Introduction to OpenMP

Christian Halloy
JICS at UT / ORNL
halloy@jics.utk.edu
Mar 20, 2012

Outline

I. About OpenMP
II. OpenMP Directives
III. Data Scope
IV. Runtime Library Routines and Environment Variables
V. Using OpenMP
I. About OpenMP

• Industry-standard *shared memory programming model*

• Developed in 1997

• OpenMP Architecture Review Board (ARB) determines additions and updates to standard
Advantages to OpenMP

• Parallelize small parts of application, one at a time (beginning with most time-critical parts)
• Can express simple or complex algorithms
• Code size grows only modestly
• Expression of parallelism flows clearly, so code is easy to read
• Single source code for OpenMP and non-OpenMP – non-OpenMP compilers simply ignore OMP directives
OpenMP Programming Model

- Application Programmer Interface (API) is combination of
  - Directives
  - Runtime library routines
  - Environment variables

- API falls into three categories
  - Expression of parallelism (flow control)
  - Data sharing among threads (communication)
  - Synchronization (coordination or interaction)
Parallelism

- Shared memory, thread-based parallelism
- Explicit parallelism (parallel regions)
- Fork/join model

Source: https://computing.llnl.gov/tutorials/openMP/
II. OpenMP Directives

- Syntax overview
- Parallel
- Loop
- Sections
- Synchronization
- Reduction
Syntax Overview: C/C++

- Basic format
  
  ```
  #pragma omp directive-name [clause] newline
  ```

- All directives followed by newline

- Uses pragma construct (pragma = Greek for “thing”)

- Case sensitive

- Directives follow standard rules for C/C++ compiler directives

- Long directive lines can be continued by escaping newline character with ‘\’
Syntax Overview: Fortran

• Basic format:
  \textit{sentinel} \textit{directive-name} [\textit{clause}]

• Three accepted sentinels: !$\textit{omp}$ *$\textit{omp}$ c$\textit{omp}$

• Some directives paired with \texttt{end} clause

• Fixed-form code:
  – Any of three sentinels beginning at column 1
  – Initial directive line has space/zero in column 6
  – Continuation directive line has non-space/zero in column 6
  – Standard rules for fixed-form line length, spaces, etc. apply

• Free-form code:
  – !$\textit{omp}$ only accepted sentinel
  – Sentinel can be in any column, but must be preceded by only white space and followed by a space
  – Line to be continued must end in “&” and following line begins with sentinel
  – Standard rules for free-form line length, spaces, etc. apply
OpenMP Directives: Parallel

• A block of code executed by multiple threads

• Syntax: C
  
  ```c
  #pragma omp parallel private(list)\ 
     shared(list)
  {
     /* parallel section goes here */
  }
  ```

• Syntax: Fortran
  
  ```fortran
  !$omp parallel private(list) &
  !$omp shared(list)
  ! Parallel section goes here
  !$omp end parallel
  ```
Simple Example (C/C++)

```c
#include <stdio.h>
#include <omp.h>
int main (int argc, char *argv[]) {
    int tid;
    printf("Hello world from C threads:\n");
    #pragma omp parallel private(tid)
    {
        tid = omp_get_thread_num();
        printf("<%d>\n", tid);
    }
    printf("I am sequential now\n");
    return 0;
}
```
program hello

integer tid, omp_get_thread_num

write(*,*) 'Hello world from Fortran threads:'

!$OMP parallel private(tid)

  tid = omp_get_thread_num()

!$omp end parallel

write(*,*) '<', tid, '>'

write(*,*) 'I am sequential now'
end
Output (Simple Example)

Output 1
Hello world from C
threads:
<0>
<3>
<2>
<4>
<1>
I am sequential now

Output 2
Hello world from
Fortran threads:
<1>
<2>
<0>
<4>
<3>
I am sequential now

Order of execution is scheduled by OS!!!
OpenMP Directives: Loop

• Iterations of the “for” or “do” loop following the directive are executed in parallel

• Syntax: C

```c
#pragma omp for schedule(type [,chunk]) \ 
private(list) shared(list) nowait
{
    /* for loop goes here */
}
```

• Syntax: Fortran

```fortran
!$OMP do schedule(type [,chunk]) & 
!$OMP private(list) shared(list)
! do loop goes here
!$OMP end do nowait
- type={static, dynamic, guided, runtime}
- If nowait specified, threads do not synchronize at end of loop
```
Which Loops Are Parallelizable?

