モンテカルロ木探索の Root 並列化とコンピュータ囲碁での有効性について

副島 佑介[†1] 岸本 章宏[†1,†2] 渡辺 治[†1]

コンピュータ囲碁では，モンテカルロ木探索を利用する方法が最も有効であり，モンテカルロ木探索の並列化は，囲碁プログラムの強さを改善する手法の一つである．

本論文では，合議制と総和制という 2 つの Root 並列化を強い囲碁プログラムの一つである Fuego に実装し，その性能を調査した．64CPU コアを用いた実験では，Chaslot らの先行研究とは異なり，Root 並列化の有効性だけでなく限界も示すことができた．また，合議制は，総和制よりも有効であるという明確な結果は得られなかったが，総和制よりも良い着手を選出する傾向があることが分かった．

Root Parallelization of Monte Carlo Tree Search and Its Effectiveness in Computer Go

Yuusuke Soejima ,[†1] Akihiro Kishimoto [†1,†2]
and Osamu Watanabe[†1]

Since Monte Carlo Tree Search (MCTS) has been most promising in computer Go, parallelizing MCTS has been considered to be a way to improve the strength of computer Go programs.

In this paper, we analyze the performance of two root parallelization methods for determining the next move - majority vote based one and total score based one, both implemented on top of Fuego, which is one of the best Go programs. Unlike Chaslot's previous work, results obtained from experiments with 64 CPU cores clearly demonstrate that root parallelization not only is effective but also has limitations. Additionally, although we have not yet obtained results vividly demonstrating the superiority of majority vote, we show an empirical evidence proving that majority vote based one tends to select better moves than total score based one.

1. はじめに

1.1 研究背景 - モンテカルロ木探索

現在の強い囲碁プログラムは，UCT 探索[10] を代表とするモンテカルロ木探索を利用している．その実力は，９路盤では少なくともアマ高段者クラスである．また，１９路盤でもプロ棋士に置き碁で７子で勝利を収めている．

モンテカルロ木探索は，プレイアウトと呼ばれるモンテカルロ・シミュレーションと木探索より構成される．ある局面 P のプレイアウトでは，P からゲーム終了局面 Q に至るまで，各プレイヤーはランダムに合法手を選択し続け，Q の勝ち負けを P の勝率計算に利用する．プレイアウトを繰り返せば，勝率の高い着手ほど有望な手であることが分かる．しかし，少ないプレイアウト数で計算された着手の勝率は不正確であるので，その着手にもプレイアウトを行う必要がある．

木探索は，内部節点での各着手の勝率と訪問回数よって計算された UCB 値に基づいて，次にプレイアウトを行う節点を選択する．勝率が高い着手，または訪問回数が少なく勝率が不正確な着手の UCB 値は大きくなる傾向がある．木探索では，ルート節点から UCB 値が最大の着手の選択を葉節点 L に至るまで繰り返し，L を展開し，プレイアウトを行う．このプレイアウトの結果に基づき，木探索が選択した経路上にある着手の UCB 値を更新する．モンテカルロ木探索では，次の一手選択の制限時間が来るまで，この過程を繰り返し，訪問回数が最大の手を次に打つ着手として選ぶ．

モンテカルロ木探索の性能向上手法の研究は，コン

†1 東京工業大学 大学院情報理工学研究科 数理・計算科学専攻
Department of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
Email: soejima.y.aa@m.titech.ac.jp, {kishimoto, watanabe}@is.titech.ac.jp
†2 科学技術振興機構さきがけ
PRESTO, Japan Science and Technology Agency

ピュータ囲碁において主要なテーマである．例えば，強いプログラムでは，もっともらしい勝率を計算するために囲碁特有の知識を利用したプレイアウトを行っている[3),8)]．RAVE[7)] では，ある局面のプレイアウト結果を，同様の目的を持ちうる着手に再利用することにより，プレイアウト情報をより多くの節点に伝搬させている．RAVE は，Fuego[4)] などの強いプログラムに利用されている．

1.2 モンテカルロ木探索と並列化

モンテカルロ木探索をさらに改良する方法として，より大きな探索木の構築と，プレイアウト数の増加が挙げられる．探索木を改善すれば，より有望な局面でプレイアウトを行える．また，プレイアウト数の増加によって，局面の優劣の評価がより正確になる．

