AND節点の並列探索を加えたAND/OR木階層的挟み撃ち探索

鷹野 芙美代[†] 佐田 宏史[†] 前川 仁孝[†]
六沢 一昭[†] 宮崎 収兄[†]

本論文では，AND/OR木の並列探索手法であるAND/OR木階層的挟み撃ち探索において，プロセッサを割り当てる節点を増やすことで，さらに多くのプロセッサを有効活用する並列探索手法を提案する．AND/OR木階層的挟み撃ち探索は，評価の高いOR節点に多くのプロセッサを割り当て，評価の低い節点も並列に探索する．これにより，逐次探索では解の発見に時間のかかる評価の低い節点が解である場合，探索時間を大きく短縮することができる．AND/OR木階層的挟み撃ち探索におけるOR節点のみの並列探索は，並列探索可能な節点が少ないとすべてのプロセッサを使用できない場合もある．そこで，AND/OR木階層的挟み撃ち探索で使用されないプロセッサを用いてAND節点も並列に探索することで，より多くのプロセッサを有効利用する．これにより，AND/OR木階層的挟み撃ち探索で多くのプロセッサを利用することができない問題に対し，より高速に解が求まる．最後に，提案手法を共有メモリ型並列計算機上で評価した結果，AND/OR木階層的挟み撃ち探索よりも最高16.8倍，相乗平均1.5倍高速化されることが確認できた．

Adding Parallel AND Node Search to AND/OR Tree Hierarchical Pincers Attack Search

Fumiyo Takano,[†] Hiroshi Sata,[†] Yoshitaka Maekawa,[†]
Kazuaki Rokusawa[†] and Nobuyoshi Miyazaki[†]

This paper proposes a parallel search algorithm making more efficient use of processors by increasing parallel searchable nodes in AND/OR tree hierarchical pincers attack search (AOHPAS). AOHPAS searches nodes with high evaluation value and low evaluation value simultaneously. Also, AOHPAS allocates many processors to OR nodes with high evaluation value. Thus, when the node with low evaluation value is a solution and sequential searches need long computation time, it is able to reduce computation time greatly. In parallel search of only OR nodes, some processors are not able to be used because there are a few parallel searchable nodes. Therefore, in the proposed algorithm, processors not used by AOHPAS can be used to search AND nodes in parallel. Consequently, the proposed algorithm is able to make more efficient use of processors and to solve problems faster that AOHPAS is not able to use many processors. Finally, the proposed algorithm is implemented on a shared memory multiprocessor and evaluated. It results up to 16.8 times speedup and 1.5 times speedup on the geometric average compared with AOHPAS.

1. はじめに

AND/OR木の探索には，探索木のすべての節点を探索する深さ優先探索や，ヒューリスティックを用いて有効だと思われる節点のみを探索する最良優先探索など，様々なアルゴリズムが提案されている．最良優先探索は，深さ優先探索よりも探索空間が大きい問題を解くことができる可能性が高いが，探索するために多くの記憶領域が必要となる．

そこで近年，反復深化法を用いた深さ優先探索を行うことで，最良優先探索のような探索順を実現することにより，使用する記憶領域の削減が行われている．またAND/OR木探索では，節点の評価値として節点を証明・反証するためのコストを表した証明数・反証数[1]が有効であるとされている．証明数・反証数が小さい方が節点を証明・反証するコストが小さいため，証明数・反証数の小さい節点から探索すると効果的である．証明数と反証数を用いた深さ優先探索として，証明数と反証数のうちの片方もしくは双方を閾値として反復深化する探索法が提案されている[2)~7)]．その中でも詰将棋の探索では，証明される部分木の探索と反証されるべき部分木の探索を効率良く行う，証明数と

† 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

反証数双方を閾値とした反復深化法 PDS[4] が有効であるとされている[8]．

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は指数関数的に増加し，探索時間は膨大になる．そのため，並列処理による探索時間の短縮手法も多く研究されている[9)~12)]．その多くは，ヒューリスティックに従い，評価の高い節点から並列に探索を行うことにより，多くの問題において平均的に逐次探索よりも高速化を図る．ヒューリスティックな評価値を用いた並列探索でも逐次探索と同様に，評価の高い節点が解である場合には早く解が求まるが，評価の低い節点が解である場合には大きな高速化は望めない．

そこで我々は，逐次探索で解を得るまでに長時間かかる評価の低い節点が解である問題に対して短時間で解を求め，さらに逐次探索で高速に解が求まる評価の高い節点が解である場合でも逐次探索より遅くならないことを目的として，PDS の探索木の OR 節点に複数のプロセッサを階層的に割り当て，評価の高い節点と評価の低い節点を並列に探索する AND/OR 木階層的挟み撃ち探索（AOHPAS）を提案した[13]．AOHPAS は，評価の高い節点を多くのプロセッサで探索しながら評価の低い節点も並列に探索することで，評価の低い節点が解の場合にも早く解を求めることができ，逐次探索に比べて大きく高速化することができる．さらに，1 プロセッサは PDS と同等の探索をするため，PDS の探索よりも大きく遅くなることはない．AOHPAS は評価の低い節点が解の場合に大きく高速化することを目的としているため OR 節点のみを並列探索するが，評価の高い節点が解の場合には，すべての子節点を証明する必要のある AND 節点も並列に探索することで，より高速化できると考えられる．

そこで本論文では，OR 節点だけでなく AND 節点にも階層的にプロセッサを割り当て並列に探索することで，AOHPAS よりもさらに高速化する AAA-Search（AOHPAS Added And node Search）を提案する．

