Using OpenMP Tasking

Christian Terboven, Dirk Schmidl, RWTH Aachen University
{terboven, schmidl}@itc.rwth-aachen.de

EU H2020 Centre of Excellence (CoE)

1 October 2015 – 31 March 2018

Grant Agreement No 676553

OpenMP Tasking

- Introduced with OpenMP 3.0 in 2008

C/C++

```
#pragma omp task [clause]
... structured block ...
```

- Each encountering thread/task creates a new task
  - Code and data is being packaged up
  - Tasks can be nested
- Task barrier: `taskwait`
  - Encountering task is suspended until child tasks complete

C/C++

```
#pragma omp taskwait
```

Recursive computation of Fibonacci

```
int main(int argc,
         char* argv[])
{
   [...]
   fib(input);
   [...]
}
```

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

- **On the following slides we will show three approaches to parallelize this recursive code with Tasking.**

First version with Tasks (omp-v1)

```
int main(int argc,
         char* argv[])
{
   [...]
#pragma omp parallel
{
#pragma omp single
{
   fib(input);
}
}
   [...]
}
```

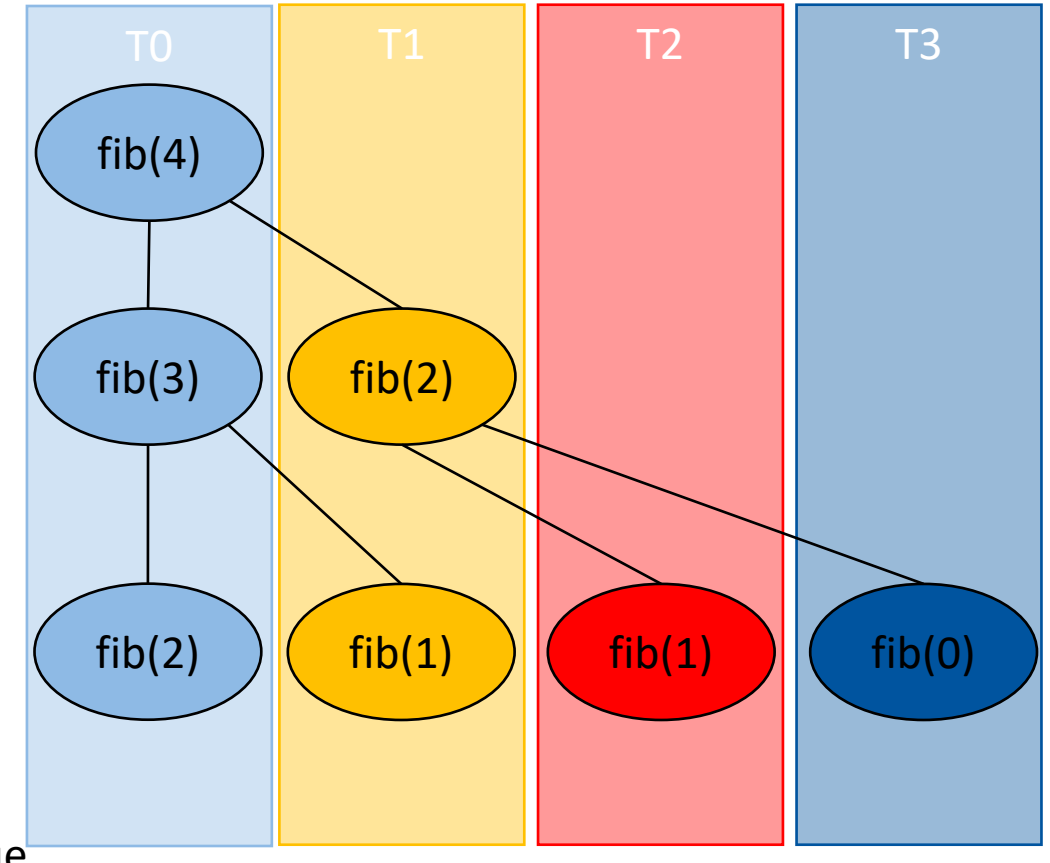
```
int fib(int n)   {
   if (n < 2) return n;
int x, y;
#pragma omp task shared(x)
{
   x = fib(n - 1);
}
#pragma omp task shared(y)
{
   y = fib(n - 2);
}
#pragma omp taskwait
   return x+y;
}
```

