kMAF: Automatic Kernel-Level Management of Thread and Data Affinity

Matthias Diener, Eduardo Cruz, Philippe Navaux – UFRGS, Brazil

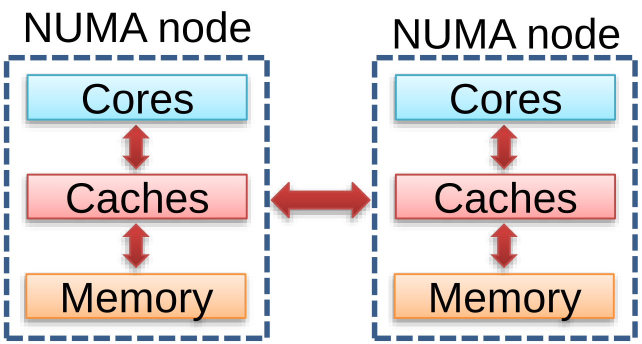
Anselm Busse, Hans-Ulrich Heiß – TU Berlin, Germany

PACT 2014 – Edmonton, Canada

August 26th, 2014

Memory accesses of parallel applications

- **Memory accesses** present challenges for parallel architectures
  - Performance, energy consumption



- In shared-memory: copy data between
  - Levels of the memory hierarchy
  - Processors / NUMA nodes
- **Cost of memory accesses is not uniform**
  - Local vs. remote accesses

Optimize performance and energy consumption of memory accesses by improving **locality**

- **Affinity-based thread and data mapping**

Improved thread and data mapping

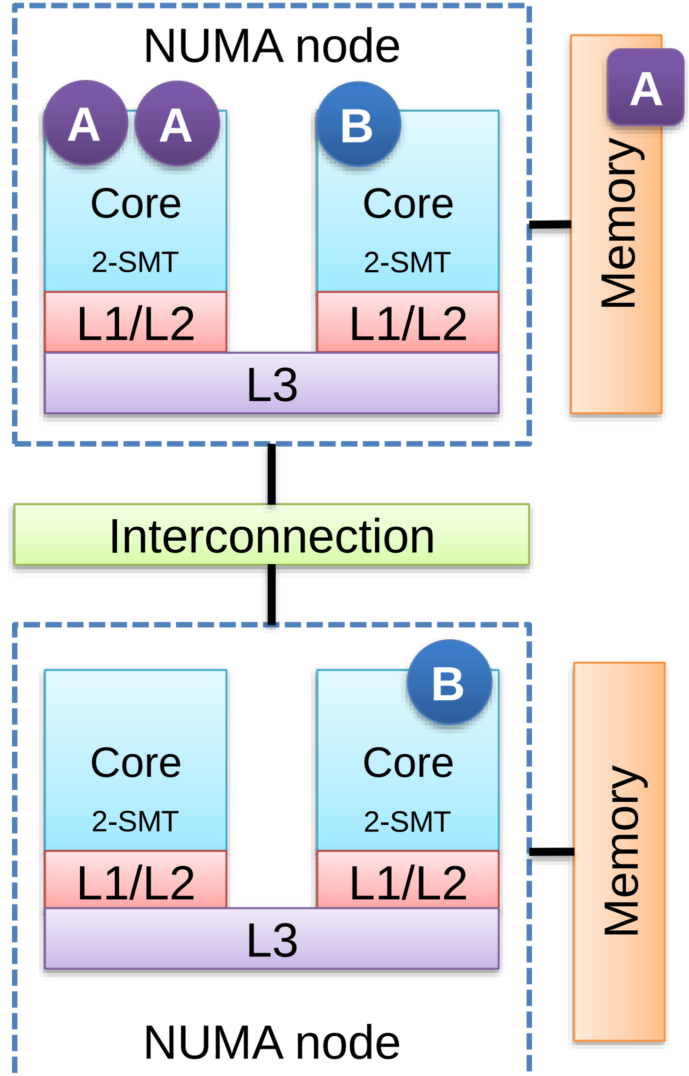
Locality can be improved in 2 ways:

1. Thread mapping

- Assign threads to cores
  - Traditionally: locality not a goal
- **Affinity-based** mapping
  - Analyze accesses to **shared data**
  - Lots of sharing → map closely A
  - Little sharing → map distantly B

2. Data mapping

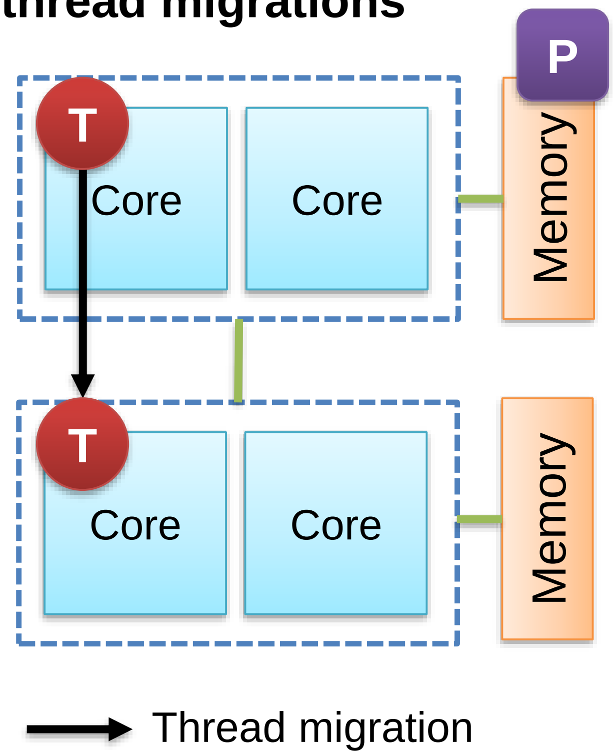
- Assign memory pages to NUMA nodes
  - Traditionally: first touch
- **Affinity-based** mapping
  - Map pages to the node where they are accessed the most A



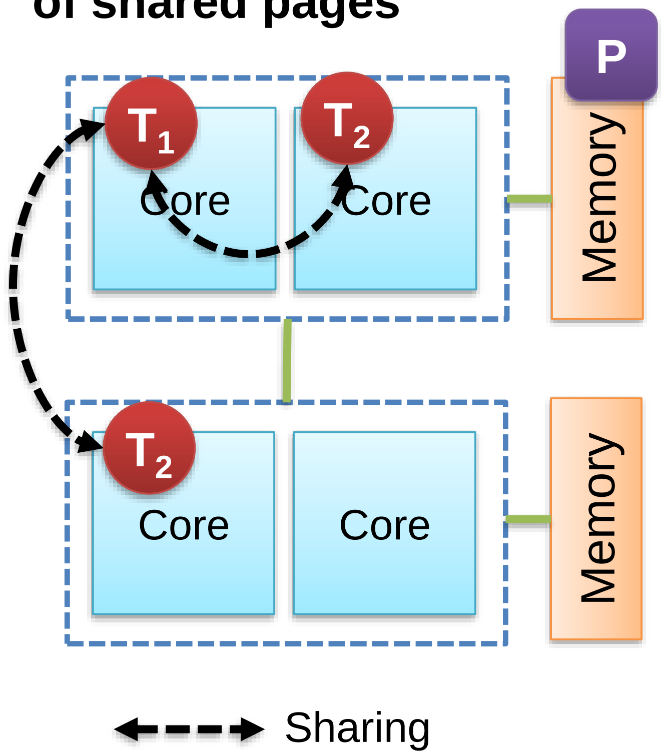
Combining thread and data mapping

Thread mapping is **essential** for data mapping

1. Prevent unnecessary thread migrations



2. Increase data mapping gains of shared pages



Which type of behavior is suitable for mapping?

