The OpenMP Common Core

A journey back to the roots of OpenMP

Tim Mattson
Intel Corp.
timothy.g.mattson@intel.com

Tutorial materials are available here: https://github.com/uob-hpc/openmp-tutorial
The Growing OpenMP Specification

- OpenMP started out in 1997 as a simple interface for the application programmers more versed in their area of science than computer science.
The roots of OpenMP: A slide from my SC2003 OpenMP tutorial

OpenMP Overview:

- C$OMP FLUSH
- C$OMP THREADPRIVATE (/ABC/)
- CALL C$OMP SET NUM THREADS(10)
- Nthrds = OMP_GET_NUM_PROCS()
- omp_set_lock(lck)

OpenMP: An API for Writing Multithreaded Applications

- A set of compiler directives and library routines for parallel application programmers
- Makes it easy to create multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 15 years of SMP practice
The roots of OpenMP: A slide from my SC2003 OpenMP tutorial

OpenMP Overview:

- C$OMP FLUSH
- #pragma omp critical
- C$OMP THREADPRIVATE(/ABC/)
- CALL OMP_SET_NUM_THREADS(10)

OpenMP: An API for Writing
Multithreaded Applications
(= applications that use multiple threads)

- A set of compiler directives and library routines for parallel application programmers
- Makes it easy to create multi-threaded (MT) programs in Fortran, C and C++
- Standard for parallel programming practice

Nthrds = OMP_GET_NUM_PROCS()
omp_set_lock(lck)

Our primary goal was to make parallel computing easy
The Growing OpenMP Specification

- Rewrote the specification to merge languages … quality of the specification improved dramatically! It became a spec we could be really proud of.

Page counts (not counting front matter, appendices or index) for versions of OpenMP

- Fortran spec
- C/C++ spec
- Merged C/C++ and Fortran spec
- OpenMP Spec plus the tools/debug API

Merged C and Fortran specs
The Growing OpenMP Specification

- SPMD patterns worked well but for most programmers, it was all about parallel loops
OpenMP Basic Definitions: Basic Solution Stack

- **End User**
- **Application**
- **Directives, Compiler**
  - **OpenMP library**
  - **Environment variables**
- **OpenMP Runtime library**
- **OS/system support for shared memory and threading**
- **Shared address space (SMP)**
The Growing OpenMP Specification

- Tasks greatly expanded the scope of OpenMP.
- Required a major rewrite of the spec to express semantics in terms of tasks.

Tasks added to OpenMP ... supports irregular parallelism
The Growing OpenMP Specification

- OpenMP started out in 1997 as a simple interface for the application programmers more versed in their area of science than computer science.

- The complexity has grown considerably over the years!
OpenMP Basic Definitions: Basic Solution Stack

End User

Application

Directives, Compiler
OpenMP library
Environment variables

OpenMP Runtime library

OS/system support for shared memory and threading

Program layer

System layer

Hardware

Shared address space (SMP)

SIMD units
The Growing OpenMP Specification

- We knew the world was more complex than SMP when we started …. With 4.0 we finally embraced this complexity in OpenMP

![Graph showing page counts for versions of OpenMP](image-url)

Target constructs and NUMA added to OpenMP

Page counts (not counting front matter, appendices or index) for versions of OpenMP

<table>
<thead>
<tr>
<th>Year</th>
<th>1.0</th>
<th>1.1</th>
<th>2.0</th>
<th>2.0</th>
<th>2.5</th>
<th>3.0</th>
<th>3.1</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>1.0</td>
<td>1.0</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OpenMP basic definitions: Basic Solution stack

- **User layer**
  - End User
  - Application

- **Prog.**
  - Directives, Compiler
  - OpenMP library
  - Environment variables

- **System layer**
  - OpenMP Runtime library
  - OS/system support for shared memory and threading

- **HW**
  - Shared address space (NUMA)
  - CPU cores
  - SIMD units
  - GPU cores
The Growing OpenMP Specification

- And with 5.0, the complexity accelerates!!!!
With a 600+ page spec, we risk scaring people away from OpenMP

• Can we make OpenMP less scary?