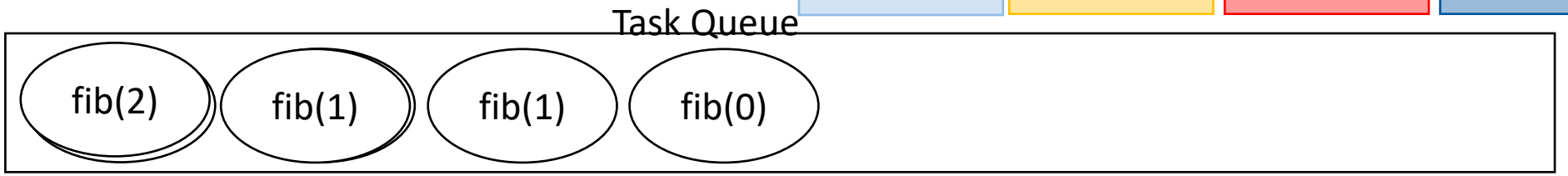
- **Only one Task / Thread enters `fib()` from `main()`, it is responsable for creating the two initial worker tasks**
- **Taskwait is required, as otherwise `x` and `y` inputs would be lost**

Dirk, how does that work in practice?

Illustration of Tasking

- T0 enters fib(4)
- T0 creates tasks for fib(3) and fib(2)
- T0 and T1 execute tasks from the queue
- T0 and T1 create 4 new tasks
- T0 - T3 execute tasks

