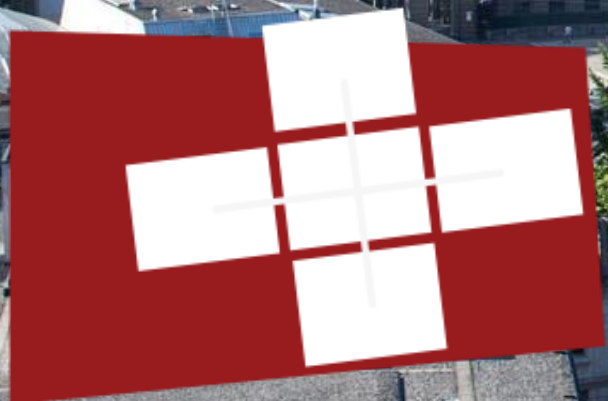


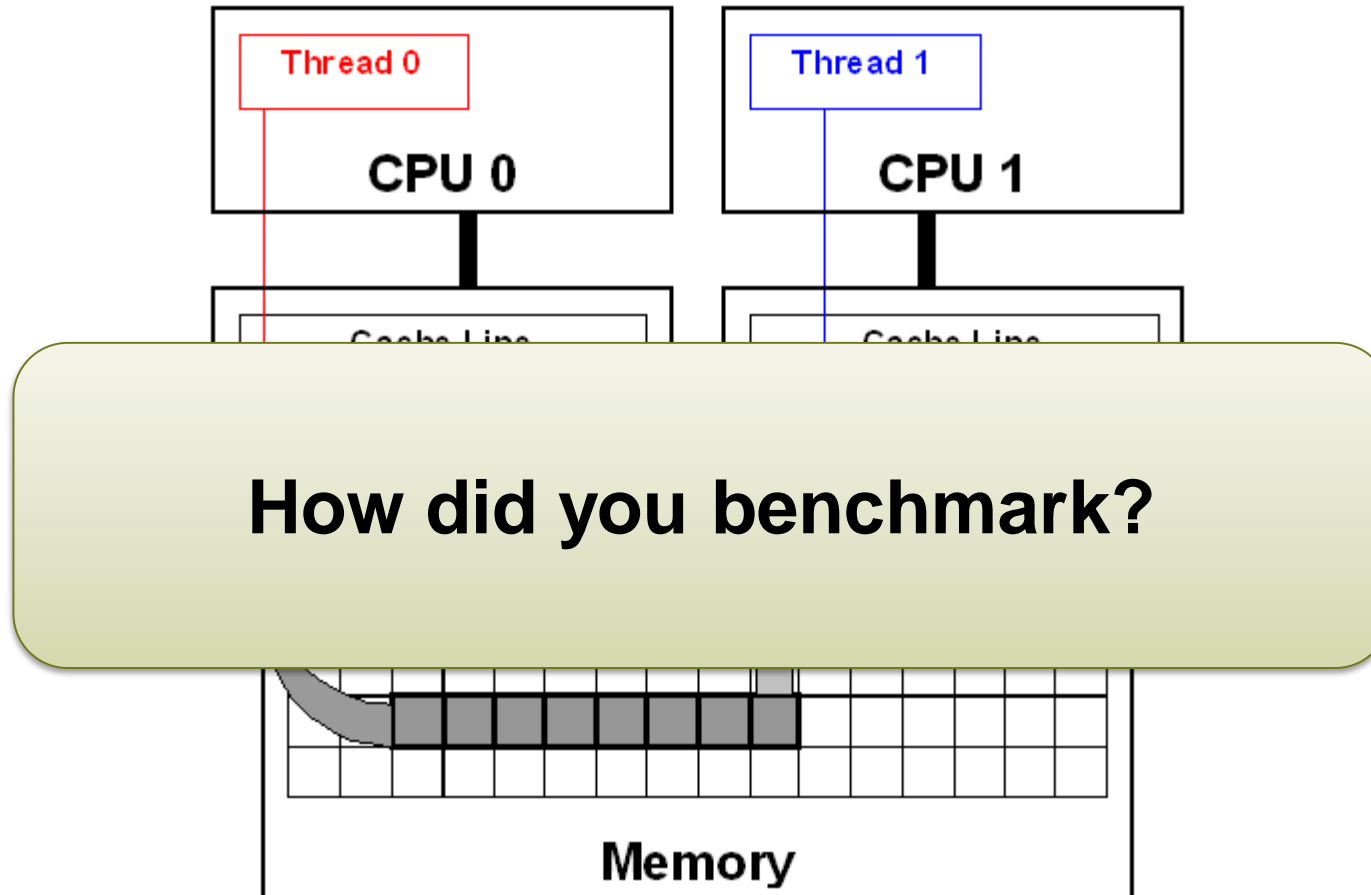
SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: Assignment Discussion

