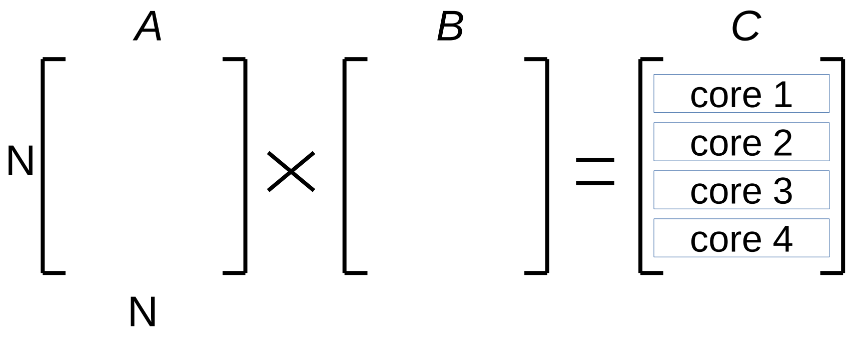
Solutions for the Re-exam in EDA284 (Chalmers) and DIT361 (GU) Parallel Computer Architecture, Monday, January 7th, 2019, 14:00h – 18:00h

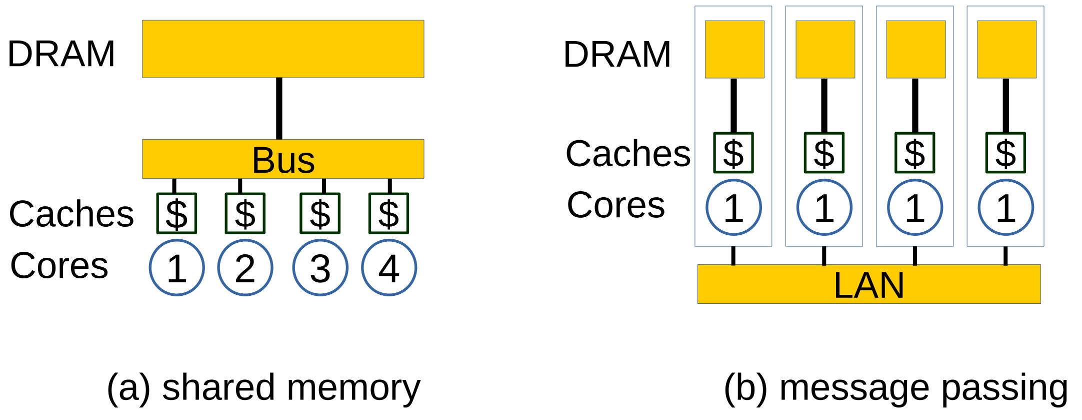
Problem 1

The two main programming paradigms for parallel computers are shared memory and message passing. In the course, the parallelization of a matrix multiplication ($A \cdot B = C$) was used to exemplify both paradigms. In this problem we want to analyze the performance of both approaches.

Assume that the matrices are square and each consists of N rows and columns. A typical way to decompose (parallelize) the problem over four cores is shown in the following figure:



This strategy can be used to parallelize the algorithm for both (a) shared memory and (b) message passing systems. The figure below shows two such systems.



The task is to discuss whether the five following statements are correct in the context of the two paradigms and the two shown systems. You should write 1-2 sentences for each statement and paradigm (i.e., total 10 answers). Just stating *true* or *false* will not be considered sufficient.

Unless otherwise specified, the following assumptions are to be considered:

- In both systems the DRAM, Bus and LAN support up to 32 GB/s of bandwidth.
- Initially the matrices *A* and *B* are stored in the DRAM memory connected to core 1. The algorithm finalizes when matrix *C* is stored back in the memory of core 1.
- The matrix multiplication is parallelized into multiple threads, each consuming a constant 4 GB/s of DRAM bandwidth on each core.
- The DRAM capacity is not a limiting factor
- The cache coherence protocol used in the shared memory system (a) is MSI-invalidate

Statement A "In order to function correctly, it is necessary to explicitly copy both matrices A and B into the DRAM memory connected to each core."

