コンピュータブリッジの並列処理

小田和 友仁[†] 塙 敏博[†] 上原貴夫[†]

従来からコンピュータによるチェッカーやチェスを強化するために，ゲーム木を分割して複数のプロセッサに割り当てる並列処理が研究されてきた．ゲーム木分割は並列処理による高速化を可能とするが，一方でオーバヘッドの原因となる．本論文ではブリッジのための並列処理を提案する．ブリッジは不完全情報ゲームであり，多くの可能な手札の配置に対してゲーム木探索を行わなければならない．我々の方法では，スケジューラが，1 つの手札配置と候補手のうちの 1 つに対応した部分木を，各プロセッサに割り付ける．この方法により，64 プロセッサで 40 倍から 50 倍のスピード向上が可能になることを実験によって明らかにした．

Computer Bridge Using Parallel Processing

Tomohito Otawa,[†] Toshihiro Hanawa[†] and Takao Uehara[†]

In order to make Checkers or Chess programs strong, various parallel processing methods have been studied, where a game-tree is decomposed and assigned to each processing unit. Althogh the tree decomposition scheme offers the potential for speedups, it also introduces certain overheads. We propose a parallel processing method for a computer Bridge. Since Bridge is incomplete information game, game-tree searches must be done for many possible deals. A possible deal and a sub-tree corresponding to a move from root-node is given to a processor element by a scheduler in our methods. We demonstrate that the speedup is about 40 to 50 for 64 processing units by an evaluation.

1. はじめに

ゲーム研究の主流である完全情報ゲームの分野では，古くから並列処理による高速化の研究が行われてきた．コンピュータチェッカーである Chinook に並列 $\alpha\beta$ 探索の 1 つである PVSplit（Principal Variation Splitting）を採用した 1993 年の研究報告では，16 台で 3.32 台分の効果しかなかった[1)]．非同期な並列探索法である APHID（Asynchronous Parallel Hierarchical Iterative Deeping）を採用した Chinook でも 16 台で 8.35 台分，64 台で 14.35 台分程度である[2)]．チェス専用の並列コンピュータである DeepBlue は 512 個のチェス専用アクセラレータ・チップを持ち，1 秒間に 2 億手を探索するという力わざともいえる探索能力により 1997 年に世界チャンピオンである Garry Kasparov に勝利した．この DeepBlue が，LSI チップの設計ミスのためにハードウェアによるゲーム木の枝刈りをあきらめるかどうか，Kasparov との試合の直前まで悩んだという話からも，ソフトウェアによる高速化アルゴリズムとハードウェアによる並列化を両立させることの難しさが分かる[3)]．

一方でさらに複雑なゲームとして不完全情報ゲームも研究されている．我々は不完全情報ゲームとしてコントラクトブリッジをテーマに選択し，上級者と対等にプレイできるコンピュータブリッジの実現を目標に研究を進めてきた．近年のコンピュータブリッジが採用するカード選択法はモンテカルロ法に基づくアルゴリズムが主流であり，我々のコンピュータブリッジもこれを採用している．現在の局面から得られる情報に矛盾しない手札の仮説を多数生成し，各々の仮説が成立する世界では完全情報ゲームと見なして探索することで，総合的に最善手を決定する方法である．このアルゴリズムをもちいて確率的に十分良さそうな解を得るためには，多くの仮説を生成する必要がある．また，それぞれの仮説についても十分に深く探索しなければならない．よって少ない時間でいかに多くの仮説について探索ができるかが課題となる．しかし不完全情報ゲームにおいて並列処理による高速化が適用された例は報告されていない．

本論文ではこのアルゴリズムに基づき，高い台数効果を得ることができる並列処理の実装法を提案する．

† 東京工科大学
Tokyo University of Technology

2. 完全情報ゲームの並列処理

チェスに代表される完全情報ゲームをコンピュータにプレイさせる研究は，ゲーム研究の主流として人工知能の分野に大きな功績を残してきた．コンピュータによる手の決定法はゲーム木探索が一般的である．ゲーム木探索により状況に応じた良い手を決定するためには，いかに深く探索し，最善手を早く見つけ出せるかが課題となる．そのためには短時間で効率良く多くの探索をしなければならない．完全情報ゲーム研究の中心は主に高速化アルゴリズムに絞られてきた．これまでに $\alpha\beta$ 法や Transposition Table をはじめとし多くの高速化アルゴリズムが研究されてきた．これらはゲーム木において無駄な探索を省く手法であり，高速化には限界がある．

さらなる高速化の手法としてゲーム木を分割して複数のコンピュータで並列に探索する方法が研究されている．ゲーム木を分割すると分割された部分木の間に処理の順序関係が生じる．また従来の高速化アルゴリズムを活かすためにはメモリ共有か通信によるデータ共有を必要とする．これらの問題はオーバヘッドを引き起こし，並列効果を上げるうえでの障害となる．