Recitation session



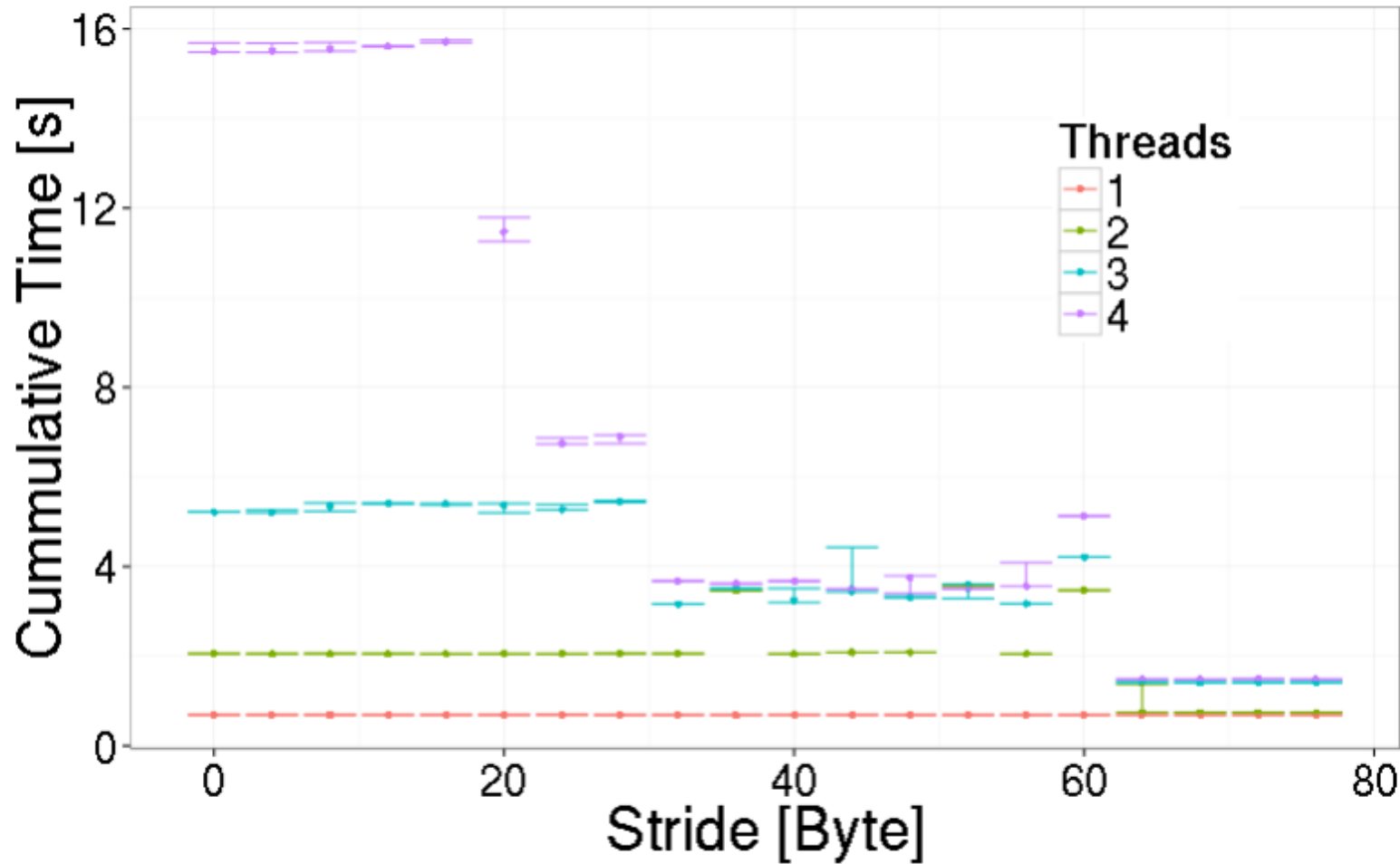
False Sharing Benchmark



False Sharing Benchmark

- **Idea: Allocate `uint8_t` array `a`, let core 0 write to `a[0]` and core 1 to `a[x]`**
- **If `x` is larger than the size of one CL, this should be “fast” because both cores operate on their own cached copy of different CLs**
- **If `x` is smaller than one CL it will be slow, due to false sharing**
- **In practice it is a bit harder to get it right :)**
 - If we write only once it might not really be parallel -> do it in a large enough loop
 - If we write only one Byte in each iteration we will not see much because of loop overhead (incrementing counter, jump) -> write 8 bytes in inner loop
 - Make sure the compiler does not “optimize” your loop by removing it!

False Sharing Benchmark

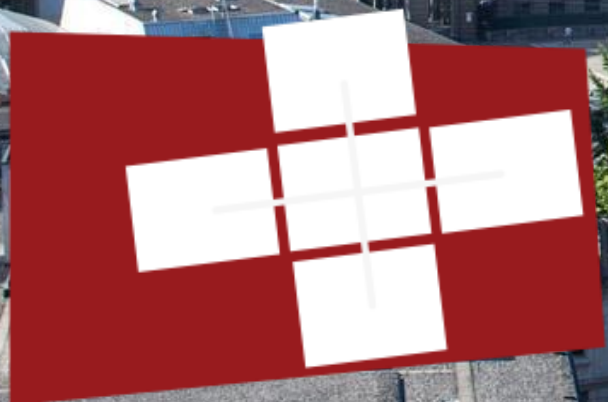


Machine: Intel Core i5 3230M; **Compiler:** gcc 4.9.1 -O3 -fopenmp -std=gnu11

SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: Sequential Consistency

Recitation session



Consistency vs Coherence

- **Writes to same location**
 - **Coherence**
 - a) Write Serialization: all processors see writes to the same location in the same order*
