Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

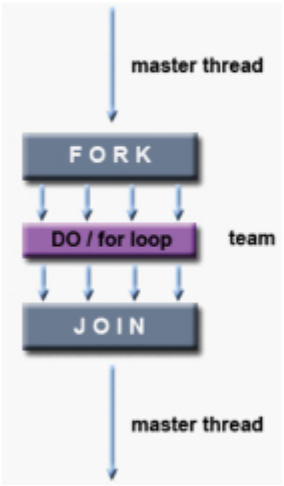
Content

- High Performance Parallel Computing
  - Overview (done)
  - Concepts and Terminology (done)
  - Parallel Computer Memory Architectures (done)
  - Parallel Programming Model (done)
    - ✔ Parallel Implementations – OpenMP Programming
    - ✔ Parallel Implementations – MPI Programming
  - Parallel Examples and Exercises

OpenMP Directives

- Work-Sharing Constructs
  - A work-sharing construct divides the execution of the enclosed code region among the members of the team that encounter it.
  - Work-sharing constructs do not launch new threads
  - There is no implied barrier upon entry to a work-sharing construct, however there is an implied barrier at the end of a work sharing construct.

Types of Work-Sharing Constructs:



DO / for - shares iterations of a loop across the team. Represents a type of "data parallelism".

SECTIONS - breaks work into separate, discrete sections. Each section is executed by a thread. Can be used to implement a type of "functional parallelism".

SINGLE - serializes a section of code.

Restrictions:

- A work-sharing construct must be enclosed dynamically within a parallel region in order for the directive to execute in parallel.
- Work-sharing constructs must be encountered by all members of a team or none at all
- Successive work-sharing constructs must be encountered in the same order by all members of a team

DO / for Directive

- Purpose:
  - The **DO / for** directive specifies that the iterations of the loop immediately following it must be executed in parallel by the team. This assumes a parallel region has already been initiated, otherwise it executes in serial on a single processor.
- Format:

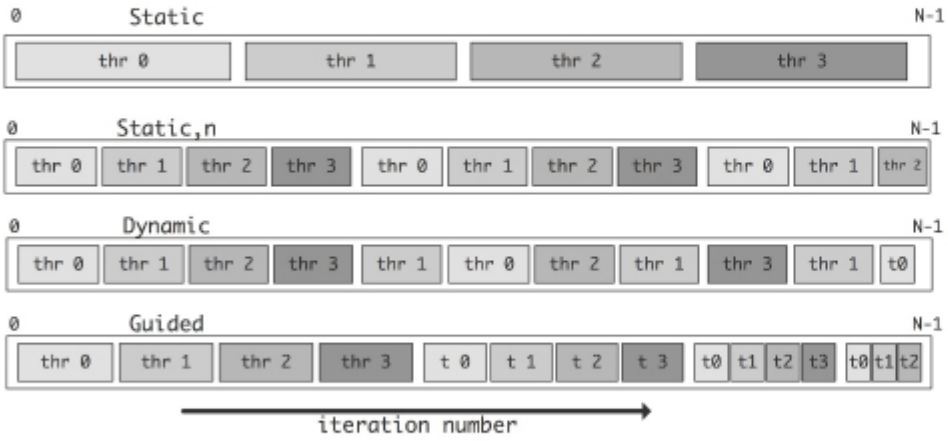
C/C++

```
#pragma omp for [clause ...] newline
        schedule (type [,chunk])
        ordered
        private (list)
        firstprivate (list)
        lastprivate (list)
        shared (list)
        reduction (operator: list)
        collapse (n)
        nowait

    for_loop
```

OpenMP: For Loop & Scheduling

#pragma omp parallel for schedule(scheduling-type)



- schedule(static, chunk-size)
- schedule(dynamic, chunk-size)
- schedule(guided, chunk-size)
- schedule(auto)

https://www.openmp.org/spec-html/5.1/openmpsu48.html#x73-730002.11.4

Example: DO / for Directive – Vector Addition

```c
// C / C++ - for Directive Example

#define N 1000


main(int argc, char *argv[]) {

int i       ;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;



  {


  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }

}
```

```c
// C / C++ - for Directive Example
#include <omp.h>
#define N 1000
#define CHUNKSIZE 100

main(int argc, char *argv[]) {

int i, chunk;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,chunk,N) private(i)
  {

  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }   /* end of parallel region */

}
```

OpenMP Directives

- Data Scope Attribute Clauses
  - Global (shared) variables:
    - ✔ File scope variables and static (in C)
  - Private variables:
    - ✔ Loop index variables
    - ✔ Stack variables in subroutines called from parallel regions

https://www.openmp.org/spec-html/5.1/openmpsu6.html#x13-120001.2.6

Data Scope Attribute Clauses

- The OpenMP Data Scope Attribute Clauses are used to explicitly define how variables should be scoped. They include:
  - PRIVATE
  - FIRSTPRIVATE
  - LASTPRIVATE
  - SHARED
  - DEFAULT
  - REDUCTION
  - COPYIN

PRIVATE Clause

- Purpose:
  - The PRIVATE clause declares variables in its list to be private to each thread.

- PRIVATE variables behave as follows:
  - A new object of the same type is declared once for each thread in the team
  - All references to the original object are replaced with references to the new object
  - Should be assumed to be uninitialized for each thread

SHARED Clause

- Purpose:
  - The SHARED clause declares variables in its list to be shared among all threads in the team.

- Notes:
  - A shared variable exists in only one memory location and all threads can read or write to that address
  - It is the programmer's responsibility to ensure that multiple threads properly access SHARED variables

Data Scope Attribute Clauses

- Data Scope Attribute Clauses are used in conjunction with several directives (**PARALLEL, DO/for,** and **SECTIONS**) to control the scoping of enclosed variables.

- These constructs provide the ability to control the data environment during execution of parallel constructs.

- Data Scope Attribute Clauses are effective only within their lexical/static extent.

Example: Shared / Private Variables

```c
// C / C++ - for Directive Example
#include <omp.h>
#define N 1000
#define CHUNKSIZE 100

main(int argc, char *argv[]) {

int i, chunk;
float a[N], b[N], c[N];

/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,chunk,N) private(i)
  {

  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }   /* end of parallel region */

}
```

Example: Shared / Private Variables

```
#pragma omp parallel for
{
   for(i=1; i<=n; i++){
      temp = 2.0*a[i];
      a[i] = temp;
      b[i] = c[i]/temp;
   }
}
```

Loop level Parallelization

- Requirements for Loop Parallelization
  - no dependencies between loop indices
  - an element of an array is assigned to by at most one iteration
  - no loop iteration reads array elements modified by any other dependency
  - due to overhead of parallelization - use only on loops where individual iterations take a long time

```
#pragma omp parallel for
  for(i=1; i<=n; i++)
     a[i] = b[i] + c[i]
```

```
#pragma omp parallel for
  for(i=2; i<=5; i++)
     a[i] = a[i] + a[i-1];
```

Quiz 1

- Will take a quiz on Feb 5th, Friday.
- Start at 11AM.
- Online on Canvas -> Quizzes
- Most are choice questions or multiple-choice questions.
- Everyone MUST turn on the camera; otherwise the quiz score will be treated as 0.

- Stay safe!
- See you next class!

Next Lecture will Continue:

Hands-on Lab: OpenMP Programming