How To Get OpenMP Tasks To Do Your Work

Leverage Tasking to Make Your Life Easier

Ruud van der Pas

Webinar, September 22, 2021

$ whoishe

My background is in mathematics and physics

Previously, I worked at the University of Utrecht, Convex Computer, SGI, and Sun Microsystems

Currently I work in the Oracle Linux Engineering organization

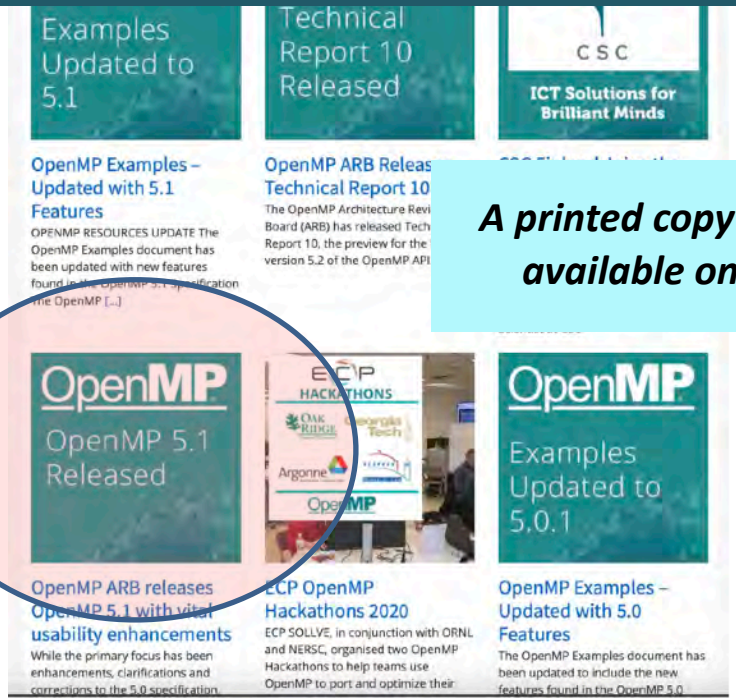
I have been involved with OpenMP since the introduction

I am passionate about performance and OpenMP in particular

Copyright © Ruud van der Pas 2021, All Rights Reserved

Your OneStop Place for OpenMP

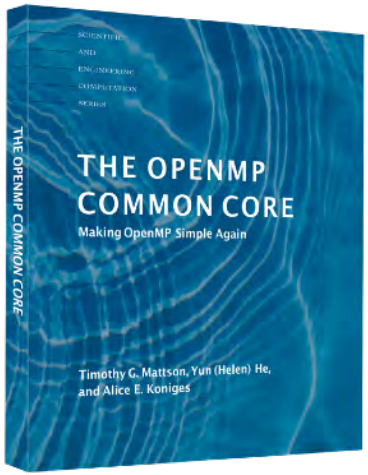
https://www.openmp.org



A printed copy of the 5.1 OpenMP specs is also available on Amazon (700+ pages, 1.7 kg)

Copyright © Ruud van der Pas 2021, All Rights Reserved

Food for the Eyes and Brains



OpenMP 2.5 and intro Parallel Computing

Covers the OpenMP Basics to get started

Focus on the Advanced Features

What goes on under the hood

Copyright © Ruud van der Pas 2021, All Rights Reserved

Why Tasks are Needed

Copyright © Ruud van der Pas 2021, All Rights Reserved

BT - The Time Before Tasking

The OpenMP Compiler Had To Know Everything
And in advance, (right) before execution
For example, the loop length, number of parallel sections, etc
Gets hard with more dynamic problems like processing linked lists, divide and conquer, recursion, etc
A solution was ugly. At best

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Solution

Tasking came to the rescue!

In this talk we will show it all works

But no formal terminology, definitions, etc.

These are covered in the specs

Copyright © Ruud van der Pas 2021, All Rights Reserved

Talking about the Specifications - 3.0

OpenMP Application Program Interface

Version 3.0 May 2008

About 4+ pages on the tasking feature (aside from definitions, scoping, etc)

Copyright © Ruud van der Pas 2021, All Rights Reserved

Talking about the Specifications - 5.1

About 22+ pages on the tasking feature (aside from definitions, scoping, etc)

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Tasking Concept

Copyright © Ruud van der Pas 2021, All Rights Reserved

What Is a Task?

A task is a chunk of independent work

You guarantee that different tasks can be executed simultaneously

```
#pragma omp task
{"this is my task"}
```


Copyright © Ruud van der Pas 2021, All Rights Reserved

When are Tasks Executed?

