Georgia Institute of Technology

A System-on-a-Chip Lock Cache with Task Preemption Support

By

Bilge S. Akgul, Jaehwan Lee and Vincent J. Mooney

Georgia Institute of Technology
School of Electrical and Computer Engineering

Outline

- Introduction
- Background
- Lock Synchronization Problems
- Our Methodology
- Hardware and Software Designs
- Experiments and Results
- Conclusion

Introduction

- Multi-processor shared memory SoC
- Intertask/interprocess synchronization
- Lock synchronization overheads
  - Lock delay, lock latency
  - Memory bandwidth consumption
- Aim:
  - Reduce overheads
  - Improve Real Time (RT) predictability

Background

- Critical Section
  - Code section where shared data between multiple execution units is accessed
  - E.g., multiple readers and multiple writers
  - A lock is necessary to guarantee the consistency of shared data (e.g., global variables)
- Lock Delay
  - Time between release and acquisition of a lock
- Lock Latency
  - Time to acquire a lock in the absence of contention

Problems

- Ensuring mutual exclusiveness
- Communication bandwidth consumption
- Eliminate busy-wait problems
  - Busy-wait: If lock is busy, processors spin on memory bus
- Effective lock hand off necessary
  - Fair
  - Predictive

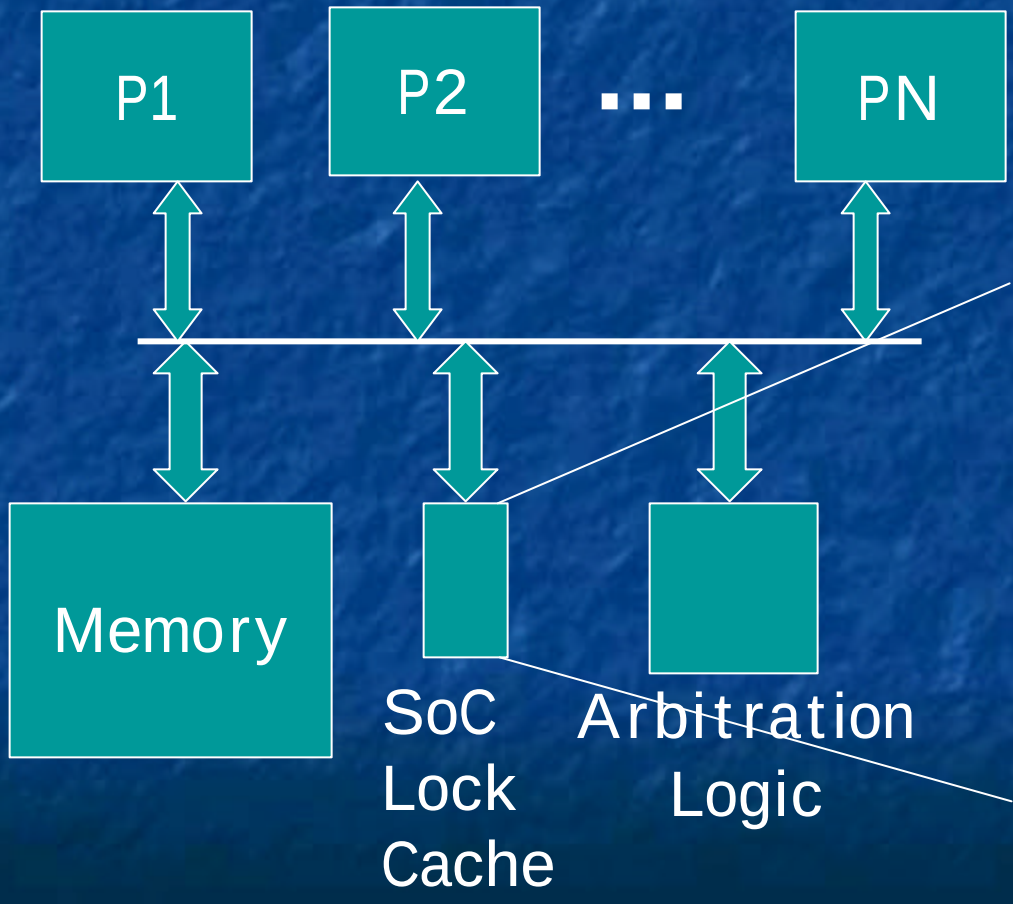
Previous Work

- Spin-lock alternatives (Anderson ’90)
  - Spin-on-read (spin on cache), delays in spin-loops
- Queue based software locks
  - Array based queuing (Anderson ’90)
  - MCS locks (Mellor-Crummey, Scott ‘91)
  - LH and M locks (Ladin, Hagerston, Magnusson ’94)
- Queue based hardware locks
  - QOLBY (Kagi ’99) – makes use of collocation
- Cache-based locks (Ramachandran’96)
  - Memory consistency model
  - New cache design, extra cache states for locks

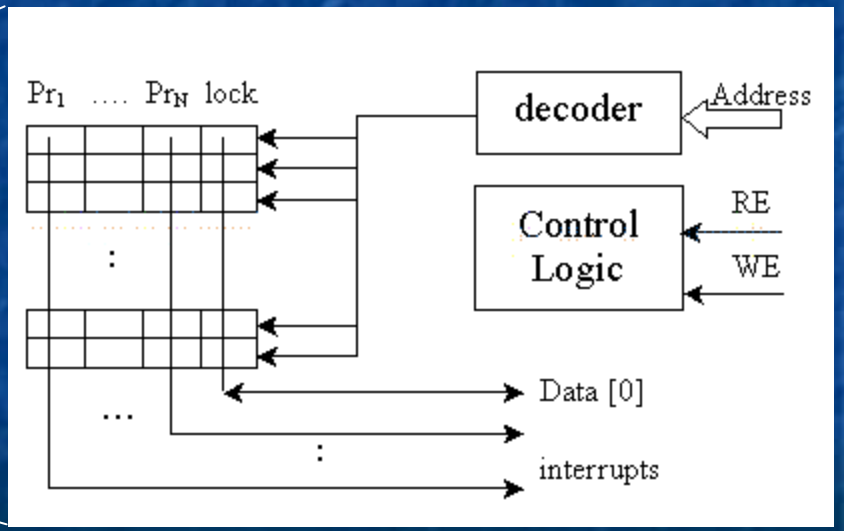
Methodology

- Custom hardware unit: SoC Lock Cache
- Utilize advantages of SoC Design
- Short Critical Sections covered in DATE ’01
- Critical Sections may be long or short
- Support preemption of tasks when necessary
- Hardware-interrupt triggered notification
- Lock requests handled on a processor-by-processor basis
- Separate the lock variables according to the critical section lengths

