OpenMP* 検証サンプル taskloop 初級編

このセッションは、「インテル® コンパイラーを使用した OpenMP* 5.0 による並列プログラミング・セミナー【パート 1】」で公開されているオンラインセミナーの補足セッションです

内容

- ワークシェアを taskloop に移行する利点はあるか
  - 典型的な pi の計算の例
  - ループ反復の負荷が均一ではない prime の例
  - 並列構造の入れ子

このセッションは、
「**インテル® コンパイラーを使用した OpenMP* 5.0 による並列プログラミング･セミナー【パート 1】**」
で公開されているオンラインセミナーの補足セッションです

動画を再生する際、お客様情報を入力していただく必要がありますので、予めご了承ください

ここで使用する環境

- Windows* 10 Pro 1909
- インテル® Core™ i7-9700K プロセッサー (8 コア、8 スレッド)、メモリー 16GB
- インテル® C++ コンパイラー 19.1 Update 2 (VS2017 ライブラリー)
- インテル® VTune™ プロファイラー 2020 Update2 (日本語パッケージ)

taskloop ディレクティブ

#pragma omp taskloop*[節[[,]節] ...]*
for ループ

1 つ以上の関連するループ反復を OpenMP* タスクを使用して並列に実行します

節:
shared(list)、default(shared | none)
private(*list*)、 firstprivate(*list*)、lastprivate(*list*)
reduction(*[default,][reduction-identifier: list)]*
in_reduction(*reduction-identifier: list*)
grainsize(*grain-size*)、num_tasks(*num_tasks*)
collapse(n)、priority(*priority-value*)
untied、mergeable、nogroup
allocate(*[allocator:] list*)
if(*[*taskloop*:]scalar-expression*)
final(*scalar-expression*)

赤字は OpenMP* 5.0 で追加

https://www.isus.jp/products/c-compilers/openmp-ref-5-0-0519-released/

parallel for と taskloop の動作を確認 (pi)

for ワークシェアを単純に taskloop にする利点はあるのか?

```
#ifdef TASKLOOP
   #pragma omp parallel
   #pragma omp single
   #pragma omp taskloop simd reduction(+:sum) private(x)
#else
   #pragma omp parallel for simd reduction(+:sum) private(x)
#endif
     for (i=0;i< num_steps; i++){
       x = (i+0.5)*step;
       sum = sum + 4.0/(1.0+x*x);
     }
   pi = step * sum;
```

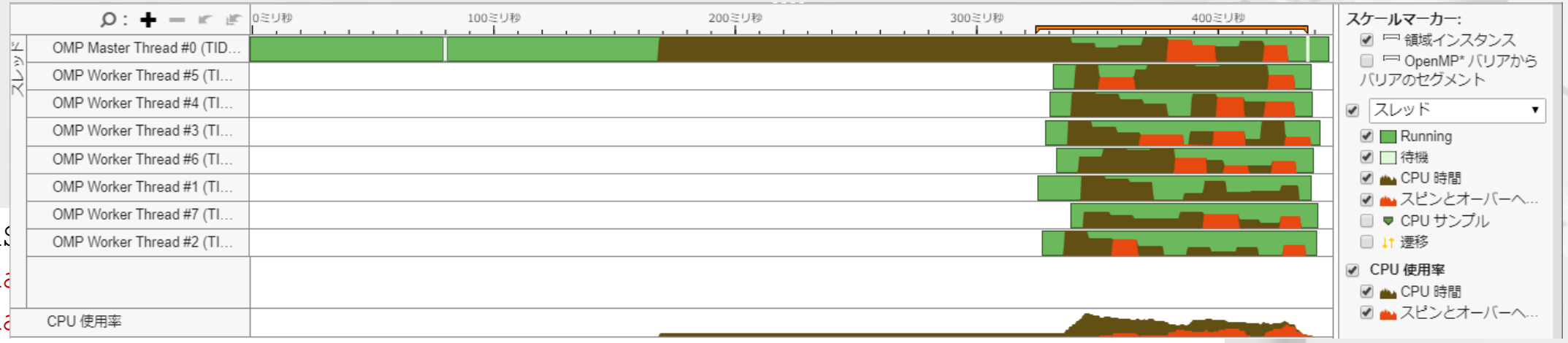
Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> pi
Pi = 3.141592653589793
Pi = 3.141593 Time = 0.065000