Parallelizable

- Number of iterations known upon entry, and does not change
- Each iteration independent of all others
- No data dependence

Not Parallelizable

- Conditional loops (many while loops)
- Iterator loops (e.g., iterating over a `std::list<...>` in C++)
- Iterations dependent upon each other
- Data dependence
Example: Parallelizable?

```c
/* Gaussian Elimination (no pivoting):
   x = A\b */

for (int i = 0; i < N-1; i++) {
    for (int j = i; j < N; j++) {
        double ratio = A[j][i]/A[i][i];
        for (int k = i; k < N; k++) {
            A[j][k] -= (ratio*A[i][k]);
            b[j] -= (ratio*b[i]);
        }
    }
}
```
Example: Parallelizable?
Example: Parallelizable?

- **Outermost Loop (i):**
  - N−1 iterations
  - Iterations depend upon each other (values computed at step \(i-1\) used in step \(i\))

- **Inner loop (j):**
  - N−i iterations (constant for given \(i\))
  - Iterations can be performed in any order

- **Innermost loop (k):**
  - N−i iterations (constant for given \(i\))
  - Iterations can be performed in any order
Example: Parallelizable?

/* Gaussian Elimination (no pivoting): 
   \[ x = A\backslash b \] */

for (int i = 0; i < N-1; i++) {
    #pragma omp parallel for
    for (int j = i; j < N; j++) {
        double ratio = A[j][i]/A[i][i];
        for (int k = i; k < N; k++) {
            A[j][k] -= (ratio*A[i][k]);
            b[j] -= (ratio*b[i]);
        }
    }
}

Note: can combine `parallel` and `for` into single `pragma` line
OpenMP Directives: Loop Scheduling

• Default scheduling determined by implementation

• Static
  – ID of thread performing particular iteration is function of iteration number and number of threads
  – Statically assigned at beginning of loop
  – Load imbalance may be issue if iterations have different amounts of work

• Dynamic
  – Assignment of threads determined at runtime (round robin)
  – Each thread gets more work after completing current work
  – Load balance is possible
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000
int main () {
    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) private(i)
    {
        #pragma omp for schedule(dynamic,chunk) nowait
        for (i=0; i < N; i++)
            c[i] = a[i] + b[i];
    } /* end of parallel section */
    return 0;
}
OpenMP Directives: Sections

- Non-iterative work-sharing construct
- Divide enclosed sections of code among threads
- Section directives nested within sections directive

Syntax:

C/C++

```c
#pragma omp sections
{
    #pragma omp section
    /* first section */
    #pragma omp section
    /* next section */
}
```

Fortran

```
!$OMP sections
!
!$OMP section
C First section
!$OMP section
C Second section
!$OMP end sections
```
#include <omp.h>
define N 1000
int main () {
    int i;
double a[N], b[N], c [N], d[N];
    /* Some initializations */
    for (i=0; i < N; i++) {
        a[i] = i * 1.5;
        b[i] = i + 22.35;
    }
    #pragma omp parallel \
    shared(a,b,c,d) private(i)
    {
        #pragma omp sections nowait
        {
            #pragma omp section
            for (i=0; i < N; i++)
                c[i] = a[i] + b[i];
            #pragma omp section
            for (i=0; i < N; i++)
                d[i] = (a[i] * b[i]) + 0.2;
        } /* end of sections */
    } /* end of parallel section */
    return 0;
}
OpenMP Directives: Synchronization

