*ECAI 2016*
*G.A. Kaminka et al. (Eds.)*

*doi:10.3233/978-1-61499-672-9-1726*

# GPU-Accelerated Value Iteration for the Computation of Reachability Probabilities in MDPs

**Zhimin Wu**[1] and **Ernst Moritz Hahn**[2] and **Akın Günay**[1] and **Lijun Zhang**[2] and **Yang Liu**[1]

## 1 INTRODUCTION

Computation of reachability probabilities is an important subroutine for determining approximately optimal policies of MDPs [3]. Value iteration (VI) [2] is a well-known method to compute these values. However, sequential implementation of VI is computationally expensive both in terms of time and memory. Hence, we propose a highly parallel version of VI to solve general MDPs utilizing the GPU, which is widely used in the recent years to accelerate execution performance of various computational methods in many areas [1, 7, 6]. Our approach explores algebraic features (e.g., matrix structure) of MDPs, and uses action-based matrices to achieve massive parallelism for efficiency. We empirically evaluate our approach on several case studies. Our results show that we can achieve up to 10X$\sim$ speedup compared to sequential VI, and outperform topological value iteration (TVI) [4] in most of the cases. Particularly, for MDPs which do not contain strongly connected components (SCCs) with more than one state, or which contain a small number of large SCCs, our approach achieves up to 17X speedup compared to TVI.

Our main contributions are: (1) We take advantage of the algebraic structure of MDPs to define action-based matrices and corresponding data structures for efficient parallel computation of reachability probabilities on GPUs. (2) We develop an efficient parallel VI algorithm for computing reachability probabilities that utilizes features of modern GPUs, e.g., dynamic parallelism and memory hierarchy.

## 2 BACKGROUND AND RELATED WORK

An *MDP* is a tuple $M = (S, s_{init}, Act, P, R)$, where $S$ is a finite set of *states*, $s_{init} \in S$ is the *initial state*, and $Act$ is a finite set of *actions*. The (partial) *transition probability function* $P\colon S \times Act \mapsto Dist(S)$, where $Dist(S)$ is the set of discrete probability distributions over the set $S$, assigns probability distributions to combinations of states and actions. The *reward function* $R\colon S \times Act \mapsto Dist(R)$ assigns a numeric reward to each state/action pair. By $Act(s) = Dom(P(s, \cdot))$ we denote the actions that are *activated* in state $s$. We require $|Act(s)| \geq 1$ for every $s \in S$.

Given an MDP $M$, we are interested in computing the minimal (maximal) probability to reach a set of target states $T \subseteq S$ in an infinite horizon, formally: $P_{\min}(s, T) = \inf_{\alpha \in Adv} Prob^{\alpha}(s, T)$. $Adv$ is the set of all *schedulers*, which choose the action to be performed in a state depending on the sequence of states and actions seen so far. $Prob^{\alpha}(s, T)$ is the probability of reaching $T$ when starting from state $s$ and following the scheduler $\alpha$. The computation process is defined by the following equation:

$$x_s^n = \min_{\alpha \in Act(s)} \sum_{s' \in S} P(s' \mid s, a) \cdot x_{s'}^{(n-1)} \text{ for } s \notin T, n > 0 \qquad (1)$$

*Value iteration* [2] is a general dynamic programming method to solve MDPs. It is an iterative computation process to update the value function of every state, which follows Equation 1 and terminates when it satisfies a convergence criterion.

There are some existing approaches which optimize VI using graphical features of MDPs. From these approaches, TVI [4] is the most relevant one to our approach. TVI utilizes the SCC structure of an MDP to construct an acyclic MDP for performing VI backups in the best order, and to only perform them when necessary. While TVI is based on the structure of SCCs in MDPs, our approach utilizes the algebraic features of MDPs that are related to the representation matrix of MDPs and the matrix-vector multiplication during the Bellman backup process. In addition, our approach is independent from the structure of SCCs, which may affect TVI's efficiency.

GPUs have several advantages over CPUs such as high memory bandwidth, computation capability, and massive parallelism. To the best of our knowledge, our approach is novel both in terms of fully utilizing the algebraic features of MDPs and parallel computing techniques in GPU to improve efficiency of VI for the computation of reachability probabilities.

