並列ゲーム木探索のための分散共有ハッシュ法の評価

佐藤信弘[†,☆] 新藤雅也[†,☆☆]
野下浩平[†] 中山泰一[†]

ゲーム木探索における局面表（トランスポジション表）は，局面の探索結果を表に登録し，同一局面の探索を表の参照ですませる技法である．本論文では，並列的なゲーム木探索のために計算機間で共有する局面表を実現し，その有効性を実験的に評価する．実験対象として並列選択の比較回数を決定する探索問題をとりあげる．通信速度の遅いネットワークで結合された並列計算環境において，分散的に共有するハッシュ法による共有局面表の実現方法を2種類比較する．代表的な並列アルゴリズムを2つ選び，実行時間や各種オーバヘッドを測定し，共有局面表の効果を調べる．著しい結果として8台の計算機により逐次計算の7倍以上の速度向上を得た．また逐次計算では時間がかかりすぎ解けなかった問題が共有局面表による並列計算で解くことができた．本論文の分散共有ハッシュ法により，並列計算の性能向上の目標である十分良い台数効果が実現できることを実証した．

Some Experiments on the Distributed Shared-Hashing Method for Searching Game-Trees in Parallel

Nobuhiro Sato,[†,☆] Masaya Shindo,[†,☆☆] Kohei Noshita[†]
and Yasuichi Nakayama[†]

In game-tree searching, transposition tables are used for eliminating repetitions of the identical computation for reappeared positions. For parallel searching on a distributed parallel computer-cluster, two types of the shared-hashing method are presented for implementing the global transposition table. For evaluating our method in terms of various overheads as well as the computation time, a certain selection problem is used and two parallel algorithms are implemented to solve it. As one of our experimental results, more than seven-fold speedups have been achieved on eight processors. By our method, several instances of the problem have been solved, which could not be solved on a single computer. The experiments have proved that our distributed shared-hashing method is efficient enough to show a good performance near the maximum on a distributed parallel environment with slow interprocessor communication.

1. はじめに

チェスなどの多くのゲームのプログラムは，ゲーム木探索のアルゴリズムが基礎となっている．ゲーム木探索の高速化技法の中で最も重要なものの1つとして局面表（transposition table，トランスポジション表）がある[4),9)]．局面表技法は，局面に対して探索した結果を表に保存して，同一局面の探索を表の参照ですませるという表計算の一種である．一方，ゲーム木探索のための並列アルゴリズムの研究も数多くある[2),5)~7),9),12)]．並列探索においても局面表は重要である．局面表を複数の計算機で共有し，それぞれの計算機で同じ局面に対する探索を避ける必要がある．局面表を共有しないと，一般に探索数が増加し，十分な速度向上が得られない．

本論文では，ネットワーク（Ethernet）で複数の計算機を結合した分散的な（非共有メモリ型）並列計算システムにおいて，ゲーム木の並列探索向きの局面表の構成法を調べ，（チェスでない）応用例を用いて能率の実験的評価を行う．ここでは，共有する局面表（共有局面表）は，分散した計算機の間で共有するハッシュ表によって実現する．これを分散共有ハッシュ法とよぶことにする．本論文の議論では分散的な並列計算システムに限定する．専用ハードウェアや共有メモリ型並列計算機では，データの共有方法がまったく異なるからである．本論文のような並列計算システムで局面

† 電気通信大学電気通信学部情報工学科
Department of Computer Science, The University of Electro-Communications
☆ 現在，株式会社アイザック
Presently with ISAC, Inc.
☆☆ 現在，サン・マイクロシステムズ株式会社
Presently with Sun Microsystems, Inc.

表を実現するための並列探索用通信ライブラリについては筆者らが文献 8) で報告したが，本論文でもそれを改良して使用する．

実際のチェスプログラムの並列探索では，局面表を使った実験が行われているが，並列探索向きの局面表の実現法，特に表の共有方法，チェス以外の応用という見方ではほとんど報告はない．文献 5)，12) は本論文と関係が深いが，もっぱらチェスを実験対象にしており，(局面表というより) 並列アルゴリズムの改良を目的とする研究である．さらに，CPU やネットワークの性能など並列計算の環境がかなり異なる．なお，筆者らはミニ・オセロに関する実験結果を報告した[6]．本論文では，実験対象のゲームとして，並列選択 (parallel selection) の比較回数を決定する問題をとりあげる[1),3)]．これはチェスやオセロとは異なる性質を持つ．すなわち，チェスのような探索時間が短い実時間的応用ではなく，十分時間をかけて解く探索問題であり，さらに，チェスやオセロに較べて (逐次計算における) 局面表のヒット率が非常に高い．

