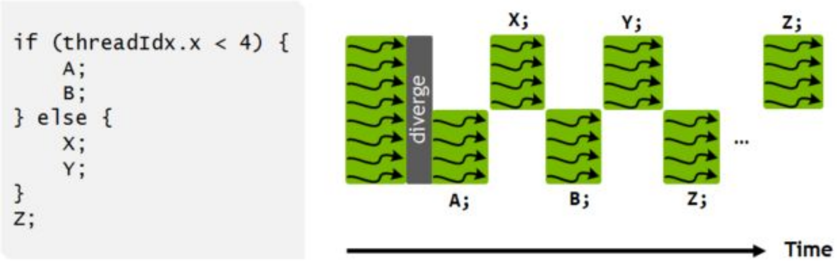
Synchronized Actor in detray

Beomki Yeo

Thread divergence in detray propagator

- In the SIMT model, If there is a conditional branch, instructions for different branches are only operated serially by disabling the threads not on the current branch



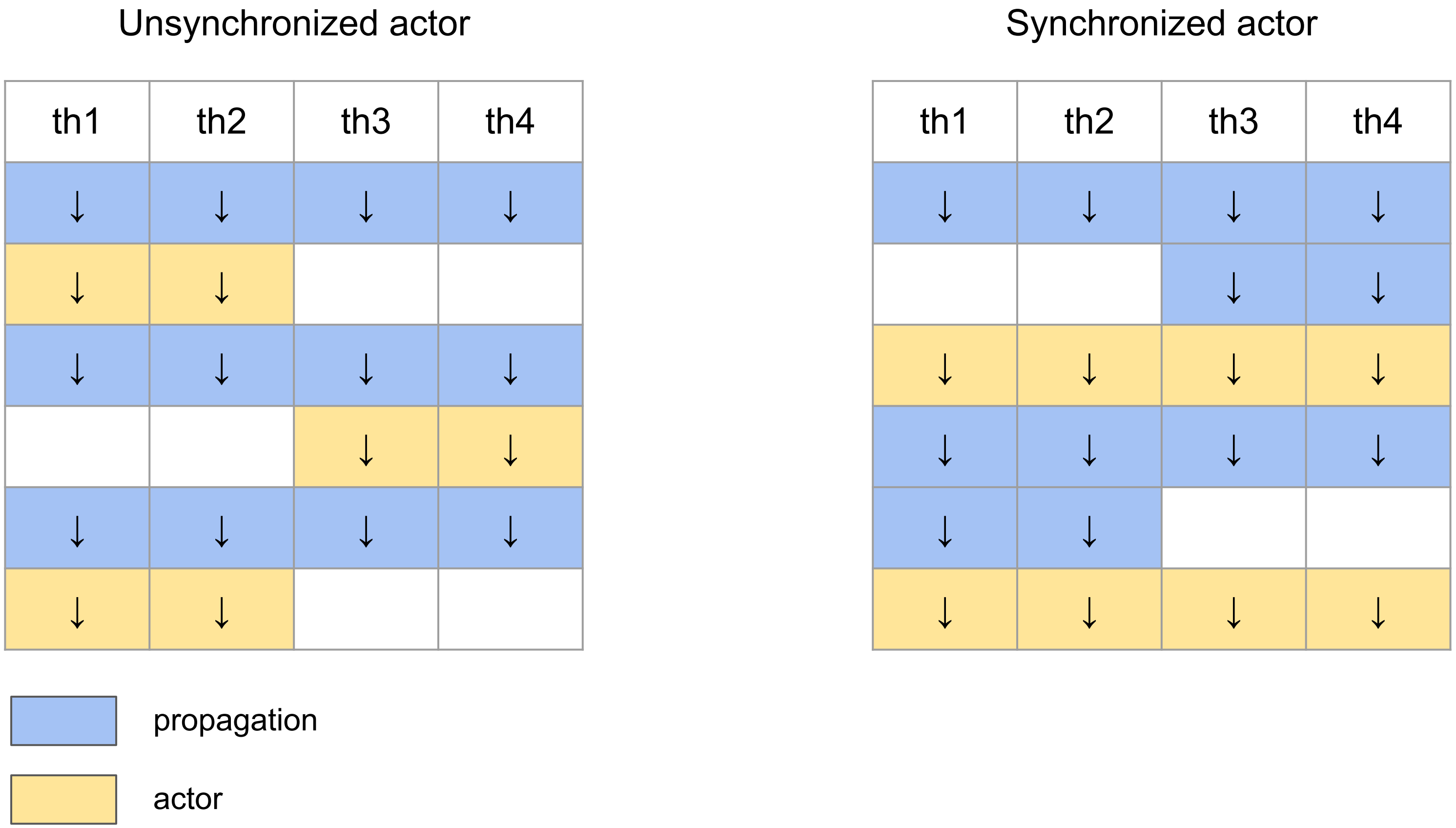
```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

- This thread divergence happened frequently in detray propagator, where some threads are on the surfaces (actor operation) and the others are still stepping toward surfaces

