

スレッドローカル記憶テーブル

Maple 17 では、[記憶](#)テーブルがデフォルトでスレッドローカルになりました。記憶テーブルを使用するプロシージャが複数のスレッドから呼び出された場合、格納された結果は、その結果を作成したスレッドでのみ利用できます。これは、従来の Maple バージョンの動作とは異なります。従来は、記憶テーブルがスレッド間で共有されていたため、1 つのスレッドでテーブルが変更されると、別のスレッドにも影響が生じていました。

この変更によって、記憶テーブルをどのように使用するにかかわらず、1 つのスレッドでの実行時に動作する記憶テーブルを使用するプロシージャが、並列での実行時でも動作するようになりました。従来の Maple バージョンでは、プロシージャが記憶テーブルを操作したことにより、プロシージャの並列実行の挙動が失敗になることがありました。

以下のプロシージャは、Fibonacci と似たようなシーケンスを実行します。各項は前の 2 つの項の合計ですが、ユーザが初期値を指定することができます。このプロシージャの処理効率を改善するために、記憶テーブルが使用されます。初期値は FibsHelper の記憶テーブルに格納され、漸化式の終了条件となります。

```
> FibsHelper := proc (n::nonnegint)
    option remember;
    FibsHelper(n-1)+FibsHelper(n-2)
end proc;
FibsHelper := proc(n::nonnegint)
    option remember, FibsHelper(n - 1) + FibsHelper(n - 2)
end proc (1)
```

```
> Fibs := proc (n::nonnegint, t1 := 1, t2 := 1)
    subsop(4 = NULL, op(FibsHelper));
    FibsHelper(0) := t1;
    FibsHelper(1) := t2;
    FibsHelper(n)
end proc;
Fibs := proc(n::nonnegint, t1 := 1, t2 := 1)
    subsop(4 = NULL, op(FibsHelper));
    FibsHelper(0) := t1;
    FibsHelper(1) := t2;
    FibsHelper(n)
end proc (2)
```

```
> seq(Fibs(i), i = 0 .. 10);
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 (3)
```

```
> seq(Fibs(i, 2, 2), i = 0 .. 10);
2, 2, 4, 6, 10, 16, 26, 42, 68, 110, 178 (4)
```

```
[ > seq(Fibs(i, 2, 38), i = 0 .. 10);
      2, 38, 40, 78, 118, 196, 314, 510, 824, 1334, 2158 (5)
```

このプロシージャはシングルスレッドで実行する場合に動作します。また、Maple 17 では並列実行でも動作します。

```
[ > task := proc (n, t1, t2) local i; [seq(Fibs(i, t1, t2), i = 0 .. 10)
  ] end proc;
      task := proc(n, t1, t2) local i; [seq(Fibs(i, t1, t2), i=0..10)] end proc (6)
```

```
[ > Threads:-Task:-Start(passed, Tasks = [task, [10, 1, 1], [10, 2, 2],
  [10, 2, 38]]);
  [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89], [2, 2, 4, 6, 10, 16, 26, 42, 68, 110, 178], [2, 38, 40, 78,
  118, 196, 314, 510, 824, 1334, 2158] (7)
```

従来の Maple バージョンでは、並列での評価は失敗する可能性があります。

shared オプションを指定することで、共有記憶テーブルを使用するプロシージャ (Maple の従来の挙動) を作成することができます。以下のプロシージャは、シングルスレッドでは動作しますが、並列では動作しません。

```
[ > SharedFibsHelper := proc (n::nonnegint)
  options remember, shared;
  SharedFibsHelper(n-1)+SharedFibsHelper(n-2)
  end proc;
  SharedFibsHelper := proc(n::nonnegint)
    option remember, shared;
    SharedFibsHelper(n - 1) + SharedFibsHelper(n - 2)
  end proc (8)
```

```
[ > SharedFibs := proc (n::nonnegint, t1 := 1, t2 := 1)
  subsop(4 = NULL, op(SharedFibsHelper));
  SharedFibsHelper(0) := t1;
  SharedFibsHelper(1) := t2;
  SharedFibsHelper(n)
  end proc;
  SharedFibs := proc(n::nonnegint, t1 := 1, t2 := 1)
    subsop(4 = NULL, op(SharedFibsHelper));
    SharedFibsHelper(0) := t1;
    SharedFibsHelper(1) := t2;
    SharedFibsHelper(n)
  (9)
```

`end proc`

```
> seq(SharedFibs(i), i = 0 .. 10);  
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 (10)
```

```
> seq(SharedFibs(i, 2, 2), i = 0 .. 10);  
2, 2, 4, 6, 10, 16, 26, 42, 68, 110, 178 (11)
```

```
> seq(SharedFibs(i, 2, 38), i = 0 .. 10);  
2, 38, 40, 78, 118, 196, 314, 510, 824, 1334, 2158 (12)
```

SharedFibs が並列実行で呼び出された場合、不正確な結果が計算されます。ただし、並列実行の性質により、正しい結果が表示されることもあります。

```
> sharedTask := proc (n, t1, t2)  
  local i;  
  [seq(SharedFibs(i, t1, t2), i = 0 .. 10)]  
end proc;  
sharedTask := proc(n, t1, t2) local i; [seq(SharedFibs(i, t1, t2), i=0..10)] end proc (13)
```

```
> Threads:-Task:-Start(passed, Tasks = [sharedTask, [10, 1, 1], [10,  
  2, 2], [10, 2, 38]]);  
[1, 1, 2, 3, 5, 8, 13, 21, 34, 650, 89], [2, 2, 4, 6, 10, 16, 26, 42, 68, 110, 178], [2, 38, 40, 78,  
  118, 196, 314, 510, 34, 1334, 2158] (14)
```

間違いの可能性は様々あります。1つのスレッドで記憶テーブルをクリアすることで、別のスレッドの初期値が削除されることがあります。これにより、SharedFibsHelperが無効な値(マイナスの数値)で呼び出されます。1つのスレッドの初期値を設定することによって、別のスレッドが上書きされることがあります。これにより、そのスレッドで間違った結果が計算されることがあります。1つのスレッドで記憶された値を別のスレッドでも使用できますが、正しくない値が記憶されることもあります。

共有記憶テーブルを使用するプロシージャが並列で正しく動作し、計算結果をスレッド間で共有できる場合があります。ただし、共有記憶テーブルを使用するプロシージャの記述は複雑で、注意が必要になります。Maple 17のスレッドローカル記憶テーブルを使用すれば、シングルスレッドでの実行用に記述されたプロシージャを簡単に並列実行で動作するようにできます。

参照

[remember](#)