1E2-3

Structured Zobrist Hashによる効率的な並列最良優先探索

Structured Zobrist Hash: An Efficient Work Distribution Method for Parallel Best-First Search

陣内　佑　Alex Fukunaga
Yuu Jinnai

東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, University of Tokyo

Hash Distributed A* (HDA*) is an efficient parallel best first algorithm that asynchronously distributes work among the processes using a global hash function. Zobrist Hash is known to be an efficient hash function for HDA*. We propose a new work distribution method for parallel search, Structured Zobrist Hash. Structured Zobrist Hash is an extension of Zobrist Hash which mitigates the communication overhead by reducing the amount of node sendings without relying on knowledge of hardware-level locality. This paper evaluates Zobrist Hash and Structured Zobrist Hash in shared memory environment with up to 16 cores. Our experimental results show that HDA* with Structured Zobrist Hash outperforms HDA* with Zobrist Hash.

1. はじめに

A*をはじめとする最良優先探索はグラフ探索問題を解く為の手法として広く用いられている [Hart 68]。最良優先探索の解答能力と実行速度を改善する為の方法として並列化が挙げられる。HDA*はシンプルなA*探索の並列手法であり、共有メモリ環境および分散メモリ環境で効率的である [Kishimoto 13]。HDA*はグローバルなハッシュ関数によって仕事を分配する。この為のハッシュ関数としてZobrist Hashが効率的なロードバランスを実現することが知られている [Zobrist 70]。その一方、Zobrist Hashによる仕事の分配はスレッド間での仕事の送受信が多く、通信オーバーヘッドが大きい。

通信オーバーヘッドを解決する為のハッシュ関数としてはAbstractionが提案されている [Burns 10]。しかしながら、Abstractionはロードバランスが悪く、結果として探索オーバーヘッドが大きいという問題点がある。

本論文ではHDA*の仕事分配の為の新しい手法としてStructured Zobrist Hashを提案する。Structured Zobrist Hashは効率的なロードバランスを実現しつつ、スレッド間の通信回数を減らし通信オーバーヘッドを抑える為の手法である。また、初期化以外はZobrist Hashと同様に計算出来るシンプルな手法である。共有メモリ環境にて16コアまで性能比較実験を行い、Structured Zobrist HashによるHDA*がZobrist HashによるHDA*よりもパフォーマンスに優れることを示す。

2. HDA*

Hash Distributed A* (HDA*) は各スレッドにローカルにオープンセットとクローズドセットを持つ。グラフのノードはハッシュ関数によって各スレッドに割り当てられる。各スレッドは自分に割り当てられたノードのみを展開し、他スレッドに割り当てられたノードを生成した場合はそのスレッドにノードを非同期的に送信する。HDA*は同期オーバーヘッドが殆どないので、特に分散メモリ環境で有効な手法である。

HDA*の処理は以下の通りである。HDA*はまず、ルートプロセスで初期状態を展開する。その後、それぞれのスレッド P は最適解を発見するまで以下のループを繰り返す。

連絡先: 陣内佑　東京大学大学院総合文化研究科
5027052513@mail.ecc.u-tokyo.ac.jp

1. P は自分のメッセージキューに新しいノードが入っているかを確認する。もし入っていたら、それぞれのノードを P のクローズドセットと照合し、オープンセットに入れるべきかを確認する。

2. メッセージキューが空なら、P のオープンセットから最も優先度の高いノードを展開し、新しいノードを生成する。生成されたノード s に対してそれぞれハッシュ値 $K(s)$ を計算する。ノード s は $K(s)$ を担当するプロセッサのメッセージキューに送信される。この送信は非同期的で、ロックを必要としない。P は送信先からの返信を待たずに次の計算に移る。