The OpenMP run time system decides on the scheduling of tasks

At certain points (implicit and explicit) tasks are guaranteed to be completed

Copyright © Ruud van der Pas 2021, All Rights Reserved

Division of Labour

This is what you need to do:

- *Identify independent portions of work, the tasks*
- *Use **the #pragma omp task** construct to define the tasks*

The OpenMP runtime system handles everything else:

- *When a thread encounters a task construct, a new task is created*
- *It is up to the runtime system when this task is executed and by whom*
- *Execution of a task is either immediate, or may be delayed/deferred*
- ***Task synchronization** may be used to enforce task completion*

Copyright © Ruud van der Pas 2021, All Rights Reserved

Task Creation and Execution in OpenMP

```
#pragma omp task
```

Defines a task

```
barrier (implied or explicit)
```

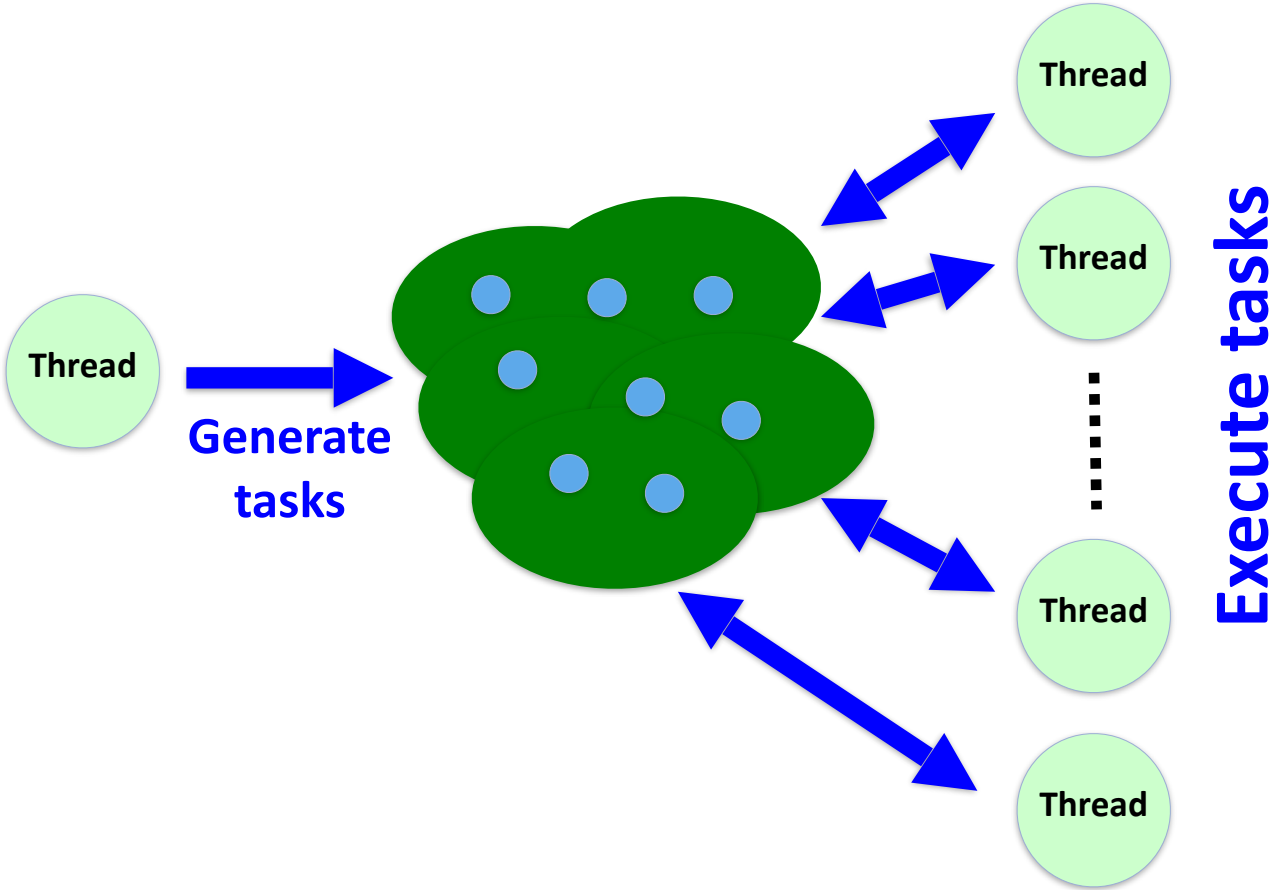
```
#pragma omp taskwait
```

Task Synchronization Points

```
#pragma omp taskgroup
```


Copyright © Ruud van der Pas 2021, All Rights Reserved

A Common Execution Model with Tasks



Copyright © Ruud van der Pas 2021, All Rights Reserved

A Day at the Races

Copyright © Ruud van der Pas 2021, All Rights Reserved

Today's Task

Write a program that prints either "A race car" or "A car race" and maximize the parallelism

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Starting Point

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

          printf("A ");
          printf("race ");
          printf("car ");

   printf("\n");
   return(0);
}
```

```
$ gcc hello.c
$ ./a.out
A race car
$
```

What will this program print ?

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Parallel Version

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
          printf("A ");
          printf("race ");
          printf("car ");

   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Copyright © Ruud van der Pas 2021, All Rights Reserved

Let's Run It

```
$ gcc -fopenmp hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car A race car
```

Note that this program could (for example) also print

"A A race race car car" or

"A race A car race car", or

"A race A race car car", or

.....

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Next Version

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
          printf("A ");
          printf("race ");
          printf("car ");
      }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Copyright © Ruud van der Pas 2021, All Rights Reserved

Let's Try It

```
$ gcc -fopenmp hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
```

But of course only 1 thread executes now …

Copyright © Ruud van der Pas 2021, All Rights Reserved

Time to Add the Tasks!

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");

         #pragma omp task
          {printf("race ");}

         #pragma omp task
          {printf("car ");}
      }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

Copyright © Ruud van der Pas 2021, All Rights Reserved

Mission Accomplished!

