Scientific approaches to Thread and Data Placement

Fabian Eckert

NUMA Seminar

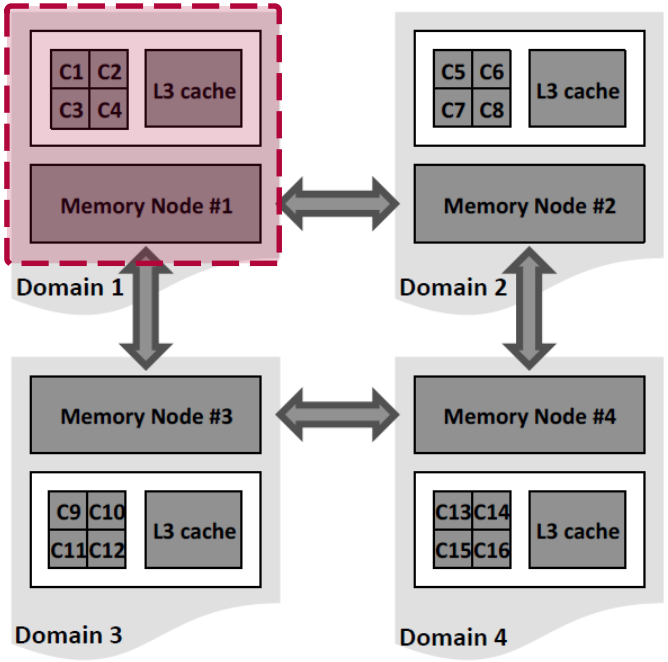
26.11. 2014

[1]

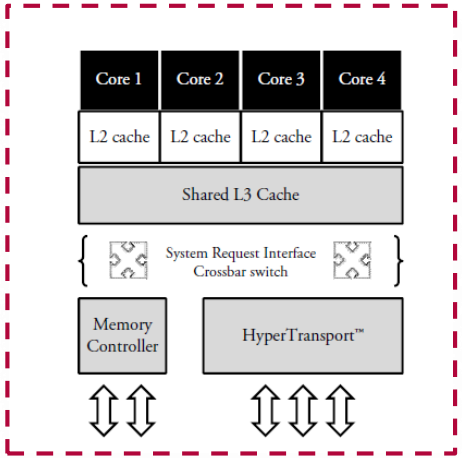
Agenda

1. Challenges of Thread and Data placement
2. **DINO**
3. **Carrefour**
4. Data Shuffling
5. Insights

Challenges of Thread and Data placement

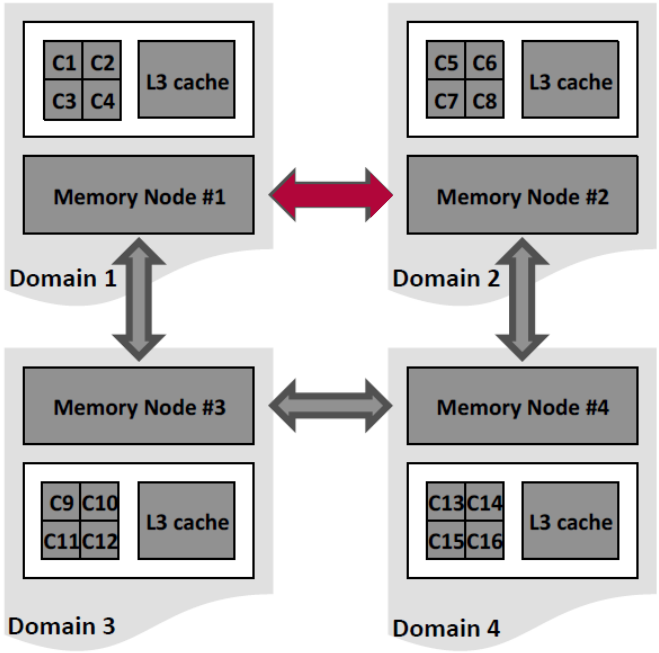


[2]



[3]

Challenges of Thread and Data placement



[2]

