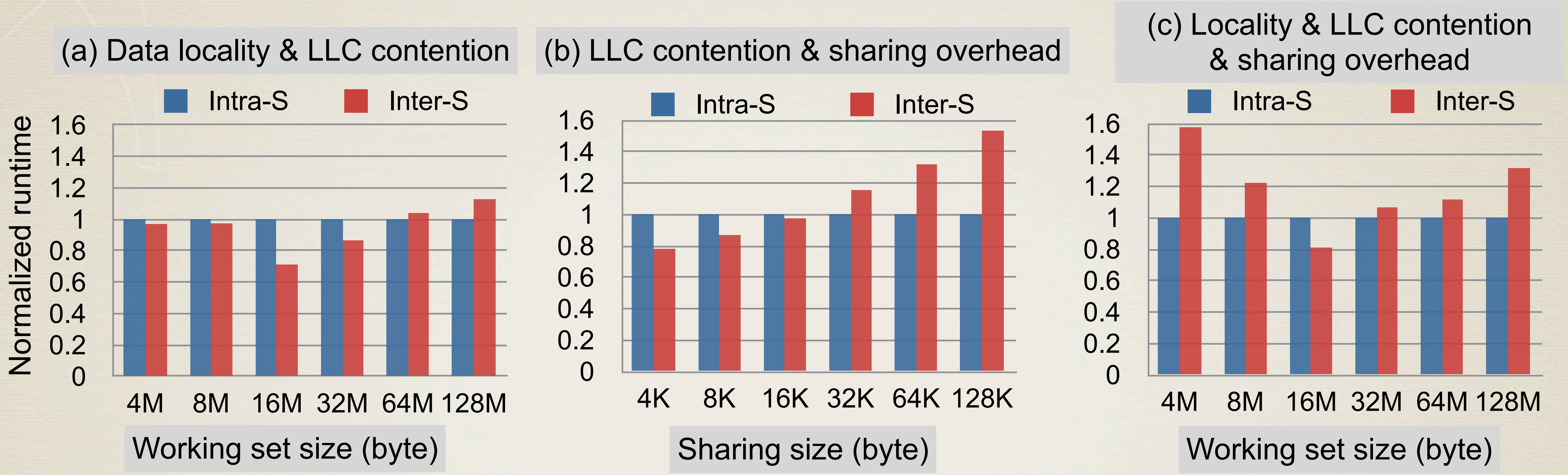
2. When WSS is beyond LCC capacity, the larger the WSS, the larger impact remote penalty hits performance

Memory footprint, thread-level parallelism, and inter-thread sharing pattern determine how much each factor affects performance

1. Inter-socket sharing overhead increases initially but decreases as more threads are used

University of Colorado
Colorado Springs

Multiple Factors on Performance


(a) Data locality & LLC contention

(b) LLC contention & sharing overhead

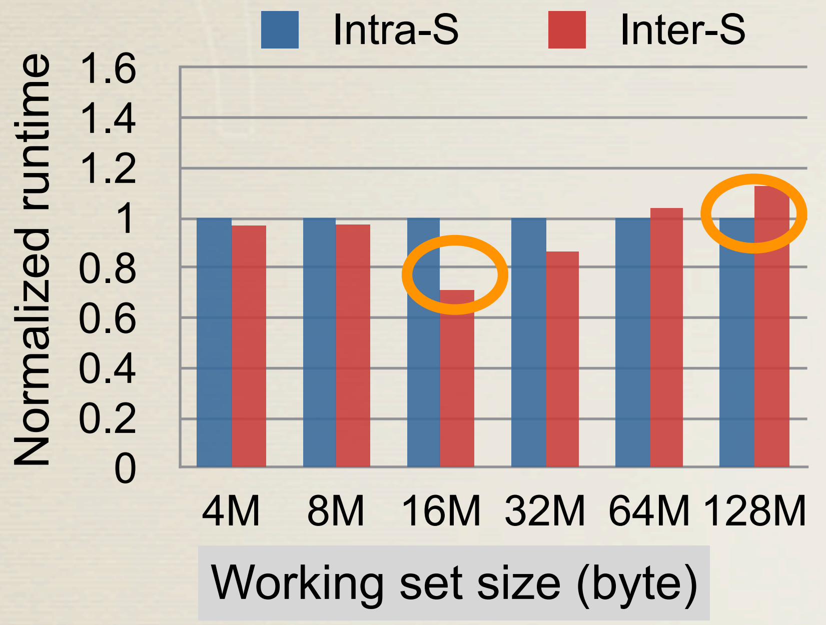
(c) Locality & LLC contention & sharing overhead

🟥 Intra-S: clustering threads
on node-0

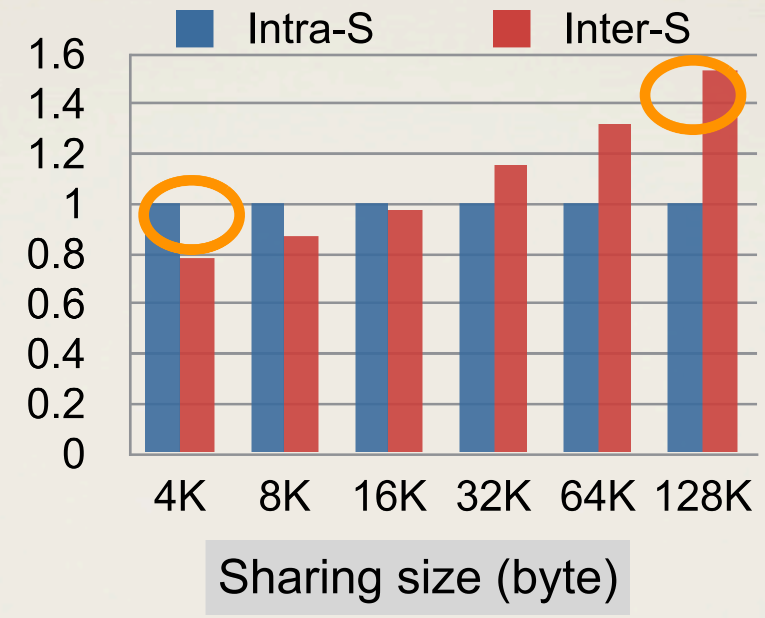
🟦 Inter-S: distributing threads

Multiple Factors on Performance

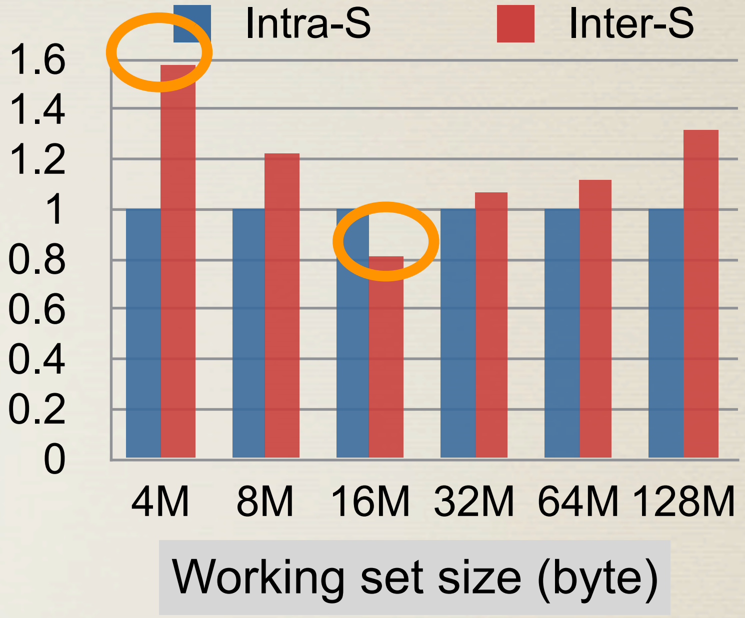
(a) Data locality & LLC contention



(b) LLC contention & sharing overhead



(c) Locality & LLC contention & sharing overhead



🟥 Intra-S: clustering threads on node-0

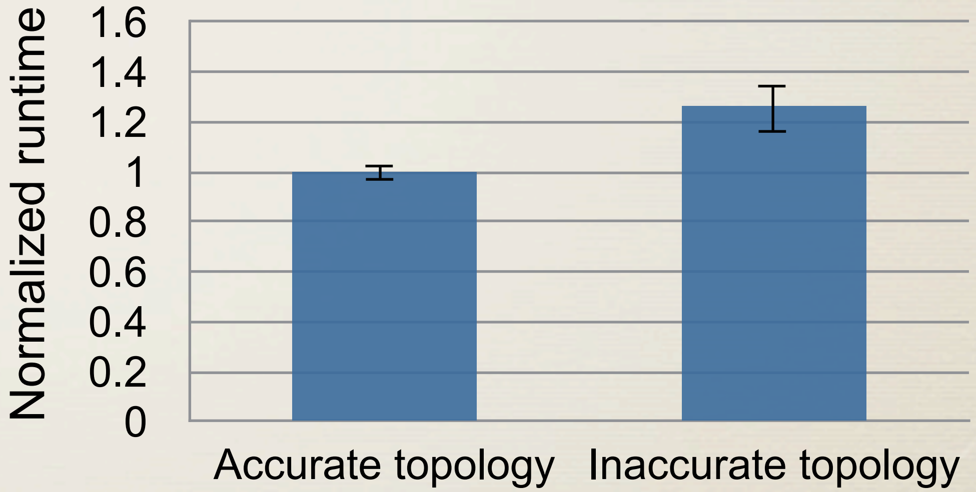
🟦 Inter-S: distributing threads

1. Dominant factor determines performance

2. Dominant factor switches as program characteristic changes

Virtualization

- Application and guest OS see virtual topology
- Virtual topology ≠ physical topology
  - flat topology
  - inaccurate topology
    - Static Resource Affinity Table (SRAT)
    - inaccurate due to load balancing



`set_mempolicy` in libnuma

4 threads, 32MB WSS, 128KB sharing size

University of Colorado
Colorado Springs

Related Work

- Optimization via scheduling
  - Contention management: [TCS'10], [SIGMETRICS'11], [ISCA'11], [ASPLOS'10]
  - Thread clustering: [EuroSys'07]
  - NUMA management: [ATC'11], [ASPLOS'13]
- Program and system-level optimizations
  - Program transformation: [PPoPP'10], [CGO'12]
  - System support: libnuma, page replication and migration

Related Work

- Optimization via scheduling
  - Contention management: [TCS'10], [SIGMETRICS'11], [ISCA'11], [ASPLOS'10]
  - T
  - N
- Pr
  - F
  - System support: libnuma, page replication and migration

Our work:

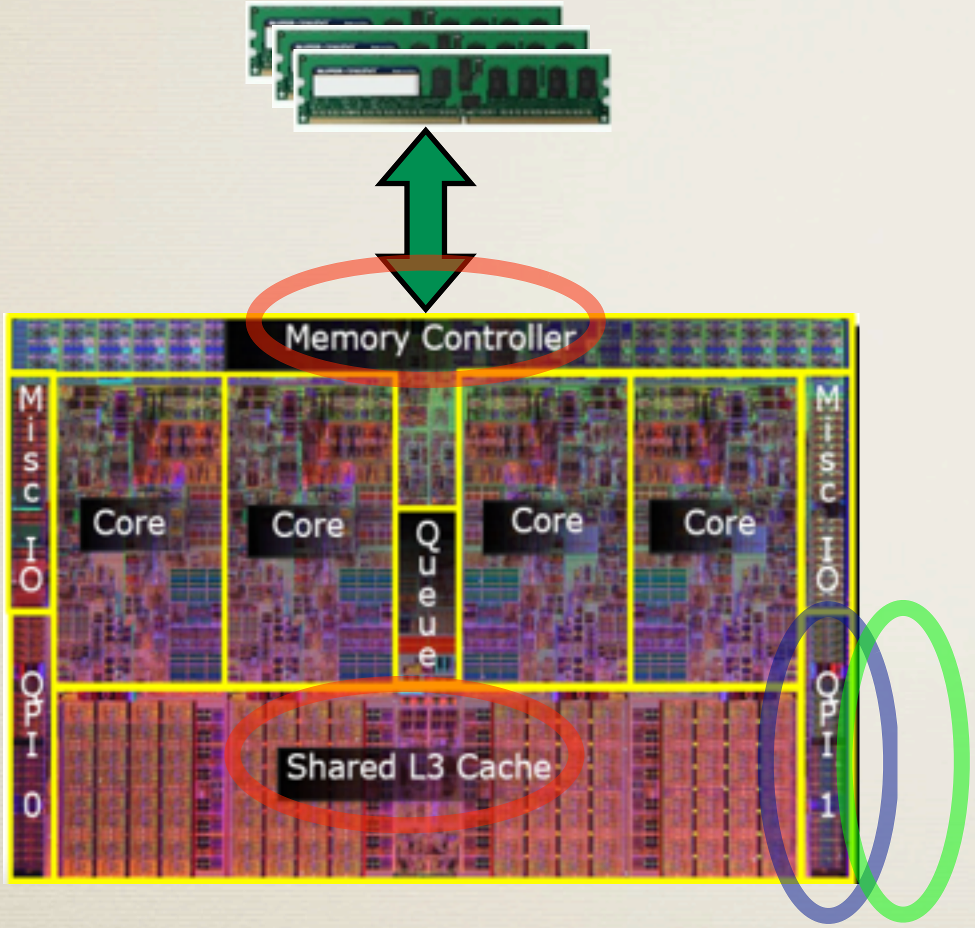
1. requiring no offline profiling, online

2. addressing complex interplays

3. assuming no knowledge on virtual topology

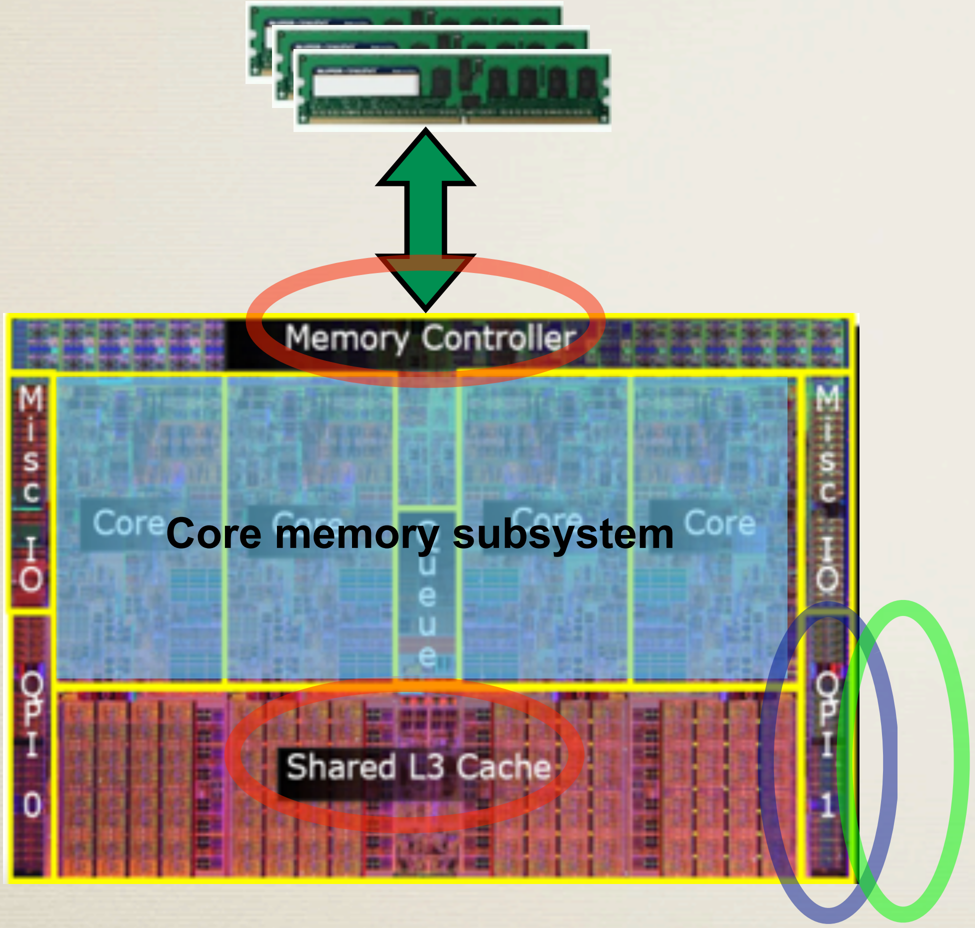
Uncore Penalty as a Performance Index

🟥 Shared resource contention
🟩 Remote memory access
🟦 Inter-skt communication overhead



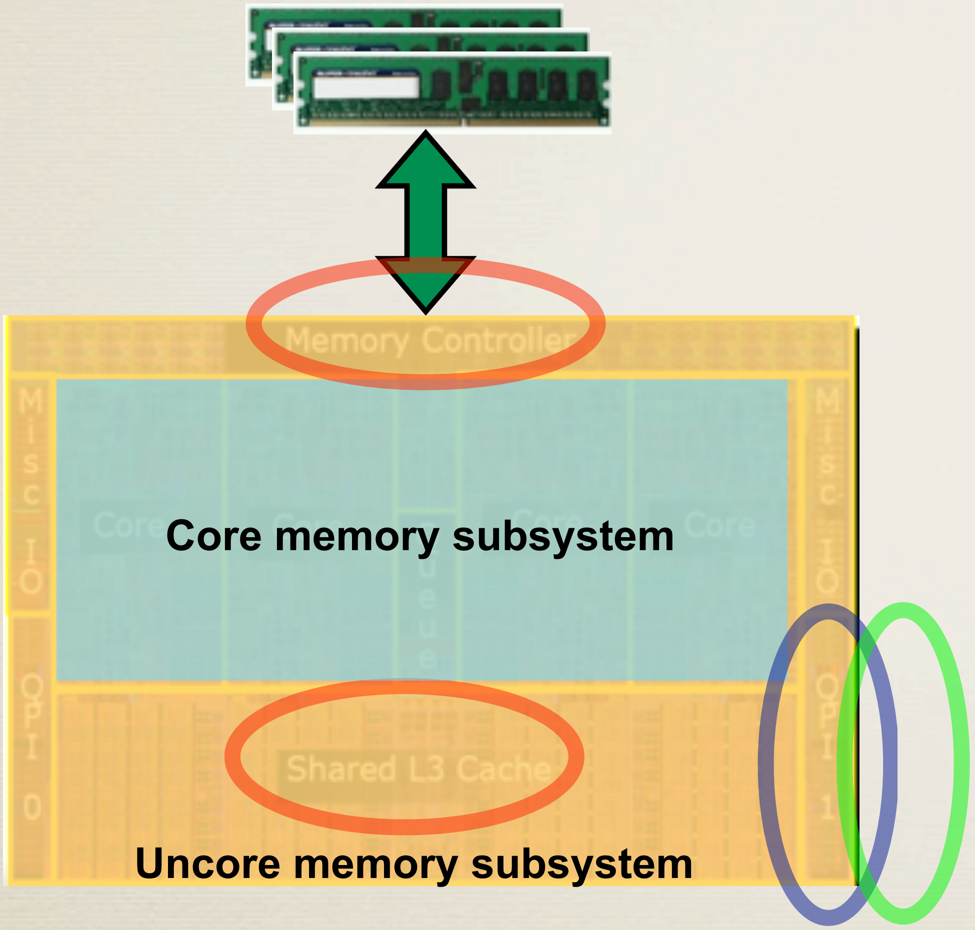
Uncore Penalty as a Performance Index

🟥 Shared resource contention
🟩 Remote memory access
🟦 Inter-skt communication overhead



Uncore Penalty as a Performance Index

🟥 Shared resource contention
🟩 Remote memory access
🟦 Inter-skt communication overhead



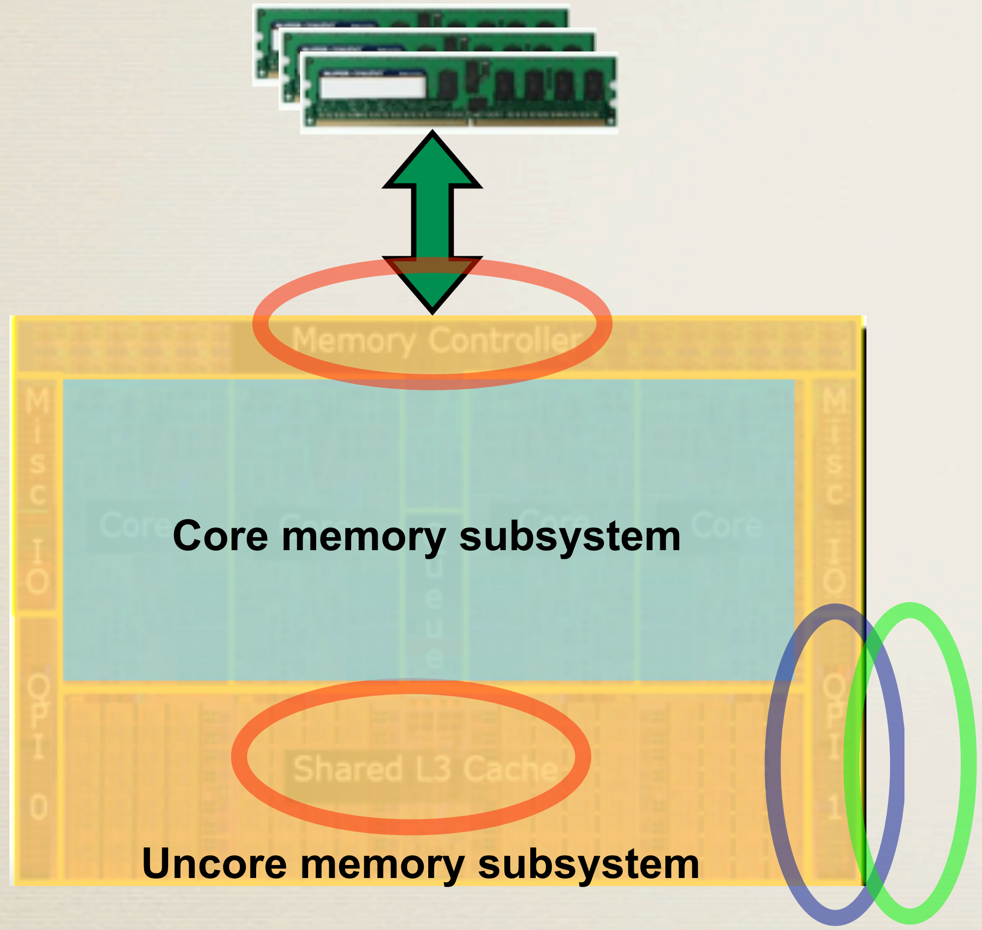
Core memory subsystem

Uncore memory subsystem

Uncore Penalty as a Performance Index

🟥 Shared resource contention
🟩 Remote memory access
🟦 Inter-skt communication overhead

