OpenMP 4.0 & Beyond

Christian Terboven
terboven@itc.rwth-aachen.de

IT Center, RWTH Aachen University

Member of the OpenMP Language Committee (since 2006)
HPC at RWTH Aachen University

- Tier 0 – PRACE
- Tier 1 – Gauss Center
- Tier 1.5 – JARA-HPC
- Tier 2 – Gauss Alliance
The IT Center is a central institution of RWTH Aachen University,
→ supporting all major processes at the university,
→ providing basic and individually tailored IT services for all univ. institutions,
→ supports the Simulation Sciences as an important pillar of the RWTH strategy.

CSE Department:
→ Chair f. High Perf. Computing
→ Chair f. Immersive Viz.
New Features in OpenMP 4.0
OpenMP 4.0 brought many new features

- **Device constructs**: support for compute acceleration devices and heterogeneous computing
Execution + Data Model

- **Data environment is lexically scoped**
  - Data environment is destroyed at closing curly brace
  - Allocated buffers/data are automatically released

- **Use target construct to**
  - Transfer control from the host to the device
  - Establish a data environment (if not yet done)
  - Host thread waits until offloaded region completed

```
#pragma omp target
 alloc(…)
 from(…)
 to(…)

map(alloc:...) \ map(to:...) \ map(from:...) { ... }
```
Device constructs

This is a very simple example, but you see functionality similar to OpenACC 1.0. Device support has to come with the implementation.
SAXPY: Serial (Host)

```c
int main(int argc, const char* argv[]) {
    int n = 10240; float a = 2.0f; float b = 3.0f;
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    // Initialize x, y
    // Run SAXPY TWICE
    for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
    }
    // y is needed and modified on the host here
    for (int i = 0; i < n; ++i){
        y[i] = b*x[i] + y[i];
    }
    free(x); free(y); return 0;
}
```

OpenMP 4.0 & Beyond
Christian Terboven | IT Center der RWTH Aachen University
```c
int main(int argc, const char* argv[]) {
    int n = 10240; float a = 2.0f; float b = 3.0f;
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    // Initialize x, y

    // Run SAXPY TWICE
    #pragma acc data copyin(x[0:n])
    {
        #pragma acc parallel copy(y[0:n])
        #pragma acc loop
        for (int i = 0; i < n; ++i){
            y[i] = a*x[i] + y[i];
        }
        // y is needed and modified on the host here
        #pragma acc parallel copy(y[0:n])
        #pragma acc loop
        for (int i = 0; i < n; ++i){
            y[i] = b*x[i] + y[i];
        }
    }

    free(x); free(y); return 0;
}
```

OpenMP 4.0 & Beyond
Christian Terboven | IT Center der RWTH Aachen University
int main(int argc, const char* argv[]) {
    int n = 10240; float a = 2.0f; float b = 3.0f;
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    // Initialize x, y

    // Run SAXPY TWICE
    #pragma omp target data map(to:x[0:n])
    {
        #pragma omp target map(tofrom:y[0:n])
        #pragma omp teams
        #pragma omp distribute
        #pragma omp parallel for
        for (int i = 0; i < n; ++i){
            y[i] = a*x[i] + y[i];
        }
        // y is needed and modified on the host here
        #pragma omp target map(tofrom:y[0:n])
        #pragma omp teams
        #pragma omp distribute
        #pragma omp parallel for
        for (int i = 0; i < n; ++i){
            y[i] = b*x[i] + y[i];
        }
        free(x);
        free(y);
        return 0;
    }
int main(int argc, const char* argv[]) {
    int n = 10240; float a = 2.0f; float b = 3.0f;
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    // Initialize x, y

    // Run SAXPY TWICE
    #pragma omp target data map(to:x[0:n])
    {
        #pragma omp target map(tofrom:y[0:n])
        #pragma omp parallel for
        for(int i = 0; i < n; ++i){
            y[i] = a*x[i] + y[i];
        }
    }

