OpenMP

25 January 2007
CMPT370
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review last time

- Memory models:
  - Shared (SMP)
  - Distributed (cluster)
  - Hybrid
- Programming models:
  - Threads (PThreads, OpenMP)
  - Message passing (MPI)
  - Data-parallel (HPF)
  - Hybrids
- Automatic vs. manual parallelization

OpenMP

- An industry standard API for shared-memory parallelism for high-performance computing
- Programmers interface to OpenMP via:
  - Compiler directives (#pragma omp parallel)
  - Library subroutines (omp_get_num_threads())
  - Environment variables (OMP_NUM_THREADS)
- Fork/join threading model:
  - Fork at start of a parallel construct
  - Join (implied barrier) at end of construct

OpenMP parallel constructs

- #pragma omp parallel
  - Code duplicated to all threads
- #pragma omp for
  - Distribute iterations of a for loop
- #pragma omp sections
  - #pragma omp section
  - #pragma omp section
  - Each section has different code, one thread per section

Compiling with OpenMP

- OpenMP has newly been added to gcc/g++ 4.1.0
  - Also in MSVC 2005 (not .NET)
- Include: #include <omp.h>
- Compile with flag: -fopenmp
- Link with: -lgomp (GNU OpenMP)
- See sample Makefiles on carmel under /home/seanho/cmpt370/

Shared vs. private variables

- By default, most variables in OpenMP are shared by all threads
  - Except variables declared within a block inside a parallel region
  - Or can declare a variable to be private to each thread
  - Also a reduce operation to combine partial results from each thread (more later)

- helloworld.c example on carmel:
  - /home/seanho/cmpt370/helloworld/

OpenMP synchronization pragmas

- #pragma omp parallel
  - Next block (use {}) is a parallel section
- #pragma omp critical
  - Next block should be one-thread-at-a-time
- #pragma omp single
  - next block should be done by only one of the threads
- #pragma omp barrier
  - Wait here for all threads: synchronization point
- Others: master, ordered, atomic, flush

How many threads?

- Can have fewer threads than physical processors
  - Wasting the other processors
- Or more threads than processors
  - Threads will queue, waiting for a free CPU
- By default, OpenMP will use as many threads as there are processors (8 on carmel)
- Change at runtime with environment variable:
  - OMP_NUM_THREADS=1 ./helloworld
- Can also change inside program with a library subroutine

OpenMP library routines

- int omp_get_num_threads()
- int omp_set_num_threads()
  - How many threads are currently in use
- int omp_get_thread_num()
  - Which thread id I am
- double omp_get_wtime()
  - Get wall-clock time in number of ticks
- double omp_get_wtick()
  - Get length of a tick in seconds
- A few others (not many) for locking

Scheduling a for loop

- How is work distributed amongst threads?
  - schedule(static) *(optional chunk-size)*
    - Divide iterations into chunks, distribute evenly amongst threads
  - schedule(dynamic) *(optional chunk-size)*
    - Queue of chunks: threads take next available chunk
  - schedule(guided) *(optional chunk-size)*
    - Like dynamic, but chunk size is exponentially reduced
  - schedule(runtime)
    - Follow OMP_SCHEDULE environment variable

Reduce

- The reduction(op:var) flag to an OpenMP pragma specifies a var that each thread contributes towards; the results are combined using the op
  - e.g.: finding sum of a vector

```
#pragma omp for reduction(+:sum)
for (i = 0; i < num_iters; i++)
    sum += vector[i];
```

  - Ops: sum(+), product(*), and(&&), or(||).

- See pi-leibniz.c example

Lab2: Your own OpenMP program

- Ideas:
  - Numerical integration (like pi-leibniz.c)
  - Generating fractals: see mandelbrot/ example
  - Dictionary/ brute force encryption cracking?
  - Prime number generation?

TODO

- Lab2 due Tue 6Feb
  - Design + implement your own OpenMP program
  - Lab write-up