AOHPAS は 1 プロセッサが PDS で探索し，その探索経路上の OR 節点にその他のプロセッサを 1 つずつ割り当てることで並列探索する．このため，プロセッサ数が多く，プロセッサが割り当てる節点が探索木に少ないと，すべてのプロセッサを使用できない．そこで AAA-Search では，AOHPAS で使用されないプロセッサを AND 節点にも割り当てて並列探索する．これにより，多くのプロセッサを効果的に使用することができ，より大きな高速化を得ることができる．AAA-Search は OR 節点の探索も AOHPAS と同様に行うため，AOHPAS よりも大きく遅くなることはない．また，AAA-Search は AOHPAS ですべてのプロセッサを使用できない，深さのあまり深くない探索木などで特に有効であると考えられる．深さの深い探索木においては，AOHPAS の OR 節点を並列に探索する効果は大きく，AND 節点を並列に探索することによる高速化が大きくなくても，AAA-Search は逐次探索に対して AOHPAS 以上であり，非常に大きくな高速化が得られると考えられる．

以降の章では，まず 2 章で AAA-Search において評価値として用いる証明数と反証数について，また，それを用いた逐次探索法である PDS について説明する．3 章では PDS を用いて OR 節点のみを階層的に挟み打ち探索する AOHPAS について述べ，4 章で AOHPAS に加えて AND 節点も並列に探索する提案手法，AAA-Search について述べる．5 章で AND/OR 木探索の例として詰将棋の求解を用いて提案手法の評価を行い，6 章でまとめる．

2. PDS

AND/OR 木はすべての子節点が true とならなければ true とならない AND 節点と，子節点のうちどれか 1 つが true となれば true となる OR 節点からなる．AND/OR 木の探索法の 1 つに，証明数探索[1] がある．証明数探索は，節点を証明もしくは反証するのにかかるコストを表す証明数・反証数を評価値として用いる最良優先探索である．証明数はある節点を true とするために true であることを示さなければならない最小節点数を表し，反証数は false とするために false であることを示さなければならない最小節点数を表す．

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
    $\mathrm{pn}(n)=0 \quad \mathrm{dn}(n)=\infty$
  (b) 最終的な評価が false
    $\mathrm{pn}(n)=\infty \quad \mathrm{dn}(n)=0$
  (c) 最終的な評価が不明
    $\mathrm{pn}(n)=1 \quad \mathrm{dn}(n)=1$
(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

証明数探索のような最良優先探索では，探索したすべての節点を記憶しておく必要があるため，多くの記憶領域が必要となる．そこで，証明数探索とほぼ同じ振舞いを，反復深化を行う深さ優先探索として実現する PDS がある．PDS は証明数と反証数を閾値とし，閾値以内の証明数と反証数を持つ節点を探索する．証明数と反証数の閾値をそれぞれ 1 から開始し，徐々に増やしながら反復深化する．根節点の証明数（反証数）が 0 となれば証明（反証）されたこととなり探索を終了する．根節点の証明数も反証数も 0 にならなければ，証明数または反証数の閾値を増やし再探索する．PDS では，OR 節点は証明数が小さい子節点から展開し，AND 節点は反証数が小さい子節点から展開する．また，反復深化を行うことにより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．ある節点が証明もしくは反証され，その節点の先祖節点の証明数や反証数が変化した場合には，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，局所的にただの深さ優先探索となるので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つねに閾値を 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．

3. AND/OR 木階層的挟み撃ち探索

証明数・反証数を閾値とした反復深化法である PDS は詰将棋の求解に有効[8] であり，証明数・反証数は AND/OR 木探索において優れたヒューリスティックである．しかし，PDS では証明数の小さい節点から探索するために，ヒューリスティックが有効でなく証明数が大きい節点が解である場合には，解を見つけるまでに長時間必要となる．そこで我々は，証明数の大きい節点が解である場合でも早く解を見つけることができる並列探索手法として，AOHPAS を提案した[13]．

AOHPAS は，**図 1** のように，1 つのリーダプロセッサが探索木の左側（証明数の小さい節点）から右側（証明数の大きい節点）へと，根節点から PDS で探索する．それ以外のスレーブプロセッサは，リーダプロセッサの探索経路上の OR 節点に 1 つずつ割り当てられる．そして割り当てられた節点の子節点のうち，根とする節点を証明数の大きい節点から証明数の小さい節点の順番に選び，選ばれた節点以下のグローバルな証明数と反証数の閾値[13] 内を PDS により探索する．スレーブプロセッサは割り当てられた節点の子節点において，ローカルな閾値をグローバルな証明数と反証数の閾値の範囲で徐々に増加させ，証明数・反証数ともグローバルな閾値以下のローカルな閾値の探索が終了したら，次の節点の探索に移る．スレーブプロセッサに割り当てる節点は，リーダプロセッサの探索経路を全プロセッサに通知すれば，割当て深さを指定することで一意に決定することができる．このように，プロセッサを階層的に割り当て，探索木の左右から挟み撃つように並列に探索することで，解に近いであろう証明数の小さい節点以下を多くのプロセッサで探索しながら，証明数の大きい節点も並列に探索することができる．

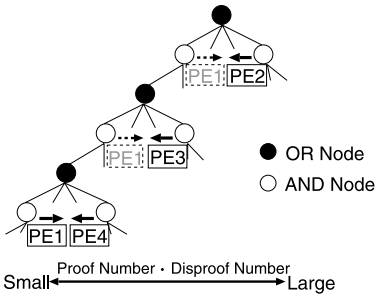

図 1 AND/OR 木階層的挟み撃ち探索
Fig. 1 AND/OR tree hierarchical pincers attack search.

