

COMPUTATIONAL MATH, SCIENCE AND ENGINEERING DEPARTMENT

MICHIGAN STATE
UNIVERSITY

OpenMP

```

5312 0000488B 15470B00 004889D6 4889C7E8 6E050000 488B1535 0B000048 89D64889 C7E85C05 0000488D 35790600 00488B05 140B0000 4889C7E8 7C050000 BE080000
5376 004889C7 E85D0500 00488B15 000B0000 4889D648 89C7E827 050000E8 10050000 4889C348 8D354A06 0000488B 05070A00 004889C7 E83F0500 004889D6 4889C7E8
5440 1C050000 488B15C5 0A000048 89D64889 C7E8EC04 0000E8CF 04000048 89C3488D 351E0600 00488B05 9C0A0000 4889C7E8 04050000 4889D648 89C7E8E1 04000048
5504 8B158A0A 00004889 D64889C7 E8B10400 00488B15 780A0000 4889D648 89C7E89F 04000048 8D35E705 0000488B 05570A00 004889C7 E8BF0400 000B0800 00004889
5568 C7E8A004 0000488B 15430A00 004889D6 4889C7E8 6A040000 488B1531 0A000048 89D64889 C7E85804 0000488D 35B70500 00488B05 100A0000 4889C7E8 78040000
5632 BE040000 004889C7 E8590400 00488B15 FC090000 4889D648 89C7E823 040000E8 F4030000 660F7EC3 488D3588 05000048 8B05D209 00004889 C7E83A04 0000660F
5696 6EC34889 C7E8A004 0000488B 15BF0900 004889D6 4889C7E8 E6030000 E8B10300 00660F7E C3488D35 5B050000 488B0595 09000048 89C7E8FD 03000066 0F6EC348
5760 89C7E8CD 03000048 8B158209 00004889 D64889C7 E8A90300 00488D35 36050000 488B0561 09000048 89C7E8C9 030000BE 18000000 4889C7E8 9E030000 488B154D
5824 09000048 89D64889 C7E87403 0000488B 153B0900 004889D6 4889C7E8 62030000 488D3509 05000048 8B051A09 00004889 C7E88203 0000BE08 00000048 89C7E863
5888 03000048 8B150609 00004889 D64889C7 E82D0300 00E8E602 00006648 0F7EC348 C000488B 05D80800 4889C9C7 E8430300 0066480F 6EC34889 C7E80603
5952 0000488B 15C70800 004889D6 4889C7E8 EE020000 E8A10200 0066480F 7EC3488D 35AC0400 00488B05 9C080000 4889C7E8 04030000 66480F6E C34889C7 E8C70200
6016 00488B15 88080000 4889D648 89C7E8AF 02000048 8D358704 0000488B 05670800 004889C7 E8CF0200 00BE3500 00004889 C7E8A402 0000488B 15530800 004889D6
6080 4889C7E8 7A020000 488B1541 08000048 89D64889 C7E86802 0000488D 355B0400 00488B05 20080000 4889C7E8 88020000 BE100000 004889C7 E8690200 00488B15
6144 0C080000 4889C7E8 7A020000 488B1541 08000048 89D64889 C7E86802 0000488D 355B0400 00488B05 20080000 4889C7E8 88020000 BE100000 004889C7 E8690200 00488B15
6208 00004889 D64889C7 E85D0500 00488B15 000B0000 4889D648 89C7E827 050000E8 10050000 4889C348 8D354A06 0000488B 05070A00 004889C7 E83F0500 004889D6
6272 4889D648 89C7E827 050000E8 10050000 4889C348 8D354A06 0000488B 05070A00 004889C7 E83F0500 004889D6 4889C7E8 7C050000 BE080000
6336 00004889 D64889C7 E85D0500 00488B15 000B0000 4889D648 89C7E827 050000E8 10050000 4889C348 8D354A06 0000488B 05070A00 004889C7 E83F0500 004889D6
6400 07000048 8B05F606 00004889 C7E86C01 0000C9C3 554889E5 BEFFF000 00BF0100 0000E8A5 FFFFFF5D C3554889 E5B80080 FFFF5DC3 554889E5 B8FF7F00 005DC355
6464 4889E5B8 00000080 5DC35548 89E5B8FF FFFF7F5D C3554889 E5488B00 00000000 0000805D C3554889 E5488BFF FFFF7F5D C3554889 E58B0589 01000066
6528 4889E548 B8000000 00000000 00B8A010 00004889 45F08955 F8DB6D0F 5DC35548 89E548C7 C0FFFFF7 FFBABE7F 00004889 45F08955 F8DB6D0F 5DC35548 89E548C7 C0FFFFF7
6656 FF252A06 0000FF25 2C060000 FF252E06 0000FF25 30060000 FF253206 0000FF25 34060000 FF253606 0000FF25 38060000 FF253A06 0000FF25 3C060000 FF253E06
