A Peek Under the Cover of the New OpenMP Specification

Bronis R. de Supinski

Chair

OpenMP Language Committee

November 18-21, 2024

LLNL-PRES-868704

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

OpenMP 6.0 includes many major new features

- Officially released on November 14, 2024
  - → Reflects three years of work since release of OpenMP 5.2
  - → Includes 415 enacted issues, covering a wide range of content and complexity
- Free-agent threads significantly change execution model, implementations
- New concept for task dependences: transparent tasks
  - → Enables asynchronous `target data` (also enables other future extensions)
- User-defined induction and `induction` clause expand parallelism support
- Many significant device support improvements (e.g., `workdistribute`)
- Several additional (sequential) loop transforming directives
- Supported compound constructs are now defined based on a grammar
- Significant improvements to usability and correctness of specification

What is the effect of the following code?

```
// assume in main with initialization omitted
// assume no OpenMP directives omitted

TS = 4096
#pragma omp taskloop grainsize(TS)
for (i = 0; i < SIZE; i++) {
  A[i] = A[i] * B[i] * s;
}
```

- Pre-6.0 need `parallel masked` directive so multiple threads execute tasks

```
// assume in main with initialization omitted
// assume no OpenMP directives omitted

TS = 4096
#pragma omp parallel masked
#pragma omp taskloop grainsize(TS)
for (i = 0; i < SIZE; i++) {
  A[i] = A[i] * B[i] * s;
}
```

OpenMP execution model evolves significantly in 6.0

```
// assume in main with initialization omitted
// assume no OpenMP directives omitted

TS = 4096
#pragma omp taskloop grainsize(TS) threadset(omp_pool)
for (I = 0; I < SIZE; i++) {
  A[i] = A[i] * B[i] * s;
}
```

- OpenMP 6.0 defines OpenMP threads as members of logical thread pool
  - → Pool size can be specified by `OMP_THREAD_LIMIT` environment variable
- OpenMP 6.0 also adds the concept of free-agent threads
  - → Do not need `parallel masked` directive
  - → Instead `threadset` clause can specify that unassigned threads may execute tasks

Task dependences constrain modularity

```
// assume library must ensure fine-grain dependences are honored
int my_func(double *M, double *v) {
  int i, j, k;

  for (i = 0; i < N_ROWS; i += ROWS_PER_TASK) {
    #pragma omp task depend(inout:M[i*N_COLS])
    for (j = 0; j < ROWS_PER_TASK; j++) {
      for (k = 0; k < N_COLS; k++) {
        M[(i+j)*N_COLS + k] = M[(i+j)*N_COLS + k] * v[k]; } } }
  return 0;
}
```

- Successive calls to `my_func` with the same `M` are ordered correctly in OpenMP 5.2 and earlier if they are issued in the same task
  - → Ensures all uses of `task` construct will not deadlock
  - → Other synchronization can alleviate constraint by eliminating concurrency of tasks from different calls so this solution does not provide the desired result

Transparent tasks support richer dependence graphs

```
// assume my_func as in previous example
double M[N_ROWS*NCOLS], v[NUM_VS][N_COLS];
int i;

// code to initialize M and v omitted for brevity

for (i = 0; i < NUM_VS; i++) {
  #pragma omp task depend(inout:i) transparent(omp_impex)
  my_func(M, &v[i*N_COLS]);
}
```

- The calls to `my_func` are ordered because of the dependence shown
- These tasks are transparent importing and exporting ("`omp_impex`") tasks
  - → Dependences expressed in the calls are now imported and exported
  - → Deadlock freedom is still guaranteed

Advances in OpenMP tasking model have pervasive impact

- Other major additions to 6.0 include:
  - → Support for dependences and affinity of tasks generated by `taskloop` directives
  - → A new `taskgraph` directive that enables optimized task generation
- Task-generating constructs are fundamental to OpenMP offload model
  - → Most device constructs (e.g., `target` and `target_update` directives) already generate them
  - → Another major change: `target_data` is now a dependence sequence of three tasks
    - → Middle task is transparent by default
    - → The construct now is also a `taskgroup` region by default
    - → Can specify `no_wait` and `no_group` to rely only on dependences for ordering
- Other constructs (e.g., `parallel` and `teams`) are composed of implicit tasks
  - → While not adopted for 6.0, expect to add `transparent` clause to many of them eventually
  - → Will enable `no_wait` to be supported for `parallel` construct

Inside the implementation of the 6.0 specification

11.5 `reverse` Construct

Name: `reverse` **Category:** executable	**Association:** loop nest **Properties:** generally-composable, loop-transforming, order-concurrent-nestable, pure, simdizable, teams-nesta… Go to page 74

Clauses

`apply`

Loop Modifiers for the `apply` Clause

loop-modifier	Number of Generated Loops	Description
`reversed` *(default)*	1	the reversed loop

Semantics

The `reverse` construct has one transformation-affected loop, the outermost loop, where $0, 1, \ldots, n-2, n-1$ are the logical iteration numbers of that loop. The construct transforms that loop into a loop in which iterations occur in the order $n-1, n-2, \ldots, 1, 0$.

Cross References

- `apply` clause, see Section 11.1

Lawrence Livermore
National Laboratory