Programming OpenMP

Cut-off strategies

Christian Terboven

Example: Sudoku revisited

Parallel Brute-force Sudoku

- This parallel algorithm finds all valid solutions

	6						8	11			15	14			16
15	11				16	14				12			6		
13		9	12					3	16	14		15	11	10	
2		16		11		15	10	1							
	15	11	10			16	2	13	8	9	12				
12	13			4	1	5	6	2	3					11	10
5		6	1	12		9		15	11	10	7	16			3
	2				10		11	6		5			13		9
10	7	15	11	16				12	13						6
9						1			2		16	10			11
1		4	6	9	13			7		11		3	16		
16	14			7		10	15	4	6	1				13	8
11	10		15				16	9	12	13			1	5	4
		12		1	4	6		16				11	10		
		5		8	12	13		10			11	2			14
3	16			10			7			6				12	

- (1) Search an empty fie

first call contained in a
`#pragma omp parallel`
`#pragma omp single`
such that one tasks starts the execution of the algorithm

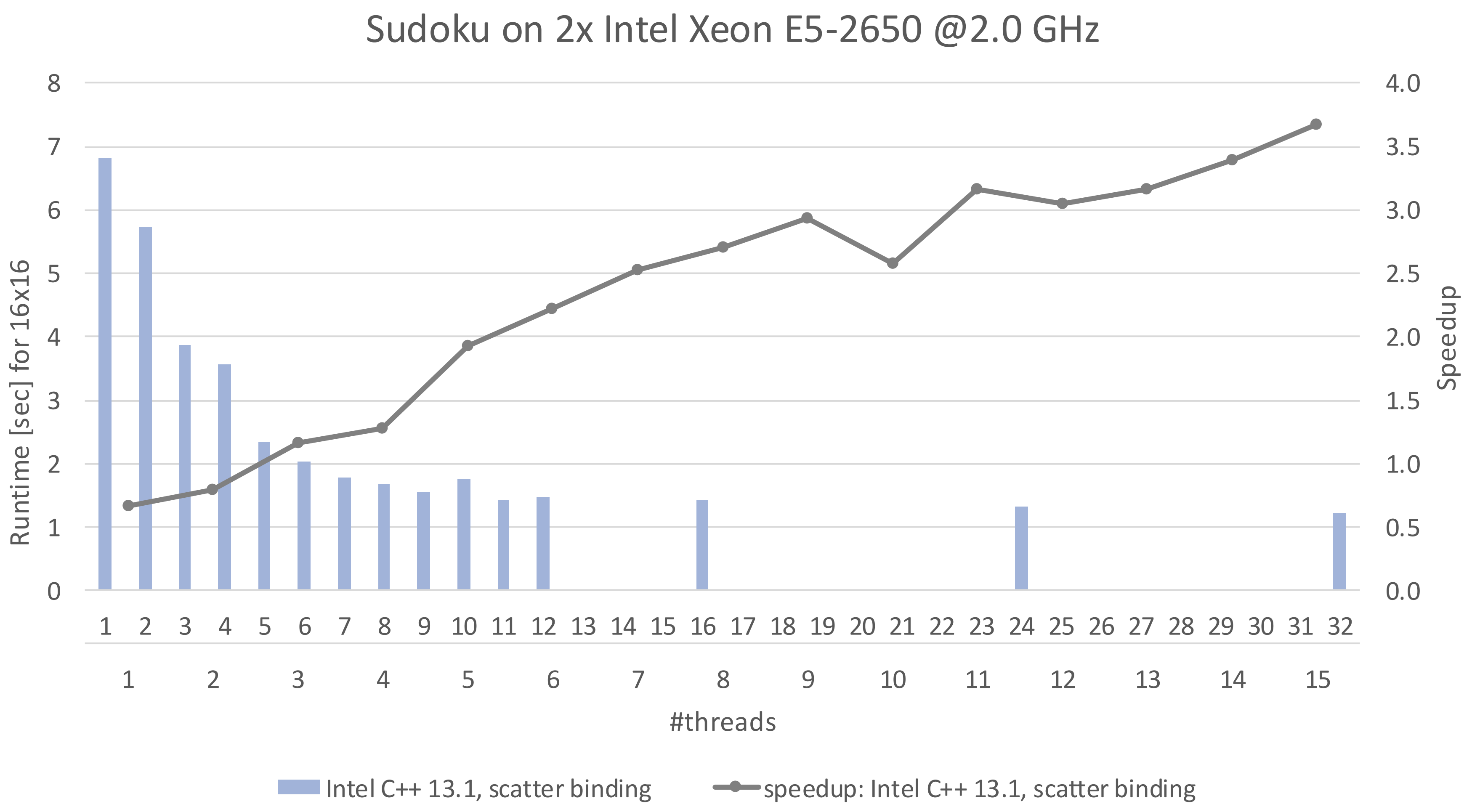
- (2) Try all numbers:
  - (2 a) Check Sudoku
    - If invalid: skip
    - If valid: Go to nex field