6720 0000FF25 40060000 FF254206 0000FF25 44060000 FF254606 0000FF25 48060000 FF254A06 0000FF25 4C060000 FF254E06 0000FF25 50060000 FF255206 0000FF25
6784 54060000 4C8D1D95 05000041 53FF2585 05000090 68000000 00E9E6FF FFFF6819 000000E9 DCF7FFFF 682B0000 00E9D2FF FFFF683D 000000E9 54060000
6848 00E9BEFF FFFF6861 000000E9 B4FFFFF7 68730000 00E9AAFF FFFF6885 000000E9 A0FFFFF7 68970000 00E996FF FFFF68B7 000000E9 8CFFFFF7 68F70000 00E982FF
6912 FFFF0000 00000000 FFFF77F7 FFFF77FF 53697A65 206F6620 73686F72 743A0053 6D616C6C 65737420 73686F72 743A004C 61726765 73742073 686F7274 3A005369
6976 7A05206F 6620696E 743A0053 6D616C6C 65737420 696E743A 006C6172 70697374 3A005369 66206C6F 66E73A00 65737420 686F7274 3A005369
7040 673A004C 61726765 7374206C 6F6E673A 0053697A 65206F66 206C6F6E 67206C6F 6E672069 6E743A00 53697A65 206F6620 666C6F61 743A0053 6D616C6C 65737420
7104 666C6F61 743A004C 61726765 73742066 6C6F6174 3A004469 67697473 20696E20 6D617469 7376612C 20666C6F 61743A00 53697A65 206F6620 646F7562 6C653A00
7168 536D616C 6C657374 20646F75 626C653A 004C6172 67657374 20646F75 626C653A 00446967 69747320 696E206D 61746973 73612C20 646F7562 6C653A00 53697A65
7232 206F6620 6C6F6E67 20646F75 626C653A 00536D61 6C6C6573 74206C6F 6E67206A 6F75626C 653A004C 61726765 7374206C 6F6E6720 646F7562 6C653A00 00000000
7296 44696769 74732069 6E206D61 74697373 612C206C 6F6E6720 646F7562 6C653A00 14000000 00000000 017A5200 01781001 100C0708 90010000 34000000 1C000000
7360 69FCFFFF FFFFFFFF 0B000000 00000000 00040100 00000E10 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 54000000 3CFCFFFF FFFFFFFF
7424 0B000000 00000000 00040100 00000E10 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 00000000 00000000 00000000 00000000
7488 00040100 00000E10 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 C4000000 E2FBFFFF FFFFFFFF 0B000000 00000000 00040100 00000E10
7552 86020403 00000000 06040600 00000C07 08000000 00000000 34000000 FC000000 B5FBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 00000000
7616 06040B00 00000C07 08000000 00000000 34000000 34010000 8DFBFFFF FFFFFFFF 10000000 00000000 00000000 00040100 00000E10 86020403 00000000 06040B00
7680 08000000 00000000 34000000 6C010000 65FBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 00000000 06040B00 00000C07 08000000 00000000
7744 34000000 A4010000 3DFBFFFF FFFFFFFF 10000000 00000000 00040100 00000E10 86020403 00000000 06040B00 00000C07 08000000 00000000 34000000 DC010000
7808 15FBFFFF FFFFFFFF 15000000 00000000 00040100 00000E10 86020403 00000000 06041000 00000C07 08000000 00000000 34000000 14020000 F2FAFFFF FFFFFFFF
7872 15000000 00000000 00040100 00000E10 86020403 00000000 06041000 00000C07 08000000 00000000 CFFAFFFF FFFFFFFF 1F000000 00000000 00000000 00000000
7936 00040100 00000E10 86020403 00000000 06041A00 00000C07 08000000 00000000 34000000 84020000 B6FAFFFF FFFFFFFF 1C000000 00000000 00040100 00000E10
8000 86020403 00000000 06041700 00000C07 08000000 00000000 34000000 BC020000 02FAFFFF FFFFFFFF 6A050000 00000000 00040100 00000E10 86020403 00000000
