Shared Memory Programming with OpenMP

Synchronisation

archer2

epcc

Why is it required?

Recall:

- Need to synchronise actions on shared variables.
- Need to ensure correct ordering of reads and writes.
- Need to protect updates to shared variables (not atomic by default)

barrier directive

- No thread can proceed past a barrier until all the other threads have arrived.
- Remember that there is an *implicit* barrier at the end of **`for`** and **`single`** directives.

- Syntax:

C/C++: **`#pragma omp barrier`**

- Either all threads or none must encounter the barrier: otherwise DEADLOCK!!

barrier directive (cont)

Example:

```
#pragma omp parallel private(myid,neighb) shared(a,b,c)
{
   myid = omp_get_thread_num();
   neighb = myid - 1;
   if (myid.eq.0) neighb = omp_get_num_threads()-1;
   ...
   a[myid] *= 3.5;
#pragma omp barrier
   b[myid] = a[neighb] + c;
   ...
}
```

- Barrier required to force synchronisation on **a**

Critical sections

- A critical section is a block of code which can be executed by only one thread at a time.
- Can be used to protect updates to shared variables.
- Mutual exclusion is enforced between all critical sections in the code

Syntax:

C/C++: **`#pragma omp critical`**

structured block

critical directive (cont)

Example: appending to a shared list

```
#pragma omp parallel for shared(list, N) private(newitem_p)
for (int i=0; i<N; i++) {
   newitem_p = createitem(i);
#pragma omp critical
   {
     append(&list,newitem_p);
   }
}
```

critical directive (cont)

Example: pushing and popping a task stack

```
#pragma omp parallel shared(stack) private(p_next,p_new,done)
{
while (!done) {
#pragma omp critical
  {
    p_next = pop(&stack);
  }
   p_new = process(p_next);
#pragma omp critical
  {
    if (p_new != NULL) push(p_new,&stack);
    done = isempty(&stack);
  }
  }
}
```

atomic directive

- Used to protect a single update to a shared scalar variable (or array element) of basic type.
- Applies only to a single statement.

Syntax:

C/C++:

```
#pragma omp atomic
            statement
```

where *statement* must have one of the forms:

x binop **=** *expr*, *x***++**, **++***x*, *x***--**, or **--***x*

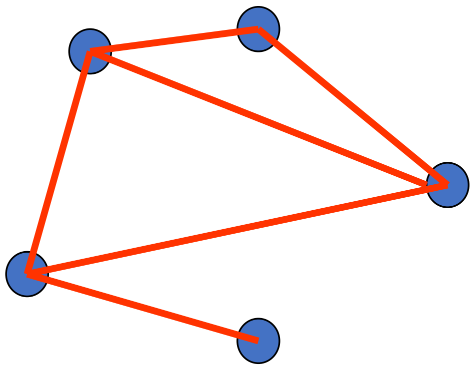
and *binop* is one of **+**, *****, **-**, **/**, **&**, **^**, **<<**, or **>>**

- Note that the evaluation of *expr* is not atomic.
- Should be more efficient than using **`critical`** directives, e.g. if different array elements can be protected separately.
- No interaction with **`critical`** directives

atomic directive (cont)

Example (compute degree of each vertex in a graph):

```
#pragma omp parallel for
      for (j=0; j<nedges; j++){
#pragma omp atomic
         degree[edge[j].vertex1]++;
#pragma omp atomic
         degree[edge[j].vertex2]++;
      }
```



Lock routines

- Sometimes we require more flexibility than is provided by `critical` or `atomic` directives.
- A lock is a special variable that may be *set* by a thread. No other thread may *set* the lock until the thread which set the lock has *unset* it.
- Setting a lock can either be blocking or non-blocking.
- A lock must be initialised before it is used, and may be destroyed when it is not longer required.
- Lock variables should not be used for any other purpose.
- OpenMP locks are equivalent to mutexes in other APIs.
- A critical construct is equivalent to setting a lock on entry to the block of code and unsetting it on exit.

Lock routines - syntax

C/C++:

```
#include <omp.h>
  void omp_init_lock(omp_lock_t *lock);
  void omp_set_lock(omp_lock_t *lock);
  int omp_test_lock(omp_lock_t *lock);
  void omp_unset_lock(omp_lock_t *lock);
  void omp_destroy_lock(omp_lock_t *lock);
```

Lock example

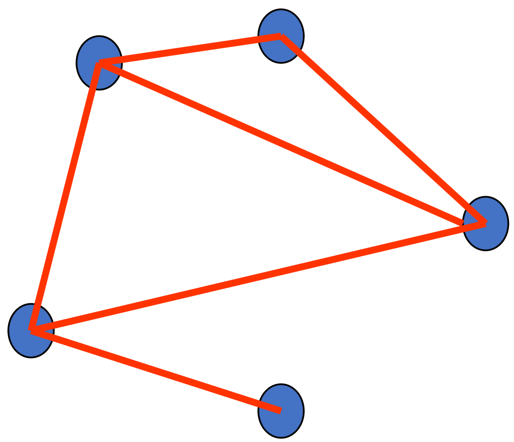
Example (compute degree of each vertex in a graph):

```
omp_lock_t lockvar[nvertices];

for (i=0; i<nvertices; i++){
  omp_init_lock(&lockvar[i]);
}

#pragma omp parallel for
      for (j=0; j<nedges; j++){
         omp_set_lock(&lockvar[edge[j].vertex1]);
           degree[edge[j].vertex1]++;
         omp_unset_lock(&lockvar[edge[j].vertex1]);

         omp_set_lock(&lockvar[edge[j].vertex2]);
           degree[edge[j].vertex2]++;
         omp_unset_lock(&lockvar[edge[j].vertex2]);
       }
```



Exercise:

- Redo the Mandelbrot example using critical, atomics or locks to avoid the race condition on **`numoutside`** instead of a reduction clause.

- How does the performance differ?

Reusing this material

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

https://creativecommons.org/licenses/by-nc-sa/4.0/

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Acknowledge EPCC as follows: “© EPCC, The University of Edinburgh, www.epcc.ed.ac.uk”

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.