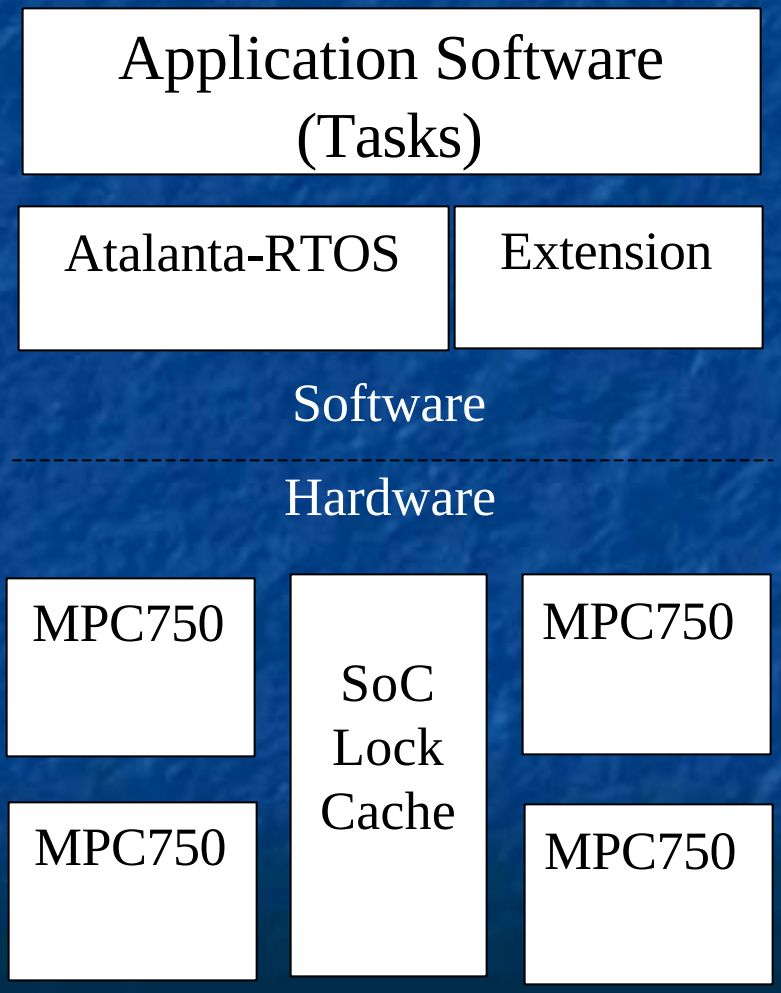
SoC Lock Cache Hardware Mechanism



SoC Lock Cache

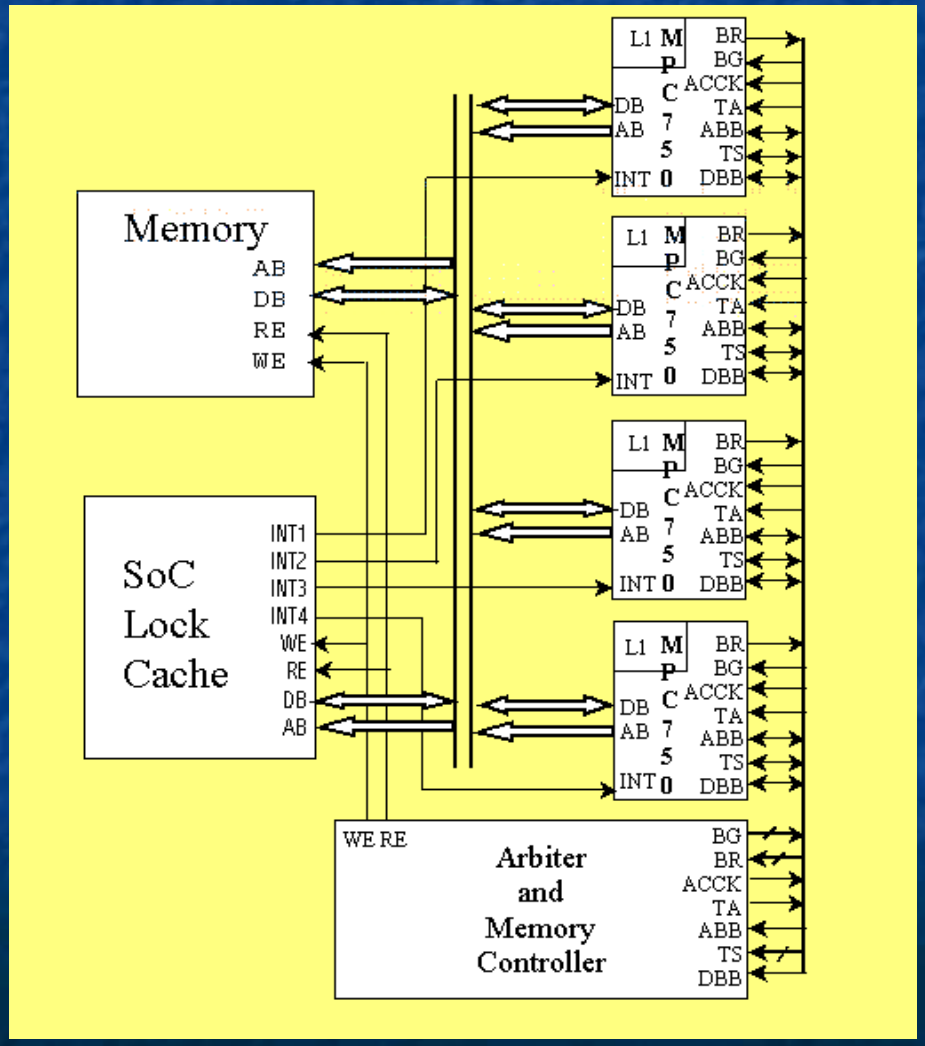


Methodology