2.1 PC クラスタによる並列処理

近年ではコンピュータの普及が進み，高性能な汎用コンピュータと高速なネットワーク環境が安価で手に入るようになった．これら汎用ハードウェアによる並列環境の構成は低コストではあるが，ソフトウェアを工夫する必要がある．汎用コンピュータで構成する並列環境は基本的に分散メモリ環境であり，コンピュータ間のデータ共有をメッセージパッシングにより実現しなくてはならない．データ共有が多いほど通信頻度が増し，通信時間がかさんでしまう．データ共有を効率良く行う，あるいはデータ共有をなるべく回避する工夫が必要である．

汎用ハードウェアによる並列処理としては 1993 年にコンピュータチェッカーである Chinook に並列 $\alpha\beta$ 探索の 1 つである PVSplit（Principal Variation Splitting）を適用したとの研究報告がなされている[1]．PVSplit では n 回目の反復深化探索において $n-1$ 回目に見つかった最善手をたどるすべての局面に続く部分木を複数のコンピュータで分担し探索する．$\alpha\beta$ 法を活かすためには最善手をたどる各局面において同期をとる必要がある．また，それぞれの部分木は処理時間が不均一である．これらがオーバヘッドの原因となり，PVSplit を実装した Chinook では 16 台で 3.32 台分程度の効果しか得られなかった．

1997 年には非同期な並列探索法である APHID（Asynchronous Parallel Hierarchical Iterative Deepening）を採用した Chinook の報告もなされた[2]．APHID では 1 つのマスタが一定の深さ（d'）まで $\alpha\beta$ 探索し，その葉を複数のスレイブで分担し反復深化探索する．スレイブが非同期に探索を行うことでつねに仕事が割り当てられているため，アイドルな CPU は存在しない．しかしマスタがスレイブの結果を得るたびに木全体に反映するため d' までを再探索するというオーバヘッドが存在する．APHID を実装した Chinook では 16 台で 8.35 台分，64 台で 14.35 台分程度と台数効果が伸び悩んだ．

2.2 専用ハードウェアによる並列処理

並列処理用の専用ハードウェアを設計するうえで，従来の高速化アルゴリズムをそのまま回路にすることは困難である．そこでハードウェア構成に特化した独自の設計が必要となる．そのような設計は汎用性を著しく低下させるため多大なコストがかかる．またソフトウェアも専用ハードウェアを制御するために特化したアルゴリズムを必要とする．専用ハードウェアとソフトウェアによる高速化アルゴリズムの両立は困難な課題といえる．

専用ハードウェアによる並列処理としては IBM の開発したチェス専用の並列コンピュータである Deep Blue がある．Deep Blue は 1997 年には当時の世界チャンピオンである Garry Kasparov に勝利した．Deep Blue のベースである並列コンピュータ RS/6000 SP は 32 のノードを持つ．1 ノードは 8 個のチェス用アクセラレータ・チップを搭載するアクセラレータ・ボードを 2 枚持つ．すなわち，512 個のアクセラレータ・チップにより並列に処理する．1 つのノードがマスタとなり，他ノードをスレイブとしてコントロールする．n 手先までを探索するとして，$0 < n'' < n' < n$ とすると，最初の n'' 手をマスタノードが探索し，n''〜n' 手をスレイブノードで並列に探索，そして n'〜n 手までをそれぞれのスレイブノードが持つアクセラレータ・チップで探索する．アクセラレータ・チップは与えられた木に対して単純に $\alpha\beta$ 探索を行い，葉では静的評価を行うことで，ハードウェアのシステム構成を単純化している[4]．この DeepBlue が，LSI チップの設計ミスのためにハードウェアによるゲーム木の枝刈りをあきらめるかどうか，Kasparov との試合の直前まで悩んだという話からも，ソフトウェアによる高速化アルゴリズムとハードウェアによる並列化を両立させることの難しさが分かる[3]．

3. コントラクトブリッジ

コントラクトブリッジ（以下ブリッジ）は欧米で広く親しまれているカードゲームである．4 人がテーブルの四方に座り，対面のプレイヤ同士がペアを組む．各プレイヤは 13 枚ずつ配られたカードを他人に見えないように持つ．4 人が順番に 1 枚ずつ出したカードの強弱で勝者を決める．この 1 手順をトリックと呼ぶ．13 トリックを繰り返し，ディクレアラと呼ばれる人が勝利すると宣言したトリック数を勝つことができるかで点数が決まる．切札と勝つべきトリック数の宣言であるコントラクトは，プレイの前に行われるオークションで決定する．オークションとは文字どおりの競り合いであり，この競り合いにおける駆け引きにより勝利条件の難易度や勝利時の点数が変化する．

