AND/OR木における証明数・反証数を用いた階層的挟み撃ち探索

鷹野 芙美代[†] 関根 敦史[††] 佐田 宏史[†]
前川 仁孝[†] 六沢 一昭[†]

本論文では階層的挟み撃ち探索を用いたAND/OR木の並列探索手法について提案する．証明数の小さい節点は解である可能性が高い．そのため，証明数を用いるAND/OR木の並列探索の多くは証明数の小さい節点から探索する．しかし証明数の大きい節点が解である場合は，解を見つけるのに多くの時間を必要とする．そこで，証明数の小さい節点と証明数の大きい節点の探索を並列に行い，さらに証明数の小さい節点に多くのプロセッサを割り当てる並列探索手法を提案する．このように複数プロセッサで探索することにより，従来の探索法では解を見つけるまでに時間がかかる証明数の大きい節点が解である場合にも，探索時間を短縮することができる．提案手法を共有メモリ型並列計算機に実装して評価した．評価の結果，逐次探索では時間のかかる問題に対し，スーパリニアスピードアップが確認された．

Hierarchical Pincers Attack Search Using Proof Numbers and Disproof Numbers for AND/OR Tree

Fumiyo Takano,[†] Atsushi Sekine,[††] Hiroshi Sata,[†]
Yoshitaka Maekawa[†] and Kazuaki Rokusawa[†]

This paper proposes a parallel AND/OR tree search algorithm using the hierarchical pincers attack search. Parallel processing algorithms of AND/OR tree search are proposed. Many of them search a tree from a node having a small proof number. When a solution is a node having a large proof number, they need a long computation time. Therefore, the proposed algorithm uses the hierarchical pincers attack search. It is able to search from a node having a small proof number and a node having a large proof number simultaneously. Additionally, many processors search some nodes having a small proof number, and a few processors search some nodes having a large proof number. The proposed algorithm is implemented on a shared memory multiprocessor and evaluated. The result shows super linear speedup by parallel processing when sequential search needs a long computation time.

1. はじめに

コンピュータ上でチェス，将棋，囲碁などのゲームを行うことは，ルールが明確であること，勝ち負けがはっきりつくことなどを理由に人工知能のよい例としてさかんに研究されている．これらはAND/OR木を探索することとして一般化できる．AND/OR木の代表的な探索方法には深さ優先探索や最良優先探索などがある．最良優先探索は，何らかの評価値を用いて有効だと思われる節点のみを探索する，人間の思考に近い探索法である．1994年にはAllisらによって節点の証明・反証をするためにかかるコストを表した証明数・反証数という概念が生み出され，証明数・反証数を用いた最良優先探索である証明数探索が提案された[1]．しかし最良優先探索では多くの記憶領域を必要とするので，探索木が大きくなると解が求まらない．そこで証明数や反証数を閾値として反復深化を行うことにより，少ない記憶領域しか必要としない深さ優先探索で，最良優先探索のような振舞いをする探索法が提案された．例として，証明数のみを用いるものにC*[2]やPN*[3]，証明数と反証数の2つを用いるものにPDS[4]，df-pn[5),6)]，df-pn+[7]などがある．しかしこれらのような探索法を用いても，探索深さが深くなるにつれ探索節点数が指数関数的に増加するため探索時間は膨大になる．

探索の高速化のためにAND/OR木の並列探索手法

† 千葉工業大学情報科学部情報工学科
Department of Computer Science, Faculty of Information and Computer Science, Chiba Institute of Technology
†† 株式会社両毛システムズ
Ryomo Systems Co., Ltd.

もさかんに研究されている．代表的なものにYBWC[8]，APHID[9] などがある．YBWC は PV（Principal Variation：最善応手）を最初に探索する PVSplit[10] を拡張したものである．本手法は節点の動的並べ替えをし PV を探索してから他の節点の探索をするので多くの枝刈りができる．しかしすべてのプロセッサの探索の終了を待つ必要があり，同期のオーバヘッドが存在するため並列処理による速度向上が得難い．同期オーバヘッドをなくし通信を削減する手法に APHID や ParaPDS[11),12)] などが提案されている．APHID はマスタ・スレーブモデルであり，マスタが $\alpha\beta$ 法で一定の深さまで探索し，残りの深さをスレーブが反復深化法により探索するので同期点はまったくない．またマスタがスレーブの負荷分散を行うので効率よく探索できる．ParaPDS は APHID を AND/OR 木探索に応用し，スレーブの探索に PDS を用いた並列探索手法である．これらの並列探索アルゴリズムは逐次探索とほぼ同じ探索木を複数のプロセッサで分割して探索するものである．

最良優先探索など，ヒューリスティックな評価関数を用いる探索において，評価値の悪い節点が解であった場合，解を見つけるまでに時間がかかる．これは APHID や ParaPDS などの並列探索でも同様である．そこで本論文では，AND/OR 木の並列探索アルゴリズム AOHPAS（And/Or tree Hierarchical Pincers Attack Search）を提案する．このアルゴリズムでは詰将棋の逐次探索において単純な実装では最も性能がよいとされている，証明数・反証数を閾値とした反復深化法である PDS[13] を，階層的挟み撃ち探索を用いて並列化したものである．証明数の小さい節点からの探索と大きい節点からの探索を並列に行うことで証明数の大きい節点が解であった場合にも短時間で解を得ることが期待できる．

