Xeon Phi によるモンテカルロ囲碁プログラムの高速化

丸山真佐夫[†1]　八田拓也[†2]

概要：本報告では，インテル Xeon Phi を用いたモンテカルロ囲碁の高速化について述べる．Xeon Phi は，60 コア 240 スレッドのメニーコア構成と 512 ビットのベクトル演算器を特徴とするコプロセッサであり，約 1TFlops の理論性能を有する．しかしシングルスレッドでのスカラ性能は通常の CPU の数分の 1 にとどまる．われわれは，ビットボードによる碁盤ルーチンに 512 ビットベクトル演算を適用することで，プレイアウトの高速化を実現した．Xeon Phi 5110P を用いた評価実験では，シングルスレッドで 2450 プレイアウト/s，240 スレッドでは約 24 万プレイアウト/s という結果が得られた．さらに，単なる並列プレイアウトではなく，Xeon Phi 上で動作するツリー並列化によるモンテカルロ木探索プログラムを実装した．このプログラムは毎秒 22 万回の探索を行う．

キーワード：囲碁，モンテカルロ木探索，インテル Xeon Phi

Accelerating Monte-Carlo Go by Xeon Phi Coprocessor

MASAO MARUYAMA[†1]　TAKUYA HATTA[†2]

Abstract: In this paper we describe implementation and evaluation of our fast Monte-Carlo Go program for Intel Xeon Phi Coprocessor. A Xeon Phi consists of 60 cores with 4 hardware threads and a 512-bit vector processing unit per core. It has the theoretical performance of 1 TFlops, but on the other hand, it has very poor single thread performance. Use of the vector operations is therefore required to achieve good performance. We developed a bit-board based Go program optimized for Xeon Phi's vector operations. Our program performs 2.45k playouts/s with single thread, and 240k playouts/s with 240 threads. We implemented not only the playout routine but also a Monte-Carlo tree search (MTCS) program based on tree parallelization method. It evaluates 220k games per second.

Keywords: Go, Monte-Calro tree search, Intel Xeon Phi Coprocessor

1.　はじめに

モンテカルロ囲碁プログラムにおいて，シミュレーション回数の増加は棋力を向上させる重要な要素の一つである．一方，近年のハイパフォーマンスコンピューティング分野では，GPGPU など，計算処理に特化したアクセラレータによって計算能力を高める技術が利用されている．われわれは，そうした計算アクセラレータの一種である Intel 社の Xeon Phi コプロセッサをモンテカルロ囲碁プログラムに適用し，処理性能の向上を目指している．

本報告では Xeon Phi 向けのモンテカルロ囲碁プログラムの実装と評価について述べる．

2.　Xeon Phi コプロセッサ

Xeon Phi[1]は 60 コア，240 ハードウェアスレッド（われわれが利用した Xeon Phi 5110P の場合）のメニーコアアーキテクチャプロセッサである．1TFlops を超える理論性能を有するが，これは各コアが備える 512 ビット幅のベクトル演算命令によって達成されたものである．Xeon Phi のコアは汎用 CPU と比較すると単純であり，また動作クロック周波数が 1GHz 程度にとどまるため，ベクトル演算以外の通常の命令での性能は高くない．

囲碁は Xeon Phi に実行させるプログラムとしては，大規模な行列計算などとは大きく異なる特徴を持つ．すなわち，長いベクトルの計算は含まず，条件分岐等の多い複雑なプログラムである．このようなプログラムに対して Xeon Phi が有効であるかどうかを評価することは興味深い．

3.　Xeon Phi における囲碁プログラムの実装

本節では，Xeon Phi による囲碁のモンテカルロ木探索プログラムの実装について述べる．このプログラムの実行時間の多くが，プレイアウトに費やされる．さらにプレイアウト処理の大半は，ルールに基づく局面の更新である．

3.1　碁盤の表現

上述のとおり Xeon Phi は 60 コア（240 スレッド）のメニーコアアーキテクチャと 512 ビットのベクトル演算命令によって高い演算性能を実現する．一方，コア単体のスカラ演算性能は低く，現行の Xeon プロセッサと比較すると数分の 1 から 10 分の 1 程度である．したがって，Xeon Phi を用いて高い性能を達成するためには，マルチスレッド等による並列化とともにベクトル演算の利用が重要になる．本実装では，ベクトル演算を適用しやすい碁盤表現としてビットボードを採用した．

Xeon Phi のベクトル命令は，16 個の 32 ビットまたは 8

†1 木更津工業高等専門学校
National Institute of Technology, Kisarazu College
†2 木更津工業高等専門学校，現（株）TID
National Institute of Technology, Kisarazu College
Presently with TID Limited

ⓒ2016 Information Processing Society of Japan

個の 64 ビットの整数を 1 命令で処理することができる. 19 路の碁盤のベクトルデータに格納する方法として,

(a) 32 ビット整数に碁盤 1 行を割り当てる
(b) 64 ビット整数に碁盤 3 行を割り当てる

の二つを検討した.