4. コンピュータブリッジ

我々はブリッジの上級者と対等にプレイできるコンピュータブリッジの実現をめざし研究を続けてきた．ブリッジはパートナシップを必要とするゲームであるため，コンピュータブリッジは人間のパートナとして対等に行動できる必要がある．そこで我々はブリッジのプレイヤをモデル化し，エージェントとしてコンピュータブリッジに組み込んだ．エージェントの実装には制約論理プログラミング言語である ECL^{i}PS^{e} [5] をもちいた．コンピュータブリッジはオークション部とプレイ部で構成される．

4.1 オークション部

オークション部はプレイヤの代わりにビッドを決定するエージェントとして設計した．エージェントはビッドの経過を観察し，知識ベースとして用意されたビッドに関する約束をもとに各プレイヤのハンドに関して推論を行い，その結果を制約条件として表現する．この制約条件を参考にし行動基準に従い自律的にビッドを決定する．行動基準は「味方と協調して利益を最大にし，敵と競合して損失を最小にする」とした[6]．

4.2 プ レ イ 部

プレイ部はオークション部のエージェントを基盤として設計した．エージェントはビッドとプレイの経過を観察し，ルールやビッドに関する約束，経験則からなる知識ベースをもとに各プレイヤのハンドに関して推論を行い[7]，その結果を制約条件として表現する．この制約条件に矛盾しないディール（配られた手札）を多数生成する．各ディールに対しゲーム木探索し，モンテカルロ法の原理にのっとり次の一手を決定する．

本研究ではプレイ部を高速化の対象とする．

4.3 モンテカルロ法に基づくアルゴリズム

コンピュータブリッジにおけるカード選択法として何人かの研究者によりモンテカルロ法に基づくアルゴリズムが提案された[8]．カード候補の集合を M としたとき，次のようにして最善手を決定する．

[Step1] それまでのビッドおよびプレイと矛盾しないようにカードをくばり，ディールの集合 D を作る．

[Step2] 各ディール $d \in D$ ごとに，各候補 $m \in M$ を選択した場合どのような結果になるかをダブルダミーで評価し，スコア $s(m,d)$ を計算する．

[Step3] $\sum_d s(m,d)$ が最大となるような行動 m を選ぶ．

本研究ではディールの仮説を含む局面情報をワールドと呼ぶ．このアルゴリズムによるコンピュータブリッジのモデルは，**図 1** のようにいくつかのワールドと，ワールドごとに 1 つのゲーム木を持つ形をとる．

4.4 ワールド生成数

4.3 節で述べたアルゴリズムをもちいて妥当な解を得るためには，ある程度多くのワールドを生成し深く探索する必要がある．生成すべきワールド数は考慮すべきワールドが生成する確率に依存する．

図 2 はセーフティプレイの例題である[9]．West が ♡K と ♡A で 2 トリックを勝利し，♡8 を出した．♡Q, ♡9 と続き South は何を出すべきか．切り札である ♠2 を出してトリックを勝ち，♠3 を出すのが正解である．続く West の ♠6 に対して South が ♠9 で勝利し，West の切り札を狩ることができる．

4.3 節で述べたアルゴリズムによりこのセーフティプレイを発見するには East の ♠ が 0 枚のワールドが生成される必要がある．我々のコンピュータブリッジに実験的に 100 個のワールドを生成させたところ

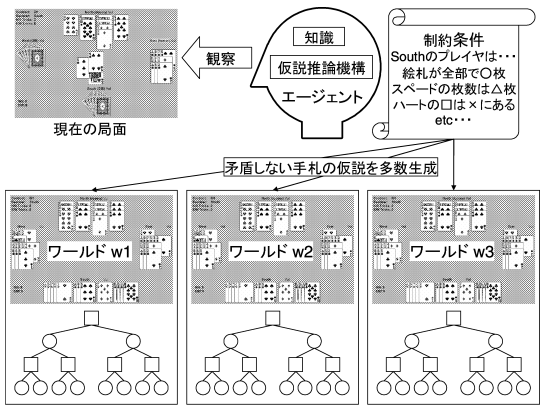
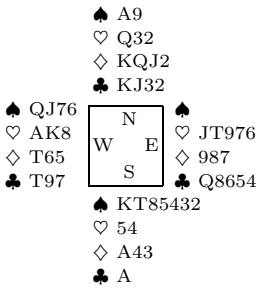


図 1 コンピュータブリッジのモデル
Fig. 1 Model of the Computer Bridge.



Dealer: West
Bidding:

West	North	East	South
Pass	1NT	Pass	4♠
Pass	Pass	Pass	

Playing:

West	North	East	South
♡K	♡2	♡6	♡4
♡A	♡3	♡7	♡5
♡8	♡Q	♡9	?

