HPC.NRW

Introduction to OpenMP

Dr. Christian Terboven

THE COMPETENCE NETWORK FOR HIGH PERFORMANCE COMPUTING IN NRW.

Sudoku (example)

Dr. Christian Terboven

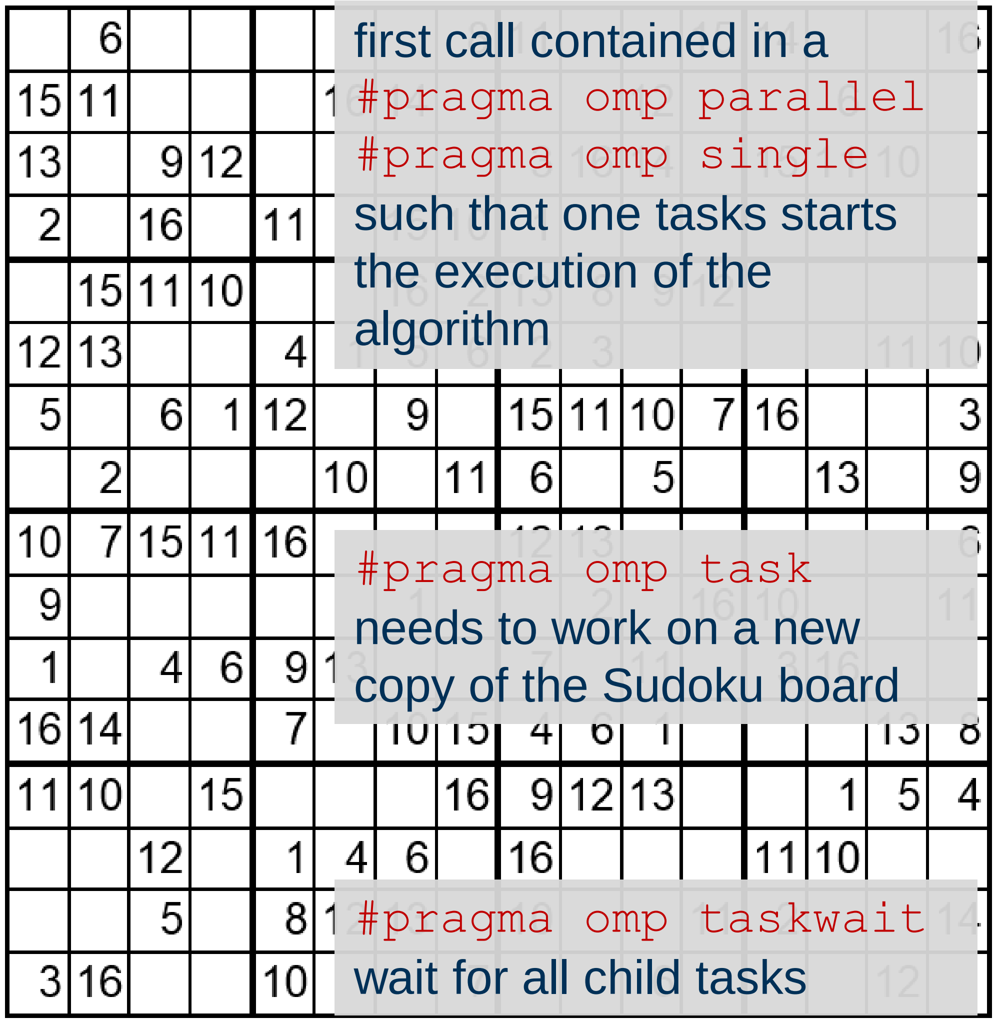
Introduction to OpenMP

Parallel Brute-force Sudoku

	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	

- Lets solve Sudoku puzzles with brute multi-core force
  - (1) Search an empty field
  - (2) Try all numbers:
    - (2 a) Check Sudoku
      - If invalid: Skip
      - If valid: Go to next field
  - Wait for completion

Parallel Brute-force Sudoku



first call contained in a

```
#pragma omp parallel
#pragma omp single
```

such that one tasks starts the execution of the algorithm

```
#pragma omp task
```

needs to work on a new copy of the Sudoku board

```
#pragma omp taskwait
```

wait for all child tasks

- Lets solve Sudoku puzzles with brute multi-core force
  - (1) Search an empty field
  - (2) Try all numbers:
    - (2 a) Check Sudoku
      - If invalid: Skip
      - If valid: Go to next field
  - Wait for completion

Parallel Brute-force Sudoku

- OpenMP parallel region creates a team of threads

```
#pragma omp parallel
{
#pragma omp single
   solve_parallel(0, 0, sudoku2,false); ur
} // end omp parallel
```

  - Single construct: One thread enters the execution of `solve_parallel`
  - The other threads wait at the end of the `single` …
    - … and are ready to pick up threads „from the work queue“

- Syntactic sugar (either you like it or you don‘t)

```
#pragma omp parallel sections
{
   solve_parallel(0, 0, sudoku2,false);
} // end omp parallel
```

Parallel Brute-force Sudoku

– The actual implementation

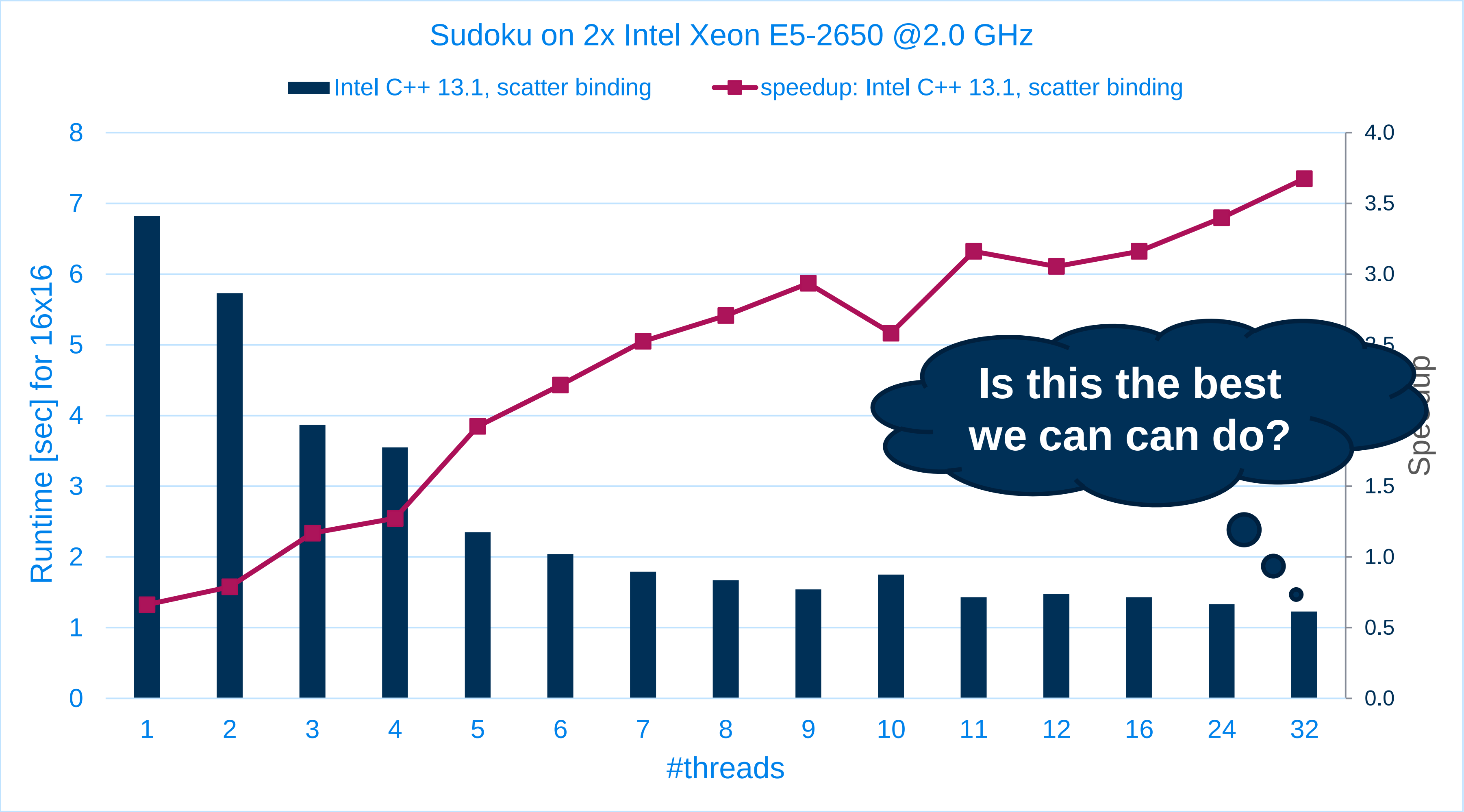
```
for (int i = 1; i <= sudoku->getFieldSize(); i++) {
   if (!sudoku->check(x, y, i)) {
      #pragma omp task firstprivate(i,x,y,sudoku)
      {
         // create from copy constructor
         CSudokuBoard new_sudoku(*sudoku);
         new_sudoku.set(y, x, i);
         if (solve_parallel(x+1, y, &new_sudoku)) {
            new_sudoku.printBoard();
         }
      } // end omp task
   }
}

#pragma omp taskwait
```

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

#pragma omp taskwait
wait for all child tasks

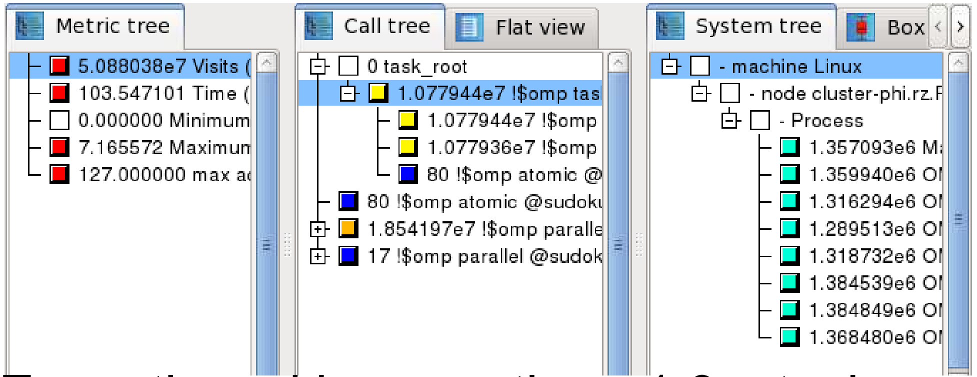
Performance Evaluation



Is this the best we can can do?

Performance Analysis

Event-based profiling gives a good overview :



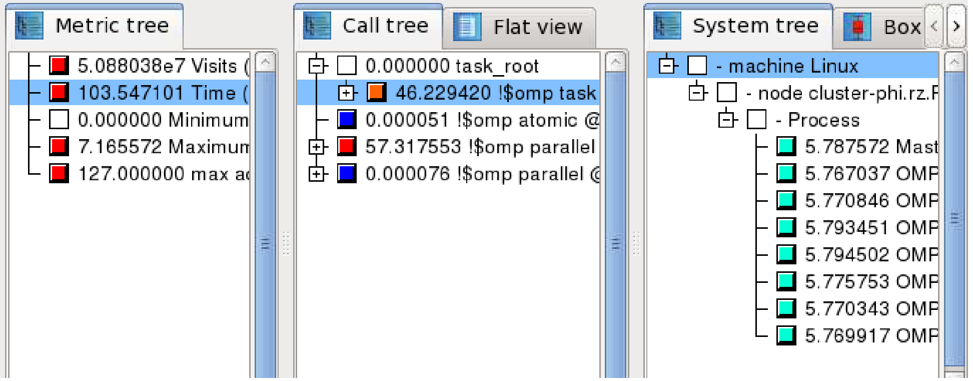
Metric tree | Call tree | Flat view | System tree | Box

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

```
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
```

```
- machine Linux
  - node cluster-phi.rz.F
    - Process
      1.357093e6 Ma
      1.359940e6 O
      1.316294e6 O
      1.289513e6 O
      1.318732e6 O
      1.384539e6 O
      1.384849e6 O
      1.368480e6 O
```

Every thread is executing ~1.3m tasks…



Metric tree | Call tree | Flat view | System tree | Box

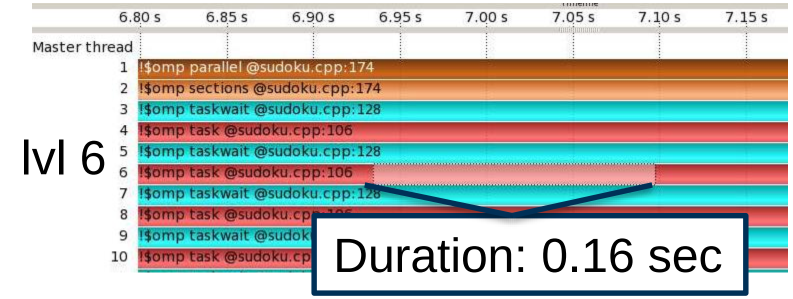
- 5.088038e7 Visits (
- 103.547101 Time (
- 0.000000 Minimum
- 7.165572 Maximum
- 127.000000 max ac

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

```
- 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
```

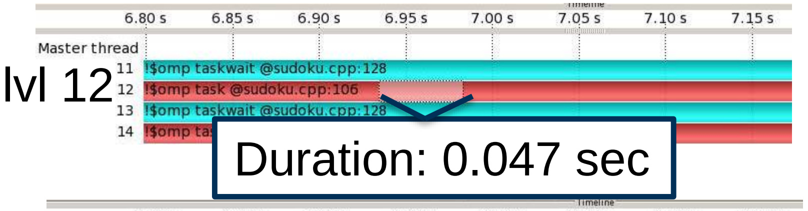
… in ~5.7 seconds.
=> average duration of a task is ~4.4 µs

Tracing gives more details:


6.80 s
6.85 s
6.90 s
6.95 s
7.00 s
7.05 s
7.10 s
7.15 s
Master thread
1 !$omp parallel @sudoku.cpp:174
2 !$omp sections @sudoku.cpp:174
3 !$omp taskwait @sudoku.cpp:128
4 !$omp task @sudoku.cpp:106
5 !$omp taskwait @sudoku.cpp:128
6 !$omp task @sudoku.cpp:106
7 !$omp taskwait @sudoku.cpp:128
8 !$omp task @sudoku.cp
9 !$omp taskwait @sudok
10 !$omp task @sudoku.cp

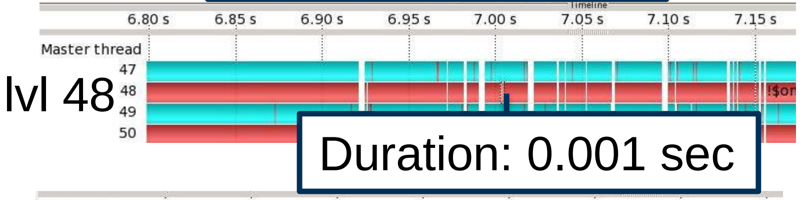
lvl 6

Duration: 0.16 sec


6.80 s
6.85 s
6.90 s
6.95 s
7.00 s
7.05 s
7.10 s
7.15 s
Master thread
11 !$omp taskwait @sudoku.cpp:128
12 !$omp task @sudoku.cpp:106
13 !$omp taskwait @sudoku.cpp:128
14 !$omp ta

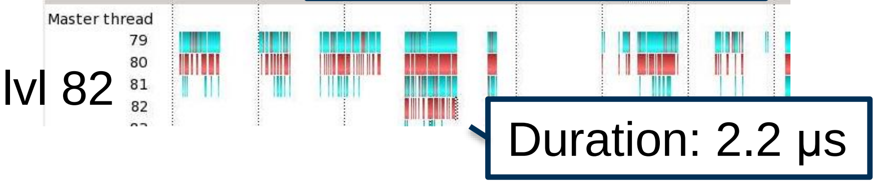
lvl 12

Duration: 0.047 sec



lvl 48

Duration: 0.001 sec



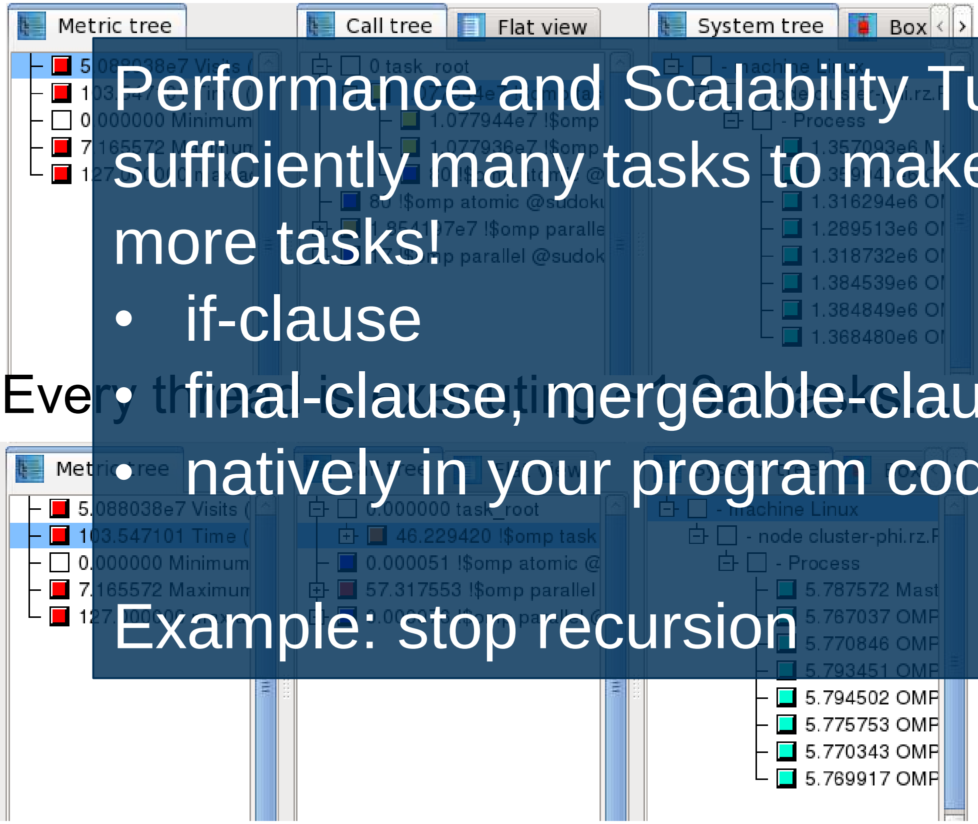
lvl 82

Duration: 2.2 µs

Tasks get much smaller down the call-stack.

Performance Analysis

Event-based profiling gives a good overview :



Performance and Scalability Tuning Idea: If you have created sufficiently many tasks to make you cores busy, stop creating more tasks!

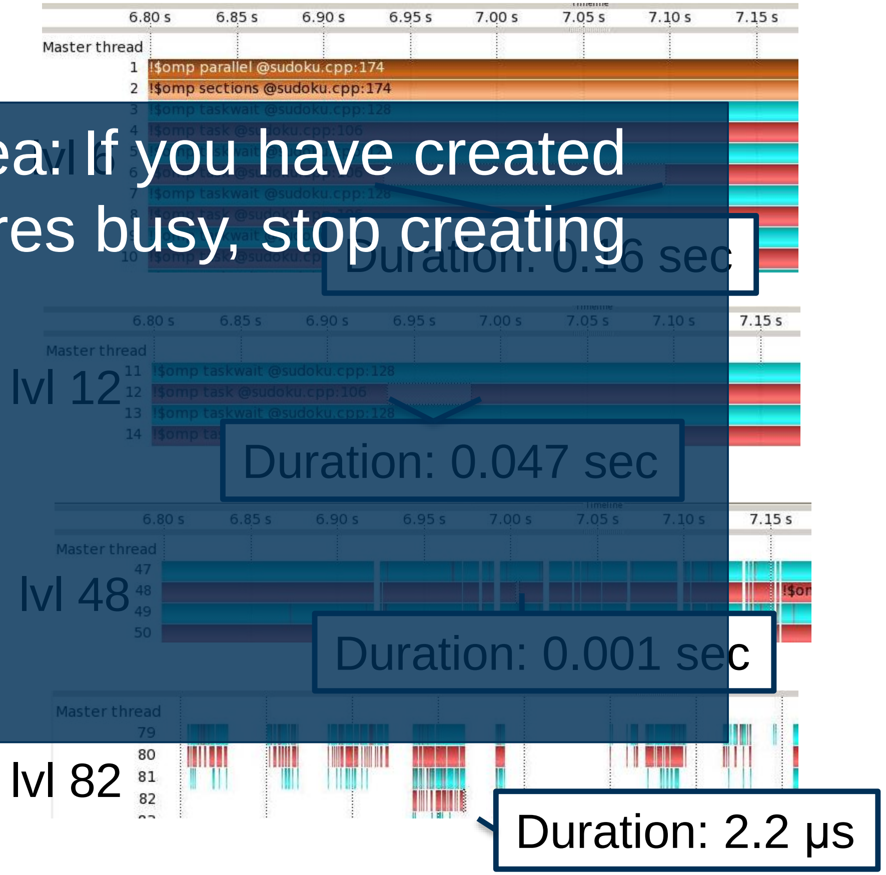
- if-clause
- final-clause, mergeable-clause
- natively in your program code

Example: stop recursion

Eve

… in ~5.7 seconds.
=> average duration of a task is ~4.4 μs

Tracing gives more details:


6.80 s
6.85 s
6.90 s
6.95 s
7.00 s
7.05 s
7.10 s
7.15 s
Master thread
!$omp parallel @sudoku.cpp:174
!$omp sections @sudoku.cpp:174
!$omp taskwait @sudoku.cpp:128
!$omp task @sudoku.cpp:106
lvl 12
lvl 48
lvl 82

Duration: 0.16 sec

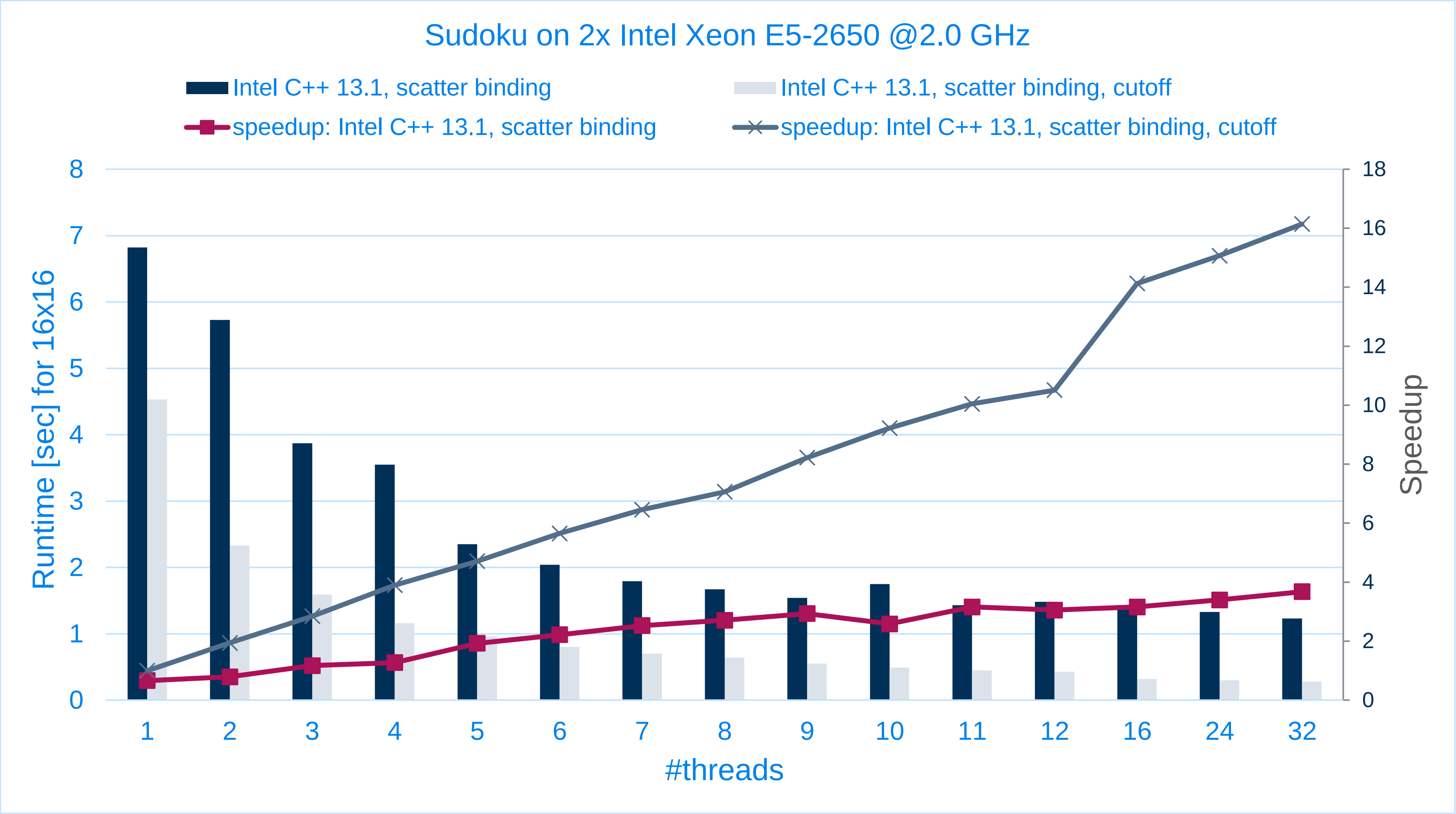
Duration: 0.047 sec

Duration: 0.001 sec

Duration: 2.2 μs

Tasks get much smaller down the call-stack.

Performance Evaluation



Questions?