St. B "As long as the (Bus / LAN) interconnect bandwidth is larger than 16 GB/s, the execution time is independent of the speed of the interconnect (Bus / LAN)"

St. C "The execution time does not depend on the DRAM bandwidth, as long as the bandwidth exceeds 8 GB/s"

St. D "As the matrix size N increases, the speed-up compared to a single core approaches 4x"

St. E "Neither of the two parallelization paradigms requires Operating System support."

Please justify your answers. Simply answering *Correct* or *False* will not be considered not enough.

Answers to Problem 1

Statement (A)
(a): For the Shared memory system no copies are necessary as all cores have access
(b): Copies are necessary in this case, but the matrix A does not need to be copied in full, only the rows belonging to each core

Statement (B)
(a) This is correct, as the MxM kernel needs 4x 4GB/s to operate without contention
(b) Given that matrices need to be copied, the execution time will always depend on the LAN bandwidth

Statement (C)
(a) this is not correct, as the maximum memory bandwidth required is 4x 4GB/s = 16 GB/s
(b) this is correct, as only 4 GB/s are required for each node (each node executes a single MxM kernel)

Statement (D)
(a) true, since the 4 cores are not able to saturate the system's bandwidth

(b) true, for the same reason and (a), and also because the relative amount of communication required for the MxM (proportional to N^2) becomes negligible compared to computation (proportional to N^3) when N becomes large

Statement (E)
(a) incorrect, as there is always a need to create threads, which requires OS support (e.g. pthread_create)
(b) incorrect. In addition to creation of processes it is necessary to communicate data, which involves the OS

Problem 2

Consider a shared memory multiprocessor that consists of four processor/cache units and where cache coherence is maintained by an MSI protocol. The table shows the access sequence taken by the four processors to the same block but to different variables (A,B,C,D) in that block.

Nr	**Processor 1**	**Processor 2**	**Processor 3**	**Processor 4**
1	RA			
2		RB		
3			RC	
4				RD
5	WA			
6		WB		
7			WC	
8				WD
9				RD
10			RD	
11		WB		
12	RA			

The task is to classify each access as a hit, cold miss, true sharing miss, or false sharing miss. Furthermore, which of the misses could be ignored and still guarantee that the execution is correct? If each bus-upgrade costs 10 bytes, and a bus read request costs 38 bytes (32 for cache block + 6 bytes bus-header) what is the total essential traffic of these accesses?

Solution to Problem 2

1. Cold miss (38 Bytes)
2. Cold miss (38 Bytes)
3. Cold miss (38 Bytes)
4. Cold miss (38 Bytes)
5. Hit, Invalidates 2,3,4 (10 Bytes)
6. Miss, false sharing, can be ignored, invalidates 1 (0 Bytes essential traffic)
7. Miss, false sharing, can be ignored, invalidates 2 (0 Bytes essential traffic)

8. Miss, false sharing, can be ignored, invalidates 3 (0 Bytes essential traffic)
9. Hit (0 Bytes)
10. Miss, True sharing (38 Bytes)
11. Miss, False sharing, invalidates 3,4 (0 Bytes essential traffic)
12. Miss, false sharing, (0 Bytes essential traffic)

Problem 3

A design team needs to choose an appropriate cache coherence protocol to be used for a shared memory multiprocessor with a number of processor/private cache units connected by a shared single-transaction bus. The team is considering three invalidation-based snoopy protocols: MSI, MESI and MOESI.