図 2 セーフティプレイの例題
Fig. 2 Example for safety play.

4 個のワールドが該当した．また 36 個連続して該当ワールドが生成されない場合があった．

いくつかの実例から判断し，50 個程度のワールドを生成し探索することを目指して本研究を実施した．

5. コンピュータブリッジの並列処理

4.3 節で述べたモデルにおける各ワールドのゲーム木探索は MinMax 法を基本としており，完全情報ゲームの研究で培われた従来の高速化アルゴリズムが適用可能である．我々は $\alpha\beta$ 法と Partition Search を各ワールドのゲーム木探索に適用した．しかしコンピュータ 1 台の処理速度には限界があり，リアルタイムに人間と対戦するコンピュータブリッジの実現は難しい．そこで並列処理によりさらなる高速化を図った．

ワールドをタスクの単位とすると，完全情報ゲームとして探索するためのデータが独立しているためメモリ共有を抑えられる．しかし粒度が粗いため負荷が偏りやすい．そこでゲーム木を分割しタスクとする必要がある．2 章で述べたとおり，ゲーム木を分割すると様々なオーバヘッドが生じる危険性がある．そこでメモリ共有を抑えるため，カード選択アルゴリズムにおける $\alpha\beta$ 法の適用範囲に着目しゲーム木を分割した．

5.1 $\alpha\beta$ 法の適用範囲

4.3 節のアルゴリズムの［Step2］における候補ごとの評価値 $s(m,d)$ は必ず結果を返す必要がある．**図 3** のようにワールドにおけるゲーム木全体を $\alpha\beta$ 法の適用範囲とすると，ルートの子ノードでカットが起きたときに評価値が確定しない場合がある．たとえば図 3 の場合，4 以下の値になることが分かった時点でルートにおいて選ばれることはないのでそれ以降の探索をカットする．しかし，このときルートの子ノードは 4 以下の評価値であることしか分からず確定しない．そこで我々は**図 4** のようにルートの子ノードを頂点とする個々の部分木をそれぞれ $\alpha\beta$ 法の適用範囲とした．分割された部分木は探索のためのデータが独立しており非同期に探索できるため，タスク並列性が高い．我々はこの部分木をタスクの単位として採用した．

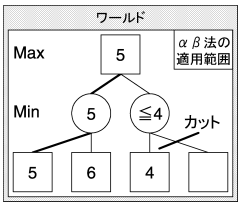


図 3 ゲーム木全体への $\alpha\beta$ 法の適用
Fig. 3 Application of alpha-beta cutoff to a game-tree.

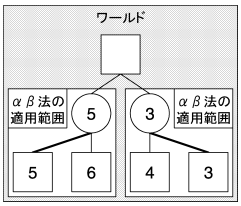


図 4 ルートに連なる部分木への $\alpha\beta$ 法の適用
Fig. 4 Application of alpha-beta cutoff to each sub-tree from the root.

5.2 Partition Search

Partition Search は，Transposition Table を応用した高速化技法である．厳密に一致しなくても同様と見なせる局面をまとめて Transposition Table に保持することで，参照による探索の回避頻度を向上させる．

4.3 節のアルゴリズムでは，1 つの Transposition Table の適用可能範囲は，同じワールドに対するゲーム木全体となる．本研究では実装の簡単化とオーバヘッドを考慮して，図 4 の $\alpha\beta$ 法の適用範囲と Transposition Table の適用範囲を一致させた．

6. 実　　装

システムの構成を**図 5** に示す．システムは 3 つのプロセスで構成される．マスタプロセス，スレイブプロセス，そしてワールド生成プロセスである．マスタプロセスはゲームの進行を担当し，人間との CUI を持つ単一のプロセスである．タスク生成とタスク分配，結果集計を行い，ブリッジエージェントとして振る舞う．スレイブプロセスはタスクの処理に専念するプロセスであり，複数が並列に動作する．ワールド生成プロセスはマスタの要求に従いワールドを生成する．

タスクの生成はワールドの生成を含む．ワールドの生成には 4 章で述べたとおり制約プログラミングの手法をもちいる．そこで制約論理プログラミング言語である ECL$^{\text{i}}$PS$^{\text{e}}$ により実装した．ECL$^{\text{i}}$PS$^{\text{e}}$ はインタプリタ言語であり，コンパイラ言語である C 言語に比べると処理速度は劣る．しかし制約プログラミング

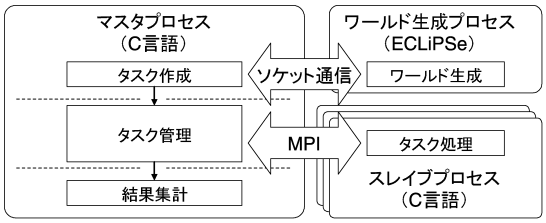


図 5 システム構成
Fig. 5 System architecture.

