Sorting Things Out
(with tasks)

Ruud van der Pas
Distinguished Engineer
SPARC Microelectronics

Santa Clara, CA, USA

SC’16 Talk at OpenMP Booth
Tuesday, November 15, 2016
The Dark Ages Of OpenMP

<table>
<thead>
<tr>
<th>Big Brother Had To Know Everything</th>
</tr>
</thead>
<tbody>
<tr>
<td>And in advance, (right) before execution</td>
</tr>
<tr>
<td>For example, the loop length, number of parallel sections, etc</td>
</tr>
<tr>
<td>Gets hard with more dynamic problems like processing linked lists, divide and conquer, recursion</td>
</tr>
<tr>
<td>A solution was ugly. At best</td>
</tr>
</tbody>
</table>
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

```c
#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:
RTFM!

OpenMP coders do it with #pragmas

OpenMP coders do it with #pragmas

Recognize The Fabulous Masters
The Tasking Concept In OpenMP

Thread

Generate tasks

Execute tasks

Thread

Thread

Thread

Thread
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

$\texttt{cc -fast hello.c}$
$\texttt{./a.out}$

\texttt{A race car}

What will this program print?
#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);
}
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)
#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);
}
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 .......
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);
}
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)
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);
}
Tasks are executed at a task execution point

$ 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
$
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);
}
$ 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

OpenMP Booth – Sorting Things Out With Tasks
Ruud van der Pas
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

<table>
<thead>
<tr>
<th>8</th>
<th>5</th>
<th>7</th>
<th>3</th>
<th>9</th>
<th>initial values</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>choose pivot, keep index</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>swap pivot and last element</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>7</td>
<td>scan array, swap if smaller</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>7</td>
<td>5 &lt; 7 =&gt; move to position 0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>7</td>
<td>3 &lt; 7 =&gt; move to position 1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>continue, but nothing found</td>
</tr>
</tbody>
</table>
A Simple Example/2

5 3 9 8 7  
restore pivot

5 3 7 8 9  
pivot is in final position

repeat for left branch

5 3

Repeat for right branch

8 9

OpenMP task

OpenMP task
The Recursive Sequential Code

```c
1 void Quicksort(int64_t *a, int64_t lo, int64_t hi) {
2   if ( lo < hi ) {
3     int64_t p = partitionArray(a, lo, hi);
4
5     (void) Quicksort(a, lo, p - 1); // Left branch
6
7     (void) Quicksort(a, p + 1, hi); // Right branch
8   }
9 }
10 }
```
And Now With Tasks

```c
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

```cpp
#include "openmp.h"

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
    }
}
```

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

Performance of the OpenMP quicksort algorithm
(40M elements)

Elapsed time (s) | Speed up
--- | ---
30.4 | 45
15.0 | 30
7.6 | 20
4.1 | 15
2.1 | 10
1.4 | 7
0.9 | 5
0.7 | 4

Elapsed time (seconds)

Number of OpenMP threads

*) 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:
RTFM!

OpenMP coders do it with #pragmas

OpenMP coders do it with #pragmas

Recognize The Fabulous Masters
Thank You And ..... Stay Tuned!

ruud.vanderpas@oracle.com