Speculative parallelization: combining architectures and compilers to parallelize sequential code without knowing what is safe to run in parallel.
Key idea: Task trees for effective parallelization

Prior work: chains of task spawns

- Data dependence results in many aborted tasks
- If dependence is violated, all later tasks abort and re-execute
- Serial task spawn & commit

Task trees avoid serial bottlenecks

- Independently spawned leaf tasks enable selective aborts
- Distributed spawn & commit

Spawners
Workers
Dependence ⇒ abort single leaf task

ISCA 2020 T4: COMPILING SEQUENTIAL CODE FOR EFFECTIVE SPECULATIVE PARALLELIZATION IN HARDWARE
T4: Trees of Tiny Timestamped Tasks

T4 compiler systematically uncovers fine-grained parallelism
- Timestamps encode order, let tasks spawn out-of-order
- Trees unfold branches in parallel for high-throughput spawn
- Efficient parallel spawns support tiny tasks (tens of instructions)
- Tiny tasks can exploit locality, reduce communication

T4 exploits the Swarm architecture [Jeffrey et al. MICRO’15]
- Tasks appear to run sequentially, in timestamp order
- Selectively aborts dependent tasks
- Distributed task units can
  » Spawn and commit many tasks per cycle
  » Run hundreds of concurrent speculative tasks
Parallelizing entire real-world programs

T4 automatically divides a whole program into tasks
- Tasks boundaries at loop iterations and function calls

T4 introduces novel code transformations:
- Progressive loop expansion
- Call stack elimination
- Optimizations to make task spawns cheap
- Spatial-hint generation

T4 scales hard-to-parallelize C/C++ benchmarks from SPEC CPU2006
- Modest overheads: 31% on 1 core
- Speedups up to 49× on 64 cores

T4 is open source

swarm.csail.mit.edu