の特性を活かして無作為にディールを生成するだけのワールド生成部は，ECL^{i}PS^{e} によるプログラムでも十分に高速である．100 個程度のワールド生成ならば 0.5 秒以内ですべてを生成し終えることができる．マスタプロセスは ECL^{i}PS^{e} のワールド生成プロセスとソケット通信で連携をとる．

タスクの管理にはタスクプールをもちいた動的スケジューリングを実装した．並列処理を実装するためにはクラスタの管理や，クラスタ間のメッセージ通信が必要である．これらを実装するために MPI ライブラリである LAM をもちいた．

6.1 MPI ライブラリ

MPI（Message Passing Interface）は分散メモリ環境における並列プログラミングのためのライブラリの仕様である．MPI はメッセージパッシングのための関数群の仕様を定めている．

MPI の仕様に準拠した実装ライブラリに LAM（Local Area Multicomputer）がある[10]．LAM は UNIX ベースの OS のみをサポートし，Windows 系列の OS では動作しない．UNIX のデーモンとして常駐し，これを介して通信するため高速な通信が可能である．LAM は並列処理の環境をサポートするために，プログラムを配布してのいっせい起動や，各クラスタの監視，デバッグの機能などを有する．

6.2 SPMD

MPI の提供するプログラミングモデルは，各ノードで異なるプログラムを実行する MPMD（Multi Program Multi Data stream）ではなく，SPMD（Single Program Multi Data stream）である．すなわち全ノードが 1 つのプログラムを読み込み実行する．ただし各ノードに割り当てられたランクと呼ばれる識別 ID により処理を分岐することは可能である．

我々はマスタとスレイブのプロセスに分けるためにランクを利用した．ランクが 0 のクラスタをマスタ，それ以外のクラスタをスレイブとする．

6.3 マスタプロセス

マスタプロセスは 4.3 節のアルゴリズムにおける［Step1］と［Step3］を担当する．**図 6** のフローチャートに沿って流れを説明する．

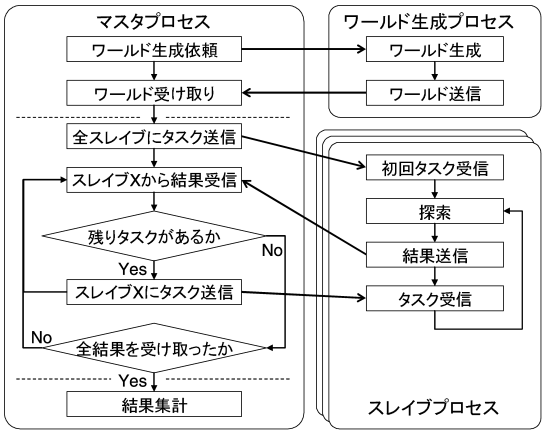


図 6 カード選択のフローチャート
Fig. 6 Flowchart of the card selection.

はじめに 4.3 節の［Step1］に従いタスク生成を行う．局面の情報と生成すべきワールドの個数をワールド生成プロセスに送信し，ワールド生成を依頼する．ワールドの受け取り待ちに移り，ワールド生成プロセスが送信してきた結果を受け取る．結果を受け取ると，手札から出しうるカードの候補を発生し，ワールドと候補の全組合せをタスクとする．ワールド生成数を N_W，候補数を N_C とするとタスク数 N_T は次の式で求められる．

$$N_T = N_W \times N_C \tag{1}$$

続いてタスク管理に移る．まず全スレイブに対して 1 つずつタスクを送信する．処理が終了し，結果を送信してきたスレイブに対して，タスクが残っているのであれば次のタスクを渡す．すべてのタスクを送り出し，その結果が帰ってくるまでこれを繰り返す．

スレイブから受け取る結果は候補のカード情報と，そのスコア $s(m,d)$ とする．全タスクの結果がそろったところで，4.3 節の［Step3］に従い集計に移る．結果として受け取ったカードの候補ごとにスコアの総和 $\sum_d s(m,d)$ を計算し，最大となるようなカードの候補 m を最善手として決定する．

6.4 ワールド生成プロセス

ワールド生成プロセスはマスタからワールド生成のための情報を受け取る．この情報には自分とダミーの手札情報や，それまでのプレイの経過などが含まれる．これらの情報やルール，経験則といった知識をもとに制約条件をまとめあげる．この制約条件に矛盾しないよう手札をランダムに配置し要求された個数のワールドを生成する．生成したワールドをまとめてマスタに