AOHPAS のように，1 つの節点に対してリーダプロセッサとスレーブプロセッサで挟み撃つように並列に探索するのではなく，スレーブプロセッサも証明数の小さい子節点から証明数の大きい子節点へと探索した場合には，逐次探索でも早期に発見することができ，さらに AOHPAS でも多くのプロセッサによって探索される節点の解をより早く発見することになる．しかし，このような探索順では証明数の大きい節点の解を早期に発見することは困難である．また，リーダプロセッサの探索経路上の節点をスレーブプロセッサが並列探索するため，その探索順ではアルゴリズム上の問題点もある．**図 2** において，一番証明数の小さい節点 a をリーダプロセッサ PE1，節点 b をスレーブプロセッサ PE2 が探索しているとする．ここで PE1 の節点 a の探索が終わったときに，次に探索する節点として PE2 が探索中であっても節点 b を選択する方法と，まだ他の PE に探索されていない節点 c を選択する方法が考えられる．前者は PE2 が探索した領域を重複して探索するため，PE2 が他の節点にすぐに再割当てされても，探索効率は低下する．後者では，**図 3** のように，その時点で証明数最小の節点 b は 1 つのスレーブプロセッサ PE2 のみで探索し，最小ではない節点 c をリーダプロセッサ PE1 とスレーブプロセッサ PE3 の複数で探索することとなり，証明数の小さい節点ほど多くのプロセッサを割り当てることができ

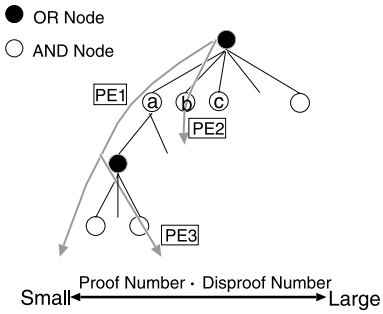

図 2 全プロセッサが証明数の小さい節点から探索
Fig. 2 The search from a node with small proof number by all processors.

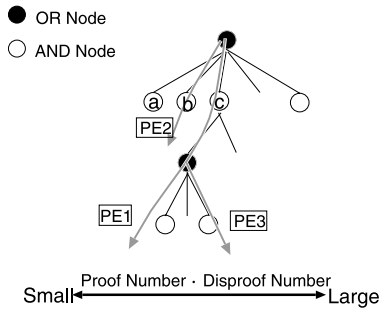

図 3 証明数最小でない節点を複数プロセッサで探索
Fig. 3 The search of a node with not minimum proof number by processors.

ない．PE2 を新たにリーダプロセッサとすることも考えられるが，各プロセッサの役割の変更に必要なコストも大きい．以上のような理由から，AOHPAS では図 1 のように各階層で 2 つのプロセッサが証明数の小さい節点からと証明数の大きい節点から挟み撃つように探索している．

AOHPAS におけるリーダプロセッサの探索は基本的には PDS であるが，多重反復深化は行わない．これは，多重反復深化を行うとスレーブプロセッサの探索範囲が狭くなり，再割当てをするオーバヘッドが大きくなるためである．スレーブプロセッサは，割り当てられた節点の子節点をすべて探索し終わったら，すぐに他のプロセッサがまだ探索していない節点に再割当てされる．探索が打ち切られた後の他の節点に再割当てするオーバヘッドを減らすために，再割当ては深さの浅い節点から行う．スレーブプロセッサの探索が，リーダプロセッサの探索と重複していた場合や，リーダプロセッサの探索によって必要なくなった場合は，スレーブプロセッサがそのことを検知して節点の探索を打ち切る．割り当てられた節点の探索を打ち切る必要があるかどうかは，リーダプロセッサの探索経路と自分の探索経路を比較することで判定する．

AOHPAS では，逐次の PDS 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問題に対しては，証明数が小さい節点から探索するリーダプロセッサが解を発見するため，並列処理による速度向上の効果はあまりなく，探索時間は PDS の探索時間とほぼ同じとなる．しかし，逐次の PDS で探索するまでに長時間かかるような証明数の大きい節点が解である問題に対しては，証明数の大きい節点から探索するスレーブプロセッサが解を発見することが多く，その場合は PDS と比べて探索時間が大幅に短縮する可能性が高いことが確認されている[13]．

4. AND 節点並列探索を追加した AOHPAS

AOHPAS のスレーブプロセッサは，割り当てられた 1 つの節点の探索が終了すると，すぐに他の未探索の節点に再割当てされる．しかし，探索木の深さがあまり深くないなどのために，再割当てされるべき未探索の OR 節点が存在しない場合には，リーダプロセッサが新たな OR 節点を展開するまで，そのスレーブプロセッサはアイドル状態となる．より多くのプロセッサを有効利用するために，OR 節点だけでなく AND 節点にもプロセッサを割り当て，並列探索することが考えられる．しかし AND 節点に多数のプロセッサを割り当てても，プロセッサ数分の高速化を得ることは難しい[14]．

そこで本論文では，AOHPAS において割り当てられる OR 節点がなく，アイドル状態となってしまうスレーブプロセッサのみを AND 節点に割り当て，並列探索する AAA-Search を提案する．これにより，AOHPAS の OR 節点の並列探索を妨害することなく，より多くのプロセッサを有効活用することができる．特に AOHPAS ですべてのプロセッサが有効利用できない，あまり探索木の深さが深くない問題に対して AOHPAS よりも高速化率が向上すると考えられる．また，AOHPAS と同様の OR 節点を探索するので，AOHPAS の求解時間よりも大きく遅くなることはない．深さの深い探索木に対しては，AOHPAS でも高い高速化を得ることができるため，さらに AND 節点を並列に探索することによって高速化すれば，逐次探索に比べて非常に大きな高速化となる．

