Parallel Architectures

- Shared Memory
  - UMA
  - NUMA
- Multicomputer
  - like NUMA but no sharing only message passing

Consistency Models

- Sequential Consistency: every CPU sees the same interleaving
  - Program order
  - Actions of different threads interleaved arbitrarily
- Release Consistency: writes visible after sync

Parallel Patterns

- Bag of Tasks
  - One bag with synced access
  - Collection of bags with task-stealing
- Pipeline: sequential producer-consumer relationship
- Interacting Peers: info exchanged in both directions

Shared Variable Synchronisation

- MutEx: only one accesses a resource at a time
- Condition Synchronisation: delay until condition met
- Deadlock: two or more trying to enter CS, none succeeds
- Absence of Delay: if only one, thread is not prevented
- Eventual Entry: thread attempting to access CS succeeds

Locks

- Shared boolean variable with await: requires SC
- Test-and-Set: too many atomic TS operations
- Test-and-Test-and-Set: loop on cached, check again
- Bakery Algorithm: requires SC

Barriers

- Sense Reversing Barrier
- Symmetric Barrier
- Dissemination Barrier

Structured Primitives

- Semaphores
  - Buffer: binary semaphore
  - Bounded Buffer: n-semaphore
  - Producers-Consumers: n-semaphore + 2 bin. semaphores
- Monitors: one thread active within a method at any time

Pthreads

- threads
  - type: `pthread_t`
  - `pthread_create(tid, attr, f, arg)`
  - `pthread_join(tid, result)`
- semaphores
  - type: `sem_t`
  - `sem_init(sem, share, init)`
  - `sem_wait(&s)`: $P(s)$
  - `sem_post(&s)`: $V(s)$
- MutEx
  - type: `pthread_mutex_t`
  - `pthread_mutex_init(&m, attr)`
  - `pthread_mutex_lock(&m)`/`pthread_mutex_trylock(&m)`
  - `pthread_mutex_unlock(&m)`
- Condition Variables
  - type: `pthread_cond_t`
  - `pthread_cond_wait(&cv, &mut)`: `mut` is lock already held
  - `pthread_cond_signal(&cv)`
  - `pthread_cond_broadcast(&cv)`

Java

Messaging

- Synchronisation: async/sync
- Addressing: wildcard/destination
- Collective Ops: point-to-point or not
  - Broadcast: everyone gets a copy of same value
  - Scatter: everyone gets a portion of an array
  - Gather: opposite of scatter
  - Reduction: combine gathered values at one node
  - Scan (Prefix): Reduction with partials as well

MPI Concepts

- Communicator: set of processes
- `MPI_Init(argc, argv)` - before any MPI calls
- `MPI_Finalize()` - after all MPI calls
- `MPI_COMM_WORLD` - global communicator
- `MPI_Comm_size` - size of communicator
- `MPI_Comm_rank` - rank within communicator
- `MPI_Comm_spawn(command, argv, p, info, root, comm, intercomm, errcodes)` - create p new processes with intercomm being MPI_COMM_WORLD for new processes
- `MPI_Comm_get_parent(comm)` - gets the communicator used in spawn

MPI Sends and Receives

- `MPI_Send(buf, count, type, dest, tag, comm)` - send `count` sized chunk of type `type` stored in `buf` to `dest` in communicator `comm` with (non-negative) id `tag`
- `MPI_Recv(buf, count, type, src, tag, comm, status)` - receive `count` sized chunk of type `type` into `buf` from `src` in `comm` with id `tag`, store status into `status`
  - `MPI_ANY_SOURCE` and/or `MPI_ANY_TAG` can be used as wildcards
- A receive will match send with matching communicator, tag, and source in order
- Status info such as `s.MPI_SOURCE` and `s.MPI_TAG`

Blocking: Returns only when it is safe to reuse the buffer. **Not** after a matching operation has been executed.

Types of blocking communication

- `MPI_Rsend` works only if `MPI_Recv` already called
- `MPI_Ssend` returns only after matching receive found
- `MPI_Send` returns after buffer safe to reuse blocking if buffer full
- `MPI_Bsend` same as previous, error if buffer full
- `MPI_Recv` blocks until message completely in buffer

Types of non-blocking communication

- Put `I` in front of everything
- Extra `request` parameter - handle for `MPI_Wait` and `MPI_Test` to wait or check for completion
- `MPI_Isend` returns before buffer safe for reuse

Message Probing

- `MPI_Probe(src, tag, comm, status)` - fills status without receiving, inspect buffer size
- `MPI_Iprobe(src, tag, comm, flag, status)` - flag signals whether message available
- `MPI_Get_count(status, type, count)` - sets count to number of `type` items described by status

Collective Operations

- `MPI_Bcast(buf, count, type, root, comm)` - broadcast of `buf` from `root` to all processes in `comm`
- `MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)` - i-th chunk of size `sendcount` from `sendbuf` is sent to `recvbuf`
- `MPI_Allreduce(sendbuf, recvbuf, count, sendtype, op, comm)` - reduces `sendbuf` from all within `comm` point-wise into everyone's `recvbuf`

Communicator Splitting:

: `MPI_Comm_split(old, colour, key, new)` - processors are split based on whether key == colour.

TBB

- Task scheduler manages thread pool with task stealing
- `parallel_for(range, body)`
  - Range
    - copy constructor and a destructor
    - defines `is_empty`
    - defines `is_divisible`
    - defines splitting constructor `R(R &r, split)`
    - predefined ranges: `blocked_range`, `blocked_range2d`
  - Body
    - copy constructor and a destructor
    - defines `operator`
  - `parallel_for(0, N, addone)` - runs `addone` for i in 0..N
- `parallel_reduce`
  - Range: same as `parallel_for`
  - Body
    - splitting constructor and destructor
    - defines `operator`
    - defines `join`

Linda

- global, content-addressable memory
- processes run asynchronously, have six operations
- `out(exp1, exp2, ..., expN)` - add to tuple space
- `in("Green", ?y, ?r, FALSE)` - take from tuple space
- `rd(template)` - check existence of tuple
- `eval(exp1, exp2, ..., expN)` - add to tuple space, expX ran in separate thread
- non-blocking `inp`, `rdp` - return immediately