OpenMP Overview

For Parallel Computing Overview Tutorial

OpenMP: Conceptual Overview

- Serial “process” and process address space
- OpenMP parallel region & threads
- Worksharing constructs
- Shared and Private variables
- OpenMP compiler directives
- OpenMP functions
- OpenMP build & runtime: tools & environments
- Documentation & resources

Serial Process

Process starts

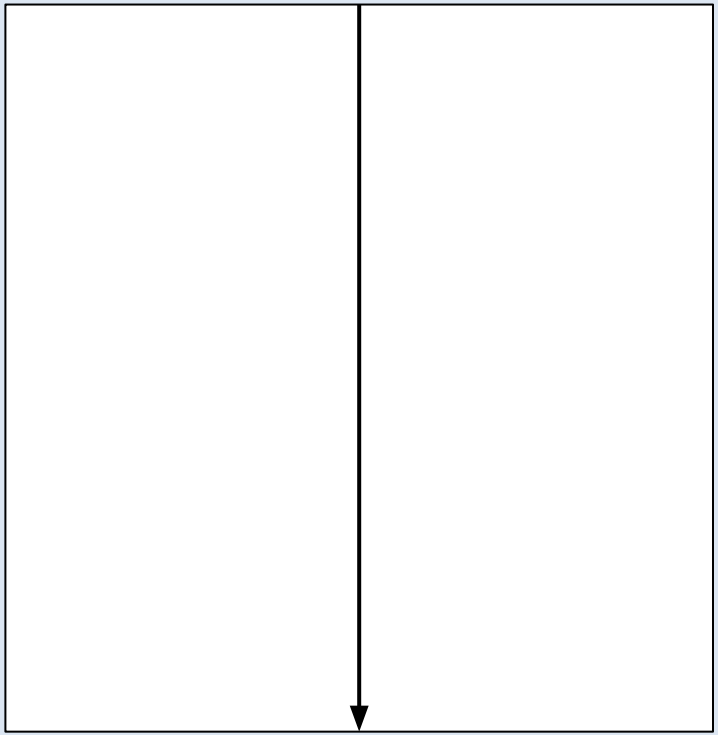
Memory Address space

- Variables
- code

Single thread

Sequential operations

Process ends



```
PROGRAM serial_example_1
```

Serial / Sequential Code

```
stop
ebd
```

Parallel with OpenMP Threads

Process starts

Single “master” thread

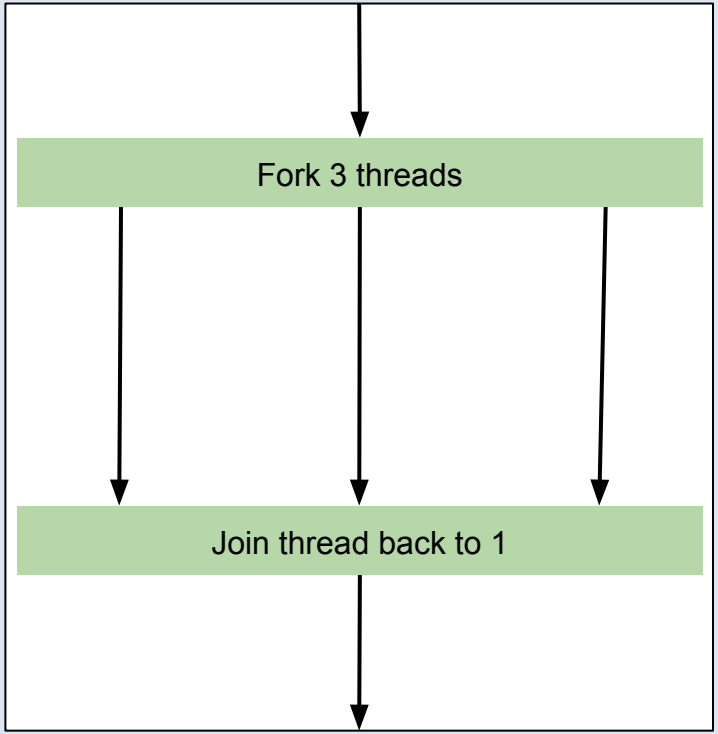
Fork

- **Concurrent execution**
- **Independent threads**
- **Shared variables**
- **Private variables**