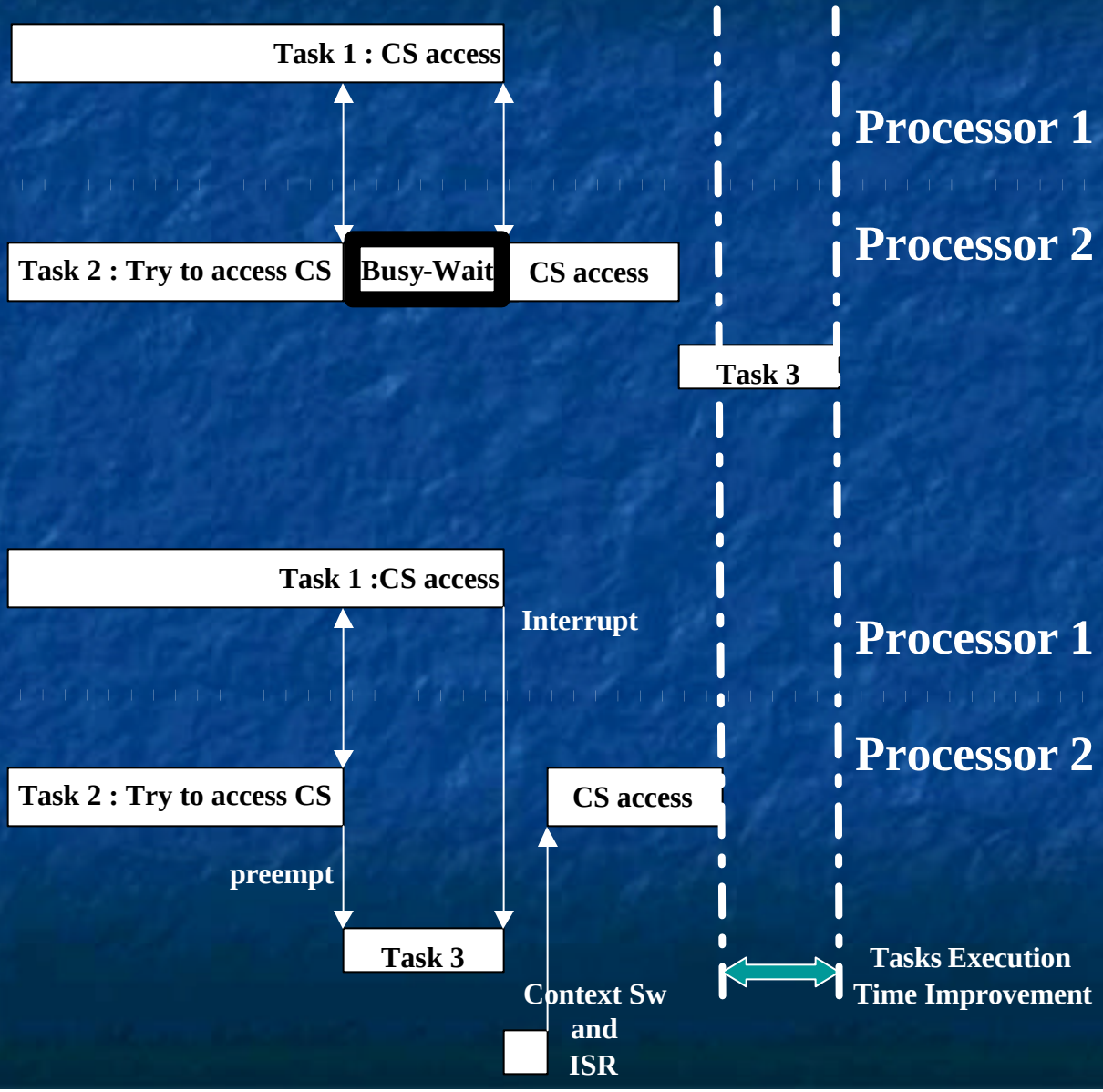
- Multiple application tasks
- Atalanta-RTOS
- Multi-processor set-up with MPC750s
- SoCLC provides lock synchronization among processors