(a)では, 512 ビットのベクトルデータが二つ必要になり, 演算も 2 回に分かれる. (b)は, 512 ビットベクトルデータ一つに 19 路盤が収まる一方, 碁盤を上下左右にシフトする演算が (a)と比較して複雑になるのが欠点である. 開発初期に両方の碁盤表現を実装して性能を測定したところ, 処理スピードは同程度であった. データサイズが小さいことは, 今後パタンデータを導入するときに有利になると考えられる.

さらに, 現行の Xeon Phi (Knights Corner) のベクトル命令セット (IMCI) は, 64 ビットのシフト演算をサポートしないため, 複数の 32 ビットシフト演算等を組み合わせて同等の処理を実現している. これに対して第 2 世代以降の Xeon Phi (Knights Landing) のベクトル命令セット (AVX-512) は 64 ビットシフト命令を持つので, 将来的に (b) の碁盤シフトの処理スピードは現在より改善されると予想される.

これらのことから, われわれは 64 ビット整数に碁盤 3 行を割り当てる (b) の 64 ビット整数に碁盤 3 行を割り当てるデータ表現を採用した.

(1) ビットボードによる碁盤表現

図 1 にビットボードの表現を示す. 64 ビット整数に碁盤 3 行分を格納し, 行間には 1 ビットのスペースを挟む. 19 路盤を格納するためには, 7 個の 64 ビット整数が必要である. Xeon Phi のベクトルデータは 8 個の 64 ビット整数を持つことができるので, 1 ベクトルで碁盤を表現可能である. このビットボードを黒石, 白石, 空点について保持する.

(2) 連の表現

上下左右に連結した同色石の集合である連は, 捕獲の単位になる. すなわち, 着手によって駄目数 (その連に隣接する空点の数) が 0 になった連は捕獲され, 盤上から取り除かれる. 本実装では, 盤上の連一つにつき 1 個のビットボードを持つ. 連ボードを用いると, 連の除去は, 石ボードと連ボードの XOR 演算で実現できる.

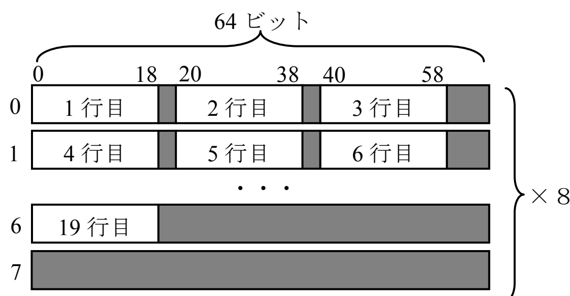

図 1 512 ビットベクトルによるビットボードの表現

連ボードは連に属する石集合を保持するが, 逆に石の属する連を参照するのには適さない. 石 (点) から連を検索ために, 各点の属する連番号を格納した 2 次元配列を持つ. 3.2.2 節で述べるように, この連番号の更新にもベクトル演算を利用している.

3.2 ビットボード操作の実装

大規模な数値計算では, 一般的にきわめて大きな配列データを扱うため, 配列データをメモリから連続的にロードして, ベクトル演算器で演算し, 結果を配列等にストアするというのが典型的な処理の流れである. それに対して, 本システムが扱うビットボードは 64 ビット整数が 8 個 (実際に使うのは 7 個) という, きわめてベクトル長の短いデータである. そのため数値計算と同様のスタイルのプログラムでは, メモリとベクトルレジスタ間のロード, ストアのコストがベクトル演算による速度向上を打ち消してしまうと予想される.

一方, Xeon Phi には 32 個のベクトルレジスタが搭載されている. この豊富なレジスタを用いることで, 必要なデータをすべてベクトルレジスタ上に置いたまま計算できる可能性がある.

以下にビットボード操作の実装について述べる.

3.2.1 コーディングスタイル

コンパイラがビットボードデータをベクトルレジスタに保持するコードを生成することを確実にするため, 本実装では次のようなコーディングスタイルを採用した.

(a) 低レベルな組込み関数の利用

ベクトル演算はインテル C/C++ コンパイラが提供する組込み関数を用いて行い, ビットボードの表現として配列ではなく「__m512i」型 (ベクトル組込み関数の引数や戻り値のベクトルデータを保持するための, 特殊な型) を利用する. 実際にはビットボード型を「__m512i」と配列の共用体として定義しており, プログラムからは配列表現でもアクセスできる.

