oneAPI DevSummit at SC

oneAPI

UNIVERSIDAD DE MÁLAGA

DAC

Enhancing Online Planning under Uncertainty via Bloom Filter Based Memory

Denisa Constantinescu* Angeles Navarro Rafael Asenjo

Juan-Antonio Fernández-Madrigal Ana Cruz-Martín

*dencon@uma.es

Motivation

Solve real world sized decision making & planning under uncertainty

- mobile platforms
- low-power
- real time response

Online planning for robot navigation use-cases

- autonomous robot navigation
- service robotics

e.g. healthcare, logistics, rescue missions

oneAPI solution

- CPU+GPU execution
- easy to program
- portable

Problem definition as POMDP

- Partially Observable Markov Decision Process
- *Stochastic* actions, sensors have uncertainty
- *Behavior* defined as reward function
- *Policies* map beliefs to actions
  - action = policy(belief state)
  - maximizes reward R(s,a)

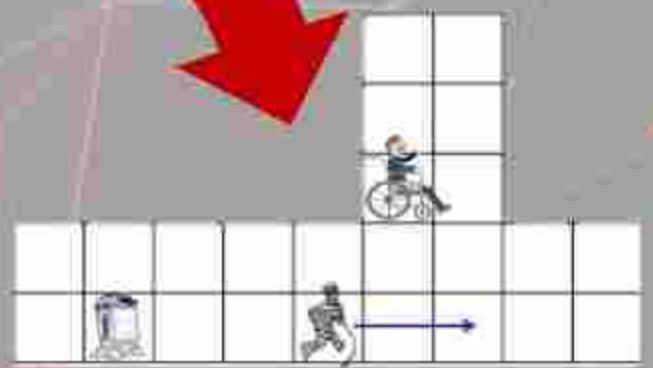
Defined as n-tuple { S, A, Z, T, O, R }:

S = state set	*T: Pr(s'|s,a)* = state-to-state transition probabilities
A = action set	*O: Pr(z|s,a)* = observation generation probabilities
Z = observation set	*R(s,a)* = reward function

observation < state => *belief state*

Intelligent agent

observe



action 1

?

new belief state b' and reward r'

action 2

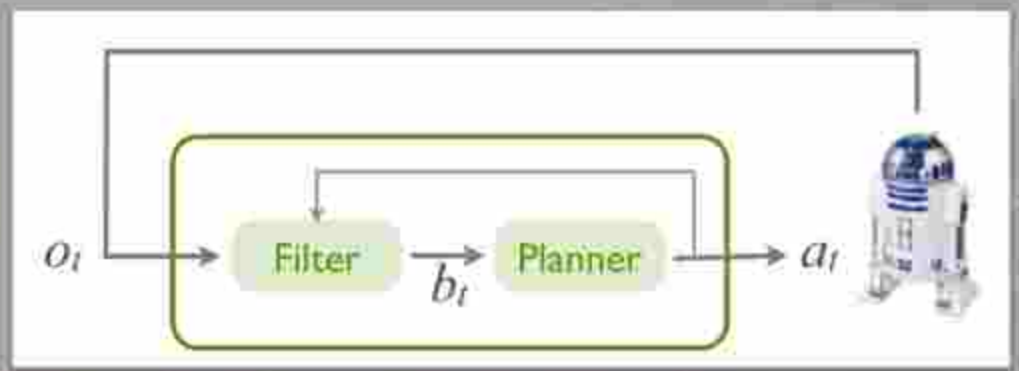
new belief state b" and reward r"

...

