Vol.2009-GI-22 No.7
2009/6/26

マルチコアプロセッサとSIMD演算による モンテカルロ木探索を用いたオセロの実装

久保田 悠司[†1] 佐藤 佳州[†1] 高橋 大介[†1]

本稿では，モンテカルロ木探索を用いたコンピュータオセロの高速化を図る．近年，モンテカルロ木探索はコンピュータ囲碁などで注目されている．モンテカルロ木探索は playout の実行速度が性能に大きく影響するため，より強いプログラムを作るには playout の高速化が必要であると考えられる．本稿では，モンテカルロ木探索を用いたオセロを実装するとともに，マルチコアプロセッサである Cell B. E. を用いて評価した．高速化の手法としては，SIMD 演算とマルチコアプロセッサによる並列化を用いた．

SIMD 演算を用いて石の反転処理を高速化し，マルチコアプロセッサを用いて playout を並列に実行することで，playout の高速化に成功するとともに，勝率の向上を確認した．

Implementation of an Othello Program Based on Monte-Carlo Tree Search by Using a Multi-Core Processor and SIMD Instructions

YUJI KUBOTA,[†1] YOSHIKUNI SATO[†1]
and DAISUKE TAKAHASHI[†1]

Recently Monte-Carlo Tree Search is attracting attention in Computer-Go. Because the performance of Monte-Carlo Tree Search is related to the execution speed of playouts, programs become stronger by executing playouts faster. In this paper, we aim to accelerate playouts of an Othello program based on Monte-Carlo Tree Search. We implemented an Othello program based on Monte-Carlo Tree Search by using Cell B. E. that is a multi-core processor and SIMD instructions and evaluated the speed-up of playouts.

We accelerated the processing to reverse pieces by using SIMD instructions and parallelized playouts by using a multi-core prosessor. Furthermore our experiments showed that the speed-up of playouts raised win rate.

1. はじめに

近年，囲碁や将棋などのゲームプログラミングの分野ではモンテカルロ木探索[1)2)] が注目されている．モンテカルロ木探索とは乱数を用いたシミュレーション（playout）を繰り返し行い，その結果を用いて最善手を求める手法である．この手法の利点は評価関数を必要としないことである．評価関数を用いた手法では，ある盤面の状態を評価関数によって数値化し最善手を求めるため，この評価関数によって強さが左右される．囲碁や将棋などでは，評価関数を人間の知識に基づいて作成するが，どのような局面にも対応できる評価関数の作成は困難である．

一方，モンテカルロ木探索では playout を行い勝率をシミュレートすることにより，人間の知識によらず局面を評価することができる．Playout の回数が多いほど，正確な勝率を求めることができ，深く探索できるためより強いプログラムになる．つまり，この手法では playout の回数がそのプログラムの強さに大きく影響する．しかし，多くのゲームプログラムでは，１回の playout に時間を要する．したがって，より強いプログラムを作るためには playout の高速化が有効である．

また，最近ではマルチコアプロセッサが身近なものとなっており，市販のノート PC にも搭載されている．さらに，現在のマルチコアプロセッサでは，複数のデータに対して同一の演算を同時に実行する Single Instruction Multiple Data（SIMD）演算を用いることができる．モンテカルロ木探索ではそれぞれの playout は独立に実行できるため，これらを用いることで playout を高速に行うことが可能である．

本稿では，モンテカルロ木探索を用いたオセロを実装し，SIMD 演算とマルチコアプロセッサである Cell Broadband Engine（Cell B. E.）を用いて，playout の高速化を行い，評価する．

2. 関連研究

コンピュータオセロは，1980 年にはじめて人間とコンピュータの大会が開催された．その後 1980 年代には様々な国でコンピュータオセロの研究がなされ，より強くなっていった．そして 1997 年に，Logistello[3)4)] が初めて人間の世界チャンピオンに 6 対 0 で勝利した．こ

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

の Logistello には，探索と評価関数を用いた手法が取り入れられている．この手法は，オセロや将棋といったゲームプログラムでは一般的に用いられている手法である．

しかしコンピュータ囲碁において，探索と評価関数を用いた手法は良い性能が得られなかった．そこで，モンテカルロ木探索が提案された[1]．これは，探索とモンテカルロ法による評価を組み合わせた手法で，近年注目を集めている．モンテカルロ木探索では，playout の回数が性能に大きく影響するため，playout の高速化が重要となる．

