Scalable Hardware Architecture for Real-Time Dynamic Programming Applications

Brad Matthews*, *Student Member, IEEE,* Itamar Elhanany+, *Senior Member, IEEE*
ECE Department, The University of Tennessee, Knoxville, TN 37922
email: {*bradmatthews,+itamar}@ieee.org

***Abstract*— This paper introduces a novel architecture for performing the core computations required by *dynamic programming* (DP) techniques. The latter pertain to a vast range of applications that necessitate an optimal sequence of decisions to be issued. An underlying assumption is that a complete model of the environment is provided, whereby the dynamics are governed by a Markov decision process (MDP). Existing DP implementations have traditionally been realized in software. Here, we present a method for exploitating the data parallelism associated with computing both the value function and optimal action set. An optimal policy is obtained four orders of magnitude faster than traditional software-based schemes, establishing the viability of the approach for real-time applications.**

I. Introduction

Dynamic Programming applies Markov decision processes (MDPs) formalism to determine the optimal policy, or control mechanism, in a dynamic stochastic environment. Existing research efforts [1][2] have traditionally focused on software-based realizations of DP techniques to determine the mapping of states to actions, or *policy*, required to derive an optimal controller. The probability of transitioning to a next state, s', given a current state/action pair, s and a, is defined as

$$P^a_{ss'} = \Pr\left[s_{t+1} = s' | s_t = s, a_t = a\right]. \quad (1)$$

However, it does not provide information regarding the expected reward that is to be received upon arrival at a given state. For a given current state/action pair, the expected reward for each possible subsequent state, $s\prime$, is given by

$$R^a_{ss'} = E\left[r_{t+1} | a_t = a, s_t = s, s_{t+1} = s'\right]. \quad (2)$$

The reward function embeds information that can be used to determine the value of the next state given that a specific action is taken. The long-term value associated with each state is provided by the *value function*. The latter is commonly defined as the expected sum of discounted rewards [3]. where γ is the discounting factor, given that the system starts in state s and follows policy π such that

$$V^\pi(s) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]. \quad (3)$$

The common goal in DP application is to derive an optimal policy that maximizes the value function. Utilizing (2) and (3), the optimal value function, in terms of the transition probabilities and reward function, can be expressed as

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s\prime)]. \quad (4)$$

TABLE I
Generic Memory Requirements for GPI

	States	Action(s)	Bits	Memory
Probability	225 (15^2)	11	12	6.683 Mbits
Rewards	225 (15^2)	11	9	5.012 Mbits

Attainment of the optimal policy is often achieved by utilizing the generalized policy iteration (GPI) algorithm [3]. GPI involves an *evaluation* phase, whereby a policy, $\pi(s)$, is evaluated to determine its value function, and an *improvement* phase, during which the policy is updated in a greedy manner with respect to the value function. In the evaluation phase, the value function improves according to the policy, whereas in the improvement phase, the policy is improved according to the value function. Through this successive iterative improvement process, it has been shown that convergence to the optimal value function is guaranteed, from which an optimal policy can be derived [4].

II. Optimization for Hardware Realization

A principle limitation of hardware design for the policy iteration algorithm pertains to its prohibitive storage requirements. It is apparent that storage of both the transitional probabilities, $P^a_{ss'}$, and the reward function, $R^a_{ss'}$, mandate an $O(S^2A)$ memory requirement. To illustrate the impact of this storage requirement, we will first consider an application that consists of a small state set, comprised of 225 states, and restrict the total number of actions that can be taken at each state to be 11. Furthermore, the transitional probabilities and rewards will be limited to 12-bit and 9-bit representations, respectively. The memory requirement, presented in Table I, needed to store the transitional probabilities and the reward function is 11.7 Mbits, which exceeds the on-chip memory resources available in current FPGA devices.

In order to realize the policy iteration algorithm in hardware, it is imperative that the storage requirements be reduced. To facilitate such reduction, we first rewrite the policy improvement expressions as

$$\begin{aligned} \pi(s) &= \arg\max_a \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V(s')\right] \quad (5) \\ &= \arg\max_a \left[G(s,a) + \sum_{s'} H^a_{ss'} V(s')\right] \end{aligned}$$