以下では，まず 2 章で証明数と反証数について述べる．3 章で証明数と反証数を用いた逐次アルゴリズムである PDS について説明し，4 章で並列探索，5 章で階層的挟み撃ち探索，6 章で提案手法である AOHPAS について述べる．7 章で提案手法の評価を行い，8 章でまとめる．

2. 証明数・反証数

証明数と反証数は証明数探索の評価値として提案された[1]．証明数や反証数は，信頼性の低い静的評価に時間をかけるよりも，より多くの局面を探索して動的な評価に置き換えるという考えに基づいたものである．証明数はある節点を true と証明するために true であることを示さなければならない最小節点数であり，反証数は false と証明するために false であることを示さなければならない最小節点数である．節点 n の証明数を $\mathrm{pn}(n)$，反証数を $\mathrm{dn}(n)$ とすると $\mathrm{pn}(n)$ と $\mathrm{dn}(n)$ は以下のように計算される．

(1) 節点 n が先端節点
  (a) 最終的な評価値が true
    $\mathrm{pn}(n)=0$
    $\mathrm{dn}(n)=\infty$
  (b) 最終的な評価が false
    $\mathrm{pn}(n)=\infty$
    $\mathrm{dn}(n)=0$
  (c) 最終的な評価が不明
    $\mathrm{pn}(n)=1$
    $\mathrm{dn}(n)=1$

(2) 節点 n が内部節点
  (a) n が OR 節点
    $\mathrm{pn}(n)$=子節点の pn の最小値
    $\mathrm{dn}(n)$=子節点の dn の和
  (b) n が AND 節点
    $\mathrm{pn}(n)$=子節点の pn の和
    $\mathrm{dn}(n)$=子節点の dn の最小値

詰将棋においては，証明数は，各節点が詰みとなるために詰みを示さなければならない先端節点の最小数，つまり先手に対する後手の玉の逃げ方の総数である．また，反証数は，各節点が不詰となるために不詰となることを示さなければならない先端節点の最小数，つまり後手に対する先手の王手のかけ方の総数である．証明数が小さいほどその局面は詰みやすいといえるので先手（OR 節点）は証明数最小の手を選ぶことが解である可能性が高く，また反証数が大きいほどその局面は詰みにくいといえるので後手（AND 節点）は反証数が最小となる手を選ぶことが解である可能性が高い．

3. PDS

証明数探索をはじめとする最良優先探索では，1 度探索した局面をすべて記憶しておく必要があるため，非常に多くの記憶領域が必要となる．そこで，最良優先探索である証明数探索の振舞いを，記憶領域をあまり必要としない深さ優先探索として実現する PDS が提案された[4]．PDS は証明数と反証数を閾値とし徐々に増やして探索する，反復深化を行う深さ優先探索である．反復深化法では同じ節点を何度も探索するため，各節点の情報をトランスポジションテーブルに保存し，節点の再展開を防ぐ．PDS では，まず根節点におけ

る証明数と反証数の閾値を 1 として，その範囲内の証明数と反証数の節点を深さ優先探索する．ここで根節点の証明数あるいは反証数が 0 となれば探索は終了する．証明数が 0 であれば証明できた（詰み）こととなり反証数が 0 であれば反証された（不詰）こととなる．根節点の証明数も反証数も 0 にならなければ，証明もしくは反証しやすそうな方の閾値を増やし再び探索する．OR 節点では証明数が小さい節点から展開し，AND 節点では反証数が小さい節点から展開する．また，ある節点が証明や反証され，その節点の先祖節点の証明数や反証数が変化した場合，未探索の節点に 1 よりも大きな閾値が与えられることがある．未探索の節点を 2 以上の閾値で探索すると，証明数探索と同じ振舞いをせず，局所的にただの深さ優先探索となり，効率が極端に悪くなる．そこで，初めて探索される節点では閾値を 1 とし，根節点と同様に反復深化する多重反復深化[2] を行う．

4. 並列探索アルゴリズム

PDS は詰将棋の探索に非常に有効である[13]．これは証明数（反証数）が小さい節点は，詰む（不詰）と証明するためのコストが低い節点であり，コストが低い節点から探索すれば解が早く求まる可能性が高いためである．しかし，証明数が大きい節点が解である場合は，正解節点を探索するまでに時間がかかる．

図 1 の探索木を例に説明する．節点 d，e，f は詰んだ局面であり正解は c 以下の節点である．根節点を展開した時点での証明数は節点 a は 1，節点 b は 2，節点 c は 3 となり，a，b，c の順に小さい．よって，PDS では a，b，c の順に探索する．このような場合，証明数の大きい節点 c も並列に探索することで，探索時間が短縮する．「続詰むや詰まざるや」[14] の最初の 50 問において，根節点の証明数最小の子節点が正解手順と何問一致するかを調べた．根節点の子節点において，証明数の最も小さい節点が正解節点であったのは，50 問から不詰問題などを除いた 49 問中 15 問，約 30%であった．また，正解節点の証明数が，根節点の子節点中，何番目に小さいかを節点数 2 以上の問題に対し調べた．根節点の子節点数を 10 とし，相対的な順番を式 (1) のように計算する．正解節点番号とは証明数が最小の節点を 0 番目とした正解節点の証明数順の番号である．

$$\text{相対正解節点番号} = \frac{\text{正解節点番号}}{\text{節点数} - 1} \times 9 \qquad (1)$$

式 (1) により，節点数を 10 としたの時の相対正解節点番号の平均を計算した結果，約 2.2 番目であった．

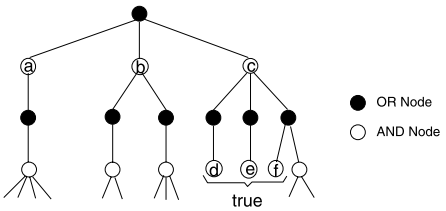

図 1 探索木の例
Fig. 1 An example of search tree.

