High-Performance Computing

Parallel Programming in

OpenMP

Programming OpenMP

Outline

❏ OpenMP Overview

❏ OpenMP

  ❏ Basics

  ❏ Syntax: Directives

  ❏ Syntax: Clauses

  ❏ Example

OpenMP – Overview

What is OpenMP?

OpenMP – Overview

❏ OpenMP: stands for

   Open Multi Processing

❏ parallel programming model for shared memory multiprocessors

❏ 'de-facto' standard, not an industry *standard*

❏ not a new language, but

   ❏ compiler directives

   ❏ support function library

OpenMP – Overview

❏ OpenMP development is community driven

❏ Architecture Review Board (ARB):

  ❏ hardware and software vendors

  ❏ government and academia

❏ Official OpenMP website:

  ❏ https://www.openmp.org/

OpenMP – Overview

❏ standard versions:

  ❏ C/C++: version 2.0 (March 2002)

  ❏ Fortran: version 2.0 (November 2000)

  ❏ version 2.5 (Fortran and C/C++ / May 2005)

  ❏ version 3.0 (May 2008)

  ❏ version 3.1 (July 2011)

  ❏ ...

OpenMP – Overview

- ❑ 4.0 (Jul 2013)
    - ❑ exception handling
- ❑ 4.5 – update (Nov 2015)
    - ❑ support for accelerators (GPUs, Xeon Phi)
- ❑ 5.0 (Nov 2018)
    - ❑ is implemented in most compilers now
- ❑ 5.1 – update (Nov 2020)
- ❑ The OpenMP standard specifications:
    - ❑ http://www.openmp.org/specifications/

OpenMP – Overview

OpenMP Literature:

- ❑ "Using OpenMP – The Next Step" by R. van der Pas, E. Stotzer and C. Terboven, MIT Press (2017) – available via findit.dtu.dk
- ❑ "The OpenMP Common Core" by T.G. Mattson, Y. He, and A.E. Koniges, MIT Press (2019)
- ❑ "Using OpenMP", B. Chapman, G. Jost, R. van der Pas, MIT Press (2007)

OpenMP Basics

Basic elements of OpenMP

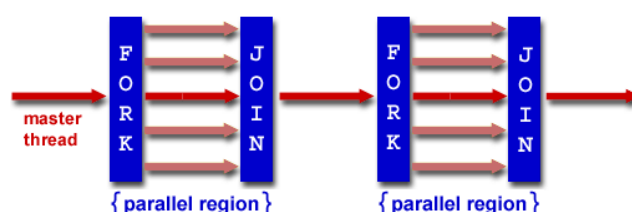
OpenMP Basics

OpenMP uses the "Fork-Join Model":

❏ All programs begin as a single process: the master thread.

❏ FORK: the master thread creates a team of parallel threads (parallel region).

❏ JOIN: synchronization and termination of the worker threads.

OpenMP Basics

OpenMP is mostly based on compiler directives:

❏ C/C++:

```
#pragma omp directive [clause]
    {...code block...}
```

❏ Fortran:

```
!$OMP directive [clause]
    ...code block...
!$OMP end directive
```

OpenMP Basics

The OpenMP API has also

❏ a set of support library functions:
    omp_... ()

  e.g. omp_get_thread_num()

❏ control via environment variables:

    OMP_...
  e.g. OMP_NUM_THREADS

OpenMP Basics

First OpenMP version of "Hello world":

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    #pragma omp parallel
    {
    printf("Hello world!\n");
    } /* end parallel */
    return(0);
}
```

Programming OpenMP

OpenMP Basics

Second version of "Hello world":

```c
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

int main(int argc, char *argv[]) {
    int  t_id = 0;
    #pragma omp parallel private(t_id)
    {
    #ifdef _OPENMP
    t_id = omp_get_thread_num();
    #endif
    printf("Hello world from %d!\n", t_id);
    } /* end parallel */
    return(0);
}
```

Programming OpenMP

OpenMP Basics

```
$ ./hello2
Hello world from 0!

