CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik
Avdelningen för datorteknik

Exam in EDA284 (Chalmers) and DIT361 (GU) Parallel Computer Architecture, Saturday, March 20th, 2021, 8:30h - 12:30h

---

Language/Språk: Answers shall be given in English. The results should be submitted via Canvas in pdf (preferred) or in .docx format.

Solutions/Lösningar: Solutions will be posted on Wednesday, March 24th, 2021 on the Canvas page.

Exam review/Granskning: The review date will be posted on the course Canvas page by the time you receive the email from LADOK.

Aids/Hjälpmedel: Since this is a take-home exam, all aids are allowed during examination. Plagiarism is of course strictly prohibited and checks will be conducted on all exams.

Grades:

Chalmers				
Points	0-23	24-35	36-47	48-60
Grade	Failed	3	4	5

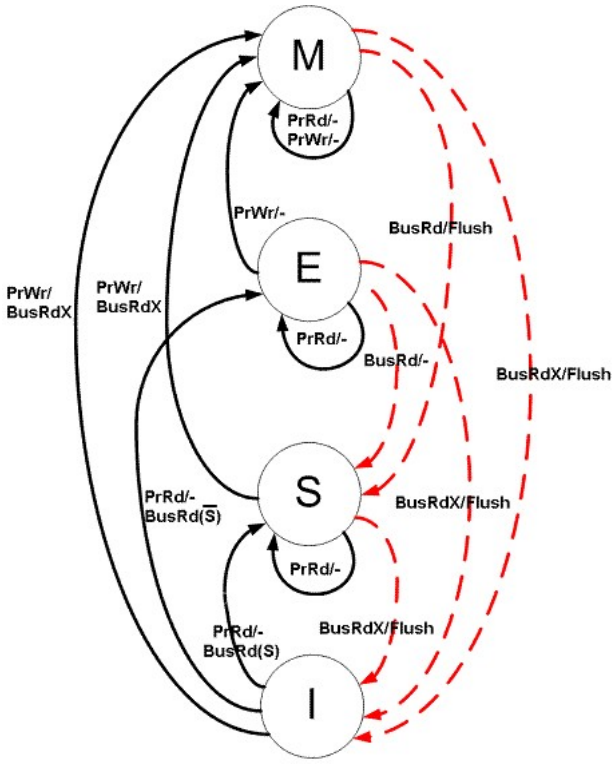
GU				
Points	0-23	24-41	42-60	
Grade	Failed	G	VG	

Good Luck!

Problem 1 (12 points)

Given a bus-based system implemented as a Chip Muliprocessor (CMP) that has 3 processors (namely P1,P2,P3) where each has a private L1 direct-mapped cache connected to system wide DRAM memory controller, consider the following sequence of memory operations on two shared 32-bit integers A and B.

```
1. P1: read  B
2. P2: read  B
3. P1: write B, 1
4. P2: write B, 2
5. P3: write A, 1
6. P3: read  B
7. P2: write A, 2
8. P1: write B, 10
```



Assume A and B are initialized to 0, cache lines are in an invalid state and the cache line size is 64 bytes. Consider also that **A and B are located on the same cache line**. Using the MESI cache coherence protocol with the state diagram given above (showing both bus and processor requests transitions), determine:

(a) The cache line coherence state in each processor.

(b) The contents of A and B in the cache and the memory.

(c) The processor action (`PrRd`, `PrWr`).

(d) The bus action(s) (`BusRd`, `BusRd(s)`, `BusRd(s̄)`, `BusRdX`, `BusUpgr`, `Flush`).

(e) The question states that the two integers (A and B) are shared across the 3 processors. Does this constitute a false-sharing problem? Why or why not?

(f) Assume now a non-atomic cache subsystem. State two consecutive instructions from the above sequence that might pose a race condition on the transitions. Mention and briefly explain (in 2 lines) an approach to resolve the mentioned race condition?

Note: use the tables to answer parts (a, b, c and d). A docx file that contains the tables is available for download in this link: https://chalmersuniversity.box.com/s/lwcdsj4xo3lyrucv6b0ull9fgjjwjbr5.