以下，4.1 節で AND 節点のみを並列探索することによる効果について，4.2 節で AAA-Search の具体的な探索法について述べる．

4.1 AND 節点の並列探索

AND/OR 木では，OR 節点を true と証明するには子節点のうち 1 つでも true であることを証明できれ

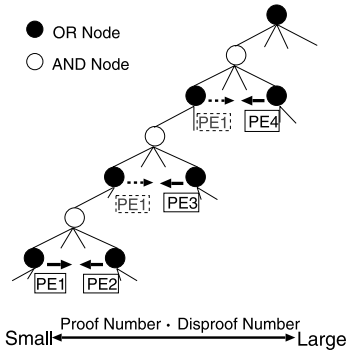

図 4 AND 節点の並列探索
Fig. 4 Parallel search of AND nodes.

ばよいが，AND 節点を true と証明するためには，そのすべての子節点が true であることを証明しなければならない．このため，正解手数が長かったり，AND 節点での分岐が多かったりする場合は，根節点を証明するために証明する必要がある最小節点数も多くなり，探索に多くの時間が必要となる．OR 節点のみを並列に探索すると，各節点に付けた評価値が正しい場合には，解となる節点を探索するプロセッサ以外のプロセッサによる探索は，求解には無関係である．しかし AND 節点を並列に探索すると，各節点の評価値が正しい場合にも，複数プロセッサで子節点を分割して証明するため，高速化が期待できる．

また，AND 節点を並列探索する際に，PDS の探索経路上のすべての AND 節点を並列探索するのでは，プロセッサを割り当てるオーバヘッドが大きい．PDS の探索順では，AND 節点は反証数が最も小さい（最も反証しやすい）子節点から探索する．このため，その最も反証しやすい節点が証明できたとすれば，その他の子節点は証明できる可能性が高いと考えられる．よって，すでに証明された節点の先祖節点にのみプロセッサを割り当てると，オーバヘッドも小さく高速化しやすいと考えられる．

以上のことより，文献 14) において，PDS の探索木の AND 節点のみを並列探索する有効性を検証し，さらに，どのように AND 節点にプロセッサを割り当て並列探索することが効果的であるかについて評価するために，各 AND 節点へプロセッサを割り当てる順番や，1 つの節点に割り当てるプロセッサの数を変更して実験を行った．その結果，**図 4** のように，PDS の探索経路上の AND 節点にスレーブプロセッサを深さの深い節点から階層的に 1 つずつ割り当て，反証数の大きい子節点からグローバルな証明数と反証数の閾値以内を並列に探索するのが最も効率が良く，逐次での探索時間によらず多くの問題で平均的に速度が向上することが確認された．また，多数のプロセッサを用いても，すべてのプロセッサを利用できない場合や排他制御のオーバヘッドがあるため，プロセッサ数分の高速化は得られないことも確認された．

4.2 AAA-Search の探索

AAA-Search は，4.1 節で述べたように AND 節点が少数のプロセッサでの並列探索において効果があるため，AOHPAS における探索すべき OR 節点がないプロセッサのみを AND 節点に割り当てて並列探索する．この探索をすることにより，以下の 3 点のような効果が得られると考えられる．

- 逐次での探索時間が短いものは PDS で探索するプロセッサがすぐに解を発見するので，AOHPAS と同様にあまり探索時間は短縮されない．
- 逐次での探索時間が非常に長いものは，探索木の深さも深くなることが多く，多くのプロセッサが OR 節点に割り当てられ，AND 節点にはあまり割り当てられないことが多い．そのため，AAA-Search は AOHPAS よりも若干高速化する．
- 逐次での探索時間が中程度の問題に対しては，AOHPAS では探索する OR 節点がなくアイドル状態となっていたプロセッサが AND 節点を探索することとなり，AAA-Search は AOHPAS よりも探索時間が短縮できる．

4.2.1 各プロセッサの探索法

AAA-Search では，AOHPAS と同様な並列探索を行い，さらに AND 節点にもスレーブプロセッサを割り当てて並列に探索する．各スレーブプロセッサへの割当ては OR 節点を優先し，OR 節点に割り当てられないときのみ AND 節点へ割り当てる．

まず，リーダプロセッサは AOHPAS と同様に，グローバルな証明数と反証数の閾値を使用した PDS で探索する．スレーブプロセッサの割当ても AOHPAS と同様に，リーダプロセッサの探索経路上の深さの浅い OR 節点から順に行う．リーダプロセッサの探索経路上のすべての OR 節点が他のプロセッサによって探索中，もしくは探索済みの場合には，深さの深い AND 節点から順に 1 つずつプロセッサを割り当てて AND 節点も並列探索する．このとき，4.1 節で述べたように，プロセッサ割当てのオーバヘッドを減らすために，リーダプロセッサの探索により子孫節点のいずれかが証明されている AND 節点のみにプロセッサを割り当てる．再割当ての場合も同様に，プロセッサは，割当て可能な未探索の OR 節点がある場合には OR 節点に，ない場合には AND 節点に再割当てされる．各