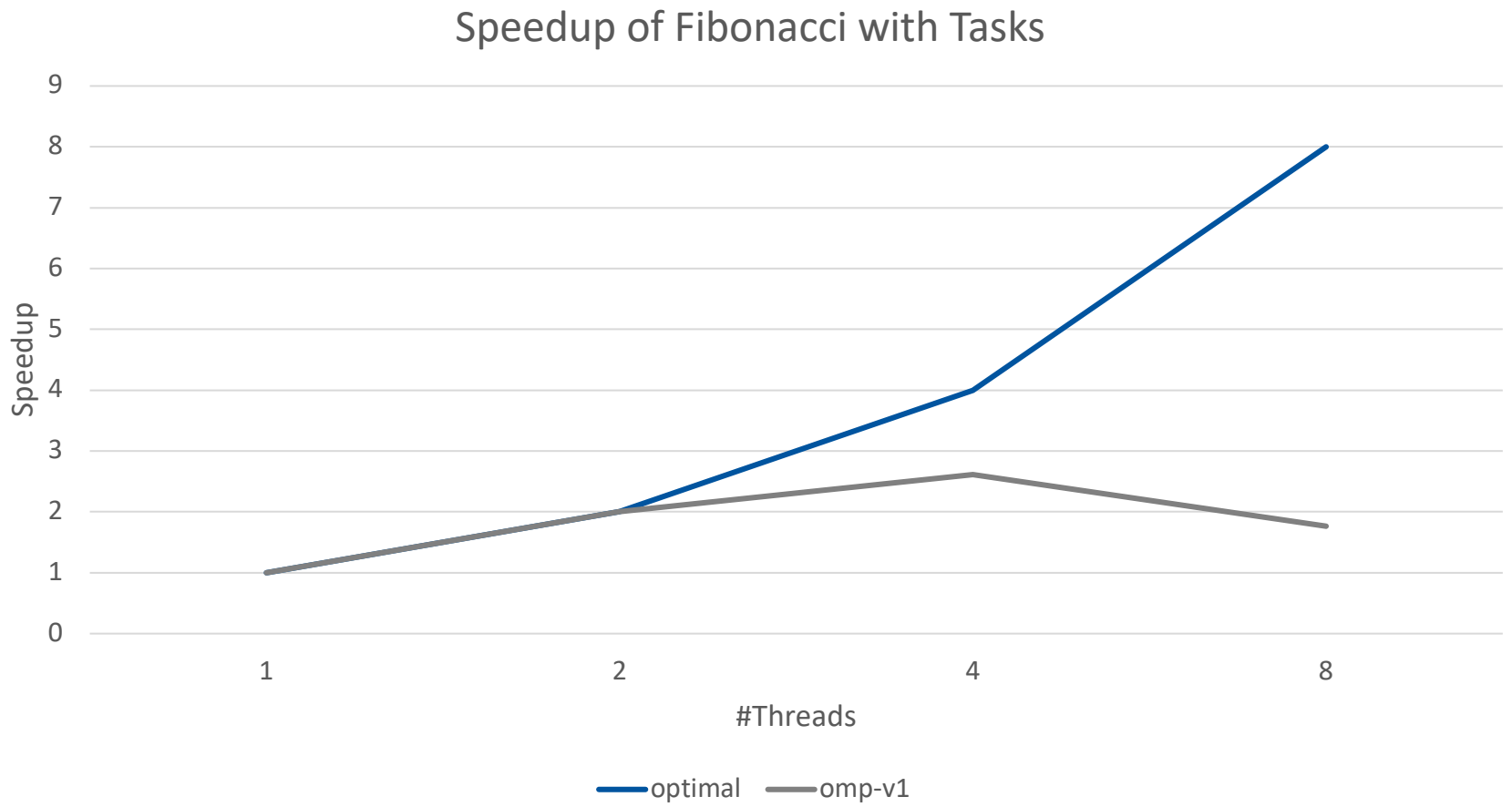




Got that.

Scalability Measurements (1/3)

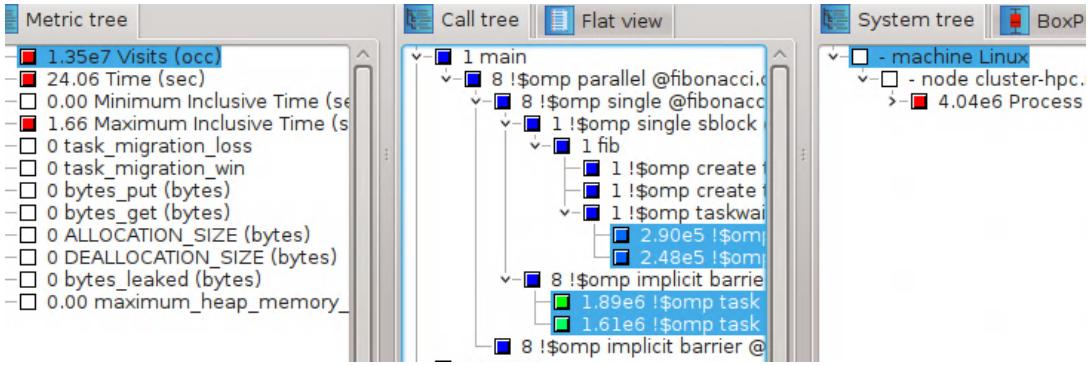
- **Overhead of task creation prevents better scalability!**



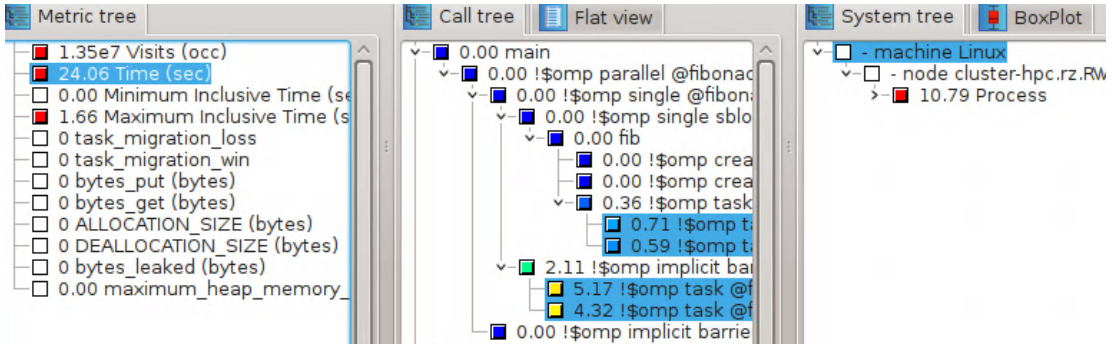
Christian, wait, let me explain!

