SC11 OpenMP Language Committee Report

November 15, 2011

Bronis R. de Supinski

OpenMP Language Committee Chair
Center for Applied Scientific Computing

Science & Technology Principal Directorate - Computation Directorate

OpenMP is a vibrant growing organization

- ARB membership at an all-time high
  - 13 permanent members (implementers)
    - Most recent addition is Nvidia
  - 8 auxilliary members (user institutions)
    - Most recent addition is TACC
- Actively pursuing new specifications
  - OpenMP 3.1 released in July 2011
  - Significant progress already on OpenMP 4.0
  - Planning always extends beyond the next specification
  - Feedback from non-members always welcome
- International Workshop on OpenMP (IWOMP) going strong

OpenMP 3.1 specification recently finished and work on the following one is already begun

- OpenMP 3.1
  - Refine and extend existing specification
  - Do not break existing code
  - Minimal implementation burden beyond 3.0
  - Enacted 87 tickets total
- OpenMP 4.0 (?)
  - Draft planned for SC12 (adopting time-based releases)
  - Address several major open issues for OpenMP
  - Do not break existing code unnecessarily
  - Already have passed 4 tickets
    - Added UDRs, atomic swap
    - Addressed some small questions on atomics

Despite incremental nature, we added several important items for OpenMP 3.1

- Extend atomics to support capture and write functionality
- Add `min` and `max` reduction operators in C/C++
- Extensions to OpenMP tasking model
  - Explicit task scheduling points (`taskyield` construct)
  - Ability to save data environment overhead
    - `final` and `mergeable` clauses
    - `omp_in_final` runtime library routine
- Initial support for thread binding
- Now allow `intent(in)` and const-qualified types in `firstprivate` clause
- Many clarifications, including improvements to examples

The final clause combines with new tasking concepts to reduce tasking overhead

- Recognizing an existing concept and creating three new ones
  - An **undeferred task** is a task for which execution is not deferred with respect to its generating task region

```
#pragma omp task if(0)
```

  - An **included task** is an undeferred task that is sequentially included in generating task region (executed immediately)
  - A **merged task** has the same data environment, including ICVs, as its generating task region
  - A **final task** forces its descendant tasks to be included
- New extensions to the task construct
  - The `mergeable` clause suggests the task may be merged
  - The `final(expr)` clause if true results in a final task

Additional kind of atomic operations addresses an obvious deficiency

- Currently cannot capture a value atomically

```
int schedule (int upper) {
   static int iter = 0; int ret;
   ret = iter;
   #pragma omp atomic
      iter++;
   if (ret <= upper) { return ret; }
   else { return -1; }  //no more iters
}
```

- Atomic capture provides the needed functionality

```
int schedule (int upper) {
   static int iter = 0; int ret;
   #pragma omp atomic capture
      ret = iter++;     // atomic capture
   if (ret <= upper) { return ret; }
   else { return -1; }  // no more iters
}
```

Adding initial high-level affinity support to the OpenMP 3.1 specification, more planned for 4.0

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

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

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

  ```
  export OMP_PROC_BIND=TRUE
  ```

  Plan additional choices (compact, spread, a list) for 4.0
- Restrict the processor set for program execution

  ```
  export OMP_PLACES 0,1,2,3,8,10,12,14
  ```

  Can also specify lists, groupings
- Planning new runtime library routines to observe and to control bindings (get_place, get/set_place_partition)
- Considering environment variables to:
  - Control thread placement within a processor set
  - Control initial placement of shared data
  - Adapt data placement at runtime

User Defined Reductions (UDRs) are a major addition already adopted for OpenMP 4.0

- Use `declare reduction` directive to define new operators
- New operators used in reduction clause like predefined ops

```
#pragma omp declare reduction (reduction-identifier :
typename-list : combiner) [identity(identity-expr)]
```

- `reduction-identifier` gives a name to the operator
  - Can be overloaded for different types
  - Can be redefined in inner scopes
- `typename-list` is a list of types to which it applies
- `combiner` expression specifies how to combine values
- `identity` can specify the identity value of the operator
  - Can be an expression or a brace initializer

A simple UDR example

- Declare the reduction operator

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

- Use the reduction operator in a reduction clause

```
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);
}
```

- Private copies created for a reduction are initialized to the identity that was specified for the operator and type
  - Default identity defined if no identity clause present
- Compiler uses combiner to combine private copies
  - `omp_out` refers to private copy that holds combined value
  - `omp_in` refers to the other private copy

We are actively discussing several major topics for OpenMP 4.0 and beyond

- Initial work to support Fortran 2003
- Development of an error model
  - The `done` directive
  - Callbacks for integrated error handling
- Interoperability and composability
  - Interactions between thread models
  - Interfaces to support interactions with distributed models
- Refinements to the OpenMP tasking model
  - Specifying task dependencies (think data flow)
  - Task reductions, task-only threads, `omp while`
- Affinity (previous slide)
- Sequentially consistent atomic operations
- How to specify subarrays in C

We are considering these and several other topics for OpenMP 4.0 and beyond

- Other topics being considered for OpenMP 4.0
  - Transactional memory and thread level speculation
  - Additional task/thread synchronization mechanisms
  - Extending OpenMP to Fortran 2003
  - Extending OpenMP to additional languages
  - Incorporating tools support
  - Other miscellaneous extensions
- How can you help shape the future of OpenMP?
  - Attend IWOMP, become a cOMPunity member
  - Lobby your institution to join the OpenMP ARB
  - Contact me and beg ;-)