Interfaces for Runtime Correctness Checking of Parallel Programs

Joachim Protze (protze@itc.rwth-aachen.de)

Motivation

- OpenMP 3 introduced tasks (2008)
- Several data race detection tools for OpenMP tasks popped up just last year

- How can we effectively reduce the porting effort for new programming paradigms?

Memory accesses

Concurrency

Synchronization

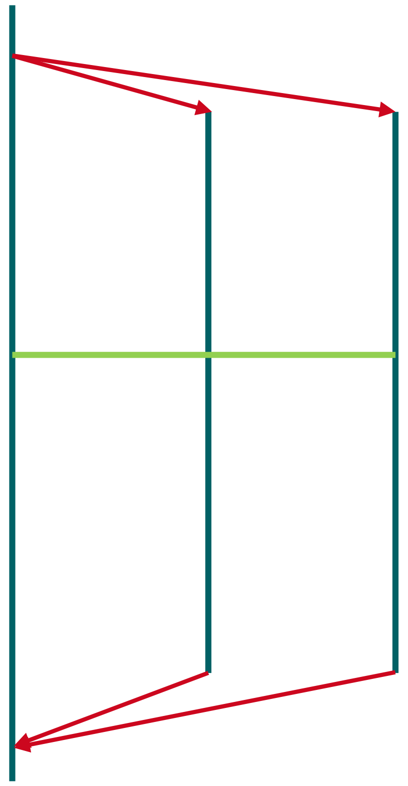
Synchronization in OpenMP

Parallel region

- Encountering a `parallel` directive *happens before* execution of the parallel region

- Encountering a `barrier` directive *happens before* execution of code following the barrier region

- Encountering the implicit barrier *happens before* the master continues code following the parallel region



parallel-begin

implicit-task-begin

barrier-begin

barrier-end

barrier-begin

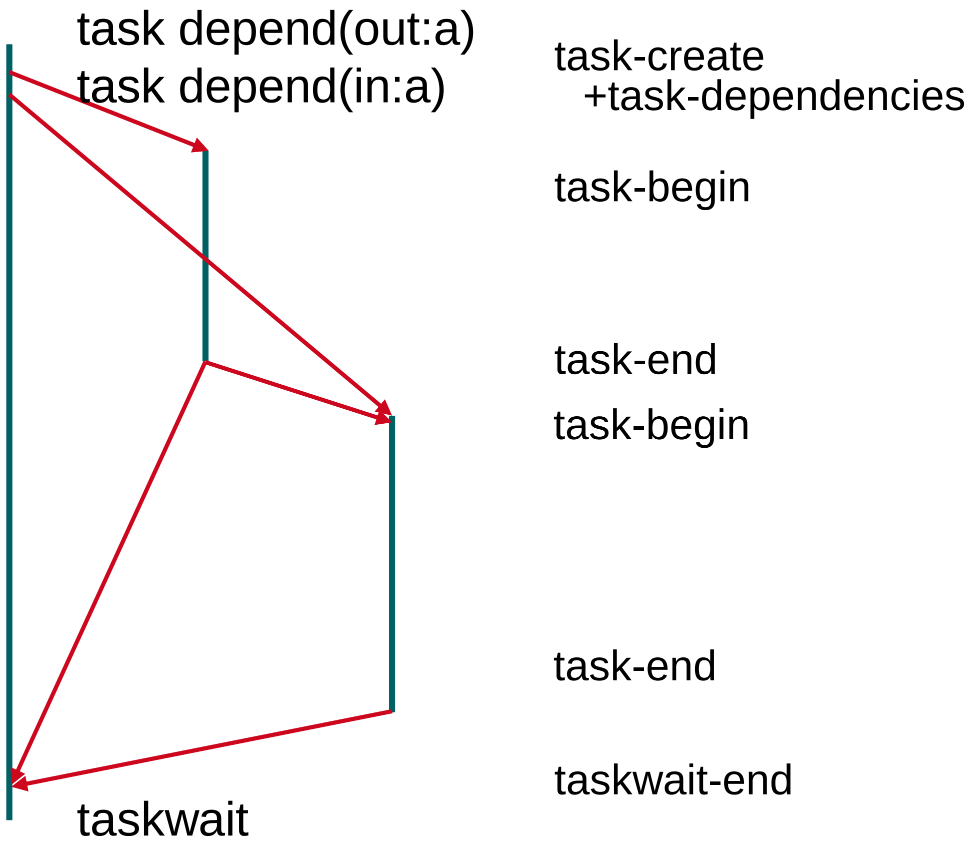
~~implicit-task-end~~ !

