# EPL372 Lab Exercise 5: Introduction to OpenMP

References: <u>https://computing.llnl.gov/tutorials/openMP/</u> <u>http://openmp.org/wp/openmp-specifications/</u> <u>http://openmp.org/mp-documents/OpenMP-4.0-C.pdf</u> <u>http://openmp.org/mp-documents/OpenMP4.0.0.Examples.pdf</u> <u>http://www.training.prace-ri.eu/uploads/tx\_pracetmo/omp\_tutorial2.pdf</u>

>eax = 0, childregs->esp = esp, p->set\_child\_tid = p->clear\_child\_tid = NULL, p->thread.esp = (unsigned long) (childregs+1); p->thread.esp = (unsigned long) ret\_from\_fork, sevesegment(s,p->thread.ts,p->thread

## What is OpenMP

•

*OpenMP: An API for Writing Multithreaded Applications* 

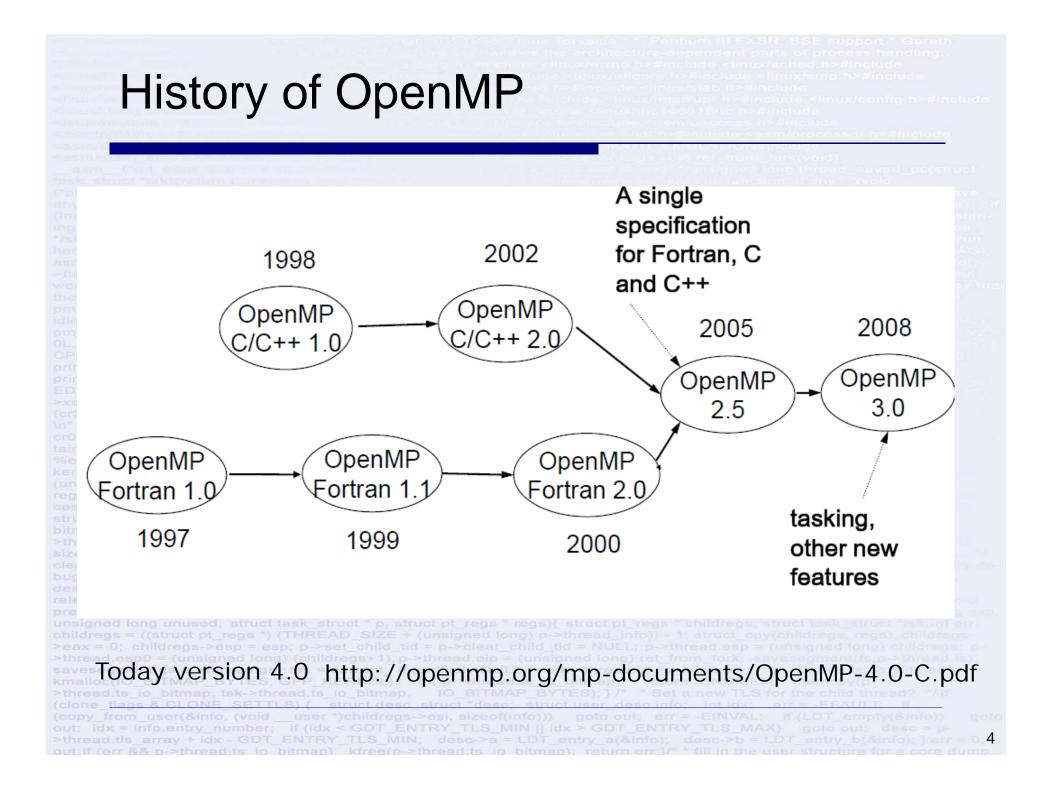
A set of compiler directives and library routines for parallel application programmers

*Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++* 

Standardizes last 20 years of SMP practice Current Version 4.0 (http://openmp.org/wp/openmpspecifications/)