高速化の方法の 1 つとして，加藤らの並列モンテカルロ木探索に関する研究[5] がある．この研究では，マルチコアプロセッサ上でのモンテカルロ木探索の並列化を行っている．

本稿ではマルチコアプロセッサによる並列化だけではなく，SIMD 演算を用いた高速化を行う．

3. コンピュータオセロ

3.1 盤面のデータ構造

オセロの盤面の表現方法である盤面のデータ構造には，配列を用いた表現や Bitboard を用いた表現がある．盤面のデータ構造によって，石の反転処理や合法手生成などの playout の大部分を占める処理の速度が違うことから，playout の高速化には盤面のデータ構造が重要である．本稿では，Bitboard を用いた表現を用いる．

Bitboard を用いた表現では，盤面の状態をビットにより表す．オセロのマスの数は $8\times8=64$ であるため 64 ビットの符号なし整数型を使う．マスの状態としては黒，白，空白の 3 種類あるため，黒の状態（black）と白の状態（white）を表す 2 つの変数を用意し，この 2 つの論理和で空白の状態を表す．

この Bitboard を用いた表現では，盤面を変数 2 つで表すことができるためデータ量が少なく，盤面の反転操作や更新操作にビット演算を用いるため高速に実行できる．**図 1** に文献 6) で示されている石の反転処理の擬似コードを示す．

このアルゴリズムは，合法手である着手箇所（position）を入力とし，そこから左方向への石の反転パターンを求め反転パターンを rev として返す．invert は引数と逆の色の Bitboard，たとえば引数が黒であるとした場合，白の Bitboard を返す．ビットシフトの方向を逆にすれば右方向への石の反転処理のアルゴリズムになる．このアルゴリズムを 8 方向に適用することで反転させるすべての石の反転処理を行う．

次に，合法手の生成について述べる．Bitboard を用いた表現では，ある 1 方向に対して，一度にすべてのマスについて合法手かどうかを判定することができるため，合法手を高速に

```
reverse_left( position, color)
  co ← invert(color) & 0x7e7e7e7e7e7e7e7e
  e1 ← (position >> 1) & co
  e2 ← (e1 >> 1) & co
  e3 ← (e2 >> 1) & co
  e4 ← (e3 >> 1) & co
  e5 ← (e4 >> 1) & co
  e6 ← (e5 >> 1) & co
  b1 ← (color << 1) & co
  b2 ← (b1 << 1) & co
  b3 ← (b2 << 1) & co
  b4 ← (b3 << 1) & co
  b5 ← (b4 << 1) & co
  b6 ← (b5 << 1) & co
  rev ← e6 & b1
  rev ← rev | (e5 & (b1 ← b1 | b2))
  rev ← rev | (e4 & (b1 ← b1 | b3))
  rev ← rev | (e3 & (b1 ← b1 | b4))
  rev ← rev | (e2 & (b1 ← b1 | b5))
  rev ← rev | (e1 & (b1 | b6))
  return rev
```

図 1 Bitboard を用いた表現における左方向の石の反転処理の疑似コード

```
check_left( color)
  w ←invert(color) & 0x7e7e7e7e7e7e7e7e
  t ← w & (color << 1)
  t ← t | (w & (t << 1))
  t ← t | (w & (t << 1))
  t ← t | (w & (t << 1))
  t ← t | (w & (t << 1))
  t ← t | (w & (t << 1))
  t ← t | (w & (t << 1))
  mobility ← blank & (t << 1)
  return mobility
```

図 2 Bitboard を用いた表現における左方向の合法手生成の擬似コード

生成できる．**図 2** に文献 7) で示されている合法手生成の疑似コードを示す．このアルゴリズムも，石の反転処理のアルゴリズムと同様に，ある 1 方向への判定であるため，これを 8 方向に適用しすべての合法手を生成する．

3.2 モンテカルロ木探索

思考ゲームの実現法としては，探索と評価関数を用いた手法と，モンテカルロ木探索がある．本稿ではモンテカルロ木探索を用いる．

モンテカルロ木探索とは評価関数を使わず，playout を繰り返し行うことによって最善手を探索していく手法である．まず，現在の盤面をルートとして木を作りその盤面より 1 手進めた状態の子ノードを作る．その後，乱数によるシミュレーションを繰り返し行い，そのノードの勝率を求める．また，より有効な手に対して多くの playout を割り当て，一定回数 playout を行ったノードでは**図 3** のようにそのノードの状態よりも 1 手進めた状態の子ノードを作り，より深く木の探索を行う．