```
$ gcc -fopenmp hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
$ ./a.out
A race car
$ ./a.out
A car race
$
```

Tasks may be executed in arbitrary order

Copyright © Ruud van der Pas 2021, All Rights Reserved

More to Come

That went well and quickly, so here is a final task to do

Have the sentence end with "is fun to watch"
(hint: use a print statement)

Copyright © Ruud van der Pas 2021, All Rights Reserved

Add the Print Statement after the Tasks

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");
         #pragma omp task
          {printf("race ");}
         #pragma omp task
          {printf("car ");}
         printf("is fun to watch ");
      }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

Copyright © Ruud van der Pas 2021, All Rights Reserved

Oops!

```
$ gcc -fopenmp hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out

A is fun to watch race car
$ ./a.out

A is fun to watch race car
$ ./a.out

A is fun to watch car race
$
```

Tasks are executed at a Task Synchronization Point (and not necessarily where you see them …)

Copyright © Ruud van der Pas 2021, All Rights Reserved

Where is the Task Synchronization Point?

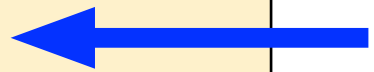
```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");
         #pragma omp task
          {printf("race ");}
         #pragma omp task
          {printf("car ");}
         printf("is fun to watch ");

      }  // Implied barrier

   } // End of parallel region

   printf("\n"); return(0);
}
```



Task synchronization point

Copyright © Ruud van der Pas 2021, All Rights Reserved

Use the Taskwait Feature

```
int main(int argc, char *argv[]) {

  #pragma omp parallel
  {
    #pragma omp single
     {
        printf("A ");
        #pragma omp task
          {printf("car ");}
        #pragma omp task
          {printf("race ");}

        #pragma omp taskwait

        printf("is fun to watch ");
     }
   } // End of parallel region

   printf("\n");return(0);
}
```

What will this program print using 2 threads ?

Task synchronization point

"The taskwait construct specifies a wait on the completion of child tasks of the current task"

Copyright © Ruud van der Pas 2021, All Rights Reserved

Mission Accomplished!

```
$ gcc -fopenmp hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
$
A car race is fun to watch
$ ./a.out
A car race is fun to watch
$ ./a.out
A race car is fun to watch
$
```

The tasks are executed first now

Copyright © Ruud van der Pas 2021, All Rights Reserved

Algorithm Examples

Copyright © Ruud van der Pas 2021, All Rights Reserved

When to Use Tasks?

If you need them
Tasks do not replace everything
For example, if the loop construct works well, don't touch it
Consider tasks when things are much more dynamic
If you can't predict how much work will be done, for example
Or if the workload is unbalanced, or if …

Copyright © Ruud van der Pas 2021, All Rights Reserved

Three Scenarios

Scenario	An Example
Irregular problems	*Scan a linked list*
Dependencies	*Overlap I/O and computation*
Recursion	*A sorting algorithm*

Copyright © Ruud van der Pas 2021, All Rights Reserved

Scan a Linked List

Copyright © Ruud van der Pas 2021, All Rights Reserved

Scan Through a Linked List

```
1 my_pointer = head_of_list;
2 while (my_pointer != NULL)
3  {
4    (void) do_independent_work(my_pointer->value);
5    my_pointer = my_pointer->next;
6  } // End of while loop
```

This is cumbersome to do without tasking:

- *First count the number of passes through the while-loop*
- *Convert the while-loop to a for-loop*

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Problem and the Idea

The Problem

- *Scanning through the linked list is a serial process*
- *Check each record until the NULL pointer is encountered*

The Idea

- *Make the execution of function do_independent_work a task*
- *Use the single construct to have a single thread generate the tasks*
- *The other threads start executing the tasks as they become available*

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Relevant OpenMP Code Fragment

```
 1 my_pointer = head_of_list;