• What if we just focused on the subset of OpenMP that people actually use?

• … which begs the question … How do people most use OpenMP?
An Interesting Problem to Play With
Numerical Integration

Mathematically, we know that:

\[
\int_{0}^{1} \frac{4.0}{1+x^2} \, dx = \pi
\]

We can approximate the integral as a sum of rectangles:

\[
\sum_{i=0}^{N} F(x_i) \Delta x \approx \pi
\]

Where each rectangle has width \( \Delta x \) and height \( F(x_i) \) at the middle of interval \( i \).
static long num_steps = 100000;
double step;
int main ()
{
    int i; double x, pi, sum = 0.0;

    step = 1.0/(double) num_steps;

    for (i=0;i< num_steps; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }
    pi = step * sum;
}

See C/pi.c or Fortran/pi.f
The SPMD Pi program (C)

```c
#include <omp.h>
#define MAX_THREADS 4
static long num_steps = 100000000;
int main ()
{
    int i,j;
    double step, pi, full_sum = 0.0, start_time, run_time, sum[MAX_THREADS];
    step = 1.0/(double) num_steps;
    #pragma omp set_num_threads(MAX_THREADS);
    start_time = omp_get_wtime();
    #pragma omp parallel
    {
        int i;
        int id = omp_get_thread_num();
        int numthreads = omp_get_num_threads();

        for (i=id,sum[id]=0;i< num_steps; i+=numthreads){
            double x = (i+0.5)*step;
            sum[id] = sum[id] + 4.0/(1.0+x*x);
        }
    }
    for(full_sum = 0.0, i=0;i<j;i++)
        pi += sum[i]*step;
    run_time = omp_get_wtime() - start_time;
}
```
The OpenMP Common Core:

#pragma omp parallel
void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()
setenv OMP_NUM_THREADS N

The fundamental building blocks of multithreading
The SPMD Pi program (Fortran)

PROGRAM MAIN
USE OMP_LIB
INTEGER i, j, id, numthreads, nthreads
INTEGER, PARAMETER :: num_steps=100000000
INTEGER, PARAMETER :: MAX_THREADS=4
REAL*8 pi, real_sum, step, full_sum, x, start_time, run_time,sum(0:MAX_THREADS-1)
step = 1.0/num_steps
start_time = OMP_GET_WTIME()
CALL OMP_SET_NUM_THREADS(j)
start_time = omp_get_wtime()

!$OMP PARALLEL PRIVATE(id,x,numthreads)
   id = omp_get_thread_num()
   numthreads = OMP_GET_NUM_THREADS()
   sum(id) = 0.0
   DO i = id, num_steps-1, numthreads
      x = (i+0.5)*step
      sum(id) = sum(id) + 4.0/(1.0+x*x)
   ENDDO
!$OMP END PARALLEL
pi = 0.0
DO i = 0, nthreads-1
   full_sum = full_sum + sum(i)*step
ENDDO
run_time = OMP_GET_WTIME() - start_time
STOP
END
The OpenMP Common Core:

`#pragma omp parallel`

`void omp_set_thread_num()`

`int omp_get_thread_num()`

`int omp_get_num_threads()`

`double omp_get_wtime()`

`setenv OMP_NUM_THREADS N`

shared(list), private(list), firstprivate(list)

default(none)
The SPMD Pi program (eliminate false sharing)

```c
#include <omp.h>
#define MAX_THREADS 4
static long num_steps = 100000000;
int main ()
{
    int i,j;
    double step, pi=0.0, full_sum = 0.0, start_time, run_time;
    step = 1.0/(double) num_steps;
    omp_set_num_threads(MAX_THREADS);
    start_time = omp_get_wtime();
    #pragma omp parallel
    {
        int i;
        int id = omp_get_thread_num();
        int numthreads = omp_get_num_threads();
        double sum=0.0;
        for (i=id;i< num_steps; i+=numthreads){
            double x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
        #pragma omp critical
        pi += sum*step
    }
    run_time = omp_get_wtime() - start_time;
}
```
The OpenMP Common Core:

```c
#pragma omp parallel
void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()
setenv OMP_NUM_THREADS N

shared(list), private(list), firstprivate(list)
default(none)
```

