Lightning Introduction to OpenMP

May, 2015
What is OpenMP?

Open Multiprocessing API
First published 1997, v.3.0 published 2008
A standard for shared-memory programming
With support from: AMD, Cray, Fujitsu, HP, IBM, Intel, Portland Group, SGI, Oracle, MS, TI; also ANL, LLNL, EPCC, NASA...
Fortran, C and C++ specifications in standard
Shared-memory ≈ inside one multi-core machine
Allows incremental parallelization
Want to Follow Along?

$ cp /home/rdickson/demo/OpenMP/* .
$ ls
hello.c  mandel.f90  saxpy.f90
job.sh   reduce.f95  saxpy.in
Makefile reduce.in
Coding a Simple Loop

saxpy.f90

subroutine saxpy(a, x, y, n)
integer i, n
real a, x(n), y(n)

do i = 1, n
   y(i) = a * x(i) + y(i)
end do
return
end
Coding a Simple Loop

saxpy.f90

subroutine saxpy(a, x, y, n)
integer i, n
real a, x(n), y(n)
!$omp parallel do
do i = 1, n
   y(i) = a * x(i) + y(i)
end do
return
end

Most work done with compiler directives:

!$omp directive in Fortran
#pragma omp directive in C, C++
Compilation

OpenMP supported by these compilers at ACEnet:
  • Portland Group (PGI)
  • Intel
  • GCC version 4 and later
  • SunStudio

Different flags for each compiler suite:

$ pgf90 -mp saxpy.f90
$ ifort -openmp saxpy.f90
$ gfortran -fopenmp saxpy.f90
$ f90 -openmp saxpy.f90
Execution

Environment variable OMP_NUM_THREADS controls number of threads

```
$ export OMP_NUM_THREADS=3
$ ./hello
Hello World from thread 0
Hello World from thread 2
Hello World from thread 1
There are 3 threads
$
```

bash syntax shown; in tcsh use “setenv OMP_NUM_THREADS 3”
Use With Grid Engine

Best practice: Set OMP_NUM_THREADS from SGE variable NSLOTS

```bash
#$ -cwd
#$ -j y
#$ -l h_rt=0:1:0,test=true
#$ -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./hello
```
Key Programming Concepts

Threads, fork, join
Shared vs. private variables
Reduction
Data dependence
Parallel regions
Scheduling
Threads

Master thread “forks” (creates) worker threads when code enters a parallel section.

Threads “join” (vanish) on leaving a parallel section --- or at least we must assume so!
Shared vs. Private Memory

Whole program has a shared memory area
Each thread is created with some private memory
Shared vs. Private Memory

Consider “saxpy” example:
Array index “i” must be private to each thread

- Array indices: $x(1\ldots n)$, $y(1\ldots n)$
- Array index $i$ must be private to each thread
  - $i = 1\ldots$
  - $i = 26\ldots$
  - $i = 76\ldots$
  - $i = 51\ldots$

Threads:
- Thread 0
- Thread 1
- Thread 2
- Thread 3
Shared vs. Private Variable

- There are default rules for **scope**
- Most variables shared by default, except Fortran loop indices
- C “for” construct is more complicated
- Most private vars must have scope declared
- Function (“stack”) locals are private
- You *can* scope everything explicitly – maybe a good idea?
- See references for details...
Mandelbrot Set Example

For each point \((x, y)\):

\[ c \leftarrow x + iy, \quad z_0 \leftarrow 0 \]

Iterate...

\[ z_{k+1} \leftarrow z_k^2 + c \]

...until \(z\) diverges (out)
or iteration limit reached (in)
Scope and Nested Loops

```fortran
mandel.f90

!$omp parallel do private(?, ?, ?, ?, ?)
do j = 1, m
  do k = 1, n
    x = xmin + j*(xmax-xmin)/m
    y = ymin + k*(ymax-ymin)/n
    depth(k, j) = mandel_val(x, y, maxiter)
  end do
end do
```

Usually best to parallelize outermost loop → ‘j’ private
What about ‘k’?
‘k’ must be private too
as must 'x' and 'y'
'mandel_val' should be a *pure function*
Scope and Nested Loops

mandel.f90

```fortran
!$omp parallel do private(k, x, y)
do j = 1, m
  do k = 1, n
    x = xmin + j*(xmax-xmin)/m
    y = ymin + k*(ymax-ymin)/n
    depth(k, j) = mandel_val(x, y, maxiter)
  end do
end do
end do
```

Usually best to parallelize outermost loop → 'j' private
What about ‘k’?
‘k' must be private too
as must 'x' and 'y'
'mandel_val' should be a pure function
Summing

reduce.f90

```fortran
accumulator = 0.0
 !$omp parallel do  ???
 do i = 1, n
   accumulator = accumulator + x(i)
 end do
 print *, accumulator
```