In order to select the protocol, the team is analyzing the following representative sequence of accesses happening on the bus in the following order:

R1/X, R2/X, R2/Y, W1/X, W2/Y

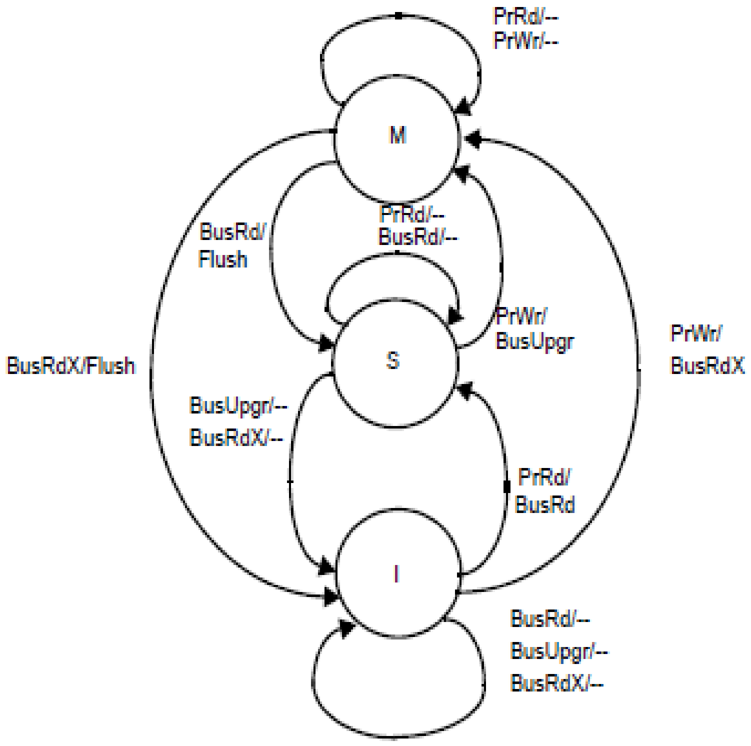
where the notation R*i*/X, W*i*/X means a read and write from processor *i* to cache block X. The latency and traffic costs of different operations considering a block size of B bytes are:

Operation	**Latency**	**Bus Traffic**
Read hit	1	N/A
Write hit	1	N/A
Request serviced by next level (BusRd,Flush)	100	6+B
Request serviced by different cache (BusRd/Flush)	20	6+B
Bus upgrade (BusUpgr)	10	10
Snoop action	5	N/A

Assumptions:

1. A bus upgrade consists of transferring the request on the bus and making a snoop action in each cache (the latency of the latter is shown in the table)
2. There is no contention on the tag directory (duplicated tag directory)
3. X and Y are not the same cache block
4. The block size B is 32 bytes
5. Read-exclusive requests (BusRdX) have the same latency and traffic as Read requests

The MSI state-transition diagram is shown below



To assess the performance of the three protocols the team must construct a table with the latency (cycles) and bus traffic (bytes) needed to complete the aforementioned access sequence (shown below).

Protocol	Latency (cycles)	Traffic (bytes)
MSI		
MESI		
MOESI		

Your task is as follows:

(a) Describe in 1-2 sentences the meaning of the states 'E' and 'O'. What problems do they address?
(b) Show the protocol transitions in each cache for the lines X and Y
(c) Complete the table with latency and traffic values

Solutions to Problem 3

(a) States E and O:
'E' means 'exclusive' . Cache has single copy, and memory is clean
'O' means 'owner'. Cache has multiple copies, only one in state O. 'Owner' block replies to BusRd requests, enables sharing of dirty data.

(b) Protocol transitions

	MSI		**MESI**		**MOESI**	
Sequence	Cache 1	Cache 2	Cache 1	Cache 2	Cache 1	Cache 2
R1/X	I → S (BusRd)		I → E (BusRd)		I → E (BusRd)	
R2/X		I → S (BusRd)	E → S	I → S (BusRd)	E → O (Flush)	I → S (BusRd)
R2/Y		I → S (BusRd)		I → E (BusRd)		I → E (BusRd)
W1/X	S → M (BusUpgr)	S → I	S → M (BusUpgr)	S → I	O → M (BusUpgr)	S → I
W2/Y		S → M (BusUpgr)		E → M		E → M

