Parallelizing POMCP to Solve Complex POMDPs

Semanti Basu, Sreshtaa Rajesh, Kaiyu Zheng, Stefanie Tellex, R. Iris Bahar
Dept. of Computer Science, Brown University, Providence, RI 02912

***Abstract*—In the real world, robots must be able to plan and operate under uncertainties. To that end, Partially-Observable Markov Decision Processes (POMDPs) are a popularly used architecture that lets us model planning problems where system states are only partially known. Planning algorithms are then used to solve POMDPs. Partially-Observable Monte Carlo Planning (POMCP) is one such planning algorithm that has been shown to be very successful in solving POMDPs, especially under larger numbers of simulations. However, POMCP becomes prohibitively slow and expensive over large and complex domains, making it unsuitable for many real world applications. This paper outlines an effort to develop a parallelized version of POMCP to aid in faster decision making in complicated scenarios. Our long term goal is to implement parallel POMCP in hardware so it can be deployed on robots for fast and accurate planning in uncertain environments.**

I. Introduction and Overview

The objective of this study is to develop a parallelized version of POMCP [6], which has been widely successful in planning under uncertainty [3, 9], to enable robots to plan in real time across large and complex domains. POMCP, which is based on Monte Carlo planning, usually relies on running a large number of simulations using a blackbox world model in order to achieve performance. The objective of solving a POMDP problem is to find a policy that maximizes return making it a popular tool to model optimal control problems. POMCP has been very successful in solving POMDPS. However, in larger domains (such as most real-world problems) that are more complex, POMCP takes an impractically long time to run a sufficient number of simulations for acceptable results. DESPOT [8], another online planning algorithm often used for solving POMDPs, was parallelized using a combination of CPU and GPU parallelization for the belief search tree in [1]. We are extending the concept of parallelization to POMCP, where we hope to achieve similar performance while decreasing the time taken to achieve those results, making POMCP more suitable for use in complex domains.

One of the major challenges in parallelizing POMCP is the fact that the algorithm is inherently sequential—there exist distinct stages: (1) sampling from the current belief state (2) conducting the look-ahead tree search (3) selecting an action, and receiving an observation that is then used to update the belief state. In this manner, each action selected is dependent on all the previous actions taken and observations received.The challenge to parallelizing it is to find a way to divide the computational expense without altering the core sequential decision-making which makes POMCP [6] so robust.

In this work, at a high level, we are exploring different techniques for parallelizing POMCP in software. POMCP conducts a modified Monte-Carlo Tree Search (MCTS) to select an action that is asymptotically optimal at each step by constructing a search tree of histories. Building the search tree requires running repeated simulations which is the most computationally expensive part of POMCP. We attempt to speed up the search tree construction and action selection by extending techniques used for MCTS parallelization [2] to POMCP. To that end, we are exploring *root parallelization* and *tree parallelization*. Our hypothesis is that even for simpler problems, parallelizing POMCP should make the algorithm run faster than the serial on large domains in terms of runtime and perform no worse than the serial version in terms of cumulative reward achieved.

Currently, we have modified the original C++ implementation of POMCP in Silver and Veness [6] to incorporate root parallelization to speed up action selection. We benchmarked the effectiveness of the parallel version developed on Rocksample, a commonly used POMDP benchmark domain in literature [7]. Preliminary results on large Rocksample domains show that even on this simpler problem, the parallel version selects actions faster than the serial version without a significant reduction in reward.

II. Technical Approach

A. Introduction to POMDP and POMCP

In POMDP [4] architecture, uncertainty is represented as a probability distribution over system states. This is called "belief." To define a POMDP problem, a tuple $\langle S, A, O, R, T, \Omega, \gamma \rangle$ is defined where S represents the set of possible system states, A gives us the set of possible actions, Ω is the set of all possible observations, O is the probability distribution over all possible observations, R is the reward function, T is the transition model, and γ is the discount factor. At time t, the environment state is s_t. When the agent takes an action a_t, the environment state transitions into s_{t+1} and receives observation o_{t+1} and reward r_{t+1}. The history h_t at any time step t is the sequence of action-observation pairs up to t, such that $h_t = (a_0, o_1, a_1, o_2, \cdots, a_{t-1}, o_t)$.

A *policy* π is a probability distribution over actions, given history. $\pi_t(h, a) = P(a_t = a | h_t = h)$. The return $R_t = \sum_{l=t}^{\infty} \gamma^{l-t} r_l$ is evaluated by measuring the discounted reward obtained from time t on wards.The value function $V^{\pi}(h) = E[R_t | h_t = h]$ gives us the expectation of the return obtained on executing a policy, given history (h) at that time step. The executed action is dependent on the policy that maximizes the value function.

POMCP is an online planning algorithm to solve POMDP problems. It takes as input the current belief state, and the POMDP problem and returns the best action at that step.

A look-ahead tree search is carried out from the current belief state to determine the best action. The tree is pruned upon receiving an observation after the action is executed.A belief-state update is carried out through Monte Carlo particle filtering, as detailed in Silver's work [6].

B. POMCP Parallelization techniques

Action-selection is based on Monte Carlo Tree Search and as such requires a large number of simulations which is computationally costly in POMCP. Thus, we seek to explore at least two schemes for parallelizing POMCP: (1) Root parallelization and (2) Tree parallelization, both of which are based on techniques used for MCTS parallelization [2].