#pragma omp barrier
#pragma omp critical

Synchronization to define ordering constraints and prevent data races
Parallel loop pi … the simple case

```c
#include <omp.h>
static long num_steps = 100000; double step;
void main ()
{
    int i; double x, pi, sum = 0.0;
    step = 1.0/(double) num_steps;
    #pragma omp parallel for reduction(+:sum)
        for (i=0;i< num_steps; i++){
            double x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
    pi = step * sum;
} 
```
The OpenMP Common Core:

```c
#pragma omp parallel
void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()
setenv OMP_NUM_THREADS N

shared(list), private(list), firstprivate(list)
default(none)

#pragma omp barrier
#pragma omp critical
```

Parallel Loops … the essence or OpenMP Programming Practice
Program: OpenMP Tasks

```c
#include <omp.h>

static long num_steps = 100000000;
#define MIN_BLK 10000000

double pi_comp(int Nstart, int Nfinish, double step)
{
    int i,iblk;
    double x, sum = 0.0,sum1, sum2;
    if (Nfinish-Nstart < MIN_BLK){
        #pragma omp for private(x) reduction(+:sum)
        for (i=Nstart;i< Nfinish; i++){
            x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
    } else{
        iblk = Nfinish-Nstart;
        #pragma omp task shared(sum1)
        sum1 = pi_comp(Nstart, Nfinish-iblk/2,step);
        #pragma omp task shared(sum2)
        sum2 = pi_comp(Nfinish-iblk/2, Nfinish, step);
        #pragma omp taskwait
        sum = sum1 + sum2;
    }
    return sum;
}

int main ()
{
    int i;
    double step, pi, sum;
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        #pragma omp single
        sum = pi_comp(0,num_steps,step);
    }
    pi = step * sum;
}
```
The OpenMP Common Core:

```c
#pragma omp parallel
void omp_set_thread_num()
int omp_get_thread_num()
int omp_get_num_threads()
double omp_get_wtime()
setenv OMP_NUM_THREADS N

shared(list), private(list), firstprivate(list)
default(none)

#pragma omp barrier
#pragma omp critical

#pragma omp for
#pragma omp parallel for reduction(op:list)

#pragma omp single
#pragma omp task
#pragma omp taskwait

Generalizing OpenMP to irregular parallelism
```
double A[big], B[big], C[big];

#pragma omp parallel
{
    int id=omp_get_thread_num();
    A[id] = big_calc1(id);
#pragma omp barrier
#pragma omp for schedule(static, 10)
    for(i=0;i<N;i++) {C[i]=big_calc3(i, A);}
#pragma omp for schedule(dynamic) nowait
    for(i=0;i<N;i++) {B[i]=big_calc2(C, i); }
    A[id] = big_calc4(id);
}
### The OpenMP Common Core: An elegant API from a more civilized age

