プログラミング-言語・基礎・実践- 11－6
(1993. 3. 10)

マルチスレッドを用いた項書換え系の並列処理系

石川 亮　山本 晋一郎　酒井 正彦　阿草 清滋

名古屋大学 工学部

464-01 名古屋市 千種区 不老町 名古屋大学 工学部

あらまし

一定個数のセルの集まりとその管理、および書換え等の操作を抽象化したクラスBOB (Bundle of Branches) を用いた、共有メモリ型マルチプロセッサ計算機上で動作する項書換え系の並列処理系について報告する。本稿では、まず書換えのfairnessを保証する並列処理の機構を示し、つぎに計算機実験によって4台のCPUを持つ計算機において2倍程度計算速度が向上することを示した。

和文キーワード　項書換え系, 共有メモリ, BOB

An Implementation of TRS on a Shared Memory Multi-Processor Machine

Akira ISHIKAWA　Shinichirou YAMAMOTO　Masahiko SAKAI　Kiyoshi AGUSA

School of Engineering, Nagoya University

School of Engineering, Nagoya University,
Furo, Chikusa, Nagoya, 464-01

Abstract

BOB proposed as a new methodology of data management for parallel term rewriting is implemented on a shared memory multi-processor machine which has 4 CPUs. We introduce two BOB queues, called Queue and interRewritePool, to guarantee the fairness of BOB's rewriting. BOBs to be processed are bufferd in them. Threads struggle for a BOB from queues and rewrite it.

英文 key words　TRS, shared memory, BOB

1 はじめに

近年の著しいハードウェア技術の進歩によって、複数個プロセッサを持つ計算機が容易に入手できるようになった。しかし、現在のシステムはその特徴を利用し、複数のプロセッサで同時にひつの処理するようなものは少ない。

また、関数型プログラムは本質的に並列性を内在しているため並列分散処理と馴染みが良いが、その並列性を十分に引き出している処理系は数少なく、実現方法も確立していない。

我々はすでに関数型プログラムの理論モデルの一つである項書換え系の並列処理を行うためのセル管理の方法として BOB (Bundle of Branches) を提案し、BOB を用いた実現が参照の局所性を維持することを計算機実験を通して定量的に示した [3]。

本稿では、BOB を用いた、密結合型計算機上でマルチスレッドにより動作する項書換え系の並列処理系の実現について報告する。

2 準備

2.1 項書換え系

本稿では，次の 2 つの制約を満たす項書換え系を対象とする。

1. 無曖昧性: 任意の 2 つの書換え規則に対して，一方の左辺が他方の左辺の変数以外の部分項と単一化可能でない。ただし，自分自身との自明な単一化を除く。

2. 左線形性: 書換え規則の左辺に同一の変数が 2 回以上出現しない。

項書換え系が上の 1 と 2 を満たすとき，その系は合流性を持ち，各々の項の正規形は (存在するならば) 一意である [1]。

項書換え系の計算とは与えられた入力項に対し、対応する正規型を求めることである。よって正規型を持つ任意の項に対してその正規型を求めることができる正規化戦略が重要である。上の 1 と 2 を満たす項書換え系に対する正規化戦略として並列最外戦略知られている。

書換えの戦略には各 BOB 内の最外リデックスを全て同時に書換える戦略を採用した。BOB 間にまたがったリデックスを無視すれば、全体としては最外リデックス集合を包含したリデックス集合を書換える戦略とみなすことができる。

ただし、BOB 間にまたがったリデックスの書換えは、後回しにされることがあるので、正確には並列最外戦略と一致しない。

3 BOB - Bundle of Branches

BOB (Bundle of Branches) は一定個数のセルの集まりとその管理、および書換え等の操作を抽象化したオブジェクトであり、同一 BOB 内のセル参照と BOB 外に位置するセルの参照を明確に区別する。

以下では、最初に BOB 間にまたがったセル参照のための機構 Port を、次に BOB の全体像を説明する。

3.1 Port

Port には Source と Sink の 2 種類の型があり、Source と Sink の組によって BOB 間のセル参照を構成している。すなわち、Source は他の BOB の Sink と双方向のポインタで結合されている。

