Is OpenMP for Users?

Bill Gropp
Argonne National Laboratory
www.mcs.anl.gov/~gropp

Quiz

- Is the following a correct program?
- #include <stdio.h>
  #include <omp.h>
  void skip(int i) {/*…*/}
  void work(int i){/*…*/}
  int main() {
  omp_lock_t lck:
  int id;
  omp_init_lock(&lck);
  #pragma omp parallel shared(lck) private(id)
  {
  id = omp_get_thread_num();
  omp_set_lock(&lck);
  printf(“My thread id I %d.\n”, id);
  omp_unset_lock(&lck);
  while(! omp_test_lock(&lck)) { skip(id); }
  work(id);
  omp_unset_lock(&lck);
  }
  omp_destroy_lock(&lck);
  return 1;
  }

Quiz Answer

- No. According to A.17, p 143-144, it must be
- ```
  #include <stdio.h>
  #include <omp.h>
  void skip(int i) {/*…*/}
  void work(int i){/*…*/}
  int main() {
      omp_lock_t lck:
      int id;
      omp_init_lock(&lck);
      #pragma omp parallel shared(lck) private(id)
      {
          id = omp_get_thread_num();
          omp_set_lock(&lck);
          printf( “My thread id I %d.\n”, id );
          omp_unset_lock(&lck);
          while(! omp_test_lock(&lck)) { skip(id); }
          #pragma omp flush
          work(id);
          #pragma omp flush
          omp_unset_lock(&lck);
      }
      omp_destroy_lock(&lck);
      return 1;
  }
  ```
  ```

Problems with Support for Multilingual Programming

- Three routines that set values (such as the number of threads to use) have the same name but different calling sequences in C and Fortran
  - Set_num_threads, set_dynamic, set_nested
- If sizeof(omp_lock_t) != 4, then all 10 omp_xxx_lock routines can fail
- If Fortran .true. and .false. don't correspond to C, then 3 more routines with logical return values can fail
- This affects libraries: E.g., a user Fortran program that calls a library written in C that uses OpenMP and that is linked in the usual and expected way will fail
- Only affects vendors whose C and Fortran compilers generate the same loader name for the same "user" name
  - This means IBM. I'm surprised IBM has not raised this issue.
- Possible fixes:
  - Add new routines for C and Fortran
    - Suggestion: Use mixed case names for C/C++, e.g., OMP_Set_num_threads(int n) and omp_set_num_threads(Fint *n)
    - Deprecate routines with conflicting bindings

Risks with “stub” version

- How does an application know whether it got the stub version or not?
  - One vendor made this mistake with their thread library. Stubs in libc meant programs linked and ran but did not have any thread capability
    - Even worse, same routines provided a mutex between processes, meaning that an application could use fork to create a new process and expect the mutex to provide a mutex
      - (yes, this vendor should be shot)
- There should be a runtime call to discover level of support

Dangerous Language Features

- "The language should make it hard to write incorrect programs"
- Many OpenMP defaults put the burden on the programmer rather than the compiler
  - Pragmatic reason: Make sure that OpenMP code will run fast with minimum intervention
- We already saw flush
  - For this reason, most thread libraries include flush as a property of the lock/unlock routines
  - Better to treat this as an optimization – if the user has evidence that performance requires fine-grain control, then provide a way to do that.
- Another example: lastprivate
  - Without lastprivate, OpenMP pragmas can change the behavior of the program
    - Violates principle of least surprise
  - Compilers are usually good at detecting dead variables, so making lastprivate the default should not affect performance
  - If the semantics of "last value of loop variable used by some thread" is desired, then there should be a pragma for that
- Many others

OpenMP – Assembly Language for Thread Parallelism?

- Not a bad thing
  - Provides a portable assembly language
    - But must fix name conflicts first
- But not the final solution
  - Still too easy to write incorrect code
  - Analysis tools that identify potential problems are not an adequate solution
    - Not ubiquitous
    - Features like atomic require whole-program analysis