TABLE II
REDUCED MEMORY REQUIREMENTS

	States	Action(s)	Bits	Memory
$P^a_{ss\prime}$	225 (15^2)	11	7	3.89 Mbits
$R^a_{ss\prime}$	225 (15^2)	11	18	2970 Bits

where the value of the state-action pair, $G(s,a)$, is given by

$$G(s,a) = \sum_{s'} P^a_{ss'} R^a_{ss'} \tag{6}$$

and the discounted probability, $H^a_{ss'}$, is represented as

$$H^a_{ss'} = \gamma P^a_{ss'}. \tag{7}$$

Furthermore, we can apply a similar technique in an effort to reduce the storage requirements, characterizing the improvement phase. To that end, the value function may be expressed in the form

$$\begin{aligned} V(s) &= \sum_{s'} P^{\pi(s)}_{ss'} \left[R^{\pi(s)}_{ss'} + \gamma V(s') \right] \\ &= G(s,\pi(s)) + \sum_{s'} H^{\pi(s)}_{ss'} V(s'). \end{aligned} \tag{8}$$

The values for $P^a_{ss'}$ and $R^a_{ss'}$ are provided by the model such that computation of $G(s,a)$ can be performed offline in order to reduce the storage requirements from $O(S^2A)$ to $O(SA)$.

Unfortunately, it is infeasible to similarly reduce the number of unique transitional probabilities from $O(S^2A)$. However, we can decrease the number of bits required to store each unique probability value. This reduction is achieved by mapping probabilities to a discrete set of 7-bit values. In representing the distribution as a discrete set of potential values, some error is introduced. However, this error constitutes less than one-percent per calculation and is distributed uniformly over what is already an inherent approximation. Moreover, we can now offer a greater dynamic range for the reward values, having reduced the storage requirements, that was not possible previously. Applying the aforementioned observations, the total memory requirement, as presented in Table II, has been reduced from 11.7 Mbits to 3.89 Mbits, thus allowing representation of the policy iteration algorithm in many of the larger FPGA devices currently available.

The final observation to be made does not relate to the storage properties, but rather to the reduction in computation requirements resulting when utilizing the offline processing method used to condense the reward function. Recognizing that the product of the discount parameter, γ, and transitional probabilities, $P^a_{ss'}$, can be computed offline, we obtain a smaller number of on-chip multipliers required. This further decreases power, area and latency associated with the hardware realization of the DP policy iteration algorithm.

III. IMPLEMENTATION RESULTS

To demonstrate the effectiveness of the optimizations presented, the generalized policy iteration algorithm was implemented in hardware with the implementation targeting an Altera Stratix II EP2S180 FPGA device. This implementation considered pertains to the car rental problem discussed in [3], utilizing 121 states with a total of 11 potential actions that can be issued at each state. The system architecture utilized S parallel 18-bit floating-point multipliers and a tree structure of 18-bit floating point adders to compute the optimal policy. The complete system, once implemented, achieved a post-fit operating frequency of 103 MHz ($9.7\ ns$). The design required $47,453$ adaptive logic modules (ALMs), or 66% of the ALMs available on the target device. Since the architecture requires a single RAM such that values for each of the 121 states can be accessed in parallel, the device consumes $493(64\%)$ of the M4K RAMs available. Additionally, a single M512 RAM unit was required for the $G(s,a)$ values. The total memory consumed was 1.772 Mbits (i.e. 18% of the RAM).

To establish the performance gain that can be achieved when implementing the hardware-based approach, a software simulation was used. When executed, the software-based policy iteration algorithm converged after an average of 33 evaluation phases and 4 improvement phases with a run time of 9.174 seconds. Similarly, an analysis of the hardware-based approach determined the number of evaluation cycles required for convergence was 163 clock cycles, and the number of improvement iteration cycles to be $1,373$ cycles. Thus, the total time required for the hardware to obtain the optimal policy is $270.1\ \mu s$. Hence, the hardware-based solution yields an improvement of four orders of magnitude in speed when compared to the software-based systems. Although the improvement observed for the particular example chosen does not necessarily reflect on that which will characterize other problems, it provides insight to the potential gain of the framework proposed.

IV. SUMMARY AND FUTURE WORK

In this paper, a novel architecture was introduced for the real-time computation of an optimal control policy employing dynamic programming (DP). It has been demonstrated that the proposed architecture allows a policy to be obtained four orders of magnitude faster than is achieved by a traditional software-based system. The framework is suitable for a broad range of model-based problems requiring real-time decision making. FPGA implementation results were presented and discussed to emphasize the viability and scalability of the proposed architecture. Future work will focus on supporting learning algorithms that employ similar computation techniques, as well as studying further architectural trade-offs to increase operating frequencies in FPGA-based realizations.

REFERENCES

[1] F. Padberg, "On the potential of process simulation in software project schedule optimization," in *29th Annual International Computer Software and Applications Conference*, vol. 2, pp. 127–130, 2005.

[2] S. Kim, M. Lewis, and I. White, C.C., "Optimal vehicle routing with real-time traffic information," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 2, pp. 178–188, 2005.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge MA, MIT Press, 1998.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.