How can the mapping be performed?

1. Characterize applications for mapping suitability
2. kMAF: Automatic thread/data mapping

Characterizing Application Behavior For Thread and Data Mapping

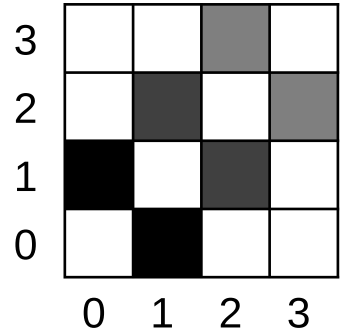
Thread affinity

Thread affinity

- Focus on improving cache usage
- How do threads share data?
- **Heterogeneity:** threads need to share data in a non-uniform way
  - Calculate variance of sharing
  - Expect more gains from high heterogeneity

Sharing matrix 1:

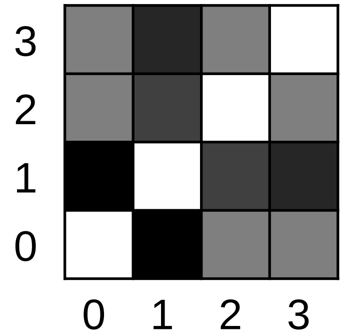
3	0	0	5	
2	0	6		5
1	7		6	0
0		7	0	0
	0	1	2	3



High heterogeneity

Sharing matrix 2:

3	5	7	5	
2	5	6		5
1	8		6	7
0		8	5	5
	0	1	2	3

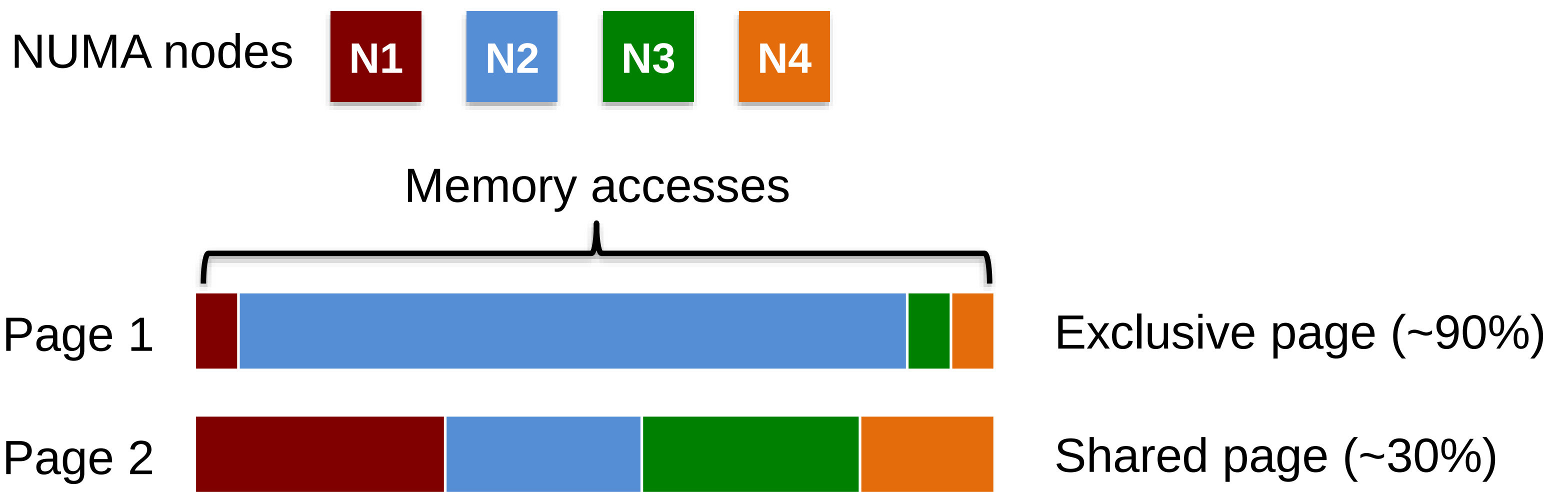


Low heterogeneity

Data affinity

Data affinity

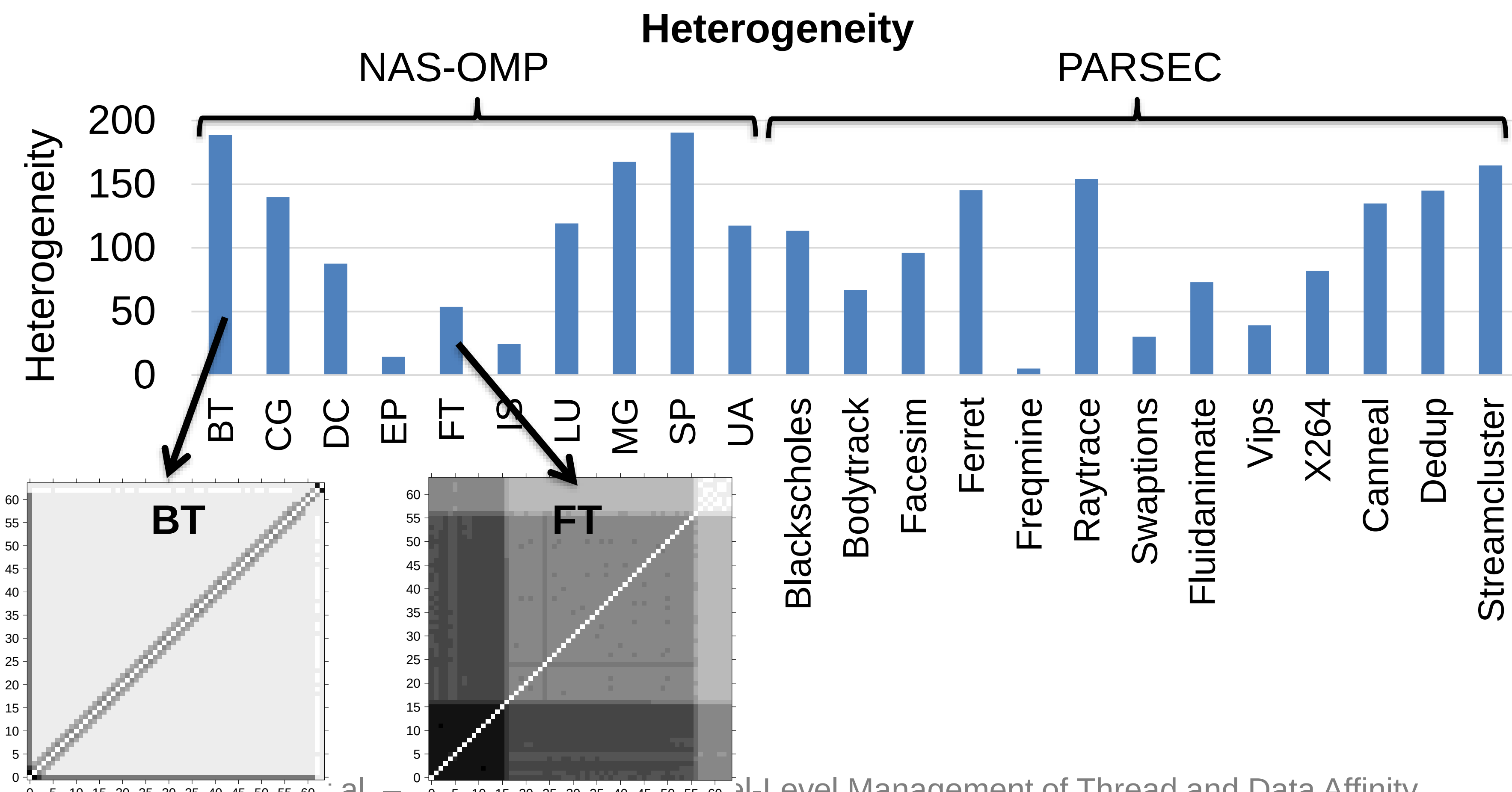
- Focus on reducing interconnection traffic between NUMA nodes
- How do threads access memory pages?
- **Exclusivity:** highest % of memory accesses from same NUMA node
  - Expect more gains from pages with a high exclusivity