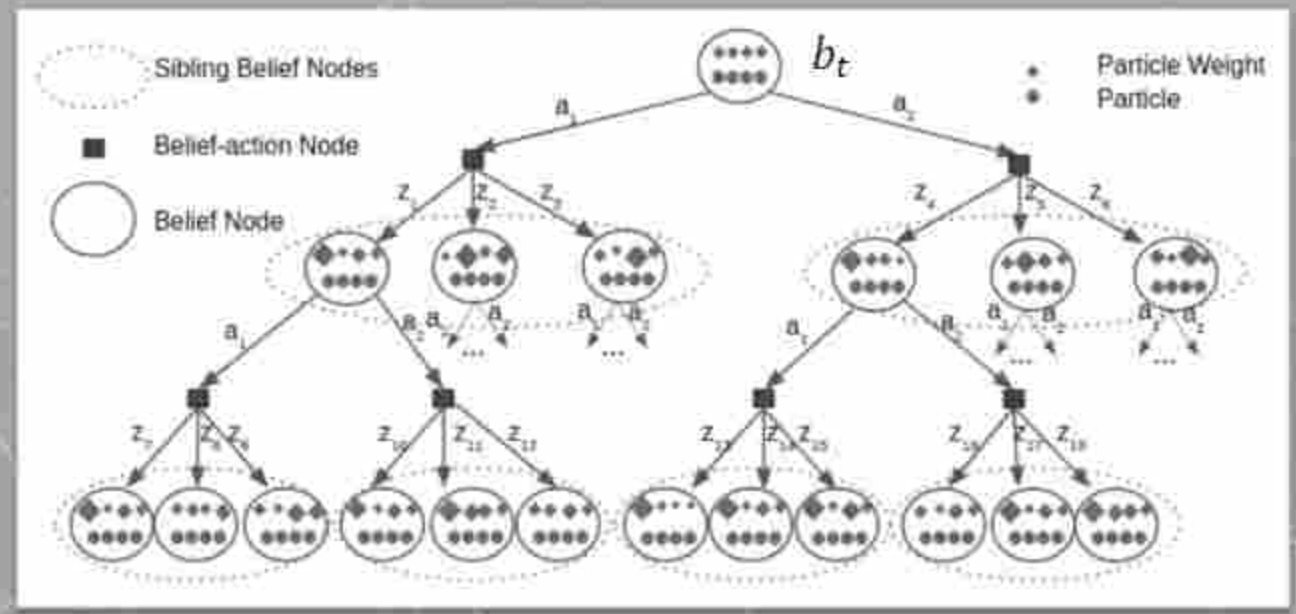
Online planning with DESPOT-α

Repeat while target not reached:

1. **Plan –** build decision tree from current belief, b_t, and choose action with highest reward
2. **Execute** – only first step of the plan, a_t
3. **Update** – actualize belief with new observation, o_t (particle filter)
4. **Discard** search tree



Online planning



We use state of the art DESPOT-α online planner*

*Garg, N.P., Hsu, D. and Lee, W.S., 2019, June. DESPOT-Alpha: Online POMDP Planning with Large State and Observation Spaces. In *Robotics: Science and Systems*.

Reusing policies computed by online methods…

Challenges

- Unlikely to encounter the same belief twice
- Limited memory, computing power & energy
- Belief trees grow exponentially
- Cannot store everything
- Fast response time required
- Portable solution

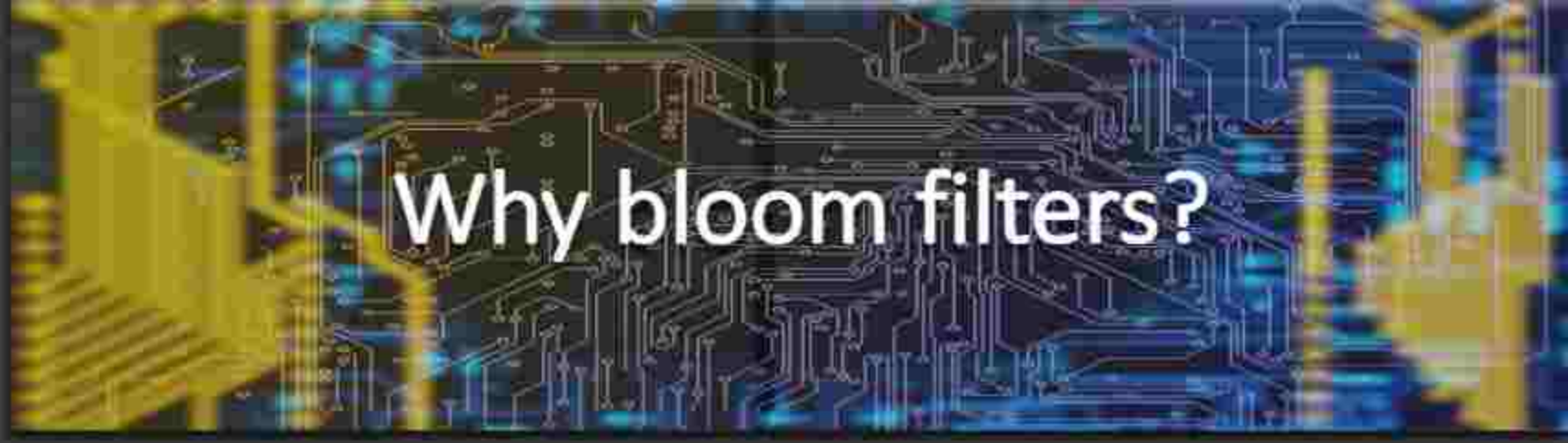
Reusing policies computed by online methods…