OR 節点に割り当てられたプロセッサは，AOHPAS のスレーブプロセッサと同様に探索する．各 AND 節点に割り当てられたプロセッサは，リーダプロセッサのグローバルな閾値を引き継ぎ，割り当てられた節点の子節点のうちから，根とする節点を，反証数の大きい節点から小さい節点の順番に選ぶ．つまり，AND 節点においても，探索木の左右からリーダプロセッサとスレーブプロセッサで挟み撃つように探索する．

AND 節点の並列探索よりも OR 節点の並列探索を優先するため，リーダプロセッサの探索によって，新たな OR 節点が作成された場合，AND 節点に割り当てられているプロセッサも OR 節点へ再割当てする．しかし，AND 節点に割り当てられたプロセッサが複数あり，探索可能な OR 節点が発生したことを複数のプロセッサが同時に検知すると，そのすべてのプロセッサが AND 節点の探索をやめてしまう．ところが実際に OR 節点を探索できるプロセッサ以外はまた AND 節点に割り当てられることとなるため，その情報へのアクセスを排他的にする必要がある．探索可能な OR 節点が新たに発生した場合，AND 節点に割り当てられたプロセッサがすぐに AND 節点の探索を中止して OR 節点に再割当てされるためには，AND 節点に割り当てられたスレーブプロセッサが，つねに探索可能な OR 節点が発生したかどうかをチェックしていなければならず，オーバヘッドとなる．そこで AAA-Search では，AND 節点に割り当てられたプロセッサは，1 つの子節点の探索が終了するまで探索を中断しない．これにより，チェックは 1 つの子節点の探索が終了したときのみでよい．また，割り当てられた節点の子節点の探索を途中で中止すると，その後また中断した節点の探索が必要となったときに，トランスポジションテーブルに保存されている情報が使用できたとしても，最初から探索しなおす必要がある．よって，AND 節点に割り当てられたスレーブプロセッサは，割り当てられた節点の 1 つの子節点の探索が終了したときのみに，リーダプロセッサの探索経路上に，未探索の OR 節点が存在するかどうかを確認し，もし存在すれば，OR 節点を探索する，未探索の OR 節点が存在しなければスレーブプロセッサは AND 節点の探索を続ける．

AAA-Search のプロセッサが AND 節点に再割当てされる様子を，**図 5** に示す．図 5 のようにリーダプロセッサである PE1 の探索経路上の OR 節点に，スレーブプロセッサ PE2，PE3，PE4 が割り当てられているとする．ここで，PE2 の探索が終了すると，この時点で他のスレーブプロセッサが探索していない OR 節点がなく，また，AND 節点 a の子節点の 1 つがすでに証明されているため，最も深さの深い AND 節点 a に再割当てされ，その子節点以下を探索する．

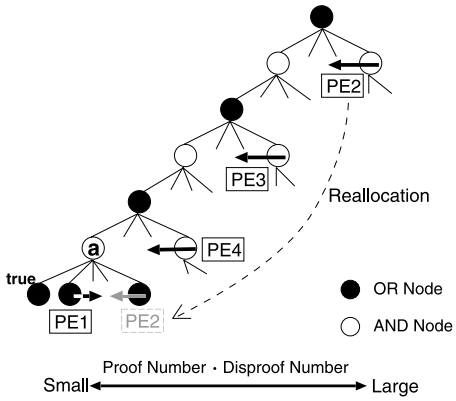

図 5 AAA-Search
Fig. 5 AAA-Search.

AAA-Search ではグローバルな閾値を用いて，スレーブプロセッサの処理粒度を増大させるため，排他制御が必要な領域へのアクセスも少なくてすみ，プロセッサ数を増加させたときのオーバヘッドも AND 節点のみを並列探索するときより小さいと考えられる．

また，逐次探索では同一局面となる節点の情報がトランスポジションテーブルに保存されているために探索する必要のない節点を，並列探索した場合にはテーブルに情報が保存される前に他のプロセッサが探索してしまう可能性がある．本論文では AAA-Search を共有メモリ型並列計算機上に実装しており，共有トランスポジションテーブルを用いている．これにより節点の再探索を削減している．トランスポジションテーブルに保存する情報を増加することで，さらに再探索を減少させることも考えられる．

4.2.2 探索打ち切り

同一の探索木を複数のプロセッサで左右から探索するため，リーダプロセッサとスレーブプロセッサの探索が重複したり，どちらかの探索結果によりもう一方の探索が必要なくなったりする場合がある．AOHPAS と同様に，節点を割り当てるために用いたリーダプロセッサの探索経路を用いることによって，スレーブプロセッサが，リーダプロセッサとスレーブプロセッサの探索の重複，つまり同じ節点の探索と，スレーブプロセッサの割り当てられた節点におけるリーダプロセッサの探索の終了とを検出する．

リーダプロセッサとの探索範囲の重複による探索の終了は，スレーブプロセッサがリーダプロセッサとの探索経路を比較し，割り当てられた節点の子節点以下までの探索経路が等しくなることにより検出できる．

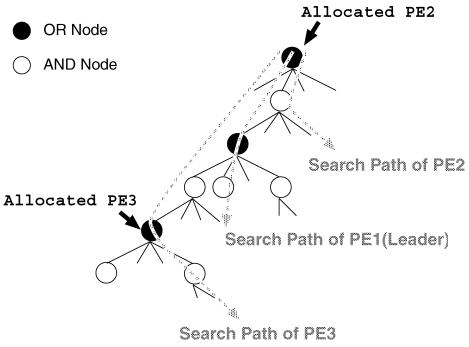

図 6 探索終了検出
Fig. 6 An example of termination detection.