## 3 GPU ACCELERATED VALUE ITERATION

We build a GPU accelerated parallel VI that can significantly improve the efficiency compared to the sequential VI by taking advantage of the algebraic structure (matrix) of MDPs and the matrix operation involved in the Bellman backup process.

In sequential VI, the complete state space should be backed up in each iteration, which requires the exploration of all states sequentially. More specifically, the term $\sum_{s' \in S} P(s' \mid s, a) \cdot x_{s'}^{(n-1)}$ in Equation 1 requires the sequential VI to compute the reachability probabilities for each state by exploring each enabled action and reachable states. This exploration process creates a bottleneck for sequential VI, since the time required for the exploration process grows exponentially with respect to the state space of an MDP. Hence, if we can accelerate the exploration process, we can significantly improve VI's efficiency.

As we show in Figure 1, an MDP can be represented using a matrix structure $\mathcal{M}$. Each row involves a set of sub-vectors, where each sub-vector represents the immediate reacability probabilities of the states with respect to actions. In VI, the calculation of Equation 1 updates $x_i$ by selecting the action $a$ that maximize/minimize the vector to vector multiplication $Succ_{a_i} \times \mathcal{X}$. This computation is independent in each state. Furthermore, we collect the vector to vector multiplication of the complete state space together, and then divide by the ac-

[1] Nanyang Technological University, Singapore
[2] Institute of Software, Chinese Academy of Sciences, China

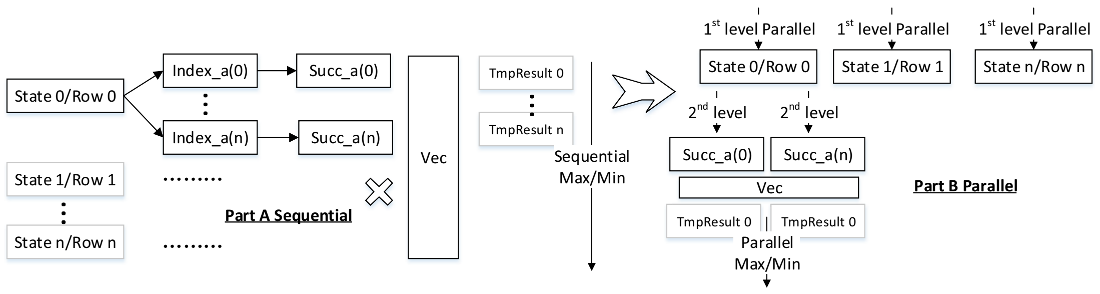


**Figure 1.** Parallelization

tions. We can find in global perspective that computation is indeed a substantial amount of synchronized sub-matrices to vector multiplications. The sub-matrices are the matrices representation of MDPs by actions. Thus, we define the *Action-based Matrices for MDP*.

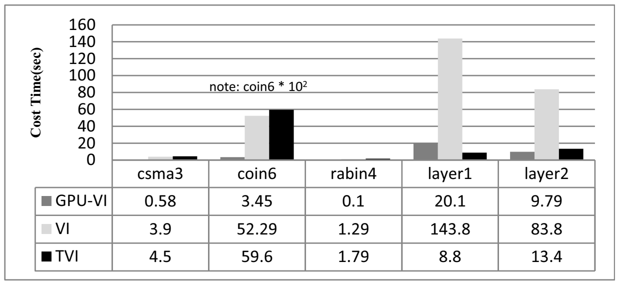
**Definition 1** ***Action-based Matrices*** *Given a MDP* $M = (S, s_{init}, Act, P, R)$ *and the representation matrix* $\mathcal{M}$*, for each action* $a \in Act$*, an action-based matrix is a tuple* $(\mathcal{M}_a = \{S_a, S'_a, P(S' \mid S, a)\})$*, where* $S_a$ *is the set of states in which action* $a$ *is activated,* $S'_a$ *is the set of states reached via action* $a$ *from states in* $S_a$*, and* $P(S' \mid S, a) \rightarrow (0, 1]$ *is the probability array.*

Each $\mathcal{M}_a$ is a $m_a$ by $m_a$ matrix, where $m_a = |S_a|$. Intuitively, an $\mathcal{M}_a$ represents the transition relationship of the states in $S_a$ with respect to action $a$. Using the action-based matrices, the value iteration process can be transformed into several interleaving action-based matrix to vector multiplications and subsequent minimisation, where each action-based matrix to vector multiplication is an independent computation. Hence, the action-based matrices and the described partitioning of matrix operations allow us to develop an efficient parallel VI that works well on GPUs. We also design compact data structures and parallel convergence detection for our approach.