証明数，反証数とも探索過程で順次再計算されるが，この結果は，証明数の小さい節点が必ずしも正解節点ではない，つまりヒューリスティックが有効でない可能性があることを示す．しかし，ヒューリスティックをまったく使わず全幅探索すると多くの探索時間を必要とする．また，ヒューリスティックを用いる従来の並列探索アルゴリズムを用いると，逐次探索とほぼ同じ探索順で並列に探索するため，証明数の大きい節点が解である場合，同様に多くの時間がかかる．そこで証明数の小さい節点からの探索と証明数が大きい節点からの探索を組み合わせ，並列に探索することにより，証明数の大きい節点が解である場合でも早く解くことができると考えられる．ここで，証明数の小さい節点が解であることが多いため，証明数の小さい節点は多くのプロセッサで探索し，証明数の大きい節点は少数のプロセッサで探索する．

そこで本論文では階層的挟み撃ち探索を AND/OR 木探索に応用した AOHPAS を提案する．AOHPAS では証明数の小さい節点から探索するプロセッサと証明数の大きい節点から探索するプロセッサの双方を用いる．また根節点から証明数の小さい節点へ深さごとに階層的にプロセッサを割り当てることで，証明数の小さい節点に多くのプロセッサを割り当てる．このような並列探索をすることにより，証明数の大きい節点が解である場合には探索時間を大幅に短縮することができ，証明数の小さい節点が解である場合には PDS による逐次探索と同程度の時間で解を発見することが期待できる．

5. 階層的挟み撃ち探索

OR 木並列探索手法の 1 つに，実行時間最小マルチプロセッサスケジューリング問題に対する最適化アルゴリズムである PDF/IHS 法などで用いられる階層的挟み撃ち探索がある[15),16)]．

階層的挟み撃ち探索では，ヒューリスティック的によい解を探索木の左に集め，**図 2** のように 1 つのプロセッサは普通の深さ優先探索と同様に，探索木の左

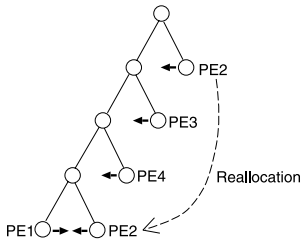

図 2 階層的挟み撃ち探索
Fig. 2 Hierarchical pincers attack search.

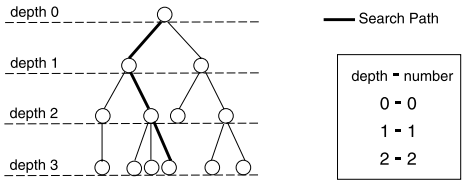

図 3 探索経路
Fig. 3 A searching path.

から右へ深さ優先探索を行う．その他のプロセッサは，左から探索するプロセッサの探索パス上の節点を根節点として右から左へと深さ優先探索を行う．このように探索することでヒューリスティックによい解を中心に，ヒューリスティックが外れた場合の節点も同時に探索するので解の早期発見の可能性が上がる．

また，左から探索するプロセッサ以外の各プロセッサの探索が終了したかどうかの判定は，左から探索するプロセッサの探索経路と自分の探索経路が重なるか，または左から探索するプロセッサの担当節点以外のすべての子節点を探索したかどうかをチェックするだけでよい．探索経路は**図 3** のように節点の深さと枝の左からの番号で表される．[深さ-番号] とすると図 3 の探索経路は [0-0][1-1][2-2] となる．左から探索するプロセッサの探索経路は，すべてのプロセッサが参照できる．PDF/IHS 法では，左から探索するプロセッサの探索経路上の深さの浅い節点から割当て待ち状態のプロセッサに割り当てられ，各プロセッサは割り当てられた節点を根節点とする部分木を探索する．左から探索するプロセッサの探索経路上では，深さが決まれば節点は一意に決まるので，深さを指定するだけでプロセッサに節点を割り当てることができる．深さを与えられたプロセッサは，左から探索するプロセッサの探索経路をその深さまでトレースし，割り当てられた節点を決定し，探索する．また割り当てられた領域の探索が終わったプロセッサは，図 2 のように他のプロセッサに探索されていない節点へと再割当てされる．

階層的挟み撃ち探索では，逐次探索では探索しない

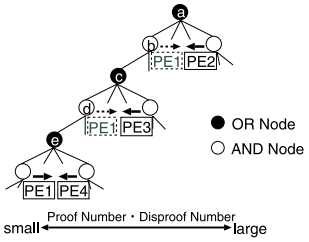

図 4 AOHPAS の探索例
Fig. 4 An example of search of AOHPAS.

