Lockality: a Scalable Synchronization Lock for MPSoCs

Frédéric Rousseau

TIMA lab – University of Grenoble Alpes

A join work with Maxime France-Pillois et Jérôme Martin
CEA LETI

Need of locks, definition and implementation

* Parallel programming requires synchronization mechanisms. A main one is the synchronization lock ensuring the mutual exclusion between processes

* A lock enables to protect a shared ressource (memory, peripherals,...) from concurrent accesses in a multi-threaded environment

* A lock is usually implemented by a distinct memory element accessed by the threads through an atomic primitive

Motivations

* Improving performances of locks
  * In MPSoC (shared memory, clustered), the access to local lock should be promoted, even if several threads access the lock

* Verified hypothesis: A core that releases a lock is usually the following one to take it back
  * Assumption stated by Kuo and all, and Rutgers and all (simulation)
  * Verified by us (HW emulation)

2 SPLASH2 benchmark applications

Application	Reuse rate by core	Reuse rate by cluster
Choleski	~63%	~66%
Water-NSquared	~68%	~75%

Lock reuse ratio of 64-thread applications
Running a on 64-core TSAR MPSoC (16 clusters)

Chen-Chi Kuo, J. Carter, and R. Kuramkote. *MP-LOCKs: replacing H/W synchronization primitives with message passing*, International Symposium in High-Performance Computer Architecture, p 284–288, 1999.

J.H. Rutgers, M.J.G. Bekooij, and G.J.M. Smit. *An efficient asymmetric distributed lock for embedded multiprocessor systems*, International Conference on Embedded Computer Systems (SAMOS), pages 176–182, July 2012.

Challenges

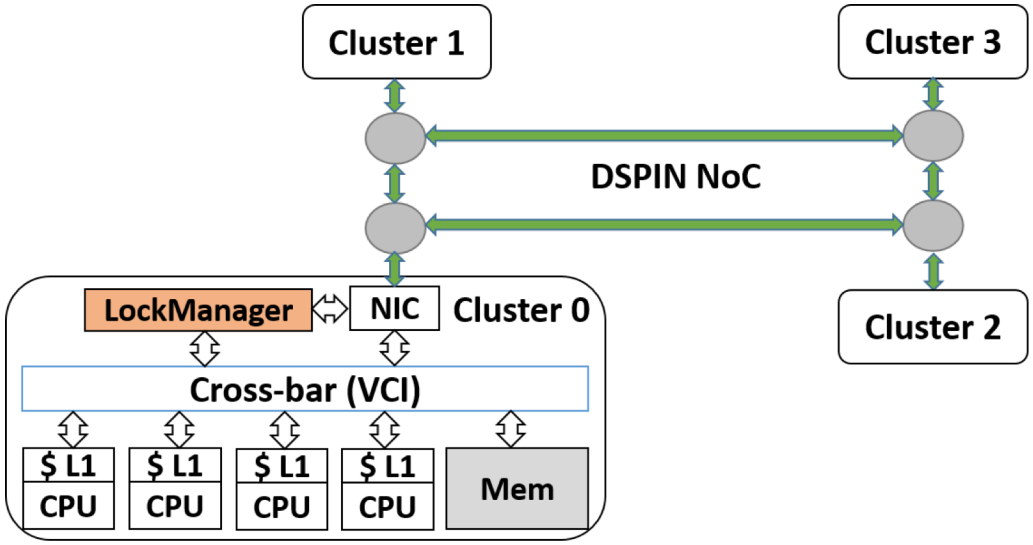
* A static lock placement in memory is non-optimal
  => A dynamic re-homing of locks should improve performances

* Existing solution
  * Kuo and all: dynamic re-homing of locks
    * A centralized solution, based on message passing
    * Weaknesses:
      * Software solutions with different processing delays
      * No scaling-up due to the centralized solution