3. Zobrist Hash

ノードを分配する為のハッシュ関数の選択はHDA*の性能に大きな影響を与える。ハッシュ関数に求められる要件は以下である。

1. 全てのスレッドに対してなるべく均等される。
2. 状態空間に対して**不偏に**均等な分配である。
3. ハッシュ値を高速に求めることが出来る。

以上の要件から、**Zobrist Hash** が広く用いられている [Zobrist 70]。ノード s の状態が整数列 $\mathbf{x}$ で表現されるとする。状態の i 番目の整数の取りうる値は l_i 通りあるとする。ノードの状態を $\mathbf{x} = (x_0, x_1, \cdots, x_n)$ と置く。この時 x_i は $0 \leq x_i < l_i$ である。R_i を l_i 個の値を持つテーブルとする。R_i は予めランダムに生成される。ここでZobrist Function $Z(s)$ は以下のように定義される。

$$Z(s) := R_0[x_0] \; xor \; R_1[x_1] \; xor \; \cdots \; xor \; R_n[x_n] \quad (1)$$

R_i はランダムに生成される為、$Z(s)$ は均等な確率で値域の全ての値が出ると期待される。また、状態の全情報を利用して生成される為、局所的な状態の変化に対しても不偏な分配を行うことが出来る。xor によって計算する為、状態の親ノードからの差分のみを計算してハッシュ値を求めることが出来る。加えて、XOR命令は非常に速い計算である。

4. Abstraction

Abstraction は状態空間を複数のブロックに分割する。それぞれのブロックに対して分配されるスレッドをランダムに設定する。ブロック内のノードは全て同じスレッドに分配される為、Abstraction はスレッド間の通信回数が Zobrist Hash と比較して小さくなる。

一方、Abstraction には効率的なロードバランスを実現することが難しい。Zobrist Hash は状態の要素の数 n 個のランダム値によって分配を行うが、Abstraction はただ 1 つのランダム値によって分配を行う。その為、状態空間に対して偏りのあるハッシュを生成してしまうことが多い。

5. 並列化にかかるオーバーヘッド

並列探索は、理想的にはスレッドの数だけ高速化する。しかしながら、一般に並列探索には、並列化に伴うオーバーヘッドが生じる。並列化のオーバーヘッドは以下の 3 つに分類される。

1. 探索オーバーヘッド

   一般に並列探索は逐次 A*よりも多くのノードの展開を必要とする。並列化に伴い余分に展開したノードの割合を**探索オーバーヘッド**と呼ぶ。本論文では、探索オーバーヘッドは以下の式によって推定する。

   $$SO := \frac{\text{並列実行によって展開したノード数}}{\text{逐次実行によって展開したノード数}} - 1 \quad (2)$$

   探索オーバーヘッドはロードバランスが良い程小さいと考えられる。探索オーバーヘッドは理想的には 0 になるが、一般に 0 よりも大きくなる。

2. 同期オーバーヘッド

   同期オーバーヘッドは、他スレッドの処理を待つ為にアイドル状態で待たなければならない場合に生じる。

3. 通信オーバーヘッド

   通信オーバーヘッドはスレッド間の通信にかかる遅延である。スレッド間の通信は仕事の分配や情報の共有の為に行われる。ここで通信にかかる遅延とは、通信そのもののコストだけではなく、通信の為のデータ構造へのアクセスなどにかかるコストも含む。また、共有メモリ環境の場合はキャッシュミスやメインメモリのコンテンションが増加する可能性もある。本論文ではノードの送信率をもって通信オーバーヘッドの大きさを推定する。

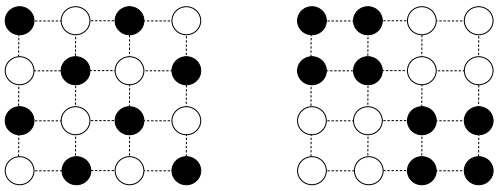
   $$\text{ノードの送信率} := \frac{\text{他スレッドに送信したノード数}}{\text{生成したノード数}} \quad (3)$$

   この値が低いほど通信オーバーヘッドが小さいと考えられる。

これらのオーバーヘッドはトレードオフの関係にある。一般的に、スレッド間の通信回数を増やすことで探索オーバーヘッドを抑えられるが、その場合同期・通信オーバーヘッドが増えることとなる。理論的に良いバランスを求めることは困難である為、多くの場合は実験的にチューニングが行われる。