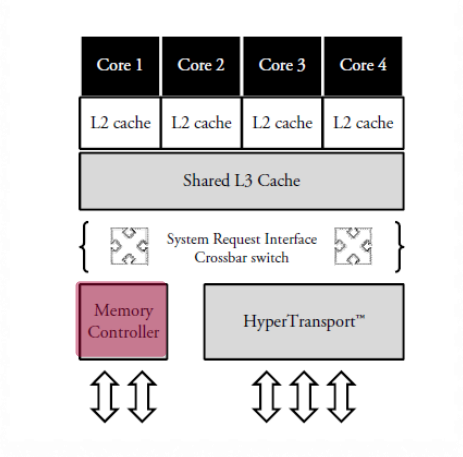
Remote memory access:

- remote wire delays / latency (RL)
- Congestion on interconnect links (IC)

Challenges of Thread and Data placement

Remote memory access:

- Contention for memory controller

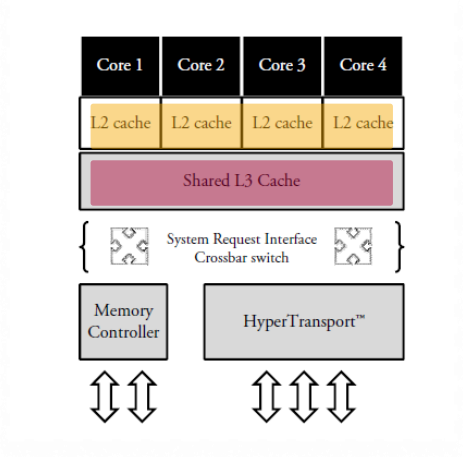


[3]

Challenges of Thread and Data placement

Local memory access:

- Contention for Shared Cache



[3]

DINO

= Distributed Intensity – NUMA Online

Key motivation:

- Contention aware UMA algorythms not suitable for NUMA
- UMA algorythms aim to avoid contention on local cache
- BUT disregard other factors (MC, IC, RL)

- Key novelty: eliminate superfluous thread migrations

DINO

- Based on Distributed Intensity:
  - Use miss-rate heuristics for detecting interfering threads
  - place to other memory domain

- Migrate threads memory along with thread
  - problem: miss-rates **can** signalize contention

- **DINO** does miss-rate classification
  and hence ignores small changes

DINO

- spreads memory intensive threads across memory domains
- accordingly migrates the corresponding memory pages

domain:	dom0	dom1	dom2	dom3
new_core:	0 1	2 3	4 5	6 7
new_class:	D t	D t	d t	d d
new_processID:	0 1	4 0	0 0	3 2
new_threadID:	0 1	7 3	4 5	6 2

- Up to 50% better than DI
- 20% better than default Linux Scheduler

Carrefour

- DINO does not address workloads by data-sharing between threads or processes

Carrefour - Mechanisms

- **Page co-location**
  = re-locate physical page to same node as the thread accessing it

- **Page interleaving**
  = evenly distributing pages across nodes (IC, MC)

- **Page replication**
  = placing copy of page on several memory nodes (MC, IC)
  - synchronization costs

- **Thread clustering**
  = colocate threads that share data on same node

Carrefour – Global Decisions

- **Step 1:**
  - Decide whether to enable Carrefour
  - Only for applications that generate substantial memory traffic
  - Memory access rate (MAPTU)

Carrefour – Global Decisions

- **Step 2:**
  - Decide whether to use **Replication**
  - Enough RAM?
  - Avoid synchronization overhead (memory read ration > 95%)

Carrefour – Global Decisions

- **Step 3:**
  - Decide whether to use **Interleaving**
  - Memory controller imbalance > 35% ?

Carrefour – Global Decisions

- **Step 4:**
  - Decide whether to enable **Co-location**
  - For pages accessed by single node
  - Thread clustering
  - Local access rate slightly less than ideal?

- Performance improvements up to 3.6 times

Carrefour Implementation

Instruction-Based Sampling (IBS) is a new profiling technique that provides rich, precise program performance information

carrefour-module

This module collects ibs samples and what should be done for each page (replication, migration, interleaving). It also has a simple runtime (carrefour.pl), but a more advanced runtime is available (https://github.com/Carrefour/carrefour-runtime).

Most of the options are available in include/carrefour_main.h, except IBS sampling frequency (src/ibs/ibs_main.c) and rbtree size (include/carrefour_rbtree.h).

If you don't have a kernel that supports replicating pages (see https://github.com/Carrefour/linux-replication), you must disable replication in include/carrefour_main.h.

Default options are those we used for the ASPLOS paper.