parallel-end

Synchronization in OpenMP

Task region

- Encountering a `task` directive *happens before* execution of the task region
- Finishing execution of a child task *happens before* execution of code following a taskwait, barrier, or taskgroup region
- Finishing a predecessor task *happens before* a dependent task starts execution
- Deferring a task *happens before* scheduling the task again


task depend(out:a)
task depend(in:a)
taskwait

task-create
+task-dependencies
task-begin
task-end
task-begin
task-end
taskwait-end

Archer based on ThreadSanitizer

- ThreadSanitizer comes with clang and gcc (-fsanitize=thread)

- Compiler instrumentation of memory accesses
  - Less overhead than binary instrumentation (e.g., PIN, valgrind)

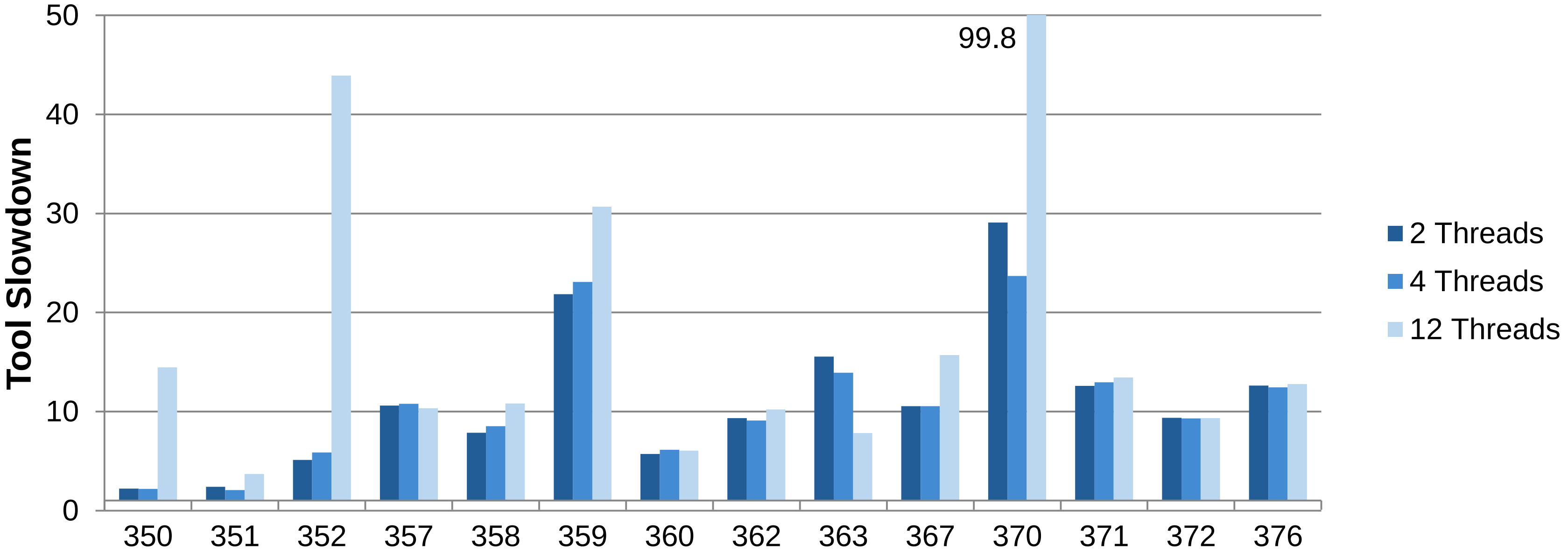
- ThreadSanitizer is not aware of OpenMP synchronization