このモンテカルロ木探索における代表的なアルゴリズムが UCT（Upper Confidence bounds

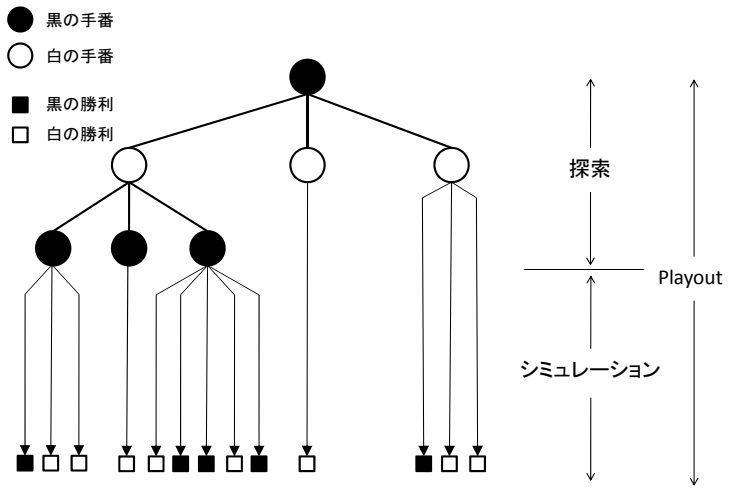

図 3 モンテカルロ木探索

applied to Trees)[8] である．これはノードの選択に，UCB1[9] を用いたアルゴリズムである．UCB1 では，以下の式 (1) が最大となるノードを選択する．

$$\overline{x}_i + \sqrt{\frac{2\log n}{n_i}} \tag{1}$$

$\overline{x}_i$ はノード i の勝率，n_i はノード i が選択された回数，n はノード i の親ノードが選択された回数を表す．

まだ選択されていないノードに対しては式 (1) の値を ∞ として，優先して選択されるようにし，選択されたことのあるノードに対してはそのノードの選択された回数と勝率から選択するノードを決定する．

4. 高 速 化

4.1 SIMD 演算を用いた高速化

SIMD 演算では，複数のスカラデータに対し同じ演算を同時に行うことができる．本稿で用いるデータ型は 128 ビットのベクタデータで，128 ビットというのは整数型で 4 つのデータ，倍精度実数型で 2 つのデータが入る大きさである．Bitboard を用いた表現では盤面を 64 ビットの符号なし整数型で表すため，2 つのデータを 1 つのベクタデータとして扱うことができる．

本稿では，SIMD 演算を用いて playout 内での処理を高速化する．1 回の playout で大部分を占める処理は，石の反転処理と合法手の生成である．**図 4** に，3.1 節で述べたアルゴリズムにおける SIMD 演算を用いた石の反転処理を示す．