→ Uncore stall cycles ↑



Core memory subsystem

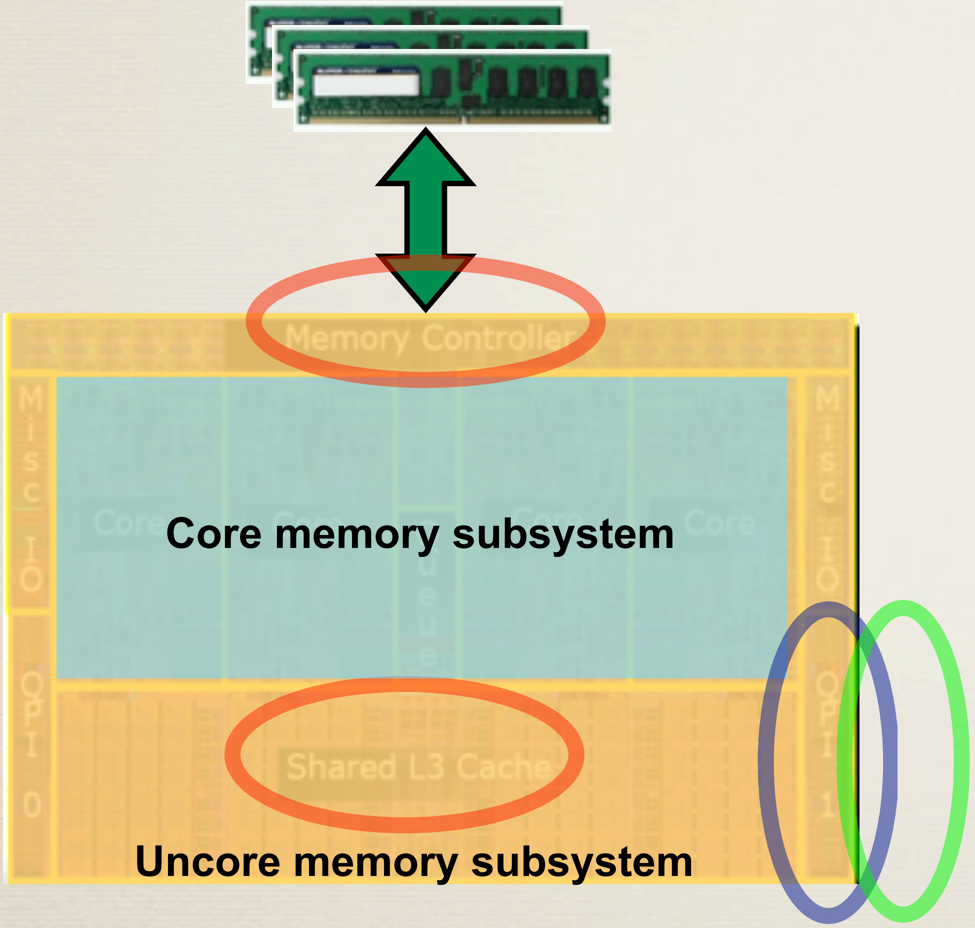
Uncore memory subsystem

Uncore Penalty as a Performance Index

🟥 Shared resource contention
🟩 Remote memory access
🟦 Inter-skt communication overhead

→ Uncore stall cycles ↑

$$\text{Uncore stall cycles} = \overset{\overset{\text{constant}}{\uparrow}}{\text{Number of L2 miss}} \times \overset{\overset{\text{uncore penalty}}{\uparrow}}{\text{L2 miss penalty}}$$



Core memory subsystem

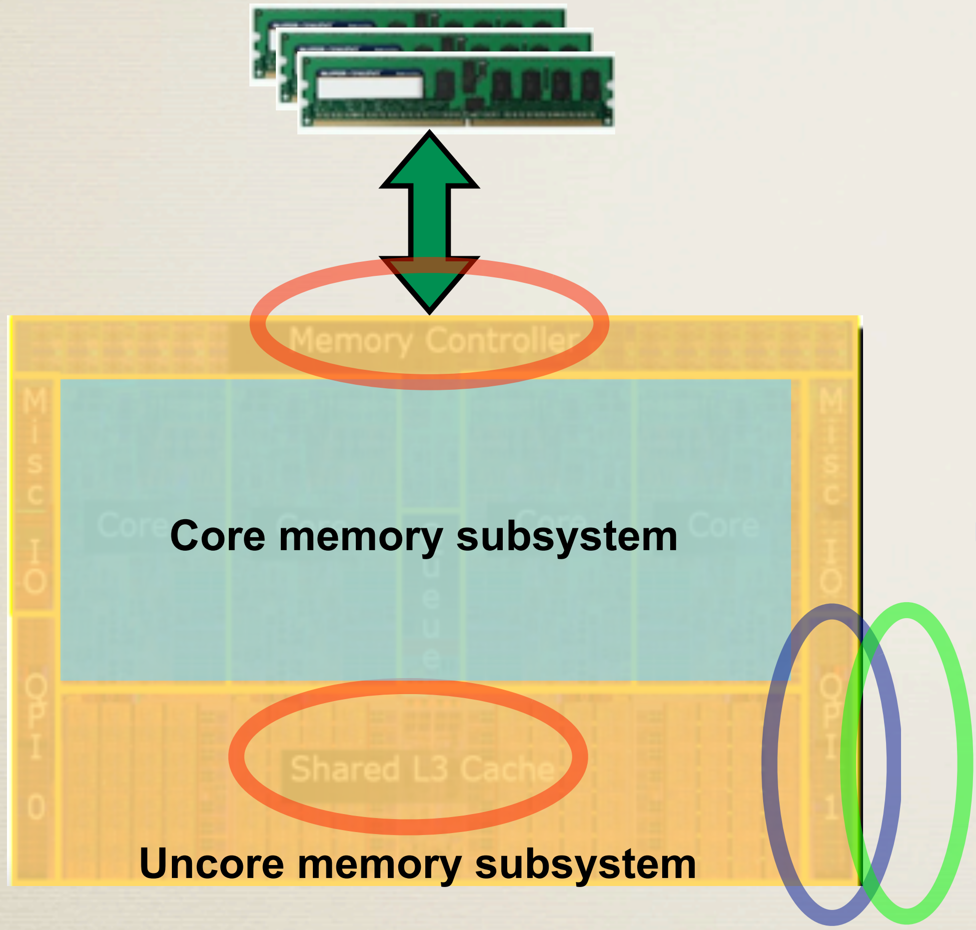
Uncore memory subsystem

Uncore Penalty as a Performance Index

🟥 Shared resource contention
🟩 Remote memory access
🟦 Inter-skt communication overhead

→ Uncore stall cycles ↑

$$Uncore\ stall\ cycles = \overset{\substack{\text{constant}\\ \uparrow}}{Number\ of\ L2\ miss} \times \overset{\substack{\text{uncore penalty}\\ \uparrow}}{L2\ miss\ penalty}$$



Little's law

$$\overset{\text{uncore latency} \nearrow}{W} = \frac{L}{\lambda}$$

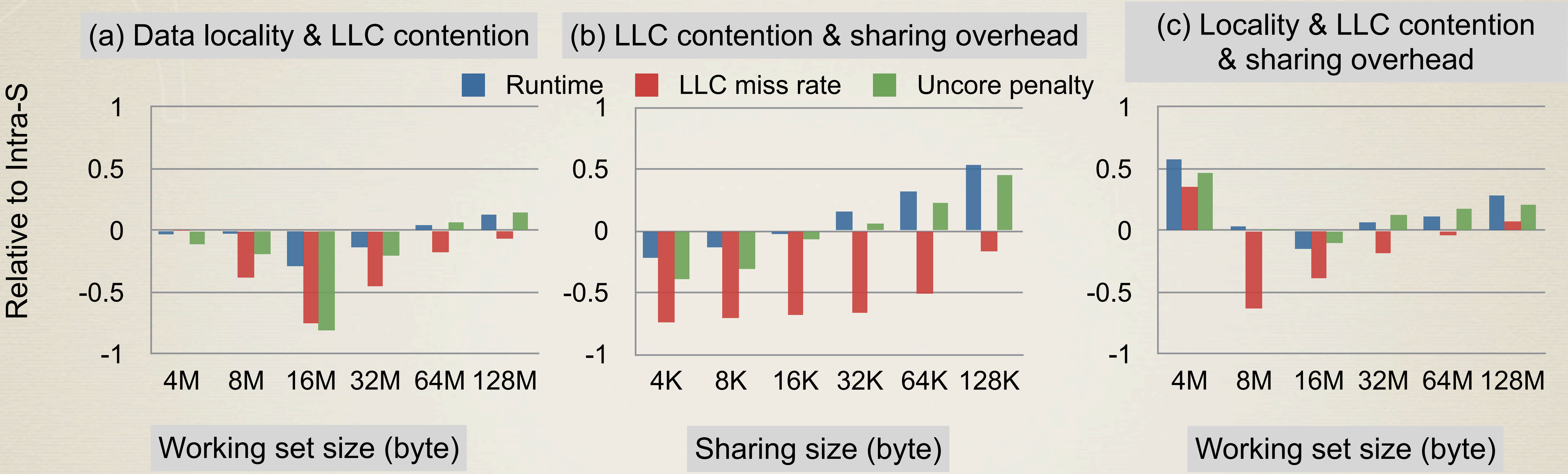
cycles at least one "critical" uncore rqst

$$\underset{\nearrow\ \text{uncore penalty}}{P} = \frac{W}{\underset{\substack{\uparrow\\ \text{outstanding parallel uncore rqst}}}{q}} = \frac{\frac{L}{\lambda}}{\frac{L}{t'}} = \frac{t'}{\lambda}$$