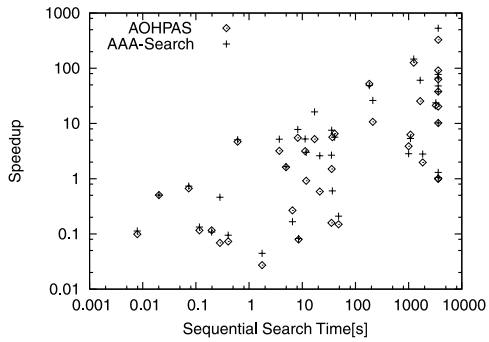

図 7 AAA-Search と AOHPAS の PDS に対する高速化率
Fig. 7 Speedup of AAA-Search and AOHPAS to PDS.

図 6 の PE1 と PE2 の探索経路は，PE2 の割り当てられた 1 つ下の節点まで重なっている．そのため，PE2 は割り当てられた節点の探索を終了する．割り当てられた節点でのリーダプロセッサの探索の終了は，その節点より浅い 2 つの探索経路が異なることで検知することができる．図 6 の PE3 の探索経路は割り当てられた 1 つ上の節点で PE1 の探索経路と異なっている．そのため，PE3 は割り当てられた節点の探索を終了する．探索を終了したスレーブプロセッサは他の節点に再割当てされる．

OR 節点を並列に探索する場合には，スレーブプロセッサが割り当てられた節点の子節点を true と証明することで，その親節点も true となる．このため，その節点以下のリーダプロセッサの探索は必要がなく，打ち切ることができる．AND 節点を並列に探索する場合には，スレーブプロセッサが割り当てられた節点を false，もしくは与えられた閾値では証明できないとすると，その親節点も証明できないこととなる．このため，その節点以下の探索は必要なく，リーダプロセッサの探索を打ち切ることができる．これらの場合には，スレーブプロセッサが割り当てられた節点の探索結果をリーダプロセッサに伝えることによって，リーダプロセッサは探索を中止する．探索の打ち切りは，リーダプロセッサが深い節点を探索しているときに，探索木の浅いところで起こる場合にその効果が大きく，スーパリニアスピードアップを得られる可能性もある．

5. 性能評価

AAA-Search の有効性を確認するために，共有メモリ型並列コンピュータ Sun Enterprize 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．評価には詰将棋の問題集「続詰むや詰まざるや」[15] の前半 100 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 99 題を解いた結果を評価する．制限時間は 3,600 秒とし，並列探索である AAA-Search，AOHPAS，逐次探索である PDS のすべてにおいて制限時間以内に解けなかった問題 49 題は評価の対象としない．並列探索時に用いるグローバルな閾値は，AOHPAS，提案手法ともに 5 ずつ増加させた[13]．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ている．

図 7 に PDS の探索時間に対する，14 プロセッサを用いたときの AOHPAS と AAA-Search の高速化率を示す．高速化率は式 (1) により計算する．PDS では解くことができないが，並列探索により解くことができた問題は PDS の探索時間を 3,600 秒とした．

$$\text{高速化率} = \frac{\text{逐次 PDS の探索時間}}{\text{並列探索時間}} \quad (1)$$

図 7 より，AAA-Search，AOHPAS ともに PDS の探索時間が短いものに対しては高速化率は低いが，PDS の探索時間が長くなるほど高速化率も高くなる．つまり，AAA-Search で AND 節点も並列に探索することによって AOHPAS の大きな高速化が急激に下がることはないことが分かる．また，PDS の探索を 3,600 秒で打ち切った問題では，実際の探索時間は 3,600 秒よりも長いため，それぞれの並列化手法の高速化率もさらに高いものとなると予想できる．

次に，**図 8** に AOHPAS の並列探索時間に対する AAA-Search の高速化率を PDS の探索時間ごとに示す．AOHPAS，AAA-Search ともに 14 プロセッサ使用したときの探索時間を用いて，高速化率を式 (2) のように計算した．AOHPAS では制限時間以内に解けず，AAA-Search では解くことができた問題は，AOHPAS の探索時間を 3,600 秒とした．

$$\text{高速化率} = \frac{\text{AOHPAS の探索時間}}{\text{AAA-Search の探索時間}} \quad (2)$$

AOHPAS に対する AAA-Search の高速化率は最高

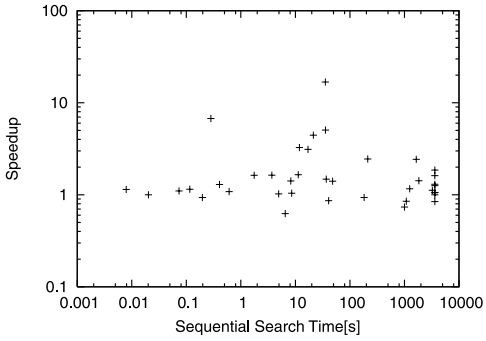

図 8 逐次探索時間ごとの AAA-Search の AOHPAS に対する高速化率
Fig. 8 Speedup of AAA-Search to AOHPAS by sequential search time.

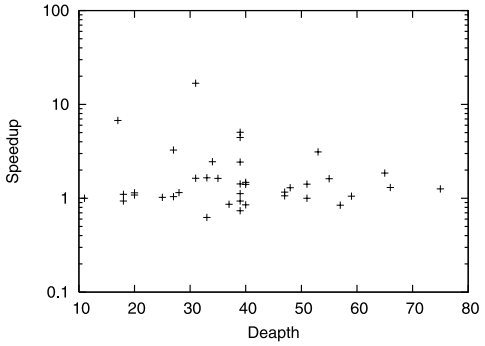

図 9 最大探索深さごとの AAA-Search の AOHPAS に対する高速化率
Fig. 9 Speedup of AAA-Search to AOHPAS by the maximum search depth.