言い替えると、Source は BOB 内の部分項の頂点を保持し、文字通り BOB の入口である。Sink は BOB 内のセルから他の BOB セルへの参照を保持し、BOB の出口である。

また、Source が BOB 内のセルを参照している状態と Sink が BOB 内のセルに参照されている状態を busy であると呼ぶ。以下では、この Source と Sink の組によるセル参照をポートリンクと呼ぶ。

3.2 BOB

部分木 (正確には枝) をひとかたまりに束ねたものが、BOB の直観的なイメージである。そのイメージを図 1に示す。BOB は一定個数の cell を持ち、BOB 内部で cell の管理を行う。項の構成に使われていな

いセル (フリーなセル) は BOB 内部のフリーセルリストにつながっている。BOB は 他の BOB とポートリンクにより結合され、複数の BOB により全体の木を表す。ある BOB に注目したとき、その BOB の source からつながる BOB を upper BOB、sink からつながる BOB を lower BOB と呼ぶ。

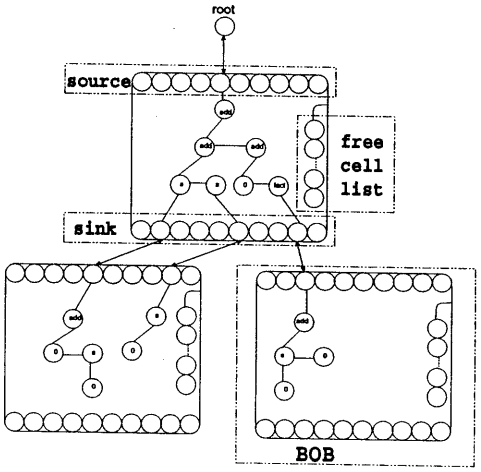


図 1: BOB の概念図

3.3 BOB の生成と消滅

BOB が生成されるのは 2 通りの場合がある。

BOB はまずはじめに入力項を構成するために生成される。次に書換えが開始され、BOB 内のフリーなセルが不足した場合、BOB 内の枝の一部を新たに生成した BOB に移動させることによって、フリーなセルを確保する。これを BOB 分割という。BOB 分割を繰り返すことにより BOB の数が増加する。BOB 分割は他の BOB との共有資源であるポートリンクを張り替える必要があるので、ポートリンクを張り替える時にはリンクをロックしなければならない。

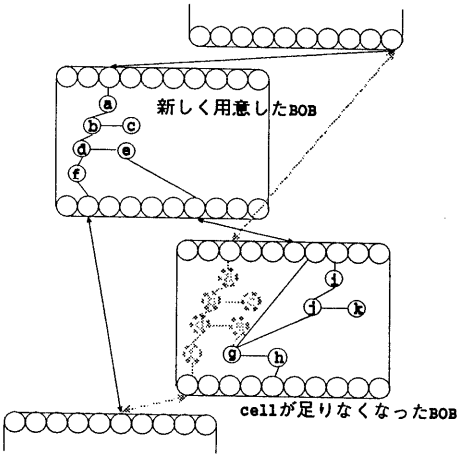


図 2: BOB 分割

3.4 BOB 内書換えと BOB 間書換え

一つの BOB 内のセルのみで構成されるリデックスのテンプレートの部分を BOB 内リデックスといい、これを書換えることを BOB 内書換えという。また、複数の BOB 間にまたがるリデックスの添付レートの部分を BOB 間リデックスといい、BOB 間リデックスの頂点を保持する BOB 内に含まれる部分を upper リデックス、残りの部分を lower リデックスという。また、これを書換えることを BOB 間書換えという。BOB 間書換えの時、書換えルールの右辺は BOB 間リデックスの頂点を保持する BOB 内に構成する。

BOB 間書換えは BOB 内書換えに比べ複数の BOB を使うという点でコストが高いので、ある BOB に着目したとき BOB 内リデックスがあるうちはなるべく BOB 内書換えを行うようにする。しかし書換えの fairness を保証するため、ある程度の頻度で BOB 間書換えも行わなければならない。