> pi_taskloop
Pi = 3.141592653589793
Pi = 3.141593 Time = 0.068000
```

デフォルトのスレッド数 = 論理コア数

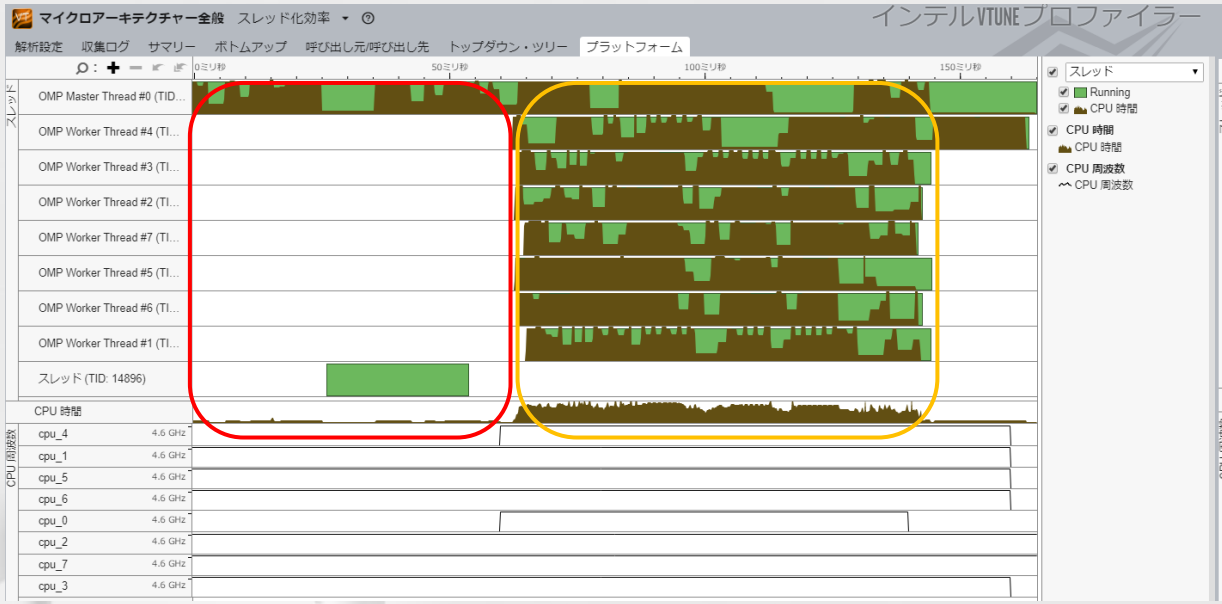
parallel for と taskloop の動作を確認 (pi)



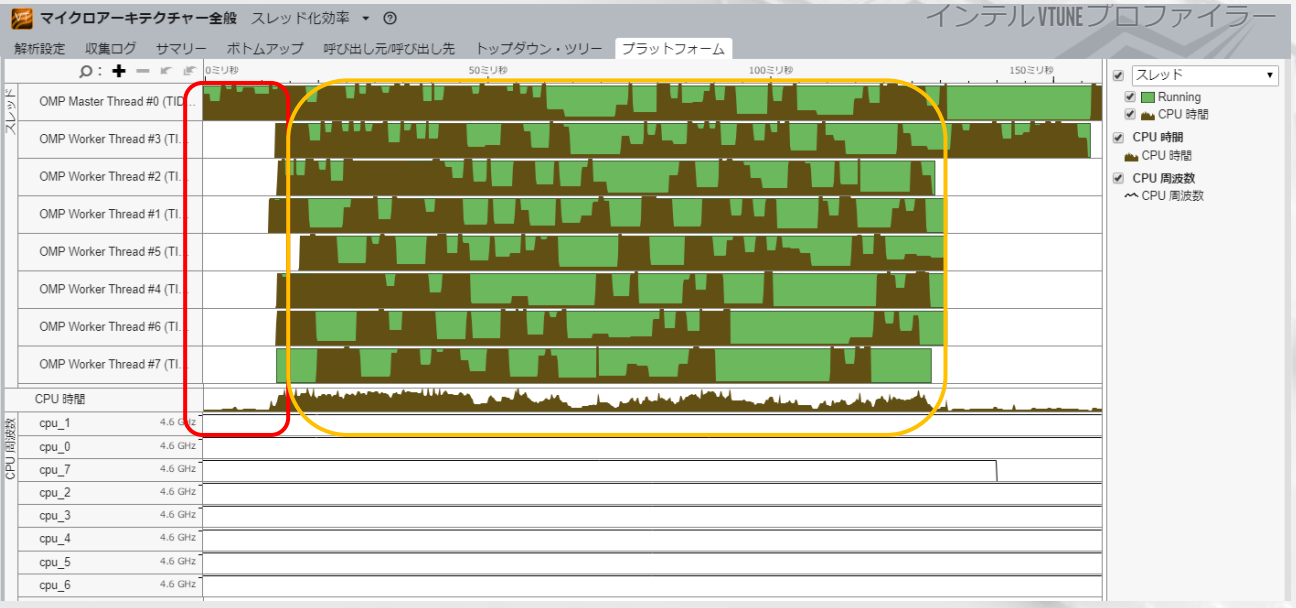