 - b) Write Propagation: a write will eventually be seen by other processors*
- **Writes to different location**
 - **Memory Model:** defines the ordering of writes and reads to different memory locations – the hardware guarantees a certain consistency model and the programmer attempts to write correct programs with those assumptions

Consistency: Example

- Multiprocessor with bus-based snooping cache-coherence and write buffer
- Initially A=B=0

```
T1:  
A=1  
if (B==0){  
  <enter critical section>  
}
```

```
T2:  
B=1  
if (A==0){  
  <enter critical section>  
}
```

Does it work (in x86)?

- This lock implementation is based on two different variables (i.e., memory location)
- The stores are intercepted by the write buffer => P1 and P2 can enter the critical section at the same time
- Cache coherence is not involved here

Is that always true?

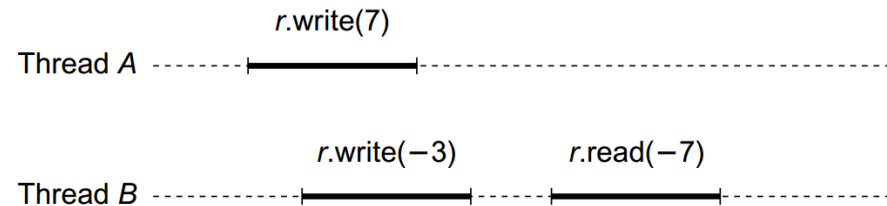
Memory Models

“A formal specification of how the memory system will appear to the programmer, eliminating the gap between the behavior expected by the programmer and the actual behavior supported by a system.” [Adve’ 1995]

