Using OpenMP for Intranode Parallelism

OpenMP 4.0 and the Future of OpenMP

Bronis R. de Supinski

Paul Petersen

Lawrence Livermore National Laboratory

intel

OpenMP 4.0 ratified last month

- End of a long road? A brief rest stop along the way…
- Addresses several major open issues for OpenMP
- Does not break existing code
- Includes 106 passed tickets
  - →Focus on major tickets initially
  - →Builds on two comment drafts (“RC1” and “RC2”)
  - →Many small tickets after RC2, a few large ones
- Final vote scheduled for July 11
- Already starting work on OpenMP 5.0

Overview of major 4.0 additions

- Device constructs
- SIMD constructs
- Cancellation
- Task dependences and task groups
- Thread affinity control
- User-defined reductions
- Initial support for Fortran 2003
- Support for array sections (including in C and C++)
- Sequentially consistent atomics
- Display of initial OpenMP internal control variables

OpenMP 4.0 provides unified support for a wide range of devices

- Use `target` directive to offload a region should be offloaded

```
#pragma omp target [clause [[,] clause] …]
```

- Creates new data environment from enclosing device data environment
- Clauses support data movement and conditional offloading
  - `device` supports offload to a device other than default
  - `map` ensures variables accessible on device
    - Does not assume copies are made – memory may be shared with host
    - Does not copy if present in enclosing device data environment
  - `if` supports running on host if amount of work is small
- Other constructs support device data environment
  - `target data` places `map` list items in device data environment
  - `target update` ensures variable is consistent in host and device

Several other device constructs support simple offload of full-featured code

- Use `target declare` directive to create device v

```
#pragma omp declare target
```

  → Can be applied to functions and global variables

  → Required for UDRs that use functions and execute on device

- N `et` region

```
#pragma omp teams [clause [[,] clause] …]
```

  → Work across teams only synchronized at end of `target` region

  → Useful for GPUs (corresponds to thread blocks)

- d e teams

```
#pragma omp distribute [clause [[,] clause] …]
```

- Several combined constructs (post-RC2) simplify device use

Reminiscent of our roots, OpenMP 4.0 provides portable SIMD constructs

- Use `simd` directive to indicate a loop should be SIMDized

```
#pragma omp simd [clause [[,] clause] …]
```

- Execute iterations of following loop in SIMD chunks
  - → Region binds to the current task, so loop is not divided across threads
  - → SIMD chunk is set of iterations executed concurrently by a SIMD lanes
- Creates a new data environment
- Clauses control data environment, how loop is partitioned
  - → `safelen(`*`length`*`)` limits the number of iterations in a SIMD chunk
  - → `linear` lists variables with a linear relationship to the iteration space
  - → `aligned` specifies byte alignments of a list of variables
  - → `private`, `lastprivate`, `reduction` and `collapse` usual meanings
  - → Would `firstprivate` be useful?

What happens if a SIMDized loop includes function calls?

- Could rely on compiler to handle
  - → Compiler could in-line function to SIMDize its operations
  - → Compiler could try to generate SIMDize version of function
  - → Inefficient default would call function from each SIMD lane
- Provide `declare simd` directive to generate SIMD function

```
#pragma omp declare simd [clause [[,] clause] …]
function definition or declaration
```

  - → Invocation of generated function processes across SIMD lanes
- Clauses control data environment, how function is used
  - → `simdlen(`*`length`*`)` specifies the number of concurrent arguments
  - → `uniform` lists invariant arguments across concurrent SIMD invocations
  - → `inbranch` and `notinbranch` imply always/never invoked in conditional statement
  - → `linear`, `aligned`, and `reduction` are similar to `simd` clauses

The loop SIMD and parallel loop SIMD combine two types of parallelism

- The loop SIMD construct workshares and SIMDizes loop

```
#pragma omp for simd [clause [[,] clause] …]
```

  - → Cannot be specified separately
  - → Loop is first divided into SIMD chunks
  - → SIMD chunks are divided across implicit tasks
  - → Not guaranteed same schedule even with `static` schedule

- Parallel loop SIMD creates a parallel region with a loop SIMD region

```
#pragma omp paralel for simd [clause [[,] clause] …]
```

  - → Purely a convenience that combines separate directives
  - → Analogous to the combined parallel worksharing constructs
  - → Would a parallel SIMD construct (i.e., no worksharing) be useful?