- Could introduce a performance degradation in algorithms with complicated actor chain (e.g. Combinatorial Kalman filtering)

- I somehow managed to implement detray propagate function that synchronizes actor operation on surface (PR#387)

Unsync vs. sync actor



Unsynchronized actor

th1	th2	th3	th4
🟦 ↓	🟦 ↓	🟦 ↓	🟦 ↓
🟨 ↓	🟨 ↓		
🟦 ↓	🟦 ↓	🟦 ↓	🟦 ↓
		🟨 ↓	🟨 ↓
🟦 ↓	🟦 ↓	🟦 ↓	🟦 ↓
🟨 ↓	🟨 ↓		

Synchronized actor

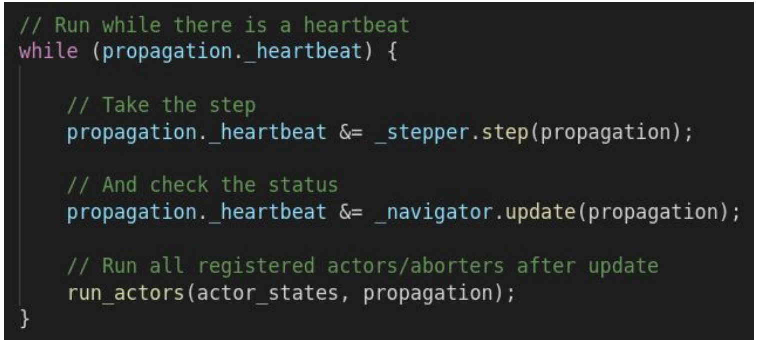
th1	th2	th3	th4
🟦 ↓	🟦 ↓	🟦 ↓	🟦 ↓
		🟦 ↓	🟦 ↓
🟨 ↓	🟨 ↓	🟨 ↓	🟨 ↓
🟦 ↓	🟦 ↓	🟦 ↓	🟦 ↓
🟦 ↓	🟦 ↓		
🟨 ↓	🟨 ↓	🟨 ↓	🟨 ↓

🟦 propagation

🟨 actor

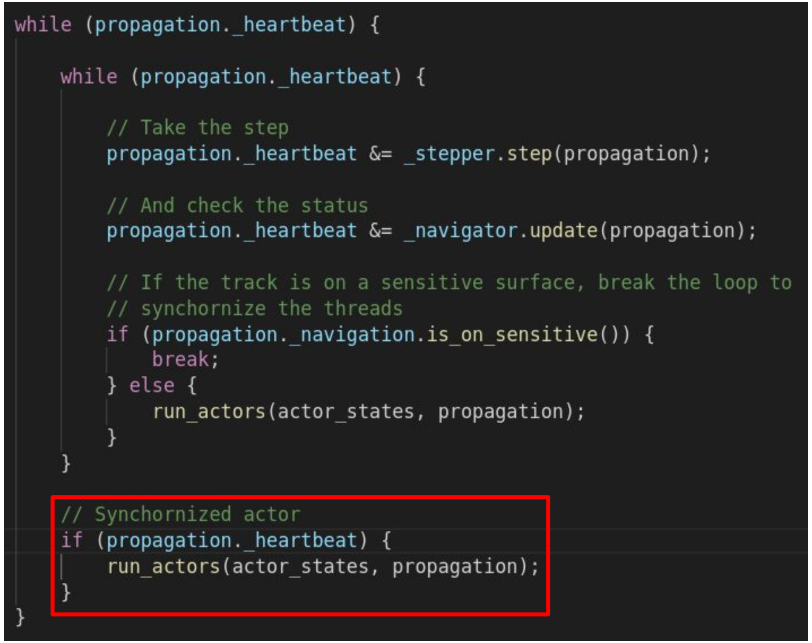
Propagate function implementation

Unsynchronized actor