```
#ifdef TAS
   #pragma
   #pragma
   #pragma omp taskloop simd reduction(+:sum) private(x)
#else
   #pragma omp parallel for simd reduction(+:sum) private(x)
#endif
      for (
        x =
        sum
      }
   pi = st
```

インテル® VTune™ プロファイラーのタイムラインで確認

parallel for ワークシェアのタイムライン



taskloop のタイムライン



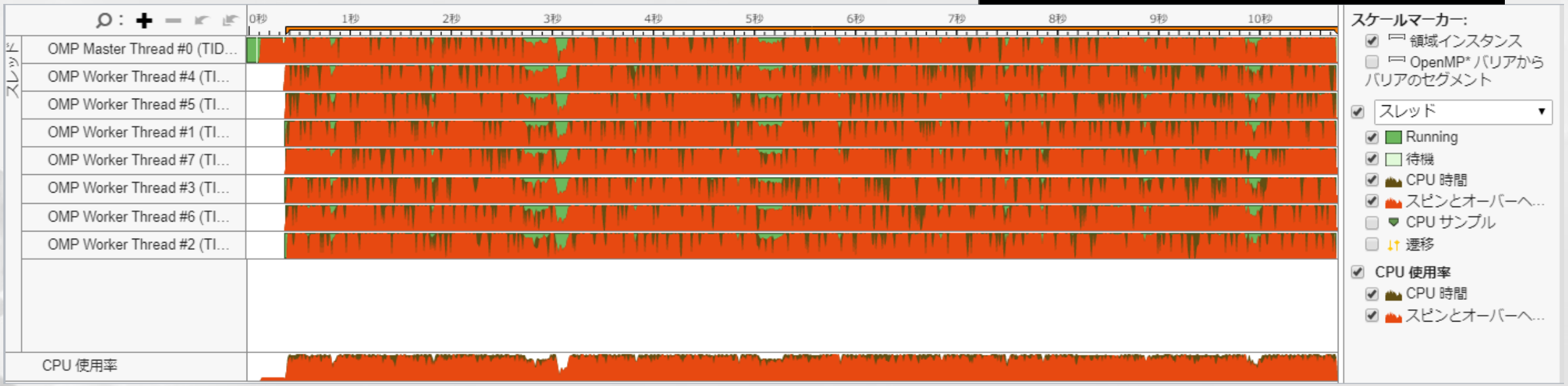
- parallel for ではスレッドのチームを構成するのに時間を要しているように見える
- taskloop は CPU 時間が連続する期間が短い
- taskloop の GRAINSIZE は自動的に調整される
- parallel for の for ループはデフォルトで (ループサイズ/スレッド数) でブロック化される

taskloop の粒度を強制 (サイズ 10)