 2 #pragma omp parallel firstprivate(my_pointer)
 3  {
 4    #pragma omp single nowait
 5      {
 6        while (my_pointer != NULL)
 7          {
 8            #pragma omp task firstprivate(my_pointer)
 9              {
10                (void) do_independent_work(my_pointer->value);
11              } // End of task
12            my_pointer = my_pointer->next;
13          } // End of while loop
14      } // End of single region
15  } // End of parallel region
```

Removes the implied barrier here

Copyright © Ruud van der Pas 2021, All Rights Reserved

Overlap I/O and Computation

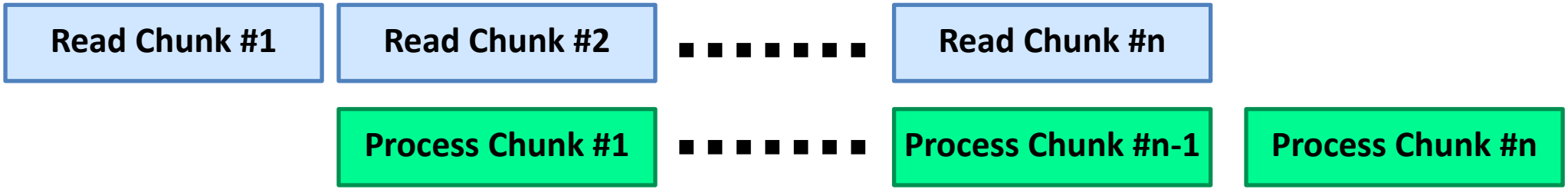
Copyright © Ruud van der Pas 2021, All Rights Reserved

A Common Problem

The processing part waits for data to arrive from disk
In case the data is read in chunks, tasking may help
The main idea is to overlap I/O and computations
This keeps the processor (more) busy
An elegant solution makes use of task dependencies

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Main Idea - Set Up a Pipeline



- *In reality, the two phases may not take the same time*
- *Regardless of that, this approach reduces the total time*
- *In the implementation there will be two tasks*
- *Through a dependence, we ensure processing does not start too early*
- *Although not shown here, a third stage may handle the output processing*

Copyright © Ruud van der Pas 2021, All Rights Reserved

Task Dependencies

It is possible to set up dependencies between tasks

This is implemented through the depend clause

Various dependence types are supported

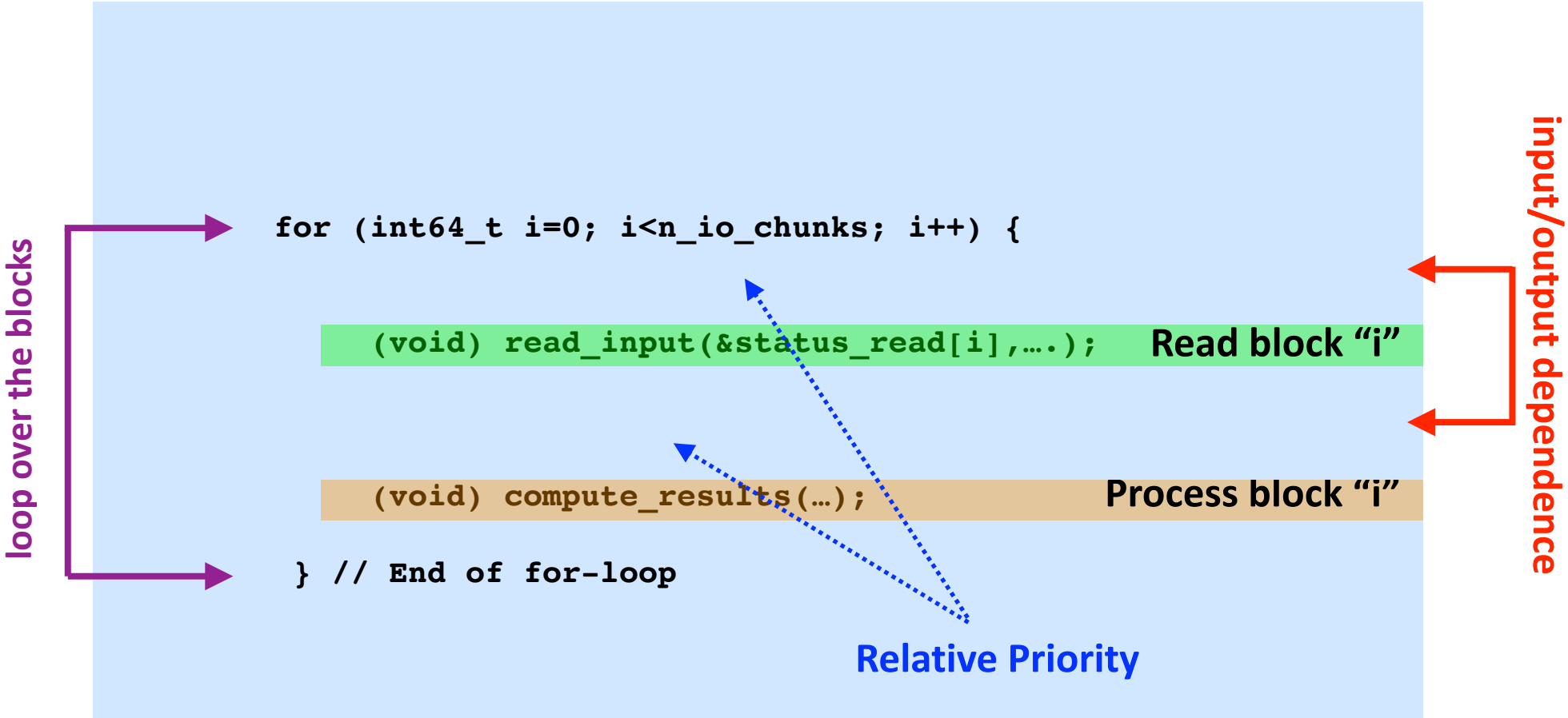
In this case, we need an out and in dependence pair

This is used to express that the computation depends on the I/O

Let's look at the code to see how this works

Copyright © Ruud van der Pas 2021, All Rights Reserved

The OpenMP Code Structure



Copyright © Ruud van der Pas 2021, All Rights Reserved

A Sorting Algorithm

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Quicksort Algorithm

A commonly used sorting algorithm

A divide-and-conquer strategy is used

The sequential algorithm:

1. Select a pivot
2. Re-order the elements in the array to be sorted such that:
   - All elements to the left of the pivot are smaller
3. Repeat for the parts to the left and right of the pivot

Copyright © Ruud van der Pas 2021, All Rights Reserved

An Example of the Quicksort Algorithm in Action

0	1	2	3	4	
8	5	7	3	9	select pivot

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Sequential Code

in the main program:

```
(void) quicksort(a, 0, n-1);
```

```
 1 int64_t quicksort(int64_t *a, int64_t lo, int64_t hi)
 2 {
 3   if ( lo < hi ) {
 4      p = partition(a, lo, hi);
 5
 6      (void) quicksort(a, lo, p - 1); // Left
 7
 8      (void) quicksort(a, p + 1, hi); // Right
 9   }
10   return(p);
11 }
```


Copyright © Ruud van der Pas 2021, All Rights Reserved

The OpenMP Implementation

In the main program

```
#pragma omp parallel …
  {
    #pragma omp single nowait
    { (void) ompQuicksort(a, 0, n-1);}
  } // End of parallel region
```

```
 1 void ompQuicksort(int64_t *a, int64_t lo, int64_t hi)
 2 {
 3   if ( lo < hi )
 4    {
 5      int64_t p = partitionArray(a, lo, hi);
 6      #pragma omp task …
 7        {(void) ompQuicksort(a, lo, p - 1);}
 8      #pragma omp task …
 9        {(void) ompQuicksort(a, p + 1, hi);}
10      #pragma omp taskwait
11    }
12 }
```

Left task

Right task

Copyright © Ruud van der Pas 2021, All Rights Reserved

Tuning The Parallel Sorting Algorithm

This algorithm requires some tuning
Tasks are not for free and carry some overhead
If an array section gets too small, it may be more efficient to switch to the sequential version
This is not so elegant, but it is efficient :-)

Copyright © Ruud van der Pas 2021, All Rights Reserved