モンテカルロ木探索を並列化すれば，単位時間あたりのプレイアウト数と探索木の大きさを増やせるので，プログラムの性能向上につながる．さらに，近年は，シングルコア当たりの CPU 速度の向上率は以前よりも低いので，ハードウェアの改善による恩恵を受けるためには，並列化は必要不可欠である．

各プロセス (またはスレッド) が探索木を共有する Tree 並列化[5),6)] は，代表的な並列化法である．この方法では，より大きな木を構築できるが，木の共有の際に生じるオーバーヘッドのため，CPU コア数の増加による探索速度改善の効率が低下するという欠点がある．

一方，Root 並列化[1)] は，プロセス間で独自にモンテカルロ木探索を行うため，CPU コア数の増加に伴い，探索速度も上昇する．しかし，並列化を行っても逐次に比べて各プロセスの探索木が改善されないので，Tree 並列化の方が，より有望な局面でプレイアウトを行っている可能性がある．

代表的な囲碁プログラムで利用されている手法は，Tree 並列化である[4),6)] が，Chaslot らは，Root 並列化は Tree 並列化よりも有効であると報告している[2)]．しかし，文献[2)] では，Root 並列化がスーパーリニアーな効果を得ているだけでなく，Root 並列化の方がなぜ有効であるかを示す具体的な実験データが示されていない．

1.3 本論文の貢献

並列化によりコア数以上の性能が出る場合には，逐次アルゴリズムの性能改良の余地を示唆していることが多い．本論文では，Computer Olympiad 大会で優勝している強い囲碁プログラム Fuego[4)] 上に Root 並列化を実装し，その効果を再調査した．本論文の貢献は以下の通りである．

- コンピュータ将棋で有望な合議制[12)] に基づく Root 並列化を提案し，総和制と性能を比較した．
- 先行研究よりも多くの CPU コア (64 コア) を用いた，9 路盤の囲碁での実験では，両 Root 並列化がある程度有効であることが確認した．
- Fuego による自己対戦では，合議制は総和制よりも有効であったが，MoGo との対戦では，同等の勝率であった．
- 合議制と Tree 並列化の性能を比較した結果，64 コアによる合議制は，6 スレッドによる Tree 並列化に負け越した．
- Root 並列化と Tree 並列化を組み合わせれば，さらに性能が向上する事を確認した．
- 合議制と総和制の棋譜を分析し，着手の性質を調べた．その結果，合議制の方が良い着手を選択する頻度が高いことを確認した．

1.4 本論文の構成

２節では，本研究で利用するプログラムである Fuego について簡単に述べる．３節で，モンテカルロ木探索の並列化に関する関連研究を紹介する．４節では，本研究の Root 並列化について説明する．５節で実験結果を示し，６節でまとめと今後の課題について述べる．

2. Fuego - 対象囲碁プログラム

Fuego[4)] は，アルバータ大学を中心に開発されているオープンソースの囲碁プログラムである．2009 年 5 月に行われた Computer Olympiad では，9 路盤で 1 位，19 路盤で 2 位という成績を残しており，現在最も強いプログラムの一つである[11)]．Fuego では，RAVE を利用した UCT 探索が用いられている．さらに，MoGo と同様に囲碁の知識に基づく様々なパターン[8)] を利用して，プレイアウトの性能を改良している．

Fuego は，スレッドを利用した共有メモリ環境での並列化と MPI を利用した分散メモリ環境での並列化に対応している[⋆1]．Fuego の共有メモリ環境での並列化法については，次節で取り扱う．

3. 関連研究 - モンテカルロ木探索の並列化

3.1 モンテカルロ木探索とオーバーヘッド

一般に，木探索の並列化には，逐次探索では起こらない，次の３つのオーバーヘッドが生じる．

探索オーバーヘッドは，逐次探索が行わなかった部分を並列探索が行う無駄な探索である．並列化によって，無駄な探索がかえって減る場合も存在するが，この場合には，通常は逐次探索自体に改良の余地があることを示している．同期オーバーヘッドは，あるプロセス (またはスレッド) の計算が終了するのを他のプロセスが待つ際に生じるアイドル時間である．例えば，同期オーバーヘッドには，共有メモリ環境で情報を共有する際に生じるロックによる排他制御がある．通信オーバーヘッドは，他のプロセスと情報の送受信を行う際に生じる通信遅延である．