送信し，マスタから次のカードのプレイに関する探索の依頼があるまでの待ち状態へと戻る．

6.5 スレイブプロセス

スレイブプロセスは 4.3 節のアルゴリズムの［Step2］を担当する．マスタから送られてきたタスクのワールドをもとにプレイの経過からすでに出されたカードを取り除き，現在の局面まで更新する．さらにタスクに付随する候補のカードをプレイした場合のスコア $s(m,d)$ を完全情報ゲームとして探索する．得られたスコアと候補のカード情報をマスタに送信する．その後は再度マスタからのタスク受け取り待ちのループに入る．

7. 評価実験

100 Mbps の LAN で接続された PC をもちいて評価を行った．すべての PC は**表 1** の A に示したスペックを持ち，LAM の実行環境が整っている．1 台をマスタとし，64 台までをスレイブとしてもちいる．LAM の機能によりマスタから各クラスタにおけるプロセスの起動を制御することができる．またワールド生成プロセスの実行用に表 1 の B に示したスペックを持つ 1 台の PC を用意した．

4.4 節で述べたとおり，我々は 50 個程度のワールド生成を目標としている．そこでワールド生成数は 64 とした．スレイブを 1，2，4，8，16，32，64 台で動かしたときの処理時間をそれぞれ測定した．

7.1 初手の処理時間と台数効果

プレイを開始して一番最初に出すカード選択にかかる処理時間を**表 2** に示す．ここでの深度は先読みの手番数を意味する．深度が深くなるにつれ，処理時間が指数関数的に増加することが確認できる．またスレイブ数を増加させることにより処理時間が減少することも確認できる．深度 16 でスレイブ数を 64 とした場合 19.53 秒程度と人間が待てないほどの時間ではない．深度 20 でスレイブ数を 64 とした場合には 152.74 秒と遅い．しかし測定した処理時間は，静的評価関数の実装法，検索の候補をしぼる前向き枝刈りによって短縮できる可能性を残しており，リアルタイムのブリッジを実現するうえで許容できる時間である．特にオープニングリードと呼ばれる初手には経験則に基づく約束があり，候補は上記実験の半分に絞ることができる．

処理時間から割り出した台数効果のグラフを**図 7** に示す．台数効果はスレイブ 1 台での処理時間を各スレイブ台数での処理時間で割った値である．どの深度もスレイブ数 1 から 16 まではほぼ理想の台数効果を示している．スレイブ数 32 では深度が深いほど台数効果が高い傾向を示す．これは並列化できない逐次処理が並列効果のボトルネックとなるというアムダール則の顕れと思われる．図 5 において並列化できる部分はタスク管理とタスク処理の部分であり，それ以外は逐次処理部である．測定したところ逐次処理部には平均で 0.5 秒程度の時間がかかっており，深度が少ない設定では大きく影響したと思われる．逐次処理部において最も処理時間がかかるのは制約プログラミングの手法によるワールド生成部である．しかし 0.5 秒程度ならば体感的には十分短いと考え，本研究では高速化の対象から除外した．

逐次処理部の影響を省いた台数効果を見るため，並列処理部のみの処理時間を測定し，台数効果を割り出した．結果を**図 8** に示す．どの深度もスレイブ数 1 から 32 まではほぼ理想の台数効果を示している．スレイブ数 64 では深度が浅いほど台数効果が高い傾向を示す．深度が深いほどタスクごとの処理時間のばらつ

表 1 スペック
Table 1 Specifications.

項目	A	B
CPU	Pentium4 2.0 GHz	Pentium4 3.40 GHz
RAM	512 MB	1,024 MB
OS	FreeBSD 5.0	Vine Linux 3.1
言語	gcc 3.2.1	ECL^{i}PS^{e}5.8
Library	LAM/MPI 6.5.9	

表 2 初手の処理時間（sec）
Table 2 Processing time of the first card selection.

スレイブ数	深度 12	深度 16	深度 20
1	107.07	944.01	6,986.40
2	54.05	472.78	3,495.97
4	27.27	236.75	1,749.04
8	13.88	119.05	879.30
16	7.19	60.90	442.35
32	3.97	32.64	230.71
64	2.37	19.53	152.74

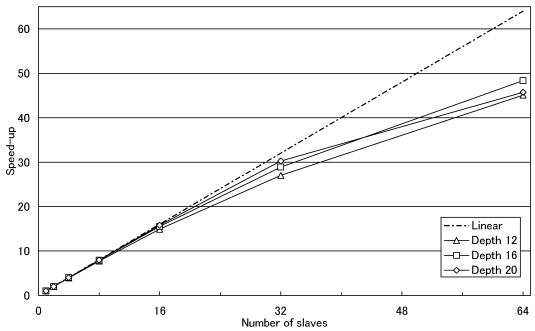


図 7 初手の台数効果
Fig. 7 SpeedUp of the first card selection.