Challenges

- Unlikely to encounter the same belief twice
- Limited memory, computing power & energy
- Belief trees grow exponentially
- Cannot store everything
- Fast response time required
- Portable solution

Mitigation strategies

- Lightweight memory based on bloom filters (BFs)
- Store condensed representation – belief markers
- Locality-sensitive hashing
- Similarity search to recall experience
- Avoid unnecessary searches in memory
- Store only reachable experience from current belief
- Parallel implementation on low-power SoC
- oneAPI



Why bloom filters?

Probabilistic data structure, we use

- ***A bit vector per dimension of POMDP belief***
- ***Locality-sensitive hash functions***
  - Encode belief markers (mean & std deviation)

Interesting properties

- ***Time*** to add and check element membership O(K)
- ***Concurrency*** – independent hashes
- ***Compact*** representation

Recall-Planner: Algorithm

Input:

model = {S, A, T, Z, R, γ} ; // pomdp model
memory_trace(n_bfs, m) ; // memory trace history
memory ; // experience memory
c_th ; // a belief certainty threshold
s_th ; // a similarity threshold
s_target ; // the target state
t ; // max time for planning per time step
d ; // max simulation depth in the decision tree
r_d ; // max decision tree depth for retaining experience
$b = b_0$; // initial belief - set of of N weighted particles

```
while s_target is not reached do
    if certainty(b) > c_th then
        b_f = extract_features(b) // mean, std
        b_marker = sim_hash(b_f, n_on, m)
        if b_marker & memory_trace == 1 then
            a,v = memory.recall(b_marker,s_th)
        else
            a,v = solve(model, b, t, d, r_d)
            memory.store(b_marker, a, v)
            memory_trace | = b_marker
    else
        a, v = solve(model, b, t, d, r_d)
    execute(a)
    o = observe()
    b = update(b, o)
```

Parallel implementation with oneAPI DPC++ Library* (oneDPL)

```
#include <oneapi/dpl/algorithm>
#include <oneapi/dpl/execution>
#include <oneapi/dpl/iterator>
#include <CL/sycl.hpp>
...
class Memory {
    std::vector<ExperienceNode> experienceNodes;
    ...
    ExperienceNode *recall(bitset<2048> *currBeliefMarker) {
        size_t size = experienceNodes.size();
        vector<int> similarityScores(size);
        // 1. compute similarity score for all experience entries/nodes in memory vs current belief
        {
            cl::sycl::buffer bufSim(similarityScores);
            cl::sycl::buffer bufExp(experienceNodes);
            auto start  = oneapi::dpl::make_zip_iterator(similarityScores.begin(), experienceNodes.begin());
            auto end    = oneapi::dpl::make_zip_iterator(similarityScores.end(), experienceNodes.end());
            auto policy = oneapi::dpl::execution::make_device_policy(queue(gpu_selector{}));
            // auto policy = oneapi::dpl::execution::make_device_policy(queue(cpu_selector{}));
            std::for_each(policy, start, end, [=](auto similExperienceTuple) {
                using std::get;
                get<0>(similExperienceTuple) = get<1>(similExperienceTuple).similarityScore(currBeliefMarker);
            });
        }
        // 2. get index of entry with the highest similarity to current belief & return corresponding experience
        int maxOverlapID = std::max_element(similarityScores.begin(), similarityScores.end()) - similarityScores.begin();
        return &experienceNodes.at(bestOverlapID);
    }
    ...
}
```