Join

- **Back to single thread**

Process ends




```
PROGRAM threaded_example_1
```

Serial/sequential

```
!$OMP PARALLEL
```

OpenMP Parallel Region

```
!$OMP END PARALLEL
```

Serial/sequential

```
stop
ebd
```

Serial Application

Process starts

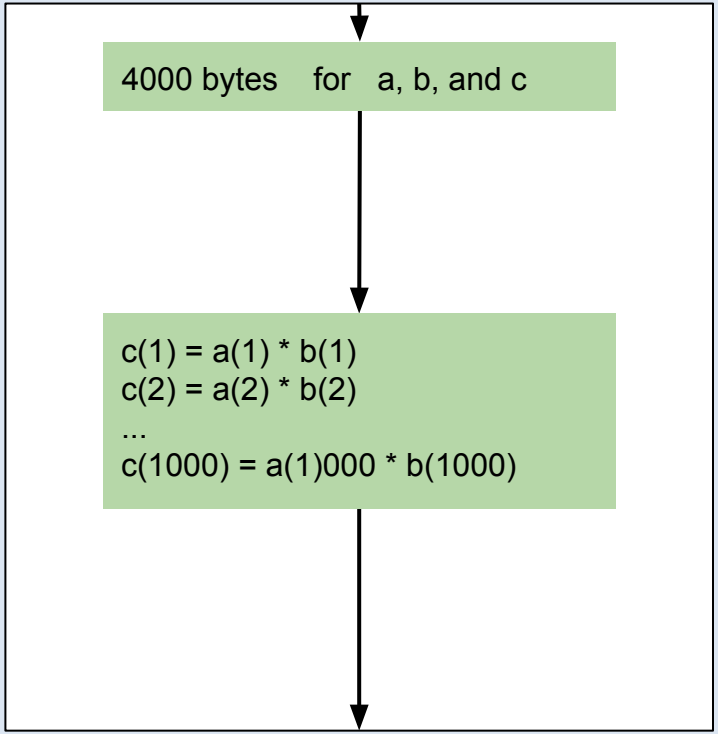
Memory Address space

- **Variables**
- **code**

Thread of execution

- **Sequential operations**

Process ends




```
PROGRAM serial_example_2
real a(1000), b(1000), c(100)

do i = 1, 1000
  c(i) = a(i) * b(i)
enddo

stop
ebd
```