`#pragma omp task`
needs to work on a new copy of the Sudoku board

- Wait for completion

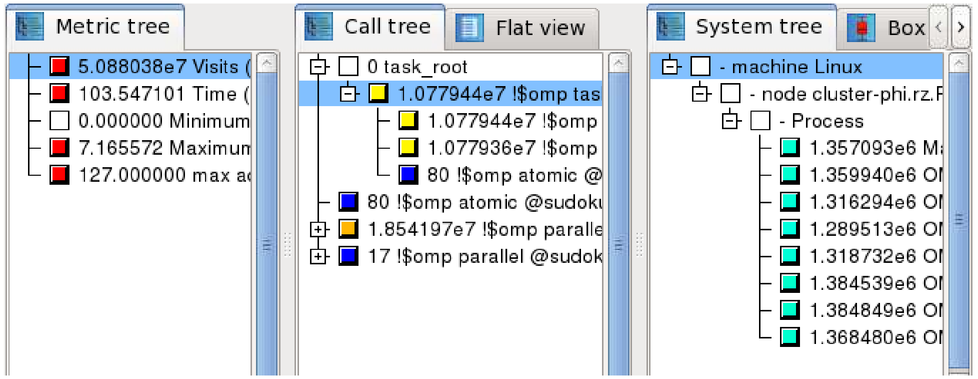
`#pragma omp taskwait`
wait for all child tasks

Performance Evaluation



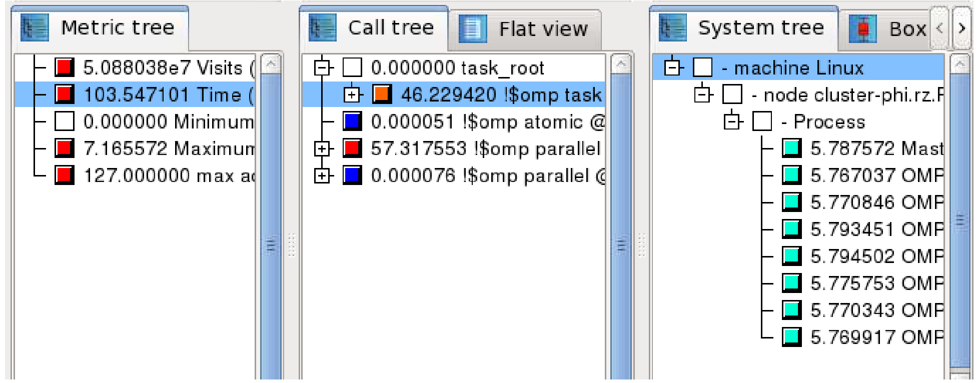
Performance Analysis

Event-based profiling provides a good overview :



```
Metric tree                  | Call tree   Flat view              | System tree   Box
5.088038e7 Visits (          | 0 task_root                        | - machine Linux
103.547101 Time (            | 1.077944e7 !$omp tas               | - node cluster-phi.rz.F
0.000000 Minimum             | 1.077944e7 !$omp                   | - Process
7.165572 Maximum             | 1.077936e7 !$omp                   | 1.357093e6 Ma
127.000000 max ac            | 80 !$omp atomic @                  | 1.359940e6 OI
                             | 80 !$omp atomic @sudoku            | 1.316294e6 OI
                             | 1.854197e7 !$omp paralle           | 1.289513e6 OI
                             | 17 !$omp parallel @sudok           | 1.318732e6 OI
                             |                                    | 1.384539e6 OI
                             |                                    | 1.384849e6 OI
                             |                                    | 1.368480e6 OI
```