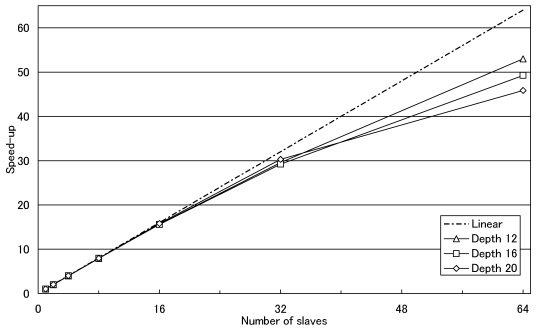


図 8 初手における並列処理部の台数効果
Fig. 8 SpeedUp of the first card selection in parallel processing section.

きが大きくなり，動的スケジューリングによるロードバランスの効果が薄くなる．特にスレイブ数 64 では 1 スレイブあたりのタスク数が少ないのでタスクごとの処理時間のばらつきが大きく影響する．

7.2 タスクごとの処理時間のばらつき

タスクごとの処理時間を測定した．**表 3** に結果を示す．深度 20 では最大 21 秒のタスクもあり大きなばらつきが見受けられた．深度 16 においてもほとんどが 0〜1 秒台に収まるものの，最大で 5 秒程度と多少のばらつきはあった．深度 12 では全タスクが 0〜1 秒台であり，ばらつきはほとんど見られない．コンピュータブリッジでは探索深度が深いほどタスクごとの処理時間のばらつきが大きい．これが台数効果を減少させる原因になっていることが分かる．

7.3 ゲーム進行と処理時間

ブリッジではゲームの進行により探索条件が大きく変動する．それにともない処理時間や台数効果も変動する．ゲーム進行とともに変化する台数効果を調べるため，我々のコンピュータブリッジに 4 人分をプレイさせ，各プレイヤの手番における処理時間を測定した．ワールド生成数は 64，深度は 20 とした．ゲーム進行にともなう処理時間の変化を**図 9** に示す．またゲーム進行にともなう候補数の変化を**図 10** に示す．

トリックの最初にカードを出すプレイヤは手札の全カードが候補となる．以降のプレイヤは基本的に最初に出されたカードのスートを出さなければならないため候補が絞られる．本研究では連続するカードを 1 つの候補として扱うなど，候補を絞り込むアルゴリズムを組み込んではあるが，それでも最初のプレイヤの選択はそれ以降に比べ処理時間がかかる傾向にある．

スレイブ数を増加させると処理時間が減少するが，ゲーム進行にともなう処理時間の変化は大体同じ傾向を示す．4，8，11 枚目や最終トリックの 4 枚など，どのスレイブ数の設定においても 0.5 秒程度に収まる手番がある．これは候補数が 1 であり探索処理を省いたためである．

序盤ではカードの候補数が多いため，処理時間がかかる．また出されたカード枚数が少なく確定しない情報が多いため，タスクごとの処理時間のばらつきによるオーバヘッドも大きいと考えられる．出されたカード枚数が増加するにつれて各プレイヤの手札枚数が減

表 3 探索時間の区分ごとのタスク数
Table 3 Number of tasks in each range of search time.

処理時間の区分（s）	深度 12	深度 16	深度 20
0〜1	512	459	159
1〜2	0	43	138
2〜3	0	6	95
3〜4	0	3	43
4〜5	0	1	25
5〜6	0	0	19
6〜7	0	0	10
7〜8	0	0	5
8〜9	0	0	4
9〜10	0	0	3
10〜	0	0	11

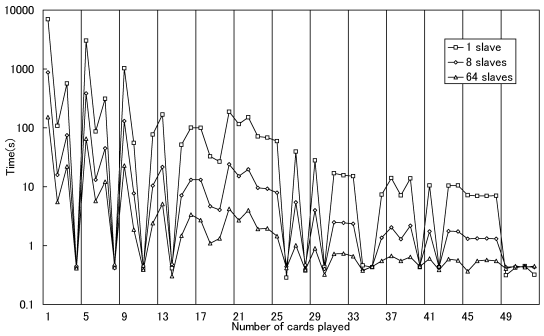


図 9 ゲーム進行にともなう処理時間の変化
Fig. 9 Transition of processing time according to game progress.

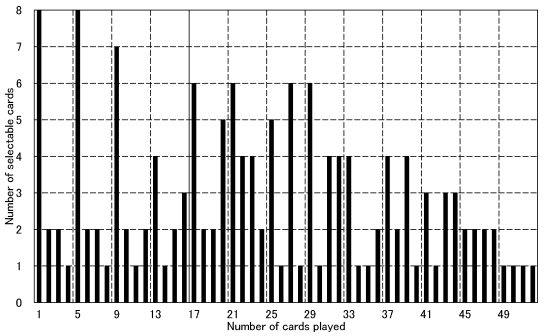


図 10 ゲーム進行にともなう候補数の変化
Fig. 10 Transition of selectable cards according to game progress.