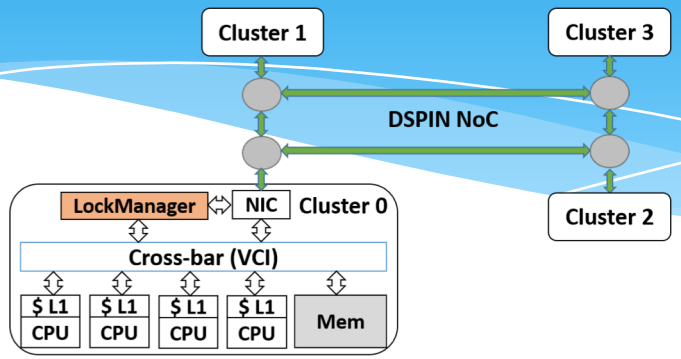
* Our new solution: Lockality
  * A fully hardware decentralized lock re-homing solution

Main idea of Lockality

* Each cluster implements a (HW) Lock Manager
* Lock Managers exchange data between them through a Network on Chip (NoC) – use of broadcast capabilities

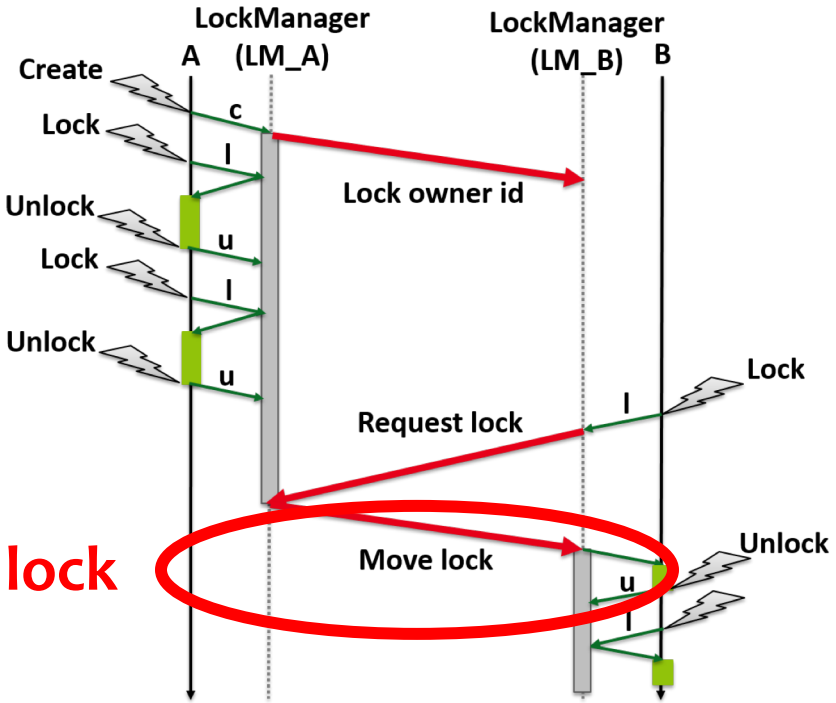


Example of use



Thread A

```
. . .
Mutex_id = mutex_create();
. . .
mutex_lock(Mutex_id, … );
. . .
mutex_unlock(Mutex_id, … );
. . .
mutex_lock(Mutex_id, … );
. . .
mutex_unlock(Mutex_id, … );
. . .
```



Thread B

```
. . .

mutex_lock(Mutex_id, … );
. . .
. . .
mutex_unlock(Mutex_id, … );
```

Re-homing of lock

Challenges in Lockality protocol

* All Lock Managers know at any moment the current manager for a given lock
  * except during the transmission of the broadcast frame, which requires specic handling to prevent race conditions

* Lockality requires a specific communication protocol. The protocol ensures:
  * Coherency and consistency of lock managers
  * A lock is owned by only one lock manager
  * The granting of a lock to at most one thread at a time.
  * All lock requests obtain an answer
  * Fair granting of a lock
  * . . .