```
#pragma omp parallel
   #pragma omp single
   #pragma omp taskloop simd reduction(+:sum) private(x) grainsize(10)
for (i=0;i< num_steps; i++){
       x = (i+0.5)*step;
       sum = sum + 4.0/(1.0+x*x);
     }
```

Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> pi_taskloop_gz10
Pi = 3.141592653589793
Pi = 3.141593 Time = 11.031000
```

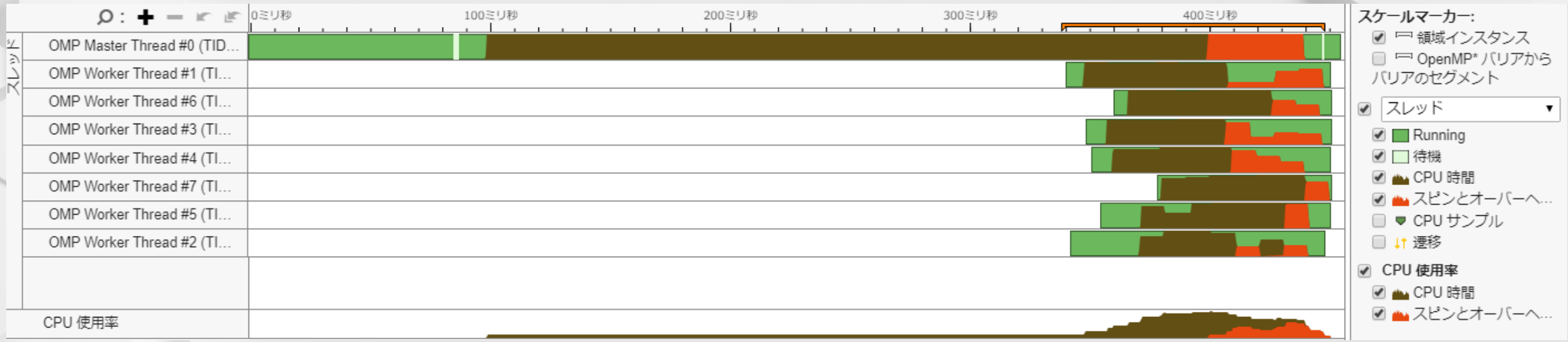


taskloop の粒度を強制 (ループサイズ/スレッド数)

```
#pragma omp parallel
   #pragma omp single
   #pragma omp taskloop simd reduction(+:sum) private(x)
for (i=0;i< num_steps; i++){          grainsize(num_steps/omp_get_num_threads())
       x = (i+0.5)*step;
       sum = sum + 4.0/(1.0+x*x);
}
```

Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> pi_taskloop_gzauto
Pi = 3.141592653589793
Pi = 3.141593 Time = 0.071000
```



taskloop の利点　OMP_NUM_THREADS=16 のケース

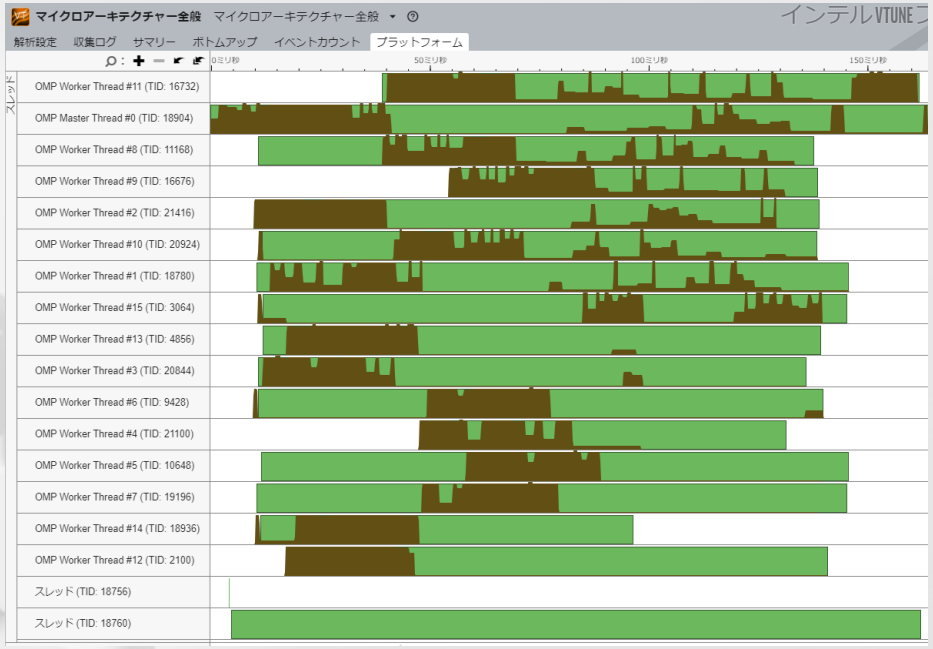
Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

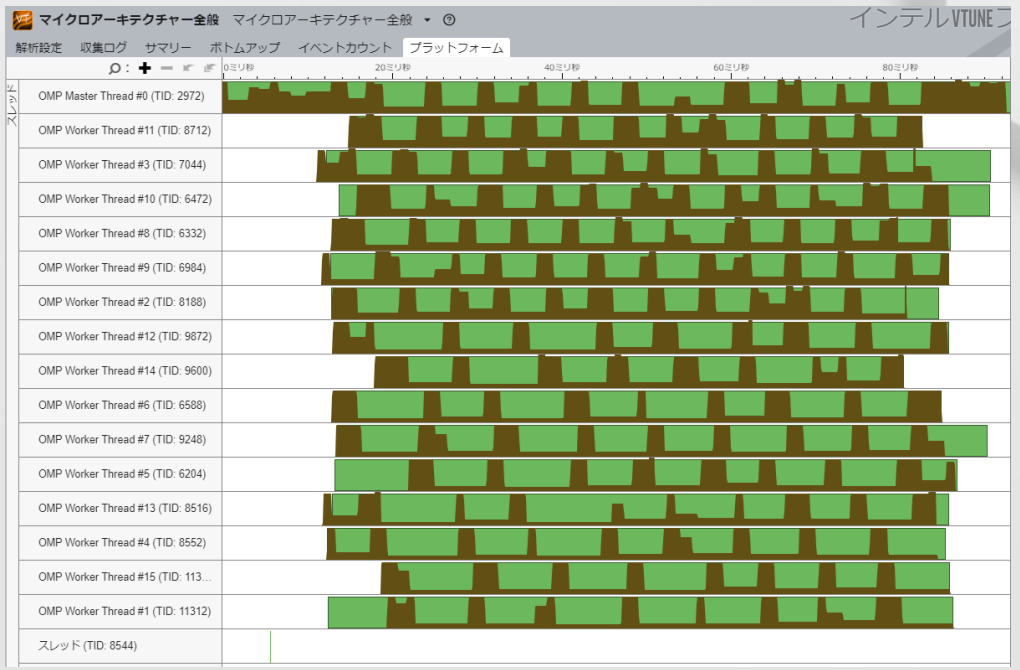
```
> set OMP_NUM_THREADS=16