Parallel But Redundant

Process starts

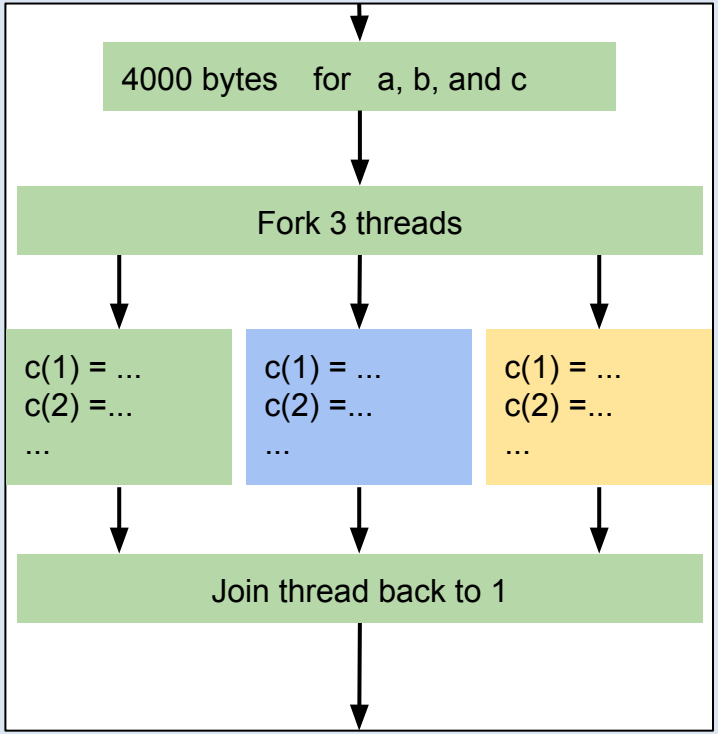
Allocate shared arrays

Concurrent threads

Parallel execution

REDUNDANT Work

Process ends




```
PROGRAM threaded_example_bad
real a(1000), b(1000), c(100)

!$OMP PARALLEL shared(a,b,c)

do i = 1, 1000
  c(i) = a(i) * b(i)
enddo

!$OMP END PARALLEL
stop
ebd
```

Work Sharing: DO

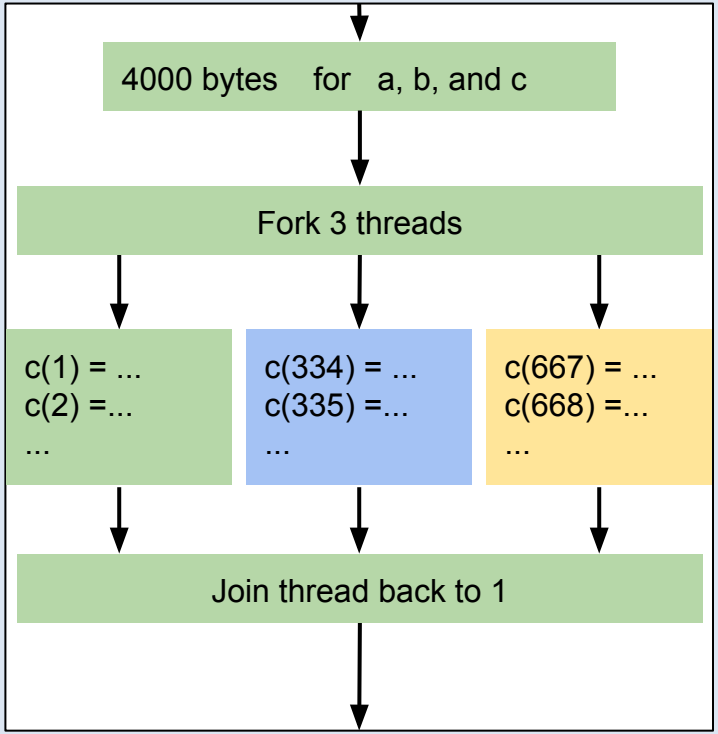
Process starts

Allocate shared arrays

Concurrent threads

- **Parallel execution**
- **Work Sharing**

Process ends




```
PROGRAM threaded_example_good
real a(1000), b(1000), c(100)

!$OMP PARALLEL shared(a,b,c)
!$OMP DO
do i = 1, 1000
  c(i) = a(i) * b(i)
enddo
!$OMP END DO
!$OMP END PARALLEL
stop
ebd
```