- Happens before analysis with simplified *fast track* algorithm.
  - 4 records of memory access to a word, storing (epoch,tid,r/w)

- Archer annotates OpenMP synchronization
  - Initially instrumentation of the LLVM/OpenMP runtime
  - Now based on OMPT events

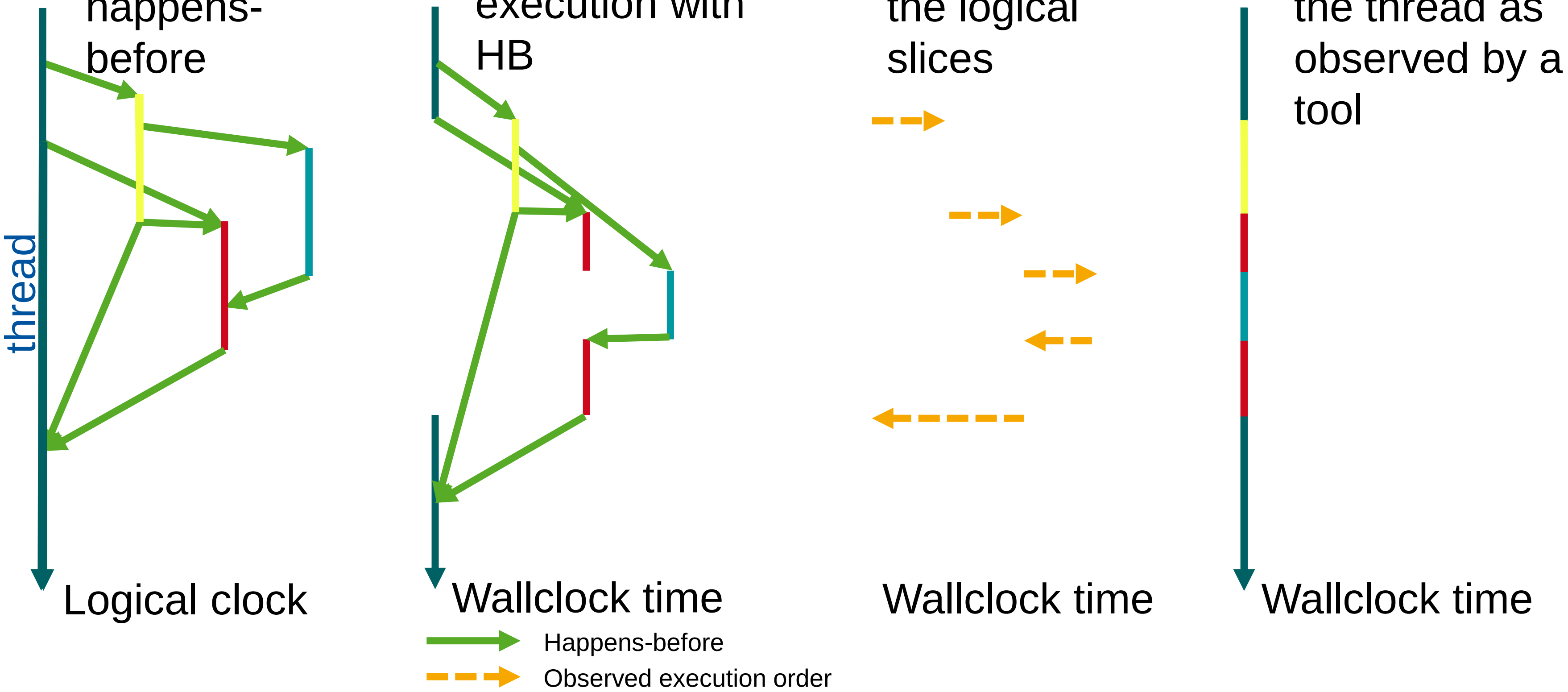
Data race analysis overhead for SPEC OMP 2012 (train)