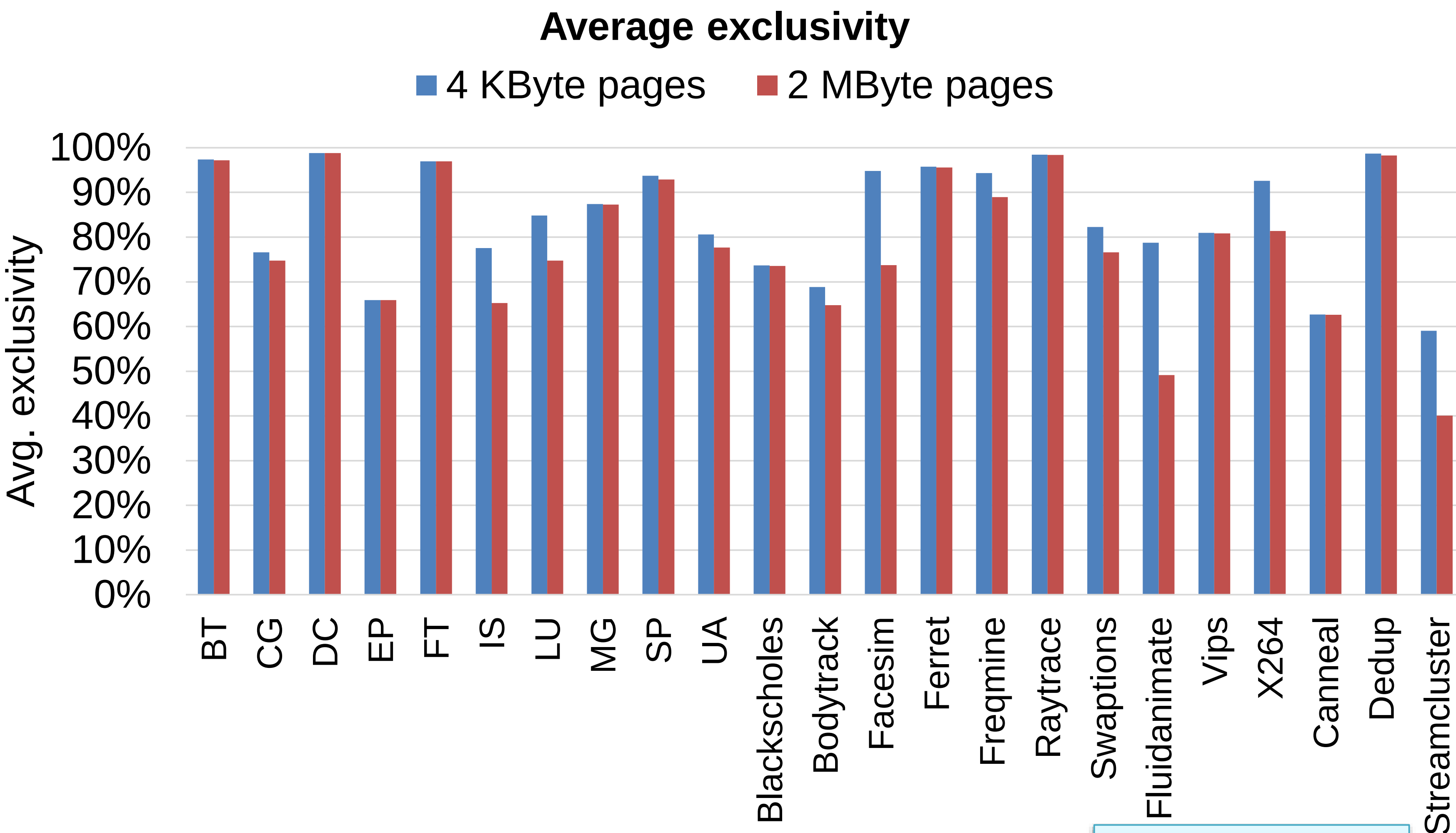
- **Average Exclusivity:** Calculate (weighted) exclusivity for all pages

Benchmark behavior (1)

- NAS-OMP (**C** input size), PARSEC (**native** input size)
- 64 threads, 4 NUMA nodes
- Pin-based simulator, trace all memory accesses



Benchmark behavior (2)



Mean:
4 KByte: 85%
2 MByte: 79%

kMAF: Automatic Kernel-Level Management of Thread and Data Affinity

kMAF – Introduction

- **Goal**: develop mechanism that performs mapping **automatically**

kMAF: kernel Memory Affinity Framework

- Features
  - Completely online/automatic
    - No need for previous information
  - Hardware-independent
  - Independent from parallel API
  - No changes to applications