*https://spec.oneapi.io/

Evaluation – testbeds

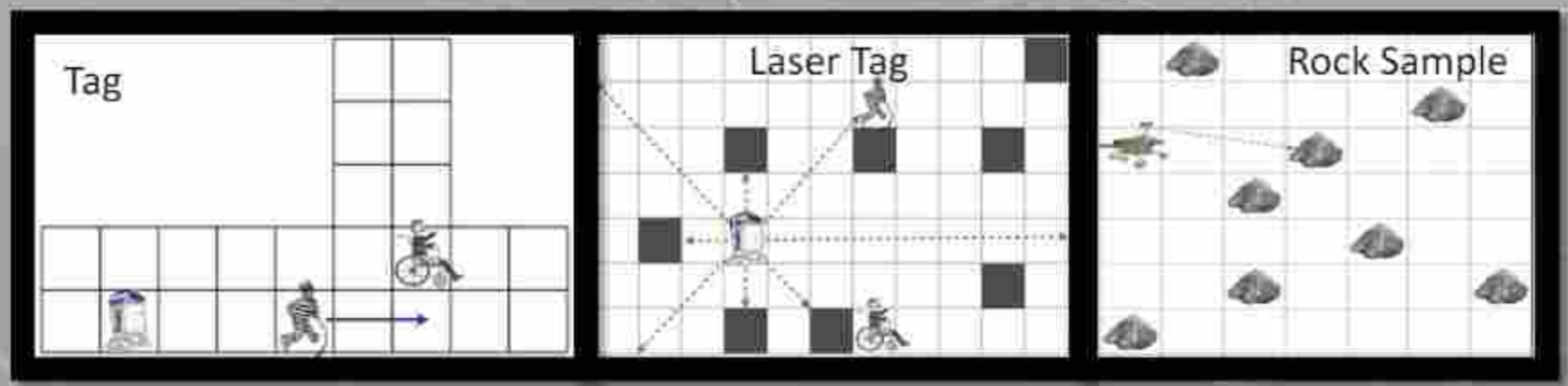
Platform	CPU	GPU	RAM	TDP (Watt)
Kaby Lake	i5-8250U @1.60GHz 4 cores	UHD 620 @300MHz 24 EUs	8GB of DDR4	10 to 15W
Tiger Lake	i7-1165G7 @2.8GHz 4 cores	Iris(R) Xe @1.3Ghz 96 EUs	16GB of LPDDR4x	12 to 28 W

Software

- oneAPI DPC++ 2021.4 Compiler
- Intel(R) NEO Graphics Driver
- Intel VTune Profiler
- Ubuntu 18.04 & 20.04