Hardware Simulation Set-up



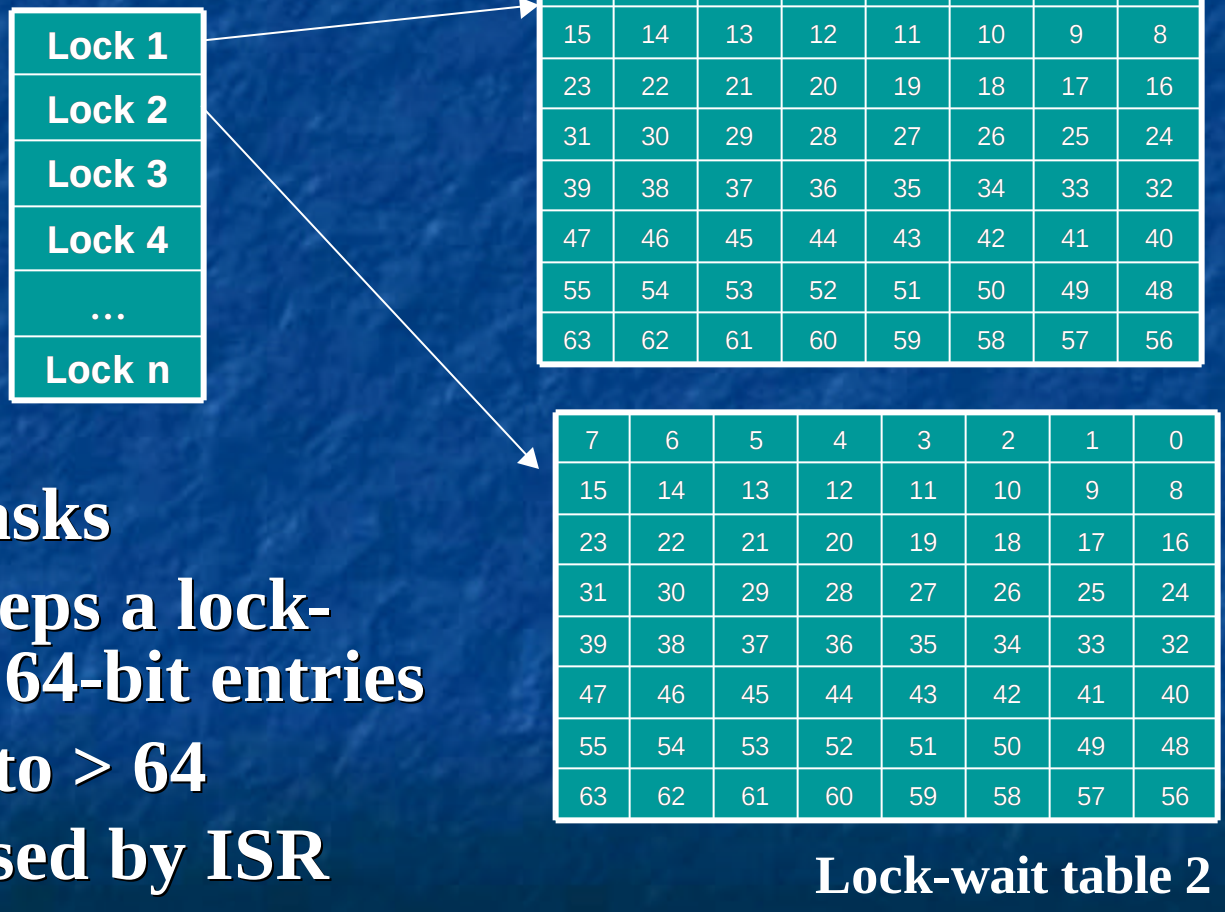
- Seamless CVE from Mentor Graphics
- 4 MPC750s
- SoC Lock Cache Unit (SoCLC)
- Shared Memory
- Interface Logic