これらの３つのオーバーヘッドは，並列探索の低下

⋆1 分散メモリ環境での並列化法については，ソースが公開されていないので，技術的な詳細については不明である．

の原因になるだけでなく，相互依存の関係にある．つまり，並列探索がよい性能を発揮するためには，これらのオーバーヘッドの組み合わせがなるべく少ない手法の開発が必要である．

モンテカルロ木探索の並列化では，構築する探索木をより大きくすることと，プレイアウト数を増やすことが重要である．しかし，並列化によるオーバーヘッドの問題だけでなく，並列モンテカルロ木探索では，探索木の大きさの増加とプレイアウト数の増加のどちらに，より重点を置くべきかが現状では明確ではない．例えば，一つのプロセスが探索木の管理を行い，残りのプロセスが葉節点でプレイアウトを行う Leaf 並列化[1),2)] は，プレイアウト数を増加できるが，同期オーバーヘッドの増大のため，性能が悪いことが報告されている．加藤らは，Leaf 並列化の同期オーバーヘッドを減らす方法を提案している[9)] が，探索木の情報が更新されない間は，同一の葉節点のプレイアウトを何度も行ってしまう可能性がある．

3.2 Tree 並列化

Tree 並列化は，モンテカルロ木探索が構築する探索木を各プロセス (またはスレッド) で共有し，木の展開，プレイアウト，および UCB 値の更新を行う．Tree 並列化は，より大きな木を作ることができるので，最も自然な並列化法の一つである．特に，CPU コア間でメモリを共有できる環境では，Tree 並列化は多くの強い囲碁プログラムで利用されている[5),6)]．

Tree 並列化では，プレイアウトは各スレッドが独立に行えるが，木の展開や UCB 値の更新の際に，他のスレッドと共有する木にアクセスする必要がある．木の安全な共有には，ロックなどの排他制御によるスレッド間での同期が必要である．このため，例えば，複数のスレッドが同一の節点の情報を更新しようとする場合には，排他制御のために同期オーバーヘッドが生じ，性能低下につながる．木を共有するスレッド数が多くなれば，同期オーバーヘッドの増加はより深刻な問題になると考えられる．

Enzenberger らは，共有メモリ環境での Tree 並列化において，C++の volatile 機能の特徴と IA-32 や Intel-64 CPU のハードウェアの性質を利用し，探索木アクセス時の排他制御処理を取り除いた[5)]．この手法では，同期オーバーヘッドを取り除けるが，UCB 値の更新の際に，更新すべき情報が時折失われることが理論上は生じる．しかし，この情報の喪失は，現実にはほとんど起こらず，囲碁プログラムの強さにはほとんど影響しないと結論付けている．

ネットワークで繋がれた PC など，メモリが分散する環境では，同期オーバーヘッドに加え，PC 間の通信も必要であるので，木共有のオーバーヘッドはさらに深刻である．Gelly らは，各プロセッサが構築した探索木の重要な部分のみを定期的にブロードキャストし，共有することによって，同期・通信オーバーヘッドを減らしている[6)]．

3.3 Root 並列化

Tree 並列化よりも実装が簡単な Root 並列化[1)] は，探索の際に，プロセス間で情報を共有しない．つまり，各プロセスは，異なる乱数シードを用いて，独自に探索木を作成する．Root 並列化が次の一手を選択する際には，一つのプロセスが，各プロセス上にあるルート局面における各着手の情報を集計し，その集計情報に基づいて最も有効な手を選択する．Root 並列化の集計方法は，訪問回数の多い着手を重要であるとみなす総和制である．総和制では，各着手の訪問回数を合計し，その合計数が最大の着手を選択する．

Root 並列化では，探索木を共有しないので，大きな探索木を構築できるとは限らない．つまり，複数プロセスが同じ探索木を構築してしまうことによる探索オーバーヘッドの増大が問題として考えられる．しかし，乱数を利用して行うプレイアウトの結果が異なる場合が存在するので，各プロセスが互いに異なる性質の木を構築できる．さらに，ルート局面を各プロセスにブロードキャストする探索開始時と着手の情報を集計する探索終了時にのみ通信・同期オーバーヘッドが生じるので，探索木を共有する方法に比べて，並列探索によるオーバーヘッドがはるかに少ない．このため，Root 並列化では，他の手法よりも単位時間あたりに行えるプレイアウト数が大きい．