8064 06040500 00000303 04600500 000C0708 34000000 F4020000 34F9FFFF FFFFFFFF 48000000 00000000 00040100 00000E10 86020403 00000000 06044300 00000C07
8128 08000000 00000000 34000000 2C030000 44F9FFFF FFFFFFFF 15000000 00000000 00040100 00000E10 86020403 00000000 06041000 00000C07 08000000 00000000
8192 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 A8190000 01000000 95190000 01000000 D8190000 01000000
8256 BF190000 01000000 75190000 01000000 85190000 01000000 4A190000 01000000 3F190000 01000000 65190000 01000000 55190000 01000000 34190000 01000000
8320 29190000 01000000 9A1A0000 01000000 9E1A0000 01000000 A81A0000 01000000 B21A0000 01000000 BC1A0000 01000000 C61A0000 01000000 D81A0000 01000000
8384 DA1A0000 01000000 E41A0000 01000000 EE1A0000 01000000 F81A0000 01000000 14190000 01000000 00000000 00000000 00000000 00000000 00000000 00000000
8448 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8512 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8576 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

CMSE 822, FS21, W.F. Punch

The main point

OpenMP (Open Multiprocessing) is an Application Programming Interface (API) that makes it easier to write threaded programs, programs that use shared memory.

Interfaces directly with C/C++ and Fortran



references

Take a look at www.openmp.org

- OpenMP 5.1 is the most recent version (2020)
- Previous to that 4.5, (2015)
 - for Ubuntu 20.04 LTS looks like we get that one.



Consists of

OpenMP consists of

- program directives to control how parallelization will be done
- a library of some simple functions
- some environmental variables



OpenMP, not OpenMPI

You can easily get confused here, but:

- OpenMP is a programming standard for shared memory/thread programming
- OpenMPI is a *particular* implementation of the MPI standard

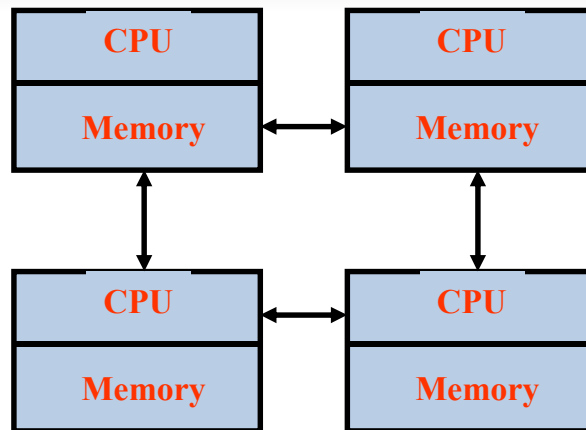


It's a standard

- as a standard it should be portable across OS and compiler
- as a standard might be (is!) implemented differently across systems.
- is pretty compact (though, not surprisingly, growing with time).

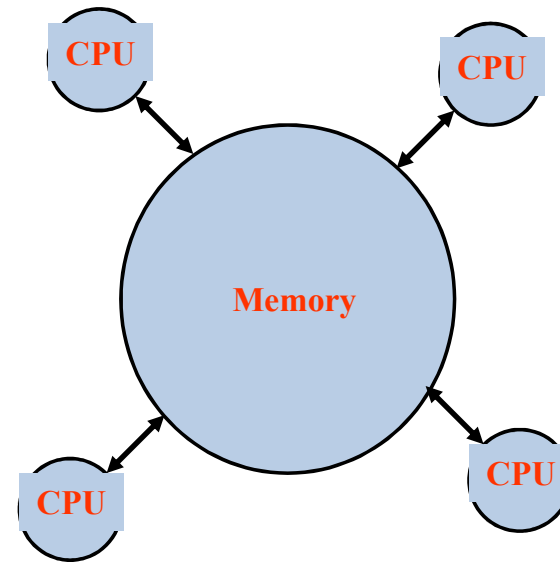


Shared Memory type



Distributed

Shared



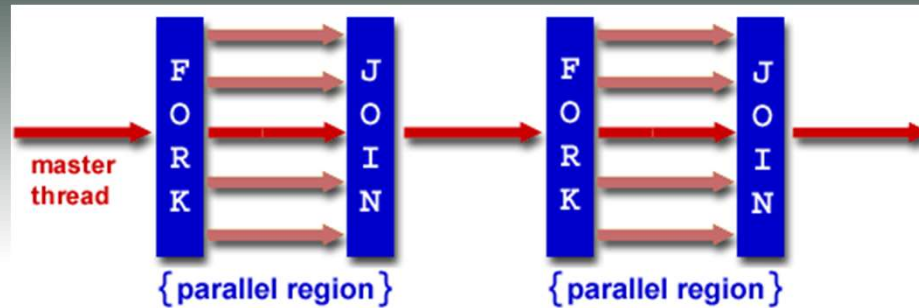
Which is better

- **Shared** - all processors share a global pool of memory
 - simpler to program
 - bus contention leads to poor scalability
- **Distributed** - each processor physically has it's own (private) memory associated with it
 - scales well
 - memory management is more difficult



- OpenMP program **starts single threaded**
- To create additional threads, user starts a parallel region
 - more threads are launched to create a team
 - original (master) thread is part of the team
 - threads “go away” at the end of the parallel region: usually sleep or spin
- **Repeat** parallel regions as necessary
 - **Fork-join model**





- start with one thread
 - called the master thread (poor term)
- indicated parallel region
 - "team" starts threads to run region code
- region finishes, wait for all threads to get there and join (just like threads). A implicit **barrier**
- back to the single master
- so on and so forth