(b) マクロによる記述

引数, 戻り値の受け渡し時にメモリに書き戻されるのを防ぐため, 基本的なビットボード操作は関数ではなくマクロで記述する.

インテルコンパイラが出力するコードを調査したところ, インライン関数を用いてもビットボードの受け渡しにスタックを利用していたことから, マクロを利用することにした. ただし, この問題はビットボードの型の定義方法の変更によっても解決できた可能性がある.

3.2.2 ビットボード操作

本実装では, 基本的なビットボードの操作はすべてベクトル演算を用いて行っている. 代表的な操作の実装について述べる.

(1) 点のセット

ビットボード b 上の点(x, y)を 1 にセットするコードは,

ⓒ2016 Information Processing Society of Japan

図 2のように実装した．ここで,「__m512i」は 512 ビットのデータを 16 個の 32 ビット整数のベクトルとして保持する型,「__mmask16」は演算の適用を制御するベクトルマスク型である．また「_mm512」で始まる関数はベクトル演算の組込み関数である．

6 行目で，セットしたいビットだけを 1 にしたビットボード t を生成する．組込み関数「_mm512_set1_epi32()」は 16 個の 32 ビット整数すべてに同じ値をセットするため, 7 行目で設定したマスク k により，セット対象の要素についてだけ t と b の 論理和をとっている（8 行目）．それ以外のベクトル要素はマスクされ, 元のボードの値を維持する．

図 2 で利用例を示したように Xeon Phi のベクトル演算命令は，AVX-2 など従来のインテルの命令セットにはなかったベクトルマスク機能を持っている. ベクトルマスクは，ビットボード演算の高速化に有効であった．点セット処理をベクトルマスクなしで書くとすると，セットしたい 1 ビットだけを立てたビットボードの配列を作ってから，それをベクトルレジスタにロードすることになる．

ベクトルマスクの効果を調べるため，セット処理の方法による処理速度の違いを測定した．結果を表 1 に示す．ベクトルマスクを利用するプログラムとしないプログラム，そしてベクトル演算を用いず配列上でスカラ命令を使って操作するプログラムについて，10 万プレイアウトの計算時間を測定した．マスク演算を用いる場合と比較して，用いない場合は 1.1%，配列に展開する場合は 8.7%実行時間が増加した．

点のセットは，1 手の着手につき 1 回だけ実行される処理である. それにもかかわらず, マスク演算によって約 1%の性能差が生ずることから，マスク演算の有効性が確認できた．さらに，配列に展開するプログラムは 9% 近い速度低下が見られる．ビットボードをベクトルのまま扱うことの重要性が示された．

なお 1 プレイアウトの平均手数を 450 手と仮定すると，点セット処理 1 回あたりの実行時間の増加はマスク無で 9.64ns，配列版では 78.8ns である．測定に用いた Xeon Phi 5110P のクロック周波数は 1.05GHz なので，それぞれ 10 クロック，80 クロックサイクル程度に相当する．

```
1:set(_m512i b,int x,int y)
2:{
3:  __m512i t;
4:  __mmask16  k;
5:
6:  t = _mm512_set1_epi32(ビットパタン(x,y));
7:  k = _mm512_int2mask(ベクトルマスク(x,y));
8:  b = _mm512_mask_or_epi32(b,k,b,t);
9:}
```

図 2　ビットボード上の点のセット処理

表 1　点セット処理の時間測定

	実行時間[s]	相対値
マスク有	40.63	1.000
マスク無	41.07	1.011
配列	44.22	1.087

(2)　連の捕獲

囲碁のルールに従い，着手によって駄目数が 0 になった連は石ビットボードから除去される．連ごとのビットボードを持っているため，連を石ボードから除去するのは簡単である．しかし除去すべきかどうかの判定には，連の駄目数を調べなくてはならない．

本実装では，駄目数を連の属性値として常に保持するかわりに，必要になった時点で計算する．ただし捕獲判定においては，駄目数の正確なカウントをするのではなく，0 であるか 1 以上かだけを区別する．

図 3 に連の駄目数の判定処理を示す. 空点ボードを上下左右に 1 マス膨張したボードと連ボードの ビット単位の論理積演算によって 1 になるビットを m に格納する（1 行目）．m とゼロボード（全ビットが 0 のボード）を 32 ビット単位で比較し，相違する要素に対応するベクトルマスク m のビットを 1 にする（3 行目）．もし m の全ビットが 0 であれば，連の駄目数は 0 であることが分かる（4 行目）．