Should `accumulator` be shared or private?
Answer: Neither. It is a reduction variable.
Reduction operators: sum, product, max, min, logical or, bitwise or, logical and, ...
OpenMP controls access to reduction variable so threads don't suffer race condition
Reduction

reduce.f90

accumulator = 0.0
!$omp parallel do reduction(+:accumulator)
do i = 1, n
   accumulator = accumulator + x(i)
end do
print *, accumulator

Should **accumulator** be shared or private?  
Answer: Neither. It is a *reduction variable*.  
Reduction operators: sum, product, max, min, logical or, bitwise or, logical and, ...  
OpenMP controls access to reduction variable so threads don't suffer race condition
Data Dependence

If one statement writes a memory location, and another statement either reads or writes the same location, then there is a data dependence between the two statements.

The order in which they are executed affects program correctness...

...and therefore affects parallelizability.
Data Dependence Examples

\[
\begin{align*}
\text{do } & i = 2, n \\
& a(i) = a(i) + a(i-1) \\
\end{align*}
\]
end do

Cannot parallelize this loop:
a(i-1) must be written in iteration (i-1) before it is read in iteration (i).

\[
\begin{align*}
\text{do } & i = 2, n, 2 \\
& a(i) = a(i) + a(i-1) \\
\end{align*}
\]
end do

Can parallelize this loop:
Even-numbered elements written; odd-numbered elements read – no dependence
Data Dependence Examples

What about this one?

```fortran
  do i = 1, n
      a(idx(i)) = a(idx(i)) + b(idx(i))
  end do
```

Depends on array `idx`

- If it's a *permutation*, then no dependence
- If it contains duplicate entries, then not okay
- Compiler can't tell!

Also: Beware common, module, global or static vars inside functions & subroutines!
Parallel Regions

```plaintext
integer myid, nthreads
nthreads = omp_get_num_threads()
 !$omp parallel private(myid)
myid = omp_get_thread_num()
call do_different_things( myid, nthreads )
 !$omp end parallel
```

Can parallelize general code, not just loops
Probe OpenMP with functions
Load Balance & Scheduling

```c
!$omp parallel do private(k, x, y) &
!$omp schedule(dynamic,10)
do j = 1, m
    do k = 1, n
        x = xmin + j*(xmax-xmin)/m
        y = ymin + k*(ymax-ymin)/n
        depth(k, j) = mandel_val(x, y, maxiter)
    end do
end do
```

Which loop iterations are assigned to which thread may affect performance – Idle threads inefficient!

`schedule` clause tells OpenMP how to balance load
Not to Mention...

Types of data dependence and ways to fix some of them
  • firstprivate, lastprivate, copyin
  • do if, nowait

Synchronization
  • !$omp critical, barrier, atomic, ordered

Work-sharing
  • !$omp sections, single, master
OpenMP + MPI

Hybrid code easy to write
Can give better performance in some cases; depends on the details
See Quinn for examples
But...
...interface with scheduler makes things tricky. With ACEnet's Grid Engine, 
\texttt{-pe ompi* 8} may give different number of slots on different hosts
Code must handle this, \textit{or}...

... use unadvertised:
\texttt{-pe 2per 8}
\texttt{-pe 4per 8}
OpenMP

THE OPENMP API SPECIFICATION FOR PARALLEL PROGRAMMING

OpenMP News

SPEC Looking For A Few Good Applications

SPEC, the Standard Performance Evaluation Corporation, is looking for realistic OpenMP applications to include in the next version of the SPEC CPU and SPEC OMP benchmark suites.

SPEC is sponsoring a search program, and for each step of the process that a submission passes, SPEC will compensate the Program Submitter (in recognition of the Submitter’s effort and skill). A submission that passes all of the steps and is included in the next SPEC CPU benchmark suite will receive $5000 US overall and a license for the new benchmark suite when released. Details on the Benchmark Search Program at http://www.spec.org/cpu6/.

Posted on May 20, 2010

IWOMP 2010: International Workshop on OpenMP

6th International Workshop on OpenMP, June 14-16, 2010, Tsukuba, Japan

“Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More”

The International Workshop on OpenMP is an annual series of workshops dedicated to the promotion and advancement of all aspects focusing on parallel programming with OpenMP. OpenMP is now a major programming model for shared-memory systems from multicore...
References

Local documentation:

- http://www.openmp.org
- http://www.accelerateddiscovery.ca/wiki/OpenMP

Examples drawn from:

- Rohit Chandra et al., *Parallel Programming in OpenMP* (Academic Press, 2001)

Example code in /home/rdickson/demo/OpenMP/*