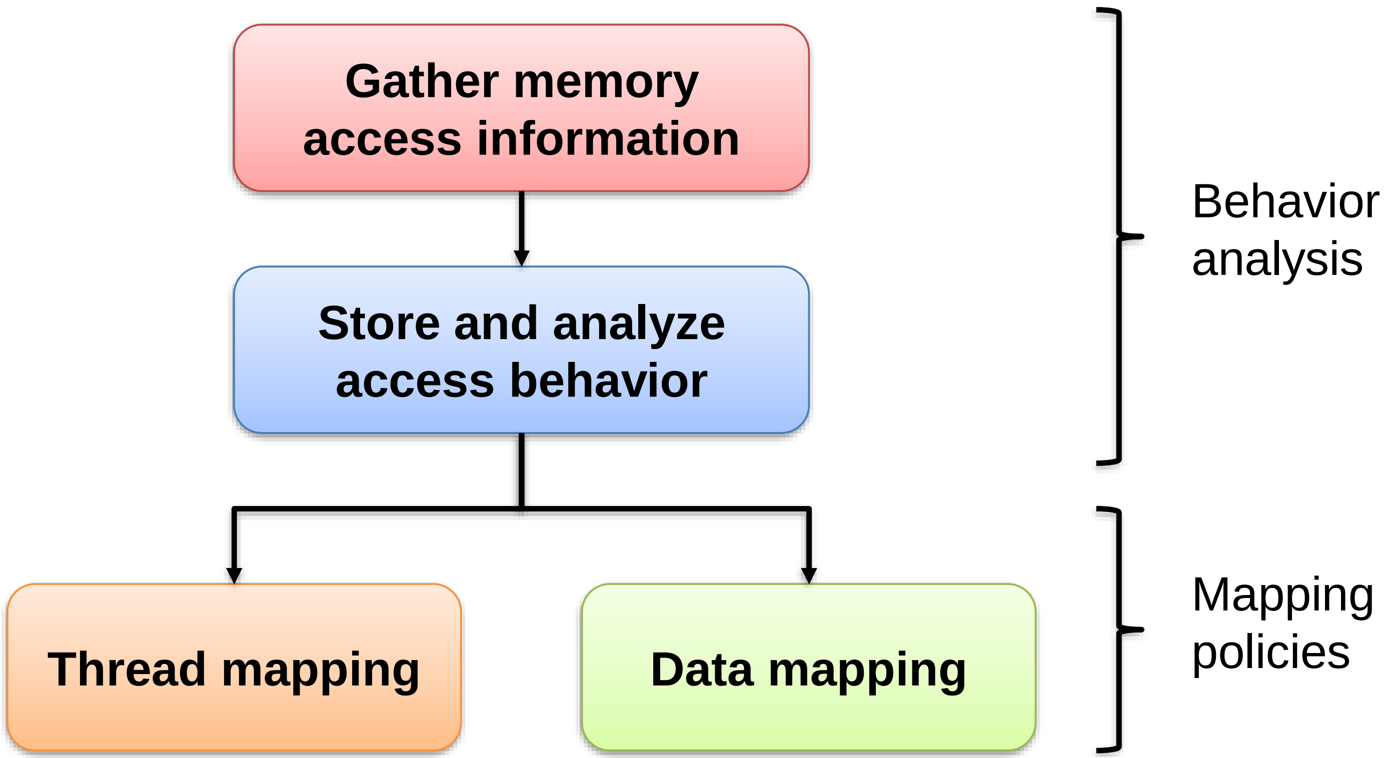
- Framework
  - Provide analysis and default policy
  - Easy to integrate other policies

Related work

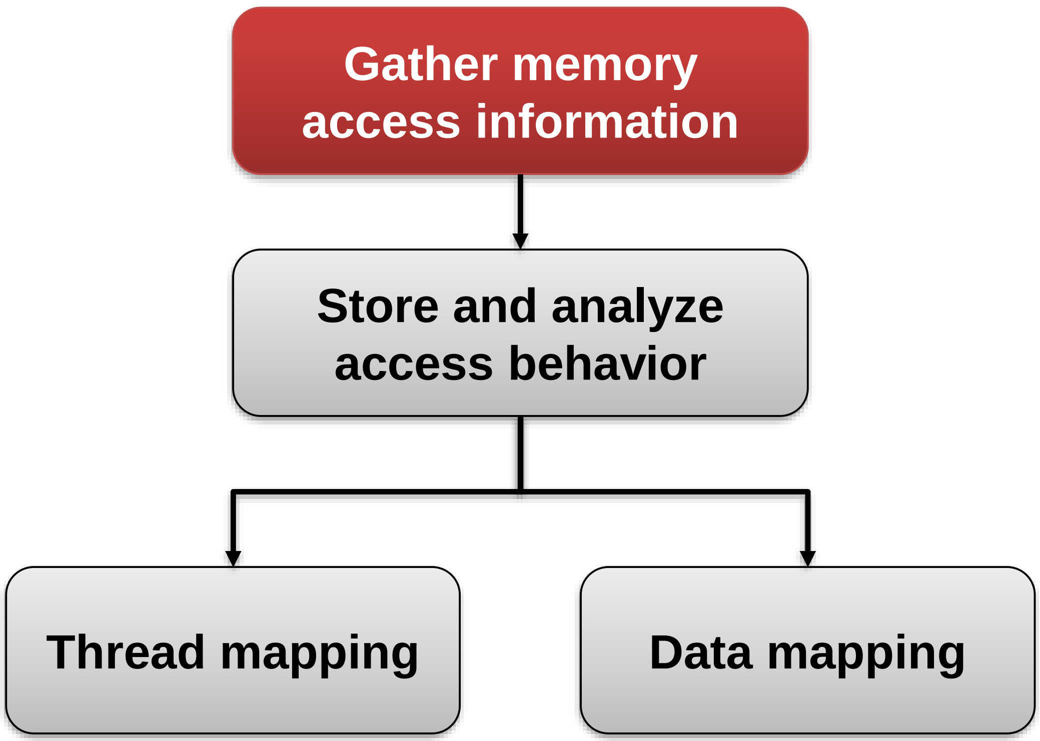
Mechanism	Type	Examples	Hardware Independent	Thread and Data Mapping
Manual	Source code changes, manual binding	libnuma, hwloc, MAi, …	✓	✓
Semi-automatic	Trace-based, compiler-based	MPIPP, Marathe et al., Minas, …	✓	✓
Automatic	Use indirect statistics: IPC, cache misses, TLB misses, page faults	Autopin, Azimi et al., Cruz et al., AutoNUMA, …	(✓)	
	kMAF		✓	✓

First mechanism to perform automatic thread and data mapping.