• Sometimes, need to make sure threads execute regions of code in proper order
  – Maybe one part depends on another part being completed
  – Maybe only one thread need execute a section of code

• Synchronization directives
  – Critical
  – Barrier
  – Single
OpenMP Directives: Synchronization

- **Critical**
  - Specifies section of code that must be executed by only one thread at a time
  - Syntax:
    - C/C++
      ```
      #pragma omp critical [name]
      ```
    - Fortran
      ```
      !$OMP critical [name]
      !$OMP end critical
      ```
  - Names are global identifiers – critical regions with same name are treated as same region

- **Single**
  - Enclosed code is to be executed by only one thread
  - Useful for thread-unsafe sections of code (e.g., I/O)
  - Syntax:
    - C/C++
      ```
      #pragma omp single
      ```
    - Fortran
      ```
      !$OMP single
      !$OMP end single
      ```
OpenMP Directives: Synchronization

• Barrier
  – Synchronizes all threads: thread reaches barrier and waits until all other threads have reached barrier, then resumes executing code following barrier
  – Syntax: C/C++
    
    ```
    #pragma omp barrier
    ```
  – Fortran
    
    ```
    !$OMP barrier
    ```
  – Sequence of work-sharing and barrier regions encountered must be the same for every thread
OpenMP Directives: Reduction

• Reduces list of variables into one, using operator (e.g., max, sum, product, etc.)

• Syntax

C    #pragma omp reduction(op : list)
Fortran    !$OMP reduction(op : list)

where list is list of variables and op is one of following:

- C/C++: +, -, *, &, ^, |, &&, or ||
- Fortran: +, -, *, .and., .or., .eqv., .neqv., or max, min, iand, ior, ieor
III. Data (variable) Scope

• By default, all variables are shared except
  – Certain loop index values – private by default
  – Local variables and value parameters within subroutines called within parallel region – private
  – Variables declared within lexical extent of parallel region – private
Default Scope Example

```c
void caller(int *a, int n) {
    int i,j,m=3;
    #pragma omp parallel for
    for (i=0; i<n; i++) {
        int k=m;
        for (j=1; j<=5; j++) {
            callee(&a[i], &k, j);
        }
    }
}
void callee(int *x, int *y, int z) {
    int ii;
    static int cnt;
    cnt++;
    for (ii=1; ii<z; ii++) {
        *x = *y + z;
    }
}
```

<table>
<thead>
<tr>
<th>Var</th>
<th>Scope</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>shared</td>
<td>Declared outside parallel construct</td>
</tr>
<tr>
<td><code>n</code></td>
<td>shared</td>
<td>same</td>
</tr>
<tr>
<td><code>i</code></td>
<td>private</td>
<td>Parallel loop index</td>
</tr>
<tr>
<td><code>j</code></td>
<td>shared</td>
<td>Sequential loop index</td>
</tr>
<tr>
<td><code>m</code></td>
<td>shared</td>
<td>Declared outside parallel construct</td>
</tr>
<tr>
<td><code>k</code></td>
<td>private</td>
<td>Automatic variable/parallel region</td>
</tr>
<tr>
<td><code>x</code></td>
<td>private</td>
<td>Passed by value</td>
</tr>
<tr>
<td><code>*x</code></td>
<td>shared</td>
<td>(actually a)</td>
</tr>
<tr>
<td><code>y</code></td>
<td>private</td>
<td>Passed by value</td>
</tr>
<tr>
<td><code>*y</code></td>
<td>private</td>
<td>(actually k)</td>
</tr>
<tr>
<td><code>z</code></td>
<td>private</td>
<td>(actually j)</td>
</tr>
<tr>
<td><code>ii</code></td>
<td>private</td>
<td>Local stack variable in called function</td>
</tr>
<tr>
<td><code>cnt</code></td>
<td>shared</td>
<td>Declared static (like global)</td>
</tr>
</tbody>
</table>
Variable Scope