本論文で用いる分散的並列計算システムでは，データを共有する場合，計算機間の通信速度を考慮することが重要である．CPU の計算速度に較べて通信速度が極端に遅いからである．また，一般的な分散的並列計算システムでは，データの一貫性 (データの書き込みと読み出しのタイミングによる矛盾が生じないこと) を強く保持 (sequential consistency) する必要がある[10)]．しかし，筆者らは，共有局面表の実現のためにはある程度緩い一貫性制御ですますことができ，(強い一貫性制御を行う場合に較べて) 相対的に局面表の管理が能率良くできることを報告したが[6)]，本論文でもこの緩い一貫性制御を使う．

本論文の結果の概要を述べる．まず，分散共有ハッシュ法により実現した共有局面表と，それによるゲーム木の並列探索システムを説明する．次に，システム全体で単一の共有局面表を持つ方法と，複数の計算機に分散し全体として共有局面表を持つ方法について，それぞれを実現して比較する．文献 5)，12) では，明示して書かれていないが，システム全体で単一の表を持つ方法を使っているようであり，文献 6) では分散管理する方法の 1 つを提案している．

次に，実験対象のゲームとしてとりあげる並列選択問題の特徴を説明する．これは，局面表のヒット率が高く，局面表の共有する効果を明確に示すのに適当であることが期待できる．実際，実験結果によれば，共有局面表の有効性の程度がチェスやオセロに較べて格段に良いことが分かる．

それぞれの計算機の中だけの局面表 (局所ハッシュ表) を使う方法に対して，局所ハッシュ表に加えて共有局面表 (共有ハッシュ表) を用いる方法の速度向上比を実験的に示す．また，本論文では共有局面表の評価のために，性質の異なる 2 つの並列アルゴリズムを代表としてとりあげる．それぞれのアルゴリズムについて，実際に並列選択問題を解き，計算時間と諸種のオーバヘッドを測定する．これにより，ある程度アルゴリズムと独立に共有局面表の有用性を確かめることができる．ここで，2 つのアルゴリズムは単純な Tree-Splitting (木分割) とチェスで有効とされている PVSplit (最強応手手順分割) である[2),5),12)]．後者のアルゴリズムは，その後いろいろの変形が提案されているが，本論文のような共有局面表の評価のためには基本となる PVSplit で評価すれば十分であると考えられる．実際，これらのアルゴリズムを用いて，十分良い台数効果 (計算機台数に近い速度向上比) を示すことができた．

本論文の著しい結果として，8 台の計算機で 7 倍を超える速度向上比を示すことができた．また，これまで逐次計算では時間がかかりすぎ解くことができなかったいくつかの問題に対して，共有局面表を持つ並列計算によってはじめて解くことができた．これらの結果をまとめると，通信速度が遅い分散型の並列計算システムでも，ゲーム木の応用例によっては，本論文で示した分散共有ハッシュ法により，十分良い (つまり並列計算の性能向上の目標に近い) 台数効果を実現できることが分かる．

2. 分散共有ハッシュ法

本章では，分散共有ハッシュ法による並列探索システムを説明し，共有局面表の構成法を 2 種類示す．一般に，分散計算環境におけるデータ共有には，データの厳密な一貫性保持のために大きいオーバヘッドが発生する．厳密な一貫性を必要としない例として，本論文で扱うようなゲーム木探索の局面表がある．その性質として，たとえば，局面に対する計算結果は，どの計算機で解いても同一であり，また局面表の結果を利用できない場合でも，独立した計算によって同一の結果を求めることができる．このような局面表の性質を利用すれば，データの一貫性を緩く保持すればよく，オーバヘッドを小さくできる[6)]．本論文は，共有局面表の管理に緩い一貫性制御を用いて，通信速度が遅い環境でも共有局面表の効果が大きいことを示す．

2.1 システム構成

図 1，**図 2** に本論文で提案するシステム構成を示

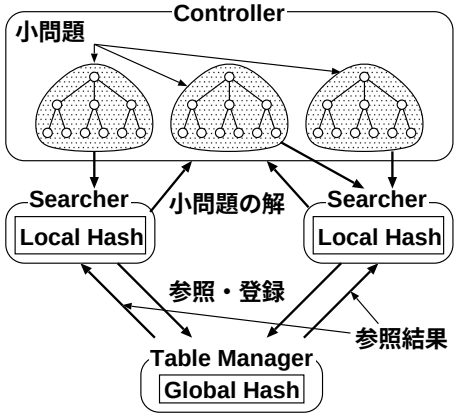

図 1 共有ハッシュ表集中型
Fig. 1 The centralized shared-hashing table.