正確な駄目数を数える場合は，連ボードの方を膨張させて空点ボードと論理積をとり，連駄目ボード中のセットされているビット数をカウントすればよい．しかし，ビットカウントを行うベクトル命令は存在しないため，ベクトルデータを配列に戻してからスカラ命令でビット数を数えなくてはならない．また，対象となる連それぞれに対して膨張処理が必要になるため，1 手あたりの膨張処理の回数が最大 4 回になる．

(3)　石の所属連情報の更新

ビットボードである連ボードは，各石の所属する連の検索には向かないため，これとは別に点の所属連を格納する整数配列（所属連ボード）を持つ．

連ボードベクトルの 1 要素（ 64 ビット）に対して，所

```
1:連駄目ボード = _mm512_and_epi32(
                 連ボード, 膨張空点ボード);
2:ゼロボード = _mm512_setzero_epi32();
3:m = _mm512_cmpneq_epi32_mask(
          連駄目ボード,ゼロボード);
4:捕獲対象連 = (_mm512_mask2int(m) == 0);
```

図 3　連の駄目数の判定

ⓒ2016 Information Processing Society of Japan

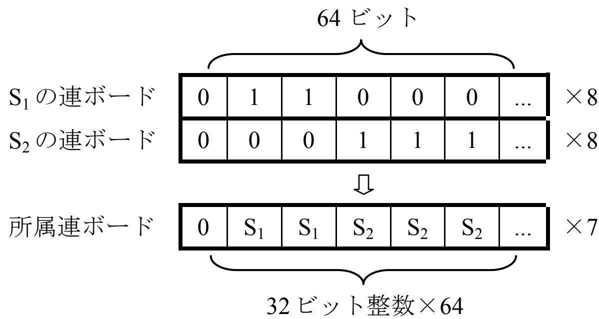

図 4　石の所属連の表現

属連ボードは要素数 64 個の 32 ビット整数からなる配列を対応させる．たとえばある局面で二つの連 S_1，S_2 が存在するとき，所属連ボードには図 4 に示すように，それぞれの連ボード上で 1 であるビットに対応する配列要素に連の番号が格納される．

着手によって連の除去，結合が発生したときは，所属連ボードを更新する．この処理にも，ベクトルマスク機能を利用している．Xeon Phi にはベクトルマスクのビット値にもとづいて，二つのベクトルのどちらかの要素を選択して一つのベクトルを作る命令があり，これを利用した（組込関数「_mm512_mask_mov_epi32()」）．連ボードのビットパタンをベクトルマスクとして利用することで，所属連ボードを効率的に更新できる．

(4)　眼の認識

プレイアウト処理中に生きている自分の眼を埋めないようにするために，眼の認識が必要である．われわれは，空点の近傍の石配置によって認識できる眼を論理式で表現し，ビットボードのベクトル論理演算命令を用いて実装した．

たとえば図 5 に示すような，周囲 8 点の石配置だけから単独で眼になると判定されるパタンは，A_0〜A_3 に同色石があり，かつ B_0〜B_3 に 3 個以上同色石があるという条件で表せる（盤端は除く）ので，これを論理式にすれば，

$$A_1A_2A_3A_4((B_1+B_2)B_3B_4+(B_3+B_4)B_1B_2)$$

となる．1 点あたりの計算コストは大きいが，ベクトル演算を用いることで，盤上のすべての点の眼認識を並列に行うことができる．

眼認識処理における演算回数を表 2 に示す．論理演算は，単一のベクトル命令で実行できる．現状でビットボードの

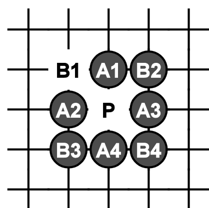

図 5　単独で眼になる例

表 2　眼認識におけるベクトル演算の回数

演算種類	演算回数
論理積	20
論理和	16
排他的論理和	5
横シフト	6
縦・斜めシフト	6

シフトを行う演算は, 方向によって 8〜11 命令を要する[a]．次世代のAVX-512 命令セットでサポートされる 64 ビット単位ベクトルシフトを利用すれば，横方向はすべて 1 命令でシフトでき，また縦および斜め方向のシフトは現在より 4 命令程度ずつ短縮される．

3.3　プレイアウト

現在のプレイアウトプログラムは，RAVE[2]などのプレイアウトの質を高める処理を含まない，シンプルなランダムシミュレーションを行う実装になっている．ただし，捕獲によって空いた領域を確実な地と認識した場合はそこに着手しないなど，終局までの手数を減らすための若干の処理を追加している．それ以外の合法手がなくなるまで，ゲームを続ける．

図 6 にプレイアウトによる終局図の例を示す．この図のようにある程度の地を残した盤面状態で終局する．

3.4　モンテカルロ木探索の実装

前節で述べたプレイアウトルーチンをベースに，UCT によるモンテカルロ木探索を行うプログラムを実装した．

Xeon Phi には，独立したプロセッサとしてそれ自身で完全なプログラム全体を実行する「ネイティブ実行」と，ホストプロセッサの制御下で一部の処理を行う「オフロード実行」の二つの動作方法がある．

