Our Progress

- The progress so far
  - We saw some data structures: arrays, stacks, queues, linked lists, and hash tables.
  - Saw applications of the above data structures too.
  - Understand parallelism in computing to a small extent.
- This topic:
  - Study a framework for writing simple parallel programs.
  - Write simple parallel programs.

The OpenMP Programming Framework

- OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs.
- The MP in OpenMP stands for Multi Processing.
- OpenMP is designed for Fortran, C and C++ to support the language that the underlying compiler supports.

A Quick Example

```
#include <stdio.h>
#include <omp.h>
/* Main Program */
main() {
int ThreadID, NoofThreads;
Omp_set_num_threads(3);
/* OpenMP Parallel Directive */
#pragma omp parallel private(ThreadID)
{
    ThreadID = omp_get_thread_num();
    printf("Hello World is being printed by the thread id %d\n", ThreadID);
    /* Master Thread Has Its ThreadID 0 */
    if (ThreadID == 0) {
        printf("Master prints Numof threads");
        NoofThreads = omp_get_num_threads();
        printf("Total number of threads are %d", NoofThreads);
    }
}
}
```

OpenMP Basics

- How to choose the number of threads to execute?
- Typically, depends on the machine used.
  - Plus other factors such as the nature of the program.

- How to set the number of threads?
- Several ways.
  1. Via environment variables.
     - *setenv OMP_NUM_THREADS 3 (if using csh/tcsh)*
     - *export OMP_NUM_THREADS=3 (if using bash/ksh)*

OpenMP Basics

- Another way is via the program.
  - **omp_set_num_threads**(*int num_threads*)
  - Allows one to change this number in between the program.

How to Compile and Run?

- The compilation and execution details of an OpenMP program may vary on different architectures.
- To compile a C-based OpenMP program:
  - ***# gcc -o <Name of executable> <program name> -fopenmp***
  - In general, use:
    - # <compiler> -o <executable-name> <program-name> <compiler-flag> -lm
- To execute:
  - ***./< Name of executable>***

Example Program Output

Hello World from thread = 0
Number of threads = 3
Hello World from thread = 2
Hello World from thread = 1

- The actual output you obtain may vary but will have the above model.

More Serious Examples

- Recall our prefix sum parallel algorithm reproduced below.

Input: Array *A of size n = 2k*

Output: Prefix sums in *C[0,j], 1 < j < n*

begin

1. for 1 < *j* < *n* pardo

    B[0,j] := A[j]

2. for *h = 1 to log n do*

    3. for 1 < *j* < *n/2*^{h} pardo

        B[h,j] := B[h–1,2j–1] + B[h–1,2j]

4. for *h = log n to 0 do*

    5. for 1 < *j* < *n/2*^{h} pardo

        if *j is even then C[h,j] := C[h+1,j/2]*

        else if *j = 1 then C[h,1] := B[h,1]*

        else *C[h,j] := C[h+1,(j–1)/2] + B[h,j]*

end

How to Write a Parallel Program?

- OpenMP lets us write for-loops to execute in parallel.
- This is done by using **#pragma omp for**.
- A simple example first.

A Simpler Example

```
#pragma omp parallel  \
            shared(n,x,y) private(i)
{
    #pragma omp parallel  for
    for (i=0; i<n; i++)
        x[i] += y[i];
} /*-- End of parallel region --*/
```

- This program adds the ith element of Y to the ith element of X.

A Simpler Example

```
#pragma omp parallel  \
               shared(n,x,y) private(i)
{
    #pragma omp parallel for
    for (i=0; i<n; i++)
        x[i] += y[i];
} /*-- End of parallel region --*/
```

- This program adds the ith element of Y to the ith element of X.
- What are these shared and private?
  - Indicate which data items are shared across threads, and which are not shared.
  - In this case, the arrays are to be shared by the threads, but the loop index should not be.

Model Execution

- Suppose we have n = 10 and we have 2 threads.
- Let X = (1, 2, 3,…,10) and Y = (2,3,5,7,11,13,17,19, 21, 23).