Performance Analysis

Event-based profiling gives a good overview :



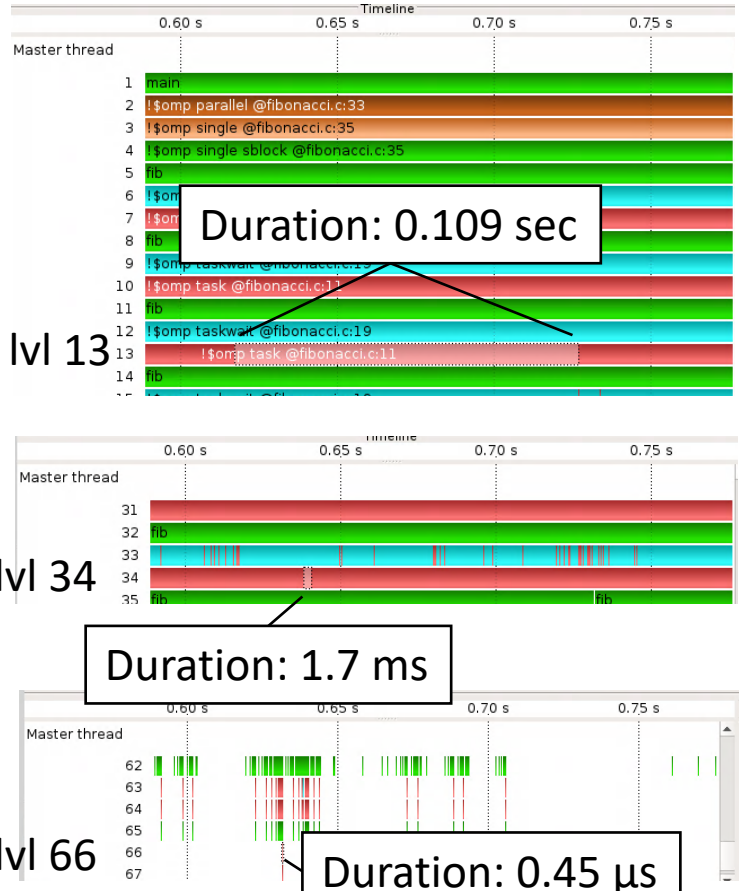
4.04 million threads are executed ...



... in ~10.79 seconds of CPU time.

=> average duration of a task is ~2.6 µs

Tracing gives more details:



Tasks get much smaller down the call-stack.

`if` Clause

- If the expression of an `if` clause on a task evaluates to `false`
  - The encountering task is suspended
  - The new task is executed immediately
  - The parent task resumes when the new task finishes

  → Used for optimization, e.g., avoid creation of small tasks

Second version with Tasks (omp-v2)