$ OMP_NUM_THREADS=4 ./hello2
Hello world from 0!
Hello world from 3!
Hello world from 1!
Hello world from 2!
```

❏ Note: The order of execution will be different from run to run!

❏ The default no. of threads depends on the OpenMP implementation

Programming OpenMP

OpenMP Components

Programming OpenMP

| **Directives**	**Environment variables**	**Runtime**
❏ Parallel regions	❏ no. of threads	❏ no. of threads
❏ Worksharing	❏ Scheduling	❏ Scheduling
❏ Synchronization	❏ Dynamic thread adjustment	❏ Dynamic thread adjustment
❏ Data scoping	❏ Nested parallelism	❏ Nested parallelism
❏ no. of threads		❏ API for timers & locking
❏ Orphaning		

OpenMP Basics

Work-sharing

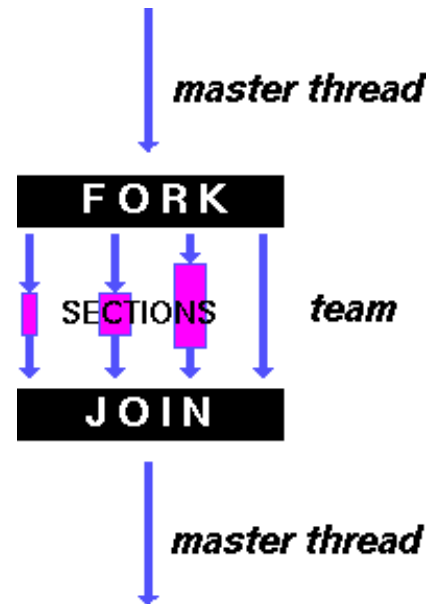
OpenMP Basics

Work-sharing
constructs – 1:

❑ do/for

❑ loop parallelism

❑ most common
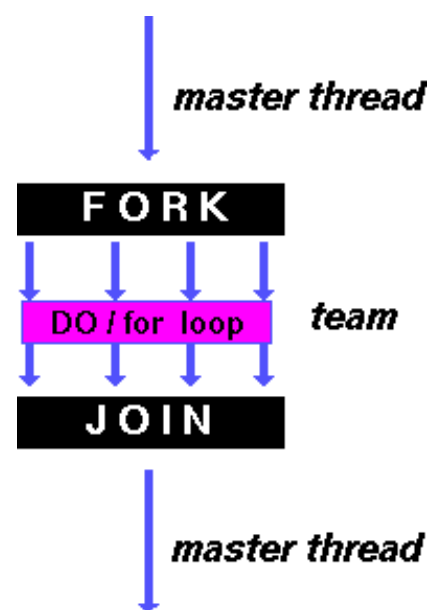
OpenMP Basics

Work-sharing
    constructs – 2:

❏ sections

❏ functional
    parallelism

❏ typically
    independent
    calculations

master thread

FORK

SECTIONS   *team*

JOIN

master thread

OpenMP Basics

Work-sharing
    constructs – 3:

❏ single

❏ work assigned to
    one thread only

❏ typically I/O

master thread

FORK

SINGLE   *team*

JOIN

master thread
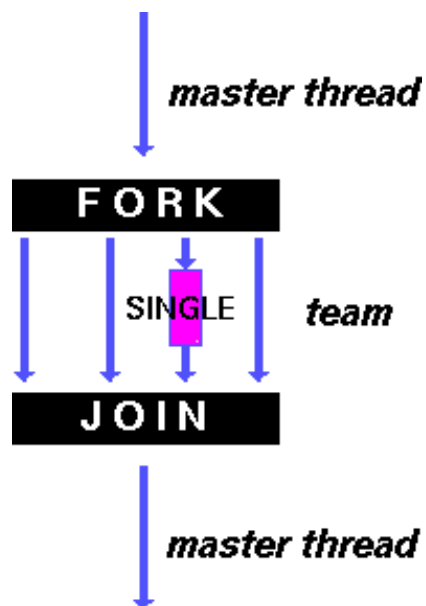
OpenMP Basics

Important rules for work-sharing constructs:

❏ must be enclosed in a parallel region

❏ must be encountered by all team members

❏ must be encountered in the same order

OpenMP Basics

Synchronization

OpenMP Basics

- ❏ most synchronization in OpenMP is implicit, but sometimes explicit synchronization is needed:
- ❏ barriers
- ❏ critical regions
- ❏ master only
- ❏ explicit locking

FORK

JOIN
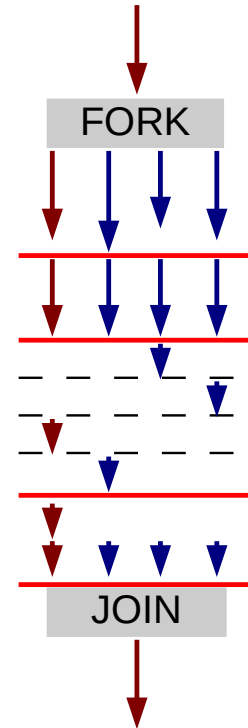
OpenMP Syntax

OpenMP programming in C/C++
Part I: Directives

OpenMP Syntax

OpenMP directives – general form:

```
#pragma omp directive [[clause] \
  [clause] ...]
{
  <statements>
} /* end of omp directive */
```

❏ **Note**: There is **no** "omp end" pragma!

❏ Best practice: add a comment at the end of the structured block!

OpenMP Syntax

Parallel region:

❏ Starts a team of parallel threads

❏ executes code in parallel

❏ synchronize/terminate threads

```
main() {
    A();
    #pragma omp parallel
    {
    B();    // all threads do B()!
    } /* end omp parallel */
    C();
}
```

OpenMP Syntax

Work-sharing – Loop parallelism:

❏ OpenMP implements parallel do/for-loops only!

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

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

#pragma omp parallel shared(a,b,c) private(i)
{
  #pragma omp for
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

} /* end of parallel region */
```

for *has to follow the pragma – no {... }!*

OpenMP Syntax

Work-sharing – Loop parallelism:

❏ Another version: combined "parallel for"

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

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

#pragma omp parallel for shared(a,b,c)\
                         private(i)
for (i=0; i < N; i++)
  c[i] = a[i] + b[i];
```

OpenMP Syntax

Work-sharing – Fortran 95 array syntax

❏ Fortran 95 allows to address parts of or whole arrays – and the compiler will translate this into loops.

❏ A special Fortran directive:

```fortran
double precision, dimension() :: A, B
double precision, dimension(N,M) :: C

!$ OMP WORKSHARE
   ...
   A(1:M) = A(1:M) + B(1:M)
   C = 0.00
   ...
!$ OMP END WORKSHARE
```

Programming OpenMP

OpenMP Syntax

Work-sharing – Functional parallelism:

❏ Parallel sections:

```c
#pragma omp parallel shared(a,b,c) private(i)
{
  #pragma omp sections
  {
    #pragma omp section
    for (i=0; i < N/2; i++)
      c[i] = a[i] + b[i];

    #pragma omp section
    for (i=N/2; i < N; i++)
      c[i] = a[i] + b[i];
  }  /* end of sections */
} /* end of parallel region */
```

Programming OpenMP

OpenMP Syntax

Work-sharing – Single thread execution:

❑ Work done by one thread only

```
#pragma omp parallel shared(a,b,c) private(i)
{
  #pragma omp single
  { read_array(a); read_array(b); }

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

  #pragma omp single
  { write_array(c); }
}  /* end of parallel section */
```

OpenMP Syntax

Work-sharing – conditional parallelism:

❑ the if(...) clause

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

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

#pragma omp parallel if (N > 10000) \
        shared(a,b,c) private(i)
{
  #pragma omp for
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];
}
```

OpenMP Syntax

The num_threads(...) clause:

```
#pragma omp parallel ... num_threads(int_expr)
{
    ...
}
```