Xeon Phi はシングルスレッド性能が低いため，並列化できない実行区間が長い場合は，オフロード実行が適している場合がある．しかしモンテカルロ木探索は実行時間の大半を並列性の高いプレイアウトが占めるので，本実装では「ネイティブ実行」を採用した．

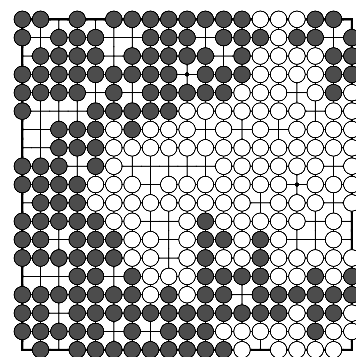

図 6　終局時の盤面例

a) 左シフトは 64 ビットのベクトル加算で代用できるため，1 命令で済む．

ネイティブ実行モードにおける Xeon Phi は，ユーザプログラムから見ると，コア／スレッドが多いという特徴を持つものの，論理的には一般的な CPU である．したがって，単にプレイアウトを並列計算するだけでなく，共有メモリにおける一般的な並列化手法を用いて，並列モンテカルロ木探索プログラムの全体を Xeon Phi 上に実装できる．この点は GPU を囲碁プログラムに用いるのと比較した場合の Xeon Phi の大きな利点になり得ると考える．

本実装では, 並列化の手段として OpenMP を用いている．

さてモンテカルロ木探索の並列化として，主に次の 3 種類の手法が提案されている[3].

(a) リーフ並列化　リーフノードでプレイアウトを並列実行する
(b) ルート並列化　別々の木を探索し，結果を統合する
(c) ツリー並列化　一つの探索木を並列に探索する

Xeon Phi はいずれの手法を実装する場合にも特に制約はないが，(c)のツリー並列化は，スレッド間の木操作の競合のため，多数のスレッドで実行した場合のスケーラビリティが課題になる．ツリー並列化で十分なパフォーマンスが得られるならば，他の 2 手法ではより高いスケーラビリティを実現できることは明白である．

一方，棋力の向上に関しては，探索木を深くすることができるツリー並列化が最も効果的であると予想される．

これらのことから，今回われわれはツリー並列探索を実装した．ツリー並列探索においては，複数スレッドによる木操作の競合の扱いが問題になる．Xeon Phi は一般的な CPU（10 コア程度）と比較してスレッド数が 1 桁多いため，競合にかかわるオーバヘッドの低減が特に重要である．

探索木の実装にハッシュテーブルなどは用いず，木構造をそのまま保持する．またリーフノードで子ノードを展開する際，すべての合法手のための領域を確保する．この方法はデータ使用量が増えるかわりに，ロック回数が少なくて済む利点がある．メモリ使用量に関しては，本研究で利用した Xeon Phi 5110P は，一般的な PC の主記憶容量に匹敵する 8GB のメモリを備えているので, ノード展開までのプレイアウト数をやや大きくすることで十分対応できる．現在は 10 回プレイアウトしたノードを展開している．

本実装では，次の 3 箇所の木操作で競合が発生する．

(1) **ノードの生成**

一定回数プレイアウトを行ったリーフノードについて，子ノードを展開する処理である．この局面における全合法手のノードの領域を配列として動的に確保し，領域のポインタを書き込む．

このポインタの書込み操作が，スレッド間で競合するため，OpenMP のロック機構を用いてアクセスを制御する．

Enzenberger らはツリー並列木探索のロックを省く手法を提案した[4]．本システムでノード生成部分に適用してテストしたところ，約 3%のスピードアップの効果が確認できたが，第 4 節の性能測定のプログラムでは利用していない．

(2) **ノードの訪問回数の更新**

ノード訪問回数は探索木を下りる際にインクリメントする．複数のスレッドが同じノードを選択したとき，書込みの競合が発生するため, OpenMP のアトミック構文を利用して，複数スレッドによるインクリメントを直列化した．

(3) **ノードの勝数の更新**

ノードの勝ち数は，そのノードの手番から見た勝ち数である．プレイアウト実行後，探索木を上ってくる際に，ノード手番が勝っていればインクリメントする．訪問回数と同じく，アトミック構文を用いてインクリメントした．

なお下りがけに訪問回数を更新する際，勝ち数は変化しない. そのため, 勝ち数を更新する前に勝率を計算すれば，この回のプレイアウトが負けであった場合と同じ値になる．これによって，一つのノードへのスレッドの集中を防ぐ Virtual Loss の効果を実現している．