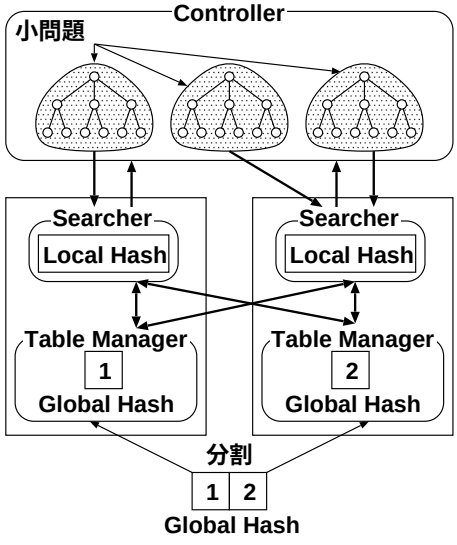

図 2 共有ハッシュ表分散型
Fig. 2 The distributed shared-hashing table.

す．本システムは Controller（制御部），Searcher（探索部），Table Manager（表管理部）から構成される．Controller は小問題を生成し，それを Searcher に渡す．Searcher は与えられた小問題を解き，結果を Controller に返す．Table Manager は全 Searcher が利用できる Global Hash 表（共有ハッシュ表）を管理する．Searcher はまず，自らが持つ Local Hash 表（局所ハッシュ表）を参照する．もし局面が登録されていなければ，Table Manager へ Global Hash 表への参照を依頼する．Table Manager は参照要求を受け付け，結果を Searcher に返す．また，Searcher が探索した局面の情報を Global Hash 表に登録する．本論文では Global Hash 表を 1 つにする方法（図 1）と分散させて複数の Table Manager で管理する方法（図 2）を提案する．

2.1.1 共有ハッシュ表集中型

図 1 は共有ハッシュ表集中型の構成である．構成は容易である．しかし，複数の Searcher から同時に要求があったとき，応答時間が長くなる．その結果，通信のオーバヘッドが増加する．

2.1.2 共有ハッシュ表分散型

図 2 は共有ハッシュ表分散型の構成である．Global Hash 表を分割し，複数の Table Manager で管理する方法である．Searcher と同数の Table Manager を用意し，1 台の計算機に Searcher と Table Manager を割り当てる．Global Hash 表を分割することにより，Searcher からの要求を分散することができる．複数の Searcher から同時に要求があっても，要求が分散し，応答時間が増加しないことを期待するものである．Table Manager と Searcher を 1 台の計算機に割り当てるので，それぞれのプロセスがもう 1 つのプロセスの負荷の影響を受け，オーバヘッドが増加する可能性がある．

3. 実 験 方 法

本章では，実験に用いるゲーム木の並列選択問題，分散共有ハッシュ法の評価に用いる並列探索アルゴリズム，並列計算機環境，実験結果の評価のための各種オーバヘッドの定義について説明する．

3.1 並列選択問題

ゲーム木の例として，並列選択問題をとりあげる．並列選択問題の定義，ゲーム木の実現方法，ゲーム木としての特徴，小さいパラメータに対する解を示す．

3.1.1 並列選択問題の定義

整数 n と t に対して $(1 \leq t \leq \lfloor n/2 \rfloor, n \geq 2)$，相異なる n 個の要素からなる全順序集合のなかで，最も大きい t 個を選択する並列比較の回数を考える．このとき，上位 t 個の相対的な順序は問わない．また，1 回の並列比較では同一の要素が 2 つ以上の比較に関与できない．さらに，並列比較の結果を利用して次回の比較の相手を選ぶことができる．これらの条件の下で，上位 t 個の要素を選択するのに（最悪の場合に）必要十分な並列比較の回数を $U(n,t)$ で表す．選択問題や $U(n,t)$ に関連する研究については文献 1)，3) を参照されたい．

3.1.2 ゲーム木の実現方法

並列比較回数 $U(n,t)$ の計算は，ゲーム木探索問題として定式化できる．

先手 比較する要素の組合せを決定する．その中で比

表 1 局面表の特性（$U(13,4)$）
Table 1 Hit rate of the transposition table for $U(13,4)$.

深さ	探索数	ヒット数	登録数	ヒット率（%）
3	536993	525854	11139	97.9
5	487062405	487037942	24463	99.9
7	11225076	11224317	759	99.9
9	18017	18013	4	99.9
11	20	20	0	100

較回数が最小となる組合せを選ぶ（ミニ側）．

後手 先手で決定した比較の組合せに対し半順序関係を与える．与えた半順序関係の中で比較回数が最大となる関係を選ぶ（マックス側）．

半順序関係（および上位の個数）はゲームの局面に対応する．先手の局面の計算結果を局面表に登録する．局面の同一性の判定は，よく知られているように，半順序関係の同型性判定になり，完全に行うことは手間がかかりすぎる問題であるとされている．ここでは，節点に接続している枝の個数（次数）について整列して半順序関係をある程度標準化した形で表に登録する．これによって，表の検索の際，同一の局面でも異なる局面と判定されることがある．そのような場合には計算する必要がありうるが，表の検索が誤った結果を返すことはない．