kMAF – Overview



kMAF – Overview



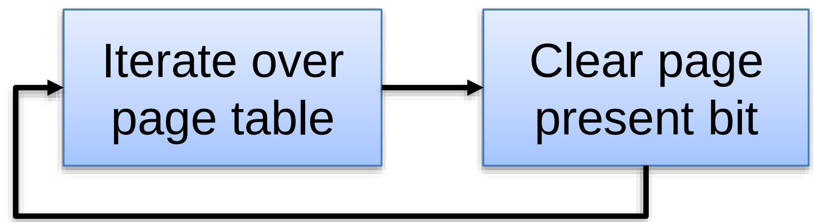
kMAF – Sample memory accesses

1. Use **page faults** to sample memory accesses

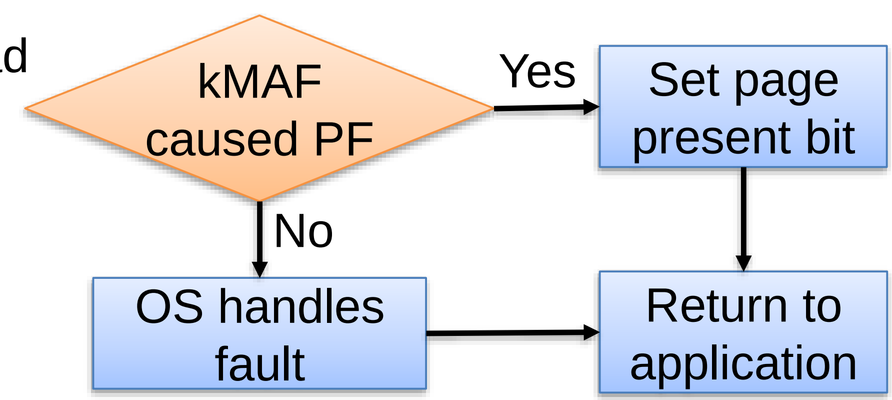
- Store information (address, thread ID)
- Page faults have full address

- Problem: only 1 page fault per page

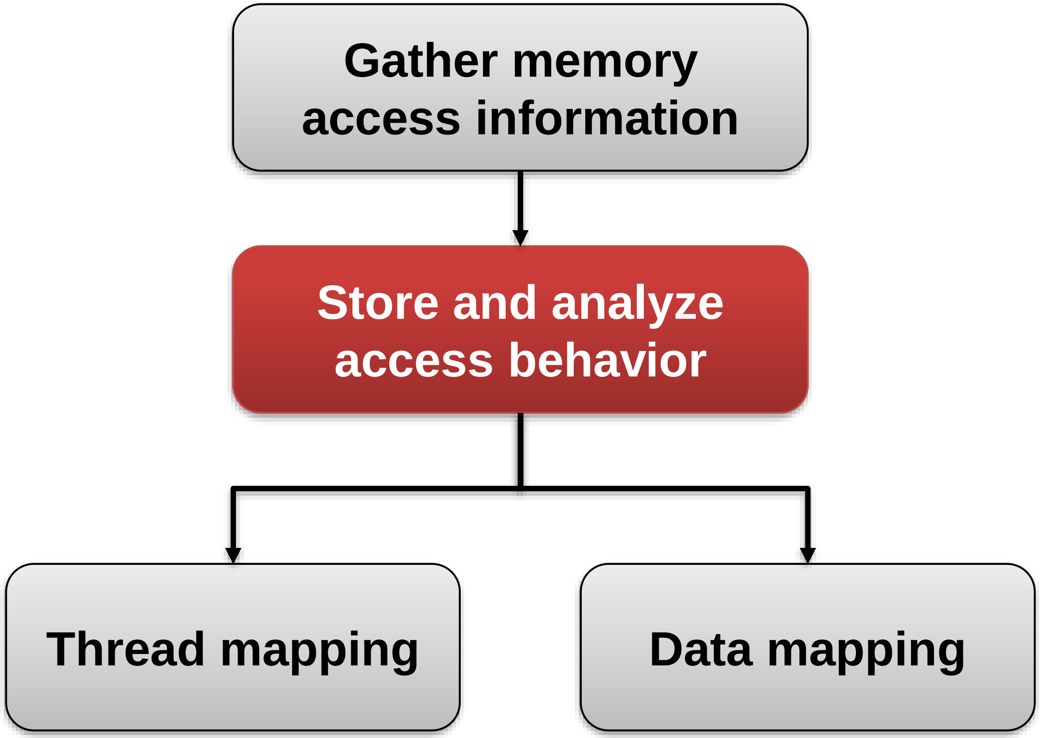
2. Insert **extra** page faults



- Can be resolved with a low overhead (no missing information)

