Communication in Shared Memory: Concepts, Definitions, and Efficient Detection

Matthias Diener, Eduardo Cruz, Marco Alves, Philippe Navaux

Federal University of Rio Grande do Sul, Porto Alegre, Brazil

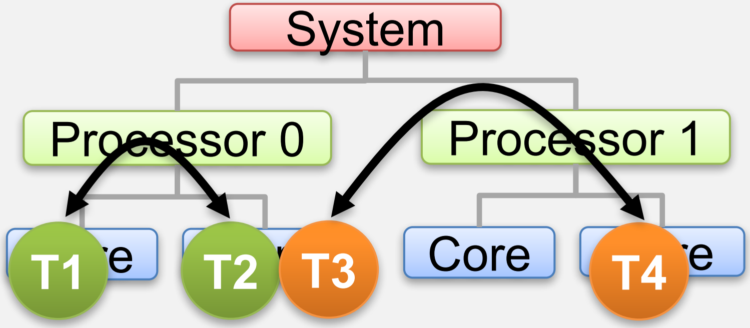
PDP 2016 – Heraklion, Greece

February 18th, 2016

Communication

- Parallel applications need to communicate
- Communication has a high overhead
  - Lower performance and energy efficiency compared to computation

- **Solution: Map tasks such that communication impact is reduced**



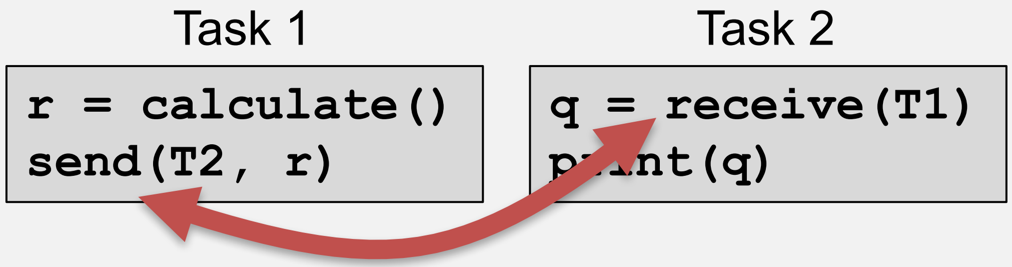
Require accurate information about communication behavior

- In shared memory
  - Communication through memory accesses
  - Mapping improves interconnection and cache usage

Communication in shared memory

Distributed memory

Explicit communication (MPI, Charm++, …)



Task 1

```
r = calculate()
send(T2, r)
```

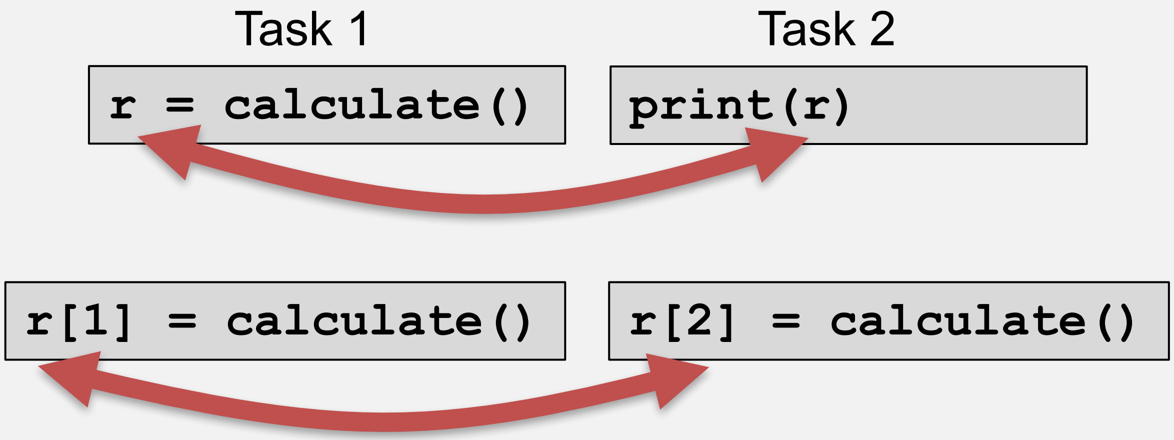
Task 2

```
q = receive(T1)
print(q)
```