また、BOB 内の cell を頂点とするリデックスがないかマッチングを行うことを BOB をチェックすると呼ぶ。

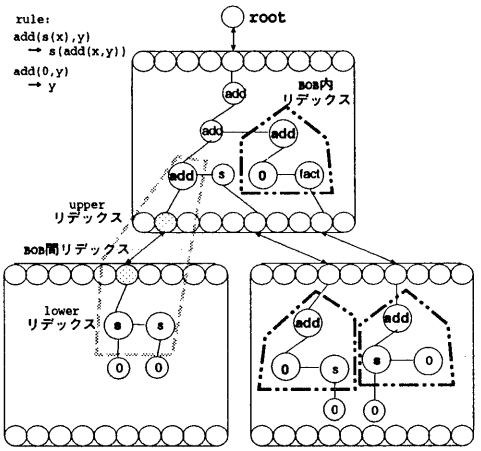


図 3: BOB 内リデックスと BOB 間リデックス

4 スレッドを用いた BOB の並列処理

4.1 並列処理の機構

書換えの fairness を保証するためには次の 2 つの条件を満たさなければならない。

1. 全ての正規化されていない BOB は有限時間内にチェックさなければならない。
2. BOB 内または BOB 間リデックスは有限時時間内に書換えられなくてはならない。

BOB の管理は書換えの fairness を保証するために queue, interRewritePool, normalizedPool を用いて行う (図 4)。

実行開始時に存在する唯一のスレッドを main thread と呼ぶ。main thread は queue から BOB を取り出し, 1 つの BOB に対し 1 つスレッドを発行し、そのスレッドに BOB 内書換えをさせる。発行されたスレッドを co-thread と呼ぶ。

BOB 間リデックスを書き換える場合は、BOB 間リデックスがまたがる BOB を interRewritePool に集め、必要な BOB が全て集まるとそれらに対し 1 つの co-thread を発行し、BOB 間書換えをさせる。

正規化された BOB は normalizedPool に集め、全ての BOB が normalizedPool に入った時に書き換えは終了する。

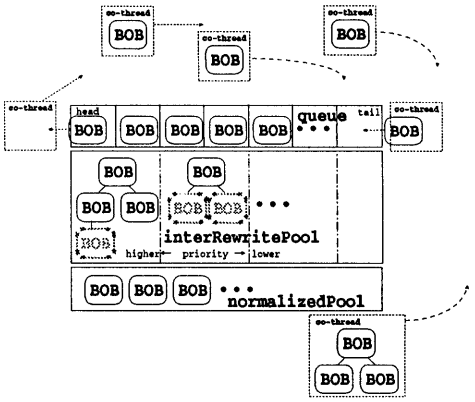


図 4: BOB の管理

4.2 BOB の状態

前述の実現法を BOB の状態という観点から整理すると以下のようなになる (図 5)。

- queue の中にある状態 (inQueue)
- interRewritePool の中にある状態 (interRewritePool)
- normalizedPool の中にある状態 (normalForm)
- BOB 内書換えを行っている状態 (innerRewrite)
- BOB 間書換えを行っている状態 (interRewrite)
- BOB 分割行っている状態 (divide)

正規化された BOB とは次の 1 かつ 2 が成立する BOB である。

1. その BOB 内に BOB 内リデックスが存在しない。かつ、
2. busy な sink がある場合、

(a) upper リデックスがない
または

(b) lower BOB との間で BOB 間リデックスがなく、upper リデックス内の sink からつながる BOB が正規化されている

BOB は一度正規化されると BOB 間書換えの lower-BOB として使われた場合でも、書換え規則の右辺は upper リデックスがある BOB 内に構成されるので正規化されたままである。

全ての BOB が正規化されたとき、項は正規型である。

入力項を構成する BOB は inQueue で生成され、その他の BOB は divide で生成される。

innerRewrite,interRewrite で busy な source がなくなった BOB は消滅する。

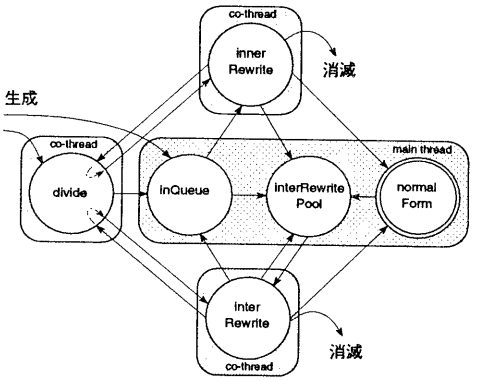


図 5: BOB の状態遷移

4.3 BOB の割り当て

interRewritePool に BOB を集める場合、BOB 間書換えの fairness を保証するため、BOB の割り当てに優先順位を設ける。優先順位は、必要な BOB を集めるよう依頼した順に高くし、ある BOB を割り当てるとき、その BOB が複数の BOB から必要とされていれば優先順位の最も高いものに割り当てる。