"critical" uncore rqst = demand data/instruction load + RFO rqst

UCCS University of Colorado Colorado Springs

Effectiveness of Uncore Penalty Metric


(a) Data locality & LLC contention

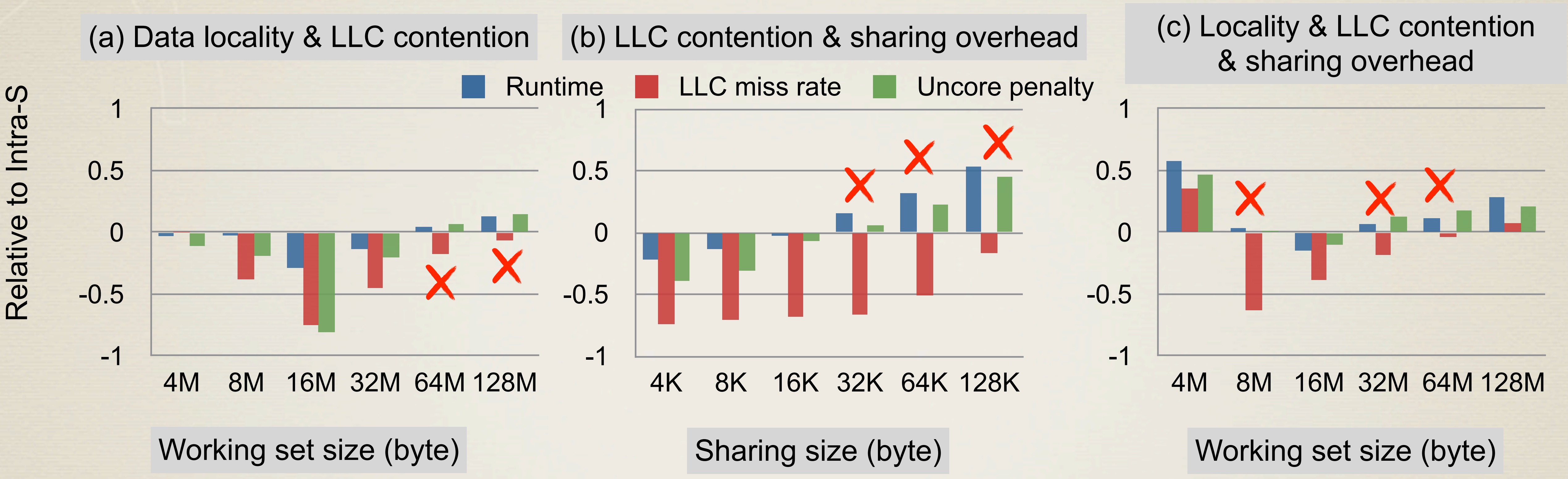
(b) LLC contention & sharing overhead

(c) Locality & LLC contention & sharing overhead

Relative diff

$$\frac{V_{Inter\text{-}S} - V_{Intra\text{-}S}}{V_{Intra\text{-}S}}$$

Effectiveness of Uncore Penalty Metric

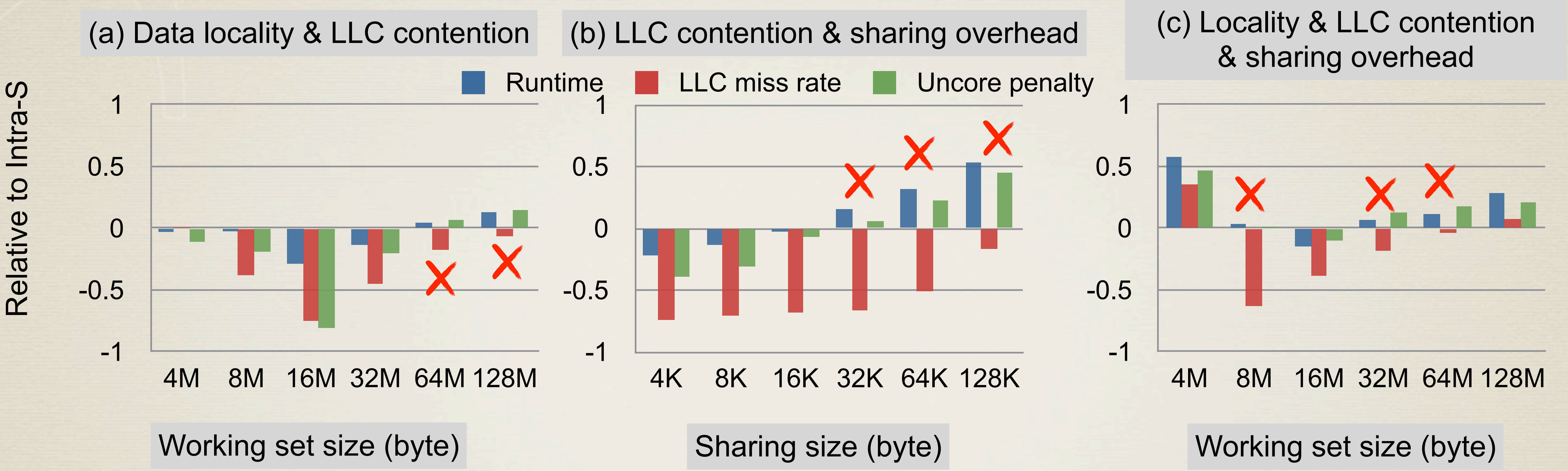


Relative diff

$$\frac{V_{Inter\text{-}S} - V_{Intra\text{-}S}}{V_{Intra\text{-}S}}$$

- LLC miss rate only agrees with runtime in a subset of runs

Effectiveness of Uncore Penalty Metric



Relative diff

$$\frac{V_{Inter\text{-}S} - V_{Intra\text{-}S}}{V_{Intra\text{-}S}}$$

- LLC miss rate only agrees with runtime in a subset of runs
- Strong linear relationship between uncore penalty and runtime

Linear correlation coefficient *r* (against runtime)

Uncore penalty	LLC miss rate
r = 0.91	r = 0.61

University of Colorado
Colorado Springs

NUMA-aware Virtual Machine Scheduling

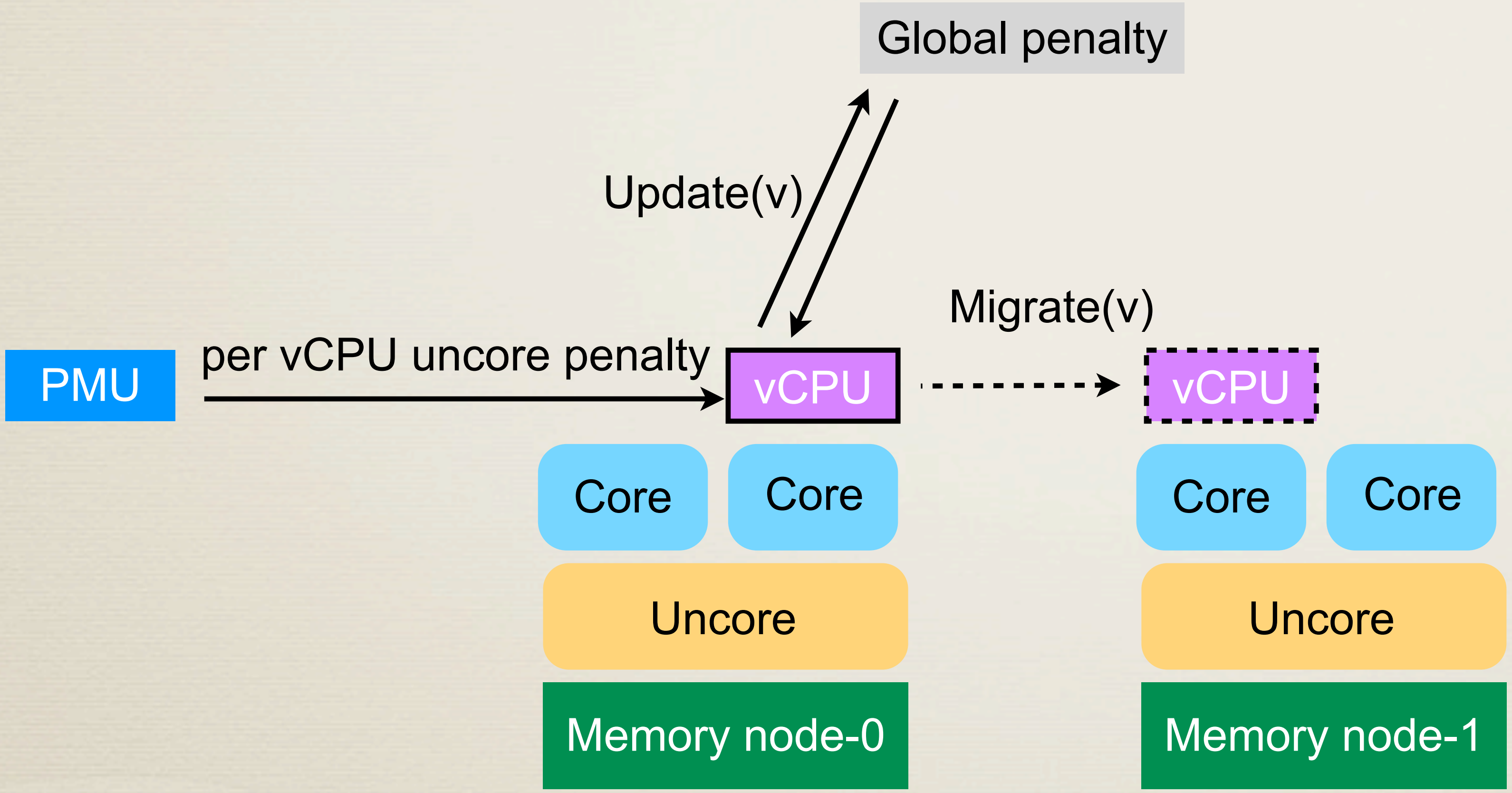
- Monitoring uncore penalty
  - calculate each vCPU’s penalty based on PMU readings
  - update penalty when performing periodic scheduler bookkeeping
- Identifying NUMA scheduling candidate
  - rely on application or guest OS to identify NUMA-sensitive vCPUs
- vCPU migration
  - Bias Random Migration (BRM)

Bias Random Migration

```
struct vcpu {
...
int c;  //current core
int bias; // migration bias
int prob; //migration probability
int unc[NODE]; //array of global penalties
...
}
```

```
Procedure Migrate(v)
    Update(v)
    if v is a migration candidate then
        new_core = BiasRandomPick(v)
    else
        new_core = DefaultPick(v)
    end if
end procedure
```

```
Procedure Update(v)
    n = NUMA_CPU_TO_NODE(v.c)
    update global penalty and save it in v.unc[n]
    min = $\underset{x}{\text{argmin}}$ v.unc[x], x= 0,..., NODE-1
    if v.unc[n] > v.unc[min] then
        v.prob ++
    else
        v.prob --
        v.bias = n
    end if
end procedure
```



Global penalty

Update(v)

Migrate(v)

PMU

per vCPU uncore penalty

vCPU

vCPU

Core

Core

Core

Core

Uncore

Uncore

Memory node-0

Memory node-1

```
Procedure BiasRandomPick(v)
    rand = random() mod 100
        if rand < v.prob then
            select a core in node v.bias
            reset v.prob
        else
            Select current core
        end if
end procedure
```