The 21 constructs most OpenMP programmers use exclusively

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#pragma omp parallel</code></td>
<td>Enables OpenMP parallelism</td>
</tr>
<tr>
<td><code>void omp_set_thread_num()</code></td>
<td>Sets the number of threads</td>
</tr>
<tr>
<td><code>int omp_get_thread_num()</code></td>
<td>Gets the thread number</td>
</tr>
<tr>
<td><code>int omp_get_num_threads()</code></td>
<td>Gets the number of threads</td>
</tr>
<tr>
<td><code>double omp_get_wtime()</code></td>
<td>Gets the wall time</td>
</tr>
<tr>
<td><code>setenv OMP_NUM_THREADS N</code></td>
<td>Sets the number of threads</td>
</tr>
<tr>
<td><code>#pragma omp for</code></td>
<td>For loop construct</td>
</tr>
<tr>
<td><code>#pragma omp parallel for</code></td>
<td>Parallel for loop construct</td>
</tr>
<tr>
<td><code>#pragma omp reduction(op:list)</code></td>
<td>Reduction operation</td>
</tr>
<tr>
<td><code>schedule (static [,chunk])</code></td>
<td>Schedule static</td>
</tr>
<tr>
<td><code>schedule(dynamic [,chunk])</code></td>
<td>Schedule dynamic</td>
</tr>
<tr>
<td><code>shared(list)</code></td>
<td>Shared data</td>
</tr>
<tr>
<td><code>private(list)</code></td>
<td>Private data</td>
</tr>
<tr>
<td><code>firstprivate(list)</code></td>
<td>First private data</td>
</tr>
<tr>
<td><code>default(none)</code></td>
<td>Default access</td>
</tr>
<tr>
<td><code>#pragma omp single</code></td>
<td>Single construct</td>
</tr>
<tr>
<td><code>#pragma omp task</code></td>
<td>Task construct</td>
</tr>
<tr>
<td><code>#pragma omp taskwait</code></td>
<td>Task wait construct</td>
</tr>
<tr>
<td><code>#pragma omp barrier</code></td>
<td>Barrier construct</td>
</tr>
<tr>
<td><code>#pragma omp critical</code></td>
<td>Critical section</td>
</tr>
</tbody>
</table>

The OpenMP Common Core: An elegant API from a more civilized age
The OpenMP Common Core

• People should master the common core, and then pick up what the need from the rest of OpenMP “as needed”

• We have a new book that introduces the Common Core:
  – Part 1: Intro to parallel programming.
  – Part 2: the OpenMP common core
  – Part 3: Beyond the common core

• The book is geared at people new to OpenMP and OpenMP educators …. Though the memory model chapter in part 3 will be beneficial for more advanced programmers.
<table>
<thead>
<tr>
<th>OpenMP pragma, function, or clause</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma omp parallel</td>
<td>Parallel region, teams of threads, structured block, interleaved execution across threads.</td>
</tr>
<tr>
<td>void omp_set_thread_num()</td>
<td>Default number of threads and internal control variables. SPMD pattern: Create threads with a parallel region and split up the work using the number of threads and the thread ID.</td>
</tr>
<tr>
<td>void omp_get_thread_num()</td>
<td>Speedup and Amdahl's law. False sharing and other performance issues.</td>
</tr>
<tr>
<td>int omp_get_thread_num()</td>
<td></td>
</tr>
<tr>
<td>int omp_get_num_threads()</td>
<td></td>
</tr>
<tr>
<td>double omp_get_wtime()</td>
<td></td>
</tr>
<tr>
<td>setenv OMP_NUM_THREADS N</td>
<td>Setting the internal control variable for the default number of threads with an environment variable</td>
</tr>
<tr>
<td>#pragma omp barrier</td>
<td>Synchronization and race conditions. Revisit interleaved execution.</td>
</tr>
<tr>
<td>#pragma omp critical</td>
<td></td>
</tr>
<tr>
<td>#pragma omp for</td>
<td>Worksharing, parallel loops, loop carried dependencies.</td>
</tr>
<tr>
<td>#pragma omp parallel for</td>
<td></td>
</tr>
<tr>
<td>reduction(op:list)</td>
<td>Reductions of values across a team of threads.</td>
</tr>
<tr>
<td>schedule (static [,chunk])</td>
<td>Loop schedules, loop overheads, and load balance.</td>
</tr>
<tr>
<td>schedule(dynamic [,chunk])</td>
<td></td>
</tr>
<tr>
<td>shared(list), private(list), firstprivate(list)</td>
<td>Data environment.</td>
</tr>
<tr>
<td>default(none)</td>
<td>Force explicit definition of each variable’s storage attribute</td>
</tr>
<tr>
<td>nowait</td>
<td>Disabling implied barriers on workshare constructs, the high cost of barriers, and the flush concept (but not the flush directive).</td>
</tr>
<tr>
<td>#pragma omp single</td>
<td>Workshare with a single thread.</td>
</tr>
<tr>
<td>#pragma omp task</td>
<td>Tasks including the data environment for tasks.</td>
</tr>
<tr>
<td>#pragma omp taskwait</td>
<td></td>
</tr>
</tbody>
</table>