- Function calls for communication
- Detection via function instrumentation

Shared memory

Implicit communication (OpenMP, Pthreads, …)



Task 1

```
r = calculate()
```

Task 2

```
print(r)
```

```
r[1] = calculate()
```

```
r[2] = calculate()
```

- Communication via memory accesses to shared data
- Detection via memory access traces

Communication in shared memory

- Detection of communication in shared memory not straightforward
  - Nevertheless, need accurate information to perform mapping
  - Focus: architectural impact of communication to perform the mapping
    - Sensitive to architectural parameters
      - Cache size, line size, …

How to describe implicit communication on the architectural level?

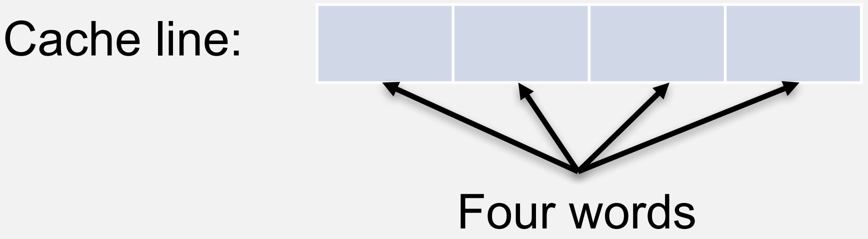
1. Types of communication
2. Detecting communication accurately and efficiently
3. Impact on performance gains

1. Communication types

True and false communication

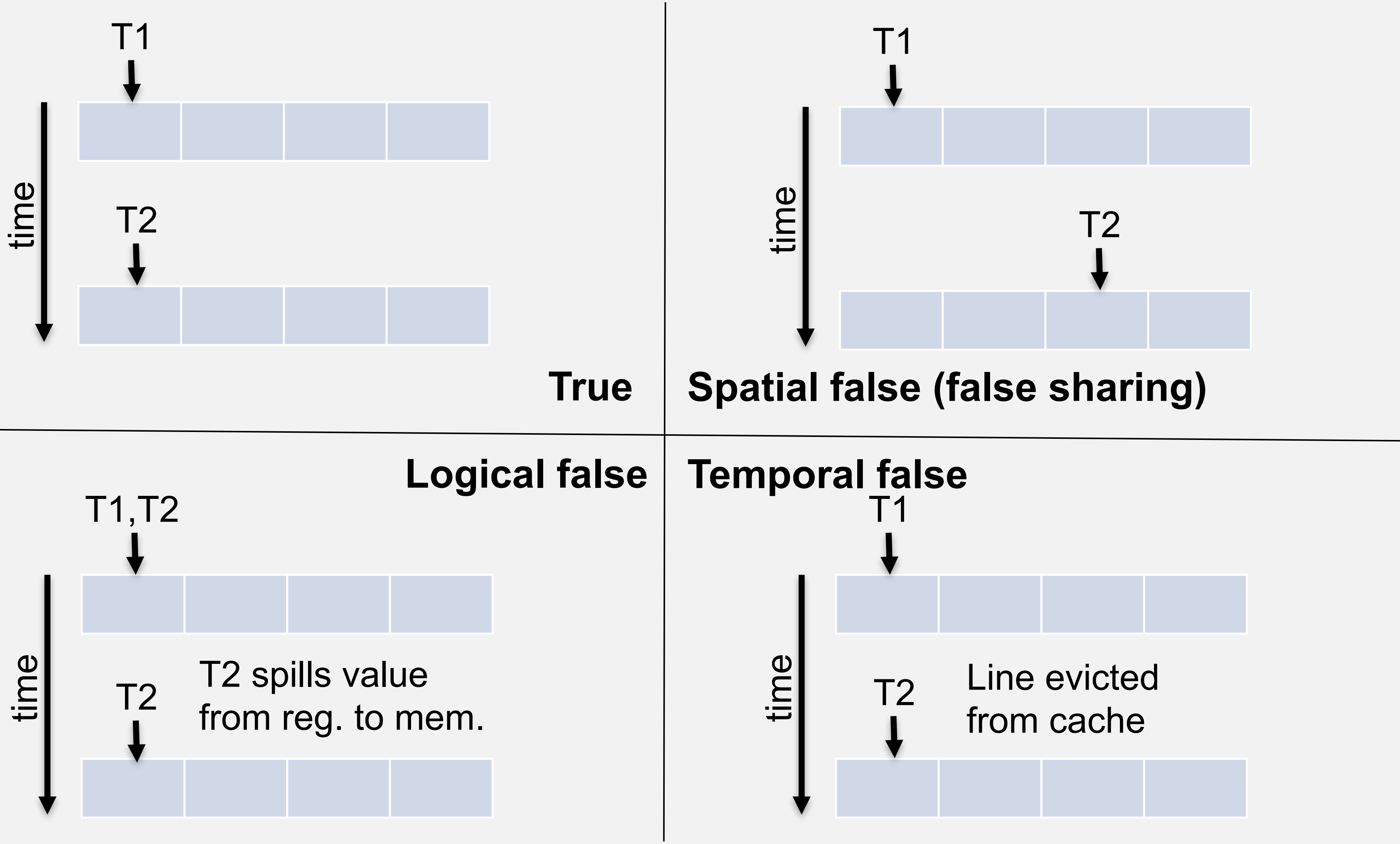
- In **explicit** communication, all communication is **true**
  - Intention to exchange data
- Not the case in **implicit** communication
  - Not every memory access to shared data is intention to communicate
  - ”Unintentional” (**false**) communication
    - **Spatial**, **logical, temporal**
  - Caused by operation of cache subsystem