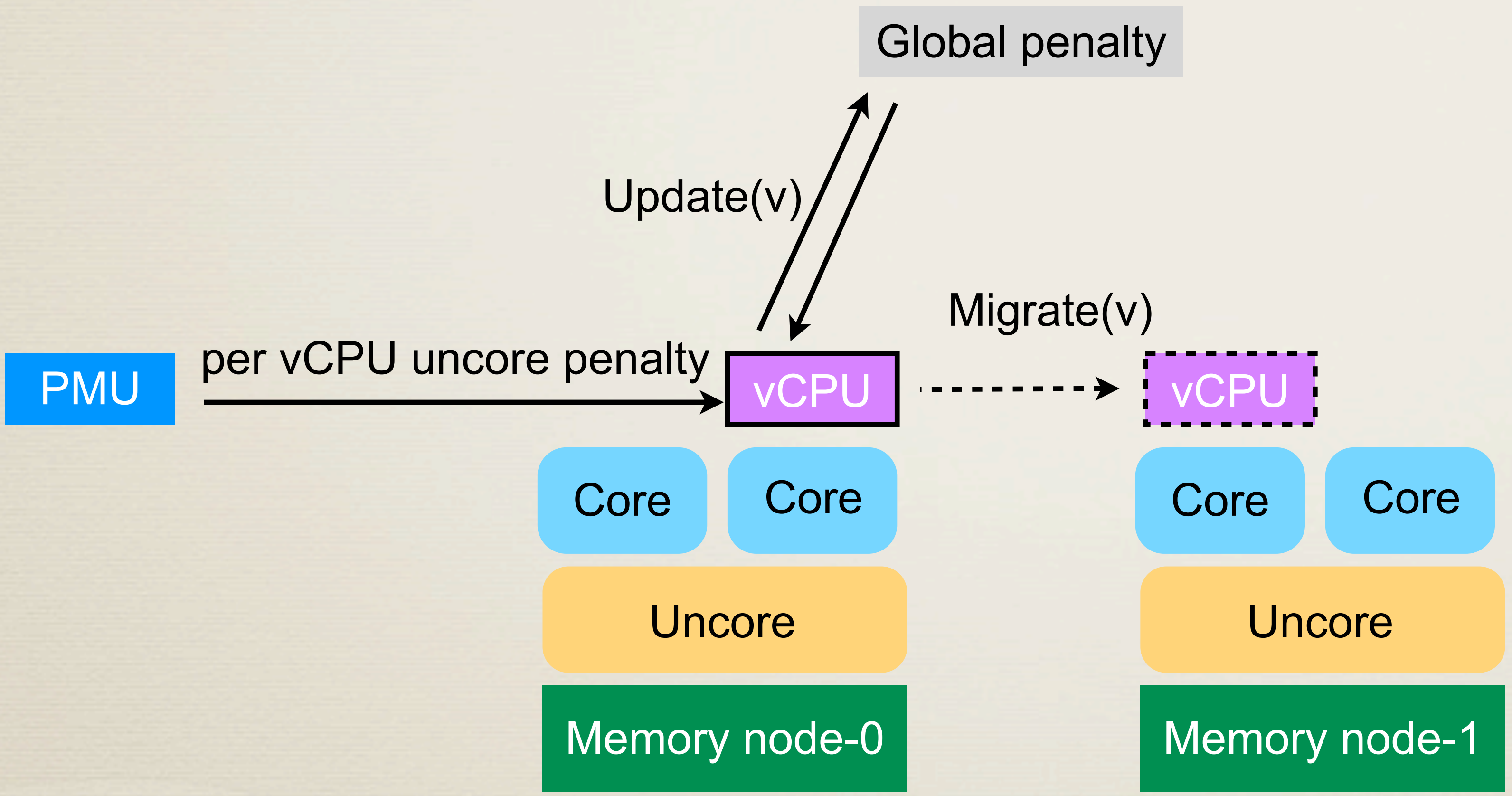
Bias Random Migration

```
struct vcpu {
...
int c;  //current core
int bias; // migration bias
int prob; //migration probability
int unc[NODE]; //array of global penalties
...
}
```

```
Procedure Migrate(v)
    Update(v)
    if v is a migration candidate then
        new_core = BiasRandomPick(v)
    else
        new_core = DefaultPick(v)
    end if
end procedure
```

```
Procedure Update(v)
    n = NUMA_CPU_TO_NODE(v.c)
    update global penalty and save it in v.unc[n]
    min = $\arg\min_x$ v.unc[x], x= 0,..., NODE-1
    if v.unc[n] > v.unc[min] then
        v.prob ++
    else
        v.prob --
        v.bias = n
    end if
end procedure
```

```
Procedure BiasRandomPick(v)
    rand = random() mod 100
        if rand < v.prob then
            select a core in node v.bias
            reset v.prob
        else
            Select current core
        end if
end procedure
```

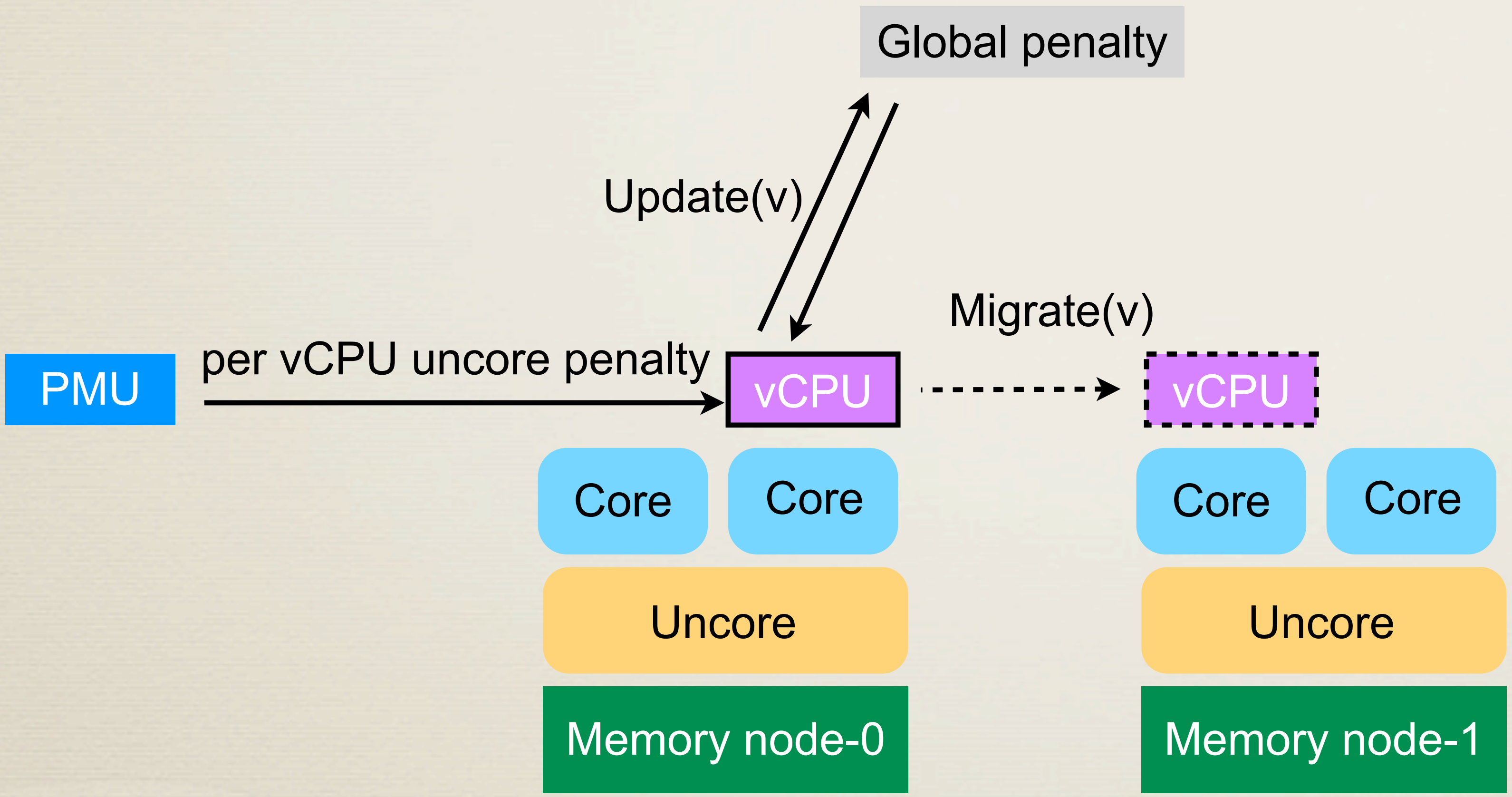


Bias Random Migration

```
struct vcpu {
...
int c;  //current core
int bias; // migration bias
int prob; //migration probability
int unc[NODE]; //array of global penalties
...
}
```

```
Procedure Migrate(v)
    Update(v)
    if v is a migration candidate then
        new_core = BiasRandomPick(v)
    else
        new_core = DefaultPick(v)
    end if
end procedure
```

```
Procedure Update(v)
    n = NUMA_CPU_TO_NODE(v.c)
    update global penalty and save it in v.unc[n]
    min = argmin v.unc[x], x= 0,..., NODE-1
               x
    if v.unc[n] > v.unc[min] then
        v.prob ++
    else
        v.prob --
        v.bias = n
    end if
end procedure
```



```
Procedure BiasRandomPick(v)
    rand = random() mod 100
        if rand < v.prob then
            select a core in node v.bias
            reset v.prob
        else
            Select current core
        end if
end procedure
```

University of Colorado
Colorado Springs

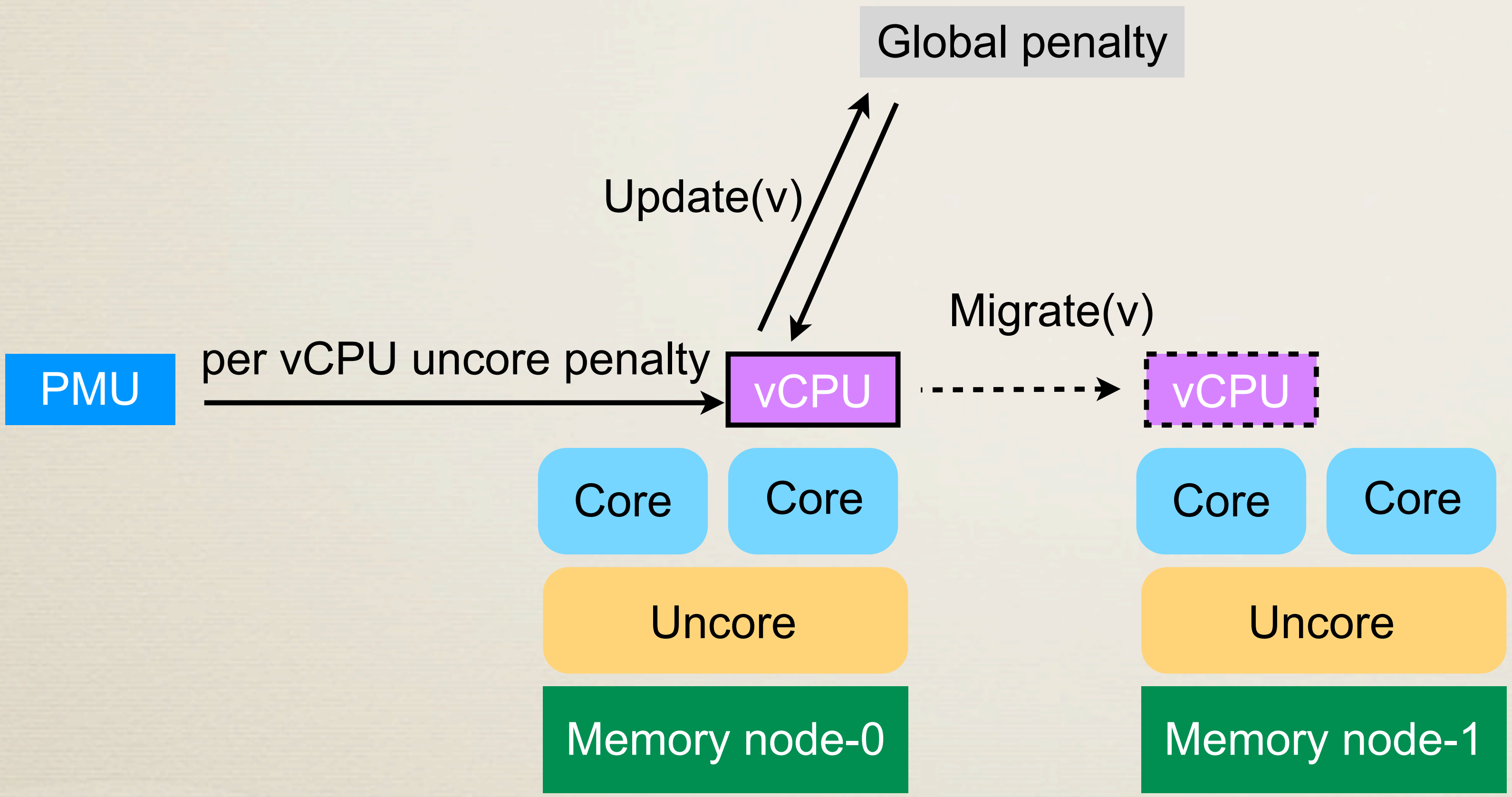
Bias Random Migration

```
struct vcpu {
...
int c;  //current core
int bias; // migration bias
int prob; //migration probability
int unc[NODE]; //array of global penalties
...
}
```

```
Procedure Migrate(v)
    Update(v)
    if v is a migration candidate then
        new_core = BiasRandomPick(v)
    else
        new_core = DefaultPick(v)
    end if
end procedure
```

```
Procedure Update(v)
    n = NUMA_CPU_TO_NODE(v.c)
    update global penalty and save it in v.unc[n]
    min = $\underset{x}{\mathrm{argmin}}$ v.unc[x], x= 0,..., NODE-1
    if v.unc[n] > v.unc[min] then
        v.prob ++
    else
        v.prob --
        v.bias = n
    end if
end procedure
```

```
Procedure BiasRandomPick(v)
    rand = random() mod 100
        if rand < v.prob then
            select a core in node v.bias
            reset v.prob
        else
            Select current core
        end if
end procedure
```