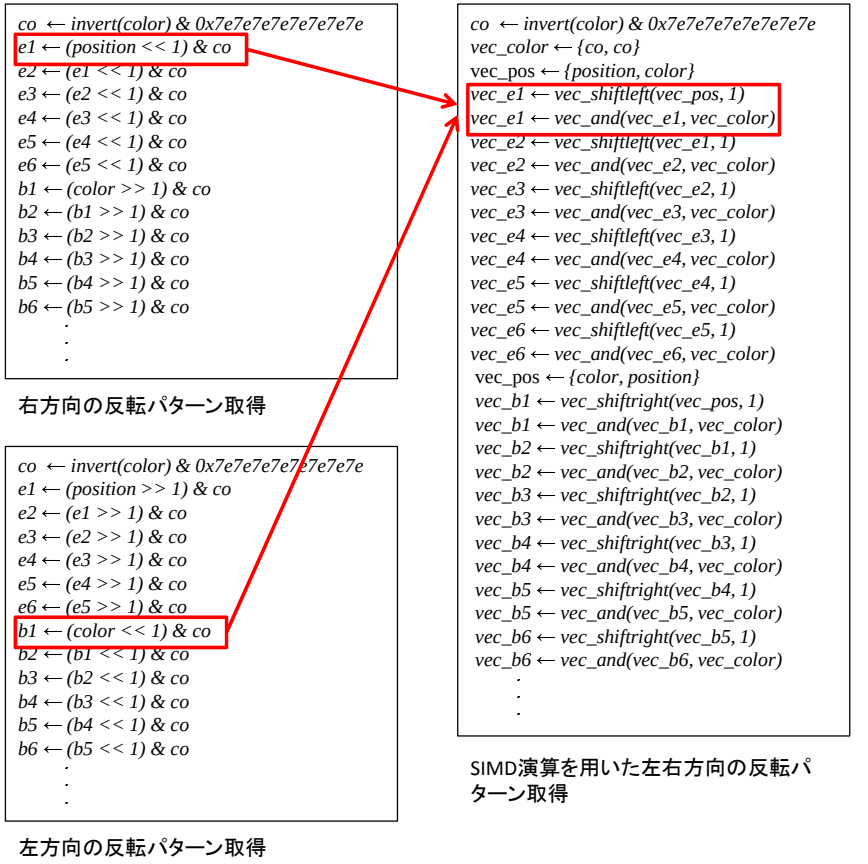

図 4 SIMD 演算を用いた石の反転処理

石の反転処理では 1 方向ずつ求めるため，異なる方向については独立に実行することが

できる．また，ある 1 方向の石の反転処理では左右両方向へのシフト演算を用い，これとは逆の方向の石の反転処理では，シフト演算のシフトする向きを逆にする．そのため，互いに逆方向の石の反転処理は，同じ方向へのシフト演算を用いるため SIMD 演算を用いて同時に求めることができる．

4.2 マルチコアプロセッサにおける並列化

次にマルチコアプロセッサ上での並列化の手法としてクライアントサーバ方式[5] について述べる．

クライアントサーバ方式では，マルチコアプロセッサ上の 1 つのコアがサーバ，他のすべてのコアがクライアントとなる．サーバでは探索木の管理，クライアントではシミュレーションの実行を行う．

まずサーバで探索木を作成し，探索木の降下を行う．末端のノードまで降下した後，クライアントにシミュレーションを割り当てる．クライアントを選ぶ方法は，まず最初はすべてのクライアントに順番にシミュレーションを割り当てる．その後はシミュレーションが終了し，結果を返したクライアントに次のシミュレーションを割り当てる．サーバからシミュレーションを割り当てられたクライアントでは，サーバからシミュレーションの実行に必要な盤面の状態などのデータを受け取り，シミュレーションを実行する．シミュレーションの終了後，サーバに結果を返し，次のシミュレーションの割り当てを待つ．サーバではクライアントから送られてくる結果から探索木を更新し，次の playout を開始する．

5. Cell Broadband Engine を用いた実装

5.1 Cell B. E. の構成

Cell B. E. は 1 個の PowerPC Processor Element（PPE）と 8 個の Synergistic Processor Element（SPE）から構成されているヘテロジニアスマルチコアプロセッサである[10]．本稿では PLAYSTATION3 に搭載されている Cell B. E. を用いるが，この Cell B. E. では SPE を 6 個のみ使うことができる．

PPE は 64 ビットの PowerPC アーキテクチャで，OS やユーザインターフェースなどを行っている．SPE は SIMD 演算に適したアーキテクチャで，それぞれが独自にローカルストア（LS）と呼ばれる 256KB のメモリを持ち，LS のみデータの参照ができるため必要なデータは LS に送る必要がある．PPE とそれぞれの SPE の間は Element Interconnect Bus（EIB）と呼ばれるリングバスで接続されている．

図 5 に Cell B. E. の構成を示す．

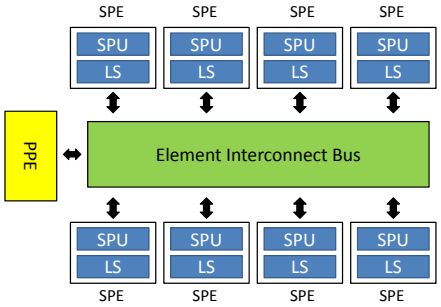

図 5 Cell B. E. の構成

Cell B. E. での SIMD 演算は，PPE では Vector Multimedia Extension（VMX）と呼ばれる SIMD 命令があり，SPE では Cell B. E. 独自の SIMD 命令（SPU SIMD）がある．SPU SIMD は VMX と比べて命令セットが少なく，演算の対象となるデータ型が限られている場合がある[11]．