少するため，出しうるカードの候補数が減少し処理時間も短くなる．後半戦は人間とリアルタイムに対戦するうえで現実的な処理時間に抑えることができた．

8. おわりに

コンピュータブリッジにおいて一般的に採用されるモンテカルロ法に基づくカード選択アルゴリズムに即した形で適応しうる並列処理の実装法を述べ，実装したコンピュータブリッジによる実験について述べた．最も処理時間がかかる初手においてスレイブ数 32 台まではほぼ理想の台数効果を得た．64 台では深度が深いほど台数効果が低い傾向を示したが，それでも 45〜49 程度と高い台数効果を示した．深い探索ではタスクごとの処理時間のばらつきが大きく，動的スケジューリングによるロードバランスの効果が薄いことが確認できた．ゲーム序盤は候補数が多いため長考が多いが，ゲーム進行とともに処理時間は減少する．これらの結果は静的評価関数の最適化や候補の絞り込みにより高速化できる可能性も含むため，人間とリアルタイムに対戦するうえで許容できる範囲内である．

コンピュータブリッジにおいて並列処理の効果が確認できたので，より複雑なアルゴリズム[11]の実行が可能となり，またリアルタイム対戦実験の環境が確保された．これは今後の研究にとって大きな成果といえる．

この実装法はブリッジに限らずカードゲームに代表される不完全情報ゲームの解法として広く応用がきく．

参考文献

1) Lu, C.-P. P.: Parallel Search os Narrow Game Trees, Master's thesis, Department of computer science University of Alberta Edmonton (1993).
2) Brockington, M.G. and Schaeffer, J.: APHID: Asynchronous Parallel Game-Tree Search, Ph.D. thesis, Department of Computer Science, University of Alberta Edmonton, Alberta T6G 2H1 Canada (1999).
3) Hsu, F.-H.: *Behind Deep Blue*, Princeton University Press (2002).
4) Hsu, F.-H.（著），吉村信弘（訳）：チェスマシン Deep Blue から学んだ教訓，共立出版 bit 別冊ゲームプログラミング，pp.26–43 (1997).
5) IC-Parc: The ECLiPSe Constraint Logic Programming System. http://www.icparc.ic.ac.uk/eclipse/
6) 安藤剛寿，小林紀之，上原貴夫：コンピュータブリッジのビッドにおける協調と競合，電子情報通信学会論文誌，Vol.J83-D-I, No.7, pp.759–769 (2000).
7) 小林紀之，小田和友仁，上原貴夫：コンピュータブリッジによるカウントシグナル，情報処理学会論文誌，Vol.45, No.6, pp.1704–1714 (2004).
8) Ginsberg, M.L.: GIB: Steps toward an expert-level bridge-playing program, *IJCAI-99* (1999).
9) ビクター・モロー，ニコ・ガードナー（著），難波田愈（訳）：カードプレイテクニック，日本コントラクトブリッジ連盟 (1998).
10) Trustees of Indiana University: LAM/MPI Parallel Computing. http://www.lam-mpi.org
11) 小田和友仁，上原貴夫：コンピュータブリッジにおける新しい探索アルゴリズムと高速化，情報処理学会ゲーム情報学研究会，*The 9th Game Programming Workshop 2004*, pp.64–70 (2004).

(平成 17 年 9 月 2 日受付)
(平成 18 年 2 月 1 日採録)

小田和友仁（正会員）

1979 年生．2002 年東京工科大学工学部情報工学科卒業，同大学大学院博士課程進学．2005 年現在，博士後期課程 2 年目にあたる．ゲームプログラミングを題材に並列処理の研究に従事．

塙　敏博（正会員）

1993 年慶應義塾大学理工学部電気工学科卒業．1998 年同大学大学院理工学研究科博士課程修了．東京工科大学工学部情報工学科講師を経て，2003 年より同大学コンピュータサイエンス学部講師．並列計算機アーキテクチャ，並列処理の研究に従事．

上原　貴夫（正会員）

1942 年生．1965 年早稲田大学理工学部電気通信学科卒業．1970 年同大学大学院博士課程修了．工学博士．同年（株）富士通研究所入社．1994 年東京工科大学工学部情報工学科教授，2004 年同大学コンピュータサイエンス学部教授．現在に至る．その間，1977 年より 1 年間 Stanford 大学客員研究員．CAD，人工知能，ゲームプログラミング等の研究に従事．人工知能学会，電子情報通信学会，日本ソフトウェア科学会，日本コントラクトブリッジ連盟会員．