Sorting Things Out
(with tasks)

Ruud van der Pas

Distinguished Engineer

SPARC Microelectronics

ORACLE

Santa Clara, CA, USA

SC'16 Talk at OpenMP Booth

Tuesday, November 15, 2016

The Dark Ages Of OpenMP

Big Brother 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

A solution was ugly. At best

Tasking Comes To The Rescue !

And we will show you how it all works

BUT !

No formal terminology, definitions, etc

A task is a chunk of independent work

You guarantee different tasks can be executed simultaneously

#pragma omp task
{“this is my task”}

The run time system decides on the scheduling of the tasks

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

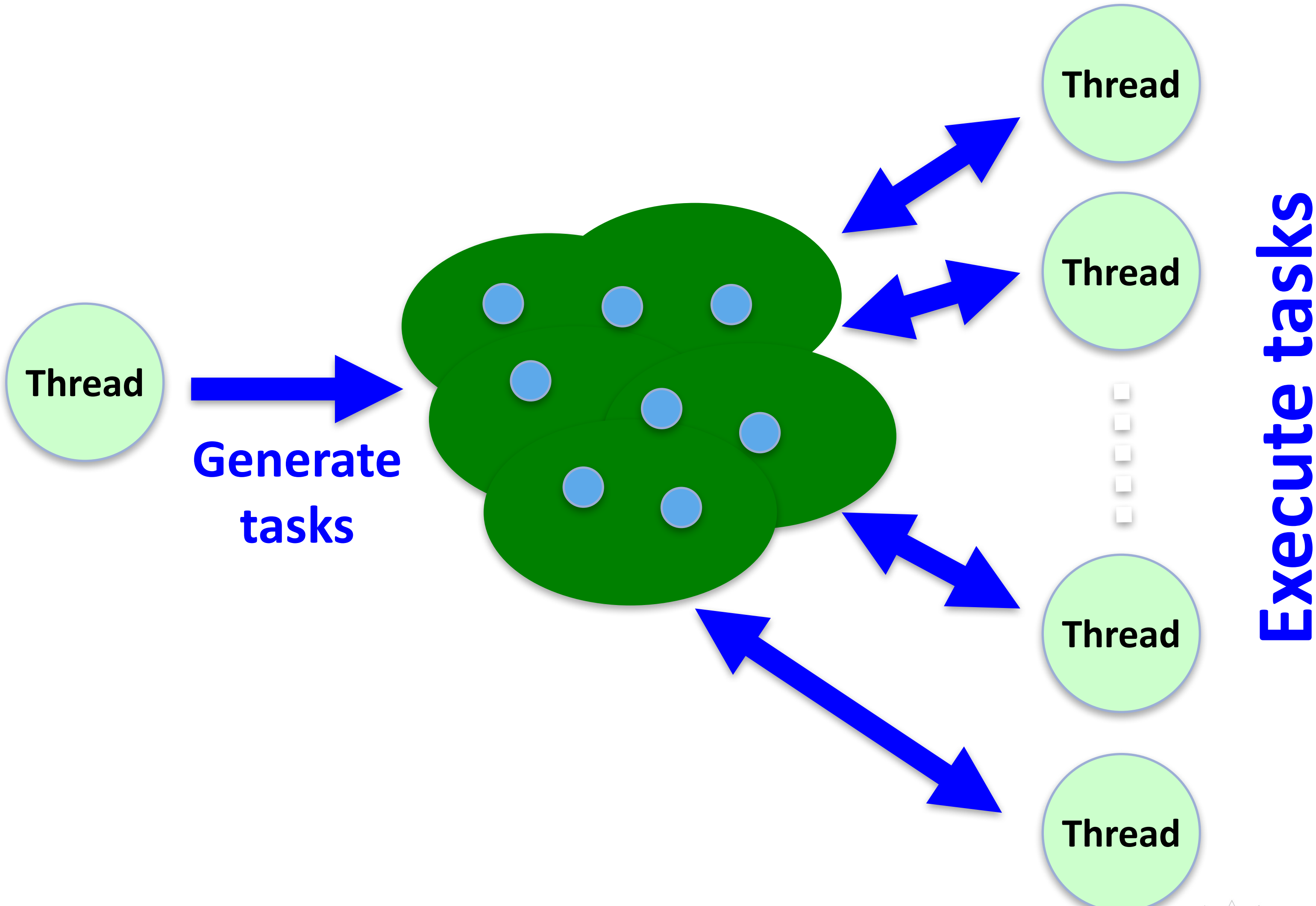
For those who love to study the fine print, the following advice:

RTFM!