Every thread is executing ~1.3m tasks…

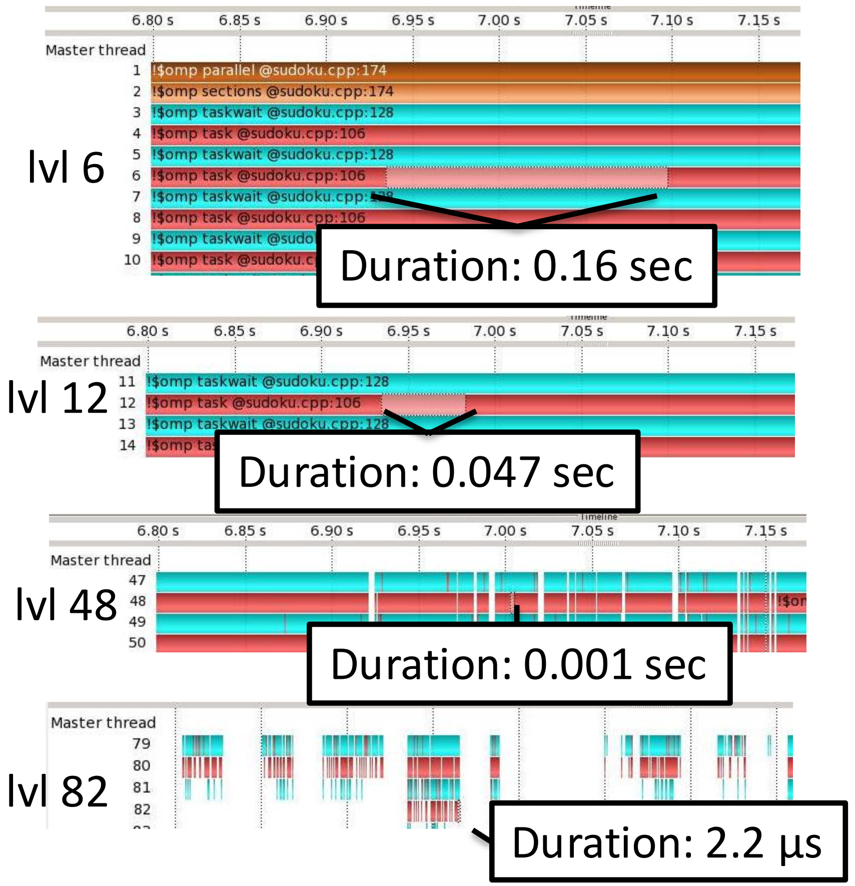


```
Metric tree                  | Call tree   Flat view              | System tree   Box
5.088038e7 Visits (          | 0.000000 task_root                 | - machine Linux
103.547101 Time (            | 46.229420 !$omp task               | - node cluster-phi.rz.F
0.000000 Minimum             | 0.000051 !$omp atomic @            | - Process
7.165572 Maximum             | 57.317553 !$omp parallel           | 5.787572 Mast
127.000000 max ac            | 0.000076 !$omp parallel @          | 5.767037 OMP
                             |                                    | 5.770846 OMP
                             |                                    | 5.793451 OMP
                             |                                    | 5.794502 OMP
                             |                                    | 5.775753 OMP
                             |                                    | 5.770343 OMP
                             |                                    | 5.769917 OMP
```

… in ~5.7 seconds.

=> average duration of a task is ~4.4 μs

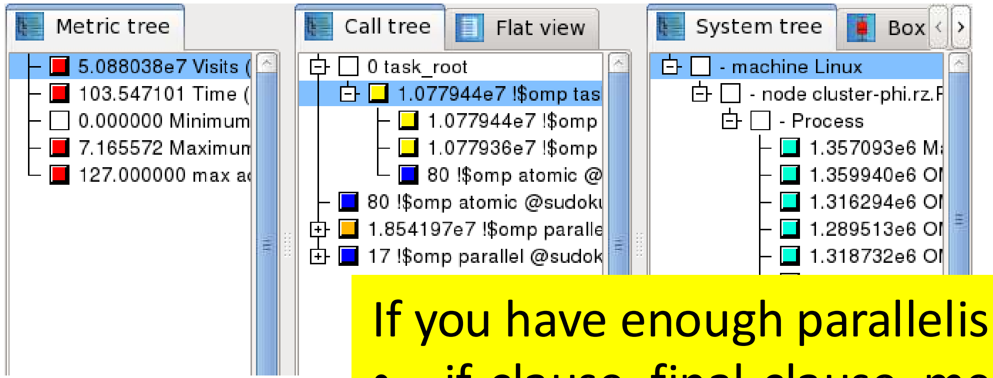
Tracing provides more details:



Tasks get much smaller down the call-stack.