Notes

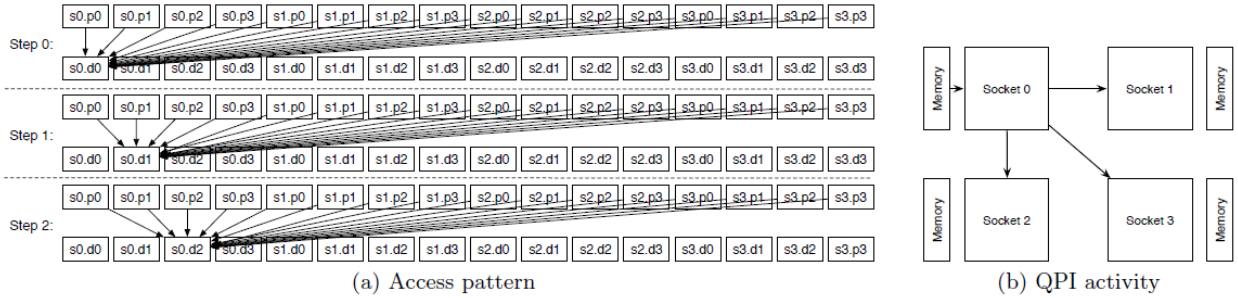
Since it uses IBS, it will only work on AMD processors. It has only been tested on AMD Family 10h. If your using carrefour-per-pid as a runtime, you MUST enable 'REPLICATION_PER_TID'. If your not, you MUST disable it.

Data Shuffling
- NUMA-awareness at an example

= threads exchanging roughly equal-sized pieces of data (partitions) among themselves

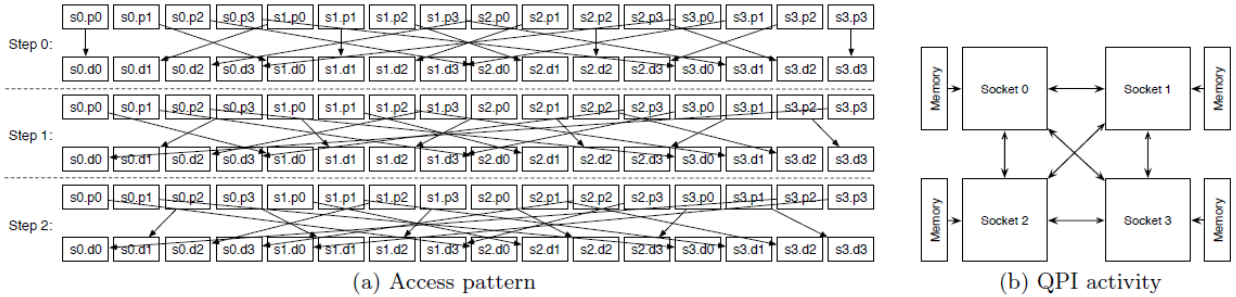
- Naive Shuffling:
  - For each thread i place data partition j on thread j

Data Shuffling
- NUMA-awareness at an example



(a) Access pattern (b) QPI activity

Figure 3: The uncoordinated, naive shuffling method (first three steps).



(a) Access pattern (b) QPI activity

Figure 4: The NUMA-aware, coordinated, ring shuffling method (first three steps). [4]

Data Shuffling
- NUMA-awareness at an example

- Coordinated Shuffling
  - Coordinate bandwidth usage
  - Up to 3x as fast
  - Can speed up join algorythm by 8%
- Thread migration for threads with small working sets
  - Can be up to 2x faster than Data Shuffling

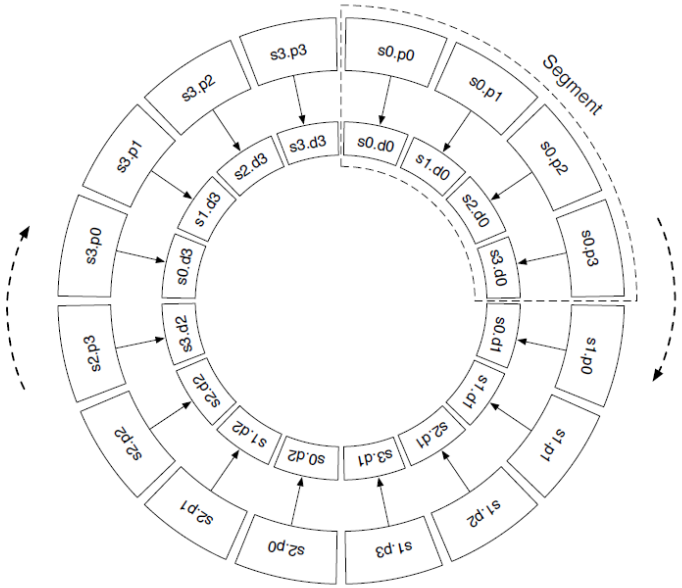


Figure 5: Ring shuffling, an example of a coordinated data shuffling policy.