And this is what it looks like:

Recognize The Fabulous Masters

The Tasking Concept In OpenMP



Who Does What And When ?

You

Use a pragma to specify where the tasks are

(The assumption is that all tasks can be executed independently)

The OpenMP runtime system

- When a thread encounters a task construct, a new task is generated
- The moment of execution of the task is up to the runtime system
- Execution can either be immediate or delayed
- Completion of a task can be enforced through **task synchronization**

Tasking Explained By Ways Of One Example

A Simple Plan

Your Task for Today:

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

Tasking Example/1

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

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

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

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

```
$ cc -fast hello.c
$ ./a.out
A race car
$
```

What will this program print ?

Tasking Example/2

```
#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 ?

Tasking Example/3

```
$ cc -xopenmp -fast 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

.....

But I have not observed this (yet)

Tasking Example/4

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

int main(int argc

    #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 ?

Tasking Example/5

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
```

But of course now only 1 thread executes ....

Tasking Example/6

```
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 ?

Tasking Example/7

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
$ ./a.out
A race car
$ ./a.out
A car race
$
```

Tasks can be executed in arbitrary order

Another Simple Plan

You did 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)

Tasking Example/8

```
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);
}
```

What will this program print using 2 threads ?

Tasking Example/9

```
$ cc -xopenmp -fast 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 execution point

Tasking Example/10

```
int main(int argc, cha

  #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 ?

Tasking Example/11

```
$ cc -xopenmp -fast 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
$
```

Tasks are executed first now

Sorting Things Out

The Quicksort Algorithm

A Commonly Used Algorithm Used For Sorting

Uses a divide and conquer strategy

Main steps:

Split the array through a pivot, such that

All elements to the left are smaller

All elements to the right are equal, or greater

Repeat for left and right part until done

A Simple Example/1

8	5	7	3	9	initial values
8	5	7	3	9	choose pivot, keep index
8	5	7	3	9	swap pivot and last element
8	5	9	3	7	scan array, swap if smaller
8	5	9	3	7	5 < 7 => move to position 0
5	8	9	3	7	3 < 7 => move to position 1
5	3	9	8	7	continue, but nothing found

A Simple Example/2

5	3	9	8	7	restore pivot
5	3	7	8	9	pivot is in final position

7

repeat for left branch	repeat for right branch
5 3	8 9
OpenMP task	OpenMP task

The Recursive Sequential Code

```
void Quicksort(int64_t *a, int64_t lo, int64_t hi)
{
  if ( lo < hi ) {
    int64_t p = partitionArray(a, lo, hi);

    (void) Quicksort(a, lo, p - 1); // Left branch

    (void) Quicksort(a, p + 1, hi); // Right branch
  }
}
```

And Now With Tasks

OpenMP

```
void Quicksort(int64_t *a, int64_t lo, int64_t hi)
{
  if ( lo < hi ) {
   int64_t p = partitionArray(a, lo, hi);

   #pragma omp task shared(a) firstprivate(lo,p)
    {(void) Quicksort(a, lo, p - 1);} // Left branch

   #pragma omp task shared(a) firstprivate(hi,p)
    {(void) Quicksort(a, p + 1, hi);} // Right branch

   #pragma omp taskwait
  }
}
```

Including The Driver Part

```
#pragma omp parallel default(none) shared(a,nelements)
{
  #pragma omp single nowait
     { (void) Quicksort(a, 0, nelements-1); }
} // End of parallel region
```

```
void Quicksort(int64_t *a, int64_t lo, int64_t hi)
{
  if ( lo < hi ) {
   int64_t p = partitionArray(a, lo, hi);

   #pragma omp task default(none) firstprivate(a,lo,p)
    {(void) Quicksort(a, lo, p - 1);} // Left branch