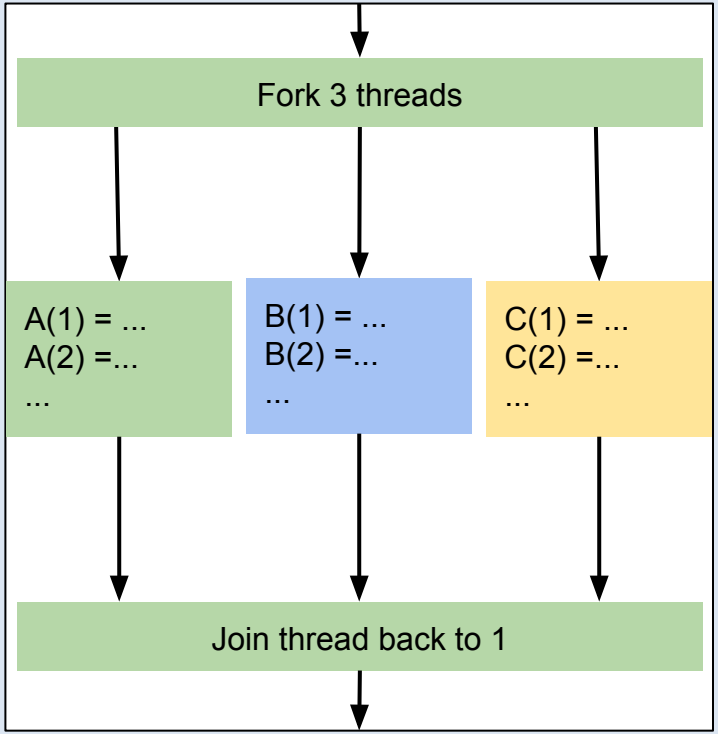
Work Sharing: SECTIONS

Inside Parallel region

Threads do different sections of code.

Max threads used:

= number of sections



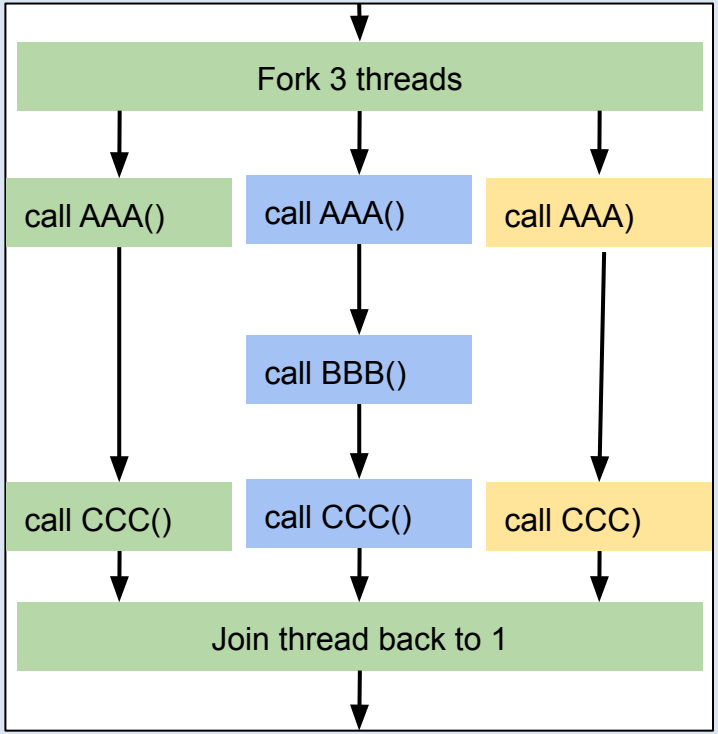

```
!$OMP PARALLEL shared(A,B,C)
!$OMP SECTIONS
!$OMP SECTION
A(:) = ...
!$OMP SECTION
B(:) = ...
!$OMP SECTION
C(:) = ...
!$OMP END SECTION
!$OMP END PARALLEL
```

Work Sharing: SINGLE

Inside Parallel region

Only one thread calls BBB().

Usually 1st thread to reach section of code will do it.




```
!$OMP PARALLEL

call AAA()

!$OMP SINGLE
Call BBB()
!$OMP END SINGLE

call CCC()

!$OMP END PARALLEL
```

