OpenMP

Bedrich Benes, Ph.D.
Professor
Department of Computer Graphics
Purdue University

Open Multi Processing

- OpenMP is an API
- For shared memory multiprocessing
- C, C++, Fortran
- Solaris, Linux, macOS, and Windows
- Compile directives, libraries, env. variables

OpenMP Language Extensions

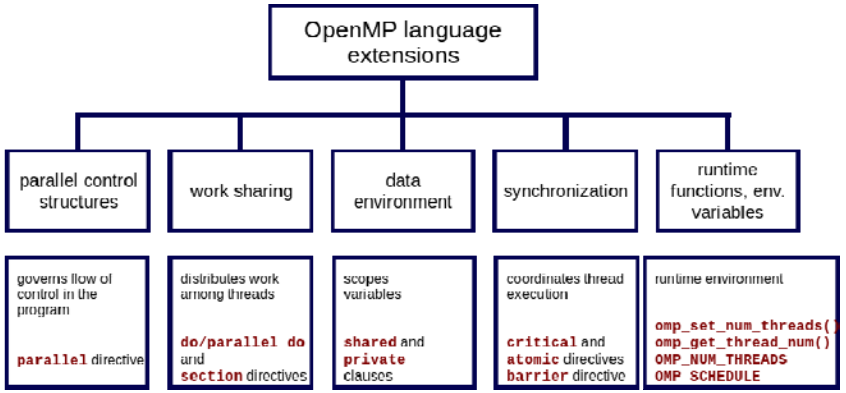


Image Courtesy Wikipedia

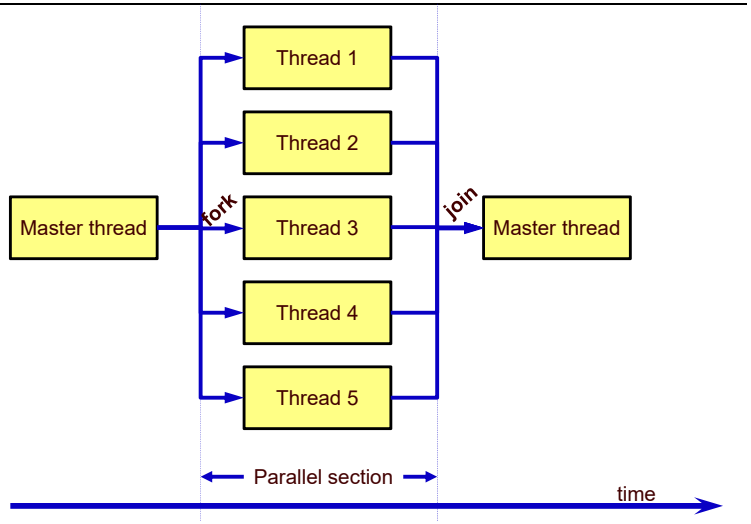
Multithreading

- OpenMP implements multithreading
- **Thread is a set of instructions executed in serial**
- Uses Fork and Join

Thread Execution

- The application starts with ONE master thread (ID 0)
- The execution is sequential until a first parallel region is reached (created by the programmer)
- This executes a *team of parallel threads*
- Runs in parallel until join
- Then again only the master thread (ID 0)

Multithreading



Parallel Section

- Parallel section is a block of code that is executed in parallel by all threads
- The master thread has ID 0
- Can be nested (more sections in one)

Multithreading

- User specifies the number of threads
- Uses shared memory model threads read/write into the same memory
- A variable can be visible to all threads
- Synchronization avoids ***race conditions***

General structure

```
#include <stdio.h>
#include <omp.h>
int main (void){
 int a,b,c;
 //Serial part
 #pragma omp parallel private(a,b) shared(c)
 { //FORK
   //Parallel part
 } //JOIN
 //Serial part again
}
```


#pragma omp parallel

```
#pragma omp parallel
structured_block
```

- Structured block is executed in parallel on all available threads

Structured block

```
#pragma omp parallel [clause ...]
{
 structured_block
}
```