Chaslot らの 16CPU コアの共有メモリ環境での実験では，Root 並列化は，Tree 並列化よりも有効であり，さらにコア数以上の性能を出すこともあった[2)]．この理由を，文献[2)] は，Root 並列化の作る木が Tree 並列化に比べて浅く，探索が局所解から脱出しやすいためと推測しているが，具体的な根拠は示されていない．

4. 本研究で実装した Root 並列化

並列木探索が，CPU コア数以上の性能を発揮する場合には，逐次探索に改良の余地がある場合が多い．このため，Chaslot らの Root 並列化の結果[2)] でも同様のことが生じている可能性がある．例えば，Chaslot らのプログラムには，逐次モンテカルロ木探索の性能を大幅に改良する RAVE が実装されていない．また，Chaslot らの実験では，16CPU コアまでしか利用しておらず，より多くの CPU を利用したときにも Root 並列化が台数効果を引き続き得られるかどうかは，研究課題の一つである．

本節で述べる Root 並列化は，合計 64 コアの CPU から構成される分散メモリ環境で，Fuego を利用して実装している．Fuego は現状で最も強い囲碁プログラムの一つであるだけでなく，様々なオプションがあり，RAVE なしのバージョンを実行することもできる．このため，Chaslot らの実験結果を再評価するのに理想的なプログラムである．

本研究では，先行研究で提案された総和制だけでなく，合議制に基づく Root 並列化も実装した．

総和制による Root 並列化の欠点は，平均化した着手を選択することである．例えば，プロセス P_1, P_2, P_3 があるとき，P_1 と P_2 の着手 m_1 に対する評価は高いが P_3 では低い場合と，全プロセスが m_2 をそれなりに評価する場合には，総和制は m_2 を選ぶことがある．合議制による Root 並列化は，コンピュータ将棋で有効である単純多数決法[12] に基づく．この手法では，従来の Root 並列化と同様に，各プロセスは，互いに異なる乱数シードで，独立にモンテカルロ木探索を行う．次の一手の決定時には，各プロセスは自身が最善と判断する着手を 1 つだけ候補手として選択する．最も多くのプロセスに選ばれた手が次にプレイする手である．我々の実装では，各プロセスが選択する候補手は，そのプロセス内の探索木で，訪問回数が最大の手とした．また，多数決により選択できる着手が 2 つ以上ある場合には，全プロセスの訪問回数の合計が最大である着手を選択した．

合議制では，総和制よりも，各プロセスが最善手と判断する手を，次善手以降であると判断した手よりもはるかに高く評価する．このため，先程の例では m_1 を選択する．その一方で，意見が複数に分かれた場合には，少数にとっては最善手であるが，他のプロセスは悪手とみなす着手を選択してしまうことがある．

5. 実験結果

5.1 実験条件

4 節で述べた手法を Fuego version 0.3.2 に実装し，ネットワークで接続された 8 ノードから成る PC クラスター上で実験した．各ノードには，8 コアの CPU(Xeon E5410 2.33GHz × 2) と 16GB の共有メモリがある．Root 並列化は，MPICH2[⋆1] と C++を用いて実装した．性能比較は，9 路盤 (コミ六目半，中国ルール) での 1 手 10 秒で，200 局対戦を行った．Fuego には序盤用の定石ファイルがあるが，実験では，定石ファイルを利用せず，各対戦では異なる乱数の初期値を設定した．このようにして，200 局の対戦棋譜の中に似た対局が現れないようにした．

5.2 Root 並列化の有効性

表 1 逐次版 Fuego に対する Root 並列化の勝率 (%)

CPU コア数	4	8	16	32	64
総和制	57.0	59.5	59.0	60.5	65.5
合議制	57.0	69.0	65.0	68.5	72.0

表 1 に，様々な CPU コア数による両 Root 並列化法の勝率を示す．合議制と総和制の対戦相手には，逐次版の Fuego を利用し，双方の Fuego のパラメータは，最適なものに設定した．また，Root 並列化版の Fuego では，元々実装されているロックなしの Tree 並列化を利用せず，利用コア数と同数のプロセスを立ち上げ，総和または合議による着手選択を行った．