Tuning Algorithms that Use Tasks

The tasking system is very dynamic
In general, with tasking, there are some things to keep in mind
How many tasks should be created?
Should small tasks be merged at some point?
Should tasks be forced to execute as soon as possible?
How about relative priorities?
Is it necessary to keep a task tied to the same thread?

Copyright © Ruud van der Pas 2021, All Rights Reserved

Tuning Controls for Tasking

Clause	Functionality
final(expr)	*If expr is true, stop generating more tasks*
mergeable	*Merge the data environment*
untied	*The task is not tied to a specific thread*
if (expr)	*If expr is false, generate an undeferred* task*
priority(hint)	*Give a priority hint to the runtime system*

**) In simple terms, an undeferred task is executed immediately*

Copyright © Ruud van der Pas 2021, All Rights Reserved

The Parallel Quicksort in Practice

Copyright © Ruud van der Pas 2021, All Rights Reserved

Preliminary Findings

We have run some tests with the parallel quicksort algorithm

Please keep in mind that this is very early work and incomplete

The quicksort algorithm is a good target for additional tuning

- *if-clause*
- *final-clause*
- *switch point for the sequential version*
- *how to handle idle threads*
- *NUMA controls*
- *…*

Copyright © Ruud van der Pas 2021, All Rights Reserved

Testing Conditions

We used an 8 core VM with 16 hardware threads

Intel Xeon Platinum 8167M CPU @ 2.00GHz ("Skylake")

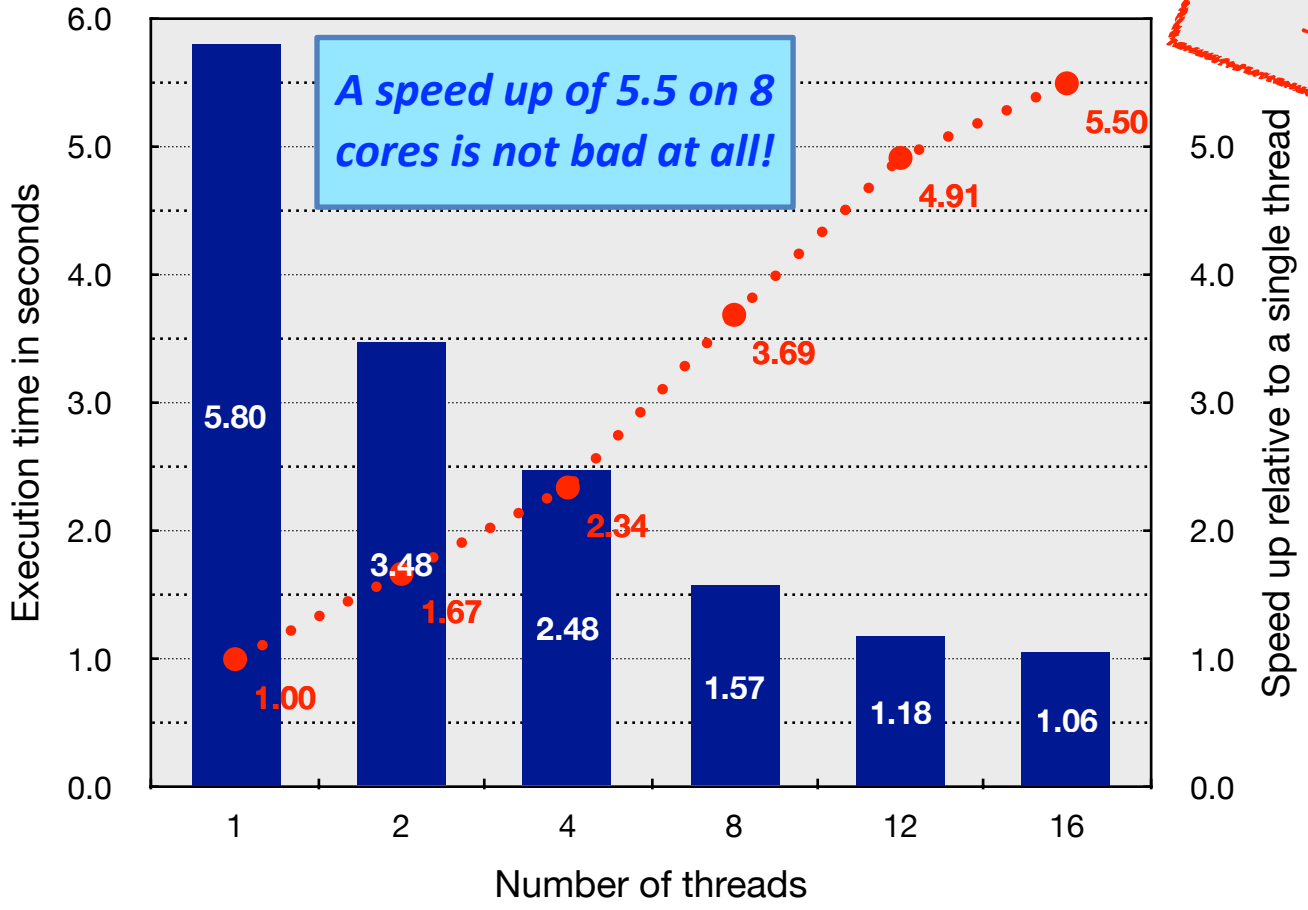
We sorted 40M 64 bit integers (320 MB of data)

The algorithmic parameters were:

- *final-clause: cutoff at 4000 elements (0.01% of the length)*
- *switch point for the sequential version: 400 (0.001% of the length)*

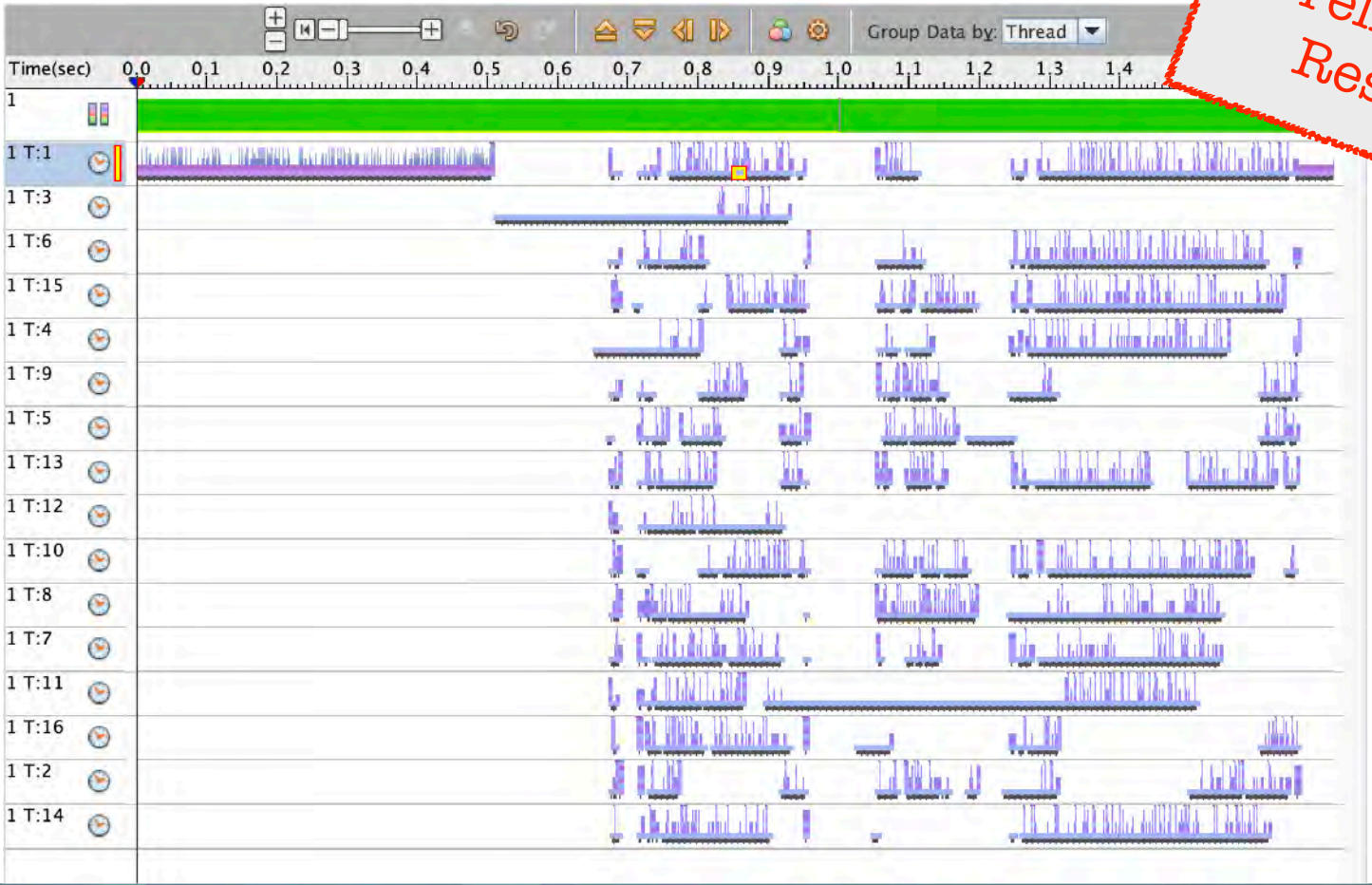
Copyright © Ruud van der Pas 2021, All Rights Reserved

The Performance Using 8 Cores



Copyright © Ruud van der Pas 2021, All Rights Reserved

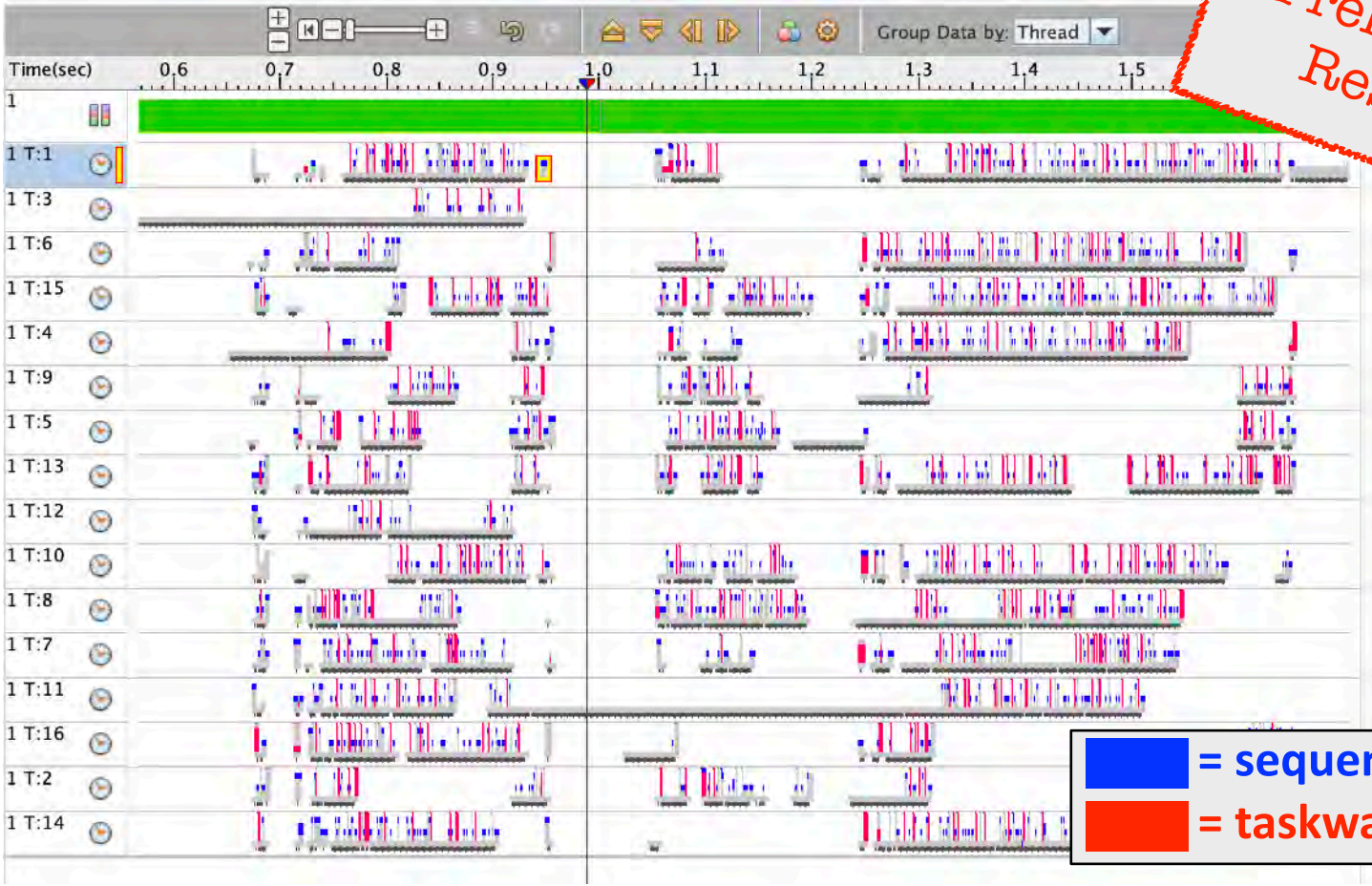
The Dynamic Behaviour Using 16 Threads



Preliminary Results!

Copyright © Ruud van der Pas 2021, All Rights Reserved

Use Color Coding to Distinguish the Phases



Preliminary Results!

= sequential version

= taskwait

Copyright © Ruud van der Pas 2021, All Rights Reserved

Many Stones Unturned

Copyright © Ruud van der Pas 2021, All Rights Reserved

What Was not Talked About

The focus of this talk has been on the functionality

Many important topics have not even been touched upon

For example, the taskloop, task_reduction, and taskyield features

Data scoping has been skipped as well

