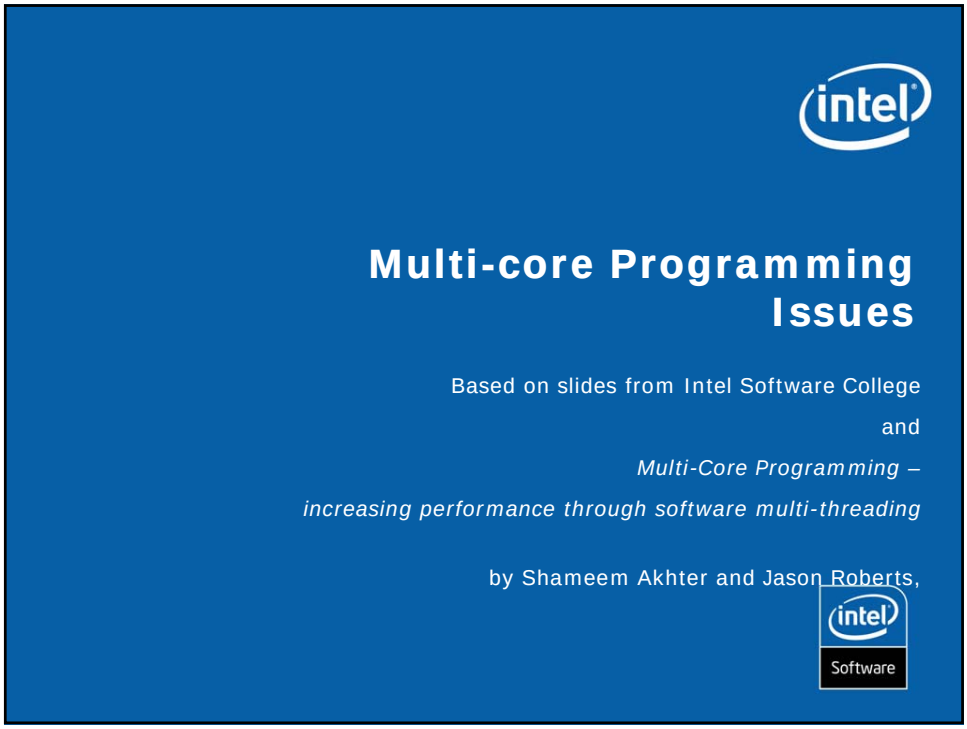

Safety & Liveness

System = Objects + Activities
- Two complimentary sets of correctness concepts
- Object-Centric: Safety
- Activity-Centric: Liveness

Safety
- "Nothing bad ever happens to the object"
- Can be extended to Reusability

Liveness
- "Any started activity eventually runs to the end"
- Can be extended to Performance

Safety & Liveness CONFLICT

Safety & Liveness Failures

Safety Failures
- lead to Unintended Behavior

Liveness Failures
- lead to No Behavior

Difficult to diagnose
- Non-deterministic (differs from runtime to runtime)
- Debugging probes can mask race conditions
- May only manifest as slight numerical deviation
- Runtime Behavior depends on
  - Hardware Platform (concurrent -> parallel)
  - OS (scheduling & timeslice)
  - MRTE implementation

Safety Failures

Causes
- Multiple threads accessing shared data
- Execution order is assumed but not guaranteed

Failures
- Race Condition
  - Read-Write Conflict
  - Write-Write Conflict
- Round-Off error

Liveness Failures

Causes

- Locking (on semaphores)
- Waiting (on monitors)
- Input (I/O based method waits)
- CPU contention
- Method Failure

Failures

- Deadlock
- Livelock
- Missed Signals
- Nested Monitor Lockouts
- Starvation
- Resource Exhaustion
- Distributed Failure

Round-off Errors

Computer arithmetic done with finite number of digits; already have errors

In parallel, operations may not be done in serial order

- can generate different results

Given: ε (a very small number such that 1.0 + ε = 1.0)

In floating point arithmetic we have that

-1.0 + (1.0 + ε) = -1.0 + 1.0 = 0.0

and that

(-1.0 + 1.0) + ε = 0.0 + ε = ε

Considerations before Multithreading

Can you apply Parallel Design on Higher Levels?
- Virtualization
- Parallel Processes
- Parallel Services on application server

Why is higher level better?
- Less Coding effort, Deployment without threading
- Less Communication required – less engineering effort
- Deployment flexibility
- Lower multithreading overhead

Other Considerations

While multithreading be aware, that your Threads ...
- Are not the only threads
  - concurrency on every level
- Do useful job when running
- Use waits if no useful job
- Are safe to shared data
- Are:
  - Balanced,
  - Ordered,
  - Prioritized for Performance

Multi-core Programming: Basic Concepts What's Been Covered

Processes and threads

Parallelism and concurrency

Design concepts
- Threading for throughput or turnaround?
- Threading for functionality or performance?
- Decomposing the work

Correctness concepts
- Race Conditions and Synchronization
- Deadlock

Performance concepts
- Speedup and Efficiency
- Granularity and load balance