queue から取り出した BOB が interRewritePool で必要とされている場合、

1. この BOB が queue に入る前、この BOB がチェックされた場合
または、
2. この BOB が interReritePool で必要とする BOB の組合わせの頂点の BOB でる場合

この BOB を interReritePool に渡す。そうでない場合は co-thread を発行し BOB 内書換えをさせる。

5 main thread と co-thread

5.1 main thread のアルゴリズム

main thread のアルゴリズムは以下に示す。

```
main_thread()
{
  while (全ての BOB が正規化されるまで) {
    queue から BOB を一つ取り出す;
    if (interRewritePool で必要である &&
         (queue に入る前にこの BOB がチェックされた ||
          interReritePool で必要とする
          BOB の組合わせの頂点の BOB である) )
    {
      この BOB を interRewritePool に渡す;
      if (interRewritePool で必要な
                組み合わせが全てそろった) {
        co-thread を発行 (innerRewrite);
                        /*BOB 間書換えをさせる */
      }
    } else {
      co-thread を発行し、BOB 内書換えをさせる;
    }
    if (normalizedPool にある BOB が
          interRewritePool で必要である) {
      その BOB を interRewritePool に渡す;
      if (interRewritePool で必要な
                組み合わせが全てそろった) {
        co-thread を発行 (interRewrite);
                        /*BOB 間書換えをさせる */
      }
    }
  }
}
```

5.2 co-thread のアルゴリズム

BOB 内書換えのアルゴリズム (innerRewrite()) と BOB 間書換えのアルゴリズム (interRewrite()) を以下に示す。

```
innerRewrite()
{
  while (busy な source について) {
    if (ポートリンクが切れている場合) {
      ガーベージコレクション;
    } else {
      その source の child からマッチングして
      書換える;
      if (フリーなセルが足りない) {
        この BOB を分割する;
      }
    }
  }
  if (一つも書換えなかった) {
    if (upper リデックスが存在しない) {
      /* この BOB は正規化された */
      この BOB を normalizedPool にいれる;
      return;
    } else {
      BOB 間書換えのためにこの BOB を
      interRewritePool に入れ、必要な
      lower BOB を集めてもらう;
      return;
    }
  }
  if (upper リデックスが存在する &&
      以前 n 回以内にチェックされたとき
            BOB 間書換えをしていない) {
      BOB 間書換えのためにこの BOB を
      interRewritePool に入れ、必要な
      lower BOB を集めてもらう;
      return;
  }
  BOB を queue に戻す;
  return;
}
```

```
interRewrite()
{
  while (全ての upper リデックスについて) {
    BOB 間書換えを行う;
    if (フリーなセルが足りない) {
      この BOB を分割する;
    }
  }
  if (更に lowerBOB が必要) {
    BOB 間書換えのためにこの BOB を
    interRewritePool に入れ、必要な
    lower BOB を集めてもらう;
    return;
  }
  if (一つも BOB 間リデックスがなかった &&
      upper リデックスの sink からつながる
      BOB が全て正規化されている) {
    /* この BOB は正規化された */
    BOB 間書換えに使った lowerBOB を解放
    (normalizedPool 又は queue に入れる);
    この BOB を normalizedPool に入れる;
    return;
  }
  BOB 間書換えに使った lowerBOB を解放;
  この BOB を queue に入れる;
  rerturn;
}
```

6 実験

この章では、項書換え系の処理系に BOB とスレッドを用い、マルチプロセッサ計算機上で実行することにより計算時間が短縮されることを計算機実験によって定量的に示す。