16.8 倍，相乗平均 1.5 倍となった．図 8 に示されている各問題は，以下の 3 つのタイプに分類することができる．

(a) PDS の探索時間が 10 秒未満と非常に短い問題
(b) PDS の探索時間が 10 秒から 100 秒程度の AAA-Search の AOHPAS に対する高速化率が特に高い問題
(c) PDS の探索時間が 100 秒以上と非常に長い問題

この 3 つのタイプについて，AAA-Search の効果を述べる．

まず，タイプ (a) の問題に対しては，リーダプロセッサがすぐに解を見つける可能性が高い．そのため，AOHPAS，AAA-Search ともにほとんどのスレーブプロセッサを使用できず，多くの場合，両手法の探索時間はほぼ同じとなる．

次に，タイプ (b) の問題は，3 つのタイプの中で AAA-Search が最も効果的であり，最高 16.8 倍，相乗平均 2.8 倍となった．これらは，リーダプロセッサがすぐに解を見つけることができず，さらに AOHPAS ではスレーブプロセッサが探索する OR 節点が多くなくアイドル状態となることも多いため，すべてのプロセッサを有効利用できていなかった．AAA-Search では，これらのプロセッサを用いて AND 節点も並列探索することにより，探索時間が短縮されたと考えられる．

最後に，タイプ (c) の問題は，探索木は深いことが多いため，OR 節点も多い．そのため AAA-Search ではプロセッサを OR 節点に優先して割り当てるため，あまり AND 節点に割り当てられない．つまり，AAA-Search でも OR 節点を多く並列探索することになる．そのため，AOHPAS に対しての高速化率はあまり高くならず，最高 2.4 倍，相乗平均で 1.2 倍である．

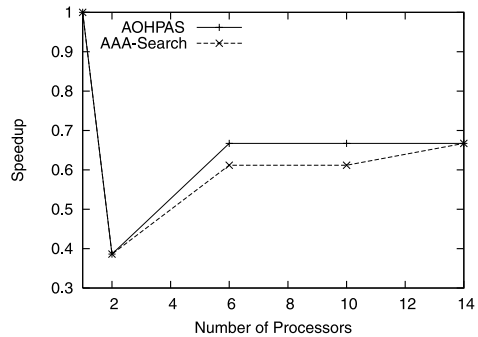

図 10 タイプ (a) の高速化率の代表例
Fig. 10 An example of speedup of type (a).

しかし図 7 より，これらの問題に対して AOHPAS の高速化率は非常に高い，そのため AAA-Search は逐次探索である PDS に対して，最高 530.2 倍，相乗平均 20.2 倍の高速化を得ることができた．

また，式 (2) により計算した AAA-Search の AOHPAS に対する高速化率を最大探索深さごとに**図 9** に示す．図 9 からも，探索木のあまり深くない問題に対して，AAA-Search の AOHPAS に対する効果が大きいことが分かる．

AAA-Search の AOHPAS に対する高速化率の全体相乗平均は 1.5 倍であるが，逐次探索，AOHPAS ともに制限時間以内に解くことができなかった問題のうち，AAA-Search を用いることにより，新たに 2 問を解くことが可能となり，合計で 50 問を解くことができた．

次に，(a)，(b)，(c) の 3 つのタイプそれぞれの具体例について述べる．**図 10** にタイプ (a)，**図 11** にタイプ (b)，**図 12** にタイプ (c) のそれぞれ代表的な例における，プロセッサ数を 2，6，10，14 としたときの式 (1) を用いて計算した AOHPAS と AAA-Search

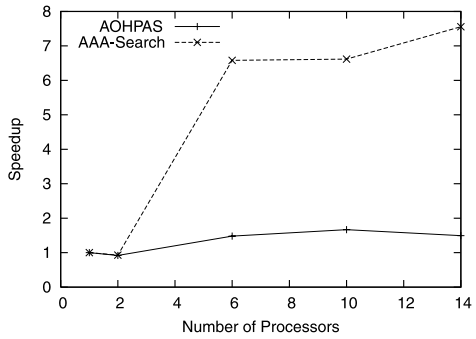

図 11 タイプ (b) の高速化率の代表例
Fig. 11 An example of speedup of type (b).

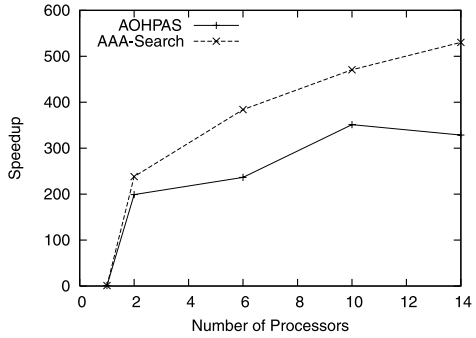

図 12 タイプ (c) の高速化率の代表例
Fig. 12 An example of speedup of type (c).

