High Performance Computing: Tools and Applications

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Lecture 4

OpenMP directives

So far we have seen

```
#pragma omp parallel
```

```
#pragma omp for
```

Implied barrier at end of `for` construct

```
#pragma omp parallel
{
  #pragma omp for
  for (i=0; i<n; i++)
  {
    a[i] = i;
  }

  // implied barrier

  // any thread sees all components of "a"
  // as updated
}
```

There is also an implied barrier at the end of `sections` and `single` constructs.

`firstprivate` and `lastprivate` clauses in `for` directive

- With the `private` clause, private variables are undefined at the beginning of the loop, and values within the loop are not visible after the loop
- `firstprivate` clause instead initializes the private variables
- `lastprivate` clause copies value of last iteration to the variable after the loop

Example: both firstprivate and lastprivate

```
#include <stdio.h>
#include <omp.h>

int main (int argc, char *argv[])
{
  int i, a = 1000;

  #pragma omp parallel for firstprivate(a) lastprivate(a)
  for (i=0; i<10; i++)
  {
    a = a + i;
  }
  printf("value of a: %d\n", a);

  return 0;
}
```

`sections` directive

- Used when different threads must execute different code
- Must still create threads with `parallel` directive
- In general, *p* threads created and *n* sections

```
#pragma omp parallel
#pragma omp sections
{
  #pragma omp section
  printf("First thread %d\n", omp_get_thread_num());

  #pragma omp section
  printf("Second thread thread %d\n", omp_get_thread_num());
}
```

What happens if $p < n$?
What happens if $p > n$?

Notes on `sections` directive

- at most *n* threads run in parallel
- can also use `firstprivate` and `lastprivate` with obvious definitions of *first* and *last*
- can combine `parallel` and `sections` directives `#pragma omp parallel sections` like `parallel for`

What is wrong with this code?

```
#pragma omp parallel
{
  a = 255;

  #pragma omp for
  for (i=0; i<n; i++)
    b[i] = a;
}
```

What is wrong with this code?

- Depending on hardware, write to `a` may not be atomic, and thread 0 may read `a` when thread 1 has only partially written to it.
- Possible solution is to use a barrier after the write.
- Multiple threads writing to `a` is also unnecesssary.
- Solution here is to use `single` directive.

single directive

- Used when code should only be executed by a single thread
- Can be executed by any thread (related master directive)

```
#pragma omp parallel
{
  #pragma single
  a = 255;

  #pragma omp for
  for (i=0; i<n; i++)
    b[i] = a;
}
```

Implied barrier at end of structured block (of single).

`atomic` directive

- thread-safe update of shared variables
- generally requires the compiler to use atomic instructions in the instruction set
- applies to single statements only (not blocks) with specific forms of updating a memory location

  ```
  #pragma omp atomic
  i = i + 1;
  ```

atomic directive: example allowed form

```
x = x binop expr;
```

where x is an l-value with scalar type, expr does not access the same storage as x, and binop is a binary operation, e.g., +.

For more details:
https://software.intel.com/en-us/node/524509

More OpenMP directives

```
#pragma omp sections

#pragma omp single

#pragma omp master
// no implied barrier on exit

#pragma omp barrier

#pragma omp ordered
// used inside parallel for loop

#pragma omp critical [name]

#pragma omp atomic
// only for statements of specific form
```

Some OpenMP clauses

- `num_threads` sets the number of threads in `parallel` directive
- `if` controls the `parallel` directive depending on a condition
- `nowait` removes the barrier at the end of `omp for` and other constructs
- `ordered` needed to indicate that an `ordered` directive is within an `omp for` loop

Example `if` clause

Only spawn threads if the “problem” is large enough:

```
#pragma omp parallel if (n > 1000)
```

More realistic particle simulations

- Particles have radius a
- Cubical simulation box has width L and *periodic* boundaries
- Particles interact with each other
  - repulsive force when they overlap
  - other forces, e.g., when particles are charged

Repulsive force when particles overlap

If distance s between particles i and j is less than $2a$, then force on particle i due to particle j is

$$f_{ij} = k_r(2a - s) \cdot \hat{n}$$

where $k_r = 100$ is the repulsion force constant and $\hat{n}$ is the unit vector from j to i.

Updating the particle positions, including repulsive force

$$x(i) = x(i) + M \cdot f(i)\Delta t + \sqrt{2\Delta t} \cdot y(i)$$

where $f(i)$ is the total force on particle i and $M = 1$ is a constant.

Code to compute forces from the positions

```
for (i=0; i<np; i++) {
    for (j=i+1; j<np; j++) {
        ri = &pos[3*i];
        rj = &pos[3*j];
        dx = remainder(ri[0]-rj[0], L);
        dy = remainder(ri[1]-rj[1], L);
        dz = remainder(ri[2]-rj[2], L);

        s2 = dx*dx + dy*dy + dz*dz;
        if (s2 < 4.*a*a) {
            s = sqrt(s2);
            f = krepul*(2.-s);

            forces[3*i+0] += f*dx/s;
            forces[3*i+1] += f*dy/s;
            forces[3*i+2] += f*dz/s;
            forces[3*j+0] -= f*dx/s;
            forces[3*j+1] -= f*dy/s;
            forces[3*j+2] -= f*dz/s;
        }
    }
}
```

How to parallelize this code using OpenMP?

Iterations on `i` are not independent, since different iterations can write to the same location in `forces`.

How to parallelize this code using OpenMP?

Iterations on `i` are not independent, since different iterations can write to the same location in `forces`.

- Update forces in a critical section

How to parallelize this code using OpenMP?

Iterations on `i` are not independent, since different iterations can write to the same location in `forces`.

- Update forces in a critical section
- Use atomic operations to update components of the force array

How to parallelize this code using OpenMP?

Iterations on `i` are not independent, since different iterations can write to the same location in `forces`.

- Update forces in a critical section
- Use atomic operations to update components of the force array
- Rewrite the outer loop so that iterations are independent (only update the forces for particle *i*, not *j*)

How to parallelize this code using OpenMP?

Iterations on `i` are not independent, since different iterations can write to the same location in `forces`.

- Update forces in a critical section
- Use atomic operations to update components of the force array
- Rewrite the outer loop so that iterations are independent (only update the forces for particle *i*, not *j*)
- Each thread sums the forces locally, and then a sequential reduction computes the total force (requires storage local to each thread)

How to parallelize this code using OpenMP?

Iterations on `i` are not independent, since different iterations can write to the same location in `forces`.

- Update forces in a critical section
- Use atomic operations to update components of the force array
- Rewrite the outer loop so that iterations are independent (only update the forces for particle *i*, not *j*)
- Each thread sums the forces locally, and then a sequential reduction computes the total force (requires storage local to each thread)
- Tabulate the overlaps in parallel, but sum the forces sequentially (could work if few overlaps; tabulation needs shared data structure)

Exercise 4

- Update your code from Exercise 2 to include repulsive interactions between particles when they overlap. Parallelize using OpenMP and run on `jinx` or `deepthought`.
- Compare the performance between using critical sections, atomic operations, and independent iterations (which do twice the number of distance computations).
- Submit your results in the `ex04` directory (do not forget to update your fork), including
  - `ex04.c` or `ex04.cpp` source file and `makefile`
  - `ex04.pdf` report with performance comparison
- *Due 10 pm, Monday, Sept. 5*