Example:



Two tasks: T1, T2

True and false communication



True

Spatial false (false sharing)

Logical false

T2 spills value
from reg. to mem.

Temporal false

Line evicted
from cache

Communication events

- Impact on mapping
  - **True** comm.
  - **Spatial false** comm.
  - **Logical false** comm.

Consider for mapping

- No impact
  - **Temporal false** comm.

Do not consider for mapping

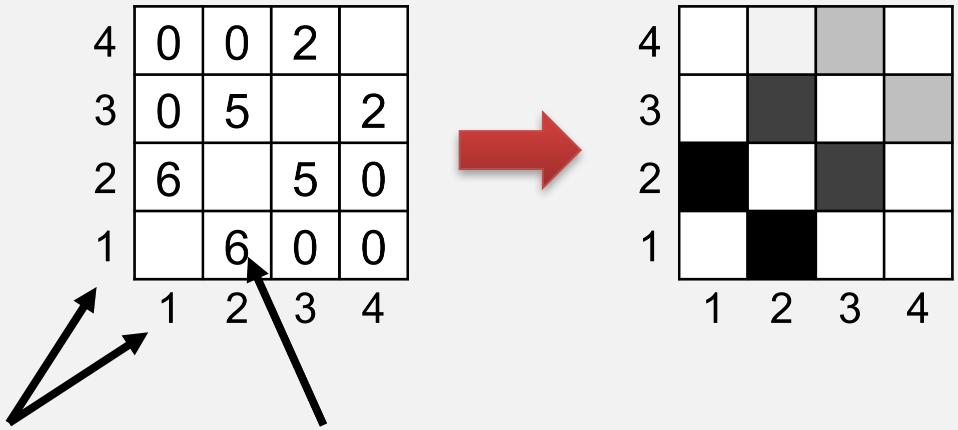
Communication event:
Two memory accesses from different threads to the same cache line while the cache line is not evicted.