Inst	Memory content
-	A(0),B(0)
1	
2	
3	
4	
5	
6	
7	
8	

Inst	P1			
	Cache content	State	PrReq	BusReq
-	-	I	-	-
1				
2				
3				
4				
5				
6				
7				
8				

Inst	P2			
	Cache content	State	PrReq	BusReq
-	-	I	-	-
1				
2				
3				
4				
5				
6				
7				
8				

Inst	P3			
	Cache content	State	PrReq	BusReq
-	-	I	-	-
1				
2				
3				
4				
5				
6				
7				
8				

Problem 2 (10p)

Consider a future 64-way CMP operating in a power-constrained environment. The CMP can automatically configure itself to run as a 1-, 8- or 64-way core CMP but always using a fixed power budget. For instance, it can run as a single-core processor by grabbing the power from the other 63 cores and putting them to sleep and using the additional power to increase its frequency. Assume for simplicity that sleep and wakeup transition times are zero, and that power and frequency have a cubic relationship. For instance, if one core uses the power of all 64 cores, its frequency can increase four-fold (since $4^3 = 64$). We call the design an EPI-throttled CMP.
Consider a parallel application running on the EPI-throttled CMP. Its execution consists of four phases and each phase uses a different number of resources. (1) The application starts as a single-threaded application. This part can not be parallelized and consists of 500 cycles in sequential mode. (2) During the following 4000 cycles, it is parallelized by using all 64 threads. (3) In the third phase, it consumes 4000 cycles running with only 8 threads due to limited parallelism. (4) During the remaining 1500 cycles, the fourth phase runs with only one thread and can not be parallelized.

(a) What is the speedup of the application when it runs on the EPI-throttled CMP compared to running on a single-core machine that uses the same power budget but can operate at a higher frequency using the aforementioned cubic relationship?

(b) What is the speedup of the application when it runs on the EPI-throttled CMP compared to running on a traditional 64-way CMP that does not provide the reconfiguration capability?

(c) Assume that a new technology is developed such that the power consumption increases linearly with frequency. What are the new speedups of problem (a) and (b) respectively? What can you conclude from the result?

Problem 3 (8p)

Consider the following lock design which we call PLock (for "Pause + Lock").

```
1 PLock:    PAUSE Rx
2           T&S R1, lock
3           BNEZ R1, PLock
4           RET
```

PAUSE Rx is a new instruction which stalls the pipeline in the EX stage for Rx cycles, where Rx is a register. A number threads 'nthreads' are using this implementation to concurrently access a critical section. Each thread runs on a classical 5-stage single-issue in-order pipeline (consisting of Fetch, Decode, Execute, Memory and WriteBack). The cores are connected by an atomic bus, and the cache coherence protocol is MSI. Assume that all instructions have a latency of one cycle.

(a) Briefly discuss the pros and cons of this lock implementation? How does this locking scheme scale?

(b) What is the average number of invalidations per cycle that will be generated, as a function of Rx and the number of threads 'nthreads' accessing the critical section? If there are 8 threads, and one want to keep the bus occupied with invalidations for only 10% of the time, what value of Rx should be chosen?

(c) Discuss briefly whether it makes sense to combine the above lock with the approach known as Test-and-Test-and-Set.

Problem 4 (12p)

The following code is to be executed on two processors based on interleaved multithreading (a.k.a fine-grained multithreading). We first consider a single-issue in-order processors with a pipeline of 10 stages.

```
1  int ivec[1024] = {some values...};
2  int ovec[1024];
3
4  for(i=0; i<1024; i++){
5      if(ivec[i] % 2)
6          ovec[i] = ivec[i];
7      else
8          ovec[i] = -1 * ivec[i];
9      }
10 }
```

(a) What is the minimum number of concurrent threads that need to be supported so that the processor can completely eliminate the logic for detecting structural and data hazards and still expect to execute at the maximum throughput of 1 instruction per cycle?

(b) Assume now that the system's caches are initially empty and that each cache miss results in a miss latency of 100 cycles. Assume that memory bandwidth is unlimited. How many threads are now required to keep the pipeline filled and avoid pipeline stalls?

Next the above code is to be executed on a GPU, such that each iteration of the loop is executed by a different thread of the same thread block. To simplify, assume that each instruction executes atomically with a latency of a single cycle and that the thread selection policy is Greedy-Then-Oldest (GTO). The code generated by the compiler for each iteration is as follows:

```
1  # GPU kernel code (i := thread index within thread block)
2  void eda284_problem4(int i)
3  {
4      if(ivec[i] % 2)
5          ovec[i] = ivec[i];
6      else
7          ovec[i] = -1 * ivec[i];
8
9  }
10
11 # Assembly output of a RISC-V-like GPU ISA (modified from a clang compilation)
12 eda284_problem4(int i):       # @eda284_problem4(int)
13         # register a0 stores parameter value (index i)
14         li64    a1, ivec     # a1 = ivec
15         slli    a0, a0, 2    # a0 = i * 4 ; size of int is 4 bytes
16         add     a1, a1, a0   # a1 = ivec + i * 4
17         lwu     a1, 0(a1)    # a1 = ivec [i]
18         andi    a2, a1, 1    # a2 = a1 & 0x1
19         bnez    a2, .LBB0_2  # if(a2 != 0) then jump to LBB0_2
20         neg     a1, a1       # else a1 = -a1
21 .LBB0_2:
22         li64    a2, ovec     # a2 = ovec
23         add     a0, a0, a2   # a0 = ovec + i * 4
24         sw      a1, 0(a0)    # ovec[i] = a1
25         ret                  # End of assembly code
```

(c) Assuming that memory access does not take any extra cycles, what will be the utilization of execution units (in percentage) and the execution time of a single warp (32 threads)?

