Be Lazy and Get Good OpenMP Performance

Ruud van der Pas, Oracle
## “OpenMP Does Not Scale”

<table>
<thead>
<tr>
<th>A common and persistent Myth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A programming model in itself can not “Not Scale”</strong></td>
</tr>
<tr>
<td>What can not scale:</td>
</tr>
<tr>
<td><strong>The tools you use (e.g. the compiler, libraries, etc.), or a mismatch between the system and the resource requirements</strong></td>
</tr>
<tr>
<td>Or ... <strong>You</strong></td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>The ease of use of OpenMP is a mixed blessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideas are easy and quick to implement</td>
</tr>
<tr>
<td>But some constructs are more expensive than others</td>
</tr>
<tr>
<td>If you write dumb code, you will get dumb performance</td>
</tr>
<tr>
<td>Just don’t blame OpenMP, please*</td>
</tr>
</tbody>
</table>

*) 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

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

Parallel region cost repeatedly incurred
No potential for the “nowait” clause

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

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

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

<table>
<thead>
<tr>
<th>Identify opportunities to use the <code>nowait</code> clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a very powerful feature, but be aware of data races)</td>
</tr>
<tr>
<td>Use the <code>schedule</code> clause in case of load balancing issues</td>
</tr>
<tr>
<td>(a good profiling tool is indispensable to find out)</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>Memory access “just happens”</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are however two things to watch out for:</td>
</tr>
<tr>
<td>Non-Uniform Memory Access (NUMA) and False Sharing</td>
</tr>
<tr>
<td>They have nothing to do with OpenMP as such and are a characteristic of using a shared memory architecture</td>
</tr>
<tr>
<td>They may impact the performance though</td>
</tr>
</tbody>
</table>
NUMA – The System Most of Us Use Today

A Generic, but very Common and Contemporary NUMA System

Cache Coherent Interconnect
NUMA - The Developer’s View
The NUMA View

<table>
<thead>
<tr>
<th>Memory is physically distributed, but logically shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared data is transparently accessible to all threads</td>
</tr>
<tr>
<td>You don’t know where the data is and it doesn’t matter</td>
</tr>
</tbody>
</table>

Unless you care about performance ...
NUMA - Local Versus Remote Access Times

My Thread Executes Here

Local Access (Fast)

Remote Access (Slow)

My Threads

My Threads

My Threads

My Threads

Be Lazy and Get Good OpenMP Performance
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?

<table>
<thead>
<tr>
<th>False Sharing occurs when multiple threads modify the same cache line at the same time</th>
</tr>
</thead>
<tbody>
<tr>
<td>This results in the cache line to move through the system (plus the additional cost of the cache coherence updates)</td>
</tr>
<tr>
<td>It is okay if this happens once in a while</td>
</tr>
<tr>
<td>It is <em>not okay</em> if this happens very frequently</td>
</tr>
</tbody>
</table>
An Example of False Sharing

```c
#pragma omp parallel shared(a)
{
  int TID = omp_get_thread_num();
  a[TID] = 0.0; // False Sharing
}
```

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

link.openmp.org/sc20  Videos and PDFs of OpenMP SC’20 presentations