4. 性能評価

4.1 予備実験

まず予備的な実験として，ベクトル演算を用いない別実装のプログラム[5]における Xeon Phi のプレイアウト速度を測定した．このプログラムは，碁盤上の各点の状態（空／白／黒）を 1 語の整数で表した碁盤サイズの配列を持つ，一般的な碁盤表現を用いている．

本実験の測定環境を表 3 に示す. 測定では 10 万回（Xeon Phi の 8 スレッド以下は 1 万回）のプレイアウトの実行時間を 10 回計測し，平均をとった．

表 4 の実験結果が示すようにシングルスレッド実行では, Xeon Phi はホスト CPU に対して 11%のプレイアウト速度だった．これは Xeon Phi のシングルスレッド性能の低さから予想されたとおりの結果である．

ホスト CPU では 32 スレッド，Xeon Phi では 120 スレッドのときが最も高性能であった．Xeon Phi ではシングルスレッドと比較して 78 倍のプレイアウトスピードが得られたが，ホスト CPU（2 基の Xeon E5-2680）に対しては，約 68%の性能にとどまった．

以上の予備実験から，Xeon Phi を囲碁プログラムで有効に利用するためには，ベクトル演算への最適化が必要であることが確認できた．

Mirsoleimani らは Feugo を Xeon Phi 上で動作させ，Xeon プロセッサと比較している[6]．それによると 8 スレッドの Xeon E5 2695 と 240 スレッドの Xeon Phi 7210（1.238GHz）のスピードが同程度であった. 本実験と同様に CPU 向けのプログラムを Xeon Phi 向けリコンパイルしただけでは，ホスト CPU より性能が低いという結果になっている．

表 3　測定環境

	ホスト CPU Xeon E5-2680×2	コプロセッサ Xeon Phi 5110P
クロック周波数	2.7GHz	1.05GHz
コア／スレッド	16 / 32	60 / 240
主記憶	128GB	8GB
OS 等	CentOS 6.4	MPSS 3.4
コンパイラ	インテル C/C++ 15.0.3.187	
最適化	-O3　-ipo	

表 4　予備実験プログラムのプレイアウト速度[k po/s]

スレッド数	CPU（相対性能）	Xeon Phi（相対）
1	6.68 (1.00)	0.77 (0.11)
2	12.80 (1.92)	1.52 (0.23)
4	21.35 (3.20)	3.04 (0.46)
8	41.81 (6.26)	5.99 (0.90)
16	65.96 (9.87)	12.24 (1.83)
32	**88.34 (13.22)**	24.08 (3.60)
60		43.90 (6.57)
120		**59.83 (8.96)**
180		59.67 (8.93)
240		57.59 (8.62)

4.2 プレイアウト性能の測定

実装システムの初手からのプレイアウト速度を測定した．測定環境と方法は予備実験と同じである．

測定結果を表 5 に示す．1 スレッド実行時に 2450 po/s, 240 スレッド実行では約 24 万 po/s という良好な性能が達成できた．まったく異なる実装であるため単純な比較はできないが， 4.1 節のプログラムと比較するとシングルスレッド実行で 3 倍，最大性能で 4 倍を超える．ホスト CPU と比較しても，約 35% であり，アーキテクチャとクロック周波数の差を考慮すると高い性能であると考える．

並列実行では 240 スレッドのときに最大性能が得られ，1 スレッドに対して約 100 倍のプレイアウト速度を達成した．各スレッドが 2 クロックに 1 回しか命令を発行できないため，マルチスレッド性能の目標はシングルスレッドの 120 倍である．本プログラムはその 83%を達成している．

予備実験のプログラムと比較すると，スレッド数に対する性能向上率に違いがあることが分かる（図 7）．
文献[7]は，スレッド数とコアあたりの性能について，スレッド数を増加させることでデータアクセスのレイテンシを隠蔽し性能を向上させる効果があると述べている．本実装では，プレイアウト中に必要なデータの多くがレジスタ上に置かれていると想定している．そうであるとすると， レイテンシの主原因は演算間の依存関係にあると推測されるが，より詳しい解析が必要である．

表 5　本システムのプレイアウト性能

スレッド数	PO 速度[k po/s]	相対性能
1	2.45	1.0
60	127.9	52.2
120	193.4	78.9
180	212.9	86.9
240	**243.3**	**99.3**