(c) Latencies and traffic values

	MSI		**MESI**		**MOESI**	
Sequence	Cache 1	Cache 2	Cache 1	Cache 2	Cache 1	Cache 2
R1/X	5 + 100 / 6+B		5 + 100 / 6+B		5 + 100 / 6+B	
R2/X	5 + 100 / 6+B		5 + 100 / 6+B		5 + 20 / 6+B	
R2/Y	5 + 100 / 6+B		5 + 100 / 6+B		5 + 100 / 6+B	
W1/X	5 + 10 / 10		5 + 10 / 10		5 + 10 / 10	
W2/Y	5 + 10 / 10		1 / 0		1 / 0	
Total	345 / 38 + 3xB (96) 345 cycles / 134 bytes		310 +21 = 331 / 28 + 3xB (96) = 124 bytes 331 cycles / 124 bytes		230 + 26 = 256 / 124 bytes 256 cycles / 124 bytes	

Protocol	**Latency (cycles)**	**Traffic (bytes)**
MSI	345	134
MESI	331	124
MOESI	256	124

Problem 4

Consider a future 8-way CMP on which you can power off some cores to allow the rest to operate at a higher frequency. You can use either 1, 2, 4, or 8 cores at the respective frequencies:

when only one core is running it operates at 0.3ns clock cycle
when two cores are running they operate at 0.4ns clock cycle
when 4 cores are running they operate at 0.5ns clock cycle,
when all 8 cores are running they operate at 1ns clock cycle,

Consider a partially parallel application which has a serial part that is 1000 Instructions and a parallel part that can be parallelized at will which is 2000 Instructions.
Parallelizing however requires you to create threads in the serial part of the application and 100 Instructions are added for every thread you create.
The serial and parallel part of the applications all run one instruction per cycle regardless of the frequency.

(a) Calculate the runtime of the application for the above processor when using 1,2,4,8 cores at their respective frequency without reconfiguring the processor while the application is running.
Which configuration is better?

(b) What if you could reconfigure the processor to run the serial part with one core at 0.3 ns clock cycle and then configure it to 2 or 4 or 8 cores for the parallel part. What is the runtime for each case? Always consider that creating each thread takes 100 Instructions that are added to the serial part (and run on the single core at 0.3ns clock cycle).

Solution to problem 4:

(a)
1-Core: 1000 Instr serial + 2000 Instr parallel = 3000 Instr at 0.3 ns = 900ns
2-Core: 1000 Instr serial + (1000 Instr parallel) + 200 Instr to create 2 threads = 2200 Instr at 0.40 ns = 880ns
4-Core: 1000 Instr serial + (500 Instr parallel) + 400 Instr to create 4 threads = 1900 Instr at 0.5ns = 950ns
8-Core: 1000 Instr serial + (250 Instr parallel) + 800 Instr to create 8 threads = 2050 Instr at 1ns = 2050ns
→ 2 Core is the best configuration

(b)
Serial part is always 1000 Instr at 0.3ns = 300ns
The parallel part :
2-Core: 200 Instr to create the threads at 03.ns + (1000 Instr parallel at 0.4) = 60ns + 400ns = 460ns
4-Core: 400 Instr to create the threads at 03.ns + (500 Instr parallel at 0.5) = 120ns + 250ns = 370ns
8-Core: 800 Instr to create the threads at 03.ns + (250 Instr parallel at 1) = 240ns + 250ns = 490ns

So the total runtime for the reconfigurable processor is
2-Core: 300ns + 460ns = 710ns
4-Core: 300ns + 370ns = 670ns
8-Core: 300ns + 490ns = 790ns

Problem 5

Two common spinlocks relying on atomic synchronization primitives are the Test-and-Set-based spinlock and the Test-and-Test-and-Set spinlock. The code for the two spinlocks are shown below.

Action/Label	**Lock #1 (Test-and-Set)**	**Lock #2 (Test-and-test-and-set)**

`Lock:`	`T&S R1, _lock`
	`BNEZ R1, Lock`
		`LW R1, _lock`
		`BNEZ R1, Lock`
		`T&S R1, _lock`
		`BNEZ R1, Lock`

`Unlock:`	`SW R0, _lock`	`SW R0, _lock`

Assume an MSI-invalidate snoopy cache coherent protocol is used to keep the caches of a shared memory multiprocessor consisting of four processors coherent.