❏ only one num_threads clause per parallel directive

❏ int_expr is evaluated before the parallel region is entered

OpenMP syntax

The collapse(n) clause

❏ a way to parallelize loop nests

```
subroutine sub()

!$omp do collapse(2) private(i,j,k)
  do k = kl, ku, ks
    do j = jl, ju, js
      do i = il, iu, is
        call bar(a,i,j,k)
      enddo
    enddo
  enddo
!$omp end do
end subroutine
```

collapse the two outer loops over k and j

OpenMP Syntax

Synchronization – Critical region:

❏ specifies a region of code that must be executed by only **one** thread at a time!

❏ can be named

```
#pragma omp parallel private(loc_sum)
{
  ...
  #pragma omp for
  for(int i = 0; i < n; i++)
    loc_sum += x[i];

  #pragma omp critical (cr_sum)
  sum += loc_sum;
}  /* end of parallel section */
```

OpenMP Syntax

Synchronization – Atomic construct:

❏ specifies a single operation(!) that must be executed by only **one** thread at a time!

❏ restricted syntax (see OpenMP standard)

```
int x = 0;

#pragma omp parallel shared(x)
{
  ...
  #pragma omp atomic
  x = x + 1;
  ...
}  /* end of parallel section */
```

OpenMP Syntax

Programming OpenMP

Synchronization – Master region:

- ❏ specifies a region of code that is executed by the master thread only!
- ❏ ignored by others – no implicit barriers!

```
#pragma omp parallel
{
   ...
   #pragma omp master
   {
      printf("Hello\n");
   }
   ...
}  /* end of parallel section */
```

DTU

OpenMP Syntax

Synchronization – Ordered:

- ❏ executes code block in the sequential order
- ❏ everything outside can run in parallel
- ❏ only within a parallel do/for loop

```
#pragma omp parallel for \
        private(i,myf) ordered
for(i = 0; i < n; i++) {
   myf = i+1;
   #pragma omp ordered
   {
      sum = sum + myf;
   }
}  /* end of parallel for */
```

```
#pragma omp parallel for \
        private(i,myf)
for(i = 0; i < n; i++) {
   myf = i+1;
   #pragma omp critical
   {
      sum = sum + myf;
   }
} /*end of parallel for*/
```

DTU

OpenMP Syntax

Output:

non-ordered   vs.   ordered

```
sum[  1] =          1      sum[  1] =          1
sum[  6] =          7      sum[  2] =          3
sum[  2] =          9      sum[  3] =          6
sum[  7] =         16      sum[  4] =         10
sum[  3] =         19      sum[  5] =         15
sum[  8] =         27      sum[  6] =         21
sum[  4] =         31      sum[  7] =         28
sum[  9] =         40      sum[  8] =         36
sum[ 10] =         50      sum[  9] =         45
sum[  5] =         55      sum[ 10] =         55
Result: 55                 Result: 55
```

OpenMP Syntax

Ordered with dependences (OpenMP 4.5)

❏ new clause depend(...) enhances ordered by adding dependences

  ❏ #pragma omp ordered depend(sink: vec)

    ❏ start the ordered region with the dependences given in vec

  ❏ #pragma omp ordered depend(source)

    ❏ ends the ordered execution defined by sink above

❏ this new feature allows to parallelize e.g. loops with dependences (see next slide)

OpenMP Syntax

doacross loops:

❏ loops with dependencies

# of dependences

```
#pragma omp parallel
{
    #pragma omp for ordered(1)
    for (int i=1; i<n; i++) {

        #pragma omp ordered depend(sink:i-1)
        a[i] = a[i-1] + b[i];
        #pragma omp ordered depend(source)

        c[i] = 2*a[i];
    } // End of for-loop
} // End of parallel region
```

dependence vector

OpenMP Syntax

ordered depend(...) and doacross loops:

❏ provides logic in the OpenMP runtime, that else would have to be done manually

❏ helps to "resolve" static dependences and allows part of the code to run in parallel

