OpenMP®

SC23 Booth Talk Series

Make OpenMP Go Fast

Ruud van der Pas
Senior Principal Software Engineer, Oracle Linux Engineering

OpenMP and Performance

You can get good performance with OpenMP

And your code will scale

If you do things in the right way

The OpenMP Performance Court

In this talk we cover the basics how to get good performance

Follow the guidelines and the performance should be decent

An OpenMP compiler and runtime should Do The Right Thing

You may not get blazing scalability, but …

The lawyers in the OpenMP Performance Court have no case against you

Ease of Use ?

The ease of use of OpenMP is a mixed blessing (but I still prefer it over the alternative)

Ideas are easy and quick to implement

But some constructs are more expensive than others

If you write dumb code, you probably will get dumb performance

Just don't blame OpenMP, please*

) It is fine to blame the weather, or politicians, or both though

My Preferred Tuning Strategy

In terms of complexity, use the most efficient algorithm

Select a profiling tool

Find the highest level of parallelism
(this should however provide enough work to use many threads)

Use OpenMP in an efficient way

Be prepared to have to do some performance experiments

Two Things You Need To Know

About Cores and Hardware Threads

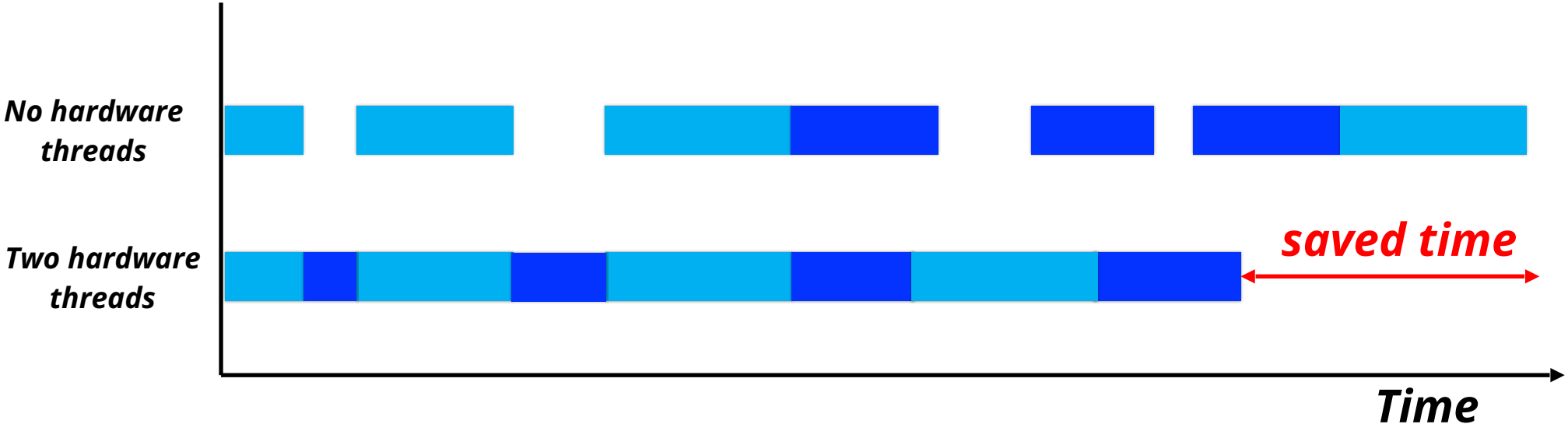
A core may, or may not, support hardware threads

This is part of the design

These hardware threads may accelerate the execution of a single application, or improve the throughput of a workload

The idea is that the pipeline is used by another thread in case the current thread is idle

How Hardware Threads Work



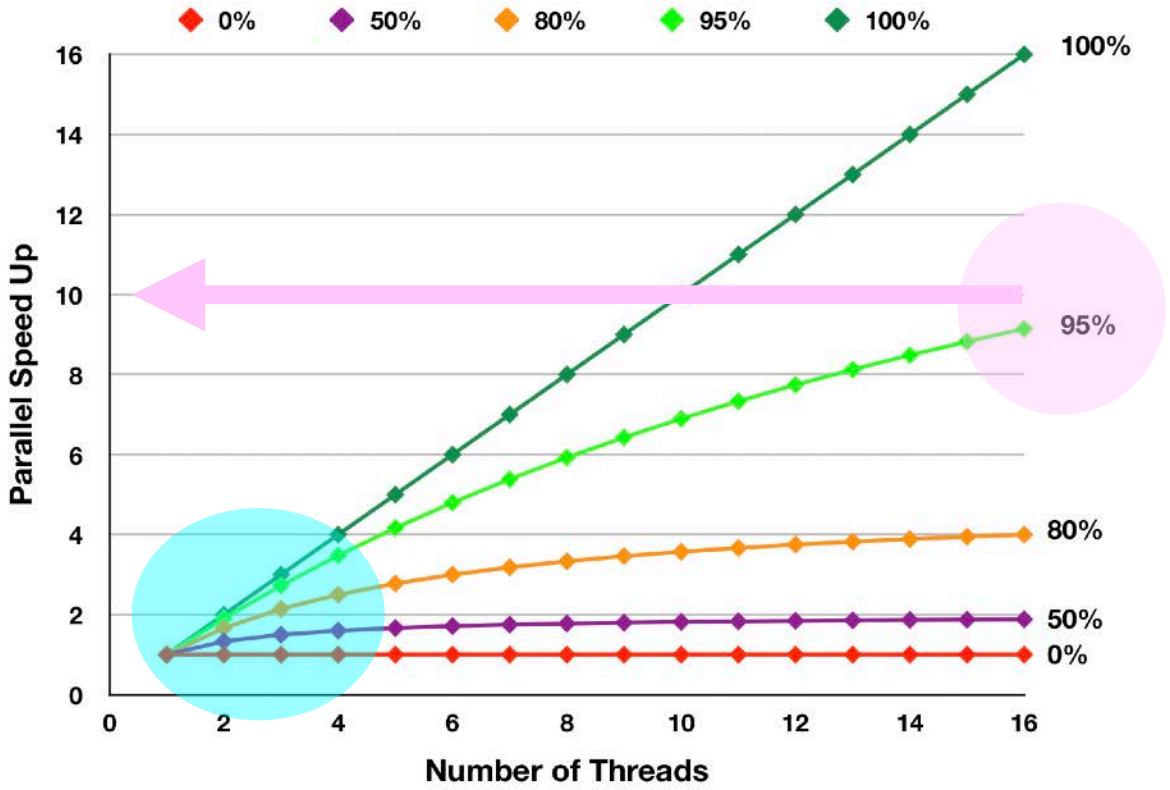
Amdahl's Law

Suppose your application needs 100 seconds to run

If 80% of this run time can execute in parallel, the time using 4 threads is 80/4+20 = 40 seconds

This means that your program is 2.5x faster, not 4x

Amdahl's Law Using 16 Threads



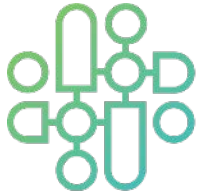
Morale

Amdahl's Law shows that you need to parallelize a significant fraction of the run time to see a decent speed up for higher thread counts

The BIG issue is that the serial, single thread, part of your code will dominate sooner than later

And this is why you should pay attention to improve the single thread performance, before parallelizing your code

How To Not Write Dumb OpenMP Code



SC23
Denver, CO | i am hpc.

The Basics For All Users

Do not parallelize what does not matter

Never tune your code without using a profiling tool

Do not share data unless you have to
(in other words, use private data as much as you can)

Think BIG
(maximize the size of the parallel regions)

One "parallel for" is fine. More, back to back, is EVIL.

The Wrong and Right Way Of Doing Things

```
#pragma omp parallel for
 { <code block 1> }
          ⋮
#pragma omp parallel for
 { <code block n> }
```

Parallel region overhead repeated "n" times
No potential for the "nowait" clause

```
#pragma omp parallel
{
   #pragma omp for
   { <code block 1> }
          ⋮
   #pragma omp for nowait
   { <code block n> }
} // End of parallel region
```

Parallel region overhead only once
Potential for the "nowait" clause

More Basics

Every barrier matters
(and please use them carefully)

The same is true for locks and critical regions
(use atomic constructs where possible)

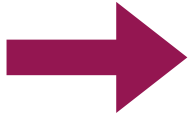
EVERYTHING Matters
(minor overheads get out of hand eventually)

Another Example

```
#pragma omp single
{
  <some code>
} // End of single region

#pragma omp barrier

  <more code>
```



```
#pragma omp single
{
  <some code>
} // End of single region

#pragma omp barrier

  <more code>
```

The second barrier is redundant because the single construct has an implied barrier already

(this second barrier will still take time though)

More Things to Consider

Identify opportunities to use the nowait clause

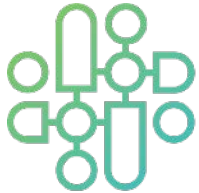
(a very powerful feature, but be aware of data races)

Use the schedule clause in case of load balancing issues

Avoid nested parallelism
(the nested barriers really add up)

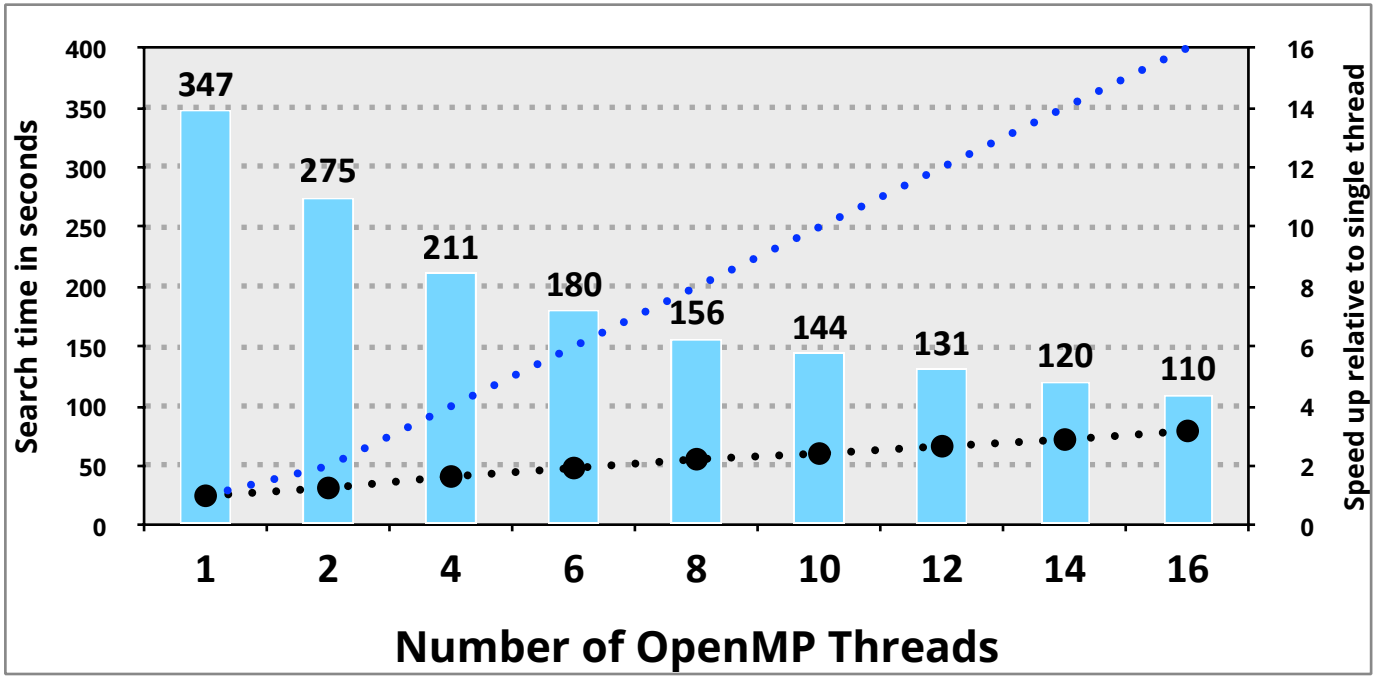
Consider tasking instead
(provides much more flexibility and finer granularity)

Case Study - Graph Analysis



SC23
Denver, CO | i am hpc.

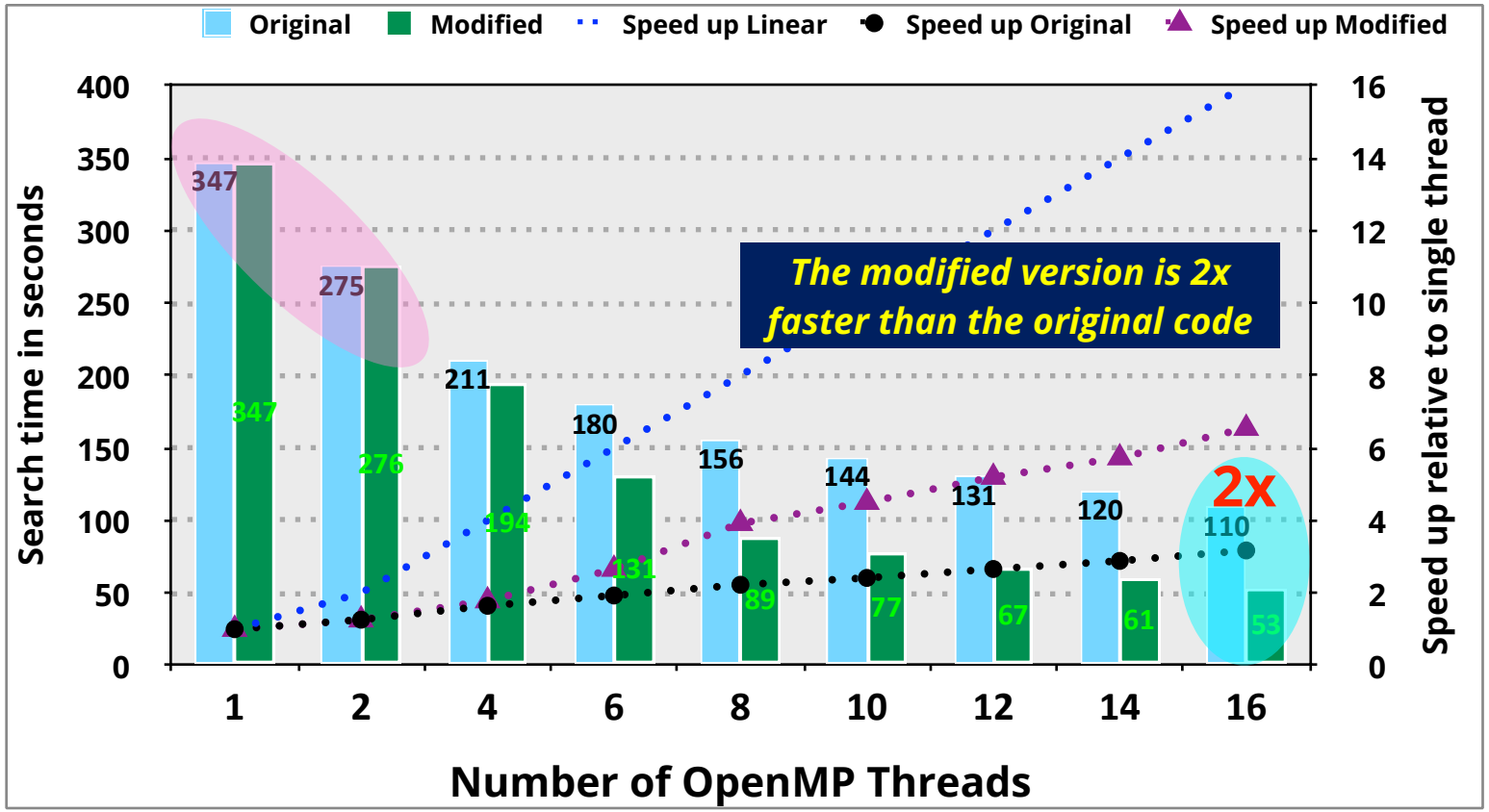
The Scalability is Disappointing



- ✓ *The data is for a 9 GB sized problem (SCALE 24)*
- ✓ *Search time reduces as threads are added*
- ✓ *Benefit from all 16 (hyper) threads*
- ✓ *The 3.2x parallel speed up is disappointing*
- ✓ *The parallel scalability is similar for larger graphs*

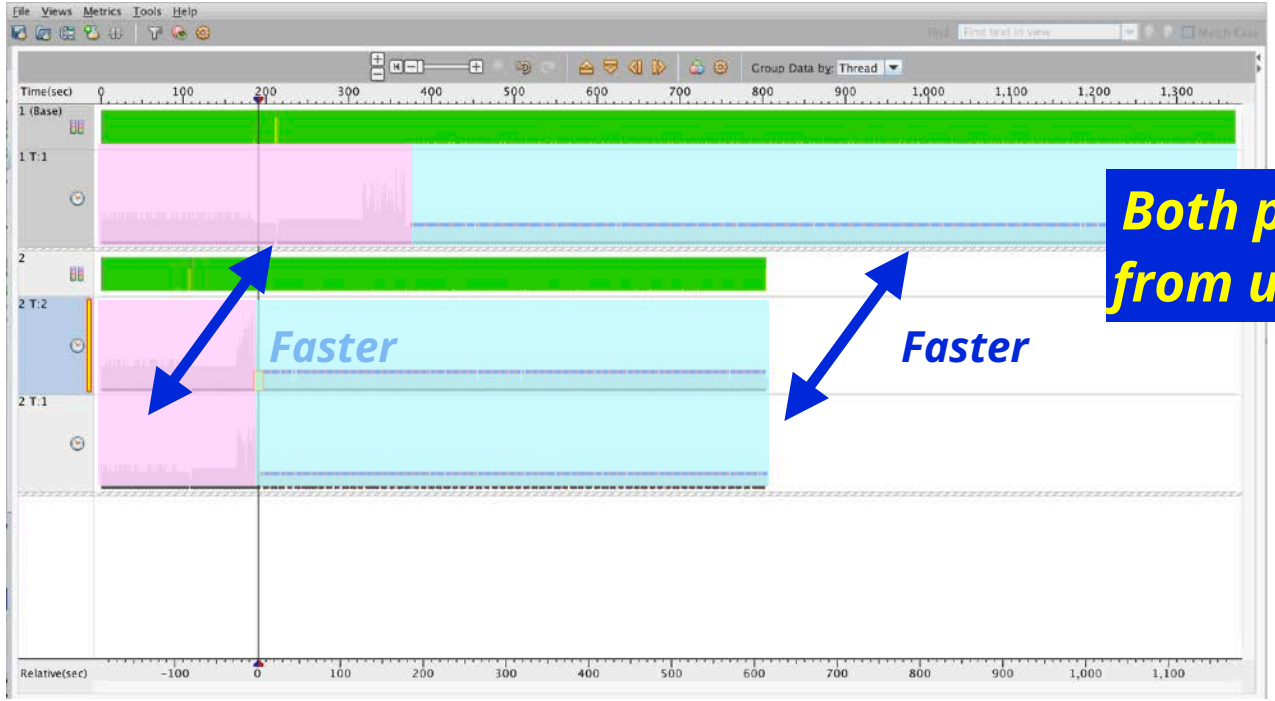
System: A cloud instance with 8 Intel Xeon Platinum 8167M CPU @ 2.00GHz ("Skylake") cores, 16 hardware threads

Performance Of The Original and Modified Code

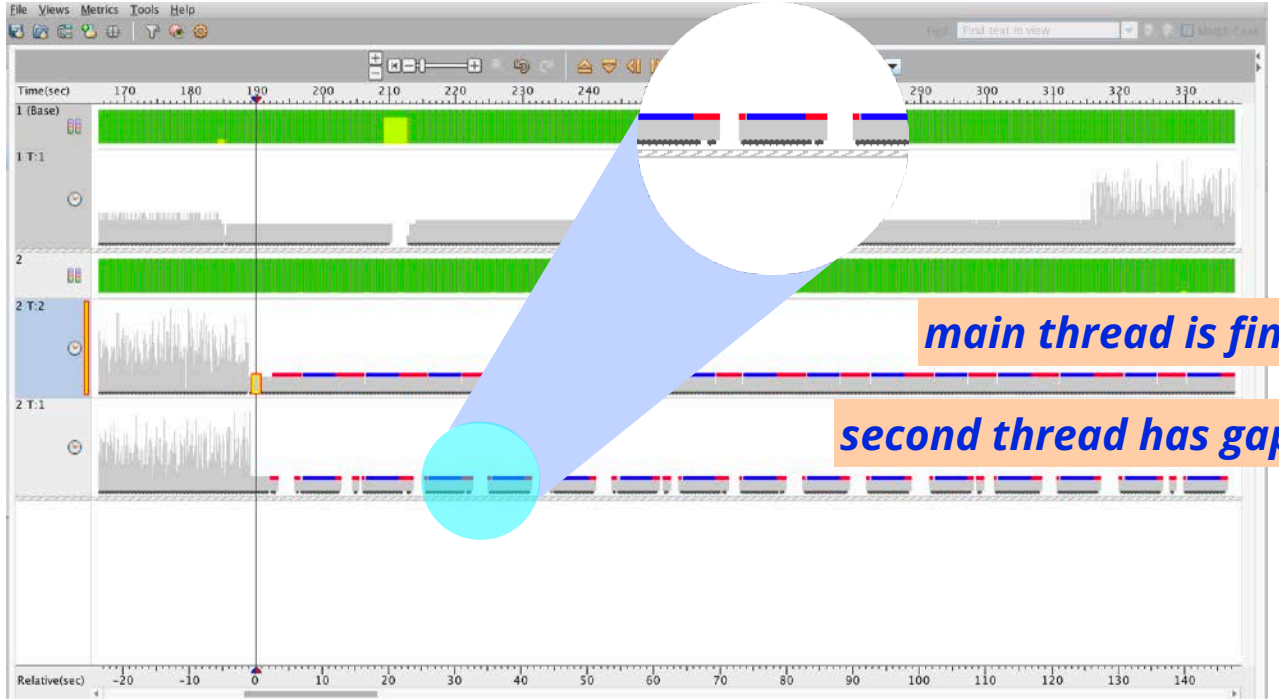


- ✓ *A noticeable reduction in the search time at 4 threads and beyond*
- ✓ *The parallel speed up increases to 6.5x*
- ✓ *The search time is reduced by 2x*

A Comparison Between 1 And 2 Threads

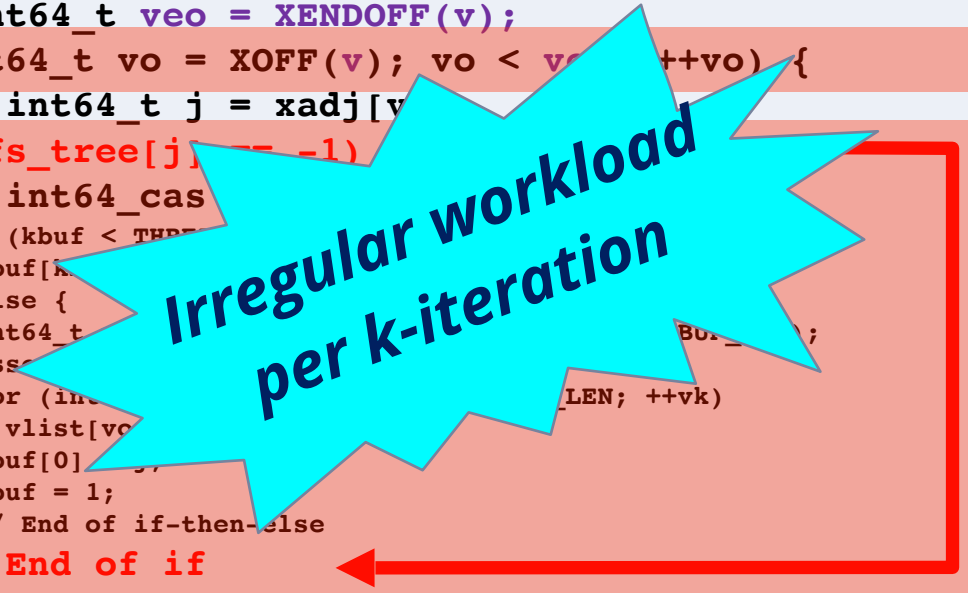


Zoom In On The Second Thread



The Issue

```
#pragma omp for
for (int64_t k = k1; k < oldk2; ++k) {
  const int64_t v = vlist[k];
  const int64_t veo = XENDOFF(v);
  for (int64_t vo = XOFF(v); vo < ve      ++vo) {
    const int64_t j = xadj[v
    if (bfs_tree[j]      -1)
      if (int64_cas
        if (kbuf < THR
          nbuf[k
        } else {
          int64_t
          ass
          for (in                    LEN; ++vk)
            vlist[vo
          nbuf[0]
          kbuf = 1;
        } // End of if-then-else
      } // End of if
    } // End of if
  } // End of for-loop on vo
} // End of parallel for-loop on k
```

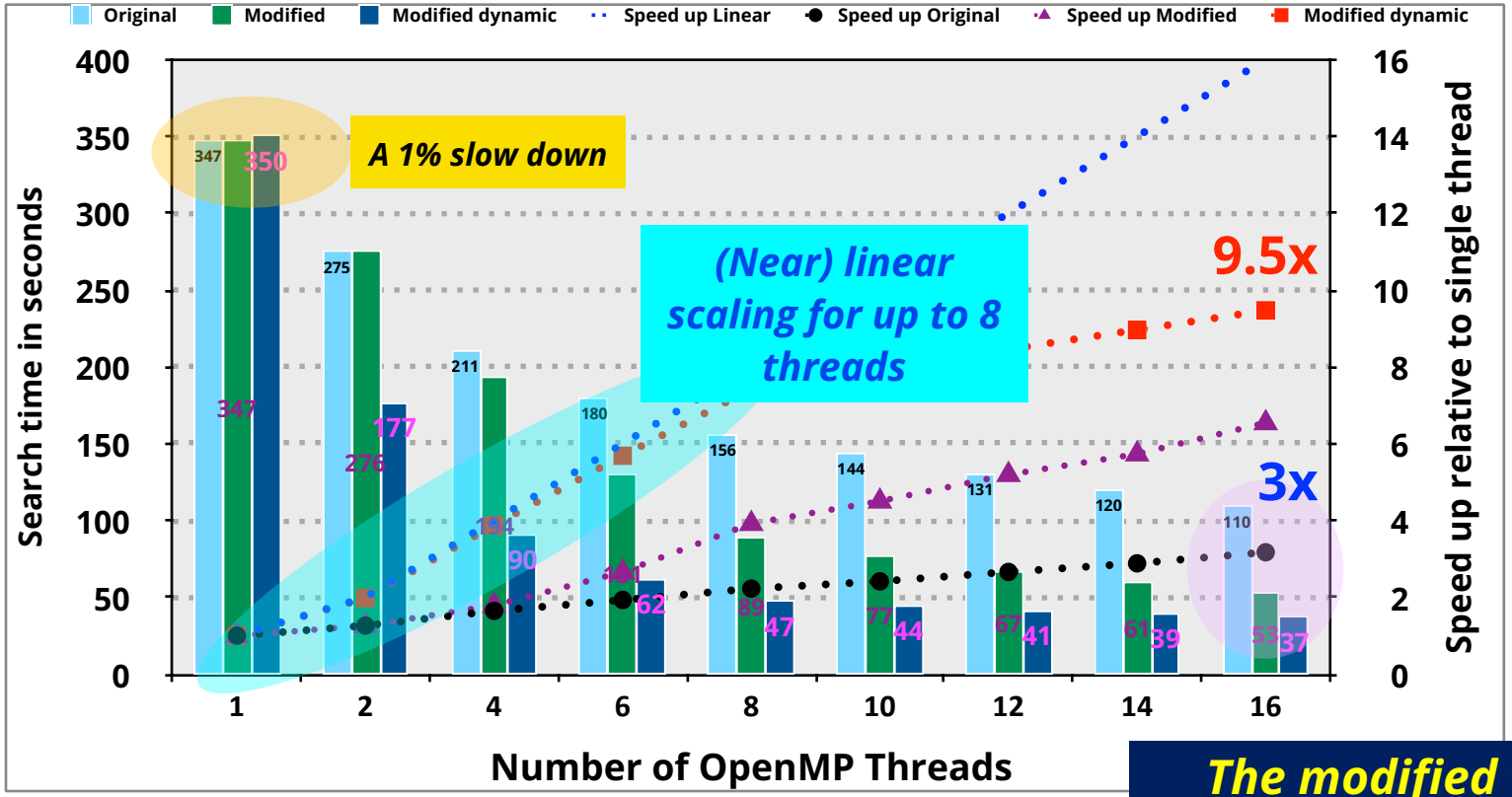


Fixed length loop

Irregular length loop

Irregular control flow

The Performance With Dynamic Scheduling Added



- ✓ *A 1% slow down on a single thread*
- ✓ *Near linear scaling for up to 8 threads*
- ✓ *The parallel speed up increased to 9.5x*
- ✓ *The search time is reduced by 3x*

The modified version is 3x faster than the original code

Takeaways

There are many opportunities to improve the performance

If you follow the advice given, you should be fine

(in most of the cases, since there are always exceptions)

Use a profiling tool to guide you

Don't guess, since it is likely you might be wrong

Thank You And … Stay Tuned!

Bad OpenMP
Does Not Scale

Ruud van der Pas

SC23 OpenMP Booth Talk

OpenMP®

SC23 Booth Talk Series

openmp.org — OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc23 — OpenMP SC'23 booth talk videos and presentations