Introducing OpenMP in a Data Structures (CS2) or Systems Class

Alan Sussman
Dept. of Computer Science
University of Maryland

SIGCSE 2019 Workshop on Modernizing Early CS Courses with Parallel and Distributed Computing
Goals

• Introduce basic ideas of OpenMP shared memory parallelism
  – At a level suitable for teaching in an intro programming or systems class
  – Mainly targeting loop level parallelism

• Provide examples of using OpenMP for some simple problems
OpenMP

• Support Parallelism for SMPs
  – provide a simple portable model
  – allows both shared and private data
  – provides parallel do loops

• Includes
  – automatic support for fork/join parallelism
  – reduction variables
  – lots of other things we won’t talk about today
OpenMP

- Characteristics
  - Both local & shared memory (depending on directives)
  - Parallelism: directives for parallel loops, functions
  - Compilers convert programs into multi-threaded (i.e. pthreads)
  - Not available on clusters

- Example
  ```c
  #pragma omp parallel for private(i)
  for (i=0; i<NUPDATE; i++) {
    int ran = random();
    table[ ran & (TABSIZEx1) ] ^= stable[ ran >> (64-LSTSIZE) ];
  }
  ```

Alan Sussman, University of Maryland
More on OpenMP

• Characteristics
  – Not a full parallel language, but a language extension
  – A set of standard compiler directives and library routines
  – Used to create parallel Fortran, C/C++, Java programs
  – Usually used to parallelize loops
  – Standardizes last >20 years of SMP practice

• Implementation
  – C/C++ compiler directives using
    #pragma omp <directive>
    • Somewhat different syntax for Java or Fortran
  – Parallelism can be specified for regions & loops
  – Data can be
    • Private – each thread has local copy
    • Shared – single copy for all threads

Alan Sussman, University of Maryland
OpenMP – Programming Model

- Fork-join parallelism (restricted form of MIMD)
  - Normally single thread of control (master)
  - Worker threads spawned when parallel region encountered
  - Barrier synchronization required at end of parallel region

Master Thread

Parallel Regions

Alan Sussman, University of Maryland
OpenMP – Example Parallel Region

• Task level parallelism – \#pragma omp parallel { … }

```c
double a[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
    int id = omp_thread_num();
    foo(id, a);
}
printf(“all done \n”);
```

OpenMP compiler

```c
double a[1000];
omp_set_num_threads(4);
#pragma omp parallel
foo(3, a);
foo(2, a);
foo(1, a);
foo(0, a);
printf(“all done \n”);
```
OpenMP – Example Parallel Loop

- **Loop level parallelism** – `#pragma omp parallel for`
  - Loop iterations are assigned to threads, invoked as functions

```c
#pragma omp parallel
{
    int id, i, nthreads, start, end;
    id = omp_get_thread_num();
    nthreads = omp_get_num_threads();
    start = id * N / nthreads; // assigning
    end = (id+1) * N / nthreads; // work
    for (i=start; i<end; i++) {
        foo(i);
    }
}
```

Alan Sussman, University of Maryland
```c
int main() {
    int n, i;
    double w, x, sum, pi;
    printf("Enter number of intervals: \n");
    scanf("%d", &n);
    /* calculate the interval size */
    w = 1.0;
    sum = 0.0;
    #pragma omp parallel for private(x), reduction(+: sum)
    for (i = 1; i <= n; i++) {
        x = w * (i - 0.5);
        sum = sum + f(x);
    }
    pi = w * sum;
    printf ("computed pi = %f\n", pi);
}
/* function to integrate */
double f(double a) {
    return (2.0 / (1.0 + a*a));
}
```

Alan Sussman, University of Maryland
Execution Context

• Every thread has its own execution context
• Execution context: address space containing all of the variables a thread may access

• Contents of execution context:
  – static variables
  – dynamically allocated data structures in the heap
  – variables on the run-time stack
  – additional run-time stack for functions invoked by the thread
Shared and Private Variables

• Shared variable: has same address in execution context of every thread
• Private variable: has different address in execution context of every thread
• A thread cannot access the private variables of another thread
int main (int argc, char *argv[]) {
    int b[3];
    char *cptr;
    int i;

    cptr = malloc(1);
    #pragma omp parallel for
    for (i = 0; i < 3; i++)
        b[i] = i;
Race Condition

• Consider this C program segment to compute $\pi$ using the rectangle rule (should look familiar):

```c
double area, pi, x;
int i, n;
...
area = 0.0;
for (i = 0; i < n; i++) {
    x = (i+0.5)/n;
    area += 4.0/(1.0 + x*x);
}
pi = area / n;
```
• If we simply parallelize the loop...

double area, pi, x;
int i, n;
...
area = 0.0;
#pragma omp parallel for private(x)
for (i = 0; i < n; i++) {
    x = (i+0.5)/n;
    area += 4.0/(1.0 + x*x);
}
pi = area / n;
Race Condition (cont.)

- ... we set up a race condition in which one process may “race ahead” of another and not see its change to shared variable 

\[ \text{area} \]

\[ \text{area} += \frac{4.0}{1.0 + x^2} \]

Thread A: 15.432
Thread B: 15.230

Answer should be 18.995
Reductions

- Reductions are so common that OpenMP provides support for them.
- May add reduction clause to `parallel for` pragma.
- Specify reduction operation and reduction variable.
- OpenMP takes care of storing partial results in private variables and combining partial results after the loop.
reduction Clause

• The reduction clause has this syntax:
  reduction (<op> : <variable>)

• Operators
  - +    Sum
  - *    Product
  - &    Bitwise and
  - |    Bitwise or
  - ^    Bitwise exclusive or
  - &&   Logical and
  - ||   Logical or
\[\pi\]-finding Code with Reduction Clause

double area, pi, x;
int i, n;
...
area = 0.0;
#pragma omp parallel for \\
private(x) reduction(+:area)
for (i = 0; i < n; i++) {
    x = (i + 0.5)/n;
    area += 4.0/(1.0 + x*x);
}
pi = area / n;
Logistics

• OpenMP is supported by modern C, C++, Fortran compilers
  – gcc has supported it since version 4.2
  – Use the `–fopenmp` argument to gcc to enable processing OpenMP directives and link the library
  – Set the environment variable `OMP_NUM_THREADS` to set the initial number of threads in parallel sections

• For Java support, try pyjama (at U. Auckland) or omp4j (at www.omp4j.org)
Introducing OpenMP in a CS2 or Systems Class

Alan Sussman
Dept. of Computer Science
University of Maryland

SIGCSE 2019 Workshop on Modernizing Early CS Courses with Parallel and Distributed Computing