可能性もある節点も探索するため，総探索節点数は増えることもある．しかし右から探索するプロセッサが担当する領域に解が存在した場合，探索節点数は減る．図 1 において，逐次探索では節点 a，b，c の順に探索するが，階層的挟み撃ち探索では節点 a と節点 c の探索を同時に行うため節点 b の探索は行われない可能性もある．また，それぞれのプロセッサは自分の担当する範囲の探索が終われば，すぐ次の節点に割り当てられるため，同期オーバヘッドはない．

6. AND/OR 木階層的挟み撃ち探索

実行時間最小マルチプロセッサスケジューリング問題を解くための PDF/IHS 法による探索木は OR 木となるが，詰将棋問題を解くための探索木は AND/OR 木である．Prolog などでは AND/OR 木は OR 木に変換し，階層的挟み撃ち探索を行うことが可能である[16]．AND/OR 木を OR 木に変換するには，AND 接続を逐次的に縦につなげばよい．しかし，将棋などの探索木では，一般的に節点は局面，枝は親局面から子局面への指し手とするため，AND/OR 木を OR 木に変換する際に，AND 接続を逐次的に縦につなげると辻褄があわない．また，詰将棋では正解手数を与えないため，探索木の深さは探索する前には分からない．よって詰将棋の AND/OR 木を OR 木に変換し，階層的挟み撃ち探索を用いて解くことは困難である．そこで本論文で提案する AOHPAS は，詰将棋問題を解くことを考慮し，証明数を用いて AND/OR 木を OR 木に変換せずに AND/OR 木のまま並列探索する．

AOHPAS では，**図 4** のように，ある節点において 1 プロセッサは証明数の小さい子節点から探索し，もう 1 プロセッサは証明数の大きい子節点から探索する．つまり，ある節点の子節点を 2 プロセッサで挟み撃つように探索する．また，図 4 の節点 c，e のように，証明数最小の AND 節点の子節点のうちの反証数最小の OR 節点と根節点 a の子節点をそれぞれ 2 プロセッサで探索する．これにより証明数・反証数の小さい節点を多くのプロセッサで探索することができる．図 4 の

例では根節点の最も証明数の小さい子節点 b 以下の節点は，PE1，PE3，PE4 の 3 プロセッサが探索し，その他の子節点以下は PE2 のみが探索する．

左から探索するプロセッサ以外は，割り当てられた節点において PDS を行う．また，左から探索するプロセッサの閾値をすべてのプロセッサで共通の閾値として用いる．これをグローバルな閾値と呼ぶ．通常の PDS では根節点が詰みとなるか不詰となるか判明するまで探索を続けるが，AOHPAS の右から探索するプロセッサは，割り当てられた節点のグローバルな閾値に至ると探索を打ち切る．割り当てられた領域の探索が終わったプロセッサは，左から探索するプロセッサが展開した節点の中で他のプロセッサに割り当てられていない節点があれば再割当てされ，なければ割当て待ち状態となる．

逐次探索でもすぐに解を発見できるような証明数の小さい節点が解である探索木を，AOHPAS で探索してもあまり効果はない．しかし，逐次探索で解を発見するのに時間がかかる，証明数の大きい節点が解である探索木は AOHPAS で並列探索することにより，証明数の大きい節点から探索するプロセッサが解を早く見つけることができるので，探索時間を大幅に短縮することができると考えられる．

以下にプロセッサを割り当てる節点，再割当てをする節点，閾値，各プロセッサの探索について述べる．

6.1 プロセッサ割当て節点

AND/OR 木の場合，プロセッサを割り当てる節点には AND 節点のみ，OR 節点のみ，すべての節点という選択肢がある．PDS では，後手番局面である AND 節点では反証数の小さな節点から展開する．反証数が小さいということは詰みにくいということであり，AND 節点では 1 つでも子節点が詰まないと分かれば詰まない．また AND 節点には，無駄合の処理や DAG による同一局面出現のため証明数が 2 重カウントされることなどがあるため，逐次展開が有効である[17]．よって AND 節点は逐次探索と順序を変えず探索し，各 OR 節点にプロセッサを割り当て，その子節点を図 4 のように並列に探索する．

6.2 再割当て節点の選択

割り当てられた領域の探索が終わったプロセッサはすぐに次の節点に割り当てられるが，節点の再割当ての方法は 2 種類考えられる．1 つは PDF/IHS 法と同様に，左から探索するプロセッサの探索経路上で他のプロセッサに割り当てられていない節点のうち最も浅い節点（**図 5** の節点 a）から割り当てる方法，もう 1 つは逆に，左から探索するプロセッサの探索経路上で最も深い節点（図 5 の節点 b）を割り当てる方法である．後者の方法で再割当てすると，深い節点は閾値が小さいので，すぐに証明数（反証数）が閾値を超え，探索が打ち切られる可能性が高い．このため割り当てられたプロセッサの探索が無駄になることが多くなる．よって再割当ては他のプロセッサが探索していない最も浅い節点から行う．図 5 に再割当ての様子を示す．PE2 が最初に割り当てられた深さ 0 の節点のすべての子節点の探索を終了したとき，他のプロセッサに探索されていない節点は a と b であるが，そのうちの深さが浅い節点 a が PE2 に割り当てられる．

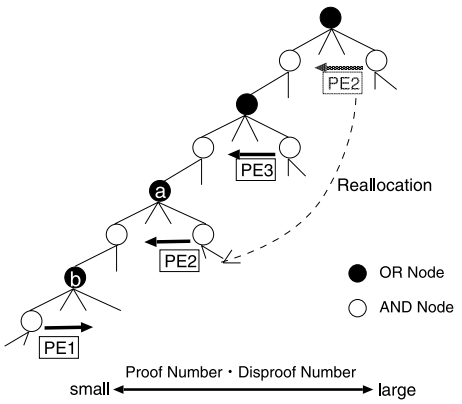

図 5 再割当て
Fig. 5 Node reallocation.