Software



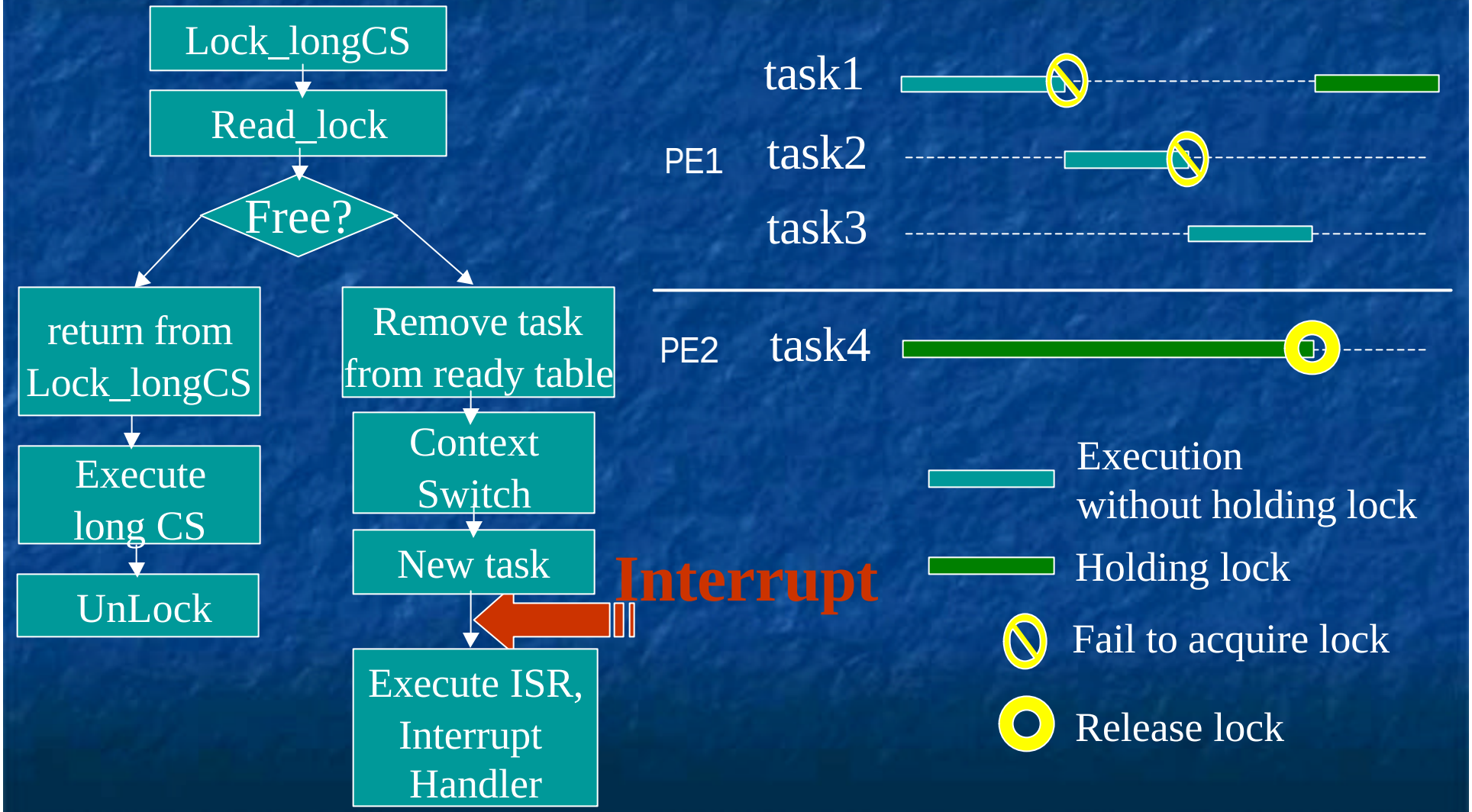
In the case of long Critical Sections, non-preemptive synchronization causes inefficient CPU utilization among tasks.