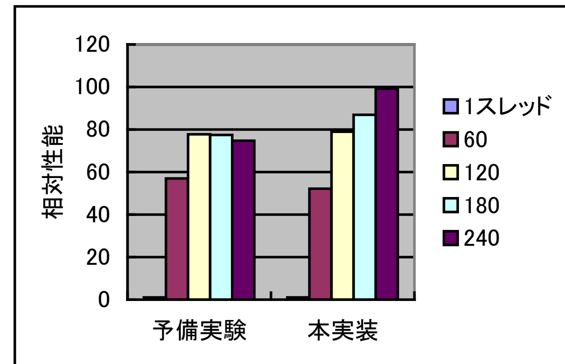

図 7　スレッド数に対する性能向上の比較

4.3 モンテカルロ木探索の性能測定

次にモンテカルロ木探索の性能測定を行った．1 手あたり 1 秒で対局を行い，木探索中に行われる探索回数（ルートノードの訪問回数）を測定した．表 6 にゲーム進行 100 手ごとの探索回数を示す．ここでも 240 スレッドでの性能が最も高く, 3 手目[b]では 1 スレッドの 100.5 倍であった．

表 6　思考 1 秒の対局における探索数[k po/s]

手数	スレッド数				
	1	60	120	180	240
3	2.20	122.6	185.6	208.7	**221.1**
100	2.58	144.8	220.6	246.1	**263.5**
200	3.27	197.2	303.2	336.4	**363.8**
300	5.60	322.0	506.6	581.9	**623.7**

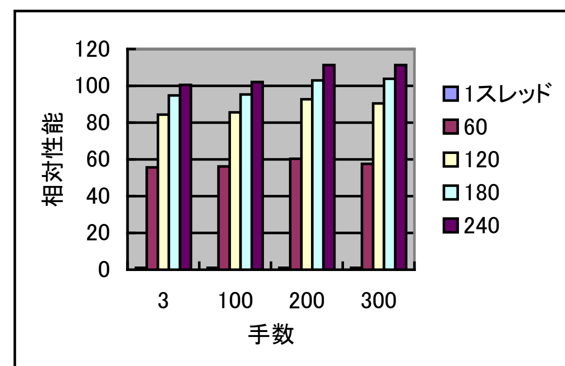

図 8　手数ごとの探索の相対性能

b) スレッド数が多いとき，2 手目までは 10%以上遅いため除外した．

ⓒ2016 Information Processing Society of Japan

表 7　3 手目の探索回数とプレイアウト数の比較

スレッド数	探索回数 / PO 数
1	0.900
60	0.959
120	0.960
180	0.980
240	0.909

局面が進むにつれて探索回数が増えている．これは終局までの手数が減少するためであるが，一方で探索木を操作する時間の比率が高くなることで，ロック等のオーバヘッドが増加する可能性もある．しかし 1 スレッド実行に対する相対性能（図 8）を見ると，むしろ手数が進むほどマルチスレッド実行時の性能が高くなった．

モンテカルロ木探索における探索回数は，木の操作が追加されるので，単純なプレイアウトよりも減少する．さらに，木操作ではベクトル演算を利用していないため，Xeon Phi の木操作コストは CPU と比較して相対的に大きい可能性がある．3 手目の探索回数を単純な初手からのプレイアウトと比較すると，240 スレッド実行時が最も低く 90.9%になった（表 7）．240 スレッドという多数のスレッドが探索木を共有しているプログラムとしては，良好な結果であると考える．

また表 7 から，モンテカルロ木探索全体に占めるプレイアウト実行時間は 9 割程度，逆に木の操作処理が 1 割程度を占めているであると推測される．

5. 関連研究

Mirsoleimani らは Feugo を Xeon Phi 上で実行し，性能を評価した[6]．このプログラムは 9 路盤対局の 2 手目で 1 秒あたり約 4 万ゲームの探索を行う．9 路盤と 19 路盤の差，利用した Xeon Phi の違い（われわれのシステムよりクロック周波数が 1.2 倍高い）を考慮すると，われわれのシステム上で 19 路盤の対局を行わせた場合には，1 万ゲーム/s 程度のスピードになると思われる．Feugo のプレイアウトには，より複雑な処理が含まれているため，合法手からランダムに選択するだけの本システムと比較することはできないが，今後 Feugo と同等の処理を追加しても，本システムの方が高速である可能性が高いと考える．

Enzenberger[3]ら，岸本[8]らはそれぞれツリー並列探索による囲碁プログラムの強さを評価し，コア数の増加に対する強さの上昇にピークがあるという結果を示した．われわれのシステムにおいても，棋力の評価や，棋力向上のスケーラビリティの高いツリー並列探索の手法が今後の重要な課題である．