Performance Analysis

Event-based profiling provides a good overview :

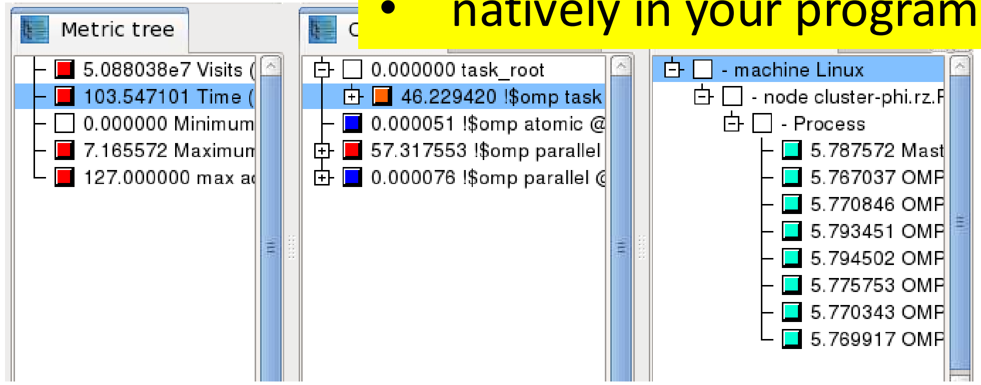


Metric tree
5.088038e7 Visits (
103.547101 Time (
0.000000 Minimum
7.165572 Maximum
127.000000 max ac

Call tree / Flat view
0 task_root
1.077944e7 !$omp tas
1.077944e7 !$omp
1.077936e7 !$omp
80 !$omp atomic @
80 !$omp atomic @sudoku
1.854197e7 !$omp paralle
17 !$omp parallel @sudok

System tree / Box
- machine Linux
- node cluster-phi.rz.F
- Process
1.357093e6 Ma
1.359940e6 OI
1.316294e6 OI
1.289513e6 OI
1.318732e6 OI

Every thread i



Metric tree
5.088038e7 Visits (
103.547101 Time (
0.000000 Minimum
7.165572 Maximum
127.000000 max ac

C
0.000000 task_root
46.229420 !$omp task
0.000051 !$omp atomic @
57.317553 !$omp parallel
0.000076 !$omp parallel @

System tree
- machine Linux
- node cluster-phi.rz.F
- Process
5.787572 Mast
5.767037 OMP
5.770846 OMP
5.793451 OMP
5.794502 OMP
5.775753 OMP
5.770343 OMP
5.769917 OMP

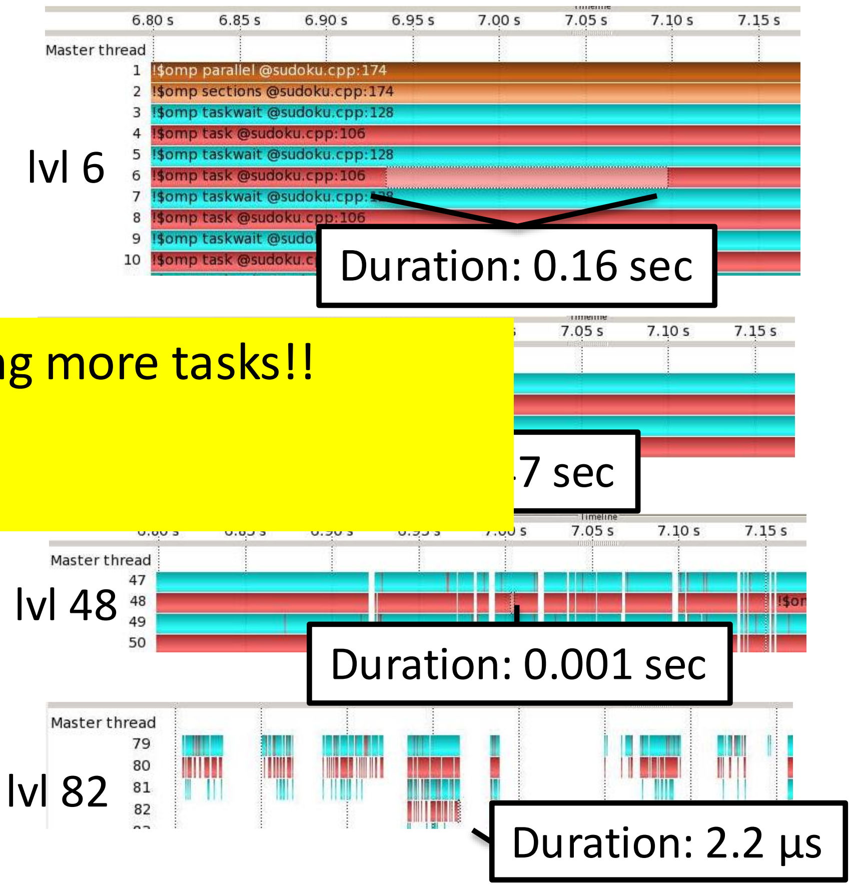
If you have enough parallelism, stop creating more tasks!!