These are useful/relevant topics though and there is more, so please check for more information about tasking

Copyright © Ruud van der Pas 2021, All Rights Reserved

The SC21 OpenMP Tutorials

At SC21 there is a full day tutorial on tasking on November 14
"Mastering Tasking with OpenMP"

Additional OpenMP tutorials at SC21:

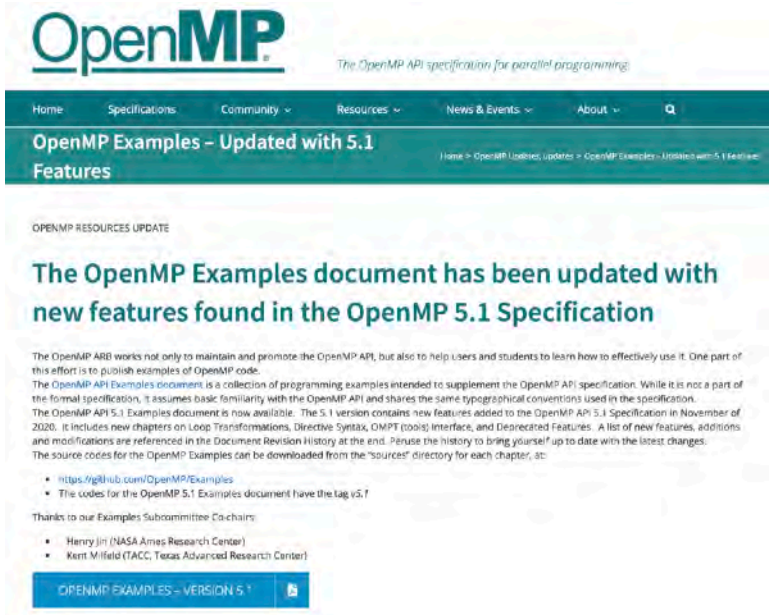
- ***The OpenMP Common Core*** *- A Hands-on Introduction*
- ***Advanced OpenMP*** *- Host Performance and 5.1 Features*
- ***Programming your GPU with OpenMP*** *- A Hands-on Introduction*
- ***See also:***

https://sc21.supercomputing.org/program/tutorials/#schedule

Copyright © Ruud van der Pas 2021, All Rights Reserved

The OpenMP Example Set

You can download a set of example codes
They have been updated for 5.1

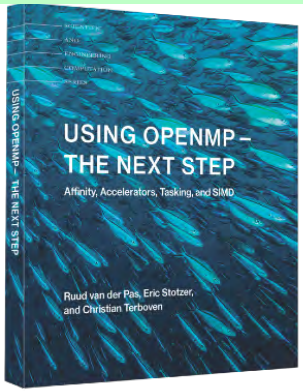


Copyright © Ruud van der Pas 2021, All Rights Reserved

Additional Reading Material

The book "Using OpenMP - The Next Step" covers many of the 4.5 features, including a full chapter on tasking

Although this covers 4.5, it provides a solid introduction into the more advanced topics and prepares you for 5.1



Copyright © Ruud van der Pas 2021, All Rights Reserved

Thank You And ... Stay Tuned!

Bad OpenMP
Does Not Scale

Ruud van der Pas

Webinar, September 22, 2021