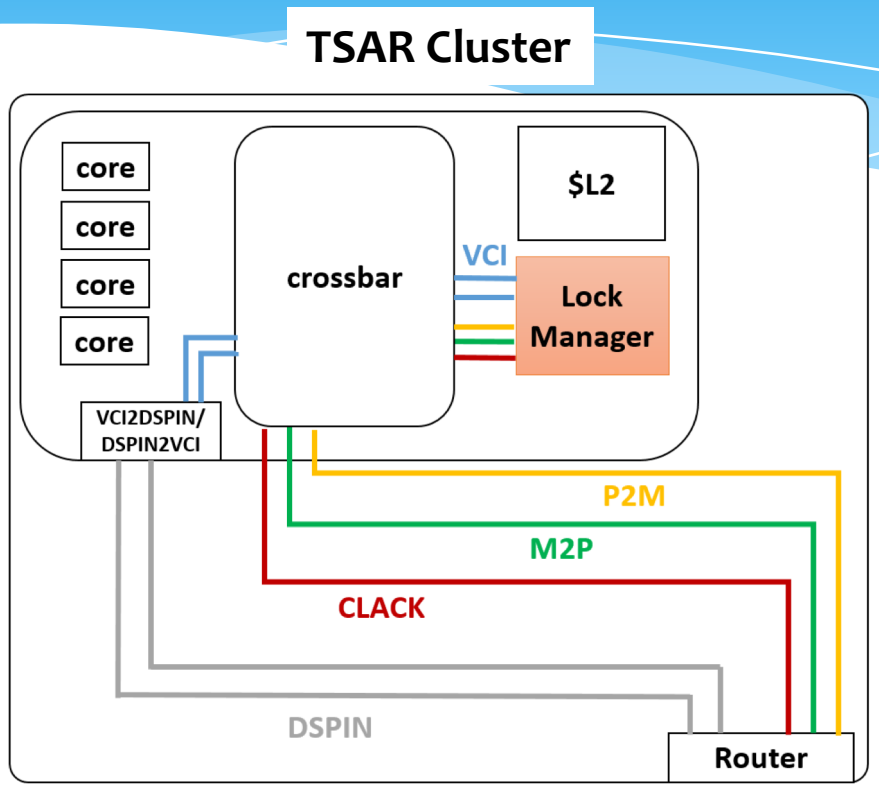
French patent: n° 1858803
Title : Lock Manager for Multi-core Architectures

Challenges in Lockality protocol

* Deadlock free solution
  * We first defined all frames needed to implement the protocol
  * We studied the dependencies between frames
  * We enumerate all dependencies, and we isolated in particular the frames that require the completion of induced frames before completing their function.

→ We concluded that at least **three** independent communication **channels** (physic or virtual) are required to ensure the deadlock free property of our solution.