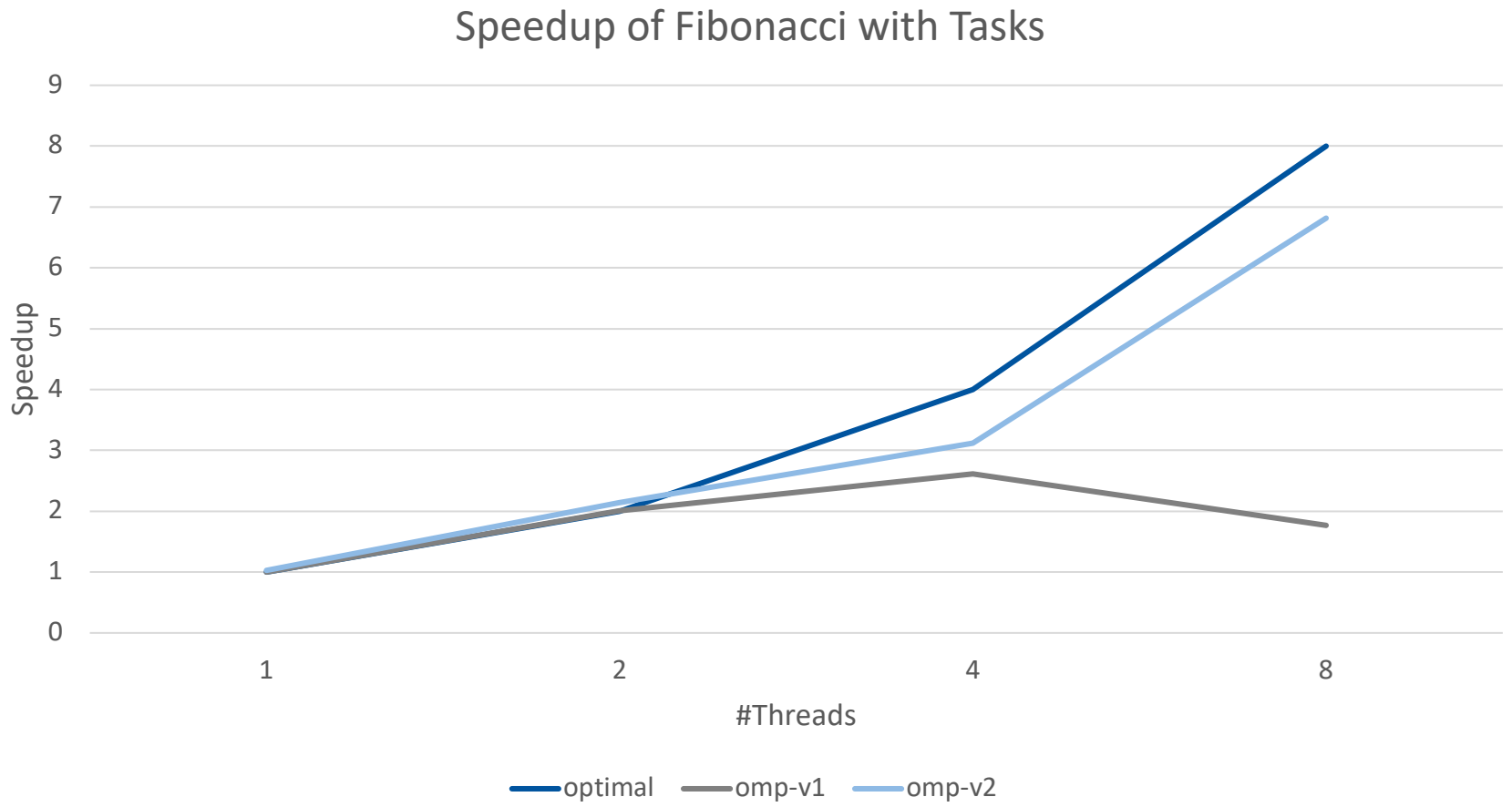
- **Improvement: Don‘t create yet another task once a certain (small enough) `n` is reached**

```
int main(int argc,
         char* argv[])
{
   [...]
#pragma omp parallel
{
#pragma omp single
{
   fib(input);
}
}
   [...]
}
```

```
int fib(int n)   {
   if (n < 2) return n;
int x, y;
#pragma omp task shared(x)
   \ if(n > 30)
{
   x = fib(n - 1);
}
#pragma omp task shared(y)
   \ if(n > 30)
{
   y = fib(n - 2);
}
#pragma omp taskwait
   return x+y;
}
```

Scalability Measurements (2/3)

- **Speedup is ok, but we still have some overhead when running with 4 or 8 threads**



Third version with Tasks (omp-v3)