[9]

Insights

Key performance factors (bottlenecks):

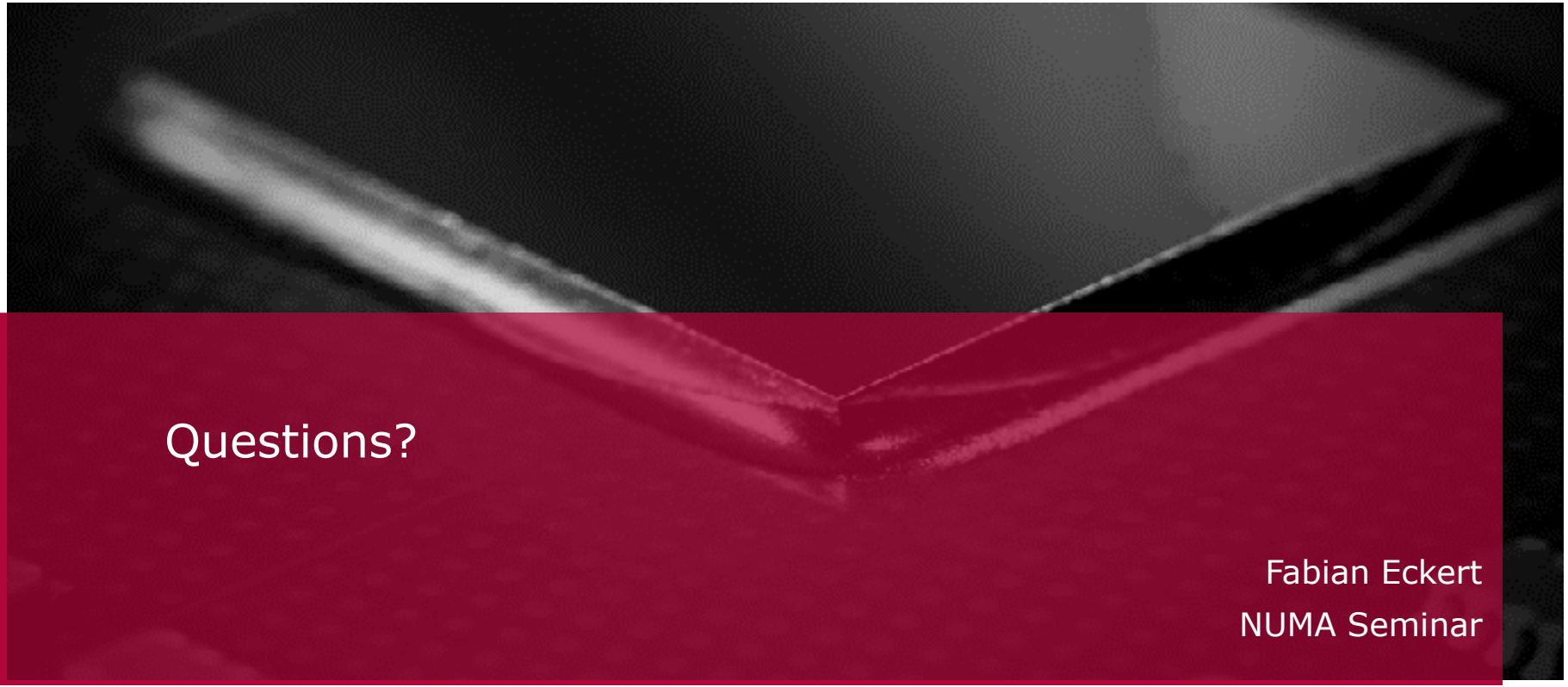
- Data locality
- Bandwidth limits
- finding a **balance** between local and remote memory accesses

- Different processor architectures require adaption of performance improvement strategies:
  - develop realistic models of the memory system that can guide operating system and compiler developers