    // y is needed and modified on the host here
    #pragma omp target map(tofrom:y[0:n])
    #pragma omp parallel for
    for(int i = 0; i < n; ++i){
        y[i] = b*x[i] + y[i];
    }

    free(x); free(y); return 0;
}
## Constructs and Clauses: *MP vs. *ACC

<table>
<thead>
<tr>
<th>OpenMP</th>
<th>OpenACC</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>parallel</td>
<td>Offload of computational work to the device (sync.)</td>
</tr>
<tr>
<td>teams, parallel</td>
<td>parallel</td>
<td>Creation of in parallel running threads</td>
</tr>
<tr>
<td>-</td>
<td>Kernel</td>
<td>Automatic parallelization by the compiler</td>
</tr>
<tr>
<td>target data</td>
<td>data</td>
<td>Structured data management between host &amp; device</td>
</tr>
<tr>
<td>distribute, do, for, simd</td>
<td>loop</td>
<td>Worksharing across parallel units</td>
</tr>
<tr>
<td>-</td>
<td>host data</td>
<td>Interoperability, i.e. with CUDA</td>
</tr>
<tr>
<td>-</td>
<td>cache</td>
<td>Move objects closer in memory to execution units</td>
</tr>
<tr>
<td>target update</td>
<td>update</td>
<td>Data movement between host &amp; device within data env.</td>
</tr>
<tr>
<td>declare target</td>
<td>declare</td>
<td>Declaration of global/static/extern objects</td>
</tr>
<tr>
<td>-</td>
<td>enter data</td>
<td>Unstructured data movement to the device</td>
</tr>
<tr>
<td>-</td>
<td>exit data</td>
<td>Unstructured data movement from the device</td>
</tr>
</tbody>
</table>
## Constructs and Clauses: *MP vs. *ACC

<table>
<thead>
<tr>
<th>OpenMP</th>
<th>OpenACC</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>tasks</td>
<td>-</td>
<td>Creation of explicit tasks for Task Parallelism</td>
</tr>
<tr>
<td>task depend</td>
<td>async(int)</td>
<td>Asynchronous execution with dependencies</td>
</tr>
<tr>
<td>-</td>
<td>wait</td>
<td>Synchronization of Streams</td>
</tr>
<tr>
<td>parallel or team</td>
<td>parallel in parallel</td>
<td>Nested parallelism</td>
</tr>
<tr>
<td>-</td>
<td>tile</td>
<td>Strip-mining of data collections</td>
</tr>
<tr>
<td>-</td>
<td>device_type</td>
<td>Device-specific tuning of clauses</td>
</tr>
<tr>
<td>atomic</td>
<td>atomic</td>
<td>Atomic operations</td>
</tr>
<tr>
<td>sections, critical, ...</td>
<td>Non-iterative worksharing, critical sections, synchronization, control flow of threads, ...</td>
<td></td>
</tr>
</tbody>
</table>

- **OpenACC** is best for GPU-programming right now. **OpenMP** is much broader (as we will see) and support more device types, but currently no GPUs in any commercial implementation.

---

**Paper at Euro-Par 2014:** S. Wienke, C. Terboven, J. C. Beyer and M. S. Müller:

A Pattern-based Comparison of OpenACC and OpenMP for Accelerator Computing
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
If you know about vectorization, you will recognize what is going on here. If not, you probably leave a lot of performance on the table…
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
- **Cancellation:** directives to request the termination of the current OpenMP region
Cancellation Constructs

- **Two constructs:**
  - Activate cancellation:
    - C/C++: `#pragma omp cancel`
    - Fortran: `!$omp cancel`
  - Check for cancellation:
    - C/C++: `#pragma omp cancellation point`
    - Fortran: `!$omp cancellation point`

- **Check for cancellation only a certain points**
  - Avoid unnecessary overheads
  - Programmers need to reason about cancellation
  - Cleanup code needs to be added manually
Stopping midstream

This feature enables new classes of (irregular) problems to be exploited with OpenMP (tasks).

```c
binary_tree_t* search_tree(binary_tree_t* tree, int val) {
    if (tree->value == val)
        found = tree;
    else {
        #pragma omp task shared(found)
        {
            binary_tree_t* found_left;
            found_left = search_tree(tree->left, val);
            if (found_left) {
                #pragma omp atomic write
                found = found_left;
                #pragma omp cancel taskgroup
            }
        } // end omp task, followed by similar code for "right" side
    }
}
```
Cancellation Semantics