Accurate definition of communication

Communication direction

- **Direction** of communication
  - **Explicit** communication: well-defined sender/receiver
  - No direction in **implicit** communication
    - E.g., two tasks read the same data

- Create **communication matrix** with communication events



4	0	0	2	
3	0	5		2
2	6		5	0
1		6	0	0
	1	2	3	4

4	□	□	░	□
3	□	▓	□	░
2	■	□	▓	□
1	□	■	□	□
	1	2	3	4

Communication matrix determines the mapping.

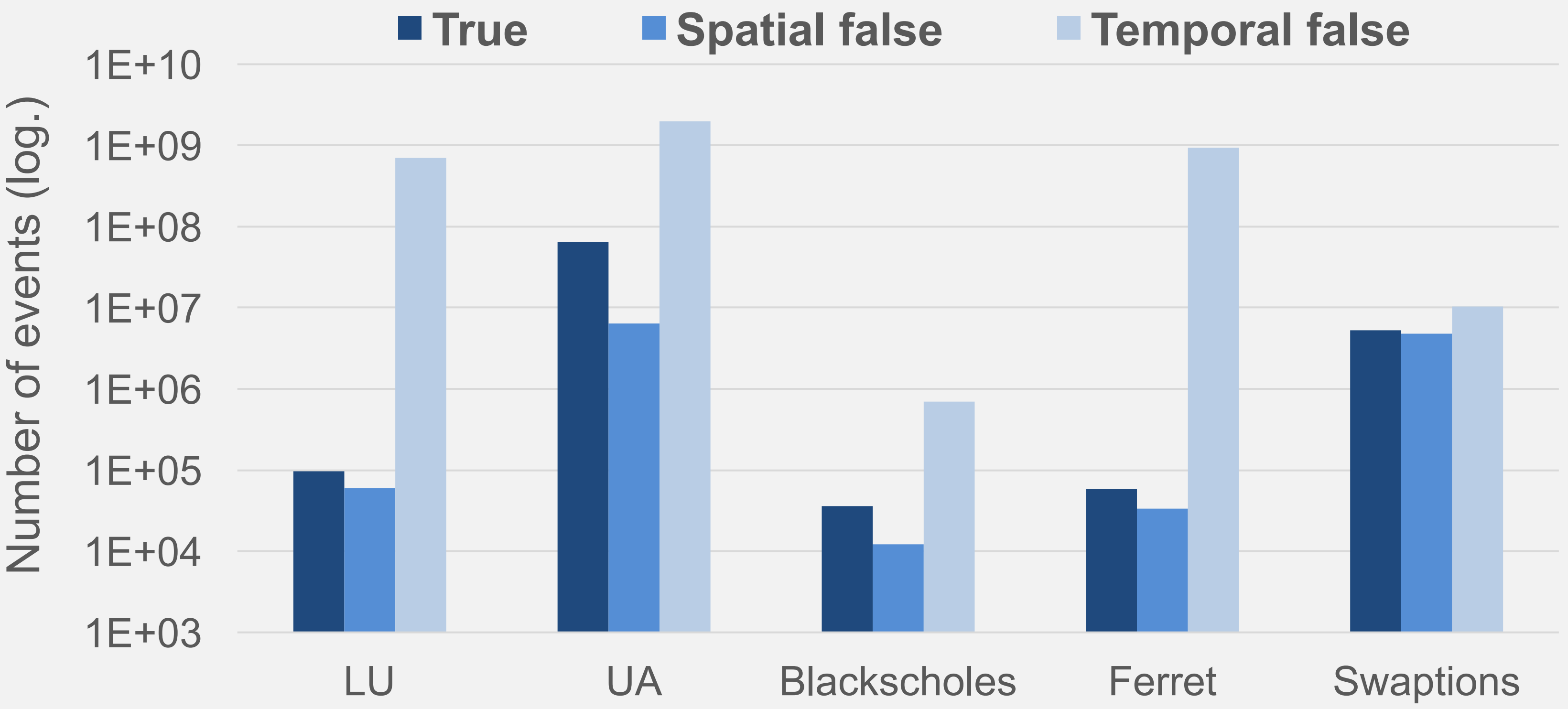
Communication behavior: methodology

- Benchmarks
  - NAS Parallel Benchmarks (NPB), OpenMP implementation; *A* input
  - PARSEC (Pthreads, OpenMP); *simlarge* input
  - Select 5 benchmarks with different behaviors
    - LU, UA
    - Blackscholes, Ferret, Swaptions

- Microarchitecture simulator
  - Sandy Bridge: 4x 8-core processors (2-SMT), 64 threads
    - 4x 18 Mbyte L3 cache (inclusive)

- Measure
  - **True**, **spatial false, temporal false** communication
    - No logical false communication, included in other types
    - Temporal false communication: simulate infinite cache