PPE-SPE 間や SPE 間のデータ転送には Direct Memory Access（DMA）転送や Mailbox を用いた転送がある．DMA 転送では CPU を使わずにメインメモリとローカルストア間のデータ転送を行い，最大で 16KB，最小で 16B のデータが転送できる．また，DMA 転送で転送するデータサイズは 16B 単位である必要がある．Mailbox を用いた転送では FIFO のキューのような構造となっており PPE-SPE 間で双方向に転送ができる．転送できるデータ量は 32 ビットである．

5.2 Cell B. E. を用いた実装

ここでは，まず Cell B. E. 上でのクライアントサーバ方式の実装について述べる．

Cell B. E. では，PPE をサーバ，SPE をクライアントとして実装し，PPE では図 3 の探索部分，SPE ではシミュレーション部分を行う．

まず，PPE で探索木を作成し，探索木の降下を行う．末端まで降下した後，待機状態の SPE にシミュレーションを割り当てる．シミュレーションを割り当てる SPE を選択するときは，最初は順番にすべての SPE にシミュレーションを割り当る．その後はシミュレーションの実行が終了して結果が返ってきた SPE に次のシミュレーションを割り当てる．PPE からシミュレーションを割り当てられた SPE は，盤面の状態などのシミュレーションの実行

に必要なデータを自分のローカルストアに DMA 転送する．その後，SPE では PPE から転送した情報をもとにシミュレーションを実行し，その結果を PPE に送る．このときのデータ転送には Mailbox を用いた転送を使う．そして PPE では SPE から送られてきたデータをもとに探索木を更新し，次の playout を開始する．

図 6 に PPE の擬似コード，**図 7** に SPE の擬似コードを示す．

```
uct()
  while 1 手にかける時間 do
    playSim(root)
  end while

playSim(root)
  node[0] ← root
  while node[i] が訪れたことのあるノード do
    if node[i] に子ノードがない then
      createChildren(node[i])
    end if
    node[i + 1] ←selectNode(node[i])
    i ← i + 1
  end while
  while シミュレーションが終了した SPE が見つかるまで do
    check(spe[j])
    j ← j + 1
    if j が（SPE の数-1）と等しい then
      j ← 0
    end if
  end while
  result ←spe_ run(spe[j])
  upDate(result, node)
```

図 6 PPE（サーバ）の疑似コード

以下に図 6 と図 7 で使われている関数を示す．

createChildren($node$): 引数 $node$ の子ノードを生成する．

selectNode($node$): 引数 $node$ の子ノードの中から式 (1) が最大となるノードを選択する．

check(spe): 引数 spe で与えられた SEP のシミュレーションが終了したかどうかを調べる．

spe_run(spe): 引数 spe で与えられた SPE にシミュレーションを実行するように命令する．

upDate($result, node$): 配列 $node$ のすべてのノードを $result$ によって更新する．

```
Sim_spe()
  loop
    if PPE からシミュレーションを割り当てられる then
      getboard()
      result ←randomSim()
      putResult(result)
    end if
  end loop
```

図 7 SPE（クライアント）の疑似コード

getboard(): シミュレーションを実行するために必要な情報を DMA 転送で取得する．

randomSim(): 乱数によるシミュレーションを行い，勝敗を返す．

putResult($result$): 引数 $result$ の値を PPE に Mailbox を用いて転送する．

さらに，本稿では SPE で行うシミュレーションを SIMD 演算を用いて高速化する．しかし，5.1 節で触れたように SPE 上では使用できる SIMD 演算が限られている．たとえばシフト演算では，ベクタデータの各要素に対して同じ方向に同じビット数分のみシフトできる．つまり，合法手の生成のように，異なる 2 方向に対する処理で，シフトする方向が同じでもシフトするビット数が違う処理には SIMD 演算を用いることができない．しかし，石の反転処理のように，異なる 2 方向に対する処理の中に，シフトする方向が同じでシフトするビット数も同じであるシフト演算が含まれている処理は SIMD 演算を用いて高速化できる．