Thread A

Thread B

Thread C

parallel region

cancellation point

cancellation point

cancellation point

cancel

cancel

?
Cancellation Semantics

Thread A
- cancel

Thread B
- cancellation point

Thread C
- cancellation point

parallel region

OpenMP 4.0 & Beyond
Christian Terboven | IT Center der RWTH Aachen University
Cancellation Semantics

Thread A
cancel

Thread B

Thread C
cancellation point

parallel region
Cancellation Semantics

Thread A

Thread B

Thread C

cancellation point

cancellation point

cancellation point

parallel region
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
- **Cancellation:** directives to request the termination of the current OpenMP region
- **Task Dependencies:** clauses to express additional constraints on the scheduling of tasks w.r.t. their siblings
Talk the talk, task the Task

Degree of parallelism exploitable in this concrete example: T2 and T3 (2 tasks), T1 of next iteration has to wait for them to be completed before T2 and T3 can be executed.

T2 and T3 can be executed in parallel.

```c
void process_in_parallel() {
    #pragma omp parallel
    #pragma omp single
    {
        int x = 1;
        ...
        for (int i = 0; i < T; ++i) {
            #pragma omp task shared(x, ...) depend(out: x) // T1
            preprocess_some_data(...);
            #pragma omp task shared(x, ...) depend(in: x)  // T2
            do_something_with_data(...);
            #pragma omp task shared(x, ...) depend(in: x)  // T3
            do_something_independent_with_data(...);
        }
    } // end omp single, omp parallel
}
```

Very simple example. But you should see the new possibilities opened up. If you do not, OmpSs has encouraging examples.
Concurrent Execution w/ Dependencies

- The following allows for more parallelism, as there is one $i$ per thread. Hence, two tasks may be active per thread.

```c
void process_in_parallel() {
    #pragma omp parallel
    {
        #pragma omp for
        for (int i = 0; i < T; ++i) {
            #pragma omp task depend(out: i)
            preprocess_some_data(...);
            #pragma omp task depend(in: i)
            do_something_with_data(...);
            #pragma omp task depend(in: i)
            do_something_independent_with_data(...);
        }
    } // end omp parallel
}
```
Concurrent Execution w/ Dependencies

- The following allows for even more parallelism, as there now can be two tasks active per thread per $i$-th iteration.

```c
void process_in_parallel() {
    #pragma omp parallel
    #pragma omp single
    {
        for (int i = 0; i < T; ++i) {
            #pragma omp task firstprivate(i)
            {
                #pragma omp task depend(out: i)
                preprocess_some_data(...);
                #pragma omp task depend(in: i)
                do_something_with_data(...);
                #pragma omp task depend(in: i)
                do_something_independent_with_data(...);
            } // end omp task
        } // end omp single, end omp parallel
    }
}
```
void blocked_cholesky( int NB, float A[NB][NB] ) {
    int i, j, k;
    for (k=0; k<NB; k++) {
        #pragma omp task depend(inout:A[k][k])
        spotrf (A[k][k]) ;
        for (i=k+1; i<NT; i++)
            #pragma omp task depend(in:A[k][k]) depend(inout:A[k][i])
            strsm (A[k][k], A[k][i]);
        // update trailing submatrix
        for (i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++)
                #pragma omp task depend(in:A[k][i],A[k][j])
                    depend(inout:A[j][i])
                sgemm( A[k][i], A[k][j], A[j][i]);
            #pragma omp task depend(in:A[k][i]) depend(inout:A[i][i])
                ssyrk (A[k][i], A[i][i]);
        }
    }
}
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
- **Cancellation:** directives to request the termination of the current OpenMP region
- **Task Dependencies:** clauses to express additional constraints on the scheduling of tasks w.r.t. their siblings
- **Thread Affinity:** both coarse- and fine-grained control of where OpenMP threads are executed
OpenMP 4.0: Places + Binding Policies