まず，ゲーム木の逐次探索によって小さい n に対する $U(n,t)$ を計算した．その結果の代表として特に $U(13,4)$ をとりあげる（ほかのパラメータについてもほぼ同様の振舞いを示す）．逐次探索による局面表の特性を**表 1** に示す．

ヒット率が高いだけでなくヒット数と比較して登録数が少ない．同一の局面が何度も発生していることが分かる．また，登録数が少ないので，メモリ使用量が少ない．本論文で作成したプログラムでは，1 つの局面の登録に 56 バイト必要である．全体の登録数を大きく見積もり 50,000 とすると約 2.8 MB で十分である．この探索問題では，メモリに関して効率の良い探索ができ，実験で扱った範囲では表のあふれ（オーバフロー）が生じない（注：深さ 11 の登録数が 0 であるのは 9 以下の深さで登録した局面を探索することによる）．

3.1.3 小さな n に対する値

小さな n に対する $U(n,t)$ の値を**表 2** に示す．この中で下線のついた値は逐次計算では求められなかったものであり後述する．星印のついた値は一般的な式から導いたものである．それ以外の値については，逐次計算で求めたものである．なお，独立して作成した別のプログラムによって計算結果の検証を行った．これらの結果を一部に用いて，理論的な考察によって，一般の n と t に対する $U(n,t)$ の上界式，下界式，正確な値などを表す式を導いた[11]．

表 2 $U(n,t)$ の値
Table 2 The exact values of $U(n,t)$.

n \ t	1	2	3	4	5	6	7	8
2	1							
3	2							
4	2	2						
5	3	3						
6	3	4	3					
7	3	4	4					
8	3	4	4	4				
9	4	5	5	5				
10	4	5	5	5	5			
11	4	5	5	6	6			
12	4	5	5	6	6	6		
13	4	5	6	6	6	6		
14	4	5	6	6*	6	6	6	
15	4	5	6	6*	6	?	?	
16	4	5	6	6*	?	?	?	?

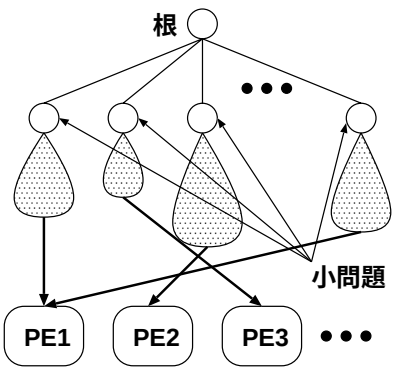

図 3 Tree-Splitting の概念図
Fig. 3 Schematic diagram of Tree-Splitting.

3.2 並列探索アルゴリズム

本節では，実験で用いる並列探索アルゴリズムの Tree-Splitting と PVSplit の概略を説明する．

3.2.1 Tree-Splitting

Tree-Splitting（木分割）は素朴な並列ゲーム木探索アルゴリズムである[2),12)]．**図 3** にアルゴリズムの概念図を示す．

Tree-Splitting はゲーム木の根の子供を 1 単位とする小問題をそれぞれの計算機に割り当てる．小問題のサイズが大きく，並列化の際の通信などのオーバヘッドは小さい．しかし，計算機が小問題をすべて解いたとき，残りのすべての計算機が小問題を解くまでアイドル状態になる．小問題のサイズが大きいことがこのオーバヘッドが大きくなる原因となる．

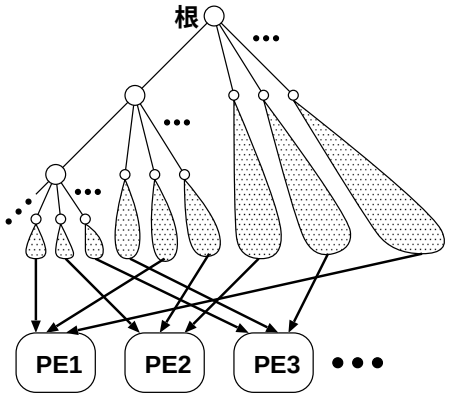

図 4 PVSplit の概念図
Fig. 4 Schematic diagram of PVSplit.