More details of the HW architecture

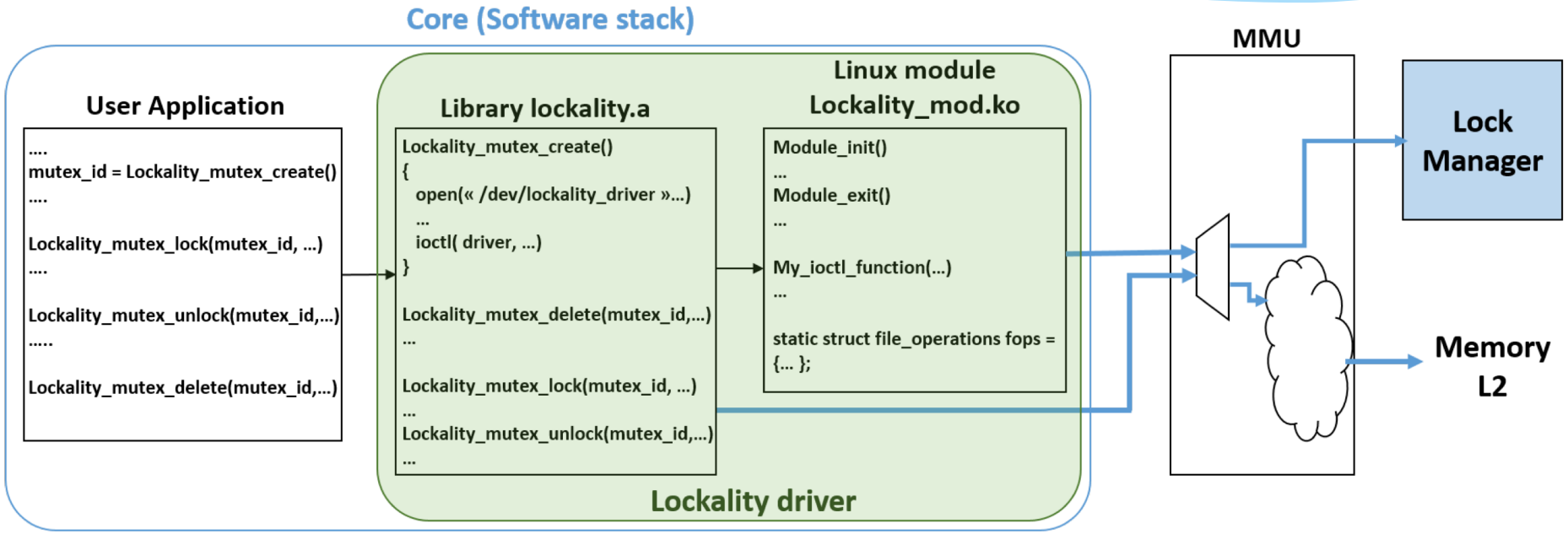


3 channels: reuse of the virtual cache coherency channels tagged "M2P", "P2M" and "CLACK"

NO communication infrastructure overhead when implementing Lockality.

More details of the SW architecture

* Software stack



Evaluation platform

* TSAR MPSoC architecture
  * a fully coherent shared memory MPSoC platform
  * a clustered MPSoC architecture based on a NoC
  * each cluster is mainly made of four MIPS32 processors with a private L1 cache
  * L2 cache is also a shared memory segment. Each L2 memory is designed to cache a section of the global memory and the L2 cache of a cluster can be accessed by cores inside and outside the cluster
* Veloce2 Quattro emulator
  * a full Register Transfer Level system model, with a cycle accurate precision
  * a port of Linux kernel 4.6 and µClibc
* We bind only one thread on each core to avoid interferences from the scheduling policy

Performances

* Comparaison with the Posix Pthread library implementation

Lock implementation	Local lock	Distant lock
Pthread	48 cycles	136 cycles
Lockality	2 cycles	34 cycles

Median delays of physical lock acquisitions

Delay between the emission of the lock request VCI frame from the core to the Lock Manager in the case of Lockality or to the L2 cache in the case of the Pthread Mutex, and the reception by the core of the frame granting the lock access.

Gain brought by Lockality: about 50 to 100 cycles.

Pthread library implements a more complex locking functions that supports recursive mutex.

Implementation	Local lock	Distant lock
Pthread : pthread_mutex_lock	373 cycles	377 cycles
Lockality : lockality_mutex_lock	115 cycles	121 cycles
Pthread : pthread_mutex_unlock	726 cycles	795 cycles
Lockality : lockality_mutex_unlock	207 cycles	-

Delays for the locking functions

Performances

* 16 lockality locks implemented
* A 64 cores architectures (16 clusters of 4 cores)

Application	Cholesky	Water-NSquared
Without *Lockality*	$30\,160 \times 10^3$ cycles	$209\,146 \times 10^3$ cycles
With *Lockality*	$26\,987 \times 10^3$ cycles	$205\,481 \times 10^3$ cycles
Gain	~10, 5%	~1, 7%

2 SPLAH2 benchmark applications

Module	Area	Number of cells
TSAR Cluster (without memories)	$559972\mu m^2$ (1)	1, 180, 837
Lock Manager	$12\,571\mu m^2$	53, 122
HW impact	2.24%	4.50%

HW implementation in 22 nm technology

Impact of Lockality difficult to determine !