❏ useful, if the parallel part of the loop(s) is more computational intensive than the ordered part

❏ hint: might need a different loop scheduling

OpenMP Syntax

Synchronization – Barrier:

❑ synchronizes all threads in a team

```
#pragma omp parallel
{
  ...

  #pragma omp barrier

  ...
}  /* end of parallel section */
```

OpenMP Syntax

Synchronization – Implied barriers:

❑ exit from parallel region

❑ exit from `omp for/omp do/omp workshare`

❑ exit from sections

❑ exit from single

No *implied* barrier on the master construct, neither on entry nor on exit!

OpenMP Syntax

OpenMP programming in C/C++
Part II: Clauses

OpenMP Syntax

Data scoping clauses:

❑ Understanding and the use of data scoping is really essential.

❑ Most problems/errors are due to wrong data scoping.

❑ Most variables are shared by default (shared memory programming model).

❑ Private variables: loop indices, stack of subroutines.

OpenMP Syntax

OpenMP Data scope attribute clauses:

❑ private

❑ shared

❑ default

❑ reduction

❑ firstprivate

❑ lastprivate

❑ copyin

OpenMP Syntax

The "private" clause:

❑ declares variables private to each thread:

`#pragma omp directive private (list)`

❑ a new variable is declared once for each thread

❑ all references are replaced with references to the newly declared variable

❑ variables declared private are uninitialized for each thread!

OpenMP Syntax

The "shared" clause:

❑ declares variables to be shared among all threads:

```
#pragma omp directive shared (list)
```

❑ a shared variable exists in only one memory location and all threads have read/write access to that address

❑ proper access to the variable is left to the programmer – that's YOU!

OpenMP Syntax

The "default" clause:

❑ allows the programmer to specify the default scope for all variables:

```
#pragma omp dir default(shared|none)
```

❑ C/C++ knows only those two types

❑ only one default clause per parallel region

❑ Best practice: use default(none) and scope all your variables explicitly

OpenMP Example
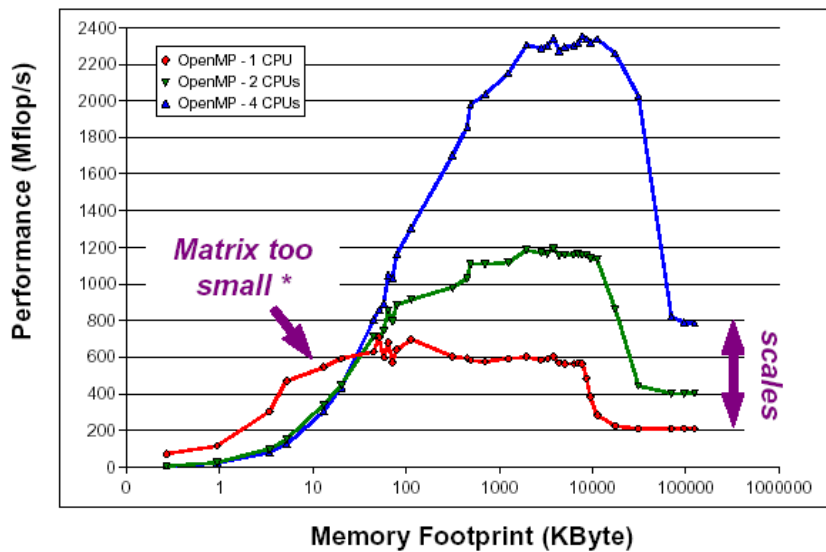
Two examples

OpenMP Example

Matrix times vector

```
void
mxv(int m, int n, double *a, double *b, double *c)
{
  int i, j;
  double sum;

  #pragma omp parallel for default(none) \
              shared(m,n,a,b,c) private(i,j,sum)
  for (i=0; i<m; i++) {
    sum = 0.0;
    for (j=0; j<n; j++)
      sum += b[i*n+j] * c[j];
    a[i] = sum;
  }
}
```

