F-033

共有メモリ型並列計算機上での強化学習の並列学習法
Parallel Learning Methods of Reinforcement Learning on Shared Memory Multiprocessors

森 紘一郎† 山名 早人‡
Kouichirou Mori Hayato Yamana

1. まえがき

強化学習は知識がない状態から試行錯誤によって学習を行う。そのため、学習が遅いという欠点があり、いかに学習を高速に行うかが大きな問題点になっている。この問題に対して、従来、価値関数を分割して各プロセッサに割り当て、並列に更新する分散メモリ型並列計算機を対象にした手法 [3]、複数のエージェントが独立した価値関数を並列に更新し、学習の途中で価値関数を結合する手法 [1] が提案されている。しかし、強化学習の性質上、分割された価値関数間で頻繁に経験を交換する必要があり、プロセッサ間通信のオーバーヘッドが大きいことが問題になっていた。そこで、本研究では、共有する1つの価値関数を複数のエージェントが非同期・並列的に更新する共有メモリ型並列計算機を対象にしたオーバーヘッドの小さい手法を提案する。

2. 提案手法

提案手法は [1] の手法と類似しているため両者を比較して述べる。[1] の手法を図 1(a)、提案手法を図 1(b) に示す。

[1] の手法では独立した価値関数を持つ複数のエージェントを用意する。各エージェントは環境との相互作用を並列に行い価値関数を更新する。そして、学習の途中で価値関数を交換して結合することによって学習を高速化する。

提案手法は並列に動作する複数のエージェントを用いる点は [1] と同じである。異なる点は、[1] では各エージェントが価値関数を独立に持っていたのに対し、提案手法ではすべてのエージェントが価値関数を共有している点である。つまり、エージェントは異なるプロセッサ上で並列に動作するが、更新する価値関数は共有メモリ上にあり、すべてのエージェントがアクセスできる。

また、同期処理による遅延を避けるため、すべてのエージェントは共有された価値関数を非同期に更新する。つまり、あるエージェントが価値関数を更新している間でも他のエージェントが価値関数にアクセスできるようにする。

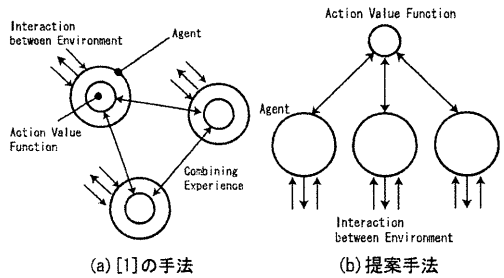

図 1: エージェントと価値関数の位置関係

†‡早稲田大学大学院理工学研究科
‡早稲田大学理工学部

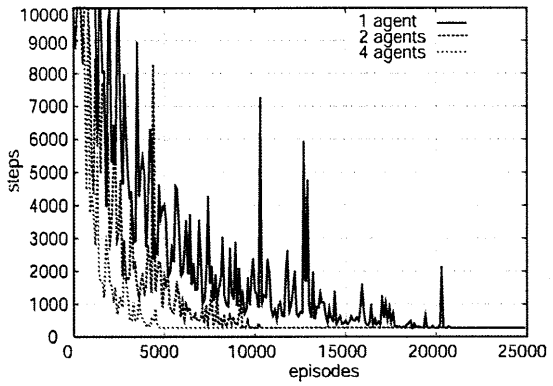

図 2: 迷路タスクの学習曲線

3. 実験

本手法の有効性を確かめるため迷路タスクを用いて実験を行った。迷路タスクは、スタートからゴールへの最短経路を見つけるタスクである。学習には Q-learning、行動選択には ϵ-greedy、価値関数にはテーブル形式を用いた。

プログラムの実装には、C 言語と pthread を用いた。使用したマシンは共有メモリ型並列計算機 Sun Ultra 80（Ultra SPARC II 450MHz 4 台、メモリ 1GB、L1 キャッシュ 16KB、L2 キャッシュ 4MB、SunOS Ver.5.8）である。

4. 結果

4.1 学習曲線

図 2 に迷路タスクにおける学習曲線を示す。横軸はエピソード数（ゴールへたどり着くまでが 1 エピソード）で縦軸はステップ数（価値関数 1 回の更新が 1 ステップ）である。複数のエージェントを用いた場合は、エージェント 1（はじめに起動したエージェント）の学習曲線を記録した。

図 2 から価値関数の更新を行うエージェント数が増えるにつれ、学習の収束が早まることがわかる。

4.2 学習時間

表 1 はプログラムの起動時から図 2 における収束点までの時間を測定した結果である。同期時は共有価値関数の更新時に同期処理§を入れ、非同期時は同期処理を入れていない。

表 1 から価値関数の更新を行うエージェント数が増えるにつれ、学習時間が短縮することがわかる。8 スレッドで学習時間が増加してしまうのは使用したマシンのプロセッサ数が 4 台だからである。また、同期時と非同期時を比べると非同期時の方が速度向上が大きいことがわかる。