3.2.2 PVSplit

PVSplit（Principal Variation Splitting，最強応手手順分割）はゲーム木の PV（Principal Variation）に沿って木を分割する方法である[2),5),12)]．**図 4** にアルゴリズムの概念図を示す．

まず，木の一番左（PV）から探索する．葉まで到達したら，最初の評価値を得て 1 つ上の節点に戻る．上に戻った時点で，一番左の子供の探索は終了している．残りの子供は Tree-Splitting と同様に分割し，小問題を各計算機に割り当てる．すべての子供の探索が終了したら 1 つ上の節点に戻る．戻った節点でも同様に，一番左以外の子供に Tree-Splitting を適用する．これを根の子供の探索がすべて終了するまで繰り返す．分割した小問題の探索は 2 種類あり，通常のウィンドウによる探索と空ウィンドウによる探索がある．

Tree-Splitting では最初に各計算機に割り当てる小問題は最大のウィンドウで探索しなければならない．しかし，PVSplit では分割する前に PV の値を得ている．この PV の値により，小問題は最初から小さいウィンドウで探索することができる．したがって，一般に Tree-Splitting より高速に計算できる．また，空ウィンドウで探索する方法は適切な探索の順序付けがされていると，速度向上に大きな効果がある．PVSplit では，1 つ上の節点に戻るためには，すべての子供の探索が終了するまで待たなければならない．これにより，Tree-Splitting と同様のオーバヘッドが発生する．

3.3 実験環境

本論文では**表 3** の計算機を用いて分散計算環境を構成する．通信媒体としては Ethernet を用いる．プロセス間のデータ通信は著者らが作成した分散的実行管理機構[8)]を一部改良して実現する．すでに述べたが，評価のための実験対象として主に $U(13,4)$ を用いる．

表 3 使用計算機
Table 3 Computers used in our experiments.

	CPU	メモリ	$U(13,4)$ の実行時間（分）
PE1	Celeron 500 MHz	128 MB	751
PE2	Celeron 500 MHz	128 MB	751
PE3	Celeron 500 MHz	128 MB	751
PE4	Celeron 500 MHz	128 MB	751
PE5	Alpha 533 MHz	256 MB	485
PE6	Alpha 533 MHz	256 MB	485
PE7	Alpha 500 MHz	256 MB	518
PE8	Alpha 500 MHz	256 MB	518

表 3 には $U(13,4)$ の逐次探索プログラムの実行時間も示す．なお，計算機は PE1，PE2，$\cdots$，PE8 の順に使用する．

各計算機内の Local Hash 表，計算機間で共有する Global Hash 表には，それぞれ 32 MB の領域を割り当てる．本論文で用いる計算機はそれぞれ処理速度が異なる．そこで，N 個の計算機に対する台数効果を表すために，各計算機の逐次探索の実行時間の調和平均を N で割った値を理想値（目標値）とする．この理想値の式を次に示す．

$$\text{理想値} = \frac{1}{\frac{1}{T_1} + \frac{1}{T_2} + \cdots + \frac{1}{T_N}} \qquad (1)$$

ここでは，最初の 4 台の計算機は同一のものであるので，4 台以下の結果は通常の評価で使われてきた台数効果の理想値と一致する．よく知られているように，調和平均（実行速度の平均の逆数）$\leq$ 相加平均（実行時間の平均）が成り立つことに注意されたい．

3.4 オーバヘッドの定義

次に，並列探索アルゴリズムの性能を評価するためのいくつかのオーバヘッドを定義する．文献 5)，12) ではすべて同一の性能の計算機に対して同様のオーバヘッドを定義している．

Time Overhead
（TO，実行時間オーバヘッド）
理想値に対する N 台の計算機による実行時間の増加の割合を表す．値が負のときは，台数効果以上の性能があることを意味する．

Communication Overhead
（CO，通信オーバヘッド）
通信の際に発生するオーバヘッドである．本論文の構成では，Searcher が結果を Controller に送ってから次の問題を割り当てられるまでの時間と，Searcher が Table Manager に Global Hash 表の検索を依頼してから結果が返ってくるまでの時間

の合計の，理想値に対する割合が CO になる．

Search Overhead
（SO，探索オーバヘッド）

N 台の計算機によって実行したとき，1 台の計算機での探索と比較して過剰に探索した割合を表す．N 台の計算機で実行したときの節点数のほうが多い場合は正，少ない場合は負の値になる．

Synchronization Overhead
（SY，同期オーバヘッド）

Searcher が小問題をすべて解くと，残りの Searcher が小問題を解き終えるまで計算機がアイドル状態になる．このオーバヘッドが SY になる．

各オーバヘッドは次の式で表される．

$$TO = T_N \times \frac{N}{T_1} - 1 = \frac{T_N}{\text{理想値}} - 1 \quad (2)$$

$$CO = \frac{T_{split} + T_{TM}}{T_1/N} = \frac{T_{split} + T_{TM}}{\text{理想値}} \quad (3)$$

$$SO = \frac{S_N}{S_1} - 1 \quad (4)$$

$$SY = TO - CO - SO \quad (5)$$

T_N ： N台の計算機での実行時間
T_{split} ： 小問題生成時間の合計
T_{TM} ： Table Manager への総アクセス時間
S_N ： N台の計算機での探索節点数