Results

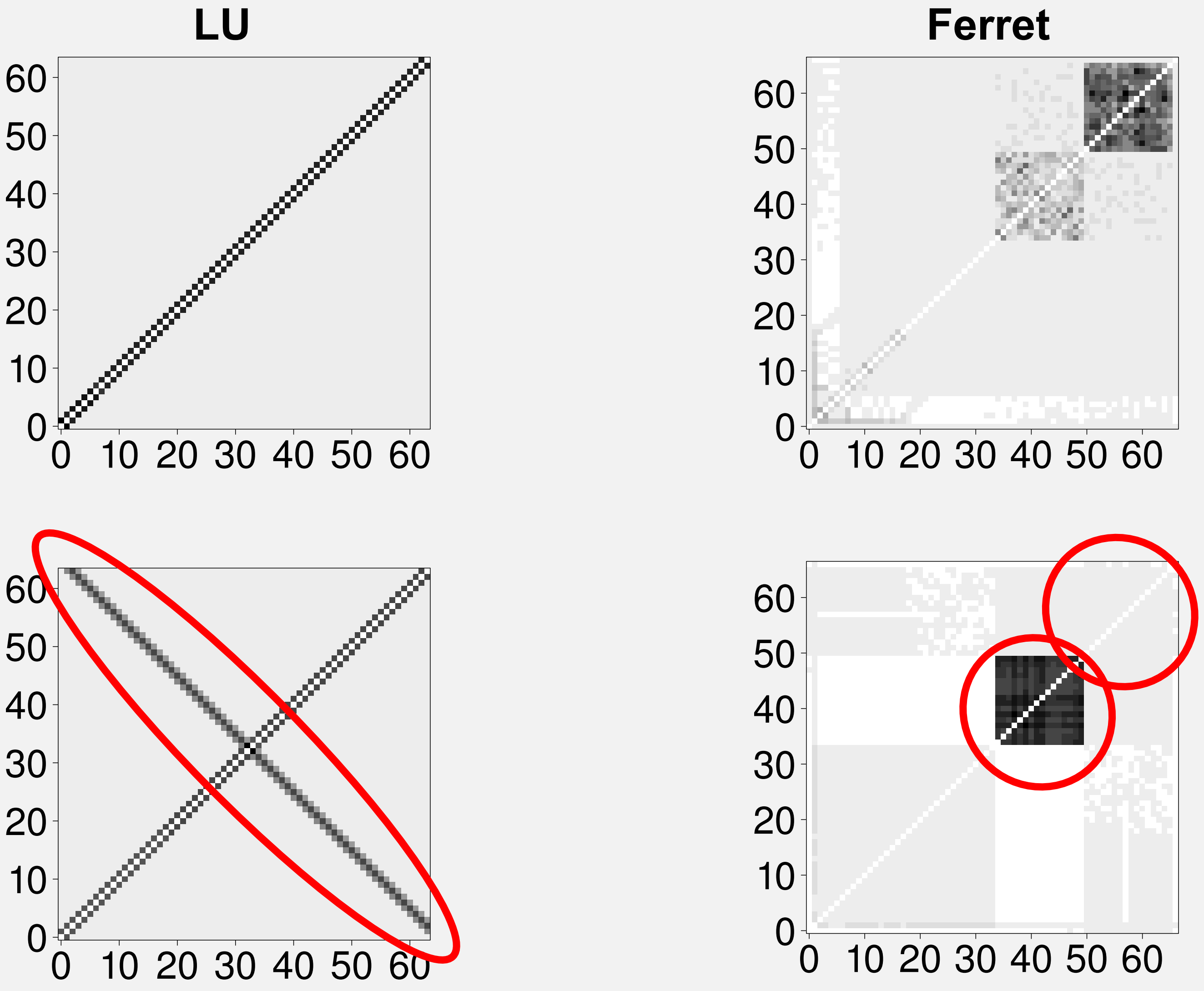


- Little spatial false communication
- High temporal false communication (esp. LU/Ferret)
  - Does it change the pattern?

Impact of temporal false communication

Accurate

Accurate + temp. false



2. More efficient detection

Relaxing the definition

- Disadvantages of previous mechanism
  - Full cache simulator
    - High runtime overhead
  - Cache line granularity
    - High storage overhead

- Provide more efficient (”**relaxed**”) definition

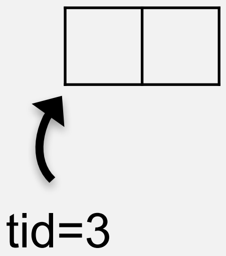
- Main idea: cache hierarchy mostly not relevant
  - Want to detect communication between **tasks**
  - Necessary to remove temporal false communication
    - Need simpler way to filter