   #pragma omp task default(none) firstprivate(a,hi,p)
    {(void) Quicksort(a, p + 1, hi);} // Right branch
  }
}
```

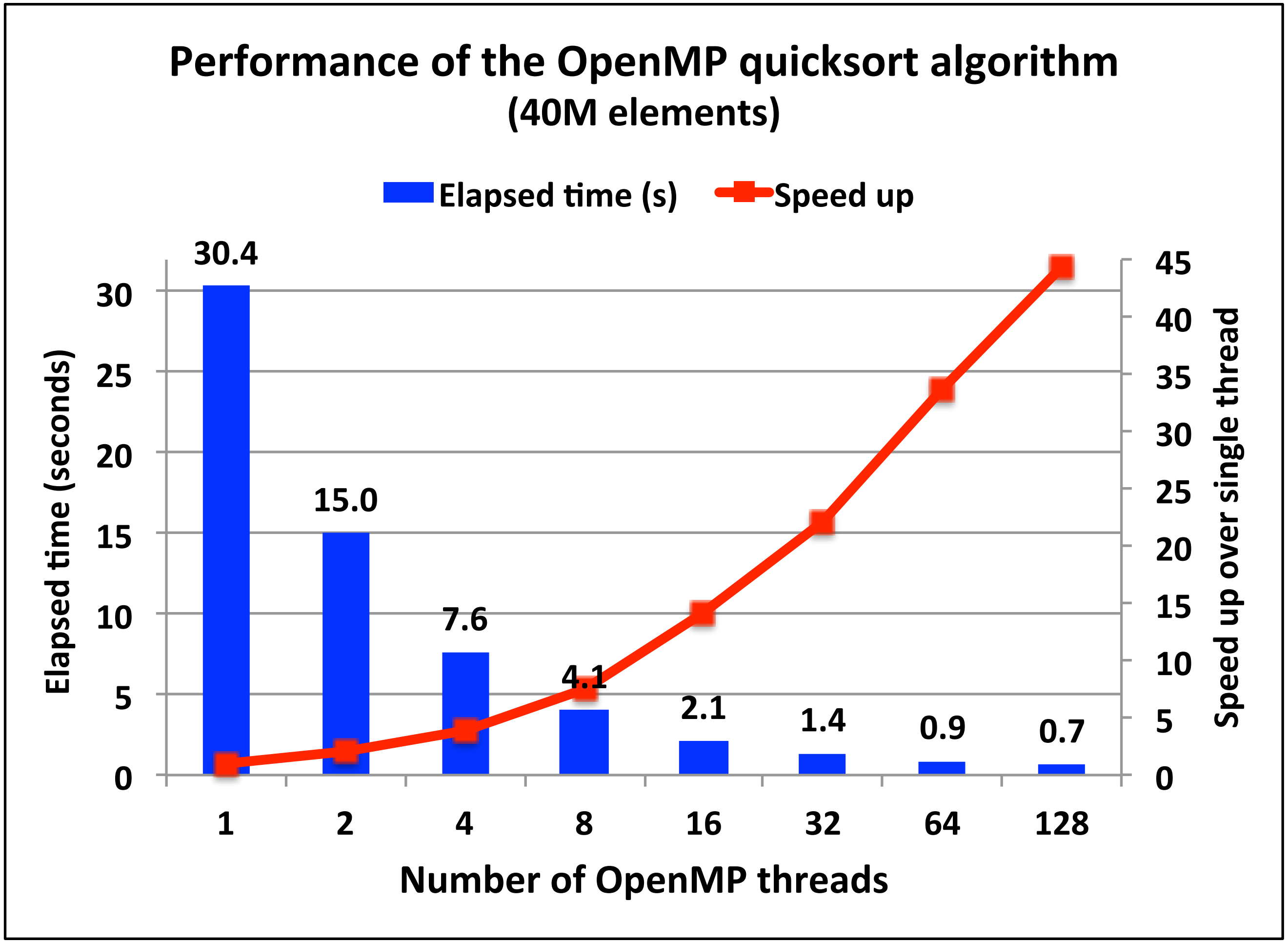
Fine Tuning The Algorithm

When the array section gets too small, it is better to switch to the sequential algorithm

May also consider the use of the if-clause plus the mergeable and final clauses

Some experimentation is recommended ;-)

A Performance Example *



**) SPARC M7-8 server @ 4.1 GHz*

Summary

Big Brother Does Not Need To Know Everything

For certain types of algorithms

Tasking is ideally suitable

Optimal performance may require some fine tuning

But ……. Remember:

OpenMP

Recognize The Fabulous Masters

SC16
Salt Lake City, Utah
hpc matters.

Thank You And ..... Stay Tuned !

ruud.vanderpas@oracle.com