処理速度が同じ計算機を使う場合 T_1/N が台数効果となる．本論文では処理速度が異なる計算機を用いるので，理想値を用いて計算する．

4. 実験結果

本章では，まず共有ハッシュ表の有無，アルゴリズムの違いによる性能の変化を示す．次に共有ハッシュ表の管理方法の違いによる性能の変化を示す．最後に並列探索により得られた $U(n,t)$ の解を示す．

4.1 局面表の共有

図 5 に局面表を共有した場合と共有しない場合の $U(13,4)$ の実行時間について，Tree-Splitting と PV-Split の結果を示す．ここでは共有ハッシュ表集中型の管理方法を用いる．

まず，局面表を共有しない場合の結果から，2 つのアルゴリズムの特性を調べる．結果はわずかであるが，PVSplit のほうが良い．ほとんど結果が変わらないのは，節点から出る枝（小問題）の数がチェスプログラムなどと比較して非常に多いからである．チェスの枝の数は通常数十である．$U(13,4)$ の根から出ている枝の数は約 85,000 であり非常に多い．数万の枝のうちの数本の実行時間が変わっても全体の実行時間にはほとんど影響がない．また，計算機がアイドル状態になったときも同様である．

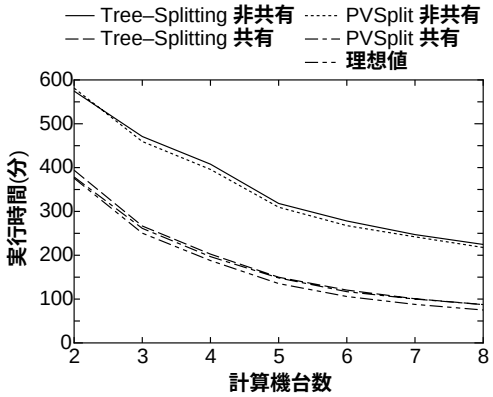

図 5 局面表の共有・非共有の違いによる実行時間の変化
Fig. 5 Comparison of computation time with/without the global transposition table.

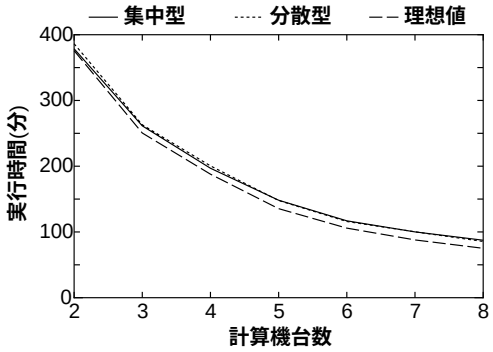

図 6 共有ハッシュ表の管理方法の違いによる実行時間の変化
Fig. 6 Comparison of computation time with the centralized/distributed shared-hashing tables.

以上の結果をふまえて局面表の有無による性能の違いを検証する．どちらのアルゴリズムも局面表を共有しないときは，理想値と比較して 100 分以上の差があるのに対して，局面表を共有したときは，理想値に近い実行時間となっている．局面表の共有の効果が大きいだけでなく，その効果は 2 つのアルゴリズムにほとんど依存していないことが分かる．これらのアルゴリズムのように，木を分割する型の並列ゲーム木探索では，本論文で用いた手法が有効であることが予想できる．

4.2 共有ハッシュ表の管理方法

前章で良い結果を示したアルゴリズムの PVSplit を用いて，共有ハッシュ表の管理方法の違いによる性能の変化を調べる．実行時間を**図 6** に示す．

実行時間はほとんど同じであるが，若干共有ハッシュ表分散型のほうが良い．大きな問題や多くの計算機を

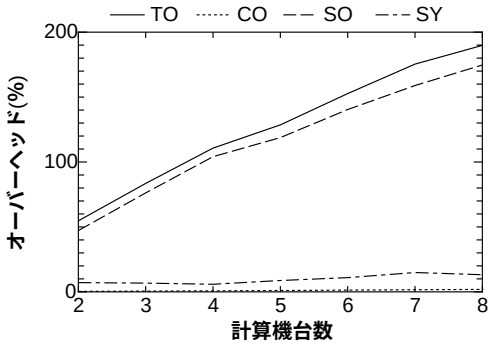

図 7 オーバヘッド（PVSplit・局面表非共有）
Fig. 7 Overheads for PVSplit without the global transposition table.

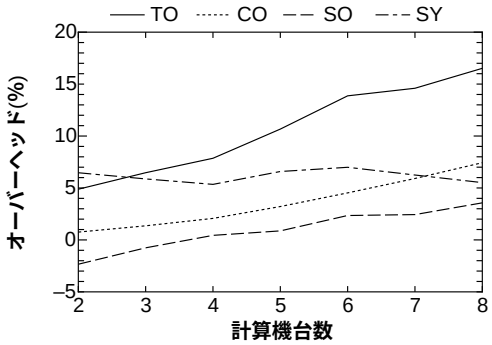

図 8 オーバヘッド（PVSplit・局面表共有）
Fig. 8 Overheads for PVSplit with the global transposition table.