よって本稿では，SPE でのシミュレーションの石の反転処理の部分のみを SIMD 演算を用いて高速化する．SIMD 演算による石の反転処理は図 4 のように，同じ方向，同じビット数分シフトする演算をまとめて 1 つの SIMD 演算として実装する．このとき SIMD 演算を用いて高速化できる処理としては，右方向と左方向，上方向と下方向といった反対方向の石の反転処理である．

また，モンテカルロ木探索では，ノードを選択する時に式 (1) のように浮動小数点除算を用いる．しかし，Cell B. E. は浮動小数点除算が遅いため，PPE での木の探索に時間がかかってしまう．そこで，本稿では playout 中の 1 回の探索ごとに複数回のシミュレーションを行うことで浮動小数点除算を行う割合を減少させている．つまり，PPE で探索を始め末端のノードまで降下した後，SPE でシミュレーションを行うときに複数回行っている．1 回の探索に対しシミュレーションを複数回行った場合は，シミュレーションの回数分 playout の回数として加算する．

Playout は図 3 のように探索部分とシミュレーション部分に分かれる．Cell B. E. 上で時間のかかってしまう浮動小数点除算は探索部分に含まれている．探索 1 回に対しシミュレーションを 1 回行う場合，より深く探索するたびに，SPE1 個でのシミュレーションの実行時間よりも PPE での 1 回の探索に時間を要する．そのため，SPE は PPE が探索を完了するまで待つ必要があり，SPE を複数利用してもほとんどの SPE が待機状態になってしまい，全体の playout 回数は増加しない．

しかし，探索ごとのシミュレーション回数を多くすることにより，PPE での 1 回の探索よりも SPE1 個でのシミュレーションの実行時間が長くなる．そのため，SPE はほとんど待機状態になることなく PPE からシミュレーションを割り当てられる．したがって，SPE を複数用いることで PPE は無駄なく SPE にシミュレーションの実行を割り当てることができ，全体の playout 回数が増加する．

6. 評　　価

6.1 実験環境

Cell B.E. での実験環境を**表 1** に示す．

表 1　Cell B.E. での実験環境

実験マシン	PLAYSTATION3
CPU	PPE & SPE
OS	Fedora 9 GNU/Linux 2.6.25-14.fc9.ppc64
開発言語	C 言語
コンパイラ	ppu-gcc 4.1.1 & spu-gcc 4.1.1
コンパイラオプション	-O3

6.2 SIMD 演算を用いた高速化による速度向上

SIMD 演算を用いた playout の速度向上について評価を行った．SPE のコア数は 1 個とし，オセロの初手を打つのに要した playout 数を測定した．測定は 400 回行い，その平均値で評価を行った．**図 8** に SIMD 演算を用いた高速化による速度向上の比較結果を示す．凡例の「sequential」は SIMD 演算を用いていないプログラムでの測定結果であり，「SIMD」は SIMD 演算を用いたプログラムでの測定結果である．

図 8 では，SIMD 演算を用いた高速化により最大で約 1.3 倍速度向上したことを示している．これは探索ごとのシミュレーション回数が 32 回以上のときであり，これよりも少なくなるにつれて速度向上比は小さくなった．この原因は，探索ごとのシミュレーション回数が

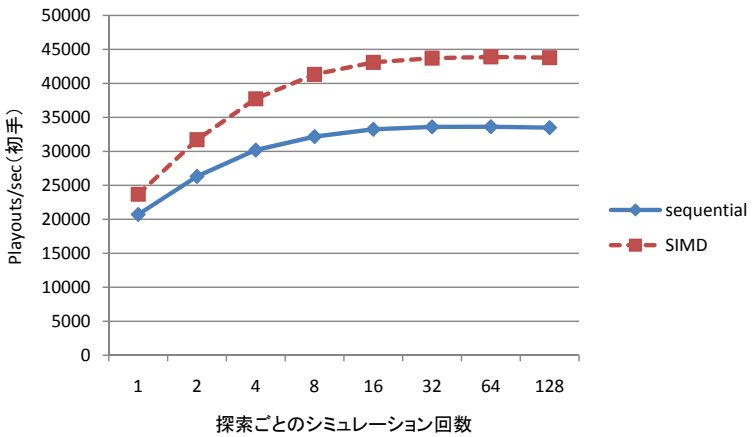

図 8　SIMD 演算を用いた並列化による速度向上

少ない場合，全体の実行時間に対するシミュレーションの実行時間の割合が低くなるためである．本稿で SIMD 演算を用いて高速化しているのは SPE でのシミュレーションであるため，シミュレーションの実行時間の割合が低い場合，SIMD 演算による高速化を行っても速度向上に大きな差はなかった．探索ごとのシミュレーション回数が 32 回以上の場合は，1 回の木の探索に要する実行時間よりもシミュレーションの実行時間が長くなり，PPE が SPE の結果待ちになっていると考えられる．したがって，SPE でのシミュレーションの実行速度が最大となり，それ以上の速度向上は見られなかった．