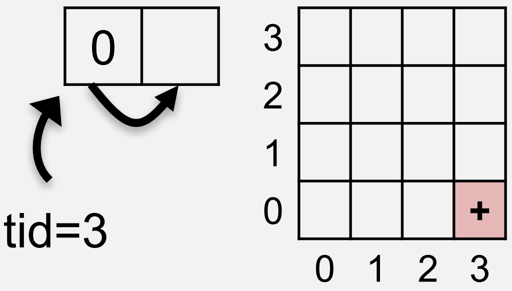
- Approximate temporal information via short queues

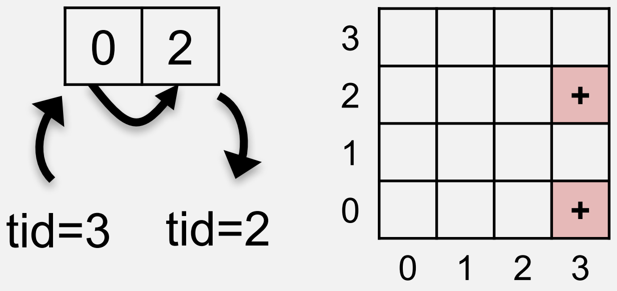
The relaxed definition

- Divide address space into memory blocks (block size >= line size)
- Maintain small FIFO queue (2 positions) of tasks that accessed the block
  - Detect communication within the queue
- For each memory access (**addr, tid**):
  1. Find memory block: **addr >> granularity**
  2. Number of other tasks that previously accessed block (<= 2)

No access	1 access	2 accesses
Enqueue **tid**. No communication.	Enqueue **tid.** Communication between old and new **tid**.	Enqueue **tid**, dequeue oldest. Communication between new **tid** and both old ones.