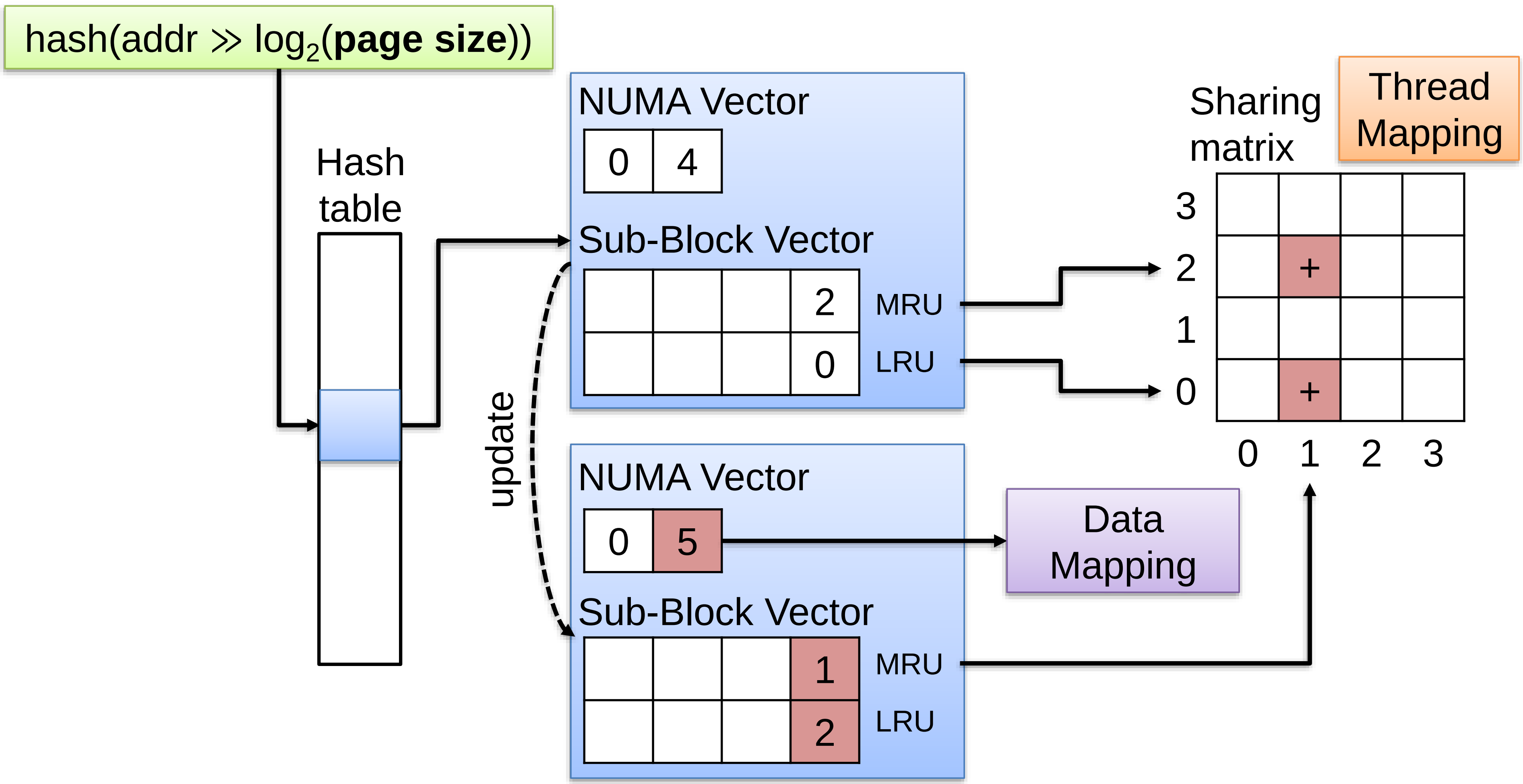


kMAF – Overview

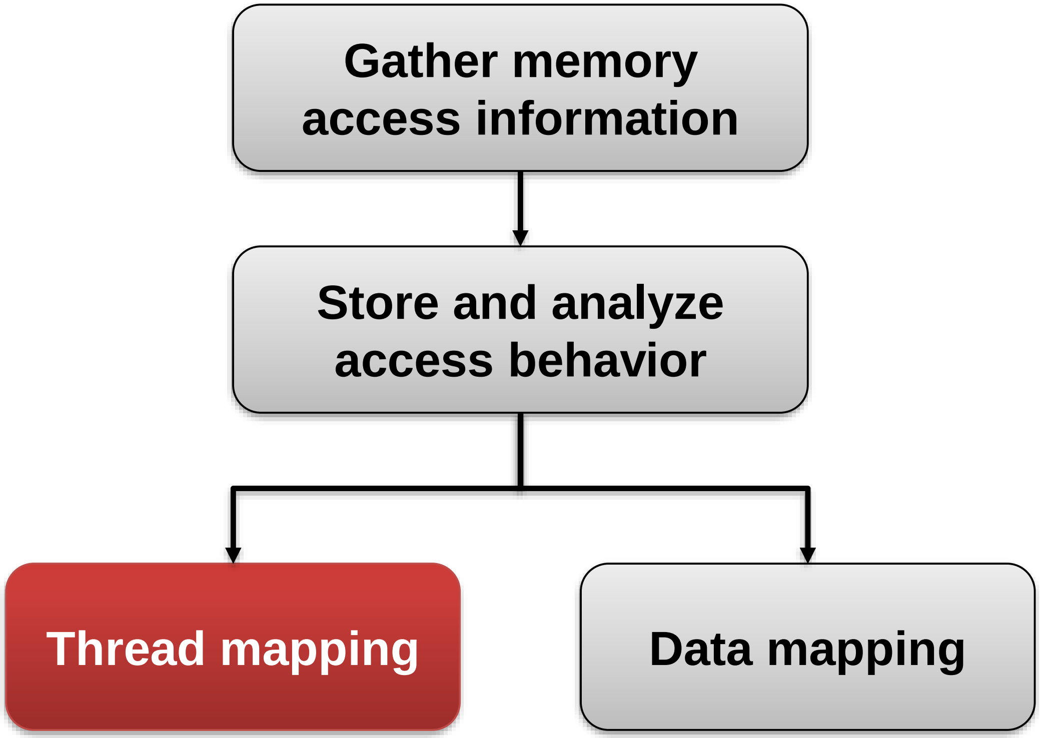


kMAF – Information storage and analysis

- Need to store sharing, page usage; different granularities
  - Ex.: **TID 1**, running on NUMA **node 2** (of 2) causes a page fault in **addr**