- if-clause, final-clause, mergeable-clause
- natively in your program code

… in ~5.7 seconds.

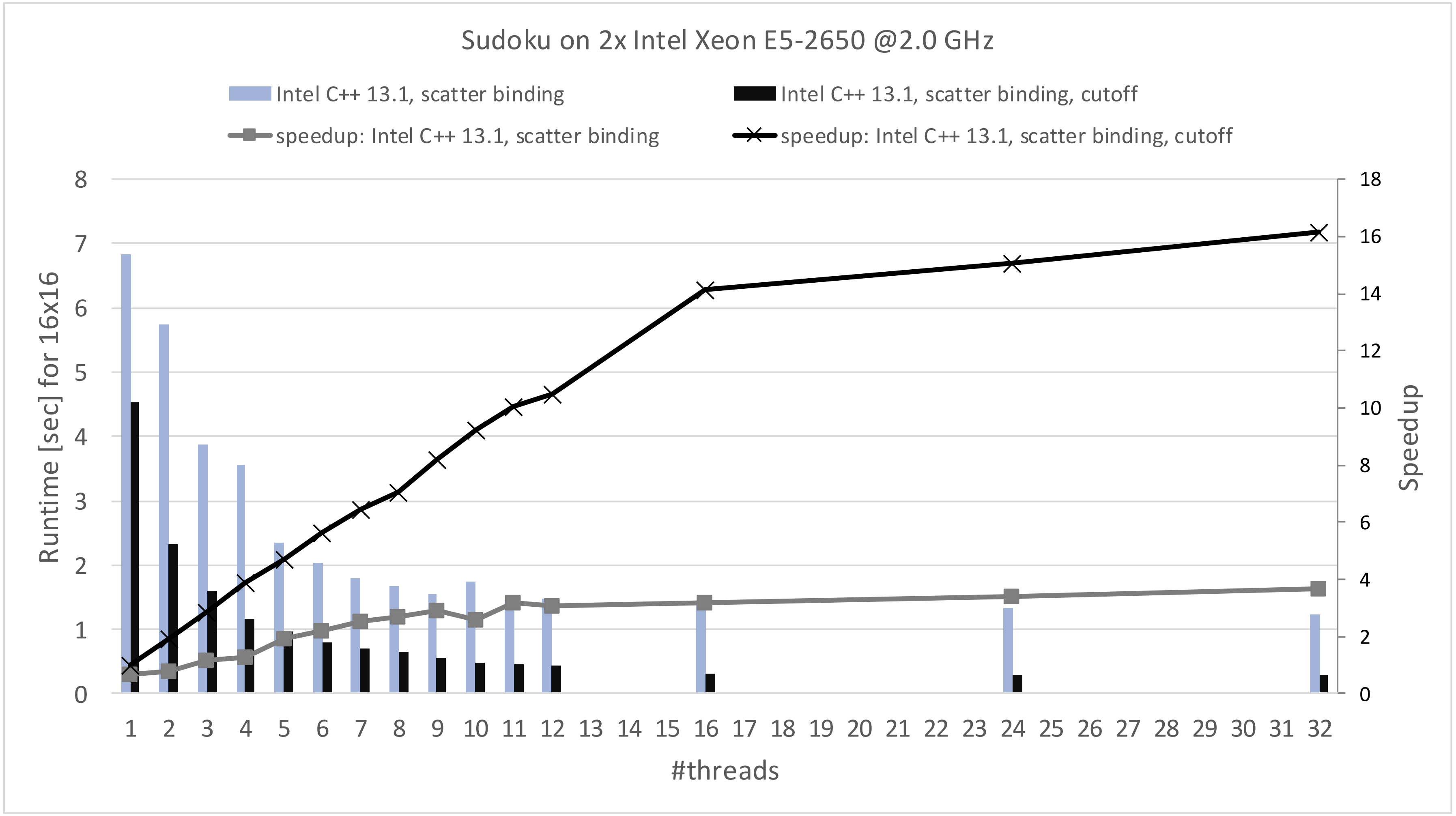
=> average duration of a task is ~4.4 µs

Tracing provides more details:



Tasks get much smaller down the call-stack.