総和制と合議制の両手法とも，コア数の増加に伴い勝率が向上している．さらに，合議制では，従来の Root 並列化法である総和制よりもこの実験では高い勝率が得られた．実際に，64 コア合議制と 64 コア総和制で 200 局対局を行った結果，合議制の総和制に対する勝率は 56.5 %と勝ち越した．

囲碁プログラムには，各プログラムの実装方法により，選択する着手に個性があるのが通常である．このため，自己対戦による実験だけでなく，他のプログラムとの対戦を行うことは重要である．

Root 並列化の対戦相手として逐次版の MoGo[⋆2] を利用し，合議制と総和制の勝率を調べた結果を表 2 に示す．この実験でも，総和制と合議制の両方で，CPU コア数を増加させれば，勝率が上昇することが確認できた．しかし，Fuego との自己対戦で得られた合議制の優位性は，MoGo との対戦結果では，特に見受けられなかった．さらに，総和制では，CPU コア数を 32 から 64 に増加させても勝率にほとんど変化しなかった．このため，64 コア以上利用する Root 並列化の勝率は，あまり上昇しない可能性もある．より多くのコア数を用いた Root 並列効果の限界の調査は，今後の課題の一つである．

表 2 MoGo に対する Root 並列化の勝率 (%)

CPU コア数	4	8	16	32	64
総和制	54.0	55.5	61.0	65.5	65.0
合議制	53.0	54.5	64.5	63.5	67.5

Chaslot らの実験では，RAVE が実装されておらず，これが Root 並列化のコア数の増加による勝率上昇率に影響を与えている可能性がある．表 3 に，逐次および Root 並列化された Fuego から RAVE オプションを外したときの自己対戦の勝率を示す．この表より，RAVE なしの場合にも自己対戦でも，合議制の方が総和制よりも勝率が高い傾向があることが分かる．

表 3 Fuego(RAVE なし) に対する Root 並列化 (RAVE なし) の勝率 (%)

CPU コア数	4	8	16	32	64
総和制	58.5	61.0	58.5	69.0	62.5
合議制	62.0	65.0	59.0	70.0	72.5

5.3 ロックなし Tree 並列化との比較

Root 並列化の探索木非共有によるデメリットを調べるために，Fuego に元から実装されているロックなし Tree 並列化と 64 コアによる合議制との対戦を行

⋆1 http://www.mcs.anl.gov/research/projects/mpich2/

⋆2 http://www.lri.fr/~gelly/MoGo.htm

表 4　64 コア合議制とロックなし Tree 並列化の勝率比較 (%)

Tree 並列化のコア数	1	4	6	8
合議制	72.0	52.0	45.0	49.5

い，性能比較を行った．合議制では，コア間での探索木の共有を行わず，64 プロセスが独自に探索を行った．一方，Tree 並列化では，Virtual Loss の利用など，最適な実装を利用した．

表 4 は Tree 並列化での利用コア数を 1 から 8 に変化させた際の勝率である．Chaslot の実験とは異なり，64 コア Root 並列化の性能は，6 コアの Tree 並列化に負け越した．Enzenberger らの Fuego での実験では，ロックなし Tree 並列化の効果は，7 コアまであり，8 コアでは，7 コアよりも弱くなっていた[5]．同様に，我々の結果も 8 コアに対する合議制の勝率は上昇している[⋆1]．

ロックなし Tree 並列化では，UCB 値更新時に更新すべき情報が消えることがある．文献[5] では，この情報喪失の頻度は少ないとあるが，多数コアを利用する際には，情報喪失の頻度が上昇する可能性も考えられる．コア数の増加に伴う情報喪失の頻度と性能への影響の調査は，今後の課題の一つである．

5.4　Root 並列化と Tree 並列化の融合

表 5　合議制・総和制の勝率比較 (%)(8 × 8Root 並列化)