The `declare simd` construct supports SIMD execution of library routines

- Tells compiler to generate SIMD versions of functions

```
#pragma omp simd notinbranch
float min (float a, float b) {
   return a < b ? a : b;   }

#pragma omp simd notinbranch
float distsq (float x, float y) {
   return (x − y) ∗ (x − y);   }
```

- Compile library and use functions in a SIMD loop

```
void minex (float *a, float *b, float *c, float *d) {
  #pragma omp parallel for simd
  for (i = 0; i < N; i++)
    d[i] = min (distsq(a[i], b[i]), c[i]);
}
```

→ Creates implicit tasks of parallel region

→ Divides loop into SIMD chunks

→ Schedules SIMD chunks across implicit tasks

→ Loop is fully SIMDized by using SIMD versions of functions

4.0 significantly extends initial high-level affinity support of OpenMP 3.1

- Control of nested thread team sizes (in OpenMP 3.1)

```
export OMP_NUM_THREADS=4,3,2
```

- Request binding of threads to places (in OpenMP 3.1)

```
export OMP_PROC_BIND=TRUE
```

- New extensions specify thread locations
  - →Increased choices for `OMP_PROC_BIND`
    - →Can still specify `true` or `false`
    - →Can now provide a list (possible item values: `master`, `close` or `spread`) to specify how to bind implicit tasks of `parallel` regions
  - →Added `OMP_PLACES` environment variable
    - →Can specify abstract names including `threads`, `cores` and `sockets`
    - →Can specify an explicit ordered list of places
    - →Place numbering is implementation defined

Affinity support now supports targeting thread binding to specific parallel regions

- Added a new clause to the `parallel` construct

```
proc_bind(master | close | spread)
```

  - →Overrides `OMP_PROC_BIND` environment variable
  - →Ignored if `OMP_PROC_BIND` is `false`

- New run time function to query current policy

```
omp_proc_bind_t omp_get_proc_bind(void);
```

- New policies determine relative bindings
  - →Assign threads to same place as `master`
  - →Assign threads `close` in place list to parent thread
  - →Assign threads to maximize `spread` across places

OpenMP 4.0 includes initial support for Fortran 2003

- Added to list of base language versions
- Have a list of unsupported Fortran 2003 features
  - → List initially included 24 items (some big, some small)
  - → List has been reduced to 14 items
  - → List in specification reflects approximate priority
  - → Priorities determined by importance and difficulty
- Strategy: Gradually reduce list until full support available in 5.0
  - → Removed procedure pointers, renaming operators on the `USE` statement, `ASSOCIATE` construct, `VOLATILE` attribute and structure constructors
  - → Will support Fortran 2003 object-oriented features next
    - → The biggest issue
    - → Considering concurrent reexamination of C++ support

4.0 adds taskgroup construct to simplify task synchronization

- Adds one easily shown construct

```
#pragma omp taskgroup
{
        create_a_group_of_tasks (could_create_nested_tasks);
}
```

  - →Implicit task scheduling point at end of region; current task is suspended until all child tasks generated in the region and their descendants complete execution
  - →Similar in effect to a deep `taskwait`
    - →3.1 would require more synchronization, more directives

- More significant tasking extension added concept of task dependence: the `depend` clause

We are already starting on the next version of OpenMP (4.1? 5.0?)

- Language Committee current primary focus is examples for new features in 4.0
- Concurrently beginning process for next version
  - →Process will be similar to 3.1/4.0
  - →Identifying potential topics
  - →Assessing priorities and significance
    - →Some issues may be considered minor (may lead to 4.1)
    - →Other issues are clearly more significant (must wait until 5.0)
- Next version will be well under way by SC13

We are considering several other topics for OpenMP 5.0 and beyond

- Support for memory affinity
- Refinements to accelerator support
- Transactional memory and thread level speculation
- Additional task/thread synchronization mechanisms
- Completing extension of OpenMP to Fortran 2003
- Interoperability, composability and modularity
- Incorporating tool support