Performance Evaluation (with cutoff)



Improving Tasking Performance: Cutoff clauses and strategies

The `if` clause

- Rule of thumb: the `if(expression)` clause as a “switch off” mechanism
  - → Allows lightweight implementations of task creation and execution but it reduces the parallelism
- If the `expression` of the `if` clause evaluates to `false`
  - → the encountering task is suspended
  - → the new task is executed immediately (task dependences are respected!!)
  - → the encountering task resumes its execution once the new task is completed
  - → This is known as *undeferred task*

```
int foo(int x) {
  printf(“entering foo function\n”);
  int res = 0;
  #pragma omp task shared(res) if(false)
  {
         res += x;
  }
  printf(“leaving foo function\n”);
}
```

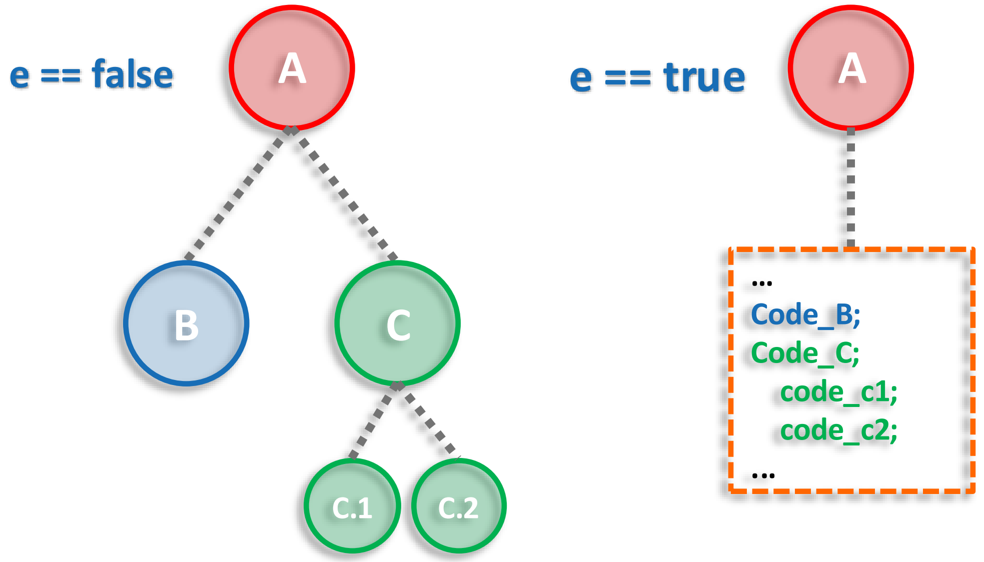
Really useful to debug tasking applications!

- Even if the `expression` is `false`, data-sharing clauses are honored

The `final` clause

- The `final(expression)` clause
  - → Nested tasks / recursive applications
  - → allows to avoid future task creation → reduces overhead but also reduces parallelism
- If the `expression` of the `final` clause evaluates to `true`
  - → The new task is created and executed normally but in its context all tasks will be executed immediately by the same thread (*included tasks*)

```
#pragma omp task final(e)
{
  #pragma omp task
  { … }
  #pragma omp task
  { … #C.1; #C.2 … }
  #pragma omp taskwait
}
```



- Data-sharing clauses are honored too!

The `mergeable` clause

- The `mergeable` clause
  - → Optimization: get rid of “data-sharing clauses are honored”
  - → This optimization can only be applied in *undeferred* or *included tasks*
- A Task that is annotated with the `mergeable` clause is called a *mergeable task*
  - → A task that may be a *merged task* if it is an *undeferred task* or an *included task*
- A *merged task* is:
  - → A task for which the data environment (inclusive of ICVs) may be the same as that of its generating task region
- A good implementation could execute a merged task without adding any OpenMP-related overhead

Unfortunately, there are no OpenMP commercial implementations taking advantage of `final` neither `mergeable` =(