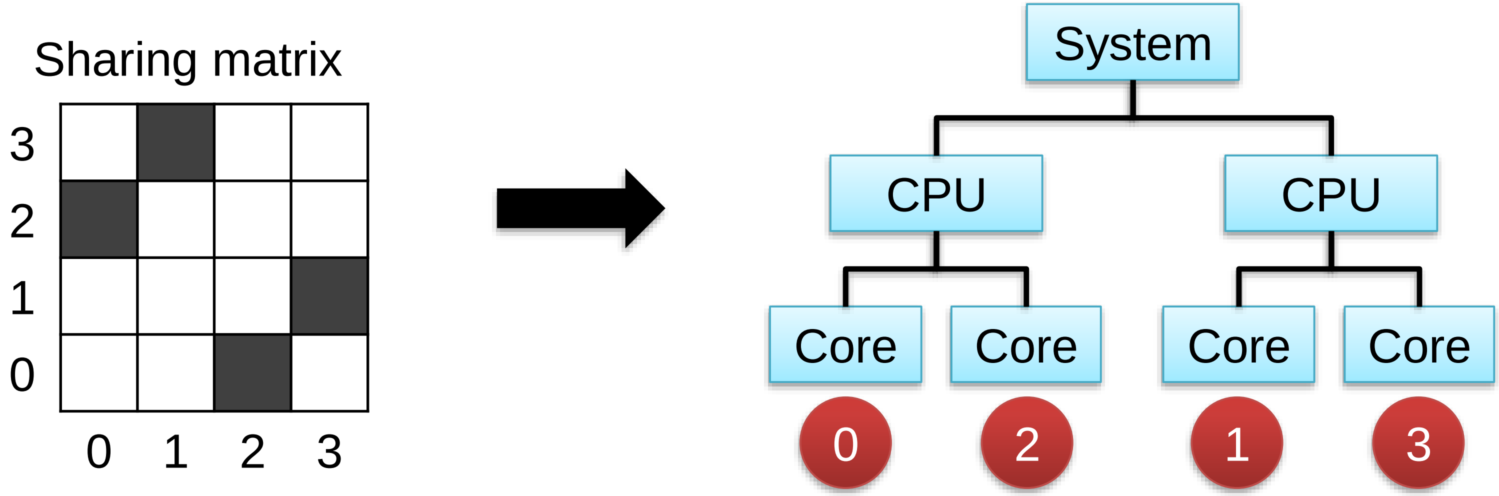


kMAF – Overview



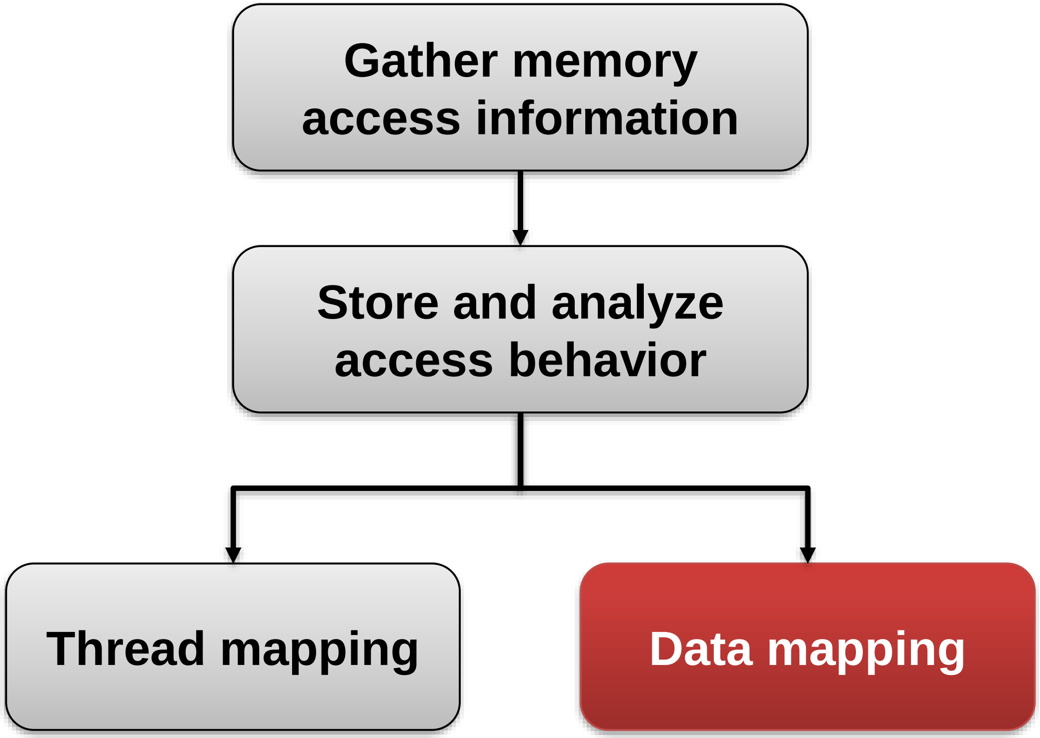
kMAF – Thread mapping

- Thread mapping
  - Assign threads to cores based on the sharing matrix



  - Periodically fetch sharing matrix
  - Use DRB algorithm from Scotch mapping library to calculate mapping
  - Use information to migrate threads

kMAF – Overview



kMAF – Data mapping

- Data mapping
  - Start with first-touch mapping
  - Evaluate NUMA vector for changes in access pattern
    - Migrate misplaced pages that have a high exclusivity
    - Prevent ping-pong of pages

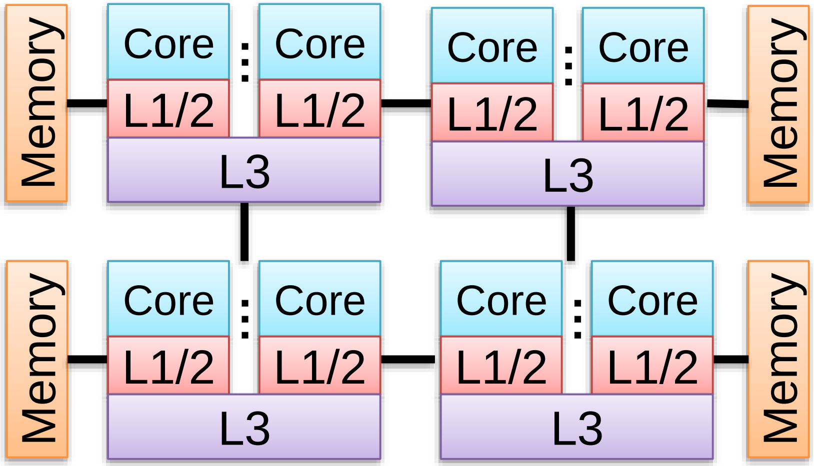
$$\max(\text{NumaVector}) > 2 * \max_2(\text{NumaVector}) + 1$$

	Node 1	Node 2	
– First touch:	1	0	→ allocate page on node 1
– 1st migration:	1	4	→ move page to node 2
– 2nd migration:	10	4	→ move page to node 1