1) Root Parallelization: In serial POMCP, the look-ahead search tree is built by running a fixed number of simulations or by running simulations until timeout. In the parallel version we seek to obtain equivalent accuracy in action selection in less time by extending the concept of *root parallelization* commonly used to parallelize MCTS [2] to POMCP. In root parallelization for MCTS, several trees are built simultaneously, and after a certain amount of time has passed, the results are merged to select a move. In serial POMCP, let us suppose the total number of simulations is t. In root-parallel POMCP, k trees are built in parallel, each running t/k simulations. Each tree is built from a root state randomly sampled from the current belief. After the trees are built, the output is merged to generate an action for that time step. Several techniques can be explored for merging [5], however for the preliminary results, the most popular action was chosen. After selecting an action, POMCP executes it and receives an observation and reward. These values are used to update and prune the search trees built in parallel. The process of action selection is repeated until termination. In the case where there was a tie between multiple actions, a random action was chosen among them.

2) Tree Parallelization: In this type of parallelization, several branches of the tree are explored simultaneously. Here, the tree is built with the same number of simulations as the serial but ideally finishes in $\frac{1}{n}^{th}$ of the time (where n represents the number of branches simultaneously explored). We hope to compare this scheme of parallelization against the root parallelized implementation in the future.

C. Preliminary results

For the following plots, we ran POMCP on the Rocksample domain [7] with a 15×15 board containing 15 rocks. The number of simulations ranged from 2^8 simulations to 2^{17}. We averaged the reward and timestep values over 20 runs of each simulation size, or until a timeout of 10000 seconds was reached. As shown in Figure 1, when the number and complexity of simulations grows large, the average time-cost per action selection of the parallelized version of POMCP is less than that of the serial version for the steps before the algorithm converges. These per-step time savings accumulate to a significant time reduction in larger domains that require long planning horizons. Additionally, we observe that, when taking into account the inherent randomness of POMCP, there appears no significant reduction in reward between the serial and parallel versions (Figure 2). We also experimented on smaller Rocksample problems and observed similar trends.

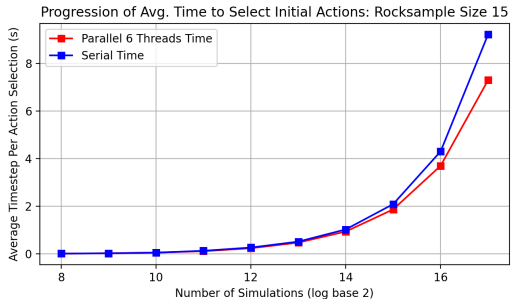

Fig. 1: Average time to select initial actions: Rocksample 15,15

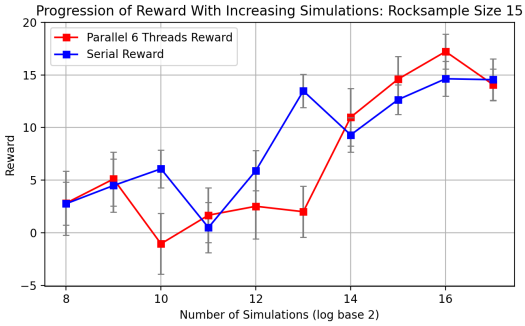

Fig. 2: Cumulative reward comparison: Rocksample 15,15

III. Conclusion and Next Steps

Currently, we have implemented a root-parallelized version of POMCP, and have benchmarked it against a serial implementation of POMCP on different-sized domains of a small problem, Rocksample. We plan to repeat this process with a tree-parallelized version of our code to see which method of concurrency provides the most significant performance gain (which we define as a reduction in time taken without compromising on reward achieved). Additionally, looking at the root-parallelized version itself, we would like to explore different ways of selecting the "best" action returned from individual threads. For our preliminary results, we simply chose the action that the majority of the threads returned, but there may be more optimal ways to select an action. In cases where there was more than one majority, a random mode was chosen. We believe that there may be more optimal ways to choose an action: for example, selecting the action with the highest average cumulative reward across threads.

The ultimate goal in parallelizing POMCP is to increase the practicality of using the algorithm to solve POMDPs in real-world applications. Thus, we plan to test the parallelized versions against the serial version's performance in more complex domains, such as robotic arm manipulation/grasping. Defining a complex task as a POMDP and integrating it with a real-world action-observation loop such as a robot will both be challenging; we might have to redesign parts of the code to efficiently integrate it with ROS, so we can apply it on a physical robot. Ultimately, we hope to implement POMCP in hardware and see whether the performance gain translates from software to the real world.

References

[1] Panpan Cai, Yuanfu Luo, and David Hsu. Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. 02 2018.

[2] Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel monte-carlo tree search. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Computers and Games*, pages 60–71, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87608-3.

[3] Jean-Alexis Delamer, Yoko Watanabe, and Caroline P.C. Chanel. Safe path planning for uav urban operation under gnss signal occlusion risk. *Robotics and Autonomous Systems*, 142:103800, 2021. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2021.103800. URL https://www.sciencedirect.com/science/article/pii/S0921889021000853.

[4] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

[5] Richard B. Segal. On the scalability of parallel uct. In H. Jaap van den Herik, Hiroyuki Iida, and Aske Plaat, editors, *Computers and Games*, pages 36–47, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-17928-0.

[6] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf.

[7] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, page 520–527, Arlington, Virginia, USA, 2004. AUAI Press. ISBN 0974903906.

[8] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/c2aee86157b4a40b78132f1e71a9e6f1-Paper.pdf.

[9] Yiming Wang, Francesco Giuliari, Riccardo Berra, A. Castellini, A. D. Bue, A. Farinelli, M. Cristani, and F. Setti. Pomp: Pomcp-based online motion planning for active visual search in indoor environments. *ArXiv*, abs/2009.08140, 2020.