将棋プログラム用ライブラリ libshogi

藤井宏行[†]　本田優樹[†]　高田浩生[‡]

[†]香川高等専門学校詫間キャンパス 電子システム工学科 藤井研究室 〒769-1192 香川県三豊市詫間町香田551

[‡]ティーソフトウェア　〒763-0095 香川県丸亀市垂水町 310-6

１．はじめに

将棋プログラムは人工知能について学習する上で良い教材となります．香川高専詫間キャンパス藤井研究室（以下，藤井研）では将棋を素材に，学生が探索などのプログラミング手法について学習しています．この学習において問題となったのが，将棋の対局をプログラムで表現することの難しさでした．ソースコードが公開されている数多くの著名な将棋プログラム，ライブラリはどれも英知の結晶であり，丹念に読むことにより素晴らしい知見を得ることができます．一方でそれらは，人を超えることを目標に磨き抜かれてきたプログラムでもあります．文法のみの知識で簡単に理解できるものではありません．

藤井研では，**C++** の基本的な文法を知っている学生が短期間で並列探索を行うプログラムを書き始めるところまで到達できることを目標に，学習用の将棋ライブラリの開発を行っています．

現在は，基本的なデータ構造の実装が完了し，ルール通りに指すための将棋機能について実装を進めています．

このライブラリには **libshogi** という名前を付けました．作者らが十分な品質に至ったと判断した時点でソースコードを公開する予定です．

２．概要

図-1 に将棋プログラムの階層モデルを示します．これは，将棋プログラムがこのような階層構造を持つべきであるという意味ではなく，このような実装も可能であろうという例です．

最も重要なのは，探索，定跡，学習などの戦略を司る部分です．ライブラリは，プログラマが大きな制約を感じることなく，戦略部分のプログラミングに集中できるよう考慮されたものであるべきです．

libshogi は網掛けされた機能を提供します．**Foudation** 層では，アトミック操作，ロック，スレッド，リスト，二分木，ハッシュ木などの基本データ構造と操作，Game 層では，ルール通り指すための将棋機能，外部プログラムとの通信機能などを提供します．

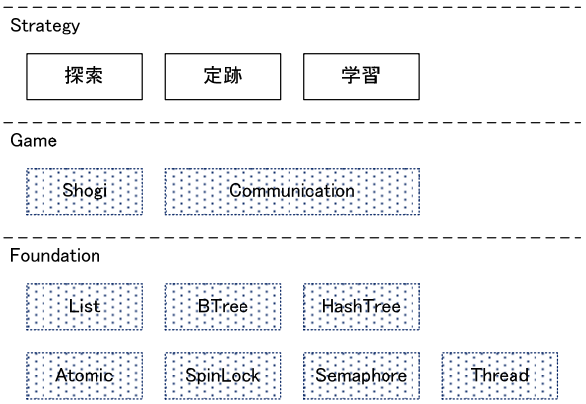

図-1　将棋プログラムの階層モデル

libshogi は **Boost** や **STL** のコンテナクラスなどを使用せずプリミティブなライブラリ API のみで実装しています．また，教材を前提とするので，速度より簡明さを優先しています（**Foundation** 層の機能はコメントを含め合計 2000 行程度で実装されています）

３．Foundation 層の機能

Foundation 層では将棋プログラムを実現するためのプリミティブな機能を提供します．

(1) Atomic

アトミック操作を行うための変数と操作のクラスです．**GCC** のビルトイン関数を使用して実装しています．インクリメント，デクリメント，テストアンドスワップなどの機能を提供します．

並列性を高めるためには共有リソースのロック粒度をできるだけ小さくする必要があります．アトミック操作で対応可能な処理はそれを使用することによりロックを回避することができます．

(2) SpinLock

相互排他機能を提供するクラスです．**glibc の NPTL** スピンロック関数を使用しています．山下宏さんの記事において紹介されていたハッシュ表のロックと同等の機能を提供します．[1]

スピンロックは OS のスケジューラに依存しない

ため軽量です．インテルプロセッサの場合，**Hyper Threading** で並列実行できる数だけスレッドを生成し，それぞれがイベント待ちなどで眠ることなく走り続けた場合に良い並列性能を期待することができます．

(3) Semaphore
スレッド間で同期をとるためのクラスです．**glibc** の **POSIX** セマフォを使用しています．

(4) Thread
glibc の **NPTL** を使用したスレッドクラスです．スレッドの生成，スレッドの終了確認，スレッド合流，結果の保持，リソースのクリーンアップ機能を提供します．

(5) List
連結リストのコンテナクラスです．要素の追加，削除，イテレータ機能を提供します．

(6) BTree
二分木のコンテナクラスです．ノードの追加，探索機能を提供します．マルチスレッドプログラミングにより木全体をロックすることなく並列探索することができます．

(7) HashTree
局面のハッシュ値をキーとして局面情報を検索するための二分木コンテナクラスです（いわゆるハッシュ木とは少し違います）キー値によるノードの追加，探索，値の更新機能を提供します．マルチスレッドプログラミングにより木全体をロックすることなく並列探索することができます．

木を平衡に保つためのロジックは含まれていません．局面ハッシュ値をキーとすることを想定すると，そのランダム性から，木が大きく偏ることはないと考えられます．

4．Game 層の機能

Game 層の機能は開発中です．今大会では合法手生成などの機能に池泰弘さんが開発された「れさぴょん」を使用させていただく予定です．[2]