Software



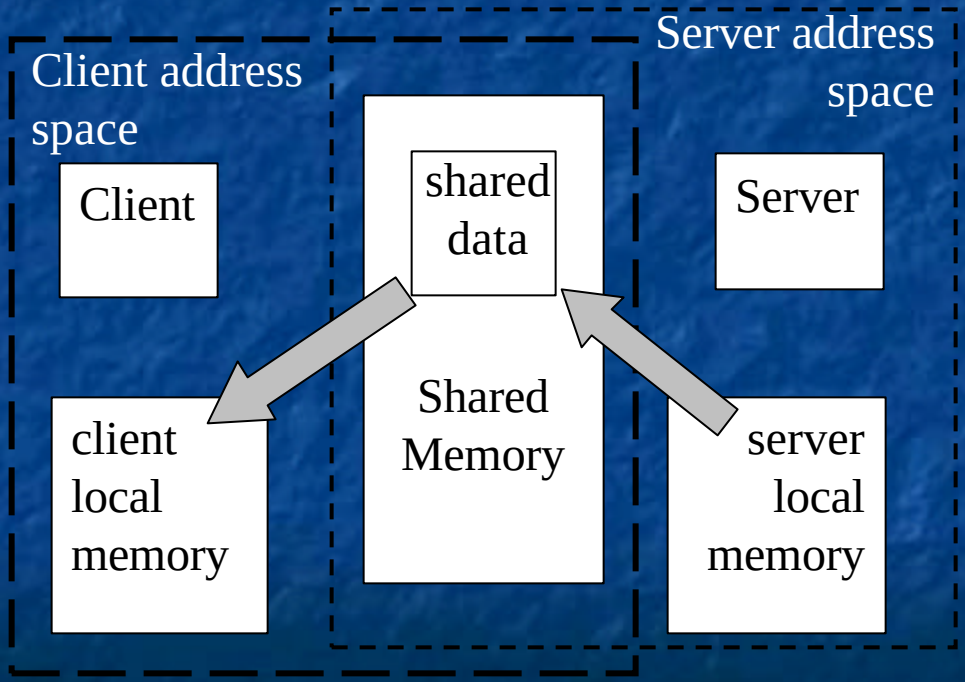
- Assume 64 tasks
- Each lock keeps a lock-wait table of 64-bit entries
- Expandable to > 64
- Tables accessed by ISR

Software


Lock_longCS
Read_lock
Free?
return from Lock_longCS
Execute long CS
UnLock
Remove task from ready table
Context Switch
New task
Execute ISR, Interrupt Handler
Interrupt
PE1
task1
task2
task3
PE2
task4
Execution without holding lock
Holding lock
Fail to acquire lock
Release lock