• Good programming practice: explicitly declare scope of all variables

• This helps you as programmer understand how variables are used in program

• Reduces chances of data race conditions or unexplained behavior
Variable Scope: Shared

• Syntax
  – `shared(list)`

• One instance of shared variable, and each thread can read or modify it

• WARNING: watch out for multiple threads simultaneously updating same variable, or one reading while another writes

• Example

```c
#pragma omp parallel for shared(a)
for (i = 0; i < N; i++) {
    a[i] += i;
}
```
Variable Scope: Shared – Bad Example

```c
#pragma omp parallel for shared(n_eq)
for (i = 0; i < N; i++) {
    if (a[i] == b[i]) {
        n_eq++;
    }
}
```

- `n_eq` will not be correctly updated
- Instead, put `n_eq++;` in critical block (slow);
  or
- introduce private variable `my_n_eq`, then update `n_eq` in critical block after loop (faster);
  or
- use reduction pragma (best)
Variable Scope: Private

• Syntax
  – private(list)

• Gives each thread its own copy of variable

• Example
  
  ```c
  #pragma omp parallel private(i, my_n_eq)
  {
    #pragma omp for
    for (i = 0; i < N; i++) {
      if (a[i] == b[i]) my_n_eq++;
    }
    #pragma omp critical (update_sum)
    {
      n_eq+=my_n_eq;
    }
  }
  ```
Best Solution for Sum

```c
#pragma parallel for reduction(+:n_eq)
for (i = 0; i < N; i++) {
    if (a[i] == b[i]) {n_eq = n_eq+1;}
}
```
IV. OpenMP Runtime Library Routines and Environment Variables

OpenMP Runtime Library Routines

C: void omp_set_num_threads(int num_threads)

Fortran: subroutine omp_set_num_threads
( scalar_integer_expression )

- Sets number of threads used in next parallel region
- Must be called from serial portion of code
OpenMP Runtime Library Routines

- `int omp_get_num_threads()`
  
  `integer function omp_get_num_threads()`
  
  - Returns number of threads currently in team executing parallel region from which it is called

- `int omp_get_thread_num()`
  
  `integer function omp_get_thread_num()`
  
  - Returns rank of thread
  - $0 \leq \text{omp_get_thread_num}() < \text{omp_get_num_threads}()$
OpenMP Environment Variables

- Set environment variables to control execution of parallel code

  - **OMP_SCHEDULE**
    - Determines how iterations of loops are scheduled
    - E.g., `setenv OMP_SCHEDULE "guided, 4"`

  - **OMP_NUM_THREADS**
    - Sets maximum number of threads
    - E.g., `setenv OMP_NUM_THREADS 4`
V. USING OPENMP
Conditional Compilation

• Can write single source code for use with or without OpenMP
• Pragmas/sentinels are ignored
• What about OpenMP runtime library routines?
  – _OPENMP macro is defined if OpenMP available: can use this to conditionally include omp.h header file, else redefine runtime library routines
Conditional Compilation

```c
#ifdef _OPENMP
   #include <omp.h>
#else
   #define omp_get_thread_num() 0
#endif
...
int me = omp_get_thread_num();
...```

RDAV/NICS/JICS – OpenMP Programming – Mar 20, 2012
Compiling Programs with OpenMP Directives on Jaguar and Kraken

• **Compiler flags:**
  - `-mp=nonuma` (PGI)
  - `-fopenmp` (GNU)
  - `-mp` (Pathscale)
  - (default w/Cray compiler) `-xomp` or `-hnoomp` to deactivate

• Many libraries already compiled with OpenMP directives

• **Libsci** (default version 11.0.04, module `xt-libsci`)
  - link with `-lsci_istanbul_mp`
Running Programs with OpenMP Directives on Jaguar and Kraken

- Set environment variable OMP_NUM_THREADS in batch script
- Use the depth (−d) in aprun command to represent number of threads per MPI process, and −N for number of MPI processes per node, and −S to distribute MPI procs per Socket.
- Example: to run 8 MPI tasks, each with 6 threads on the hex-core nodes on Jaguar or Kraken, add the following to your script requesting 48 procs (and 4 compute nodes):