- Effect on network and memory contention
- Different cache memory accesses

Very small impact in terms of area

Conclusion

* Lockality: Re-homing of locks solution
* Good performance compared to Posix Pthread library
  * Even if the Posix Pthread implements more complex locks
* Difficult to evaluate the overall gain in running applications
  * Modification of network and memory contention
  * Modification of caches accesses
* Patent of Lockality

Lockality: a Scalable Synchronization Lock for MPSoCs

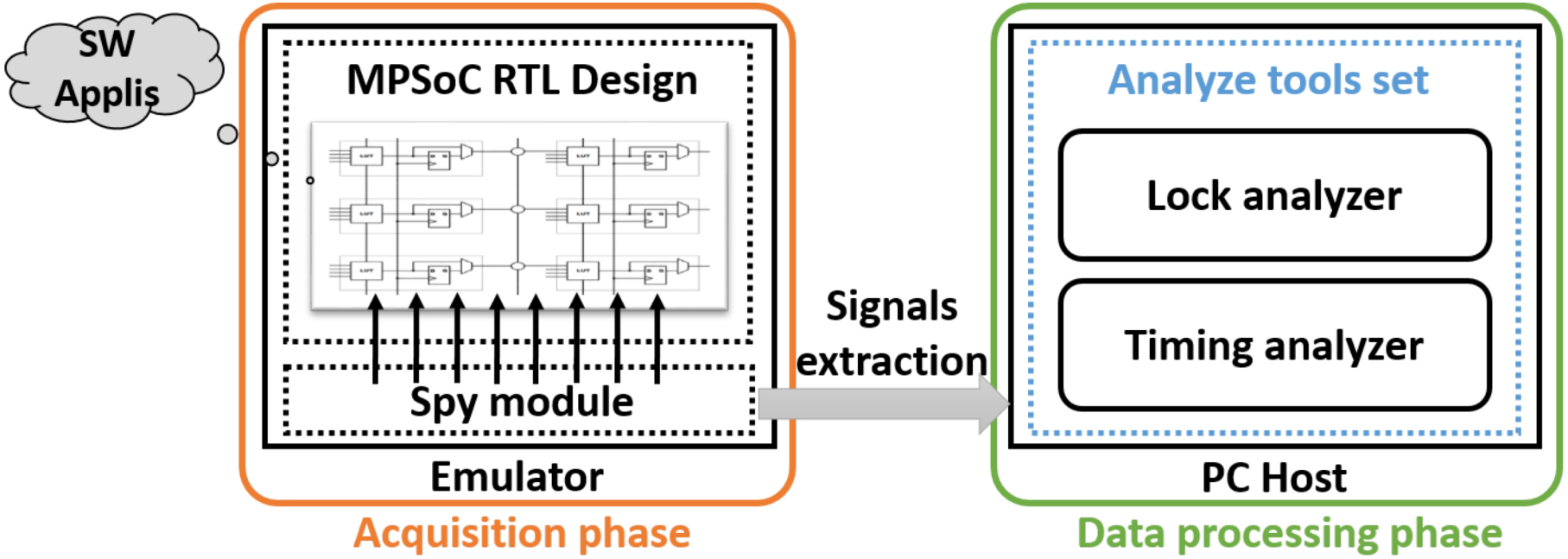
Frédéric Rousseau

TIMA lab – University of Grenoble Alpes

A join work with Maxime France-Pillois et Jérôme Martin
CEA LETI

Measurement tool-chain

* A non intrusive measurement tool-chain



Lockality protocol

* Set of the frames defined for the 3 required channels

Name	Description
LOCK_INIT	Request to a *Lock Manager* to instantiate a lock with the ID passed in parameter of the frame
LOCK_INIT_ACK	Acknowledgement of lock instantiation
LOCK_DEL	Request for removal of the lock whose ID is passed in parameter
LOCK_REQ	Request for the lock granting
LOCK_MOV	Information of a lock move (lock access grant + transfer of partial ownership) **broadcast**
LOCK_LOCKED	Information of busy lock
LOCK_MOVED	Acknowledgement of full transfer of lock ownership
LOCK_NEXT_CLUSTER	Request for the enqueuing of the *Lock Manager* ID to the list
LOCK_NEXT_ACK	Acknowledgment of LOCK_NEXT_CLUSTER processing
LOCK_MOV_ACK	Acknowledgment of lock move by a *Lock Manager*
LOCK_ERR	Error: cannot perform the requested operation