- When a thread reaches this it will
  - Creates threads and becomes a master
  - The master is one of them with a ID 0
  - The code is duplicated to all threads
  - There is a barrier at the end (join)
  - Only the master continues after join

omp parallel

- Parallel constructs
- *all within the* **`structured_block`** *is executed by multiple threads*
- This is the most important building block…

Multithreading “hello world”

```
#include <stdio.h>
int main (void){
  #pragma omp parallel
  printf(“hello world\n”);
  return 0;
}
```


Multithreading “hello world”

```
bbenes@phi581:~/Desktop/C++
[bbenes@phi581 C++]$ g++ main.cpp -o main
[bbenes@phi581 C++]$ ./main
Hello world
[bbenes@phi581 C++]$
```


Multithreading “hello world”

- What is wrong? Only one thread?

- Tell the compiler to use OpenMP
- Use g++ directive **`-fopenmp`**

```
bbenes@phi581:~/Desktop/C++
[bbenes@phi581 C++]$ g++ -fopenmp main.cpp -o main
[bbenes@phi581 C++]$
```


Multithreading “hello world”

```
bbenes@phi581:~/Desktop/C++
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
[bbenes@phi581 C++]$
```

Multithreading “hello world”

- How many threads?
- 256 on Xeon Phi Kings Landing

```
bbenes@phi581:~/Desktop/C++
[bbenes@phi581 C++]$ ./main | wc -l
256
[bbenes@phi581 C++]$
```


Number of Threads

- Controlled by environmental variables
  (bash)

```
$ export OMP_NUM_THREADS=2
```

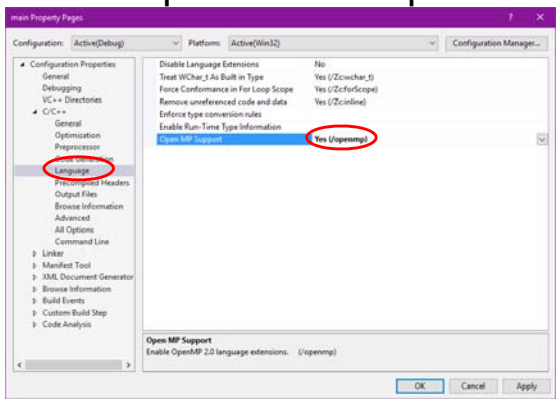
  (csh/tcsh)

```
$ setenv OMP_NUM_THREADS 2
```

```
bbenes@phi581:~/Desktop/C++
[bbenes@phi581 C++]$ export OMP_NUM_THREADS=2
[bbenes@phi581 C++]$ ./main
Hello world
Hello world
[bbenes@phi581 C++]$
```


Multithreading “hello world”

- The same in Visual Studio on Windows
- Tell the compiler to use OpenMP

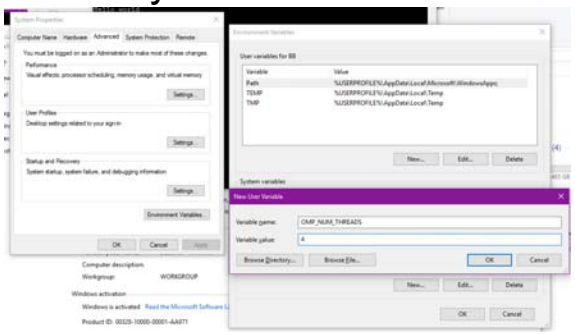


Multithreading “hello world”

```
C:\Users\BB\Desktop\openmp\Debug\main.exe
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
```