6. Structured Zobrist Hash を用いた HDA*

Zobrist Hash は不偏的にノードを分配することで効率的なロードバランスを実現する。一方、ノードの送信は通信オーバーヘッドの原因となるので、ノードの送信回数を減らすことで通信オーバーヘッドの緩和が期待される。Structured Zobrist Hash は Zobrist Hash のフレームワーク上に Abstraction のアイディアを応用した、効率的なロードバランスを実現しつつノードの送信回数を減らす為の手法である。



(a) Zobrist Hash　　(b) Structured Zobrist Hash

図 1: 仕事の分配手法の比較

図 1 は Structured Zobrist Hash の分配方法を図示したものである。それぞれの丸は状態空間の中の一つの状態を表す。白丸と黒丸はそれぞれ P_0 と P_1 に割り当てられる。Structured Zobrist Hash は隣り合うノードが同じスレッドに割り振られる確率が高くなるように分配を行う。

Structured Zobrist Hash は初期化以外は Zobrist Hash と同様に計算される。

状態を要素の整数列 $\mathbf{x} = (x_0, x_1, \cdots, x_n)$ と見なす。ただし、Structured Zobrist Hash では状態から要素の省略または簡略化を行う。

1. 要素の省略
   ハッシュ値の計算から適当な要素を除外する。要素の数が多いドメインに有効である場合が多い。

2. 要素の簡略化
   x_i が l_i 通りの値を取るとする。適当な変換によって $m_i(< l_i)$ 通りの値に簡略化し、その値を基にハッシュ値を計算する。Grid Pathfinding などの少数の要素からなるドメインで有効である場合が多い。

これらの Structure の生成はテーブル R_i の初期化時に行い、探索の実行時には Zobrist Hash と同様に計算することが出来る。よって、探索時には Structure の計算にオーバーヘッドはない。

Structured Zobrist Hash は隣接するノードが同じスレッドに割り振られる確率が大きいので、ノードの送信回数が小さくなり、通信オーバーヘッドが緩和される。一方、ロードバランスが Zobrist Hash ほど効率的でない場合がある。しかしながら、このトレードオフは Structure の生成方法によって調整することが出来る。

Structured Zobrist Hash はテーブルの初期化以外、Zobrist Hash と同じ処理を行う、シンプルな手法である。また、Zobrist Hash を用いることの出来るドメインでは必然的に Structure を適応することが出来る為、多くのドメインで有効であると考えられる。

表 1: 実行時間及びスピードアップの比較

ドメイン	分配方法	実行時間 [秒] (スピードアップ [倍]) p = 8	p = 12	p = 16
15 Puzzle	Zobrist	30.9 (3.27)	23.7 (4.28)	22.1 (4.59)
	Abstraction	14.2 (7.13)	16.6 (6.11)	15.5 (6.43)
	Structured	**12.9 (7.88)**	**12.8 (7.96)**	**10.7 (9.43)**
24 Puzzle	Zobrist	191 (4.41)	111 (7.18)	71.5 (10.7)
	Abstraction	**116 (7.10)**	112 (7.12)	109 (9.49)
	Structured	118 (7.12)	**71.8 (10.6)**	**44.5 (17.3)**
Grid Pathfinding	Zobrist	49.8 (0.51)	40.9 (0.64)	50.3 (0.52)
	Structured	**6.04 (4.23)**	**4.29 (5.99)**	**7.55 (3.41)**
MSA	Zobrist	75.4 (3.04)	115 (1.99)	58.3 (3.94)
	Abstraction	71.3 (3.22)	118 (1.94)	60.7 (3.78)
	Structured	**64.7 (3.54)**	**109 (2.10)**	**55.5 (4.13)**

