More on OpenMP Tasking

Christian Terboven

IT Center, RWTH Aachen University
terboven@itc.rwth-aachen.de
Sudoku
Parallel Brute-force Sudoku

- Lets solve Sudoku puzzles with brute multi-core force

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>8</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>12</td>
<td>3</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>11</td>
<td>15</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>10</td>
<td>16</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>12</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>15</td>
<td>11</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1</td>
<td>2</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>15</td>
<td>16</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>12</td>
<td>13</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>

- (1) Search an empty field
- (2) Try all numbers:
  - (2 a) Check Sudoku
    - If invalid: skip
    - If valid: Go to next field
- Wait for completion
## Parallel Brute-force Sudoku

### Lets solve Sudoku puzzles with brute multi-core force

1. Search an empty field

```
#pragma omp parallel
#pragma omp single
```

such that one task starts the execution of the algorithm

2. Try all numbers:
   2a. Check Sudoku
   - If invalid: skip
   - If valid:
     
     Go to next field

3. Wait for completion

```
#pragma omp task
```

needs to work on a new copy of the Sudoku board

```
#pragma omp taskwait
```

wait for all child tasks
Parallel Brute-force Sudoku

- OpenMP parallel region creates a team of threads

```c
#pragma omp parallel
{
#pragma omp single
  solve_parallel(0, 0, sudoku2, false);
} // end omp parallel
```

→ Single construct: One thread enters the execution of `solve_parallel`
→ the other threads wait at the end of the `single` ...
  → ... and are ready to pick up threads „from the work queue“

- Syntactic sugar (either you like it or you don‘t)

```c
#pragma omp parallel sections
{
  solve_parallel(0, 0, sudoku2, false);
} // end omp parallel
```
The actual implementation

```cpp
for (int i = 1; i <= sudoku->getFieldSize(); i++) {
    if (!sudoku->check(x, y, i)) {
        #pragma omp task firstprivate(i, x, y, sudoku)
        {
            // create from copy constructor
            CSudokuBoard new_sudoku(*sudoku);
            new_sudoku.set(y, x, i);
            if (solve_parallel(x+1, y, &new_sudoku)) {
                new_sudoku.printBoard();
            }
        } // end omp task
    }
} // end omp task
```

```
#pragma omp taskwait
```

```
#pragma omp taskwait
wait for all child tasks
```
Performance Evaluation

Sudoku on 2x Intel Xeon E5-2650 @2.0 GHz

- Intel C++ 13.1, scatter binding
- speedup: Intel C++ 13.1, scatter binding

Is this the best we can do?
Event-based profiling gives a good overview:

Every thread is executing ~1.3m tasks...

... in ~5.7 seconds.

=> average duration of a task is ~4.4 μs

Tracing gives more details:

Tasks get much smaller down the call-stack.
Performance Analysis

Event-based profiling gives a good overview:

- Performance and Scalability Tuning
  Idea: If you have created sufficiently many tasks to make your cores busy, stop creating more tasks!
  - if-clause
  - final-clause, mergeable-clause
  - natively in your program code

Example: stop recursion

... in ~5.7 seconds.

=> average duration of a task is ~4.4 μs

Tracing gives more details:

- Tasks get much smaller down the call-stack.
Performance Evaluation

Sudoku on 2x Intel Xeon E5-2650 @2.0 GHz

- Intel C++ 13.1, scatter binding
- Intel C++ 13.1, scatter binding, cutoff
- speedup: Intel C++ 13.1, scatter binding
- speedup: Intel C++ 13.1, scatter binding, cutoff

Runtime [sec] for 16x16

Speed up: Intel C++ 13.1, scatter binding, cutoff

OpenMP Tasking In-Depth
C. Terboven | IT Center der RWTH Aachen University
Scheduling
Tasks in OpenMP: Scheduling

- Default: Tasks are tied to the thread that first executes them → not necessarily the creator. Scheduling constraints:
  - Only the thread a task is tied to can execute it
  - A task can only be suspended at task scheduling points
    - Task creation, task finish, taskwait, barrier, taskyield
  - If task is not suspended in a barrier, executing thread can only switch to a direct descendant of all tasks tied to the thread

- Tasks created with the untied clause are never tied
  - Resume at task scheduling points possibly by different thread
  - No scheduling restrictions, e.g., can be suspended at any point
  - But: More freedom to the implementation, e.g., load balancing
Unsafe use of untied Tasks

- Problem: Because untied tasks may migrate between threads at any point, thread-centric constructs can yield unexpected results.

- Remember when using untied tasks:
  - Avoid `threadprivate` variable
  - Avoid any use of thread-ids (i.e., `omp_get_thread_num()`)
  - Be careful with `critical region` and `locks`

- Simple Solution:
  - Create a tied task region with
    
    ```
    #pragma omp task if(0)
    ```
The taskyield Directive

- **The taskyield directive** specifies that the current task can be suspended in favor of execution of a different task.

→ Hint to the runtime for optimization and/or deadlock prevention

<table>
<thead>
<tr>
<th>C/C++</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp taskyield</td>
<td>!$omp taskyield</td>
</tr>
</tbody>
</table>
#include <omp.h>

void something_useful();
void something_critical();

void foo(omp_lock_t * lock, int n)
{
    for(int i = 0; i < n; i++)
        #pragma omp task
        {
            something_useful();
            while( !omp_test_lock(lock) ) {
                #pragma omp taskyield
            }
            something_critical();
            omp_unset_lock(lock);
        }
}
#include <omp.h>

void something_useful();
void something_critical();

void foo(omp_lock_t * lock, int n)
{
    for(int i = 0; i < n; i++)
        #pragma omp task
        {
            something_useful();
            while( !omp_test_lock(lock) ) {
                #pragma omp taskyield
            }
            something_critical();
            omp_unset_lock(lock);
        }
}
Tasks and Dependencies
Tasks and Dependencies

- Catchy example: Building a house
Tasks and Dependencies

- Task dependencies constrain execution order and times for tasks
- Fine-grained synchronization of tasks

```c
int x = 0;
#pragma omp parallel
#pragma omp single
{
    #pragma omp task
    std::cout << x << std::endl;
    #pragma omp taskwait
    #pragma omp task
    x++;
}
```

```
int x = 0;
#pragma omp parallel
#pragma omp single
{
    #pragma omp task
    std::cout << x << std::endl;
    #pragma omp taskwait
    #pragma omp task
    x++;
}
```

### Dependencies

- Task's creation time
- Task's execution time
void cholesky(int ts, int nt, double* a[nt][nt]) {
    for (int k = 0; k < nt; k++) {
        // Diagonal Block factorization
        potrf(a[k][k], ts, ts);
        // Triangular systems
        for (int i = k + 1; i < nt; i++) {
            #pragma omp task
            trsm(a[k][k], a[k][i], ts, ts);
        }
        #pragma omp taskwait
        // Update trailing matrix
        for (int i = k + 1; i < nt; i++) {
            for (int j = k + 1; j < i; j++) {
                #pragma omp task
                dgemm(a[k][i], a[k][j], a[j][i], ts, ts);
            }
            #pragma omp task
            syrk(a[k][i], a[i][i], ts, ts);
        }
        #pragma omp taskwait
    }
}
Questions?