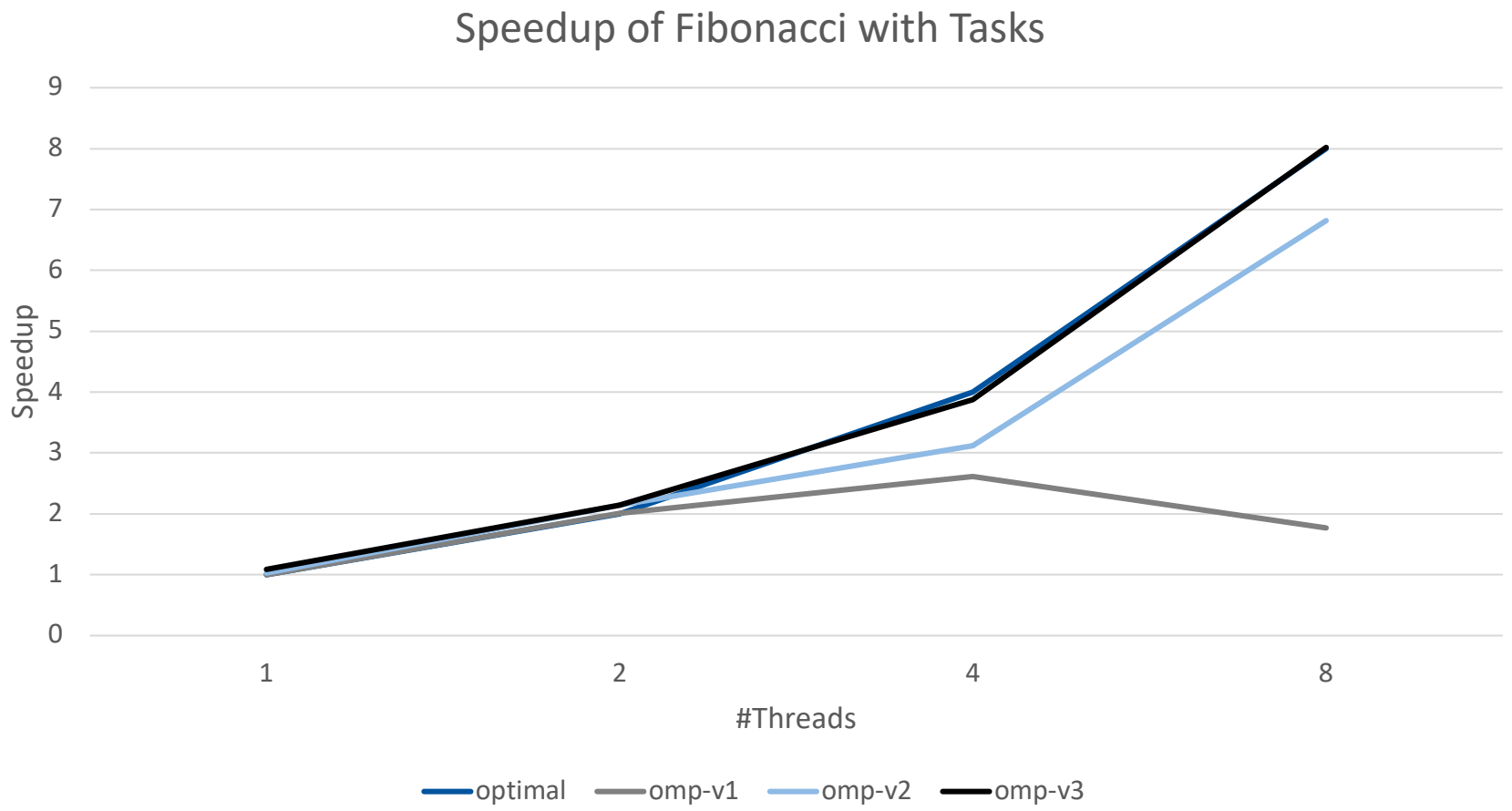
- **Improvement: Skip the OpenMP overhead once a certain `n` is reached (no issue w/ production compilers)**

```
int main(int argc,
         char* argv[])
{
   [...]
#pragma omp parallel
{
#pragma omp single
{
   fib(input);
}
}
   [...]
}
```

```
int fib(int n)   {
   if (n < 2) return n;
   if (n <= 30)
      return serfib(n);
int x, y;
#pragma omp task shared(x)
{
   x = fib(n - 1);
}
#pragma omp task shared(y)
{
   y = fib(n - 2);
}
#pragma omp taskwait
   return x+y;
}
```