Thread 1 executes for indices i = 1, 2, 3, 4, 5.

i = 1: x[1] = 3

i = 2: x[2] = 5

i = 3: x[3] = 8

i = 4: x[4] = 11

i = 5: x[5] = 16

Thread 2 executes for indices i = 6, 7, 8, 9, 10.

i = 6: x[6] = 19

i = 7: x[7] = 24

i = 8: x[8] = 27

i = 9: x[9] = 30

i = 10: x[10] = 33

Back to Our Program

Begin

#pragma omp parallel for

shared(A, B, n), private(j)

1. for $1 < j < n$ *pardo*

    B[0,j] := A[j]

Back to Our Program

```
Begin
#pragma omp parallel for
    shared(n, A, B), private(j)
    1. for 1 < j < n pardo
          B[0,j] := A[j]
```

- At the end of this statement, we wish to write the next part of the program.
- This corresponds to the following

Back to Our Program

```
Begin
Omp_set_num_threads(4)
#pragma omp parallel for
        shared(A, B, n), private(j)
    1. for  (j=1; j<=n; j++)
          B[0,j] := A[j]
    2. . for (h = 1; h<=log n; h++)
          Omp_set_num_threads(8)
          #pragma omp parallel for
            shared(B, n), private(j)
          3. for 1 < j < n/2^h pardo
               B[h,j] := B[h−1,2j−1] + B[h−1,2j]
```

- At the end of this statement, we wish to write the next part of the program.
- This corresponds to the following

Back to Our Example

- The pragma is not on the outer loop.
- The outer loop has to be done in sequence.
- The inner loop can be done in parallel.

Back to Our Example

- The third loop has a similar property. The outer loop has to run in sequence.

```
for h = log n to 0 do
    #pragma omp parallel for shared(C, B)  private(j,n,h)
        5. for 1 < j < n/2^h
            if j is even then C[h,j] := C[h+1,j/2]
            else if i = 1 then C[h,1] := B[h,1]
            else C[h,j] := C[h+1,(j–1)/2] + B[h,j]
```

Some More Issues

```
#pragma omp parallel for
For i=1 to n do
     A[i] = f(i);
```

- Load balancing
  - Consider a parallel for loop with the action being f(i) for index i.

Some More Issues

```
#pragma omp parallel for
For i=1 to n do
     A[i] = f(i);
```

- Load balancing
  - Consider a parallel for loop with the action being f(i) for index i.
  - It is likely that the time taken to finish execution f(i) can vary for each i.
  - In that case, some threads finish their work early, and some others take longer.

Some More Issues

```
#pragma omp parallel for
For i=1 to n do
    A[i] = f(i);
```

- Load balancing
  - Consider a parallel for loop with the action being f(i) for index i.
  - It is likely that the time taken to finish execution f(i) can vary for each i.
  - In that case, some threads finish their work early, and some others take longer.
  - This is called as load imbalance.

Some More Issues

```
#pragma omp parallel for schedule (dynamic)
For i=1 to n do
     A[i] = f(i);
```

- OpenMP has additional features to help such settings.
- The modified program is as above.

Some More Issues

- There may be data dependencies across threads.
- For instance, at the same time that a thread is reading a shard variable, another thread may be writing to it.
- Or two threads may be writing at the same time to a shared variable.
- OpenMP has ways to handle this.

Some More Issues

- How to see the effect of parallel programs?
- One can measure the time taken on an execution.
- Highly dependent on the machine used, but presently no widely accepted abstract standard.
- Analytic models for parallel computing is an open research area.

Some More Issues

- Even practice is tricky.
- For small inputs, one may not see any benefit.
- This is due to some overheads involved.
- So, one should try for large enough inputs.
- For instance, the prefix sum program can be run on arrays of size 1 M and above.
- Typically one plots the runtime against input size.

One More Example

```
Program Merge(A, B)
Begin
    for i = 1 to n  do in parallel
        RA[i]  = BinSrch(B, A[i]) + i
    for i = 1 to n do in parallel
        RB[i] = BinSrch(A, B[i]) + i
    for i = 1 to n do in parallel
        C[RA[i]] = A[i]
    for i = 1 to n do in parallel
        C[RB[i]] = B[i];
end
```

- Recall our parallel version of the merge procedure.

One More Example

```
Program Merge(A, B)
Begin
     #pragma omp parallel for shared(A,B) private(i)
     for i = 1 to n  do
          RA[i]  = BinSrch(B, A[i]) + i
     #pragma omp parallel for shared(A,B) private(i)
     for i = 1 to n do in parallel
          RB[i] = BinSrch(A, B[i]) + I
     #pragma omp parallel for shared(A,B) private(i)
       for i = 1 to n do in parallel
          C[RA[i]] = A[i]
     #pragma omp parallel for shared(A,B) private(i)
      for i = 1 to n do in parallel
          C[RB[i]] = B[i];
end
```

- Recall our parallel version of the merge procedure.

Summary

- Writing basic parallel programs is easy now.
- There are only a few constructs of openmp that we have to know.
- We will try a few in the tutorials and the laboratory sessions.
  - I tried the prefix sum and the merge program before class today.
- There is more but we may visit those issues when needed.
- Happy parallel programming ☺☺☺☺