(1) Shogi
合法手の生成，着手，棋譜の読み込みなどの将棋に特化した機能を提供するクラス群です．

(2) Communication
外部プログラム，フロントエンド **GUI** との通信を行うための機能を提供するクラスです．

5．開発環境

表-1 に開発環境を示します．大会へ出場する時に使用するマシンとはハードウェア構成が異なります．本文書中の性能値は表-1の環境において測定したものです．

表-1　開発環境

項目	許容範囲
CPU	Intel(R) Core (TM) i7-2600K 3.4 GHz 物理コア 4（スレッド 8）
メモリ	32 GB (DDR3-1333 Non-ECC)
OS	CentOS 6.4 (x64)
カーネル	linux 2.6.32 (358.18.1.el6)
コンパイラ	GCC 4.4.7 (4.e16)
標準ライブラリ	glibc 2.12 (1.132.e16)

6．二分木の並列処理性能と信頼性評価

HashTree の土台となる二分木（**BTree**）の並列処理性能と安定性について評価を行いました．図-2 は改良版メルセンヌツイスタ[3]により **16,000,000** 個の **64** ビット無符号整数乱数を生成した後，それを二分木に追加した時のスレッド数と処理時間の計測結果です（計測に乱数生成時間は含まれていません）

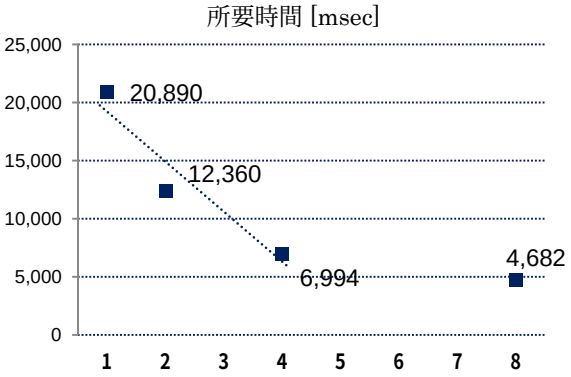

図-2 二分木並列処理性能（ノード追加）

物理コア数が 4 なのでスレッド数 4 までは，直線的に処理時間が短くなっています．これより並列処理が機能していることが分かります．8 スレッドへ増やした場合も若干の性能向上が見られました．

16,000,000 ノードを追加した後の木の高さは，62 でした（理想的な平衡二分木では $\log_2 n = 24$ 程度）

信頼性評価は，8 スレッド時に OS から見た全ての **CPU** 使用率が 100% になること，全てのノードにおいて親子関係が健全であること，追加したノード数が一致することを確認しました．また，処理性能試験とは別に **Mudflap** によりメモリリーク及び不正なメモリアクセスが無いことも確認しています．

７．HashTree による探索結果の共有

並列探索では探索結果をスレッド間で共有する必要があります．**HashTree** は探索結果の共有機能を提供します．**HashTree<T,V>** は二つの型仮引数を持つクラステンプレートです．**T** は探索結果を検索する局面ハッシュ値であり，通常 64 ビット無符号整数となります．**V** は探索結果の型です．これはプログラムの実装によって異なります．例えば，評価値や局面データのポインタなどが考えられます．いずれにしても内部でコピーしているためサイズの大きいデータ型は適しません．表-2 は **HashTree<T,V>** のインターフェースを説明のために簡略表示したものです．

表-2　**HashTree** のインターフェース

HashTree<T,V>	
bool	**add** (T *key*)
bool	**set** (T *key*, const V &*val*)
bool	**get** (T *key*, V &*val*)

add() は局面の追加を試みます．合法手を作成した後，その手を着手することにより新しい局面が作られます．**add(***新しい局面のハッシュ値***)** を実行すると，未探索局面であれば真が返ります．

すでに探索が行われている局面であれば偽が返ります．この場合，局面の評価は他のスレッドが非同期に行うことになります．**get()** は局面の評価が定まるまでスレッドをブロックします．ブロックされたスレッドは局面を追加したスレッドが評価を set() するまでスリープします．

８．デッドロック

並列探索は **HashTree** をスレッド間で共有し，局面を探索することにより行います．既知のアルゴリズムを素直に実装するとデッドロックの問題を抱えることがあります．実用的な将棋プログラムでは対策が必要となりますが，それは学習者が独自の方法を考える部分であるので **Foundation** 層には含めていません．必要に応じてサンプルプログラムなどで対策などを例示できればと考えています．

９．今後の目標

libshogi が目標とするのは第一に分かり易いということです．将棋プログラムに興味を持った学生が，すぐに **Strategy** 層から開発を開始する助けになればというのが作者らの願いです．

一方で実用性についても磨きをかけていければと考えています．**libshogi** を土台として少しでも強い将棋プログラムが開発できるよう改良を続けていければと思います．

参考文献

1) 山下 宏："YSS -『コンピュータ将棋の進歩 2 』以降の改良点"，『アマトップクラスに迫る - コンピュータ将棋の進歩 5』，1 章，共立出版，2005.

2) 池泰弘：『コンピュータ将棋のアルゴリズム』，工学社，2005.

3) SIMD-oriented Fast Mersenne Twister (SFMT) http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT