OpenMP

SC'20 Booth Talk Series

Be Lazy and Get Good OpenMP Performance

Ruud van der Pas, Oracle

“OpenMP Does Not Scale”

A common and persistent Myth

A programming model in itself can not “Not Scale”

What can not scale:

The tools you use (e.g. the compiler, libraries, etc.), or a mismatch between the system and the resource requirements

Or … You

The OpenMP Performance Court

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

If you follow these guidelines, you should expect decent performance

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
Ideas are easy and quick to implement
But some constructs are more expensive than others
If you write dumb code, you will get dumb performance
*Just don't blame OpenMP, please**

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

How To Not Write Dumb Code

About Single Thread Performance

You have to pay attention to single thread performance

Why? If your code performs badly on 1 core, what do you think will happen on 10 cores, or 20 cores, or … ?

Remember, scalability can mask poor performance
(a slow code tends to scale better, but is often still slower)

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 possible)*
One "parallel for" is fine. Multiple back to back is EVIL
Think BIG and maximize the size of the parallel regions

The Wrong and Right Way of Doing Things

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

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

Parallel region cost repeatedly incurred
No potential for the "nowait" clause

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

More on Parallel Regions

Each parallel region carries a relatively high overhead

The goal is to minimize the number of times a parallel region is encountered

For example, try to avoid to embed the parallel region inside a loop nest

More Basics

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
(a good profiling tool is indispensable to find out)

Beyond the Basics, but Don't Forget!

Every barrier matters
(needed, but please use them with care)

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

EVERYTHING Matters
(Amdahl's Law: minor overheads get out of hand quickly)

When Do Things Get Harder?

Memory access "just happens"

There are however two things to watch out for:

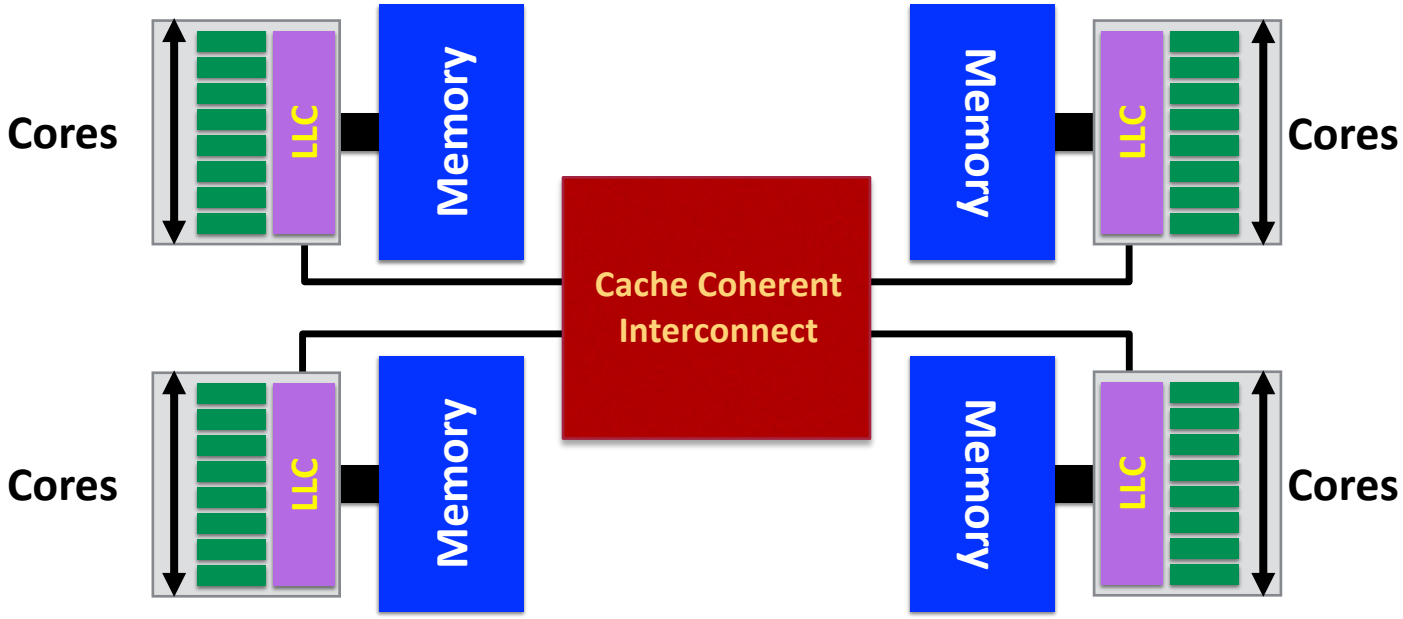
Non-Uniform Memory Access (NUMA)
and False Sharing

They have nothing to do with OpenMP as such and are a characteristic of using a shared memory architecture

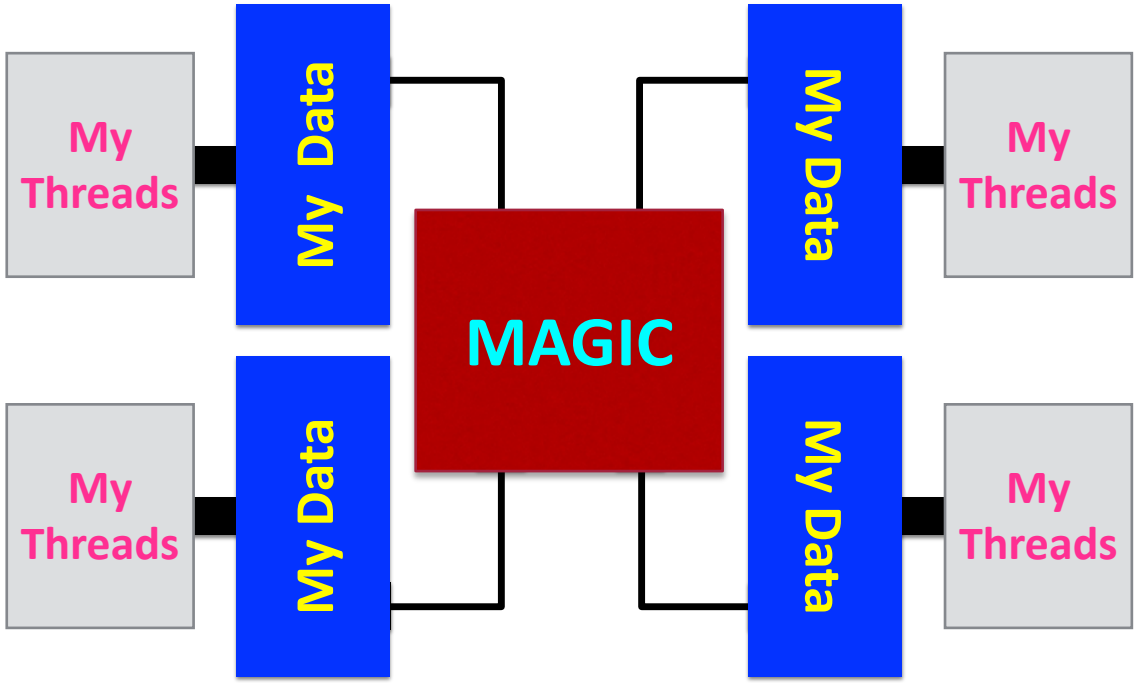
They may impact the performance though

NUMA – The System Most of Us Use Today

A Generic, but very Common and Contemporary NUMA System



NUMA - The Developer's View



The NUMA View

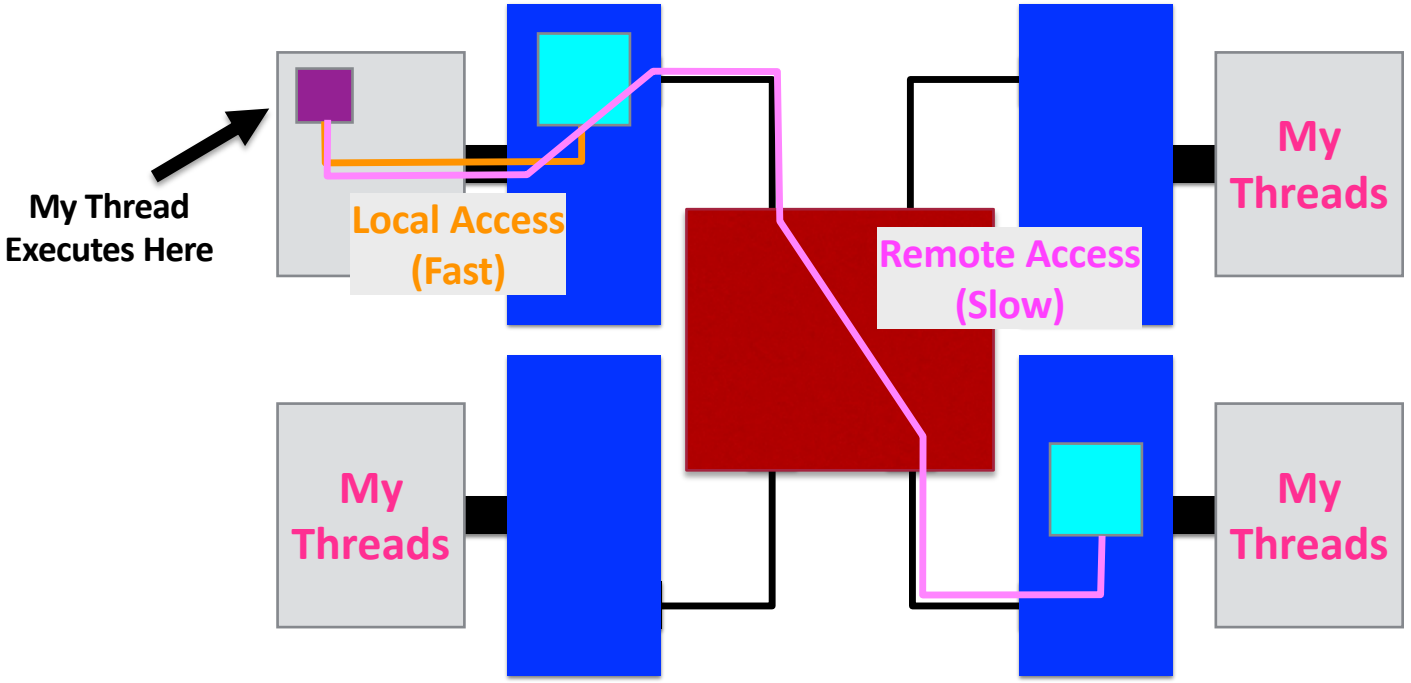
Memory is physically distributed, but logically shared

Shared data is transparently accessible to all threads

You don't know where the data is and it doesn't matter

Unless you care about performance …

NUMA - Local Versus Remote Access Times



About NUMA

Whether you like it, or not, NUMA is here to stay

And as core and node counts go up, it increasingly matters

The good news is that OpenMP has great support for NUMA

It is beyond the scope of this talk to cover this, but there is quite some information available

What is False Sharing?