6.1 準備

計算機実験を行なうために、BOB とスレッドを用いた項書換え系の並列処理系を作成した。この処理系は Mach オペレーティングシステム上で cthread ライブラリ [2] を用いて並列に動作する。処理系の作

成にはCとC++言語を用い、プログラムの規模はlexとyaccのソースを含めて7000行程度である。

実験は共有メモリ型4プロセッサ計算機(OMRON LUNA88K, Memory 64MByte)をプロセッサ数を1～4で変化させ,

1. 複数のco-threadを発行する場合(Multi Thread)

2. co-threadを発行するがmain threadから切り離さない場合(main threadは待機状態となる, Dual Thread)

3. co-threadを発行しないで単なる関数呼び出しを行う場合(Single Thread)

の3通りを行った。

書換え規則は階乗計算(fact), フィボナッチ数列(fib), クイックソート(qsort+fact), マージソート(msort+fact)の4種類を用いた。クイックソートとマージソートは整数をソートする規則であり、ソートする整数をfactで生成した。

時間の計測にはUNIXのtimeコマンドを用いた。サンプルとして階乗の項書換え系を以下に示す。

```
fact(0) ▷ s(0)
fact(s(X)) ▷ mult(s(X), fact(X))
mult(0, Y) ▷ 0
mult(s(X), Y) ▷ add(Y, mult(X, Y))
add(0, Y) ▷ Y
add(s(X), Y) ▷ s(add(X, Y))
```

6.2 実験結果

実験結果の表を付録1、グラフを付録2に示す。

この結果から、4プロセッサでマルチスレッドを用いることにより、シングルスレッド時に比べ1.5～2倍の速度向上が見られた。

ここでDual ThreadとSingle Thread(どちらも並列動作はしていない)の差はスレッド発行のオーバーヘッドであると見ることが出来る。

7 まとめと今後の課題

共有メモリ型マルチプロセッサ計算機上で動作するBOB (Bundle of Branches) を用いた項書換え系の並列処理系の機構を示し、計算機実験によって4台のCPUに対して2倍程度計算速度が向上することを示した。

謝辞

御指導いただいた稲垣康善教授、坂部俊樹助教授、直井徹岐阜大助教授、杉野花津江助手、結縁祥治助手ならびに御討論いただいた論研究室の皆様に感謝します。またLUNA88Kを貸与して頂いたオムロン株式会社に感謝致します。

本研究は文部省科学研究費 重点領域(1)(03235102)の補助を受けた。

参考文献

[1] Huet,G.: "Confluent Reductions: Abstract properties and applications to term rewriting systems", JACM,Vol.27, No.4(1980).

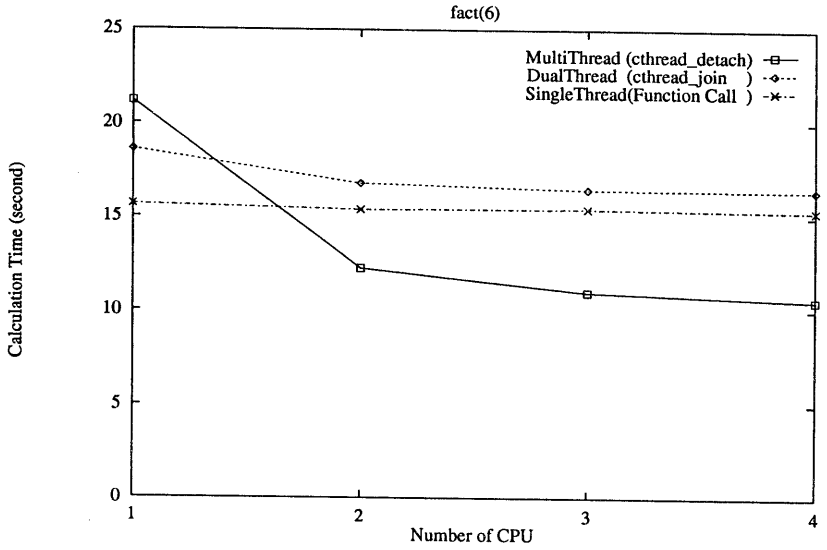
[2] E.C.Cooper, R.P.Draves, "C Thread", Draft, Dep. of Computer Science Caenegie Mellon U. Pittsburgh, Pennsylvania, Oct. 17, 1988