Experiments

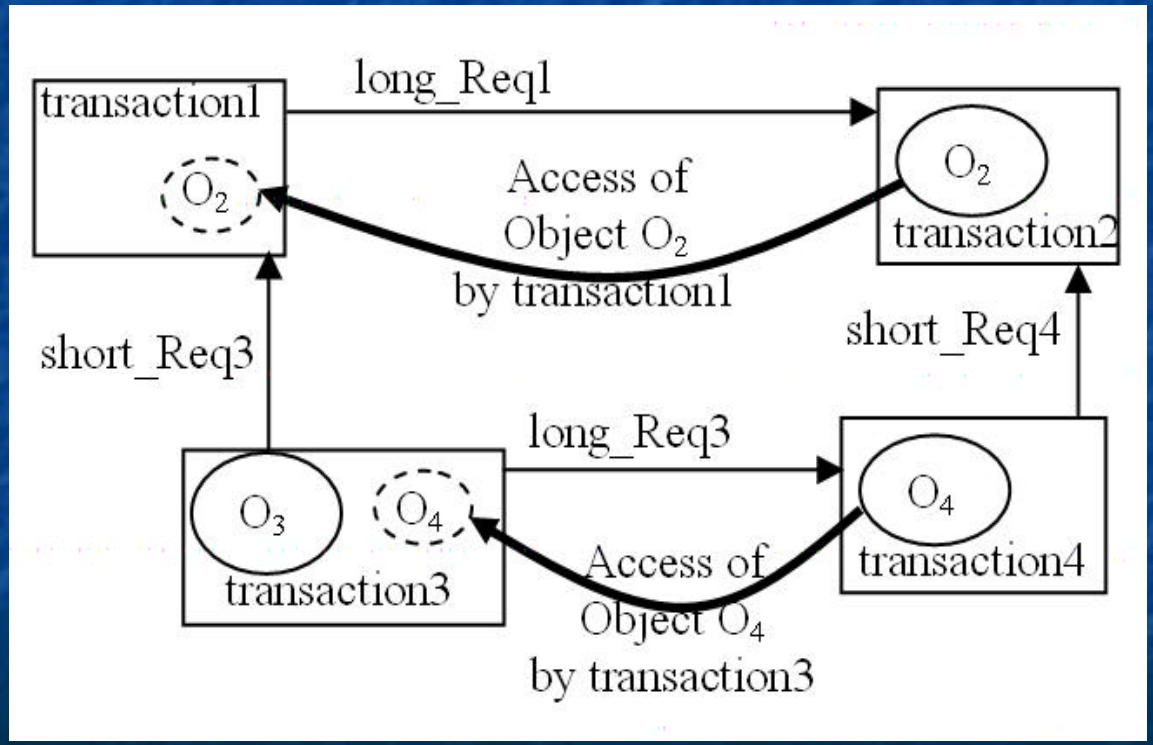
Database Application
(database object flow)



- **With Atalanta RTOS**
- **With 4 MPC750s**
- **Database Example application (run with 40 tasks)**

Experiments

Example Database Application Transactions



Observed Performance Improvement with Lock Cache Unit

- **100% speedup in lock delay**
- **32% speedup in lock latency**
- **27% speedup in total execution time**

Experiments

Long CS lock results

	Without SoCLC	With SoCLC	Speedup
Lock Latency (clk cycles)	1200	908	1.32x
Lock Delay (clk cycles)	47,264	23,590	2.00x
Exe. Time (clk cycles)	36.9M	29M	1.27x

- **Atalanta RTOS**
- **40 tasks**
- **4 PEs**

Experiments

Small CS lock results

	Without SoCLC	With SoCLC	Speedup
Lock Latency (clk cycles)	884	32	27x
Lock Delay (clk cycles)	8936	102	87.6x

- **Atalanta RTOS**
- **40 tasks**
- **4 PEs**

Synthesis of SoCLC

- TSMC 0.25 micron technology (Synopsys Behavioral Compiler)

short CS locks	long CS locks	total # of locks	Total Area (gates)	short CS locks	long CS locks	total # of locks	Total Area (gates)
S=16	L=16	T=32	2,734	S=64	L=16	T=80	4,881
	L=32	T=48	3,586		L=32	T=96	5,733
	L=64	T=80	5,288		L=64	T=128	7,435
	L=128	T=144	9,027		L=128	T=192	11,174
S=32	L=16	T=48	3,454	S=128	L=16	T=144	8,163
	L=32	T=64	4,306		L=32	T=160	9,015
	L=64	T=96	6,008		L=64	T=192	10,717
	L=128	T=160	9,747		L=128	T=256	14,456

Conclusion

- A hardware mechanism for multi-processor SoC Lock Synchronization: SoC Lock Cache
- Reduction in lock latency, lock delay
- 27% overall speedup in an example database application
- Support *both* long Critical Sections and short Critical Sections
- Allow context-switching of tasks instead of busy-waiting