8 × 8 Root 並列化	合議制	総和制
F_{tree}	67.0	68.5
MoGo	77.0	79.0

Root 並列化と Tree 並列化の欠点を補う自然な方法として，ノード内部のコアではロックなし Tree 並列化を行い，ノード間では Root 並列化を行う方法が考えられる．8 コアの Tree 並列化を行う Fuego を F_{tree} として，F_{tree} を 8 つ利用した並列化 (以後，8 × 8 Root 並列化と呼ぶ) の効果を調べた．表 5 は，8 × 8 Root 並列化の合議制と総和制について対戦相手をそれぞれ F_{tree} と MoGo にしたときの勝率である．

F_{tree} に対し，表 4 では逐次版 Fuego を 64 個利用する合議制の勝率は 49.5 %であったが，64 という同じ CPU コア数を使用する合議制の 8 × 8 Root 並列化では 67.0 % (表 5 を参照) まで上昇した．同様に，表 2 では，64 コアでの合議制の MoGo への勝率が 67.5 %あったのに対し，8 × 8 Root 並列化では 77%であった．これらの事実より，8 × 8 Root 並列化は Fuego の強さ向上に貢献していると言える．

表 5 において，8 × 8 Root 並列化では，総和制の勝率の方が合議制よりも約 2 %高く，合議制との差はほとんど見られなかった．8 × 8 Root 並列化の合議制と総和制を実際に 200 局対戦させた結果，合議制の総和制に対する勝率は 47.0 %であり，ほぼ互角であった．

5.5　総和制と合議制の着手の性質

総和制と合議制による囲碁プログラムの振る舞いを調べるため，各手法が選択する着手の性質を調べた．

Root 並列化では，各プロセスから返ってきた各手の訪問回数を集計し，次の一手を選択するだけであるので，合議制と総和制の選択する着手が一致するかどうかは簡単に確認できる．この性質を利用して，逐次 Fuego を 64 個立ち上げた合議制 Root 並列化 (64 コア Root 並列化) と逐次 Fuego との 200 局の対戦棋譜に出現する Root 並列化の手番の局面で，合議制と総和制の着手の一致率を調べた．同様に，8 × 8 Root 並列化では，合議制 8 × 8 Root 並列化と F_{tree} との 200 局の対戦棋譜を利用した．その結果，64 コア Root 並列化と 8 × 8 Root 並列化の両方では，合議制と総和制の次の一手の一致率は，およそ 95 %であった．着手が一致しなかった残りの局面 (64 コア並列化 160 局面，8 × 8 並列化 187 局面) では，合議制・総和制とも各プロセスの最善手が一意に定まらず，少なくとも二極分化した．

着手が一致しなかった局面について，合議制と総和制の選出する着手の性質をより詳しく調べるため，長い時間制限 (30 秒) で着手選択を行う F_{tree} を利用した．この実験では，F_{tree} を一番強いプログラムと仮定し，F_{tree} にこれらの局面を探索させた．そして，F_{tree} が出力した探索木の情報と，合議制・総和制の選出した着手と各プロセスの集計情報を比較し，詳しく分析した．

64 コア並列化では，着手が不一致であった 160 局面の中で，合議制が F_{tree} の着手と一致した局面数は 62 であるのに対し，総和制の F_{tree} との一致局面数は 36 であった．また，不一致局面 P での着手 m に対し，F_{tree} の探索終了後の各 m の訪問回数の大きな順番で着手の良さを順位付けた場合，合議制の平均順位は 1.54 であるのに対し，総和制の平均順位は 1.75 であった．

一方，8 × 8 並列化での着手の一致しなかった 187 局面では，合議制と F_{tree} の一致局面数は 74 であり，総和制は 37 局面であった．また合議制の着手の平均順位は 1.52，総和制は 1.82 となった．

これらより，合議制の方が総和制よりも良い手を選んでいる傾向があると推測できる．しかし，MoGo との対戦や 8 × 8 並列化の自己対戦の結果では，必ずしも合議制の方が総和制よりも効果があるとも言えず，さらに分析をする必要がある．

最後に 64 コア Root 並列化において，合議制と総和制の着手が不一致であった局面で，合議制が F_{tree} の着手と一致した局面と総和制が F_{tree} と一致した局

⋆1 Martin Müller 氏によれば，16CPU コアから構成される共有メモリマシンでの Fuego の自己対戦では，引き続き勝率の向上が見受けられている．このため，8 コアにおける合議制の勝率上昇の理由は，単に自己対戦数が少ないための可能性もある．