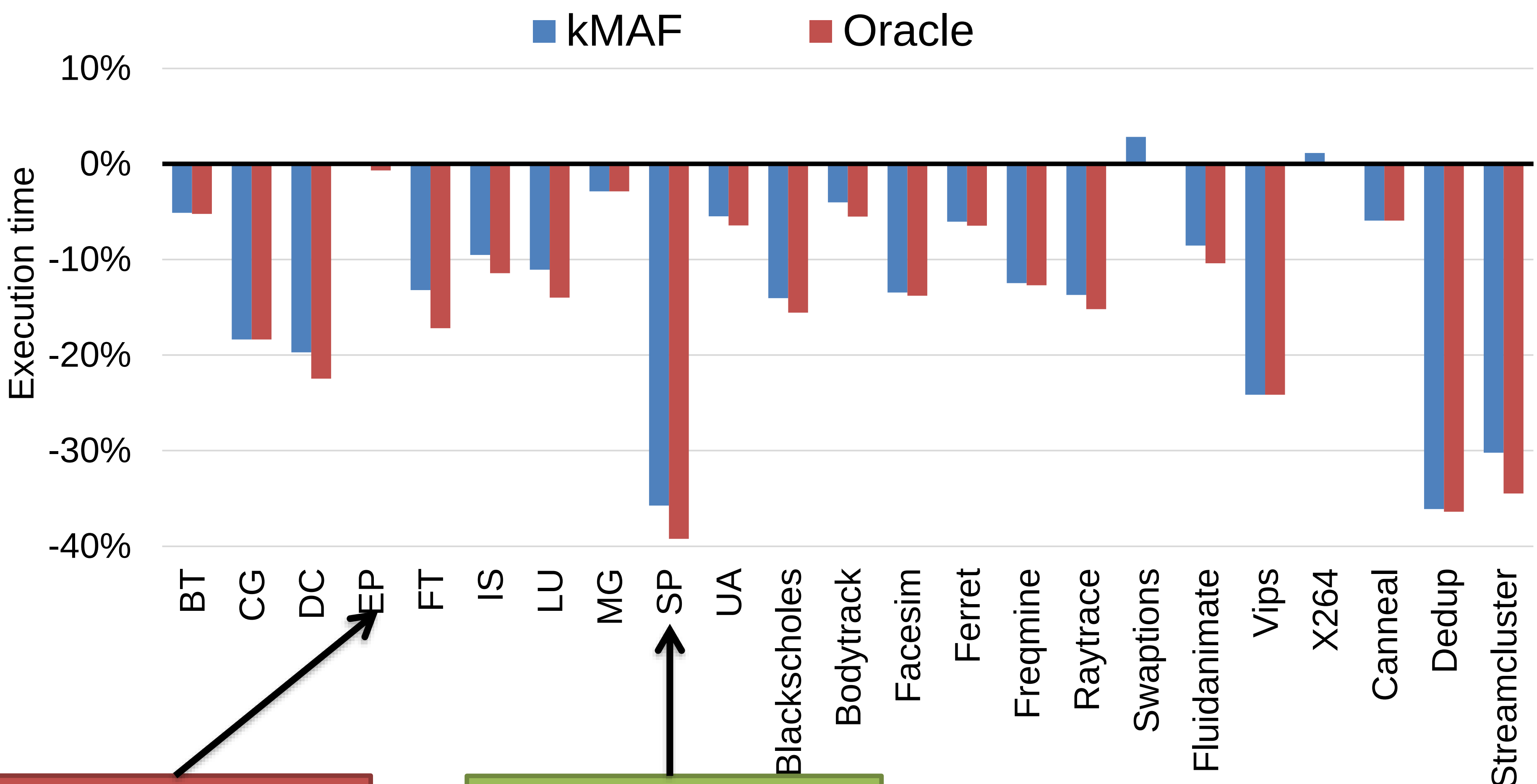
Evaluation of kMAF

Experimental methodology

- kMAF implemented in Linux kernel
- Benchmarks: NAS-OpenMP (**C** input size), PARSEC (**native** input)
- Compare kMAF to
  - OS (baseline)
  - Oracle (trace-based)
- Machine
  - 4* Intel Xeon X7550, 8 cores, 2 SMT, 2.0 GHz
  - Private L1/L2 caches, shared L3 cache
  - 64 threads, 4 NUMA nodes



kMAF – Execution time



Heterogeneity: low
Exclusivity: low

Heterogeneity: high
Exclusivity: high

Mean:
kMAF: -15%
Oracle: -18%

kMAF – Other measurements

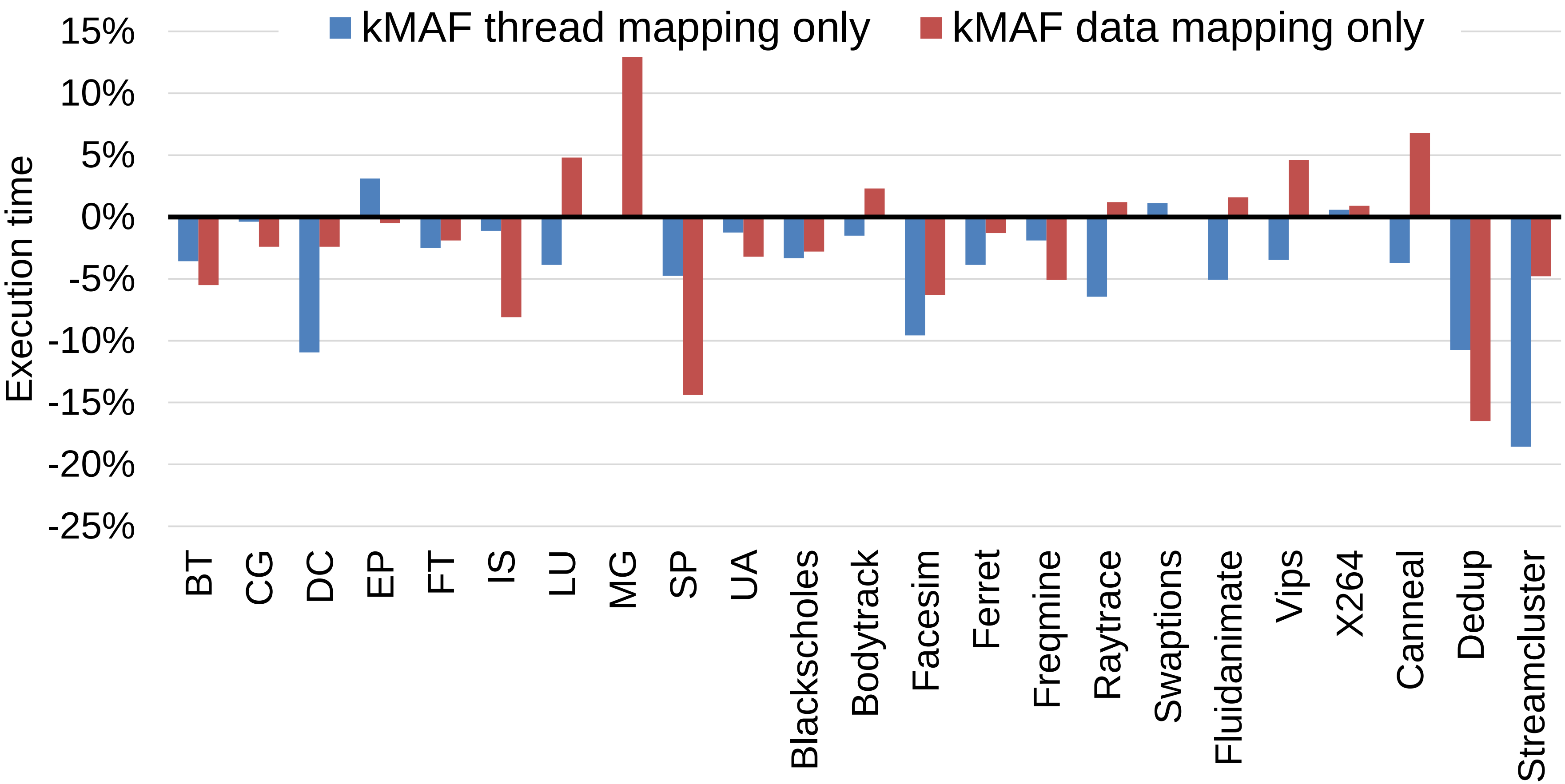
- L3 cache misses, QPI traffic, energy consumption
  – Improvements similar to execution time improvements

- L3 cache misses
  – kMAF: -9%, Oracle: -11%

- QPI traffic
  – kMAF: -18%, Oracle: -19%

- Energy consumption
  – kMAF: -9%, Oracle: -12%

kMAF – Thread/Data mapping separately



Mean:
Thread mapping: -7%
Data mapping: -3%
Both: -15%

kMAF – Overhead

- Storage overhead
  - Hash table: 20 Bytes per page (0.5% of 4 KByte pages)
  - Sharing matrix: 4 MByte for 1024 threads

- Runtime overhead
  - 1.5% of total execution time on average
  - About 10x extra page faults (highest results)

Conclusions

- Memory accesses represent big challenge for parallel applications

- **kMAF** can substantially improve performance/energy consumption
  - Large improvements (up to **40%**) compared to current OS mechanisms
  - Need integrated thread + data mapping for optimal improvements

- Future work
  - Phase-based thread mapping, balance memory pages on NUMA nodes

Download: http://inf.ufrgs.br/~mdiener

Thank you!

kMAF: Automatic Kernel-Level Management of Thread and Data Affinity

Matthias Diener, Eduardo Cruz, Philippe Navaux – UFRGS, Brazil

Anselm Busse, Hans-Ulrich Heiß – TU Berlin, Germany

Homepage: http://inf.ufrgs.br/~mdiener
Email: mdiener@inf.ufrgs.br

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

PACT 2014 – Edmonton, Canada

August 26th, 2014

TU berlin