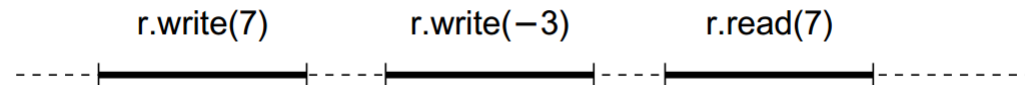
- **Memory model specifies:**
 - How threads interact through memory
 - What value a read can return
 - When does a value update become visible to other threads
 - What assumptions are allowed to make about memory when writing a program or applying some program optimization

Sequential Consistency

- Method calls act as if they occurred in a sequential order consistent with program order
 - Method calls should appear to happen in a one-at-a-time, sequential order*



- Method calls should appear to take effect in program order*

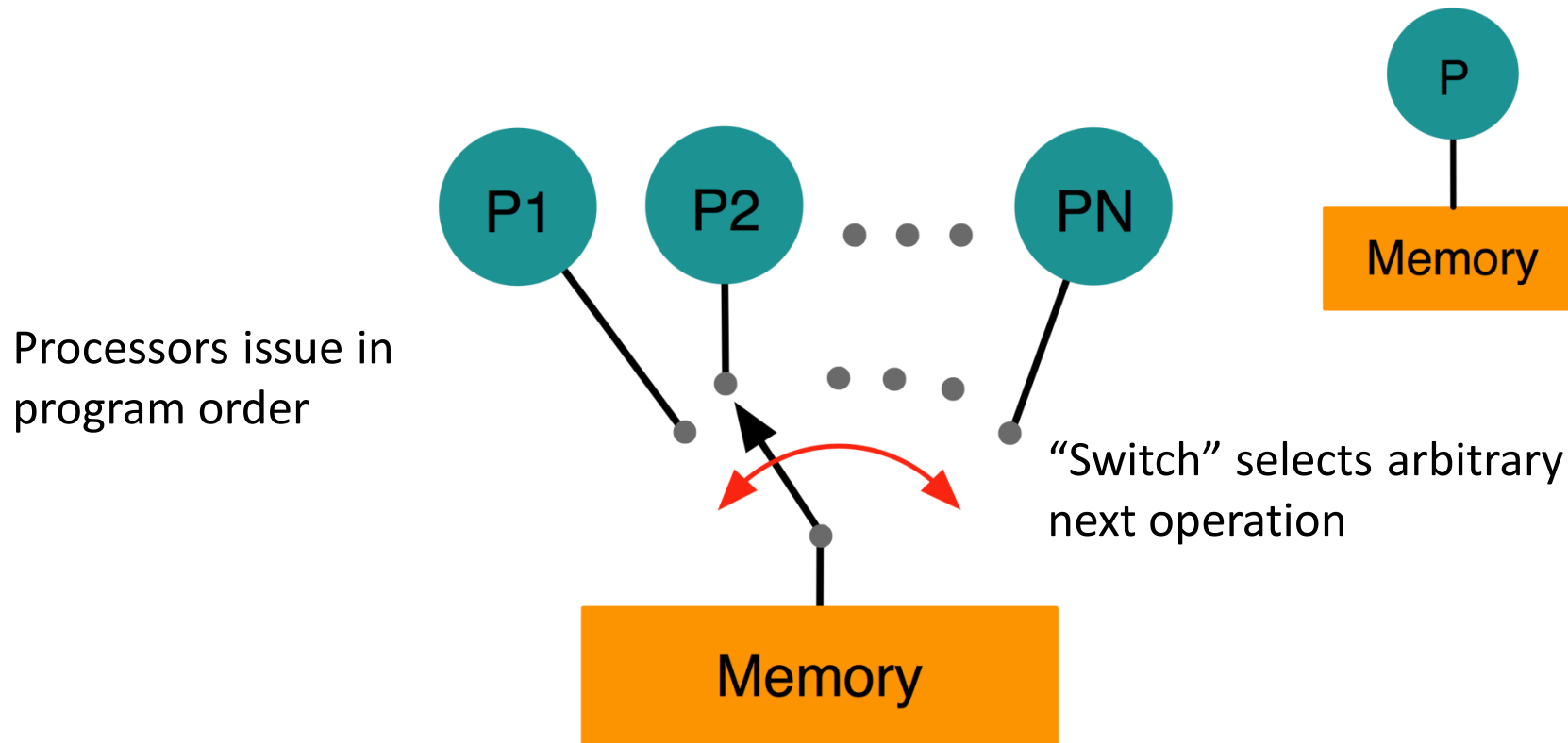


Program Order: Per-processor order of memory accesses, determined by program's *control flow*.