> pi
Pi = 3.141592653589793
Pi = 3.141593 Time = 0.089000

> pi_taskloop
Pi = 3.141592653589793
Pi = 3.141593 Time = 0.071000
```

- for ワークシェアでは、16 個のスレッドにループが均等に分割され、スレッドは OS のスケジューラーでスケジュールされる (スレッド過度の状態)
- taskloop では、生成された 16 個のスレッドが OS スケジューラーでスケジュールされアクティブなときにキューからタスクを取得して実行 (タスクがスケジュール･アウトされる可能性が低い)





prime を parallel for と taskloop で実行

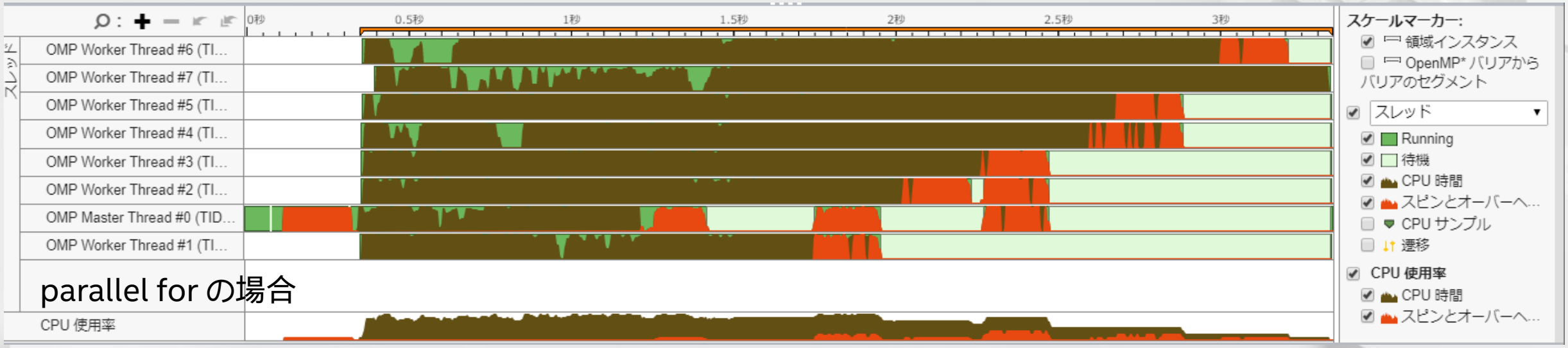
```
#ifdef TASKLOOP
  #pragma omp parallel
  #pragma omp single
  #pragma omp taskloop reduction(+:gPrimesFound)
#else
  #pragma omp parallel for reduction(+:gPrimesFound)
#endif
   for( int i = start; i <= end; i+=2 ){
      if( TestForPrime(i) )
                        gPrimesFound++;
   }
}
```

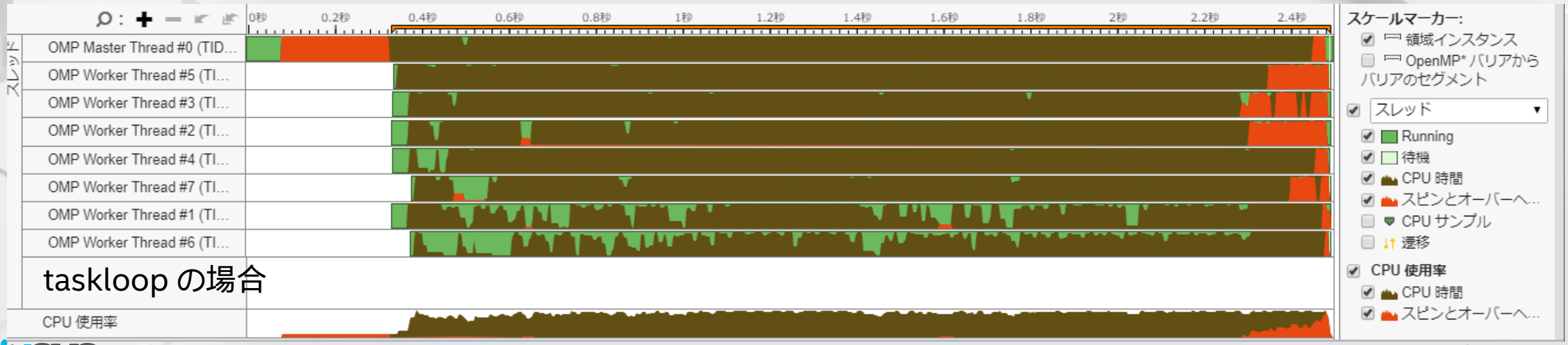
変数 end は 50000000 であり、潜在的に TestForPrime 関数は大きな値を評価するほど時間がかかります

管理者: Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> prime
(prime) 1から50000000までの範囲の素数の数は 3001135個。計算時間2.64秒
> prime_taskloop
(prime_taskloop) 1から50000000までの範囲の素数の数は 3001135個。計算時間2.09秒
```