Overhead and design parameters

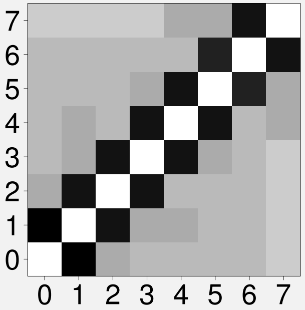
- Overhead
  - Constant time operation per memory access
  - No expensive operation (cache simulation, time stamps, …)

- Granularity/block size (default: 64 Bytes)
  - Larger: more spatial false communication, less storage required

Comparing accuracy

- UA benchmark (highest error), 8 threads

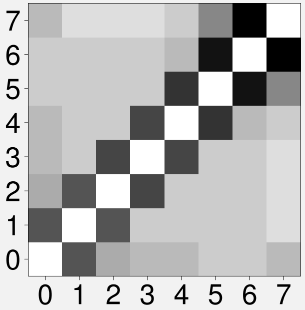
Accurate definition



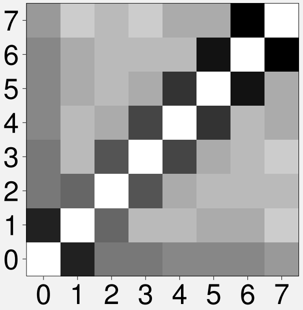
Relaxed definition

Granularity

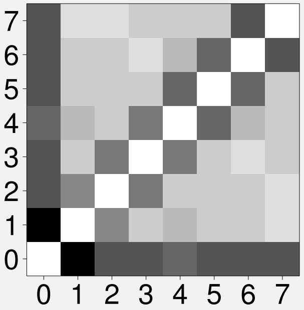
64 Byte



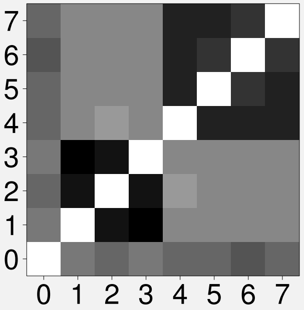
1 KByte



16 KByte



64 MByte

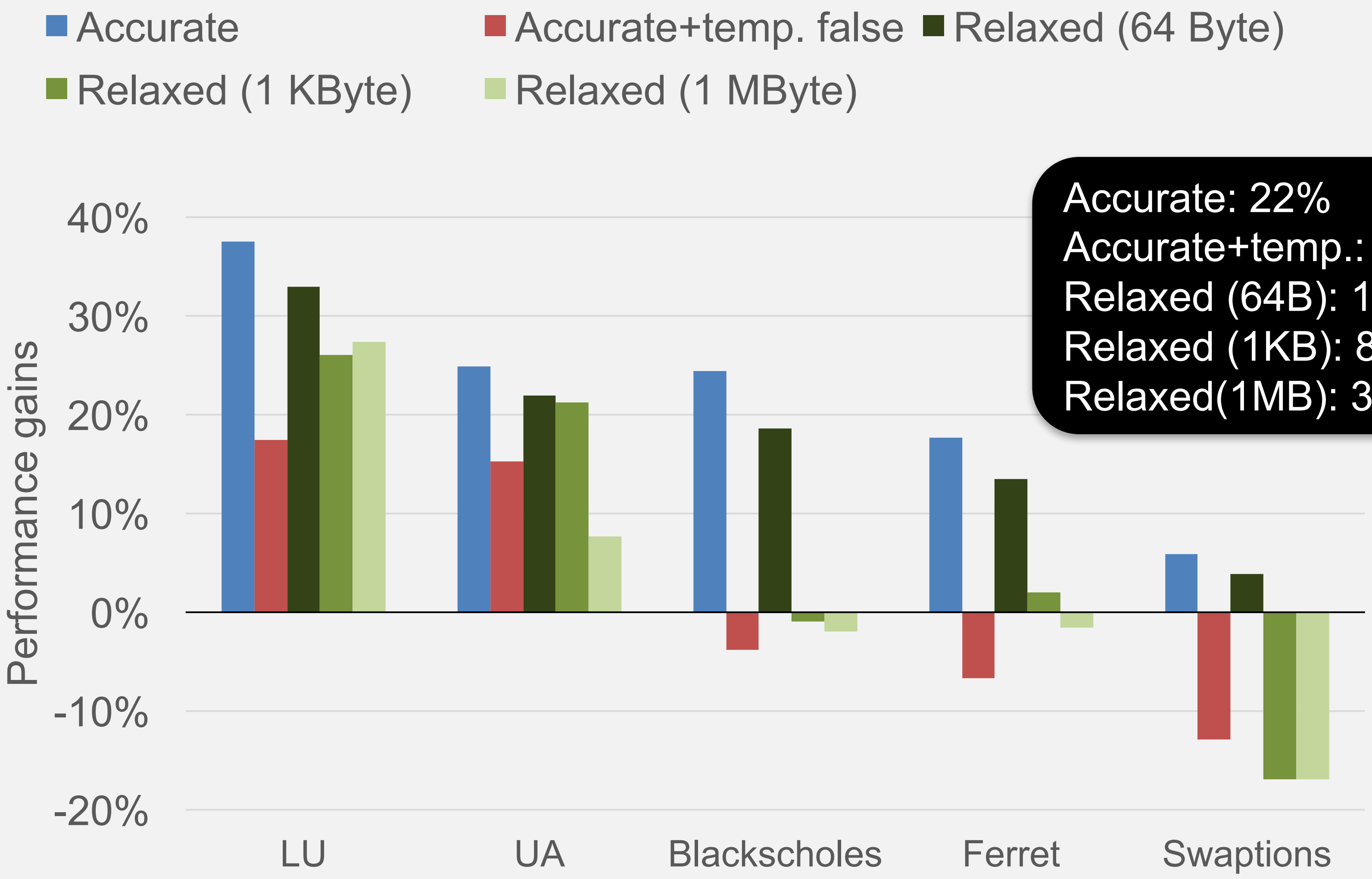


3. Performance gains

Performance experiments

- Methodology
  - Same benchmarks as before, 64 threads
  - Real machine: Intel Sandy Bridge, 4x 8-core processors (2-SMT)
    - L1/L2 caches per core, L3 cache per processor
  - Compare to default Linux 3.8 scheduler (CFS)
  - Thread mappings calculated with Scotch library
    - Static, no runtime overhead
- Compare execution time with mapping
  - Accurate definition
  - Accurate + temporal false communication
  - Relaxed definition with different granularities (64 Byte, 1Kbyte, 1Mbyte)
- Compare overhead accurate vs. relaxed communication detection

Performance results



Accurate: 22%
Accurate+temp.: 6%