  ```
  export OMP_NUM_THREADS=6
  aprun -n 8 -S 1 -d 6 myprog.exe (-N 2 not needed)
  
  NOTE: size has to be multiple of 12 !!!
  #PBS -l size=48
  ```
More about aprun

- **-n pes**
  - Allocates *pes* processing elements (PEs, think MPI tasks)

- **-N pes_per_node / -S pes_per_socket**
  - Specifies number of processing elements to place per node / per socket
  - Reducing number of PEs per node makes more resources (i.e. memory!) available per PE

- **-d depth**
  - Allocates number of CPUs to be used by each PE and its threads (default 1)
  - If you set OMP_NUM_THREADS but do not specify depth, all threads will be allocated on a single core!

- **pes * pes_per_socket * depth ≤ “size” in PBS header!**
Simple Example – performance issues!

```c
#include <omp.h>
#define N 100000    /* or 1000000 */
int main () {
    int i, iter;    double a[N],b[N],c[N];
    for (iter=0; iter < 100000; iter++) {    /* or 10000 */
        #pragma omp parallel for shared(a,b,c) private(i)
        for (i=0; i < N; i++) {
            a[i] = i * 1.5;    b[i] = i + 22.35;
            c[i] = a[i] * b[i]; }
    }
}
```

<table>
<thead>
<tr>
<th>Iterations</th>
<th>No threads</th>
<th>6 threads</th>
<th>Speedup</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter=100,000</td>
<td>33.79 secs</td>
<td>5.35 secs</td>
<td>~6.3</td>
<td>~400KB/core</td>
</tr>
<tr>
<td>iter=10,000</td>
<td>59.37 secs</td>
<td>46.94 secs</td>
<td>~1.3</td>
<td>~4MB/core</td>
</tr>
</tbody>
</table>

Where’s the catch? The catch is in the Cache! Mem= 3 * N * 8 bytes used
Performance is limited by size of Cache memory (~1 Mbytes / core)
OPENMP Hello World – Fortran Code

PROGRAM HELLO
INTEGER   NTHREADS, TID, OMP_GET_NUM_THREADS,
INTEGER   OMP_GET_THREAD_NUM
C Fork a team of threads giving them their own copies of variables
!$OMP PARALLEL PRIVATE(NTHREADS, TID)
C Obtain thread number
   TID = OMP_GET_THREAD_NUM()
   PRINT *, 'Hello World from thread = ', TID
C Only master thread does this
   IF (TID .EQ. 0) THEN
      NTHREADS = OMP_GET_NUM_THREADS()
      PRINT *, 'Number of threads = ', NTHREADS
   END IF
C All threads join master thread and disband
!$OMP END PARALLEL
END
OPENMP Hello World – C code

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

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

    /* Fork a team of threads giving them their own copies of variables */
    #pragma omp parallel private(nthreads, tid) {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d
", tid);
        /* Only master thread does this */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads); }
    }
    /* All threads join master thread and disband */
}
Compile and run OpenMP on Kraken

```bash
#!/bin/bash
#PBS -A UT-TNEDU002
#PBS -N test
#PBS -j oe
#PBS -l walltime=1:00:00,size=12
cd $PBS_O_WORKDIR
Date
export OMP_NUM_THREADS=4
aprun -n 1 -S1 -d4 ./hello
```

Hello World from thread = 0
Number of threads = 4
Hello World from thread = 3
Hello World from thread = 1
Hello World from thread = 2
Bibliography/Resources: OpenMP


- LLNL OpenMP Tutorial, [https://computing.llnl.gov/tutorials/openMP/](https://computing.llnl.gov/tutorials/openMP/)

- Thanks to Rebecca Hartman-Baker for her original set of slides (Cray XT4 workshop Apr 09)

- Thanks to Kwai Wong for reviewing examples

The End!