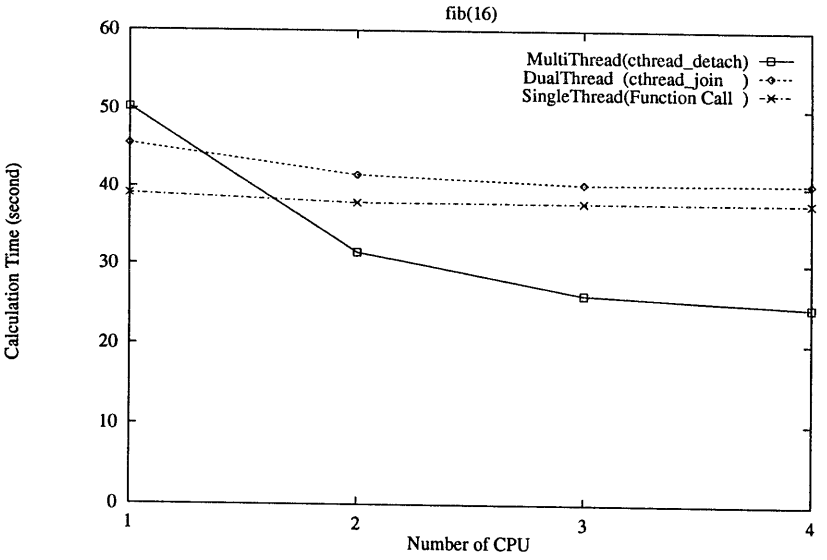
[3] 山本晋一郎, 直井徹, 坂部俊樹, 稲垣康善 "並列分散TRSシミュレータの実現について", 情処研報 SF38-10,1991