表 6　図 1 での総和制と合議制の各着手の情報 (上位 5 位まで)

総和制	候補手	平均訪問回数
1 位	A4	29,416
2 位	H5	25,984
3 位	A7	25,917
4 位	H4	20,460
5 位	H6	2,618

合議制	候補手	平均訪問回数	投票数
1 位	A7	25,917	28
2 位	H4	20,460	14
3 位	H5	25,984	13
4 位	A4	29,416	9
5 位	H6	2,618	0

表 7　図 2 での総和制と合議制の各着手の情報 (上位 5 位まで)

総和制	候補手	平均訪問回数
1 位	E8	24,637
2 位	B2	23,276
3 位	D1	20,910
4 位	E9	10,894
5 位	B1	7,535

合議制	候補手	平均訪問回数	投票数
1 位	D1	20,910	22
2 位	B2	23,276	21
3 位	E8	24,637	16
4 位	E9	10,894	5
5 位	B1	7,535	0

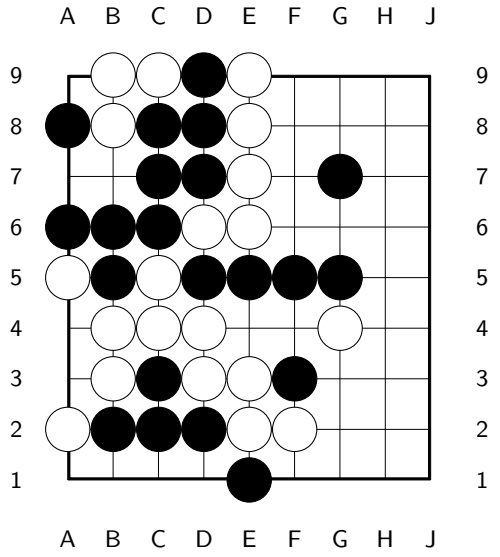

図 1　合議制が F_{tree} の着手と一致した局面 (白番)

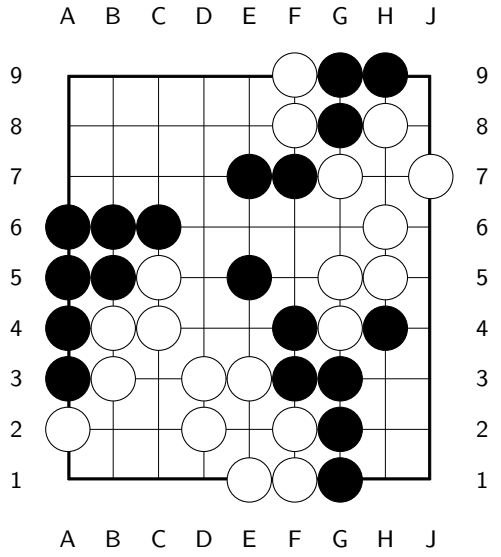

図 2　総和制が F_{tree} の着手と一致した局面 (黒番)

面をそれぞれ図 1 と図 2 に示す.

図 1 では，合議制が A7，総和制が A4 を選出した．この場合，A7 が好手であり，黒の大石を殺している．A4 につなげば，黒に A7 につながれてしまい，攻め合いに負けるので，上辺の白 5 子が取られてしまう．

図 2 における，総和制と合議制での上位 5 位までの着手に関する情報を表 6 に示す．A4 は各 CPU コアで平均的に高い訪問回数を得たために，総和制では選択されているのに対し，合議制では 64 コア中 9 コアしか A4 を選択していない．一方，総和制は A7 を 3 番目に有力な手であると考えているのに対し，合議制では 28 コアが A7 を最も有望であると考えている．この例は，実際には悪手であるが，平均的に訪問回数が高いために，最善であるとみなしてしまう総和制の弊害を合議制が防いでいると言える．

図 2 では，合議制が D1，総和制が E8 を選出した．この場合，D1 は単に当たりをするだけの手であるうえに，コウ材を一つ消費しているので損な手である．一方，E8 は，ヨセとして大きい場所である．