University of Colorado
Colorado Springs

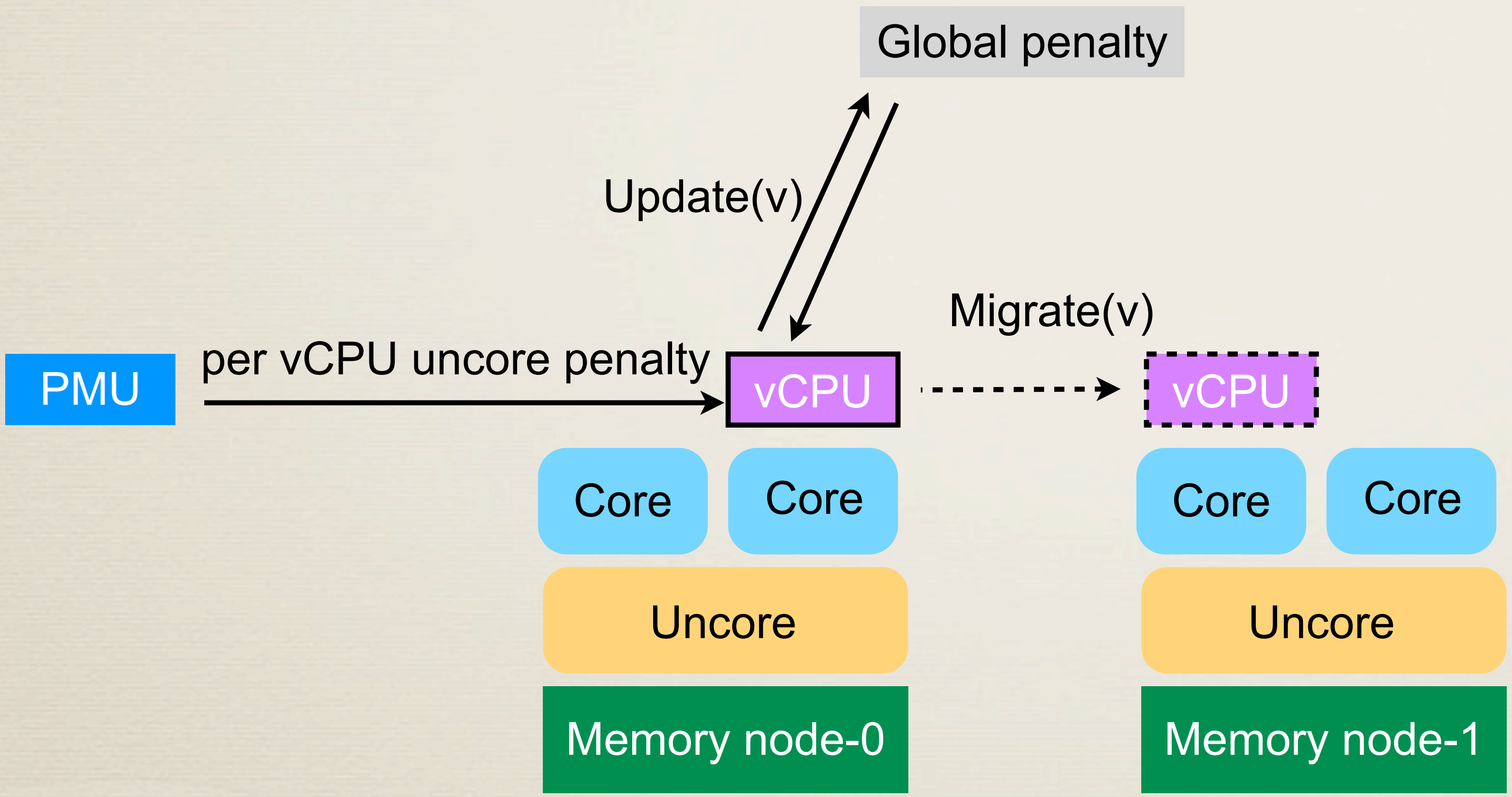
Bias Random Migration

```
struct vcpu {
...
int c;  //current core
int bias; // migration bias
int prob; //migration probability
int unc[NODE]; //array of global penalties
...
}
```

```
Procedure Migrate(v)
    Update(v)
    if v is a migration candidate then
        new_core = BiasRandomPick(v)
    else
        new_core = DefaultPick(v)
    end if
end procedure
```

```
Procedure Update(v)
    n = NUMA_CPU_TO_NODE(v.c)
    update global penalty and save it in v.unc[n]
    min = $\underset{x}{\text{argmin}}$ v.unc[x], x= 0,..., NODE-1
    if v.unc[n] > v.unc[min] then
        v.prob ++
    else
        v.prob --
        v.bias = n
    end if
end procedure
```

```
Procedure BiasRandomPick(v)
    rand = random() mod 100
        if rand < v.prob then
            select a core in node v.bias
            reset v.prob
        else
            Select current core
        end if
end procedure
```



Push migrate to a node with the minimum global penalty
Pull migrate (stealing) not touched for load balancing

UCCS University of Colorado Colorado Springs

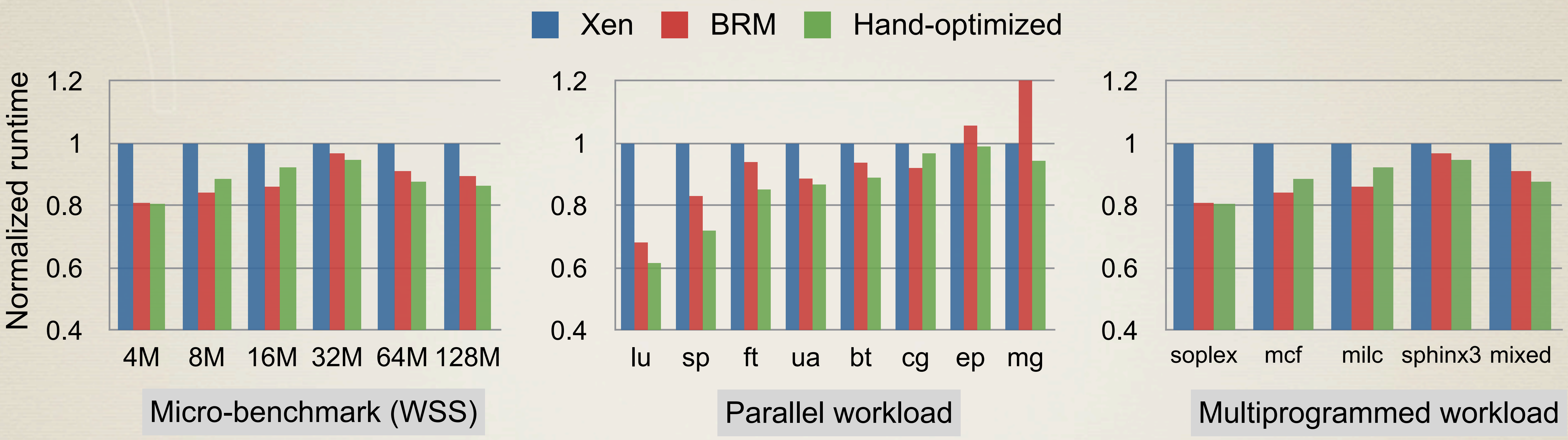
Implementation

- Xen 4.0.2
  - patched with `Perfctr-Xen` to read PMU counters
  - update uncore penalty every 10ms when Xen burns credits
  - lightweight random number generator using last 2 digits in TSC, in range [0,99]
- Guest OS, Linux 2.6.32
  - two new hypercalls: `tag` and `clear`
  - tag a vCPU as candidate if `cpus_allowed` is a subset of online CPUs

Workload

- Micro-benchmark
  - 4 threads, 128KB sharing size, WSS changes from 4MB to 8MB
- Parallel workload
  - NAS parallel benchmarks except `is`
  - Compiled with OpenMP, busy waiting synchronization
- Multiprogrammed workload
  - SPEC CPU2006, 4 identical copies of `mcf, milc, soplex, sphinx3`
  - Mixed workload = `mcf + milc + soplex + sphinx3`

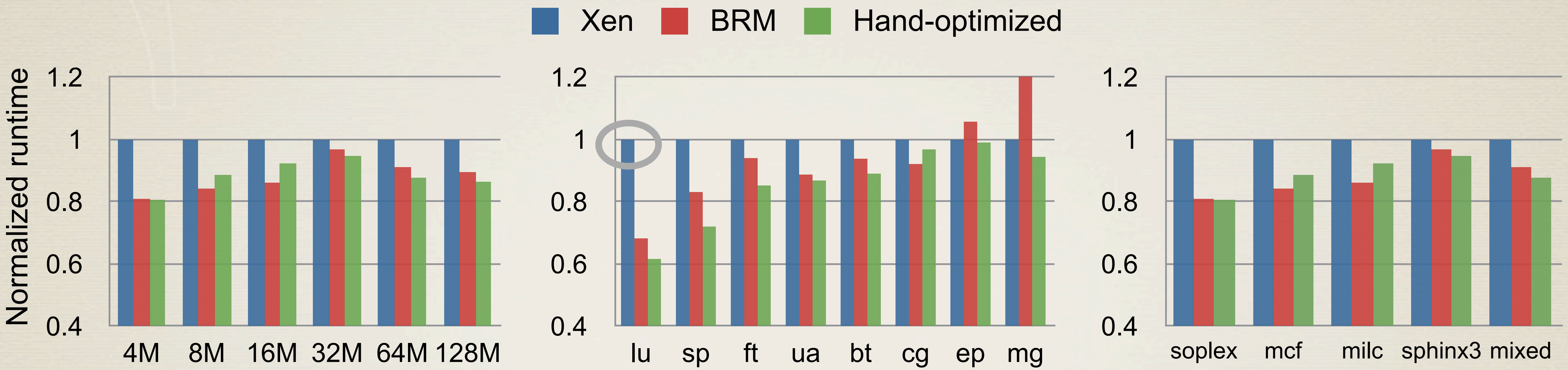
Improving Performance



🟩 Hand-optimized: offline determined best policy

UCCS University of Colorado Colorado Springs

Improving Performance



Micro-benchmark (WSS)

