OpenMP (following Matloff, chapter 4)

Shared memory programming

- pthreads ... intro in CSCI 347, will not redo here.
- OpenMP ... the "de facto" standard in SM programming
  - No multiple machines, need MPI with OpenMP for that!

OpenMP basic concepts ...

- Included in C compilers
  - Works out of the box on our linux systems for gcc
  - Works for clang, (LLVM), but not on our systems right now
  - constructs are done via #pragma ...
    - #pragma must be first non-blank, can be indented
  - Simple version (zach.c)

```
#include <omp.h>
#include <stdio.h>
int main()
{
   #pragma omp parallel for num_threads(4)
   for (int i = 1; i <= 10; i++) {
       int tid = omp_get_thread_num();
       printf("The thread %d  executes i = %d\n", tid, i);
   }
   return 0;
}
```

OpemMP (page 2)

Matloff example using Dijkstra’s algorithm for shortest paths from vertex 0 to all other vertices in a N-vertex undirected graph

- Algorithm

```
Done = {0}              # vertices checked so far
NewDone = None          # currently checked vertex
NonDone = {1,2,...,N-1} # vertices not checked yet
for J = 0 to N-1 Dist[J] = G(0,J) # initialize shortest-path lengths


for Step = 1 to N-1
  find J such that Dist[J] is min among all J in NonDone
  transfer J from NonDone to Done
  NewDone = J
  for K = 1 to N-1
    if K is in NonDone
      # check if there is a shorter path from 0 to K through NewDone
      # than our best so far
      Dist[K] = min(Dist[K],Dist[NewDone]+G[NewDone,K])
```

- Parallel solution: parallize "find J" and "for K"

OpenMP (Page 3)

dijkstra.c (with modifications from Matloff’s book)

- double omp_get_wtime();
- Main function is run by the "master thread"
- #pragma omp parallel
  - sets up threads including master for execution
  - all threads run the code inside block
  - me = opm_get_thread_num() -- gets thread number
  - #pragma omp parallel before var decls -> thread local variables
  - #pragma omp parallel after var decls -> global variables
  - #pragma omp parallel private(x,y) makes x and y local
  - Threads communicate via global variables
  - #pragma omp single \n { block } -- executed by one thread
    - nth and chunk are global
  - #pragma omp barrier -- standard barrier
  - #pragma omp critical -- critical section, one thread at a time
- #pragma omp parallel vs #pragma omp parallel for (zach1.c, zach2.c)

OpenMp (Page4)

More on the #pargma omp parallel for ...

Modification to dijkstra: Replace call to "findmymin()" with

```
mymd = largeint;
#pragma omp for nowait
for (i = 1; i < nv; i++) {
  if (notdone[i] && mind[i] < mymd) {
    mymd = ohd[i];
    mymv = i;
  }
}
```

- Body of for is done independently by threads
- Order is not maintained, unpredictable order
- "nowait" does not put a barrier at the end of the for loop

Nested loops

```
int i, j;
#pragma omp parallel for collapse(2)
for (i = 0; i < 5; i++)
  for (j = 0; j < 5; j++)
    printf ("(%d,%d)\n", i, j);
```

OpenMp (Page 5)

- default version of for: no specific thread does any specific loop
- Schedule can cause threads to take a "chunk" of the loops
  - #pragma omp for schedule(static,<chunk>)
    - chunk is the number of loop elements
  - omp_set_schedule(omp_sched_static, <chunk>) -- runtime version
  - export OMP_SCHEDULE="static,<chunk>"   (bash version)
- chunk sizes: small -> lots of parallelism, may have high overhead
  - large -> lower overhead, some threads may be idle (use guided ... see below)
- kinds of schedules
  - static: iterations in chunks, assigned statically to threads
    - threads run round robin, default is iterations/threads
  - dynamic: iterations in chunks, chunks assigned dynamicallyN
    - thread finishes, gets more work, default chunk is 1
  - guided: similar to dynamic, but chunk size decreasing as work decreases

OMP Reduction

```
int z;
...
#pragma omp for reduction(+:z)
for (i = 0; i < n; i++) z+=x[i];
```

- Independent copies of z for each thread
- When loops are done, z’s from threads summed in an atomic manner
- + => only z summed, initial values of local z is 0
- * => product, initial values would be 1.

More painful version

```
int z, myz=0;
...
#pragma omp for private(myz)
for (i = 0; i < n; i++) myz += x[i];
#prama omp critical
{ z += myz; }
```

- Eligable operators: (op, initial value)
  - (+,0), (-,0), (*,1), (&,all 1s), (|,all 0s), (^,0), (&&,1), (||,0)
  - Read the example program "Mandelbrot Set" (pg 94-97)

OMP Tasks

- Tasks to execute a block of code "at some time"
- A task gets one thread
- Can do a barrier to syncronize the tasks.
- Quicksort example ... ompqs.c
  - #pragma omp single nowait
    - required?
  - Must be a taskwait at end of qs function

Other things

- Atomic: May be faster than "critical" if just dealing with one var.

  ```
  #pragma omp atomic
  x += y;
  ```
  - expecting x is global and y is thread local
- Flush: making sure a "global variable" gets sent to cache ...
  - #pragma omp flush (x)
  - other flush points
    - barrier
    - entry/exit to/from critical, ordered, parallel
    - exit from parallel for, parallel sections, single

- And much more ... not in Matloff!