別種の計算アクセラレータであるGPUを用いてプレイアウトを高速化する試みも行われている．たとえば田野らのプログラムはNVIDIA社のTesla C1060 上で 9 路盤のプレイアウトで 7.1 万po/sと報告されている[9]．GPUの世代が古いため，そのまま速度を比較するのは不公平であるが，単精度浮動小数点演算性能（C1060 は 933GFlops，Xeon Phi 5110Pは 2.022TFlops）を基準にすると，本システムが数倍高速であると考えられる[c]．さらに田野らのプログラムはプレイアウトルーチンのみであり，モンテカルロ木探索への組込みは行われていない．

囲碁に特化した研究ではないが Rocki[10]らは，NVIDIA 社 GPU を用いたモンテカルロ木探索の並列化手法として，複数のルート並列探索を並列実行する「ブロック並列化」を提案している．GPU はブロックのプレイアウトを並列に実行する．Barriga[11]らは「ブロック並列化」をもとに，GPU で木を 1 段展開し，すべての子ノードのプレイアウトを並列に実行した．

GPU を利用したこれらのいずれも手法も，GPU を複数プレイアウトの並列実行に利用している．これは多数のスレッドが同一の命令を実行する GPU のアーキテクチャに最適化した結果であり，GPU をモンテカルロ木探索に組み込む場合，並列プレイアウトエンジンとしての利用が最適であると考えられる．

それに対して本研究は，メニーコアとベクトル演算という二つの並列性を有効に利用することで，Xeon Phi において並列モンテカルロ木探索全体を高速に実行可能であることを示した．

Mirsoleimani らは Xeon Phi 上でのモンテカルロ木探索において Grain Size Controlled Parallel MCTS (GSCPM) いう粒度をコントロールする手法を提案した．また TBB, Cilk Plus を用いたスケジューリングの比較を行っている[12]．

6. おわりに

本報告では，メニーコアコプロセッサ Xeon Phi 向けのモンテカルロ囲碁プログラムの実装と性能評価について述べた．512 ビットのベクトル演算を活用することにより 240 スレッド実行時に毎秒 24 万回の並列プレイアウトを実現した．また，ツリー並列モンテカルロ木探索を実装した．ゲーム序盤で毎秒 22.1 万回の探索を達成し，シングルスレッド実行の 100 倍となる高いスケーラビリティを持つことを示した．

今後の課題としては，強い囲碁プログラムに組み込まれている着手改善手法を実装した上で，並列モンテカルロ木探索のスピードと棋力の評価を行うことがあげられる．

参考文献

[1] インテル Xeon Phi 製品ファミリー. http://www.intel.co.jp/content/www/jp/ja/processors/xeon/xeon-phi-detail.html.

c) C1060 は倍精度浮動小数点演算が 78GFlops と極端に遅い．囲碁のプレイアウトは整数演算がほとんどであるため，倍精度性能を比較基準にするのは不適切と考えた．

[2] S Gelly, D Silver. Combining online and offline knowledge in UCT. Proceedings of the 24th international conference on Machine learning, pp.273-280(2007).
[3] Guillaume M.J-B. Chaslot, Mark H.M. Winands, and H. Jaap van den Herik. Parallel Monte-Carlo Tree Search. the 6th International Conference on Computers and Games, pp. 60-71(2008).
[4] M. Enzenberger and M. Müller. A lock-free multithreaded Monte-Carlo tree search algorithm. Advances in Computer Games, 6048:pp.14-20(2010).
[5] 丸山真佐夫．モンテカルロ囲碁対局のためのプレイアウトプログラムの実装と評価．木更津高専紀要第 46 号，pp. 27-33（2013）.
[6] S. A. Mirsoleimani, A. Plaat, J. Vermaseren, and J. van den Herik. Performance analysis of a 240 thread tournament level MCTS Go program on the Intel Xeon Phi. in The 2014 European Simulation and Modeling Conference (ESM'2014), pp.88–94(2014).
[7] Intel Developer Zone. https://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding.
[8] 副島佑介, 岸本章宏, 渡辺治．モンテカルロ木探索の root 並列化とコンピュータ囲碁での有効性について. 第 14 回ゲームプログラミングワークショップ, pp. 27-33 (2009).
[9] 田野文彦．グラフィックエンジンを用いたゲーム探索の高速化，修士論文．東京大学大学院工学系研究科，pp.35-37 (2010).
[10] K. Rocki and R. Suda. Large-Scale Parallel Monte Carlo Tree Search on GPU. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium, pp.2034-2037 (2011).
[11] Nicolas A. Barriga, Marius Stanescu, Michael Buro. Parallel UCT Search on GPUs. IEEE Conference on Computational Intelligence and Games, pp.1-7 (2014).
[12] S. Ali Mirsoleimani, Aske Plaat, H. Jaap van den Herik, and Jos Vermaseren. Scaling Monte Carlo Tree Search on Intel Xeon Phi. CoRR (2015).

ⓒ2016 Information Processing Society of Japan