6.3 グローバルな閾値

実行時間最小マルチプロセッサスケジューリング問題では，探索木の深さは上限がある．しかし詰将棋の探索では末端局面は詰むか王手がかからないかしかなく，それらの局面に至る手数は決まっていない．このため，実際の詰手数よりもはるかに深い枝がある可能性がある．たとえば，数手詰みの問題であっても数百手，数千手まで探索木が広がることも多い．また，通常の PDS は根節点が証明か反証されるまで探索するが，それでは割り当てられた節点 1 つを探索するために多くの時間がかかってしまい，他の節点を探索できないため，複数のプロセッサを効果的に使用できない．そのため，各プロセッサの探索で行う PDS で用いる閾値とは別に，探索木のすべてで共通するグローバルな閾値を設け，根節点において反復深化を行う．グローバルな閾値には，左から探索するプロセッサの閾値を用いる．左から探索するプロセッサはグローバルな閾値を用いて探索を行い，各節点ごとのグローバルな閾値を設定する．右から探索するプロセッサは割り当てられた節点のグローバルな閾値を閾値の上限として PDS を行う．また，グローバルな閾値を用いる

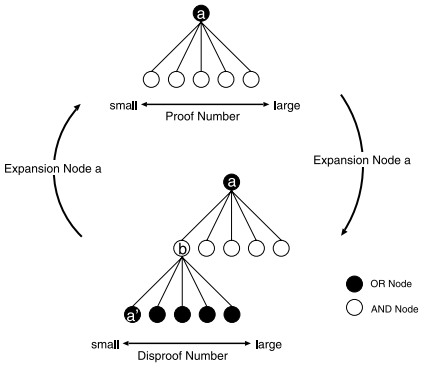

図 6 左から探索するプロセッサの節点展開
Fig. 6 Node expansion by a processor searching from left.

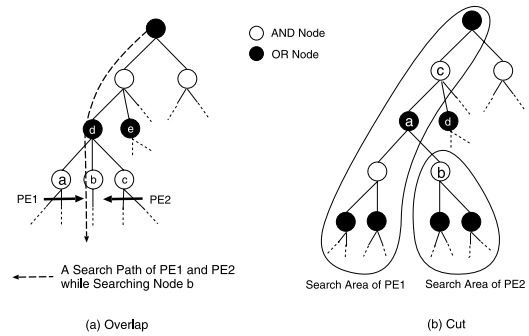

図 7 探索の重複とカット
Fig. 7 Overlap and cut of search.

ことにより，プロセッサ数が変化しても探索範囲は変化しない．

グローバルな閾値の増加幅が小さいと 1 回の探索範囲が狭くなり，プロセッサを割り当てるオーバヘッドが大きくなる．そのため，複数のプロセッサが効果的に使用できず，並列効果があまり得られない．しかし増加幅が大きいと，探索領域が突然大きく広がるために探索の効率が落ち，探索時間が遅くなる可能性がある．

6.4 各プロセッサの探索法

階層的挟み撃ち探索ではヒューリスティックによい解を探索木の左に集める．AOHPAS ではヒューリスティックに証明数を用いる．

左から探索するプロセッサの探索の様子を**図 6** に示す．まず OR 節点 a を展開し，その子節点を証明数順に並べ替える．その中で 1 番証明数の小さい AND 節点 b を展開する．その子節点の中で最も反証数の小さい OR 節点を a' とし，節点 a と同様に展開して子節点を並べ替える．これをグローバルな閾値を閾値として繰り返し，グローバルな閾値を超えたら未探索の節点までバックトラックして，展開と子節点の並べ替えを繰り返す．また，PDS では多重反復深化を行うが，左から探索するプロセッサが多重反復深化を行うと，通常の根節点における反復深化のみよりも，同じ節点を何度も探索して中止し，閾値を増やして探索するということの繰返しが多くなるので，そのような節点に右から探索するプロセッサを割り当てるオーバヘッドが大きくなる．そのため，左から探索するプロセッサは多重反復深化を行わない．

右から探索するプロセッサは，左から探索するプロセッサが展開した OR 節点に割り当てる．それぞれのプロセッサは割り当てられた節点の子節点を根節点として，閾値が 1 から割り当てられた節点におけるグローバルな閾値までの範囲で PDS を行う．右から探索するプロセッサは，割り当てられた領域で，左からの探索と重複しない限り探索を続ける．

探索の重複は，**図 7** (a) の，左から探索するプロセッサである PE1 と，節点 d に割り当てられ右から探索するプロセッサである PE2 の，割り当てられた節点よりも深い節点 b までの探索経路が同じであることから判断できる．また，あるプロセッサに割り当てられた節点を左から探索するプロセッサが詰みと証明することにより，それ以上その節点を探索する必要がなくなることも，同様に検出できる．図 7 (a) の例では PE1 が節点 a を詰みと証明したら節点 d は OR 節点なので節点 d も詰みとなり PE1 は節点 e の探索に移る．PE2 の探索経路は割り当てられた深さまで PE1 と同じはずであるが，割り当てられた深さより浅い探索経路が PE1 と異なっているため，PE2 は節点 b，c を探索する必要がないことを知る．このように探索の重複や無駄を検出したプロセッサは，その節点の探索を終了し割当て可能な節点があればすぐに再割当てされ，なければ割当て待ち状態となる．また，左から探索するプロセッサは，探索が重複しているかのチェックは行わない．そのため，左から探索するプロセッサは，右から探索するプロセッサが探索中の節点を重複して探索してしまう可能性がある．しかし，右から探索するプロセッサが探索中の節点を，左から探索するプロセッサが重複して探索しても，右から探索するプロセッサが探索が重複しているかどうかチェックしたときに，左から探索するプロセッサと重複して探索していると判断し，その節点の探索を中止するので，多くの節点を重複して探索することはない．また，右から探索するプロセッサが探索済みの節点は，右から探索するプロセッサが割り当てられた節点の情報の 1 つとして記憶しているため，左から探索するプロセッサがそれらを再度探索してしまうことはない．

表 1 Sun Enterprize 4500
Table 1 Sun Enterprize 4500.