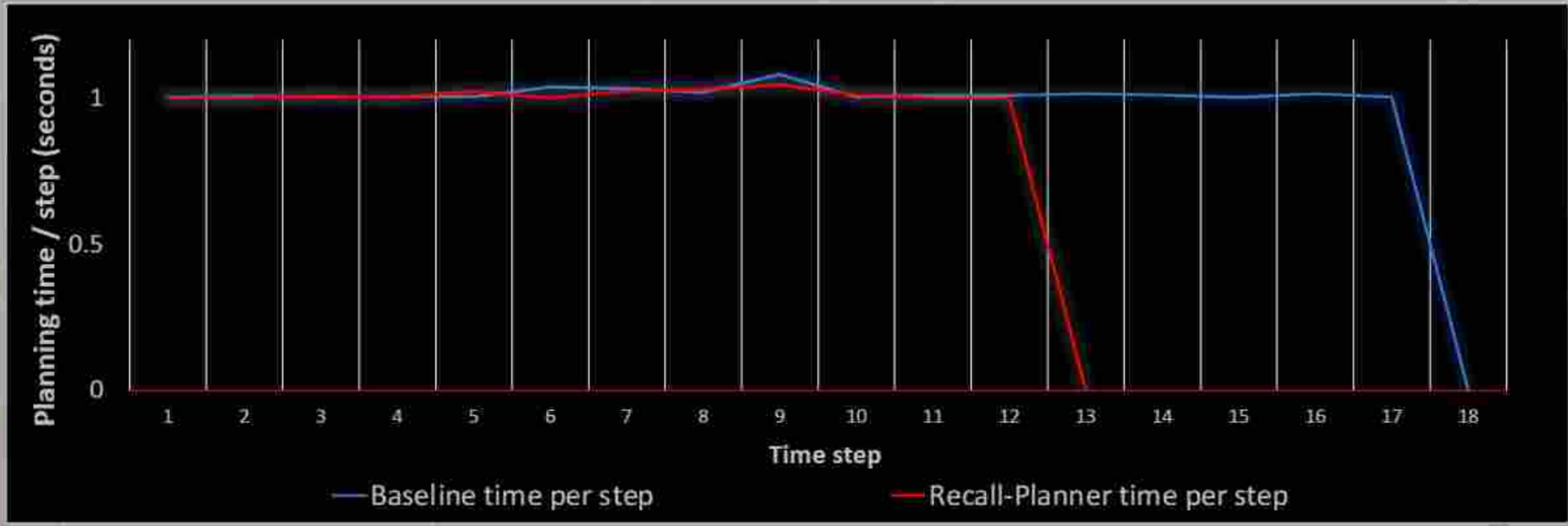
Some examples of POMDPs - benchmarks

	Tag	Laser Tag	RockSample(7,8)	RockSample(11,11)
$\|S\|$	870	4830	12544	247808
$\|A\|$	5	5	13	16
$\|Z\|$	30	1.5×10^6	3	3