Parallel workload

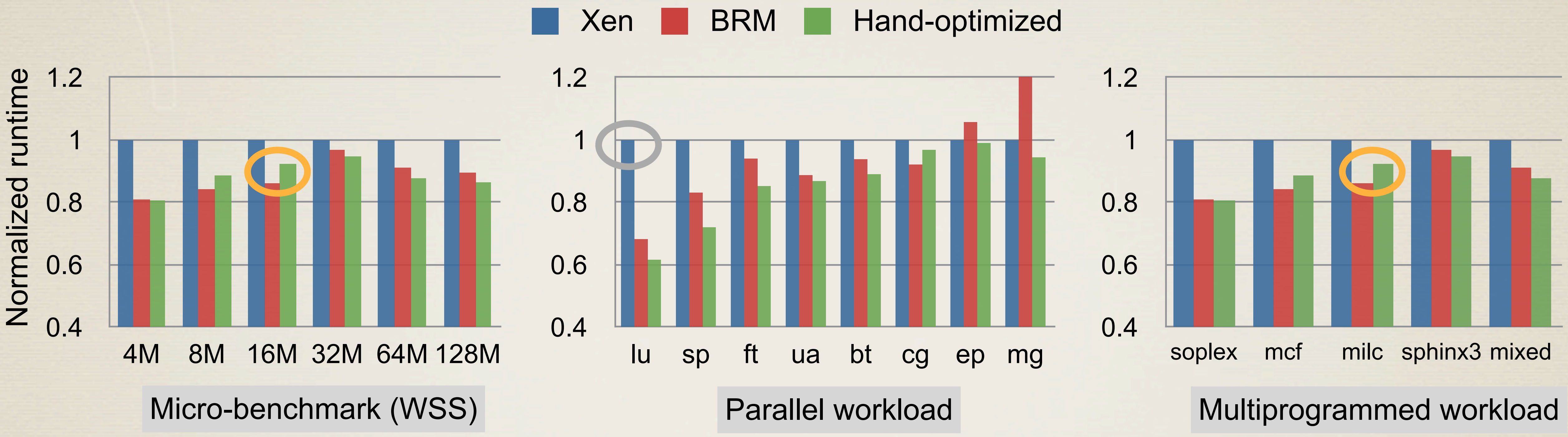
Multiprogrammed workload

1. BRM outperforms Xen in most experiments and improves performance by up to 31.7%

🟩 Hand-optimized: offline determined best policy

UCCS University of Colorado
Colorado Springs

Improving Performance

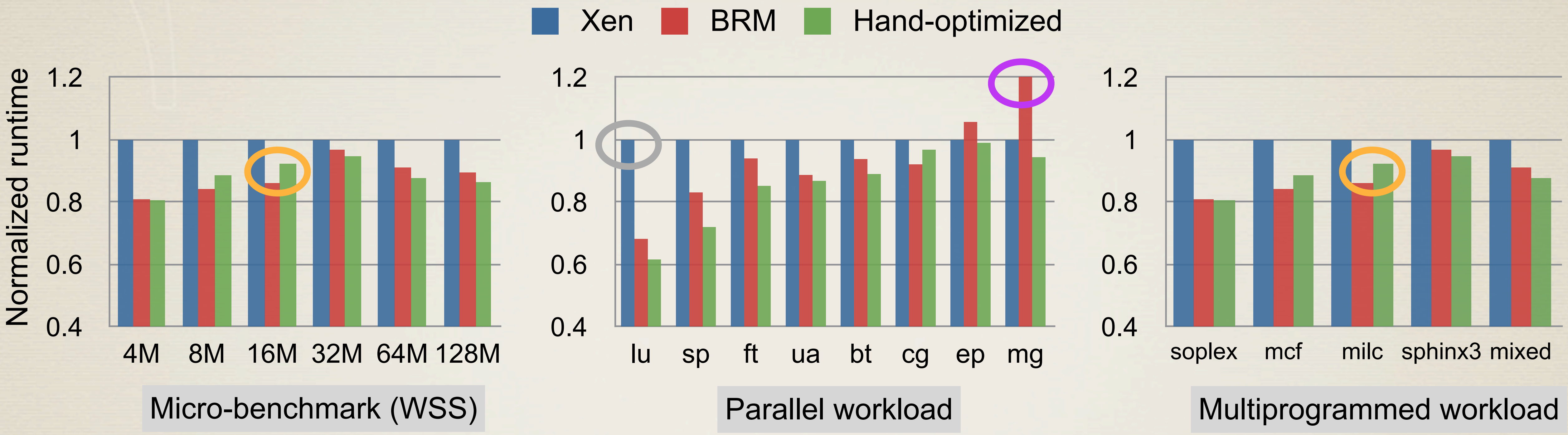


🟩 Hand-optimized: offline determined best policy

1. BRM outperforms Xen in most experiments and improves performance by up to 31.7%

2. BRM performs closely to Hand-optimized and even outperforms it in some cases

Improving Performance



🟩 Hand-optimized: offline determined best policy

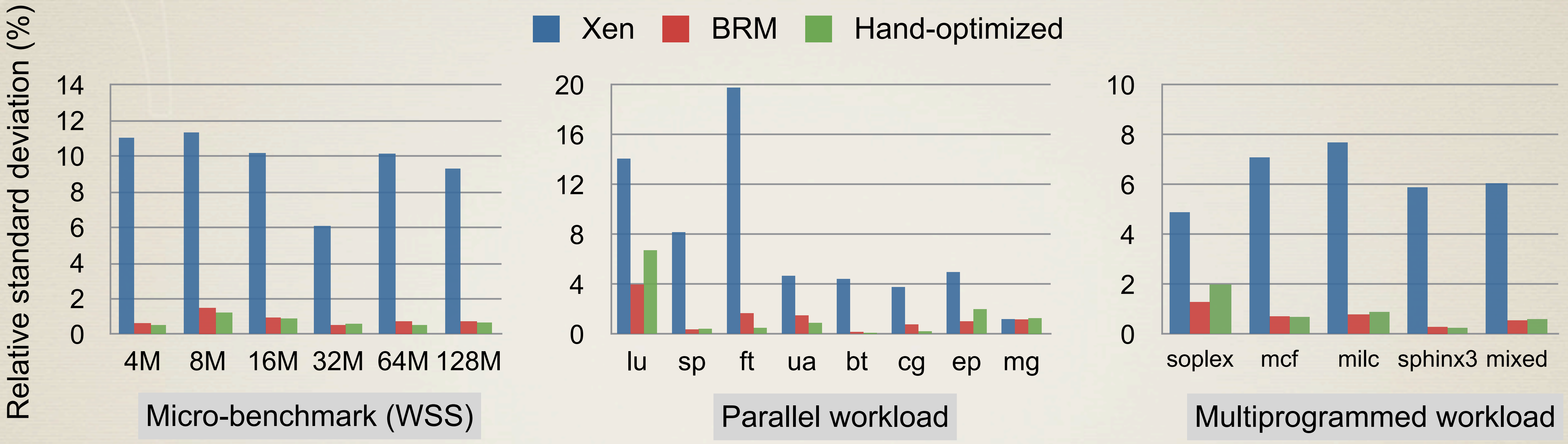
1. BRM outperforms Xen in most experiments and improves performance by up to 31.7%

2. BRM performs closely to Hand-optimized and even outperforms it in some cases

3. BRM adds overhead to NUMA insensitive workloads

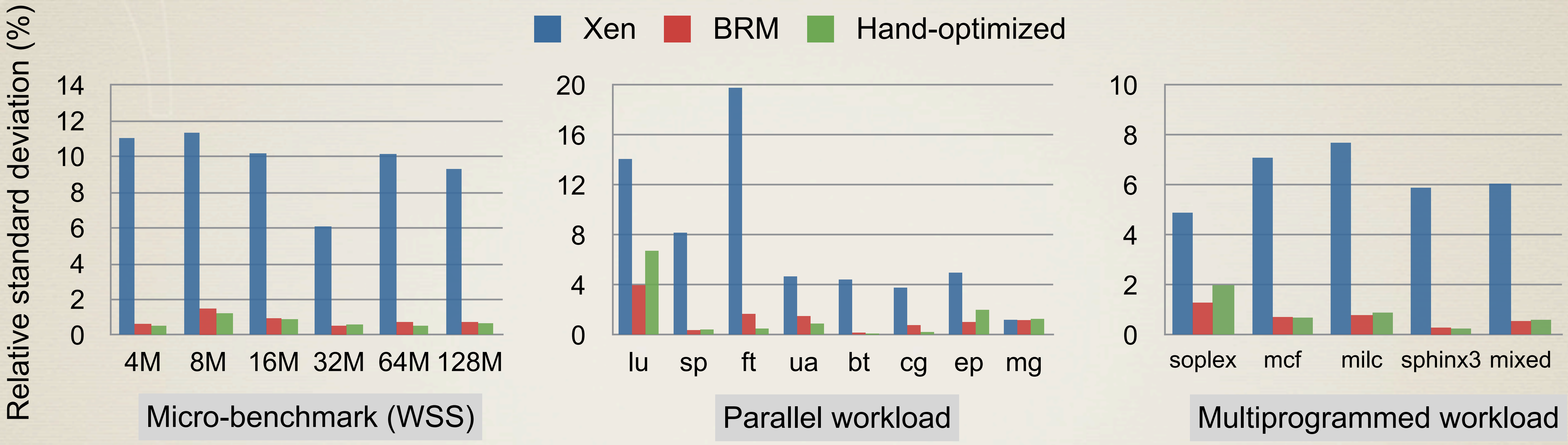
UCCS University of Colorado Colorado Springs