Scalability Measurements (3/3)

- **Everything ok now ☺**



This also sounds interesting…

Example: taskloop

```
for ( i = 0; i<SIZE; i+=1) {
   A[i]=A[i]*B[i]*S;
}
```

blocking

taskloop

```
for ( i = 0; i<SIZE; i+=TS) {
   UB = SIZE < (i+TS)?SIZE:i+TS;
   for ( ii=i; ii<UB; ii++) {
      A[ii]=A[ii]*B[ii]*S;
   }
}
```

```
for ( i = 0; i<SIZE; i+=TS) {
   UB = SIZE < (i+TS)?SIZE:i+TS;
   #pragma omp task private(ii) \
    firstprivate(i,UB) shared(S,A,B)
   for ( ii=i; ii<UB; ii++) {
      A[ii]=A[ii]*B[ii]*S;
   }
}
```

```
#pragma omp taskloop grainsize(TS)
for ( i = 0; i<SIZE; i+=1) {
   A[i]=A[i]*B[i]*S;
}
```

- In manual transformation is difficult to determine grain
  - → 1 single iteration → to fine
  - → whole loop → no parallelism
- Apply blocking techniques
- taskloop: increase programmability

Example: task reductions

- Reduction operation
  - → perform some forms of recurrence calculations
  - → associative and commutative operators
- The (taskgroup) reduction clause

```
#pragma omp taskgroup task_reduction(op: list)
{structured-block}
```

  - → Register a new reduction at [1]
  - → Computes the final result after [3]
- The (task) in_reduction clause [participating]

```
#pragma omp task in_reduction(op: list)
{structured-block}
```

  - → Task participates in a reduction operation

```
int res = 0;
node_t* node = NULL;
...
#pragma omp parallel
{
  #pragma omp single
  {
    #pragma omp taskgroup task_reduction(+: res)
    { // [1]
      while (node) {
        #pragma omp task in_reduction(+: res) \
                         firstprivate(node)
        { // [2]
          res += node->value;
        }
        node = node->next;
      }
    } // [3]
  }
}
```

OpenMP 5.0

Example: task dependencies

- Task dependences as a way to define task-execution constraints

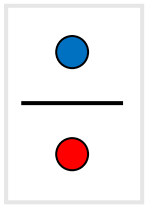
OpenMP 3.1

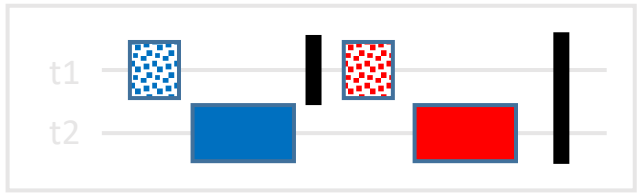
```
int x = 0;
#pragma omp parallel
#pragma omp single
{
#pragma omp task
  std::cout << x << std::endl;

  #pragma omp taskwait

#pragma omp task
  x++;
}
```

OpenMP 4.0

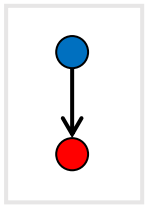
```
int x = 0;
#pragma omp parallel
#pragma omp single
{
#pragma omp task depend(in: x)
  std::cout << x << std::endl;



#pragma omp task depend(inout: x)
  x++;
}
```

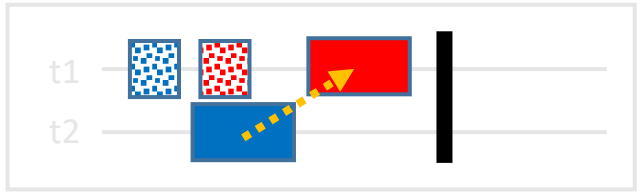
OpenMP 3.1





OpenMP 4.0





Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.