- Expected overhead according to base tool: 2-20x
- 359.botsspar and 370.mgrid331 > 20x
  - Both run <1 second with high synchronization rate
    - 359.botsspar: 353400 task switches
    - 370.mgrid331: 6383 parallel regions



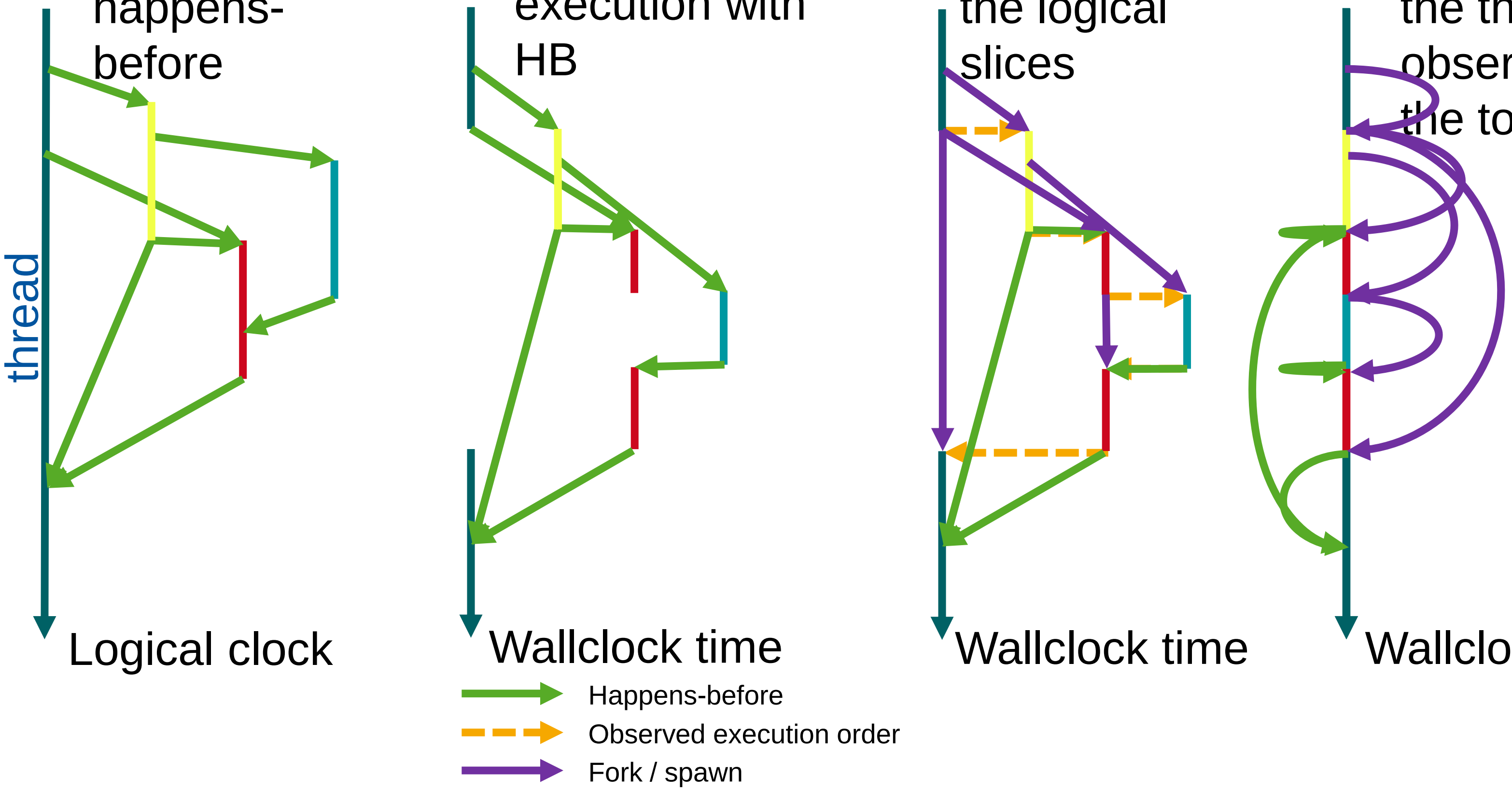
Concurrency for OpenMP Tasks

- Lamport happens-before
- Observed actual execution with HB
- Separating the logical slices
- Execution of the thread as observed by a tool