6.3 マルチコアプロセッサを用いた並列化による速度向上

マルチコアプロセッサを用いた並列化での速度向上について評価を行った．プログラムには SIMD 演算を用い，オセロの初手を打つのに要した playout 数を測定した．測定は 400 回行い，その平均値で評価を行った．**図 9** に探索ごとのシミュレーション回数によるマルチコアプロセッサを用いた並列化による速度向上の測定結果を示す．凡例は探索ごとのシミュレーション回数である．

図 9 に示したように，探索ごとのシミュレーション回数が 16 回以上のときに，SPE のコア数を 6 倍にすることで最大で約 6 倍の速度向上が得られた．これはすべての SPE がほとんど待機状態になることなく使用できていることを示している．探索ごとのシミュレーション回数が 16 回よりも少ないときは，SPE でのシミュレーションより PPE での探索に時間を

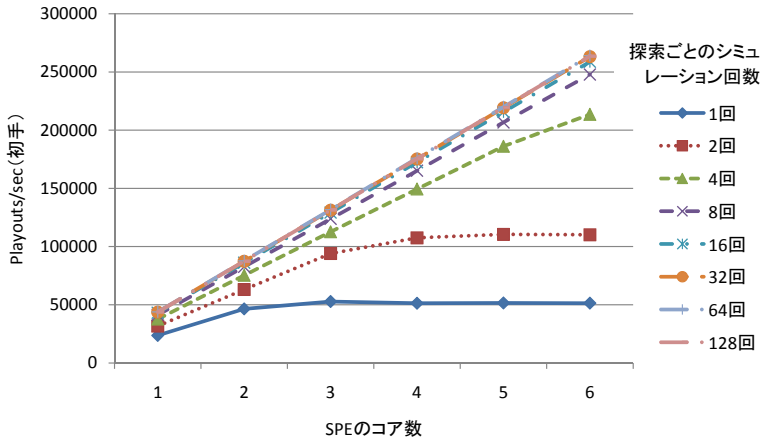

図 9 マルチコアプロセッサを用いた並列化による速度向上

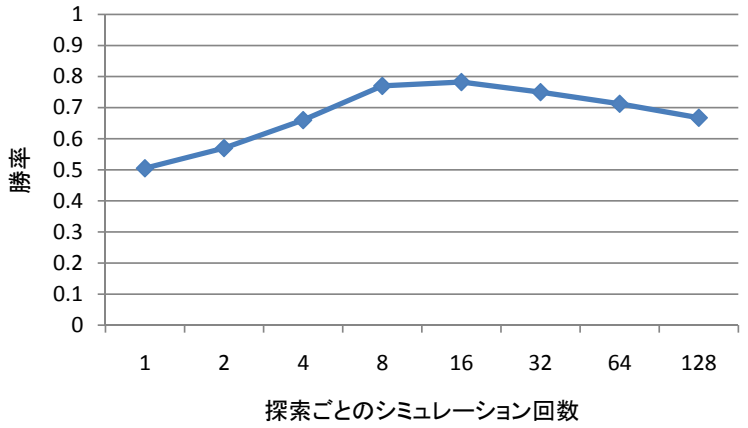

図 10 探索ごとのシミュレーション回数による勝率の変化

要するため，SPE を複数利用してもほとんどが待機状態になってしまう．そのため，playout 回数は増加しなかったと考えられる．

6.4 自己対局による性能評価

Playout の高速化による勝率の変化について評価を行った．探索ごとのシミュレーション回数を 1 回 ~128 回まで変化させて対局を行い，勝率を測定した．**図 10** は探索ごとのシミュレーション回数が 1 回のときを基準とした，探索ごとのシミュレーション回数による勝率の変化を示している．用いたプログラムはすべて SIMD 演算を用い，SPE のコア数は 6 個とした．対局はそれぞれ 400 回ずつ行い，時間制限は 1 手 1 秒とした．また，対局は 2 回で 1 組とした．まず，最初の数手をランダムに打った状態から対局を始め，その対局が終了した後，手番を入れ替え盤面をランダムに打った状態に戻してから対局を行った．