False Sharing occurs when multiple threads modify the same cache line at the same time

This results in the cache line to move through the system (plus the additional cost of the cache coherence updates)

It is okay if this happens once in a while

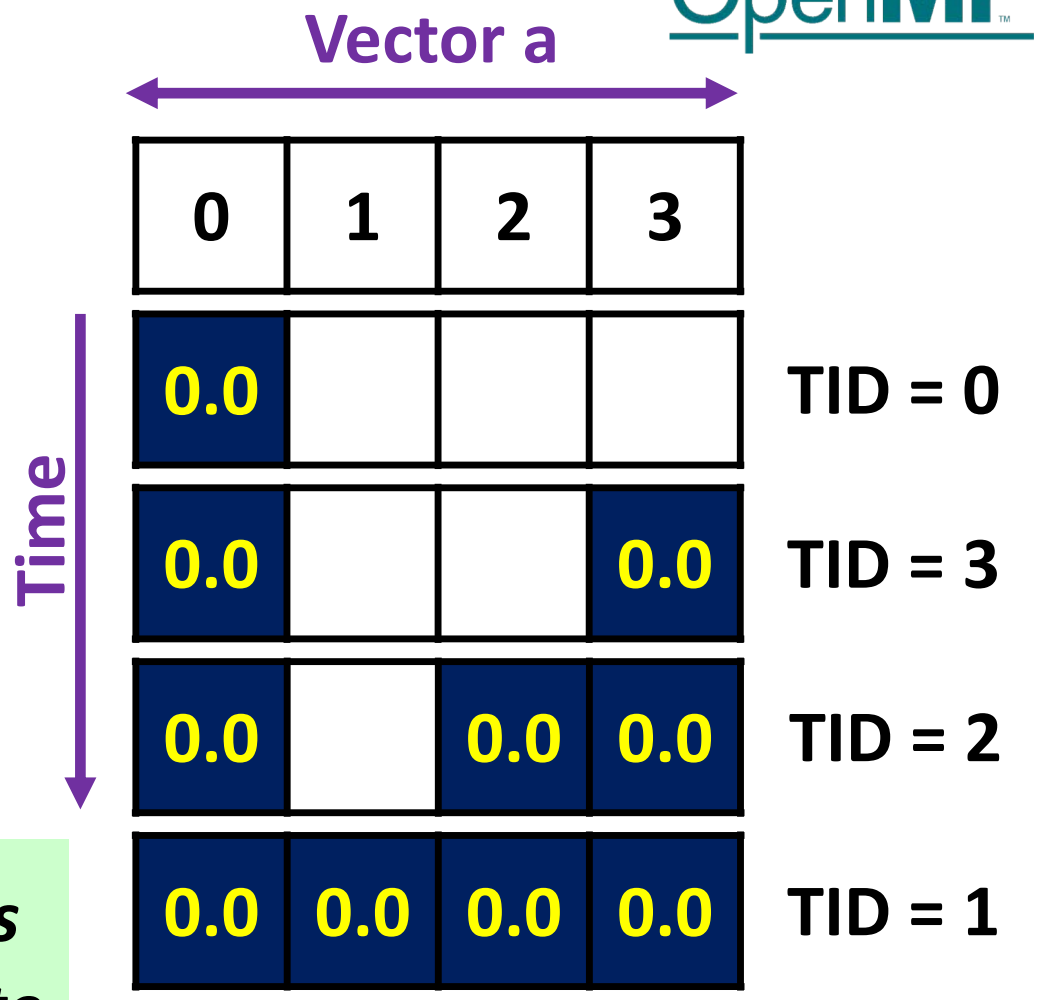
It is not okay if this happens very frequently

An Example of False Sharing

```
#pragma omp parallel shared(a)
{
    int TID = omp_get_thread_num();

    a[TID] = 0.0; // False Sharing

} // End of parallel region
```



With each update of "a", the cache line moves to the cache of the thread executing the update

Your Homework

Follow the guidelines just given and in this order
Where applicable, give it a try
Always make a profile before and after
Details sometimes make all the difference
In many cases, a performance "mystery" is explained by NUMA effects, False Sharing, or both

Thank You And ..... Stay Tuned !

Ruud van der Pas

Bad OpenMP
Does Not Scale

OpenMP

SC'20 Booth Talk Series

openmp.org	OpenMP API specs, forum, reference guides, and more
link.openmp.org/sc20	Videos and PDFs of OpenMP SC'20 presentations