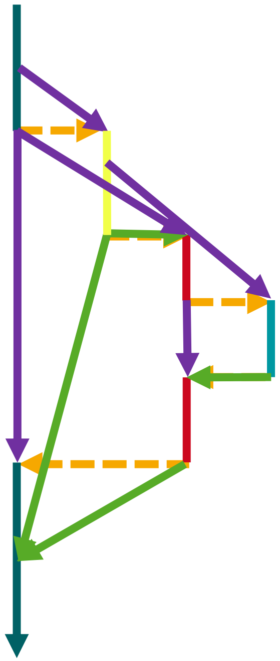
TLC: Marking execution within a thread as concurrent

- Lamport happens-before
- Observed actual execution with HB
- Separating the logical slices
- Execution of the thread as observed by the tool



Generic events

- Fork(curr, *new)
  - Fork(curr, *new, *msg)
- Join(curr, next)
- Switch(curr, next)
  - Switch(curr, next, msg)
- Send(curr, *msg)
- Recv(curr, msg)



Concurrency / Synchronization in Shared Memory

Parallel, Tasks, Loops

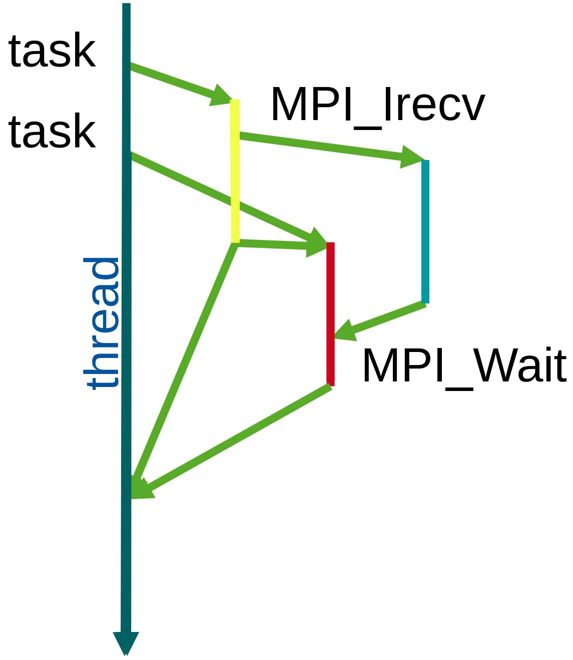
- Fork → P2P synchronization, concurrency
- Join → P2P synchronization
- Barrier → global synchronization
  - Can translate into N2N synchronization
- Dependencies → P2P synchronization
- Locks → ?
  - Should be flexible to enable lock-set and HB analysis
- Parallel loop → concurrency for each iteration
- Doacross loops → P2P synchronization

Fork(curr, *new)
Join(curr, next)
Switch(curr, next)
Send(curr, *msg)
Recv(curr, msg)

Applying this semantics to MPI

MPI Non-Blocking

- MPI_Isend / MPI_Irecv → concurrency, P2P synchronization
  - Bind the new execution unit handle to the request
- MPI_Wait → synchronize
- Buffer access → read/write



Applying this semantics to MPI

MPI One-sided

- MPI One-sided epochs → concurrency, P2P synchronization
- MPI One-sided target completion → synchronize

- Remote memory access → read/write

Device Offloading

Basic memory operations in device offloading

- Memory access
- Alloc/release memory
- (Dis-)Associate memory
- Update memory (memcopy)

OpenMP mapping semantics:

• Alloc	alloc + associate
• Map-to	((alloc +) associate +) update to device
• Map-from	update from device (+ disassociate (+ release))
• Update-to/from	update to/from device
• Release	disassociate + release

Challenge: semantics of global/static memory

Distributed Memory ?

Thank you for your attention.