Stenstrom: Lund → Chalmers (now)

- A lot of cache work (coherence/compression)

- Survey paper
  - Evaluates cache coherence schemes in hardware and software

- Consistency: given a multiprocessor system, if you have a sequence of loads and stores, how should those operations should be ordered
- Coherence: protocol that attempts to implement a consistency model

- Can be hardware or software
  - SHRIMP: software based approach to coherence. Unmap whole pages and move around. Used really fast DMA copy mechanism to copy pages across network
- Simplest version: coherent, but not cached
  - Paper mentions coherent shared memory machines without caches. (IBM RP3, Cedar)
- Snoopy: broadcast over bus
  - Each cache listens for ops on the bus relevant for things in its cache
  - Caches help improve memory access time

Shared Memory vs Message Passing

- Shared memory: pthreads
  - Most important: pthread_create (spawn a new thread), pthread_join (wait on some thread to exit), pthread_exit (called by thread to quit but not entire program)
  - Communication: implicit, through shared memory.
  - All threads are in the same address space → can pass pointers, have shared array
  - Pthreads mutex implementation calls into the OS → may want to yield if going to be blocked on a lock for a while
    - Don’t always call into the OS, spin if the lock will be released quickly
  - Pthread_mutex_init: dependent on system. May just set memory value to zero
    - High performance systems may want to use a lock manager (dedicated thread/core whose sole purpose is to manage locks), need to go to the OS
  - Futex: x86 Linux. Doesn’t work with other locking primitives, rely on specific x86 instruction
  - Condition variable: allow thread to block on some condition/value for some variable
    - More likely to yield, won’t spin like mutex

- Pthreads example
  - Program splits input array into 100 equal-sized chunks
- 100 threads each process a single chunk of the array, adding the elements in the chunk to a global sum
- One way to improve is to have a local_sum and have everyone attempt to update global_sum at the end
- Depending on the load, you may want to make final global update a tree-like structure if there's high contention.
- When making performance improvements, make sure to maintain correctness!
- function_starter() called because pthread_create() can only pass in a void *

**Message passing: MPI**
- Explicit message passing, need to specify destination
  - Can receive not knowing source or knowing the source
  - MPI does not allow for the dynamic creation of new threads/processes
- Requires single program, multiple data (SPMD)
- MPI_Init: initial argc/argv are not helpful, MPI_Init goes and gets the actual ones passed from the command line.
- MPI_Comm_size: how many processes there are
- MPI_Comm_rank: your ID in the MPI runtime
- Can build groups of processors that communicate using MPI_COMM_WORLD/other values
- MPI_Send and MPI_Receive are blocking; MPI_Isend and MPI_Ireceive are nonblocking
- **Memory is private** → can’t pass pointers
- Message is data and synchronization

**Shared memory over messaging:**
- Software: loads and stores into receives and sends
- Hardware: cores pass messages to memory

**Messaging over shared memory**
- Use queues in shared memory