OpenMP Example

Programming OpenMP

SunFire 6800
UltraSPARC III Cu @ 900 MHz
8 MB L2-cache

*) With the IF-clause in OpenMP this performance degradation can be avoided

---

OpenMP Example

Programming OpenMP

Example: numerical integration of f(x)

```
int i, n;
double h, x, sum;

h = 1.0 / (double)n;
sum = 0.0;
#pragma omp parallel for default(none) \
        shared(n,h,sum) private(i,x)

for(i=1; i<=n; i++) {
    x = h * ((double)i + 0.5);
    #pragma omp critical
    sum += f(x);
}
```

sequential code!

Race condition!

OpenMP Example

Example: numerical integration of f(x)

❑ Improvement 1

```
int i, n;
double h, x, fx, sum;

h = 1.0 / (double)n;
sum = 0.0;

#pragma omp parallel for default(none) \
        shared(n,h,sum) private(i,x,fx)
for(i=1; i<=n; i++) {
    x = h * ((double)i + 0.5);
    fx = f(x);          ⬅  function evalution in parallel
#pragma omp critical
    sum += fx;
}
```

OpenMP Example

Example: numerical integration of f(x)

❑ Improvement 2

```
int i, n; double h, x, t_sum, sum;

h = 1.0 / (double)n; sum = 0.0;
#pragma omp parallel default(none) \
        shared(n,h,sum) private(i,x,t_sum)     {
  t_sum = 0.0;
  #pragma omp for
  for(i=1; i<=n; i++) {
      x = h * ((double)i + 0.5);
      t_sum += f(x);
  }
  #pragma omp critical
  sum += t_sum;
} // end omp parallel
```

OpenMP Syntax

The "reduction" clause:

❏ performs a reduction on the variables that appear on the list:

```
#pragma omp dir reduction(op: list)
```

❏ a private copy for each thread of all variables on the list is created

❏ at the end, the reduction operation is carried out and the result(s) written to the global variable(s)

OpenMP Example

Example: numerical integration of f(x)

❏ smart OpenMP solution

```
int i, n;
double h, x, sum;

h = 1.0 / (double)n;
sum = 0.0;

#pragma omp parallel for default(none) \
        shared(n,h) private(i,x) \
        reduction(+: sum)
for(i=1; i<=n; i++) {
    x = h * ((double)i + 0.5);
    sum += f(x);
}
```
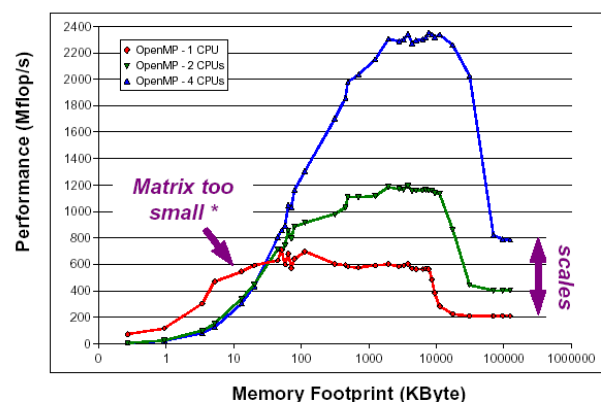
OpenMP Excercises – I

❏ Write an OpenMP code to calculate π, using

$$\pi = \int_0^1 \frac{4}{(1+x^2)} dx \approx \frac{1}{N} \sum_{i=1}^{N} \frac{4}{1+(\frac{i-0.5}{N})^2}$$

❏ implement the integrand as a function

❏ write your own reduction code

❏ use the OpenMP reduction clause

❏ compare the run-times

OpenMP Excercises – II

❏ Improve the matrix times vector example by adding an if-clause to the omp pragma – experiment with the threshold value!

SunFire 6800
UltraSPARC III Cu @ 900 MHz
8 MB L2-cache

*) With the IF-clause in OpenMP this performance degradation can be avoided