Number of Threads

- Can be controlled by environmental variables



Number of Threads

- Can be set in runtime

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

int main(void) {
 omp_set_num_threads(2);
 #pragma omp parallel
 printf("Hello world \n");
 getchar();
 return 0;
}
```


Dynamic Threads

- Dynamic Threads:
- Default: the same # of threads for each parallel region
- Two ways enable dynamic threads:

1) Use **`omp_set_dynamic()`** library function
2) **`OMP_DYNAMIC`** environment variable

OpenMP directives

`#pragma omp name [clause,…]`

- **`#pragma omp`**
  - Required for all OpenMP C/C++ directives.
- **`name`**
  - After the pragma and before any clauses.
- **`[clause, ...]`**
  - Optional.
  - Any order, repeated, unless restricted.

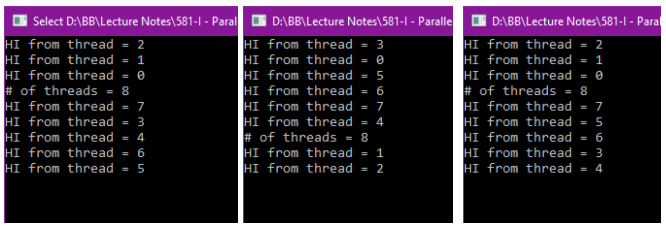
omp parallel

```
#pragma omp [clause ...] newline
   if (scalar_expression)
   private (list)
   shared (list)
   default (shared | none)
   firstprivate (list)
   reduction (operator: list)
   copyin (list)
structured_block
```


Getting the thread ID

```
#include <omp.h>
int main () {
int nthreads, tid;
 #pragma omp parallel private(tid)  {
 tid = omp_get_thread_num(); /* thread id */
  printf("HI from thread = %d\n", tid);
  /* Only master thread does this */
  if (tid == 0) {
    nthreads = omp_get_num_threads();
    printf("# of threads = %d\n", nthreads);
    }
  }  /*join master thread 0 and stop */
 }
```


Getting the thread ID



omp parallel example

```
void Sum(int *a, int *b, int *c, long int n) {
long int i;
int nthreads, tid;
#pragma omp parallel{ //get number of threads
   if (omp_get_thread_num() == 0)
            nthreads = omp_get_num_threads();
}

#pragma omp parallel private(tid, i){
   tid = omp_get_thread_num();//get thread #
   for (i = 0 + tid; i < n; i += nthreads)
                     c[i] = a[i] + b[i];
   }
}
```

structured block

structured block

omp parallel example

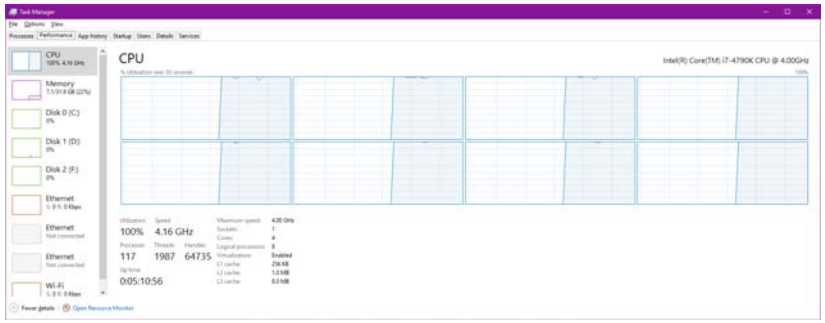
- OpenMP will
  - use all available threads
  - forks before it enters the loop,
  - divides the work, and
  - joins at the end.

omp parallel example