Reducing Variation



University of Colorado
Colorado Springs

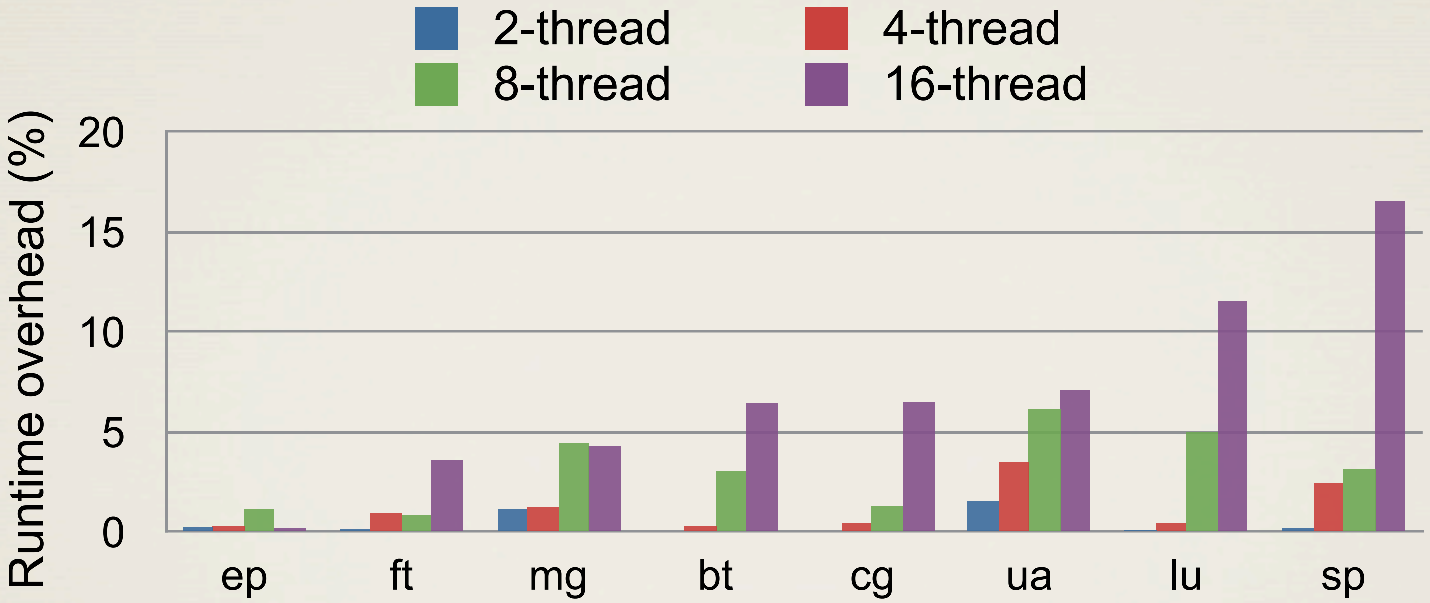
Reducing Variation



1. BRM reduces runtime variation significantly, with on average no more than 2% variations

University of Colorado
Colorado Springs

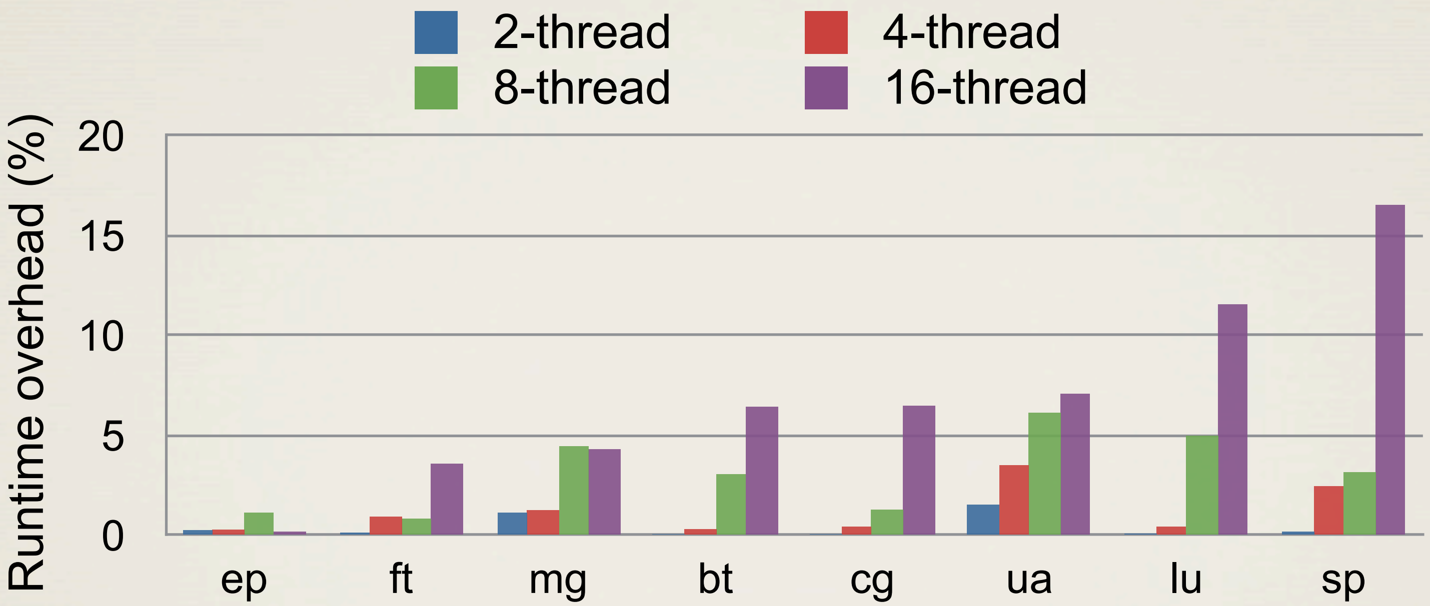
Overhead



migration disabled

University of Colorado
Colorado Springs

Overhead

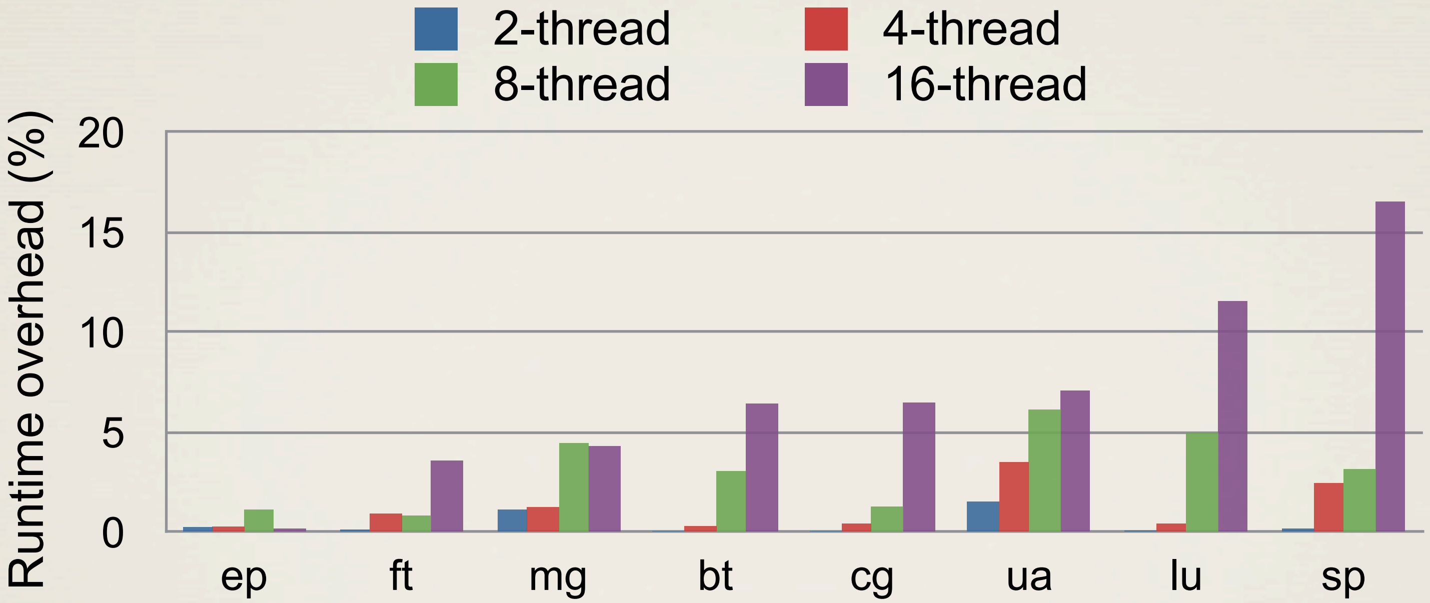


migration disabled

1. Less than 2% overhead for 2- and 4-thread workloads
2. BRM incurs up to 6.4% overhead for 8 threads, but still useful
3. Running 16-thread workloads is problematic

University of Colorado
Colorado Springs

Overhead



migration disabled

1. Less than 2% overhead for 2- and 4-thread workloads
2. BRM incurs up to 6.4% overhead for 8 threads, but still useful
3. Running 16-thread workloads is problematic

Amazon EC2's High-CPU Extra Large Instance has 8 vCPUs

UCCS University of Colorado Colorado Springs

Conclusion and Future Work

- Problem
  - sub-optimal scheduling and unpredictable performance
- Our approach
  - uncore penalty as a performance index
  - Bias random migration for online performance optimization
  - improves performance and reduces variability
- Future work
  - inferring NUMA-sensitive vCPU
  - improving scalability and considering simultaneous multithreading

Q&A

Thank you !

jrao@uccs.edu
http://cs.uccs.edu/~jrao/

University of Colorado
Colorado Springs

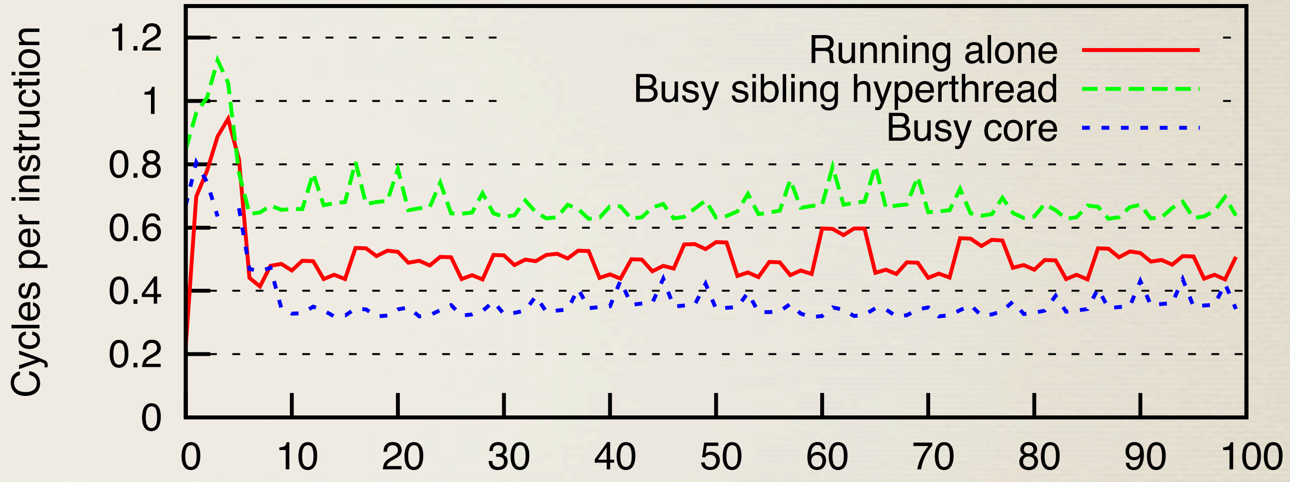
Backup Slides begin here...

University of Colorado
Colorado Springs

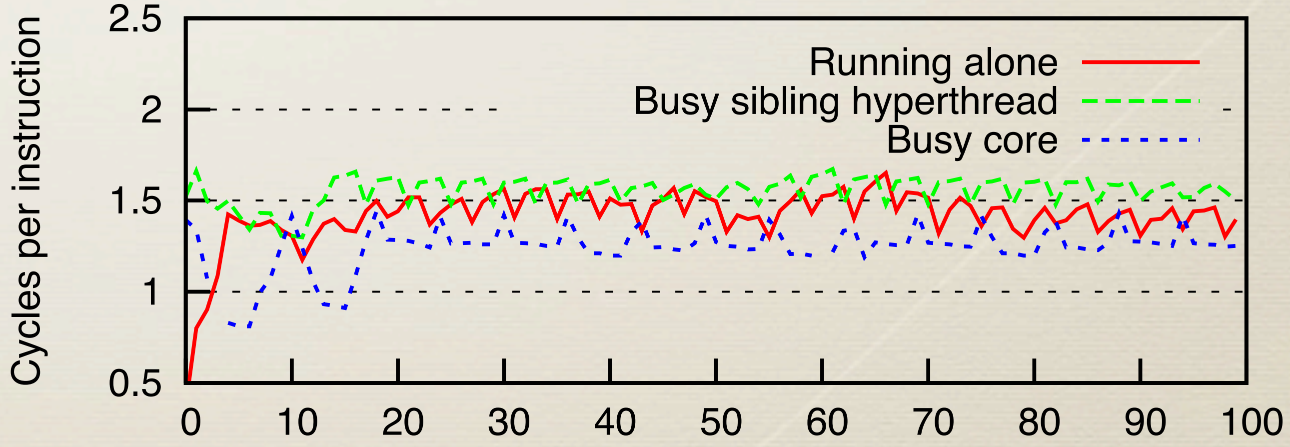
Why IPC or CPI Harmful?*

IPC or CPI does not reflect performance, even not for straggler thread

(a) Bt



(b) Lu



* Wood, MICRO'06

Questions

- Why not Cycles per Instruction (CPI)?
  - CPI is not useful for multiprocessor workloads
- How to improve scalability?
  - Li et al., PPoPP'09, relaxing consistency requirement on global update
- What workload BRM is not useful for?
  - short-lived jobs
- Does BRM affect fairness and priorities?
  - NO

University of Colorado
Colorado Springs