CPU	Ultra SPARC-IIs 400 MHz ×14
メインメモリ	4 GB
キャッシュメモリ	32 KB
外部キャッシュメモリ	8 MB

図 7 (b) において PE2 が節点 b を詰みと証明したら節点 a も詰みとなり，節点 a 以下の PE1 の探索は必要ないので PE1 の探索はカットされる．PE2 は節点 a が詰みとなったことを PE1 に知らせ，他の節点に再割当てされる．探索途中の節点が詰みと知らされた PE1 は節点 c までバックトラックし，節点 c の子節点である節点 d を探索する．

あるグローバルな閾値における各プロセッサの探索がすべて終了，つまり左から探索するプロセッサが探索する節点がなくなり，根節点が詰か不詰か分からなければ，グローバルな証明数か反証数の閾値を増加し探索を繰り返す．

7. 性能評価

AOHPAS を**表 1** に示すような共有メモリ型並列コンピュータ Sun Enterprise 4500 上で詰将棋を解くプログラムに対して実装し，評価を行った．評価の際に用いたトランスポジションテーブルは，すべてのプロセッサで共有し，容量は 50 MB とした．

以下，7.1 節で効果的な並列探索を行うために最適なグローバルな閾値の増加幅を求め，この値を利用して 7.2 節で詰将棋問題における並列効果を評価した結果について述べる．

7.1 グローバルな閾値の増加幅の決定

グローバルな閾値の増加幅を決定するため，1，3，5，10 の 4 通りと変えて実験を行った．4 プロセッサを使用し，江戸時代から昭和までの名作といわれている代表的な詰将棋が掲載されている「続詰むや詰まざるや」[14] の中から 15 手詰めから 27 手詰めまでの問題を選んで，解答時間を計測した結果を**図 8** に示す．詰手数の短いものは比較的グローバルな閾値の増加幅による変化はあまりなく，詰手数が長いものになるほど増加幅による処理時間の差は大きい．27 手詰以外では増加幅 1 と 3 はほぼ同じ，5 では処理時間がかなり短くなり，10 になると 25 手詰はさらに短くなるがその他の処理時間は増す．27 手詰では増加幅 3 が最も処理時間が短く，増加幅 5 では長くなるが増加幅 10 ではさらに急激に遅くなる．問題により最も探索時間の短くなる増加幅は違うが，増加幅が 5 のときは処理時間が比較的短く，かつあまり処理時間は増加しない．

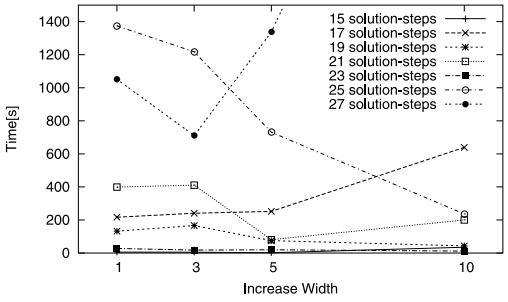

図 8 4 プロセッサでの閾値増加幅による解答時間
Fig. 8 Solving time by increase width of threshold using 4 processors.

このことから，グローバルな閾値の増加幅は 5 と決定した．

7.2 詰将棋問題求解における並列効果

評価には，「続詰むや詰まざるや」の前半 50 問を用いた．このうち 1 問は小問 4 題からなり，また不詰問題などは除くため 49 題を解いた結果を評価する．グローバルな閾値は 7.1 節より 5 ずつ増加させる．またプロセッサ数の増加による高速化率は式 (2) のように計算する．

$$\text{高速化率} = \frac{\text{逐次 PDS の探索時間}}{\text{AOHPAS の探索時間}} \qquad (2)$$

問題を解く制限時間は 3600 秒とし，PDS では解くことができないが，AOHPAS では解くことができた問題は PDS の探索時間を 3600 秒として高速化率を計算した．また，PDS，AOHPAS ともに 3,600 秒以内で解くことができない 22 問は，評価の対象としない．

表 2 に，8 プロセッサを使用したときの高速化率の相乗平均，最大値，最小値を PDS の探索時間ごとに分けて示す．ここで $3600 \leq t$ は PDS では解けなかったものを表す．表 2 から，PDS による探索時間が短いものは高速化率が低いことが分かる．これは，PDS での探索時間が短い問題は，証明数の小さい節点が解であり，左から探索するプロセッサが解を見つける場合が多いからである．また，詰手数の短い問題であることも多く，探索木が浅いので，すべてのプロセッサを使用できないためである．

高速化が得られない例を**図 9** に示す．この問題は 17 手詰であり，PDS での探索時間は 3.84 [s] である．2 プロセッサ使用で逐次探索よりも遅くなるがその後は徐々に速くなる．しかし 8 プロセッサを使用しても高速化率は 1.20 倍程度である．並列探索をした方が逐次探索よりも遅くなるのは，グローバルな閾値を 5 ずつ増やしているために，PDS に比べて無駄な節点を多く探索してしまい，PDS での探索時間よりも 1

表 2 PDS での探索時間別の高速化率
Table 2 Speedups by range of search time using PDS.