## 4 IMPLEMENTATION AND EXPERIMENTS

We evaluate the performance of our approach by comparing it with VI and TVI implementations of Dai et al. [4]. We implemented our approach in CUDA C. We conducted our experiments on a computer with two Intel(R) Xeon(R) CPU E5-2670, 2.60GHz, 16GB RAM and a Geforce Titan Black GPU. We set the parallelism to 512 threads in one block, which can reach 100% occupancy per multiprocessor according to the CUDA Occupancy Calculator [5].

The results are shown in Figure 2. Our approach achieves around 10X∼ speed up comparing to VI in MDPs with different structure. It can be observed from Figure 2 that TVI performs slightly worse with MDPs that have large number of small SCCs, since under this condition, TVI cannot reduce the backup times significantly, and the SCC detection brings additional cost. This situation is also addressed by Dai *et al.* [4]. We can see our approach can achieve up to 17X speedup compared to TVI under this condition. For the layered MDPs, we consider two strong layered MDPs. *Layered1* only has one transition between any two SCCs. The initial state is a state in the first layer and the goal state is an end state in the last layer. *Layered2* has the same size with *Layered1*. But the number of layers is considerably less than Layered1. The experiment results show that for *Layered1*, TVI outperforms our GPU accelerated VI slightly due to the large number of layers. But with *Layered2*, our approach outperforms TVI with around 1.5X speedup since the number of layers



| | csma3 | coin6 | rabin4 | layer1 | layer2 |
|---|---|---|---|---|---|
| ■ GPU-VI | 0.58 | 3.45 | 0.1 | 20.1 | 9.79 |
| ■ VI | 3.9 | 52.29 | 1.29 | 143.8 | 83.8 |
| ■ TVI | 4.5 | 59.6 | 1.79 | 8.8 | 13.4 |

**Figure 2.** Performance Evaluation

decreases considerably. Both our approach and TVI outperforms the VI for these two MDPs. In conclusion, our approach's performance is substantially better compared to VI for MDPs with all types of structures. We also achieves considerable speedup compared to TVI in MDPs which has small number of large SCCs or only SCCs with just one state. Although our approach performs slightly worse than TVI in MDPs with a deep layer, we can still conclude from the results that our approach can be more general to a wide range of MDPs.

## 5 CONCLUSION

We presented a novel GPU accelerated parallel value iteration approach for the efficient computation of reachability probabilities of MDPs. The main idea of our approach is to utilize the algebraic features of MDPs to divide the computation process of reachability probabilities into partitions, which can be computed in a massively parallel manner with an efficient parallelization granularity on GPUs. Our evaluation shows that we achieve 10X∼ speedup compared to sequential, and up to 17X speedup compared to TVI.

## References

[1] Ron Bekkerman, Mikhail Bilenko, and John Langford, *Scaling up machine learning: Parallel and distributed approaches*, Cambridge University Press, 2011.
[2] Richard Bellman, 'Dynamic programming and lagrange multipliers', *PNAS*, **42**(10), 767, (1956).
[3] Craig Boutilier, Thomas Dean, and Steve Hanks, 'Decision-theoretic planning: Structural assumptions and computational leverage', *JAIR*, **11**, 1–94, (1999).
[4] Peng Dai, Daniel S Weld, Judy Goldsmith, et al., 'Topological value iteration algorithms', *JAIR*, 181–209, (2011).
[5] CUDA NVIDIA, 'Gpu occupancy calculator', *CUDA SDK*, (2015).
[6] Anton Wijs, 'Gpu accelerated strong and branching bisimilarity checking', in *TACAS*, 368–383, Springer, (2015).
[7] Zhimin Wu, Yang Liu, Jun Sun, Jianqi Shi, and Shengchao Qin, 'Gpu accelerated on-the-fly reachability checking', in *ICECCS*, pp. 100–109, (2015).