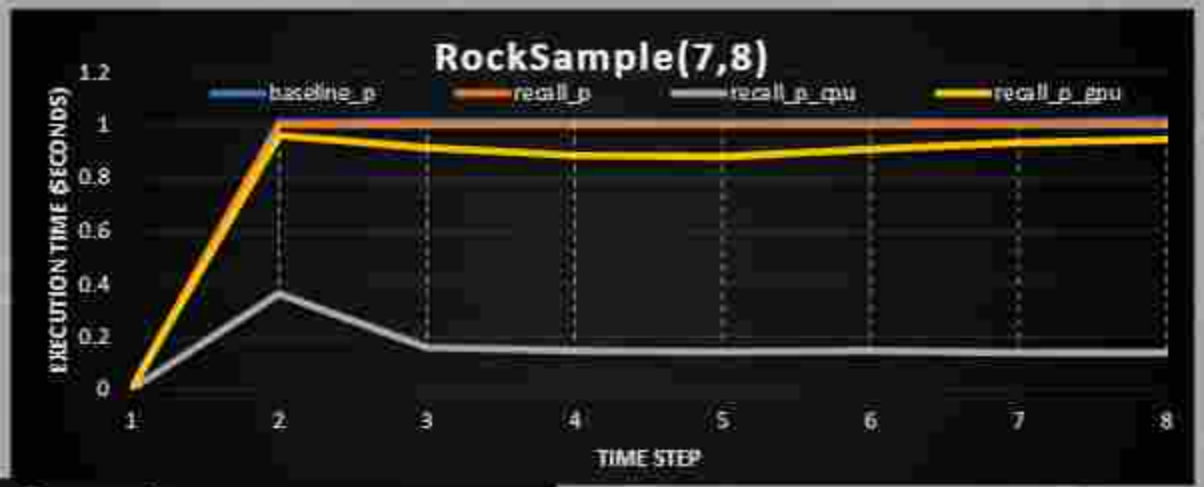


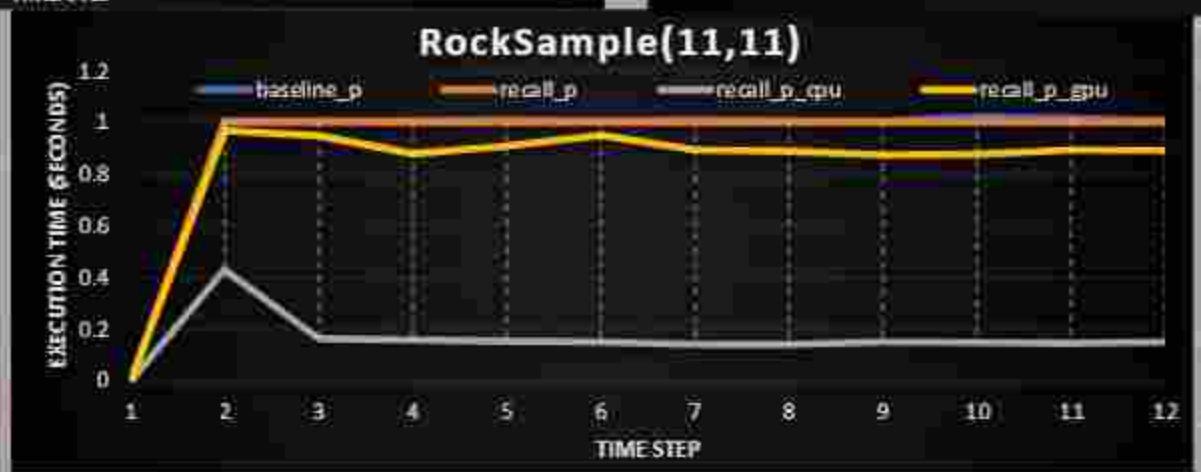
Metrics for performance

- **Number of time steps** the agent needs to reach the target state – *less is better*
- Total discounted **reward** (inversely proportional with # time steps) – *more is better*
- Execution **time** – *less is better*

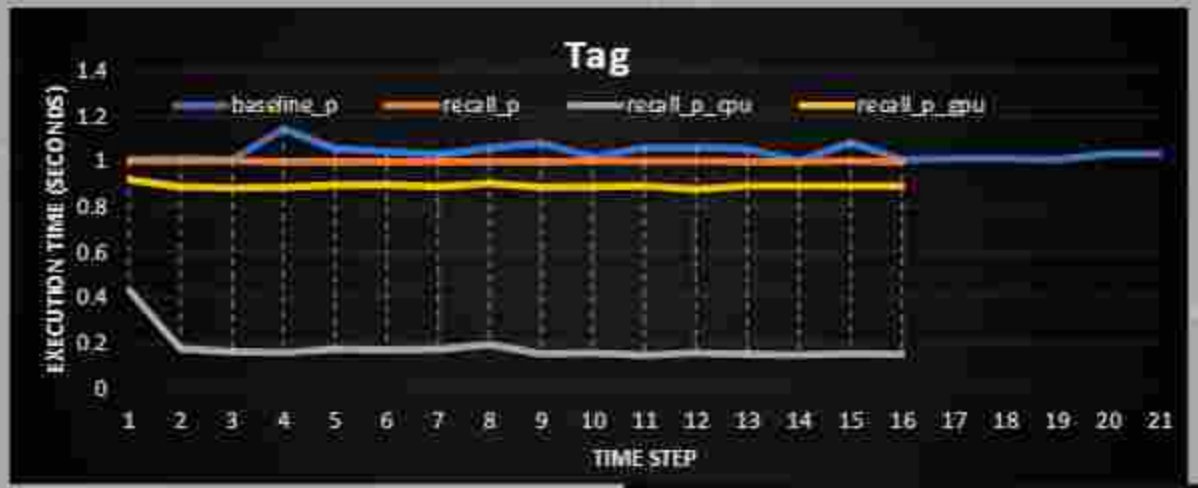


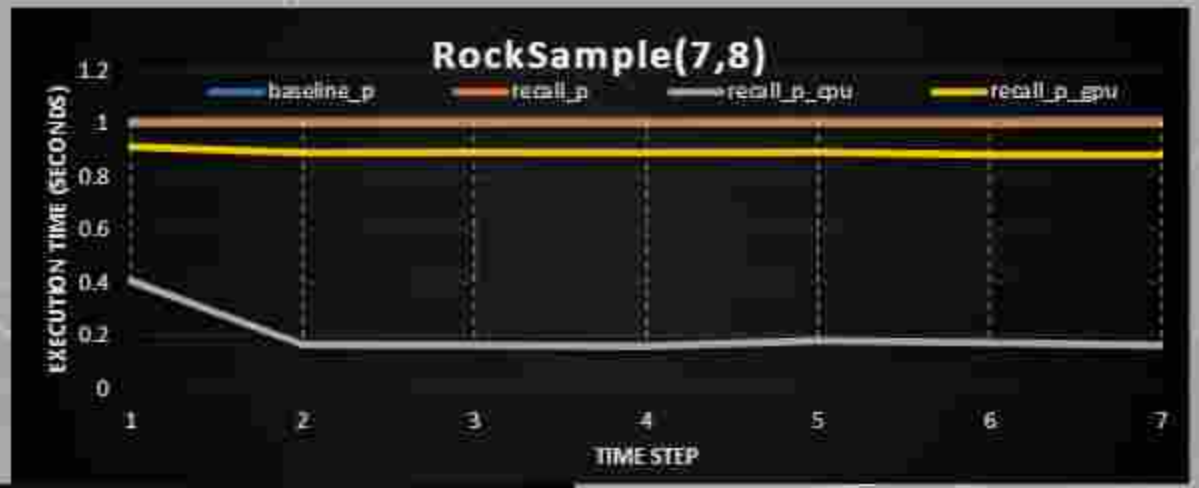
Parallel Recall-Planner vs Baseline on Kaby Lake