Relaxed (64B): 19%
Relaxed (1KB): 8%
Relaxed(1MB): 3%

Comparing detection overhead

Overhead

Slowdown vs. normal execution

Mechanism	LU	UA	Blackscholes	Ferret	Swaptions
Accurate	5944 x	2860 x	1771 x	5304 x	6157 x
Relaxed	49 x	72 x	113 x	39 x	148 x

Available at
CacheSim: `http://github.com/matthiasdiener/CacheSim`
Numalize: `http://github.com/matthiasdiener/numalize`
Both are based on Intel Pin.

Conclusions

Conclusions

- Accurate view of communication remains important challenge for thread mapping in shared memory
  - Can have huge impact on gains
  - Main challenge: temporal false communication

- Solutions: cache simulator, queue-based approximation

- Approximation solution
  - Similar gains as cache simulator (22% vs. 19%)
  - Substantially lower overhead (up to 100 times faster)

- Future work: impact of sampling

Thank you!

Communication in Shared Memory: Concepts, Definitions, and Efficient Detection

Matthias Diener, Eduardo Cruz, Marco Alves, Philippe Navaux

Federal University of Rio Grande do Sul, Porto Alegre, Brazil

mdiener@inf.ufrgs.br

CacheSim: `http://github.com/matthiasdiener/CacheSim`
Numalize: `http://github.com/matthiasdiener/numalize`

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

PDP 2016 – Heraklion, Greece

February 18th, 2016

.inf
INSTITUTO
DE INFORMÁTICA
UFRGS