Visibility Order: Order of memory accesses observed by one or more processors

Sequential Consistency Illustrated

- Method calls act as if they occurred in a sequential order consistent with program order
 - *Method calls should appear to happen in a one-at-a-time, sequential order*
 - *Method calls should appear to take effect in program order*



Sequential Consistency - Discussion

- **Programmer's view:**
 - Prefer sequential consistency
 - Easiest to reason about
- **Compiler/hardware designer's view:**
 - Sequential consistency disallows many optimizations!
 - Substantial speed difference
 - Most architectures and compilers don't adhere to sequential consistency!
- **Solution: synchronized programming**
 - Access to shared data (aka. "racing accesses") are ordered by synchronization operations
 - Synchronization operations guarantee memory ordering (aka. fence)
 - More later!

Memory Fence: special instructions that require all previous memory accesses to complete before proceeding (*sequential consistency*)

Relaxed Memory Models

- **Ideal:** intuitive programming model (i.e., sequential consistency) and high-performance
 - *Not that easy* 😊
- **Idea:** Relax some constraints, but allow the programmer to enforce them from specific portions of the code
- **Some possible relaxations (different memory locations):**
 - Relax $W \rightarrow R$: Reads may be reordered with older writes to different locations but not with older writes to the same location (x86)
 - Relax $W \rightarrow W$: Writes can be reordered with other writes
 - Relax $R \rightarrow W$: Writes can be reordered with older reads
- **A consistency model is identified by a set of constraint**

Write Buffers

- They can destroy the program order (as seen from other CPUs), hence invalidate SC
- Overtaking of messages *is desirable* and should not be prohibited in general.
- **Solution: memory barriers!**
 - x86 CPUs provide the **mfence** instruction
 - a write barrier after each write gives sequentially consistent CPU behavior (and is as slow as a CPU without store buffer)
 - **Use memory barriers only when necessary**

Exercise 1

a) For the following executions traces, either provide a sequentially consistent interleaving or show that this does not exist. $W(var, val)$ indicates the write of the value val to the variable var . $R(var, val)$ indicates a read. Assume that all the variables are initially set to 0. Justify your answer! (10p)

1. P1: $W(x,1); R(y,0)$
P2: $W(y,1); R(x,0)$
2. P1: $W(x,1); R(x,2); W(x,1)$
P2: $R(y,0); R(y,2); R(x,1)$
P3: $W(x,2); W(y,2)$
3. P1: $W(x,1); R(x,2); W(x,1)$
P2: $W(y,1); R(y,2); R(y,1)$
P3: $W(x,2); W(y,2)$
4. P1: $R(x,0); R(y,1)$
P2: $W(y,1); W(x,1);$
P3: $R(x,1); W(y,2);$
5. P1: $W(x,1); R(z,2)$
P2: $R(z,1); R(y,0); W(z,2)$
P3: $R(x,1); W(z,1); W(y,1)$

Exercise 2

c) Consider the following instructions executed by 3 processors (P1, P2, P3). All variables are stored in a memory system which is sequentially consistent. All variables are initially set to zero. All operations, including the print statements, are atomic. The function `print(int a, ...)` outputs all the passed integers.

	P1	P2	P3
Statement 1	x=3	y=1	z=2
Statement 2	print(x,y)	print(z)	print(z,x)

Are the following sequences legal outputs? Explain your answer by showing one possible interleaving of the instructions that might lead to the legal output. In case of an illegal output, explain why no possible interleaving exists. (6pt)

- 31023
- 23031
- 23231
- 02331