```
// Run while there is a heartbeat
while (propagation._heartbeat) {

    // Take the step
    propagation._heartbeat &= _stepper.step(propagation);

    // And check the status
    propagation._heartbeat &= _navigator.update(propagation);

    // Run all registered actors/aborters after update
    run_actors(actor_states, propagation);
}
```

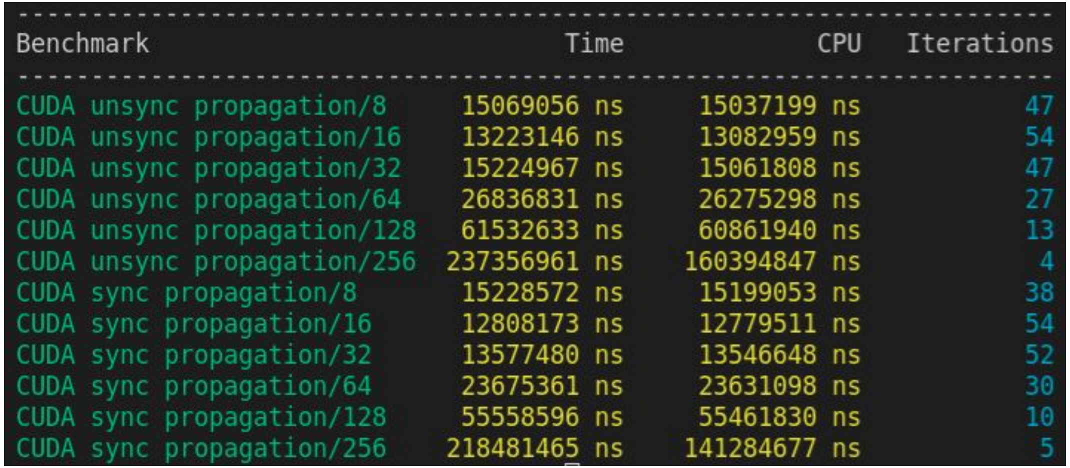
Synchronized actor



```
while (propagation._heartbeat) {

    while (propagation._heartbeat) {

        // Take the step
        propagation._heartbeat &= _stepper.step(propagation);

        // And check the status
        propagation._heartbeat &= _navigator.update(propagation);

        // If the track is on a sensitive surface, break the loop to
        // synchornize the threads
        if (propagation._navigation.is_on_sensitive()) {
            break;
        } else {
            run_actors(actor_states, propagation);
        }
    }

    // Synchornized actor
    if (propagation._heartbeat) {
        run_actors(actor_states, propagation);
    }
}
```

Benchmark

- Benchmarked with actor chain that includes covariance transport and material interaction
  - ~10 % improvement



```
----------------------------------------------------------------------
Benchmark                              Time             CPU   Iterations
----------------------------------------------------------------------
CUDA unsync propagation/8      15069056 ns     15037199 ns           47
CUDA unsync propagation/16     13223146 ns     13082959 ns           54
CUDA unsync propagation/32     15224967 ns     15061808 ns           47
CUDA unsync propagation/64     26836831 ns     26275298 ns           27
CUDA unsync propagation/128    61532633 ns     60861940 ns           13
CUDA unsync propagation/256   237356961 ns    160394847 ns            4
CUDA sync propagation/8        15228572 ns     15199053 ns           38
CUDA sync propagation/16       12808173 ns     12779511 ns           54
CUDA sync propagation/32       13577480 ns     13546648 ns           52
CUDA sync propagation/64       23675361 ns     23631098 ns           30
CUDA sync propagation/128      55558596 ns     55461830 ns           10
CUDA sync propagation/256     218481465 ns    141284677 ns            5
```

- Average number of active threads per warp
  - Unsync: 14.0
  - Sync: 15.4

Outlooks

- Might be interesting if we can test this against combinatorial kalman filtering

- The performance of synchronized actor will be degraded in case there are many steppings between the surfaces
    - Need to avoid the constraint step size if possible