In the execution under consideration, all threads are trying to modify a shared data structure via a single global lock. When one thread obtains access to a lock, it keeps the lock for a long time before releasing it.

(a) Describe the sequence of MSI protocol transitions that will occur for both **Lock #1** and **Lock #2**.
(b) Explain which spinlock behaves better in this scenario and why?
(c) If the cache coherence protocol were MESI or MOESI, are there any differences in the behavior?
(d) Assume now that different threads access the same data structure only with very low likelihood. What other solutions exist to address the problem of highly contented critical sections? Please list two such solutions.

Solution to problem 5

(a) Sequence of MSI protocol transitions

Lock #1

For each T&S, the local cache transitions from I → M, while one other cache will transition from M → I. This sequence continues until no threads attempt to access the shared data structure.

Lock #2

Initially, the local cache transitions from I → S when executing the LW instruction, leaving other caches unmodified. Eventually all caches will transition from I → S, and all coherence traffic stops. The loop is broken when the lock is released. Some threads will then attempt to execute T&S, which will result in a S → M transition, invalidating all other caches. The whole sequence then repeats while core continue to wait for the critical section.

(b) The spinlock based on Lock #2 behaves better because it avoids unnecessary traffic resulting from continuous invalidations of all other caches.

(c) MESI and MOESI protocols:

For Lock #1 there are no differences.
For Lock #2 and MESI, one cache will initially transition I → E before transitioning E → S or I → S. This difference is negligible.
For Lock #2 and MOESI, one caches will, in addition, share the value of the lock among all other caches by exploiting the 'O' value. The overall impact of this optimization is unlikely to have a high impact.

(d) Other possible solutions to alleviate the problem of contention:

If the data to be modified inside of the parallel section has a high probability of being independent, two possible solutions are:

1. Using fine-grained locks instead of a single coarse-grained lock (improves performance at the cost of complexity and potential for deadlock)

2. Use Transactional Memory (requires extra support for tracing read and write sets)

Problem 6

A system architect has three choices for an on-chip interconnection network: a uni-directional ring (UR), a bi-directional ring (BR) and an NxN mesh network (NM). In addition, the architect has two choices for providing coherence between L1 caches: snoop-based (SB) or directory-based (DB). There are 16 cores on the chip. All cores have private L1 caches and they share a L2 cache that is divided in 16 banks, with one bank attached to each core. The latency to communicate between two cores connected directly by a link is 1 cycles. The router latency is one cycle per router that the packet goes through. For directory-based coherence the access time to the directory is 5 cycles. Assume that there is no contention in any link or the directory.

a. Which combination of design choices provides the shortest latency to provide coherence? In this context, latency is considered to be the time it takes to reach all potential destination cores (not including acknowledgments). Consider the worst and average case for a coherence message to reach its destination(s)

b. Now consider 256 cores and a directory access time of 15 cycles, which combination of design choices provides the shortest latency to provide coherence in this case?

Please organize the results in a table as follows:

Case (a) 16 cores

Latency	**Coherence Type**	**UR**	**BR**	**NM**
Worst Case	SB	15	8	6
Worst Case	DB	5+15	5+8	5+6
Average	SB	15	8	5
Average	DB	5+8	5+4	5+3

Case (b) 256 cores

Latency	**Coherence Type**	**UR**	**BR**	**NM**
Worst Case	SB	255	128	30
Worst Case	DB	15+255	15+128	15+30
Average	SB	255	128	23
Average	DB	15+8	15+4	15+15

Problem 7

In order to implement a multiprocessor system that behaves correctly, it is necessary to keep caches coherent and to implement a precise memory consistency model.