図 2 の各着手の集計情報は，表 7 である．総和制では E8 の訪問回数が最も多いのに対し，合議制が E8 を選んだ CPU コアの数は 16 と D1 の 22 よりも小さい．

6. まとめと今後の課題

本論文では，強い囲碁プログラム Fuego を利用して，モンテカルロ木探索の Root 並列化の効果について再調査した．さらに，Root 並列化に合議制を取り入れ，その効果を調べた．

Root 並列化は，総和制・合議制共に有効であるが，Chaslot らの結果とは異なり，64 コアのリソースを利用しても，最大でも Tree 並列化の 8 スレッド分にしか相当しないことを確認した．合議制と総和制の比較では，自己対戦では合議制の方が総和制よりも勝率が高かったが，MoGo との対戦では勝率差が見られなかった．

さらに，本論文では，Root 並列化と Tree 並列化を組み合わせ，64 コアによる実験で合議制と総和制の有効性を示した．自己対戦および MoGo との対戦では，合議制と総和制の性能は，ほぼ同等であった．

合議制と総和制のの着手の性質について調べた所，Tree 並列化と Root 並列化を組み合わせないときと組み合わせたときとの両方で，合議制の方が総和制よりも，より良いと思われる手を選択していることが分かった．

今後の課題として，合議制・総和制の着手の傾向と勝率の関係の更なる調査が挙げられる．本論文の実験結果では，合議制の方が総和制よりも良い手を打つ可能性が高いことを示唆しているが，現状では合議制の方が効果的である明確な結果を得られていない．囲碁では，良い着手を選ぶ傾向があったとしても，一手の悪手が負けにつながる場合も多く，このような悪手を合議制が選んでしまうことも可能性としては考えられる．もし，合議制がどのような場合に悪手に投票する傾向があるかを知ることができれば，Root 並列化の更なる性能向上につながると考えられる．例えば，各着手に対して手の信頼度に基づく重み付けを行い，悪手である可能性の高い手に対しては小さな重みにすることが挙げられる．

また，19 路盤での合議制・総和制の性能比較を行いたい．64 コア Root 並列化の合議制と総和制において，1 局当たりに選択する着手が異なる場合は，9 路盤では，平均で 1 手程度である．一方，19 路盤での我々の初期実験では，1 局当たり 10 手程度，合議と総和が異なる着手を選んでいる．そのため，合議制と総和制の選択する着手による振る舞いの違いは 19 路盤での方がより明確になると考えられる．

参 考 文 献

1) T.Cazenave and N.Jouandeau. On the parallelization of UCT. In *Proceedings of the Computer Games Workshop*, pp. 93–101, 2007.
2) G. M. J-B. Chaslot, M. H.M Winands, and H.J.van den Herik. Parallel Monte-Carlo tree search. In *Proceedings of the 6th International Conference on Computer and Games*, Vol. 5131 of *Lecture Notes in Computer Science*, pp. 60–71, 2008.
3) R. Coulom. Computing Elo ratings of move patterns in the game of Go. *ICGA Journal*, Vol.30, No.4, pp. 198–208, 2007.
4) M. Enzenberger and M. Müller. Fuego - an open-source framework for board games and Go engine based on Monte-Carlo tree search. TR 09-08, University of Alberta, 2009.
5) M. Enzenberger and M. Müller. A lock-free multithreaded Monte-Carlo tree search algorithm. In *Advances in Computer Games 12*, 2009.
6) S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian. The parallelization of Monte-Carlo planning. In *Proceedings of the 5th International Conference on Informatics in Control, Automation, and Robotics*, pp. 198–203, 2008.
7) S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 273–280, 2009.
8) S. Gelly, Y. Wang, Remi Munos, and O. Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, 2006.
9) H.Kato and I.Takeuchi. Parallel Monte-Carlo tree search with simulation servers. In *Proceedings of the 13th Game Programming Workshop*, pp. 31–38, 2008.
10) L. Kocsis and Cs. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, Vol. 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer, 2006.
11) M.Müller. Fuego at the Computer Olympiad in Pamplona 2009: a tournament report. TR 09-09, University of Alberta, 2009.
12) 小幡拓弥, 塙雅織, 伊藤毅志. 思考ゲームによる合議アルゴリズム - 単純多数決の有効性について. 第 22 回ゲーム情報学研究会, 2009.