図 10 に示したように，探索ごとのシミュレーション回数が 16 回以下のときは，探索ごとのシミュレーション回数が増加するに従って勝率の向上が見られる．これは，これまでの 2 つの評価で述べた SIMD 演算を用いた高速化とマルチコアプロセッサを用いた並列化により playout の速度が向上し，より多くの playout を実行することにより，正確な盤面の評価を行っているからである．また，探索ごとのシミュレーション回数が 16 回より多いときに勝率が低下しているのは，あまり有効でない手に対しても多くの playout を行っているため，その分の playout が無駄になっているからであると考えられる．

7. おわりに

本稿では，SIMD 演算とマルチコアプロセッサである Cell B. E. を用いて，モンテカルロ木探索を用いたオセロにおける playout の高速化を行った．SIMD 演算を用いた高速化では，石の反転処理について高速化を行った．マルチコアプロセッサを用いた並列化では，Cell B. E. 上でクライアントサーバ方式を用いて並列化を行った．これら 2 つの高速化を行うことにより，SIMD 演算による高速化やマルチコアプロセッサによる並列化を行っていないときと比べて約 7.8 倍の速度向上が得られた．

また，自己対局による性能評価では playout の高速化による勝率の向上を確認できた．

これらの結果からモンテカルロ木探索において，playout の高速化が性能の向上に影響することが確認できた．また，その playout の高速化には，SIMD 演算を用いた高速化やマルチコアプロセッサを用いた並列化が有効であるといえる．

今後の課題として，石の反転処理だけでなく合法手の生成処理の SIMD 演算を用いた高速化が挙げられる．また，オセロだけでなく他のモンテカルロ木探索を用いたゲームについても playout の高速化を行いたい．

参 考 文 献

1) Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, *Proc. 5th International Conference on Computer and Games* (vanden Herik, J.H., Ciancarini, P. and Donkers, j. H. H. L.M., eds.), Lecture Notes in Computer Science, No.4630, Springer-Verlag, pp.72–83 (2006).
2) 美添一樹：モンテカルロ木探索—コンピュータ囲碁に革命を起こした新手法 (Monte-Carlo Tree Search — A Revolutionary Algorithm Developed for Computer Go)，情報処理, Vol.49, No.6, pp.88–95 (2008).
3) Buro, M.: The Evolution of Strong Othello Programs, *Entertainment Computing - Technology and Applications* (Nakatsu, R. and Hoshino, J., eds.), Kluwer, pp.81–88 (2003).
4) Buro, M.: LOGISTELLO's Homepage, `http://www.cs.ualberta.ca/~mburo/log.html`.
5) 加藤英樹，竹内郁雄：並列 MC/UCT アルゴリズムの実装，第 12 回ゲームプログラミングワークショップ，pp.23–29 (2007).
6) bitboard 反転パターン取得，`http://vivi.dyndns.ws/W/257`.
7) オセロの試合結果は何通りか？@Wiki，`http://www9.atwiki.jp/othello/pages/48.html`.
8) Kocsis, L. and Szepesvari, C.: Bandit based Monte-Carlo Planning, *Proc. 17th European Conference on Machine Learning*, Lecture Notes in Computer Science, No.4212, Springer-Verlag, pp.282–293 (2006).
9) Auer, P., Fischer, P. and Cesa-Bianchi, N.: Finite-time Analysis of the Multi-armed Bandit Problem, *Machine Learning*, Vol.47, pp.235–256 (2002).
10) 林宏雄：Cell Broadband Engine 概要 : その設計思想と応用例 (デジタル・情報家電, 放送用, ゲーム機用システム LSI, 及び一般)，電子情報通信学会技術研究報告. ICD, 集積回路， Vol.105, No.569, pp.25–30 (2006).
11) Sony Computer Entertainment Inc.: Cell Broadband Engine アーキテクチャ用 C/C++言語拡張, `http://cell.scei.co.jp/pdf/Language_Extensions_for_CBEA_v23_j.pdf`.