Your task is to describe

(a) What is the difference between coherence and consistency? (1-2 sentences)

(b) Briefly describe the two memory consistency models known as Sequential consistency (SC) and Relaxed Memory Order (RMO)? What relaxations do they allow?

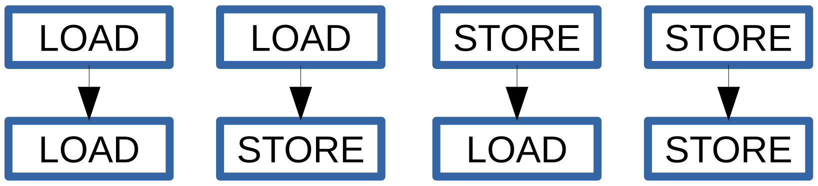
(c) Which of these two models (SC and RMO) is more restrictive in terms of hardware implementation? How do these models manage store buffers?

Solution to problem 7

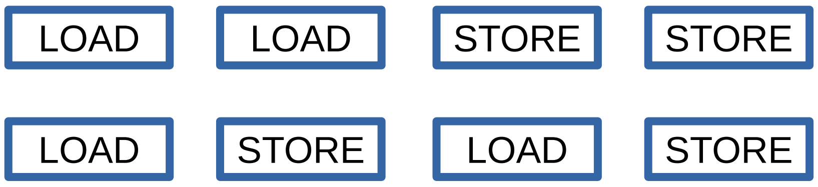
(a) Both coherence and consistency describe observable orders of updates to shared variables. However, while coherence is specified on a per-memory location basis, consistency is specified with respect to all memory locations.

(b) Memory consistency models can be defined as enabling certain relaxations on orders of memory accesses performed by loads and stores.

In SC, no orders can be relaxed, meaning that all threads observe the same orders. I.e:



RMO is a memory consistency model in which no orders are automatically enforced (except for the intra-thread order). Required orderings need to be specified via specific instructions such as memory barriers.



(c) SC imposes strong requirements on the orders in which operations can be performed, such as requiring all loads to be globally performed. This makes store buffers are ineffective, as all stores need to be globally performed before a load can execute. On the other hand RMO enables many optimizations in the memory subsystem. Among others, it allows loads to bypass older stores (to a different address), hence enabling the usage of store buffers.

Problem 8

Core multithreading is a technique to improve the utilization of resources in modern processors whose performance is limited by long latency events such as cache misses. The goal of this problem is to describe the differences between the four models of multithreaded processors that have been introduced in class: (1) block multithreading, (2) interleaved multithreading, (3) barrel processors, and (4) simultaneous multithreading.

For each alternative, the task is to describe:

(a) the granularity of thread switches, i.e. when are threads switched?
(b) in which types of pipeline (e.g. in-order, out-of-order) can the technique be implemented? What is the cost of a thread switch?
(c) what does the software need to provide to achieve best efficiency?

Solution to Problem 8

(a) Granularity

Block Multithreading:	long latency events, e.g. L1 cache misses
Interleaved Multithreading:	Single cycle
Barrel Processors:	Single cycle
Simultaneous Multithreading:	Usually single cycle, but multiple threads can potentially be fetched each cycle

(a) Pipeline

Block Multithreading:	Both in-order and out-of-order. Pipeline must be flushed at each thread switch
Interleaved Multithreading:	Only in-order pipelines. There is no cost in switching. All threads are simultaneously active.
Barrel Processors:	Only in-order pipelines. Similar to Interleaved MT, there is no cost in switching.
Simultaneous Multithreading:	Extension of IM for out-of-order pipelines. No cost in thread switching

(a) Software

Block Multithreading:	At least a few active threads (typically 2-4) to switch upon long latency events
Interleaved Multithreading:	A higher number of threads (typically 4-8) to keep the pipeline busy and reduce pipeline bubbles
Barrel Processors:	At least as many threads as pipeline stages. Barrel processors can only hole a single instruction from each thread in the pipeline.
Simultaneous Multithreading:	A few threads (typically 2-4) to hide the L1 miss latencies