- **Define OpenMP Places**
  - set of OpenMP threads running on one or more processors
  - can be defined by the user, i.e. `OMP_PLACES=cores`

- **Define a set of OpenMP Thread Affinity Policies**
  - **SPREAD**: spread OpenMP threads evenly among the places, partition the place list
  - **CLOSE**: pack OpenMP threads near master thread
  - **MASTER**: collocate OpenMP thread with master thread

- **Goals**
  - user has a way to specify where to execute OpenMP threads for
  - locality between OpenMP threads / less false sharing / memory bandwidth
OpenMP Places

- **Assume the following machine:**

  > 2 sockets, 4 cores per socket, 4 hyper-threads per core

- **Abstract names for OMP_PLACES:**
  
  > threads: Each place corresponds to a single hardware thread on the target machine.

  > cores: Each place corresponds to a single core (having one or more hardware threads) on the target machine.

  > sockets: Each place corresponds to a single socket (consisting of one or more cores) on the target machine.
Abstract names combined with the strategies allow for ease of use. However, if you do need full control, you can get it.

OpenMP 4.0 & Beyond
Christian Terboven | IT Center der RWTH Aachen University
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
- **Cancellation:** directives to request the termination of the current OpenMP region
- **Task Dependencies:** clauses to express additional constraints on the scheduling of tasks w.r.t. their siblings
- **Thread Affinity:** both coarse- and fine-grained control of where OpenMP threads are executed
- **User-defined reductions:** finally (!) support for more than * and + operators in the reduction clause
Reducers everywhere

```c++
#pragma omp declare reduction
(merge: std::vector<int> :
  omp_out.insert(omp_out.end(), omp_in.begin(), omp_in.end())
)

void schedule (std::vector<int> &v, std::vector<int> &filtered)
{
  #pragma omp parallel for reduction(merge: filtered)
  for (std::vector<int>::iterator it = v.begin(); it < v.end(); it++)
    if ( filter(*it) )
      filtered.push_back(*it);
}
```

- This was missing for C++ codes.
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
- **Cancellation:** directives to request the termination of the current OpenMP region
- **Task Dependencies:** clauses to express additional constraints on the scheduling of tasks w.r.t. their siblings
- **Thread Affinity:** both coarse- and fine-grained control of where OpenMP threads are executed
- **User-defined reductions:** finally (!) support for more than * and + operators in the reduction clause
- **Sequentially consistent atomics:** bless you!
OpenMP 4.0 brought many new features

- **Device constructs:** support for compute acceleration devices and heterogeneous computing
- **SIMD constructs:** portable description of SIMD expression and their combination with parallelization directives
- **Cancellation:** directives to request the termination of the current OpenMP region
- **Task Dependencies:** clauses to express additional constraints on the scheduling of tasks w.r.t. their siblings
- **Thread Affinity:** both coarse- and fine-grained control of where OpenMP threads are executed
- **User-defined reductions:** finally (!) support for more than * and + operators in the reduction clause
- **Sequentially consistent atomics:** bless you!

… and even more (minor) things, i.e. improved Fortran 2003 support!
My Thoughts on Future Directions
… but, honestly, what is OpenMP now?

- The OpenMP mission is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable.
Future Directions of OpenMP

- **Already released: OpenMP Tools Interface Technical Report**

- **The OpenMP Language Committee is active**
  - (multiple) weekly conference calls
  - (at least) three face-to-face meetings per year
  - OpenMP 4.1 targeted for release at SC15

- **Aims for OpenMP 4.1**
  - Many improvements to the accelerator support, i.e. unstructured data movement.
  - Initial Support for Memory Affinity
  - Interoperability with Posix-Threads
  - Reductions for Tasks
  - Some new features, i.e. DOACROSS loops, maybe unrolling and blocking
Where ever you go, OpenMP will be there
Thank you for your attention.

Some examples are taken from the *Advanced OpenMP Tutorial* regularly given at SC and ISC conferences together with Bronis R. de Supinski, Michael Klemm, Eric Stotzer and Ruud van der Pas.