References

- [1] (cc) picture from https://www.flickr.com/photos/downhilldom1984/6045320051/in/photostream/
- [2] Figure 1 in [5]
- [3] Figure 2 in [5]
- [4] Figure 3 and 4 of [7]
- [5] A Case for NUMA-aware Contention Management on Multicore Systems by Sergey Blagodurov, Sergey Zhuravlev, Mohammad Dashti, Alexandra Fedorova at Simon Fraser University
- [6] Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems by Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, Mark Roth at Simon Fraser University
- [7] NUMA-aware algorithms: the case of data shuffling by Yinan Li†, Ippokratis Pandis, Rene Mueller, Vijayshankar Raman, Guy Lohman
- [8] Memory System Performance in a NUMA Multicore Multiprocessor (Zoltan Majo, T. R. Gross)
- [9] Figure 5 of [7]

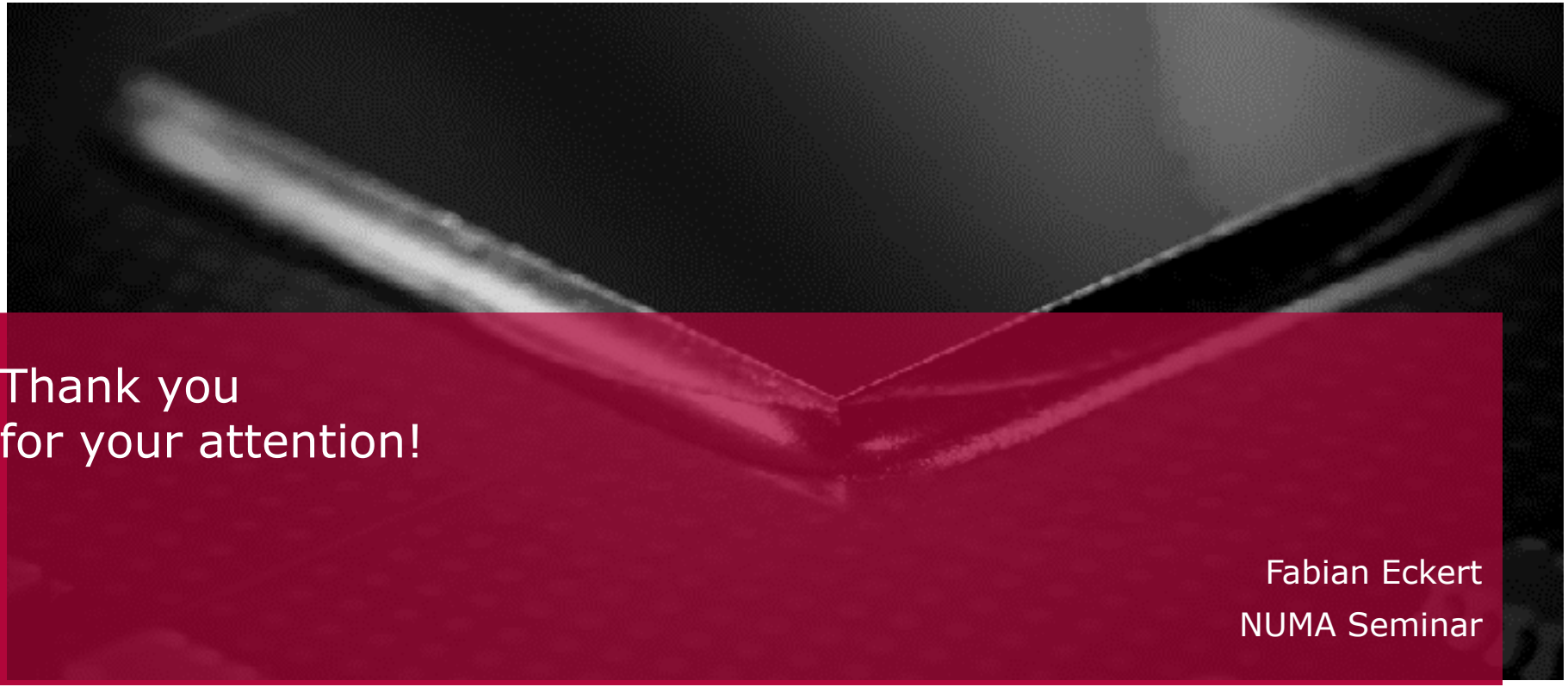
[2]



Questions?

Fabian Eckert

NUMA Seminar



Thank you
for your attention!

Fabian Eckert

NUMA Seminar