```
#include<iostream>
using std::cout; using std::endl;
#include<cstdlib>
#include<omp.h>
```

special include

```
int main (int argc, char *argv[]){
    if (argc != 2)
        cout << "Need a thread count arg" << endl;
    else {
        int thrd_cnt = atoi(argv[1]);
        omp_set_num_threads(thrd_cnt);
```

red are omp functions

```
        cout << "Max number of threads: " << omp_get_max_threads() << endl;
        #pragma omp parallel
```

green are compiler directives

```
{
    if (omp_get_thread_num() == 0){
        cout << "I'm the prime node, I'm special " << endl;
        cout << "Actual number of threads: " << omp_get_num_threads() << endl;
    }
    cout << "I'm worker " << omp_get_thread_num() << endl;
}
}
```

*blue block directly after the pragma
is the parallel section*

compiling and output

```
g++ -fopenmp hello-omp.cpp
```

```
./a.out 4
```

```
Max number of threads: 4
```

```
I'm the prime node, I'm special
```

```
Actual number of threads: 4
```

```
I'm worker 0
```

```
I'm worker 1
```

```
I'm worker 2
```

```
I'm worker 3
```



Is it really threaded?

- If you get an error in the pragma, not always clear
- by default, it just runs serial
- good to check.
 - for example, not doing `-fopenmp` is an example. No errors and no threading




Some warnings

- no order on I/O. Threads can output in any order
 - if you need order, can be done by concurrency controls
 - could output to multiple files by thread
- OpenMP is free to cache data in a thread and output to memory when it sees fit.
 - flush pragma directive



Parallel section run by all

- That parallel section is run in its **entirety by each thread** (including master)
 - Spawn the number of threads you set by the function
 - multiple ways to do this
 - `#pragma omp parallel num_threads(cnt)`
 - All threads wait at the end of the section then the master picks up again, an **implicit barrier**
- 

How many threads did I get?

- OpenMP is **not required** to give you the number of threads you requested.
 - if you want to know what you received, you have to note it somehow.



OpenMP scope and sharing

CMSE 822, FS21, W.F. Punch

thread local

- **global sharing**: any variable outside of the parallel block is global to each thread spawned.
 - shared access, no sync (unless we do it)
- **thread local**: any variables declared inside the parallel block will be local to **each thread**
 - no sharing of values
 - lost when the thread ends.



pragma control

- `private (list)`. List of thread local variables
- `shared (list)`. List of shared variables.
- `default(shared | none)`
 - everything shared
 - unknown status and programmer must specify.



Sync

OMP has a number of synchronization but the easiest to use is:

```
#pragma omp critical
```

Inside a parallel section this is a mutual exclusion: only one thread in, others wait



trap1



but we can do better

OMP is trying to make our threading life easier, and so it provides a way to reduce, change multiple value into a single (or at least fewer number) value.

```
#pragma omp reduction(op : var)
```



reductions operations

op can be: +, *, -, &, |, ^, &&, ||

Effectively, OpenMP makes a private variable for each thread and then, and then the resulting private variables are "op" together, yielding a result



trap-reduction



OpenMP parallel Region Directive

#pragma omp parallel [*clause list*]

Typical clauses in [*clause list*]

- Conditional parallelization
 - **if** (scalar expression)
 - Determine whether the parallel construct creates threads
- Degree of concurrency
 - **num_threads** (integer expression)
 - number of threads to create
- Data Scoping
 - **private** (variable list)
 - Specifies variables local to each thread
 - **firstprivate** (variable list)
 - Similar to the private
 - Private variables are initialized to variable value before the parallel directive
 - **shared** (variable list)
 - Specifies variables that are shared among all the threads
 - **default** (data scoping specifier)
 - Default data scoping specifier may be **shared** or **none**



more about default

- `default(none)` : With this clause the compiler will require that we specify the scope of each variable we use in the block and that has been declared outside the block.
- `default(shared)` : assume everything is shared, up to you to declare private.



```
#pragma omp parallel if (is_parallel == 1)
num_threads(8) shared (var_b) private (var_a)
firstprivate (var_c) default (none)
{
/* structured block */
}
```

- **if (is_parallel == 1) num_threads(8)**
 - If the value of the variable is_parallel is one, create 8 threads
- **shared (var_b)**
 - Each thread shares a single copy of variable b
- **private (var_a) firstprivate (var_c)**
 - Each thread gets private copies of variable var_a and var_c
 - Each private copy of var_c is initialized with the value of var_c in main thread when the parallel directive is encountered
- **default (none)**
 - Default state of a variable is specified as none (rather than shared)
 - Signals error if not all variables are specified as **shared** or **private**

Nested Sections

For nested sections, the default is that the outer section is parallel and the inner sections are single threaded.