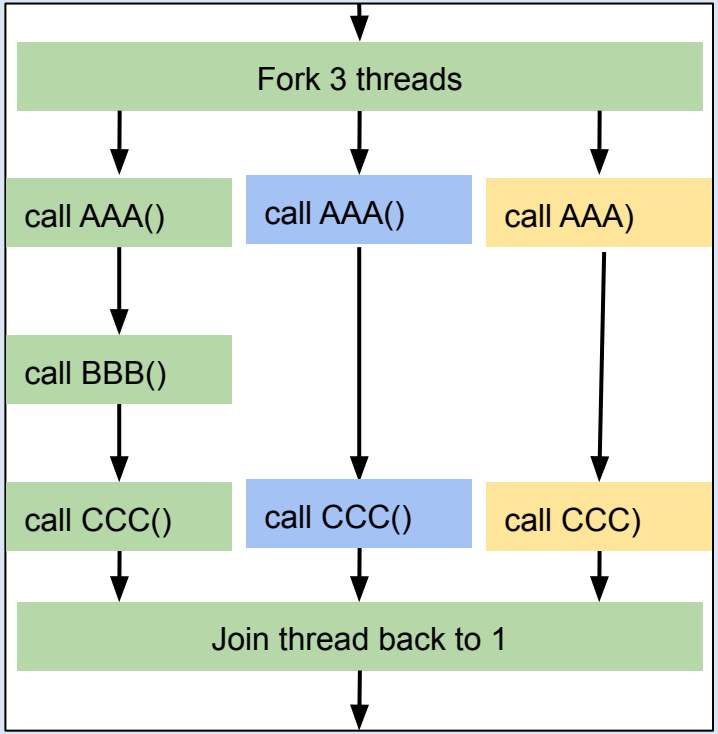
Work Sharing: MASTER

Inside Parallel region

Only MASTER thread calls BBB().

MASTER thread:

- **Initial thread of app**
- **Thread 0**
- **Full context of app**

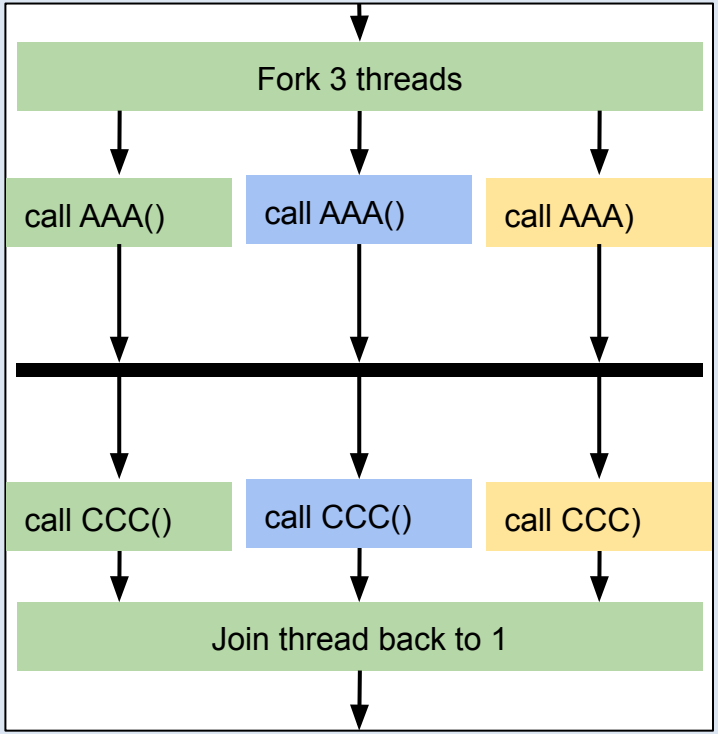



```
!$OMP PARALLEL

call AAA()

!$OMP MASTER
Call BBB()
!$OMP END MASTER

call CCC()

!$OMP END PARALLEL
```

Synchronization: BARRIER

Inside Parallel region

Threads stop at barrier and wait for others.




```
!$OMP PARALLEL
call AAA()
!$OMP BARRIER
call CCC()
!$OMP END PARALLEL
```

OpenMP Scope of Variables

SHARED	Variable is same on each thread Same memory location used ⇒ no new memory allocated Value set by one thread is seen by all threads
PRIVATE	Variable is different on each thread Different memory locations used ⇒ memory allocated for each thread Changes to variable values invisible between threads
THREADPRIVATE	Private variables with values that persist from between parallel regions New memory allocated only when variable 1st appears in a parallel region

Default Variable Scopes:

- Declared outside parallel region: shared
- Declared inside parallel region: private

OMP PARALLEL DIRECTIVE

FORTRAN	C / C++

!$OMP PARALLEL [clauses]
CODE BLOCK
!$OMP END PARALLEL
	#pragma omp parallel [clause ...]
	{
	structured block
	}