の PDS に対する高速化率を示す．

図 10 は，PDS の探索時間は 0.07 秒と短く高速化率が AOHPAS とほぼ変わらない例，第 30 番（左下隅）15 手詰である．PDS で十分に早く解が求まるため，並列探索すると，並列化のためのオーバヘッドにより逐次探索よりも遅くなっていることが分かる．本例のような問題では，探索木が小さいことが多く並列探索可能な節点も少ない．また，リーダプロセッサがすぐに解を発見するため，AOHPAS ではプロセッサ数を増やしても探索時間にほとんど変化がない．AAA-Search ではプロセッサ数 6，10 の場合は AND 節点も並列探索するための排他制御などに必要なオーバヘッドの増加により，AOHPAS よりも遅くなっている．

図 11 は，PDS の探索時間は 35.67 秒とあまり長くなく，AOHPAS よりも AAA-Search が大きく高速化する例，第 1 番 15 手詰である．このように PDS の探索時間が 10 秒から 100 秒程度の問題に対しては，AND 節点を探索するプロセッサが多くなり，AAA-Search による高速化が大きい．本例では AOHPAS の速度はプロセッサ数を増やしてもあまり変化しない，つまり，多くのプロセッサで OR 節点を並列探索する効果は少ない．しかし AAA-Search によって AND 節点を並列に探索することにより，14 プロセッサ使用した場合に AOHPAS と比べて 5.0 倍高速化された．

図 12 は，PDS の探索時間が非常に長い問題である例，第 96 番 31 手詰である．本例では 3,600 秒で探索を打ち切ると PDS では解を得ることができなかった．AOHPAS，AAA-Search ともに PDS の探索時間を 3,600 秒として高速化率を計算しても，スーパリニアスピードアップとなっている．このような場合は AOHPAS による OR 節点並列探索の効果が大きい．しかし，本例では AOHPAS では少ないプロセッサ数でも早く解を得ることができているため，プロセッサ数を増やしてもあまり高速化率は上昇しない．逆に 10 プロセッサ使用時よりも 14 プロセッサ使用時の方が解を得るまでの時間が長くなっている．しかし，AAA-Search では OR 節点だけではなく AND 節点も並列探索することにより，より多くのプロセッサを有効に使用できるため，プロセッサ数が増えても高速化率は上昇している．AOHPAS に対する AAA-Search の高速化率はそれほど高くなく，本例では 1.6 倍となったが，このとき AAA-Search の逐次探索 PDS に対する高速化率は 530.2 倍となった．

6. おわりに

本論文では，AND/OR 木の並列探索アルゴリズムである AOHPAS において，OR 節点のみではなく，AND 節点にもプロセッサを割り当てることで，さらに多くのプロセッサを効率良く使用する AAA-Search を提案した．AAA-Search では AOHPAS で使用しないプロセッサのみを AND 節点に割り当てて探索するため，AOHPAS の大きな高速化を保ちながら，さらに AND 節点の探索による並列効果を得ることができる．AAA-Search を詰将棋の求解を例に実装し評価した結果，AOHPAS では多くのプロセッサを有効に使用できない可能性の高い，逐次探索での探索時間が 10 秒から 100 秒程度の問題において，AOHPAS に対してから最低 0.9 倍から最高 16.8 倍と相乗平均 2.8 倍高速化されることを確認することができた．また，AOHPAS で多くのプロセッサを使用することのできる逐次探索での探索に長時間かかる問題でも，AOHPAS に対して相乗平均 1.2 倍の高速化が得られ，逐次探索に対しては最高 530.2 倍，相乗平均 20.2 倍と大きく高速化することが確認できた．

本論文では使用プロセッサ数を最高 14 としたが，プロセッサ数が多くなった場合には，リーダプロセッサの探索経路上の 1 節点に 1 プロセッサの割当てのみではなく，複数割り当てることによってさらにプロセッ

サの使用率を向上させる必要もあると考えられる．

また，AAA-Search は各プロセッサの探索法に詰将棋探索で有効だとされている逐次探索アルゴリズム PDS を用いている．AAA-Search は証明できる問題を対象に，高速に解を求めることを目的としているが，PDS は反証される問題に対しても非常に有効なアルゴリズムである．そこで，証明だけではなく反証される問題に対しても，並列探索によって効果的に高速化するアルゴリズムを考える必要がある．

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) 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).
9) Feldmann, R.: Spielbaumsuche auf massiv parallen Systemen (Game trees on massively parallel systems), Ph.D. Thesis, University of Paderborn (1993).
10) Brockington, M. and Schaeffer, J.: APHID GAME-TREE SEARCH, *Advances in Computer Chess*, Vol.8, pp.69–91 (1997).
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) 鷹野芙美代，関根敦史，佐田宏史，前川仁孝，六沢一昭：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，情報処理学会論文誌：コンピューティングシステム，Vol.45, No.SIG 11(ACS 7), pp.280–289 (2004).
14) 鷹野芙美代，佐田宏史，前川仁孝，六沢一昭，宮崎収兄：AND/OR 木における AND 節点に対する並列探索の評価，電子情報通信学会技術研究報告，Vol.104, No.240, pp.31–36 (2004).
15) 門脇芳雄：続詰むや詰まざるや，平凡社 (1978).

(平成 17 年 1 月 24 日受付)
(平成 17 年 5 月 6 日採録)

鷹野芙美代（学生会員）

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．

佐田 宏史

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）

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．電気学会会員．

六沢 一昭（正会員）

1981 年早稲田大学理工学部電子通信学科卒業．1983 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．同年沖電気工業（株）入社．1986〜1990 年および 1993〜1995 年（財）新世代コンピュータ技術開発機構出向．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．

宮崎 収兄（正会員）

1973 年京都大学理学部卒業．同年沖電気工業（株）入社．1979 年イリノイ大学コンピュータサイエンス学科修士課程修了．1982 年から 3 年間（財）新世代コンピュータ技術開発機構出向．1994 年千葉工業大学情報工学科教授．現在に至る．工学博士．データベースを中心とする情報システムの研究に従事．日本データベース学会，電子情報通信学会，人工知能学会等会員．