用いた探索は，Table Manager へのアクセス回数が多くなる．そして，複数の Searcher が同時に Table Manager へアクセスする確率は高くなる．これらの場合は共有ハッシュ表分散型のほうがオーバヘッドが小さくなると予想できる．

次に最も結果の良い共有ハッシュ表分散型の PVSplit と，局面表を共有しない PVSplit の結果を，オーバヘッドと速度向上により詳細を検証する．まず，局面表を共有しない PVSplit のオーバヘッドを**図 7** に示す．

オーバヘッドの主な要因は Search Overhead である．Time Overhead は，計算機台数にほぼ比例して増加するのに対して，Search Overhead も同様な増加傾向を示す．これは局面表を共有していないのが原因といえる．局面表を共有していないと，複数の計算機で同一局面を解かなければならない．計算機台数が増加すると，局所的な局面表のヒット率が低くなる．ヒットしないと自ら探索しなければならないので，探索数が増加する．局面表を共有していないので，Communication Overhead は小問題の生成時間の合計だけである．並列選択問題は通常節点から出る枝の数が多いが，その数は葉に近づくにつれて少なくなる．小問題の高さが低くなるほど，通信時間と探索時間に大きな差がなくなるが，枝の数が少ない．小問題の高さが高いものは，探索に非常に時間がかかるだけでなく，その個数も多い．したがって，Controller–Seacher 間の通信時間は，探索時間にほとんど影響しないといってよい．Synchronization Overhead は，計算機台数が増えるにつれて増加傾向を示しているが，Search Overhead ほど大きくなく，あまり影響がないといえる．

次に局面表を共有する PVSplit のオーバヘッドを**図 8** に示す．図 7 と比較すると，Search Overhead は大幅に減少した（縦軸に注意）．Search Overhead の減少により，Time Overhead も図 7 と比較して大きく減少する．特に計算機台数が少ないときは，逐次探索の探索数を下回っている．また，依然 Search Overhead は計算機台数に比例するが，局面表の共有によりその増加の割合は非常に小さくなる．Communication Overhead は計算機台数にほぼ比例する．その原因は Table Manager へのアクセス時間の増加である．図 7 と比較すると，Table Manager のアクセス時間が Communication Overhead に加わった．小問題の割当てにかかる時間は，計算機台数にかかわらず一定である（計算機台数が増えると理想値は小さくなるので実行時間に占める通信時間を表す Communication Overhead は共有局面表の有無にかかわらず増加する）．計算機台数が増加すると，1 台の計算機が探索する節点数は減少し，局所ハッシュ表のヒット率が減少する．その結果，Table Manage へのアクセス回数が増え，Communication Overhead が増加する．また，Communication Overhead は Search Overhead より大きい．しかし，図 7 の計算機が 8 台のときの Search Overhead は 150%以上であるのに対し，図 8 の Communication Overhead は 10%にも満たない．Search Overhead の減少の効果のほうがはるかに大きく，局面表を共有しても通信のオーバヘッドは影響しないといえる．

Synchronization Overhead は 5%程度で増加の傾向はない．ここでの Synchronization Overhead は Time Overhead，Search Overhead，Communication Overhead から計算した値である．したがって，これら以外のオーバヘッドが Synchronization Overhead に存在している可能性がある．また，Search Overhead は探索節点数から計算する．並列探索プログラムの探索部分は，逐次探索プログラムの探索部分の実行時間と同じになるとはいえない．すなわち，並列化の際にオーバヘッドが発生する．また，局面表によっても実

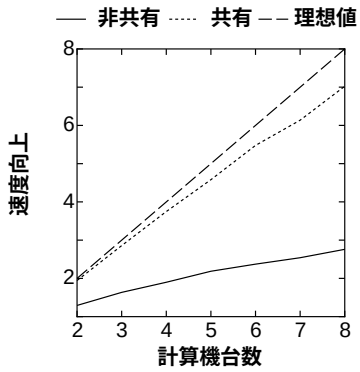

図 9 速度向上
Fig. 9 Speed-ups.