Range of Sequential Search Time t[s]	Number of Problems	avg.	max.	min.
$0 < t < 1$	3	0.17	0.37	0.07
$1 \leq t < 10$	4	0.33	1.27	0.03
$10 \leq t < 100$	5	1.59	13.12	0.52
$100 \leq t < 1000$	4	2.40	6.86	0.58
$1000 \leq t < 3600$	1	140.96	140.96	140.96
$3600 \leq t$	10	15.09	214.93	1.57

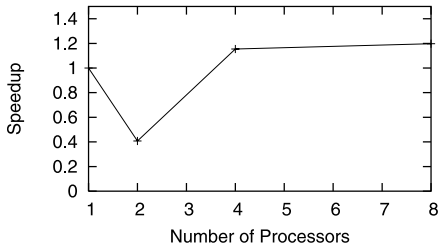

図 9 高速化が得られない例
Fig. 9 An example of no speedup.

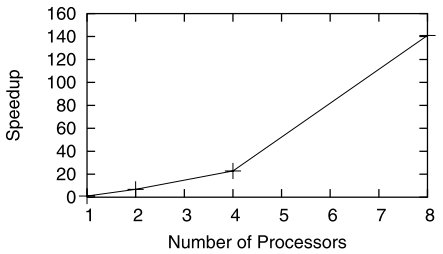

図 11 プロセッサ数以上の高速化が得られる例
Fig. 11 An example of super linear speedup.

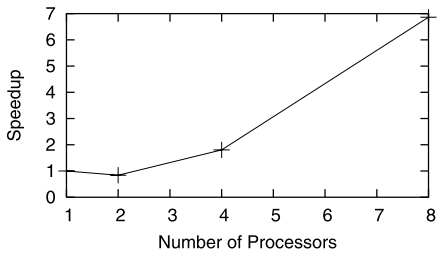

図 10 プロセッサ数以下の高速化が見られる例
Fig. 10 An example of under linear speedup.

プロセッサ使用の AOHPAS の探索時間の方が時間がかかるからである．プロセッサ数を増やすと探索時間は短くなり，高速化率は上がるのは，プロセッサ数が増えても，プロセッサの割当てなど並列化にともなうオーバヘッドが少ないことを示している．

高速化がプロセッサ数倍程度の場合は，右から探索するプロセッサは早く解を見つけていない．しかし探索木を複数のプロセッサで分割しているためにプロセッサ数を増やすことで高速化率が上がる．この例を**図 10** に示す．この場合は PDS での探索時間も中程度であり図 10 の例では 104.20 [s] であった．

PDS での探索時間が長い場合は AOHPAS による効果が大きく，並列に探索することで探索時間は大幅に減少している．**図 11** の例では逐次での探索時間は 1,009.24 [s] と長いが，8 プロセッサで並列に探索すると探索時間は 7.16 [s] と大幅に短縮する．このとき高速化率は 140.96 倍であり，スーパリニアスピードアップが得られている．このような場合は，証明数の大きい節点が解であり，証明数の小さい節点からの探索のみ行うと解を見つけるまでに時間がかかるが，証明数の大きい節点からも探索することで解が早く見つかる．図 11 のように 4 プロセッサから 8 プロセッサで高速化率が急激に上がる場合は，5 番目以降のプロセッサが正解節点を見つけたときである．また，PDS で制限時間以内に解けなかった問題も，並列に探索することで 10 問が解くことが可能となった．表 2 より，PDS で制限時間以内に解くことができなかった問題の高速化率は 1.57 倍から 214.93 倍である．逐次での探索を制限時間（3,600 秒）で打ち切り，探索時間を 3600 秒として計算しているため，実際の高速化率はこれらの数値より高いと考えられる．

また，今回評価したプログラムに使用した PDS は単純な実装であり，無駄合処理など詰将棋を解くのに必要となる技法を実装していない．そのため制限時間以内に解くことができない問題もある．しかし並列化アルゴリズム自体には，それらを実装することによる影響がない．またその他の逐次処理のための高速化手法も，高速化率に影響を与えず利用することができる．

8. おわりに

AND/OR 木探索において証明数・反証数は効果的な評価値であるが，証明数や反証数が高い節点が解であることも少なくない．そこで本論文では証明数の小さい節点から探索するプロセッサと証明数の大きい節

点から探索するプロセッサを用いて，探索時間を短縮する並列探索アルゴリズム AOHPAS を提案した．詰将棋の探索を例に実装した結果，証明数の大きい節点が解であるような，逐次探索では長い探索時間が必要となる多くの問題で，探索時間が大幅に短縮されることが確認できた．

今後は，左から探索するプロセッサの探索法を改良することでさらなる並列効果を引き出し，かつ 1 プロセッサで探索したときにも無駄な探索をせず，逐次探索よりも処理が遅くなる減速異常が起こらないようにする必要がある．また，多くの問題を解くことができるよう逐次，並列に関わらない高速化をすることで，長手数問題に対しての並列効果を計測する必要もある．さらに，本論文では AOHPAS を共有メモリ型並列計算機上に実装したが，本並列探索アルゴリズムはトランスポジションテーブルへのアクセスに関わる通信などを削減することで，分散メモリ型計算機上で実装可能であると考えられる．