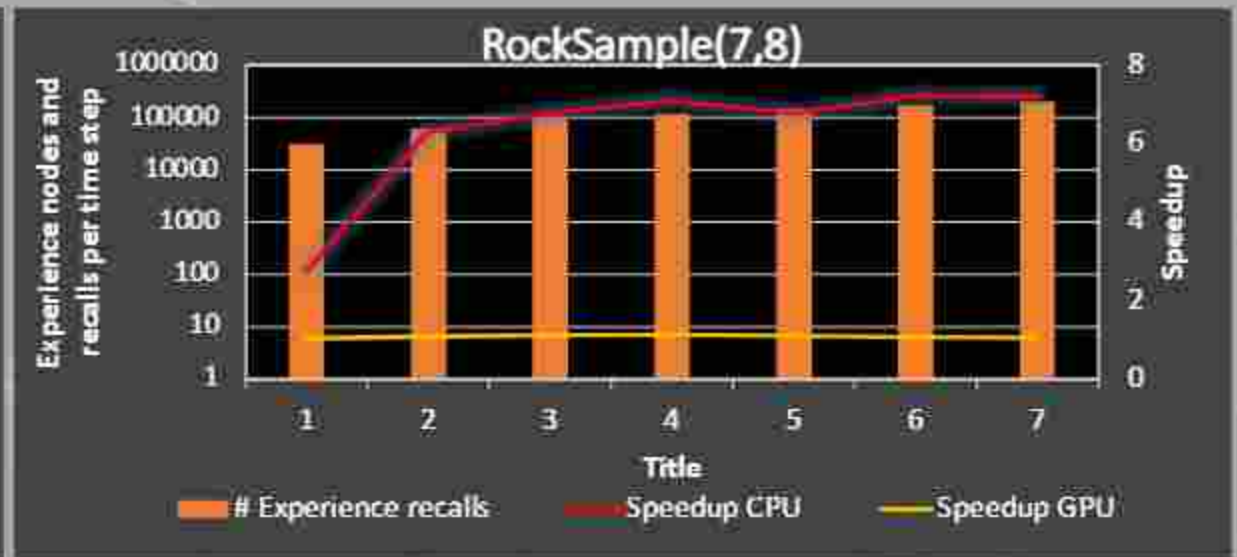


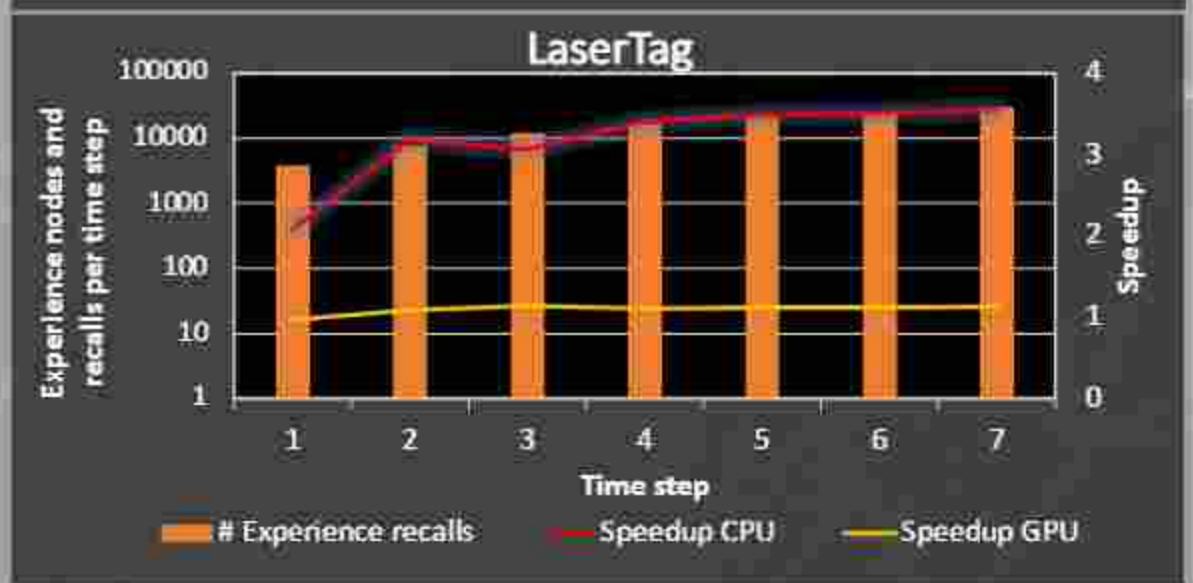
Parallel Recall-Planner vs Baseline on Tiger Lake