```
bbenes@phi581:~/Desktop/sum
top - 14:36:56 up 2 days,  4:04,  3 users,  load average: 0.29, 0.08, 0.07
Tasks: 2434 total,   2 running, 2432 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.0 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 11519324+total, 10949587+free,  3367844 used,  2329520 buff/cache
KiB Swap: 20479996 total, 20479996 free,        0 used. 11092878+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
124319 bbenes    20   0 1589856 1.495g    936 R 100.0  1.4   0:26.49 main
124320 bbenes    20   0  160056   4716   1556 R   0.6  0.0   0:02.30 top
124137 bbenes    20   0  149788   2808   1220 S   0.6  0.0   0:00.64 sshd
124255 bbenes    20   0  629284  31208  15212 S   0.6  0.0   0:02.46 gedit
   265 root      20   0       0      0      0 S   0.3  0.0   4:04.54 rcu_sched
   355 root      20   0       0      0      0 S   0.3  0.0   0:00.54 rcuos/89
  1233 root      rt   0       0      0      0 S   0.3  0.0   0:01.42 watchdog/+
  3251 root      20   0   19368   1416    964 S   0.3  0.0   4:32.22 irqbalance
     1 root      20   0  208044  21112   3944 S   0.0  0.0   0:59.84 systemd
     2 root      20   0       0      0      0 S   0.0  0.0   0:00.29 kthreadd
     3 root      20   0       0      0      0 S   0.0  0.0   0:00.02 ksoftirqd+
     4 root      20   0       0      0      0 S   0.0  0.0   0:00.86 kworker/0+
     5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0+
     7 root      rt   0       0      0      0 S   0.0  0.0   0:00.92 migration+
     8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
     9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/0
    10 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/1
```


omp parallel example



omp parallel example

- for a small array (24 elements) and 8 thread we can get:

```
# of threads = 8
Thread 2 working on 2
Thread 2 working on 10
Thread 2 working on 18
Thread 3 working on 3
Thread 3 working on 11
Thread 1 working on 1
Thread 1 working on 9
Thread 5 working on 5
Thread 5 working on 13
Thread 5 working on 21
Thread 4 working on 4
Thread 4 working on 12
Thread 4 working on 20
Thread 1 working on 17
Thread 0 working on 0
Thread 0 working on 8
Thread 0 working on 16
Thread 7 working on 7
Thread 7 working on 15
Thread 7 working on 23
Thread 3 working on 19
Thread 6 working on 6
Thread 6 working on 14
Thread 6 working on 22
```

#pragma omp for

```
#pragma omp for
for (…)
```

```
#pragma omp parallel
#pragma omp for
for (i=0;i<n;i++){
    heavy_lifting(i);
}
```

- The “for” splits up loop iterations among the threads in a team

#pragma omp master

```
#pragma omp master
{structured block}
```

```
#pragma omp parallel private (tmp){
 parallel1();
 #pragma omp master
 {only_master();}//executed by ID0, others skip
 #pragma barrier
 parallel2();
}
```


if/shared/private

- **`if (scalar)`**
  - only executed in parallel if (true)
  - otherwise serial
- **`private (list)`**
  - not associated with the original
  - all references to the local object
  - undefined values on entry and exit
- **`shared (list)`**
  - all accessible by all threads
  - all in the same address

if/shared/private - example

```
#pragma omp parallel if (n > max) \
                shared(n,x,y) private(i)
{
 #pragma omp for
 for (i=0; i<n;i++)
   x[i]=y[i]*i;
}
```

reduction

```
#pragma omp reduction(operator:list)
```

- a private copy for each list variable is created for each thread
- at the end of the reduction, the reduction variable is applied to all private copies of the shared variable
- the final result is written to the global shared variable

reduction example

```
#include <omp.h>
 int   i;
 result = 0.0;
 Init(a, b, n);
#pragma omp parallel for default(shared)
private(i)  reduction(+:result)
   for (i = 0; i < n; i++)
        result+=a[i] * b[i];
   printf("Result= %f\n", result);
   getchar();
}
```


Barrier

```
#pragma omp barrier
```

- waits until all thread reach this point
- end of a parallel section includes one

Atomic update

```
#pragma omp atomic
```

- only one thread can enter at the time
- Temporal serialization

Environmental variables

- **OMP_NUM_THREADS** (int)
  maximum number of threads
- **OMP_SCHEDULE** (chunk)
  how iterations are scheduled when a schedule clause is set to “runtime”
- **OMP_DYNAMIC** (bool)
  dynamic adjustment of threads for parallel regions
- **OMP_NESTED** (bool)
  nested parallelism true or false

Reading

www.openmp.org