参 考 文 献

1) Allis, L.V., van der Meulen, M. and van den Herik, H.J.: Proof-Number Search, *Artificial Intelligence*, Vol.66, pp.91–124 (1994).
2) 脊尾昌宏：C*アルゴリズムによる AND/OR 木の探索および詰将棋プログラムへの応用，情報処理学会人工知能研究報告，No.99, pp.103–110 (1995).
3) Seo, M., Iida, H. and Uiterwijk, J.W.: The PN*-search algorithm: Application to tsumeshogi, *Artificial Intelligence*, Vol.129, pp.253–277 (2001).
4) Nagai, A.: A new AND/OR tree search algorithm using proof number and disproof number, *Complex Games Lab Workshop*, pp.40–45 (1998).
5) Nagai, A. and Imai, H.: Proof for the Equivalence between Some Best-First Algorithms and Depth-First Algorithms for AND/OR Trees, *IEICE Trans.*, No.10, pp.1645–1653 (2002).
6) 長井　歩，今井　浩：df-pn アルゴリズムの詰将棋を解くプログラムへの応用，情報処理学会論文誌，Vol.42, No.6, pp.1769–1777 (2002).
7) Nagai, A. and Imai, H.: Application of df-pn+ to Othello Endgames, *Proc. Game Programming Workshop '99*, pp.16–23 (1999).
8) Feldmann, R.: Spielbaumsuche auf massiv parallen Systemen (Game trees on massively parallel systems), Ph.D. Thesis, University of Paderborn (1993).
9) Brockington, M. and Schaeffer, J.: APHID GAME-TREE SEARCH, *Advances in Computer Chess*, Vol.8, pp.69–91 (1997).
10) Marsland, T.A. and Popowich, F.: Parallel Game-Tree Search, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.7, No.4, pp.442–452 (1985).
11) Kishimoto, A. and Kotani, Y.: Parallel AND/OR tree search based on proof and disproof numbers, *5th Game Programming Workshop*, Vol.99, No.14, pp.24–30 (1999).
12) 岸本章宏，小谷善行：証明数・反証数を用いた AND/OR 木探索アルゴリズムの分散メモリマシンにおける効果的な並列化法について，情報処理学会ゲーム情報学研究報告，No.2, pp.1–8 (2000).
13) 作田　誠，飯田弘之：高性能な AND/OR 木探索アルゴリズムの詰め将棋問題による比較，静岡大学情報学研究，Vol.5, pp.15–22 (1999).
14) 門脇芳雄：続詰むや詰まざるや，平凡社 (1978).
15) 笠原博徳，伊藤　敦，田中久充，伊藤啓介：実行時間最小マルチプロセッサスケジューリング問題に対する並列最適化アルゴリズム，電子情報通信学会論文誌 D-I，No.11, pp.755–764 (1991).
16) 甲斐宗徳，小林和男，笠原博徳：階層型挟み打ち探索による PROLOG OR 並列処理手法，情報処理学会論文誌，Vol.29, No.7, pp.647–655 (1988).
17) 伊藤琢巳，河野泰人，野下浩平：非常に手数の長い詰将棋問題を解くアルゴリズムについて，情報処理学会論文誌，Vol.36, No.12, pp.2793–2799 (1995).

(平成 16 年 1 月 31 日受付)
(平成 16 年 5 月 29 日採録)

鷹野芙美代（学生会員）

1981 年生．2004 年千葉工業大学工学部情報工学科卒業．同年同大学大学院情報科学研究科情報科学専攻博士前期課程入学．主として，並列探索に関する研究に従事．

関根　敦史

1978 年生．2001 年千葉工業大学工学部情報工学科卒業．2003 年同大学大学院工学研究科情報工学専攻博士前期課程修了．同年株式会社両毛システムズに入社．

佐田 宏史

1980 年生．2002 年千葉工業大学工学部情報工学科卒業．2004 年同大学大学院工学研究科情報工学専攻博士前期課程修了．同年同大学大学院情報科学研究科情報科学専攻博士後期課程入学．主として，画像処理とその高速処理に関する研究に従事．電子情報通信学会，画像電子学会各会員．

前川 仁孝（正会員）

1967 年生．1990 年早稲田大学理工学部電気工学科卒業．1992 年同大学大学院理工学研究科電気工学専攻修士課程修了．1993 年日本学術振興会特別研究員．1994 年早稲田大学理工学部助手．1998 年千葉工業大学情報工学科講師．2002 年同大学助教授，現在に至る．博士（工学）．主として，各種アプリケーションの並列処理に関する研究に従事．

六沢 一昭（正会員）

1958 年生．1981 年早稲田大学理工学部電子通信学科卒業．1983 年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年沖電気工業（株）入社．1986 年〜1990 年および 1993 年〜1995 年（財）新世代コンピュータ技術開発機構（ICOT）に出向．1998 年（財）日本情報処理開発協会先端情報技術研究所客員研究員．1999 年千葉工業大学情報工学科助教授．2002 年同大学教授．現在に至る．博士（工学）．主として，論理プログラミング，並列記号処理言語の処理系実装とプログラミング環境の研究に従事．電子情報通信学会，日本ソフトウェア科学会，人工知能学会各会員．