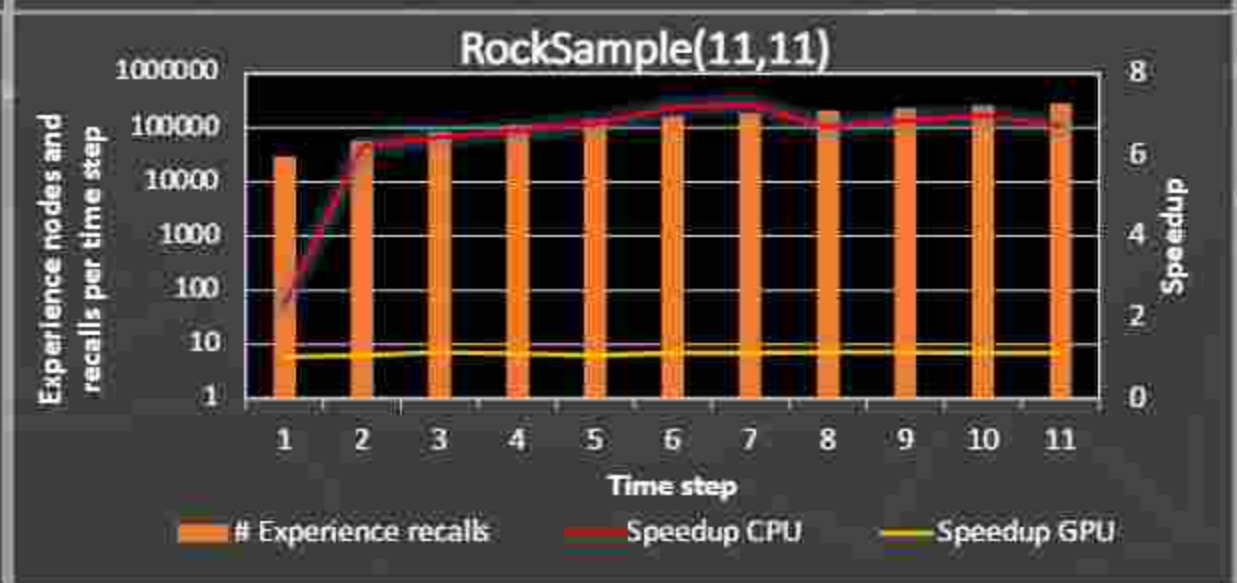




Speedup vs # experience recalls







Conclusions

- New design for online planning
- oneDPL makes it easy to implement efficient and portable code
  - Up to 7x speedup w.r.t state of the art
- New opportunities
  - Real time for mobile platforms
  - Interface on-policy & off-policy
  - Learning from example

Future work

- Heterogeneous CPU+GPU implementation
- Energy evaluation
- Forget function for Bloom Filter Memory
- Social navigation use cases on a robot

Questions?

- Problem – efficient online planning for mobile robots
- Context – POMDPs, online planning
- Proposal – online planning + bloom filter memory
- Evaluation on CPU+GPU mobile SoC
- Discussion

oneAPI DevSummit at SC

oneAPI

UNIVERSIDAD DE MÁLAGA

DAC

Enhancing Online Planning under Uncertainty via Bloom Filter Based Memory

Denisa Constantinescu* Angeles Navarro Rafael Asenjo

Juan-Antonio Fernández-Madrigal Ana Cruz-Martín

*dencon@uma.es

Thanks for your attention!