§同期処理には pthread_mutex_lock() を用いた。

表 1: 収束までの学習時間

スレッド数	1	2	4	8
同期時（秒）	96.11	71.10	69.83	122.71
非同期時（秒）	83.82	45.02	24.65	24.82

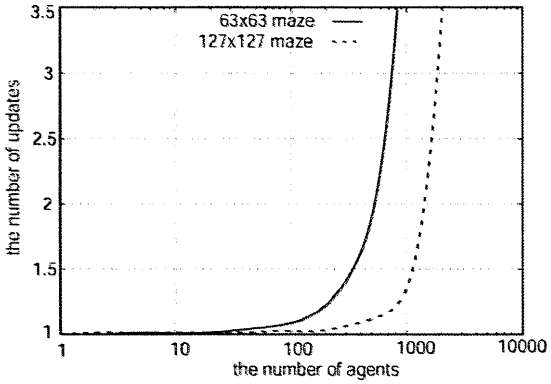

図 3: 収束までの総更新回数

4.3 収束までの総更新回数

図 3 は学習の収束までに価値関数を何回更新する必要があるかを表したグラフである。横軸は更新を行ったエージェント数、縦軸はエージェント数 1 を基準にした総更新回数の倍率である。

図 3 から更新を行うエージェント数の増加にしたがって総更新回数が指数関数的に増加することがわかる。

4.4 各エージェントの価値関数更新分布

図 4 は各エージェントが共有価値関数上のどの部分を更新しているかを表したグラフである。横軸は何回目の価値関数更新かを表すステップ番号、縦軸は更新された状態番号である。

図 4 から各エージェントが更新する場所にはばらつきがあることがわかる。つまり、各ステップでエージェントが更新する価値関数の場所は異なっている確率が高いことがわかる。

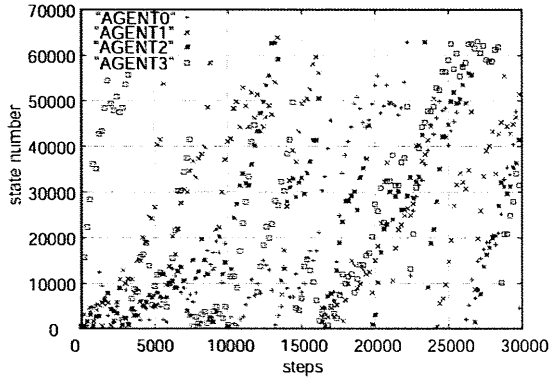

図 4: 各エージェントの価値関数更新分布

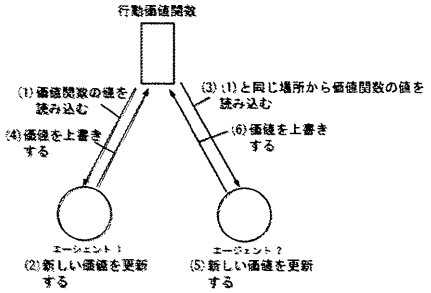

図 5: 非同期更新で無駄が生じる場合

5. 考察

非同期更新は同期処理のオーバーヘッドがないために高速という利点がある（表 1）が、図 5 のような場合に問題が生じる。すなわち、エージェント 1 が価値関数の値を読み込み（1）、更新し（2）、書き戻す（4）前にエージェント 2 が同じ場所から価値関数の値を読み込んで（3）しまう場合である。このとき、エージェント 1 の更新はエージェント 2 の更新に打ち消されて無駄になる。この問題は非同期時に特有であり、同期時には生じない。図 5 の現象が発生していることは図 3 でエージェント数を増やしたとき収束までの総更新回数が増えていることからわかる。エージェント数が増えるにしたがって無駄になる更新が増えるからである。また、図 5 の現象が頻繁に発生すると価値関数の整合性が取れなくなり収束性が失われる（4096 エージェントで収束性が失われた）。提案手法は、本実験環境では 4 エージェント（他の実験環境 [2] では 24 エージェント）までは学習が収束し、かつ学習時間が短縮されることが示されている。限界があるにせよ、適切なエージェント数の範囲（共有メモリ型並列計算機で並列に動作させることのできるエージェント数の範囲）では図 5 の影響は小さいと考えられる。その理由は図 4 のように各エージェントが更新する価値関数の場所が異なっているからである。更新場所が異なれば図 5 の現象は発生しない。

6. まとめ

本論文では、共有する価値関数を複数のエージェントが非同期・並列的に更新し、学習を高速化する手法を提案した。また、各エージェントの価値関数更新部分がばらつくことを示し、適切なスレッド数の範囲では非同期更新が有効であることを実験的に示した。

参考文献

[1] R. M. Kretchmar: "Parallel Reinforcement Learning", Proc. of the 6th World Conf. on Systemics, Cybernetics, and Informatics, Vol.6, pp.114-118, 2002.

[2] 森紘一郎, 山名早人: "強化学習並列化による学習の高速化", 情処研報（ICS）, Vol.2004, No.29, pp.89-94, 2004.

[3] A. M. Printista, M. L. Errecalde and C. I. Montoya: "A Parallel Implementation of Q-Learning Based on Communication with Cache", Journal of Computer Science and Technology, Vol.1, No.6, 2002.