行時間が変わる．したがって，Search Overhead を正確に測定するのは難しい．Synchronization Overhead も同様に難しい．しかし，実験結果によると，局面表を共有することにより，Search Overhead は大幅に減少した．それで，明らかに分散共有ハッシュ法は有効であるといえる．

次に速度向上を**図 9** に示す．局面表を共有しない PVSplit は，計算機が 8 台のとき 3 倍にも満たない結果となった．これに対し，局面表を共有する PVSplit は台数効果に近い結果がえられた．計算機が 8 台のとき約 7 倍（7.02 倍）の速度向上を得た．本実験結果によると，通信速度が遅いにもかかわらず，分散共有ハッシュ法によって十分良い台数効果を実現することに成功したといえる．

4.3 並列探索による解

逐次探索では時間がかかりすぎて解けなかった問題に対して，表 3 の 8 台の計算機により計算を行った．表 2 で下線のついた値が並列探索により得た解である．新たに $U(14,5) = 6$，$U(14,6) = 6$，$U(14,7) = 6$，$U(15,5) = 6$ の計 4 個の値を求めた．

並列探索での $U(14,7)$ の実行時間は約 20 時間である．速度向上を 7 倍とすると，逐次探索の実行時間は約 140 時間であると予想できる．したがって，100 時間以上という大きな時間短縮になる．

5. おわりに

本論文では，分散共有ハッシュ法を用いた並列ゲーム木探索プログラムの評価を行った．

一般に分散計算機環境でデータを共有すると，大きなオーバヘッドが発生してしまう．しかし，本論文では局面表の特徴を生かして，通信速度が遅い環境にもかかわらず少ないオーバヘッドでデータの共有ができることを示した．さらに，実際のゲーム木探索問題において，台数効果に近い速度向上が実現できることを示した．局面表のヒット率が高い問題に対して，分散共有ハッシュ法は有効であることが分かる．

本論文のシステムでは，木の深さが一定値より深いところでは Local Hash 表のみ参照している．局面表を共有する深さを動的に変えることは今後の課題である．また，本論文の実験の範囲内では必要でなかったが，局面表があふれたときの GC（ガーベジコレクション）の実現方法も今後の課題とする．

参考文献

1) Aigner, M.: Parallel Complexity of Sorting Problems, *Journal of Algorithms*, Vol.3, pp.79–88 (1982).
2) Gao, Y. and Yonezawa, A.: A Comparison of Parallel α-β Search Algorithms, 人工知能学会誌，Vol.11, No.2, pp.126–135 (1996).
3) Knuth, D.E.: *The Art of Computer Programming*, Vol.3 (Sorting and Searching), 2nd Edition, Addison-Wesley (1998).
4) Levy, D.N.L. (Ed.): *Computer Games I*, Springer-Verlag (1979).
5) 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).
6) 長島紀子，中山泰一，野下浩平：ゲーム木の並列探索のための分散共有ハッシュ機構の設計と実現，情報処理学会論文誌，Vol.39, No.6, pp.1581–1586 (1998).
7) Nakayama, Y., Akazawa, T. and Noshita, K.: A Parallel Algorithm for Solving Hard Tsume-shogi Problems, *ICCA Journal*, Vol.19, No.2, pp.94–99 (1996).
8) 中山泰一，赤澤忠文，野下浩平：ゲーム木の並列探索のための分散的実行管理機構，電子情報通信学会論文誌，Vol.J79-D-I, No.9, pp.572–575 (1996).
9) Newborn, M.: *Kasparov versus Deep Blue: Computer Chess Comes of Age*, Springer-Verlag (1997).
10) Nitzberg, B. and Lo, V.: Distributed Shared Memory: A Survey of Issues and Algorithms, *Computer*, Vol.24, No.8, pp.52–60 (1991).
11) 野下浩平，佐藤信弘：トーナメントの並列比較回数とゲーム木の探索，ゲーム・プログラミングワークショップ '99 論文集，pp.69–75 (1999).
12) Schaeffer, J.: Distributed Game-Tree Searching, *Journal of Parallel and Distributed Computing*, Vol.6, pp.90–114 (1989).

(平成 12 年 3 月 22 日受付)
(平成 13 年 2 月 1 日採録)

佐藤　信弘（正会員）

1996 年群馬工業高等専門学校電子情報工学科卒業．1998 年同高等専門学校専攻科生産システム工学専攻修了．2000 年電気通信大学大学院情報工学専攻博士前期課程修了．現在,（株）アイザック勤務．スケジューリング関連アプリケーション開発に従事．

新藤　雅也

1997 年群馬工業高等専門学校電子情報工学科卒業．1999 年電気通信大学情報工学科卒業．2001 年同大学大学院情報工学専攻博士前期課程修了．現在，サン・マイクロシステムズ（株）勤務．組合せゲームの理論，分散・並列処理等に興味を持つ．

野下　浩平（正会員）

1943 年生．1966 年東京大学工学部計数工学科卒業．電々公社，東京大学，中央大学等を経て，現在，電気通信大学情報工学科教授，工学博士．アルゴリズムの計算量解析，組合せゲームの理論と実験，卓球に興味を持つ．

中山　泰一（正会員）

1965 年生．1988 年東京大学工学部計数工学科卒業．1993 年同大学大学院工学系研究科情報工学専攻博士課程修了．工学博士．同年，電気通信大学情報工学科助手．現在，同学科助教授．オペレーティング・システム，並列・分散処理，ゲーム・プログラミングに興味を持つ．