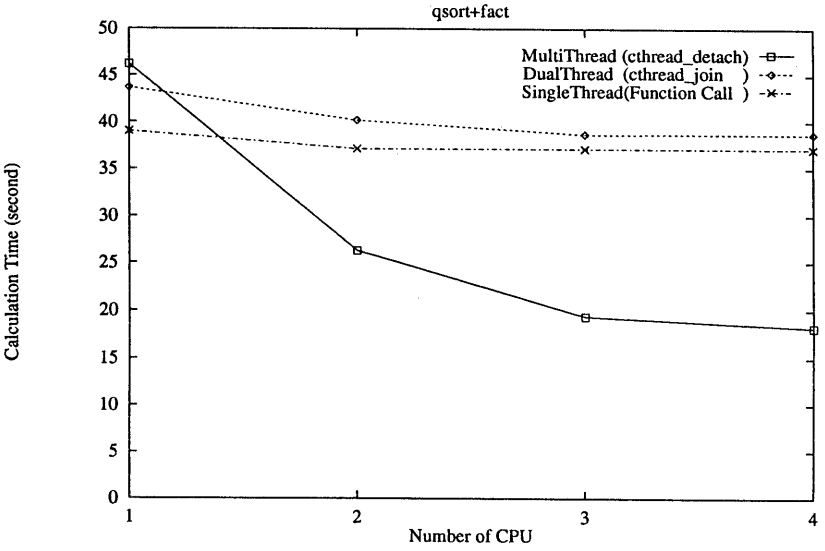
付録 1

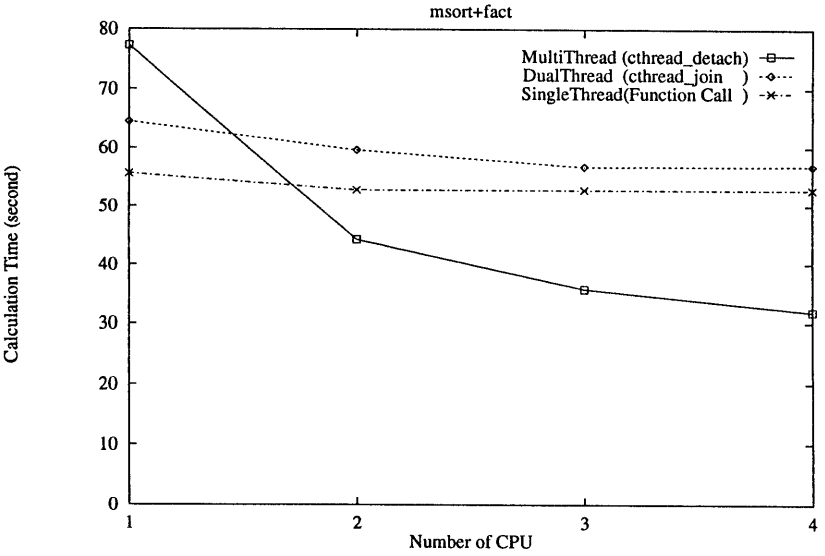
1CPU	計算時間の平均 (秒)	使用した BOB 数の平均	書換え回数の平均
Multi Thread	21.2	37	10697
fact(6) Dual Thread	18.6	36	10684
Single Thread	15.7	36	10684
Multi Thread	50.2	60	15624
fib(16) Dual Thread	45.5	63	15624
Single Thread	39.1	63	15624
Multi Thread	18.2	91.8	27150.4
qsort+fact Dual Thread	38.7	95	25354
Single Thread	37.1	95	25354
Multi Thread	31.8	92.6	25908.4
msort+fact Dual Thread	56.9	101	25940
Single Thread	52.8	101	25940
2CPU	計算時間の平均 (秒)	使用した BOB 数の平均	書換え回数の平均
Multi Thread	12.2	34.6	10699
fact(6) Dual Thread	16.8	36	10684
Single Thread	15.4	36	10684
Multi Thread	31.5	60	15624
fib(16) Dual Thread	41.6	63	15624
Single Thread	37.9	63	15624
Multi Thread	18.2	91.8	27150.4
qsort+fact Dual Thread	38.7	95	25354
Single Thread	37.1	95	25354
Multi Thread	31.8	92.6	25908.4
msort+fact Dual Thread	56.9	101	25940
Single Thread	52.8	101	25940
3CPU	計算時間の平均 (秒)	使用した BOB 数の平均	書換え回数の平均
Multi Thread	11.0	35.4	10691
fact(6) Dual Thread	16.5	36	10684
Single Thread	15.4	36	10684
Multi Thread	26.1	60	15624
fib(16) Dual Thread	40.3	63	15624
Single Thread	37.9	63	15624
Multi Thread	18.2	91.8	27150.4
qsort+fact Dual Thread	38.7	95	25354
Single Thread	37.1	95	25354
Multi Thread	31.8	92.6	25908.4
msort+fact Dual Thread	56.9	101	25940
Single Thread	52.8	101	25940
4CPU	計算時間の平均 (秒)	使用した BOB 数の平均	書換え回数の平均
Multi Thread	10.5	36	10686
fact(6) Dual Thread	16.4	36	10684
Single Thread	15.3	36	10684
Multi Thread	24.6	60	15624
fib(16) Dual Thread	40.4	63	15624
Single Thread	37.9	63	15624
Multi Thread	18.2	91.8	27150.4
qsort+fact Dual Thread	38.7	95	25354
Single Thread	37.1	95	25354
Multi Thread	31.8	92.6	25908.4
msort+fact Dual Thread	56.9	101	25940
Single Thread	52.8	101	25940

付録 2


fact(6)
MultiThread (cthread_detach)
DualThread (cthread_join)
SingleThread(Function Call)
Calculation Time (second)
Number of CPU
0
5
10
15
20
25
1
2
3
4


fib(16)
MultiThread(cthread_detach)
DualThread (cthread_join)
SingleThread(Function Call)
Calculation Time (second)
Number of CPU
0
10
20
30
40
50
60
1
2
3
4


qsort+fact
Calculation Time (second)
Number of CPU
MultiThread (cthread_detach)
DualThread (cthread_join)
SingleThread(Function Call)
0
5
10
15
20
25
30
35
40
45
50
1
2
3
4


msort+fact
Calculation Time (second)
Number of CPU
MultiThread (cthread_detach)
DualThread (cthread_join)
SingleThread(Function Call)
0
10
20
30
40
50
60
70
80
1
2
3
4