(d) Repeat task (c) but assuming now the memory system used in task (b) with caches initially empty. What will be the new utilization and execution time?

Problem 5 (12p)

The two main programming paradigms for parallel computers are shared memory and message passing. Consider the two systems shown in Figure 1. A programming team wants to implement a parallel matrix multiplication (A $\times$ B = C) to study the performance of both paradigms, where A and B are square matrices (of size N = 1000) and the matrix elements are of type `float`. The approach to parallelize the problem over the four cores is shown in Figure 2. For both systems, assume that each each multiplication instruction and addition instruction take 1 FLOP to complete. According to the simplified DRAM roofline model, the matrix multiplication algorithm has been measured to be memory-bound and the arithmetic intensity (AI) has been measured to be 1.0 FLOPs/Byte.

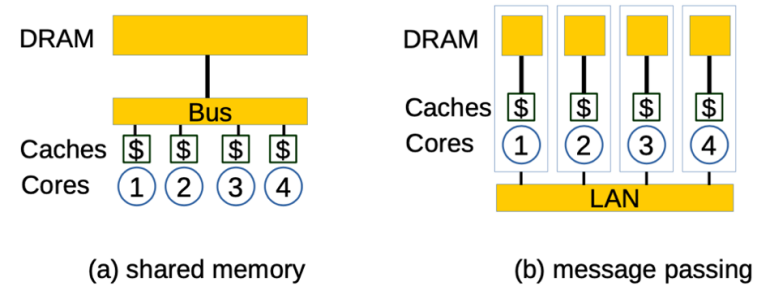


Figure 1: Two programming paradigms.

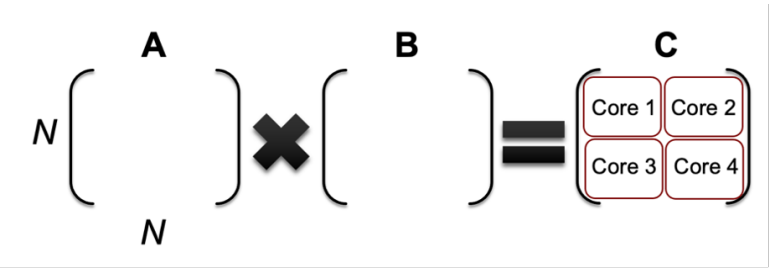


Figure 2: Matrix multiplication.

(a) First consider the shared memory system. The DRAM and Bus interconnect both provides up to 2 GB/s bandwidth and these resources are shared between 4 cores. What is the total execution time when running the above matrix multiplication on this computer?

(b) Next consider the message passing system. Assume that 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 LAN interconnect supports 1 GB/s of bandwidth (equally shared between cores requiring data transfers) and these cores can not start the computation until all required data has been received. The system provides up to 1 GB/s bandwidth in each DRAM. There is no special hardware support for message passing in this case. What will be the total execution time?

(c) Instead of using a simple memory copy such as in problem (b), the team plans to use DMA engines so that the data transfer latency can be reduced by 50% compared to problem (b). Programming one DMA engine adds an overhead of 0.001s in each step (consisting of (i) sender user area to sender system area, (ii) sender system area to network interface, (iii) network interface to receiver system area, (iv) receiver system

area to receiver user area). What will be the new total execution time when running on this improved message passing system?

(d) Briefly discuss other hardware support that can be added to further optimize message passing system efficiency?

Problem 6 (6p)

Assume an in-order processor with a store-buffer. Three threads are collaborating to solve a problem. The following sequence of loads 'L' and stores 'S' have been issued by each thread to variables 'x', 'y' and 'z'. The value returned by the load or written by the store is indicated in parenthesis.

```
Thread 1: S(x=5)  -> L(y=8)  -> L(x=5)  -> L(y=8) -> L(y=8)
Thread 2: S(y=8)  -> S(z=16) -> S(x=2)  -> L(x=10)
Thread 3: S(x=17) -> L(x=5)  -> L(z=16) -> L(x=2) -> S(x=10) -> S(y=3) -> S(z=18)
```

(a) Is this sequence of loads and stores possible under SC (sequential consistency)? Try to construct an order for the above execution that is consistent with SC.

(b) Can this order be relaxed for more efficient execution under TSO (total store order) rules? Compare with the case of Task (a) and explain the similarity or difference between the two executions. If optimization is possible, identify at least one instance (instruction) where this can be applied.