Clauses:

shared (list) Variable in list are the same (shared) between threads

private (list) Variable in list are different from thread to thread

firstprivate (list) Private with initial value copied in

default(shared | none) C/C++

default(PRIVATE | FIRSTPRIVATE | SHARED | NONE) FORTRAN

OMP PARALLEL DIRECTIVE

Clauses (continued):

num_threads (integer-expression)	Specifies number of threads in region
reduction (operator: list)	list=operator(list) returned to MASTER
if (scalar_expression)	Run region non-parallel if FALSE

Private variables & common blocks that persist between parallel regions:

threadprivate (list)	Private variable with global scope in code
copyin (list)	Copies values from MASTER thread to threadprivate

OMP DO DIRECTIVE

FORTRAN	C / C++
!$OMP DO [clauses] ***DO_LOOP*** !$OMP END PARALLEL [nowait]	#pragma omp do [clause ...] **for_loop { … }**

Clauses:

schedule (type [,chunk])	static | dynamic, size of index blocks
ordered	force sequential ordering of indices
lastprivate (list)	copy values in list from those of last loop index
collapse (n)	collapse nested loops (n deep) into single loop
nowait	allow threads to continue before all complete loop

OMP SECTION DIRECTIVES

FORTRAN	C / C++

!$OMP SECTIONS [clauses]
!$OMP SECTION
CODE_BLOCK
!$OMP SECTION
CODE_BLOCK
...
!$OMP SECTION
CODE_BLOCK
!$OMP END SECTION
	#pragma omp sections
	{
	#pragma omp section
	init_field(field);
	
	#pragma omp section
	check_grid(grid);
	}

Relevant Clauses:

private (list)
firstprivate (list)
lastprivate (list)
reduction (operator: list)

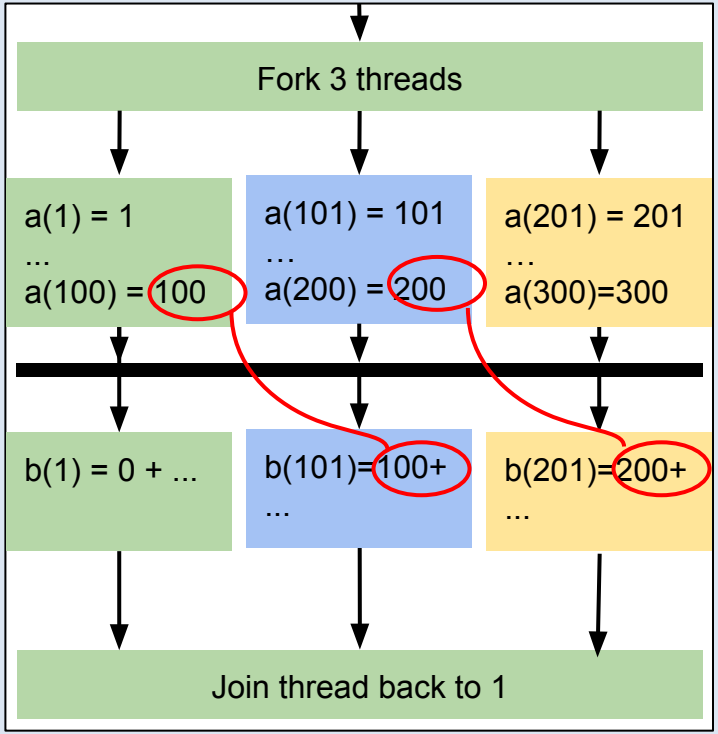
OMP PARALLEL: SHARED

Arrays a & b are shared

- **a(100) is SAME memory location on ALL threads**

OMP END DO ⇒

- **Default: wait for all thread to finish loop before continuing.**




```
real*4 a(0:301), b(0:301)
a(:) = 0.0
!$OMP PARALLEL SHARED(a,b)
!$OMP DO
do i = 1, 300
   a(i) = float(i)
enddo
!$OMP END DO
do i = 1, 300
   b(i) = a(i-1) + ...
enddo
!$OMP END DO
!$OMP END PARALLEL
```

OpenMP Functions

OMP_GET_NUM_THREADS()	Returns an Integer : number of threads in current section
OMP_GET_THREAD_NUM()	Returns an Integer: thread number [0,...,N-1]
OMP_SET_THREAD_NUM(N)	Set the default thread number for the next section to be N
OMP_WTIME()	Returns Double Precision wall clock time

There are many more functions

Most get or set properties of parallel regions or loops.