for ワークシェアと taskloop の違い





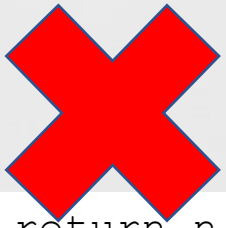
for と taskloop の入れ子の並列処理構造

```
main(){
  …
   for(int i=0; i<MAX; i++)
        re[i]=fib(i);
  …
}

int fib(int n){
int x, y;
  if(n<2) return n;
  x = fib(n-1);
  y = fib(n-2);
  return x+y;
}
```

for ループを並列実行できない?

```
#pragma omp parallel for
 for(int i=0; i<MAX; i++)
      re[i]=fib(i);
```



```
 if(n<2) return n;
#pragma omp single
  {
#pragma omp task shared(x)
  x = fib(n-1);
#pragma omp task shared(y)
  y = fib(n-2);
   }
#pragma omp taskwait
return x+y;
```

再帰呼び出しを並列実行できない?

```
#pragma omp parallel
#pragma omp single
#pragma omp taskloop
 for(int i=0; i<MAX; i++)
      re[i]=fib(i);
```

```
 if(n<2) return n;
#pragma omp task shared(x)
  x = fib(n-1);
#pragma omp task shared(y)
  y = fib(n-2);
#pragma omp taskwait
return x+y;
```

適切なタスク処理構造は?

```
main(){
  …
```

Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> fib_task
 The time to calculate 30 of fib = 0.242000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
> fib_loop
 The time to calculate 30 of fib = 0.292000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
> fib_group
 The time to calculate 30 of fib = 0.241000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
>
```

```
}
```

task と taskwait

```
#pragma omp parallel
#pragma omp single
for(int i=0; i<MAX; i++)
      re[i]=fib(i);
```

```
 if(n<2) return n;
#pragma omp task shared(x)
  x = fib(n-1);
#pragma omp task shared(y)
  y = fib(n-2);
#pragma omp taskwait
return x+y;
```

task と taskgroup

```
  #pragma omp parallel
  #pragma omp single
  #pragma omp taskloop
  for(int i=0; i<MAX; i++)
        re[i]=fib(i);
```

```
 if(n<2) return n;
#pragma omp taskgroup
  {
#pragma omp task shared(x)
  x = fib(n-1);
#pragma omp task shared(y)
  y = fib(n-2);
  }
return x+y;
```

適切なタスク処理構造は?

```
main(){
  …
```

Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> fib_task
 The time to calculate 30 of fib = 0.242000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
> fib_loop
 The time to calculate 30 of fib = 0.292000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
> fib_group
 The time to calculate 30 of fib = 0.241000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
>
```

```
}
```

```
ta                                                p
#pragma                                           el
#pragma
for(int                                           pp
      r                                           i++)

 if(n<2
#pragma                                           p
  x = f
#pragma                                           red(x)
  y = f
#pragma                                           red(y)
return
```

Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> fib_task
 The time to calculate 35 of fib = 2.570000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
78309 3524578 5702887
> fib_group
 The time to calculate 35 of fib = 2.635000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
78309 3524578 5702887
> fib_loop
 The time to calculate 35 of fib = 3.189000

Fibonacci number: 1 1 2 3 5 8 13 21 34 55 89 144
6765 10946 17711 28657 46368 75025 121393 196418
78309 3524578 5702887
>
```

まとめ

- 単純にループ･ワークシェア構造を taskloop に置き換えるメリットは少ない
  - 何らかの理由が必要
- 並列構造の入れ子では、ループ･ワークシェア構造とタスク構造を混在するのは避けるべき
- タスク構造を入れ子にする必要がある場合、taskgroup の利用も検討してください
- 並列構造の入れ子は注意深く構成する必要があります

iSUS

ソフトウェア・セミナー