表 2: Structure の大きさ毎の比較 (15 Puzzle, 16 スレッド)

s	実行時間 [秒] (スピードアップ [倍])	探索オーバーヘッド	ロードバランス	ノードの送信率
1 (Zobrist Hash)	22.1 (4.59)	**0.016**	**1.005**	0.943
2	17.2 (5.87)	0.084	1.013	0.642
4	11.6 (8.73)	0.068	1.095	0.342
8	**10.7 (9.43)**	0.195	1.179	**0.184**
Abstraction	15.5 (6.43)	0.252	1.223	0.229

7. 実験結果

共有メモリ環境において Structured Zobrist Hash の性能評価実験を行った。実験マシンは 16 コア Intel Xeon E5-2650 v2 (2.60GHz), 128GB RAM のものを使用した。OS は GNU/Linux Ubuntu 14.04 である。スレッドは pthread ライブラリで実装し、非同期通信は try_lock で実装した。実験は 15 Puzzle, 24 Puzzle, Grid Pathfinding, Multiple Sequence Alignment(MSA) の 4 つのドメインを用いた。実験インスタンスは MSA 以外はランダムに生成したものを、MSA は BAliBASE 3.0 から 6 問を用いた。Structured Zobrist Hash の実装はそれぞれのドメインで複数の実装を行い、最も良い手法を採用した。なお、Grid Pathfinding において最適な Structured Zobrist Hash の手法は Abstraction と同様になる為、Structured Zobrist Hash の結果のみを示す。

表 1 は各ドメインにおける実行時間とスピードアップ（1 コアにおける実行時間 / n コアにおける実行時間）の比較である。どのドメインにおいても Structured Zobrist Hash による HDA*が最も高速である。特に 15 Puzzle や Grid Pathfinding のようなノードの展開が非常に速いドメインだと Structured Zobrist Hash が有効であると考えられる。ノードの展開が速い場合、相対的に通信オーバーヘッドの割合が大きくなるので、それを緩和することが可能な Structure の効果は大きい。

表 2 は 15 Puzzle, 16 スレッドにおける分配手法の比較である。表の s は Structure の大きさである。15 Puzzle では要素の簡略化を用いて Structure を生成し、各要素に対して $\lfloor x_i/s \rfloor$ をハッシュ値を計算に用いた。Structure を大きくするほどノード送信率は減るが、その一方ロードバランスの効率が落ちる。15 Puzzle 以外のドメインでも同様のトレードオフが確認された。Structured Zobrist Hash は s の値を変えることによりこのトレードオフを調整することが出来る。一方、Abstraction の場合、ノード送信率が小さくなるが、ロードバランスは悪化してしまう。

8. まとめ

本論文では並列探索の為のシンプルで効率的な仕事の分配手法として Structured Zobrist Hash を提案した。共有メモリ環境にて 16 コアまで実験を行い、Structured Zobrist Hash による HDA*が Zobrist Hash または Abstraction による HDA*よりも高速であることを示した。Structured Zobrist Hash は通信オーバーヘッドを緩和する手法であるので、より通信コストの大きい分散メモリ環境ではより有効であると考えられる。分散メモリ環境での実験は今後の研究課題である。また、Structured Zobrist Hash は HDA*だけでなく、他の並列探索手法にも一般的に応用出来ると考えられる。他の並列探索手法での実験評価も今後の課題である。

参考文献

[Burns 10] Burns, E. A. and Lemons, S.: Best-First Heuristic Search for Multicore Machines, *Journal of Artificial Intelligence Research*, Vol. 39, No. 1, pp. 689–743 (2010)

[Hart 68] Hart, P. E. and Nils, J.: Formal Basis for the Heuristic Determination, *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107 (1968)

[Kishimoto 13] Kishimoto, A., Fukunaga, A., and Botea, A.: Evaluation of a simple, scalable, parallel best-first search strategy, *Artificial Intelligence*, Vol. 195, pp. 222–248 (2013)

[Zobrist 70] Zobrist, A. L.: A new hashing method with application for game playing, *reprinted in International Computer Chess Association Journal*, Vol. 13, No. 2, pp. 69–73 (1970)