OpenMP Thread Control with Functions

Take full control

Maximum flexibility

Minimum overhead

Minimal synchronization

Complete transparency

OMP_GET_THREAD_NUM()

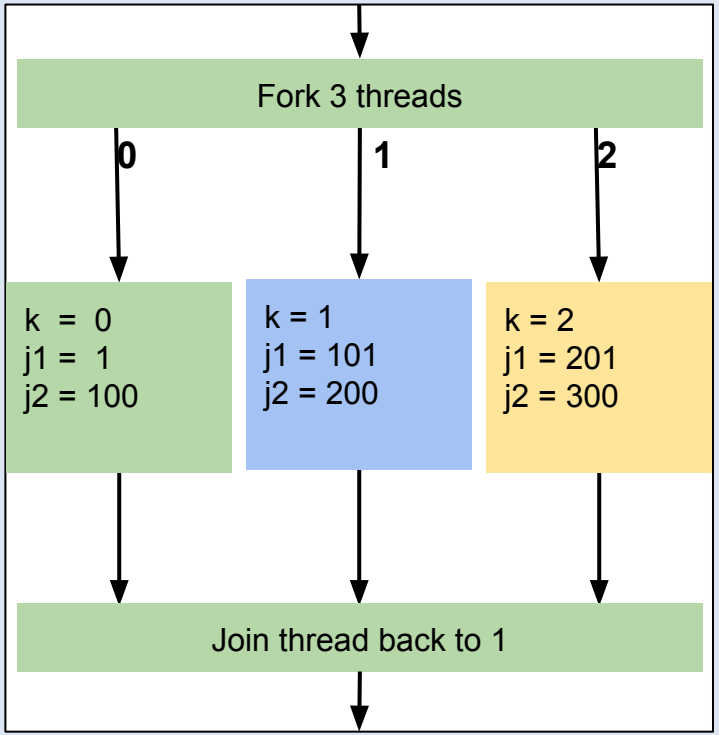
Get thread number

OMP_GET_NUM_THREADS()

Get number of threads

OMP_SET_NUM_THREADS(n)

Set default number of threads




```
integer OMP_GET_NUM_THREDS
!$OMP PARALLEL private(k,j1,j2)

k = OMP_GET_THREAD_NUM()
j1 = 1 + 100 *  j
j2 =     100 * (j+1)

!$OMP END PARALLEL
```

A Full Small Example

Making all threads say “Hello World”, in C.

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

int main(void) {
    printf("\n");
    #pragma omp parallel
    {
        const int thread_num = omp_get_thread_num();
        const int threads_total = omp_get_num_threads();
        printf("Hello World from thread = %d out of %d total threads.\n", \
          thread_num, threads_total);
    }
    printf("\n");
    return 0;
}
```

OpenMP: Environment, Build, & Run

	Intel	GNU
Modules	module load intel	module load gcc/7.2.0
Build	ifort -qopenmp code.f -o app icc -qopenmp code.c -o app	gfortran -fopenmp code.f -o app gcc -fopenmp code.c -o app
Shell Variables	export OMP_NUM_THREADS=24 export KMP_AFFINITY=granularity=core,scatter	export OMP_NUM_THREADS=24 export OMP_PROC_BIND=TURE
Run	./app numactl --interleave=0,1 ./app	./app

- **Run on MSI's HPC systems**
- **This is a sample. See documentation on next slide**

OpenMP Documentation & Resources

Tutorial: https://computing.llnl.gov/tutorials/openMP/

Summary: http://www-inst.eecs.berkeley.edu/~cs61c/resources/OpenMP3.0-SummarySpec.pdf

Intel: https://software.intel.com/en-us/fortran-compiler-developer-guide-and-reference-openmp-support

GNU: https://gcc.gnu.org/onlinedocs/libgomp/index.html#Top

Specifications (OpenMP Versions 2.5 - 5.0):

https://www.openmp.org/specifications/