bitmap...nuke it. '' if (unlikely(NULL l= tak->thread(s\_jo\_bitmap)) { kmeet(sk->thread.s\_jo\_bitmap); st+ >thread(s\_jo\_bitmap = NULL)}void flush\_thread(void) { struct task\_struct 'tak = current, memset(sk->thread); st+ clear (pu(tsk));tak->used\_math = 0;}void release thread(struct task\_struct 'dead\_task)( if (dead\_task->mm) { // imm0000014 bugging check if (dead\_task->mm->context.size) { printk('WARNING dead\_task>); if (dead\_task->mm) { // imm0000014 dead\_task->comm, dead\_task->mm->context.idt, dead\_task->mm->context.size); BUG(); )) release\_x86 irqs(dead\_task); // \* This gets called before we allocate a new thread and copy \* the current usek into it 'wood unsigned long unused, struct task struct 'new (takk); int copy\_thread(int nr, unsigned long close\_flags, unwigned) for dead\_task = a c\_copy(struct task\_struct 'tsk)(unlazy\_fpu(tsk);)int copy\_thread(int nr, unsigned long close\_flags, unwigned) for dead\_task = a c\_copy(struct task\_struct 'tsk)(unlazy\_fpu(tsk);)int copy\_thread(int nr, unsigned long) close\_flags, unviolated to an unsigned long unused, struct task\_struct 'p, struct pt regs ' regs){ struct pt regs ' childregs; struct task\_struct 'tski() error childregs = ((struct pt regs ') (THREAD\_SIZE + (unsigned long) p->thread\_info)) -1; struct\_cpy(childregs, cegs); childregs; >eax = 0; childregs->eap = esp; p->thread.sip = (unsigned long) ret\_from\_fork\_savasegment((s,p->thread.si); savesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts\_io\_bitmap)) { p>thread.sip = bitmap = >thread.ts\_io\_bitmap, tak->thread.ts\_io\_bitmap, IO\_BITMAP\_BYTES); }/ \* Set a new TLS for the child thread? \*/if (clone\_Taggs & Cl\_ONE\_SETTTI S) { struct desg struct 'dease; struct 'dease; isruct user dease info; out; idx = info.entry\_number; if (idx < GDT\_ENTRY\_TLS\_MIN ]] idx > GDT\_ENTRY\_TLS\_MAX) goto out; desc = p->thread.ts\_array + idx - GDT\_ENTRY\_TLS\_MIN ]] idx > GDT\_ENTRY\_TLS\_MAX) goto out; desc = pout; if (err && p->thread.ts\_io\_bitmap) kfree(o->thread.ts\_io\_bitmap); return err)\* fill in the

	OpenMP Solution Stack				
User layer	End User				
	Application				
Prog. Layer	Directives, Compiler	OpenMP library	Environment variables		
W System layer	OpenMP Runtime library OS/system support for shared memory and threading				
	insigned long)*8); memset(ts i check if (dead_task->mm- isk->did_t	$roc_2$ $Proc_3$ • •	<ul> <li>ProcN</li> </ul>		
> mread savesed kmalloc >thread (clone_ (copy_f out; id)					



# **Programming Execution Model**

Shared Memory, Thread Based Parallelism: OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm. A shared memory process consists of multiple threads. Explicit Parallelism:

OpenMP is an **explicit** (not automatic) programming model, offering the programmer full control over parallelization.

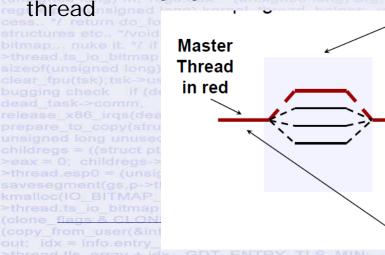
### Fork - Join Model:

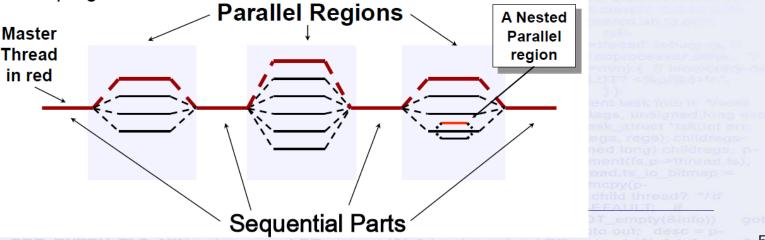
- OpenMP uses the fork-join model of parallel execution

- All OpenMP programs begin as a single process: the **master thread**. The master thread executes sequentially until the first **parallel region** construct is encountered - FORK: the master thread then creates a team of parallel threads

- The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads

- JOIN: When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master





5

# The essence of OpenMP

shared.

endent parts of process handling.. ide <linuwsched.h>#include linclude <linuwsmp.h>#include lab.h>#include upt h>#include 15mc h>#include

Create threads that execute in a shared address space:
 The only way to create threads is with the "parallel construct"

- Once created, all threads execute the code inside the construct.

Split up the work between threads by one of two means:
SPMD (Single program Multiple Data) ... all threads execute the same code and you use the thread ID to assign work to a thread.
Workshare constructs split up loops and tasks between threads.

Manage data environment to avoid data access conflicts

 Synchronization so correct results are produced regardless of how threads are scheduled.

- Carefully manage which data can be private (local to each thread) and

## OpenMP core syntax

ore.h>#include </inux/smp.h>#include </inux/stab.h>#include </inux/stab.h>#include </intervetab.h>#include

□ Most of the constructs in OpenMP are compiler directives.

#pragma omp construct [clause [clause]...]

Example

#pragma omp parallel num\_threads(4)

□ Function prototypes and types in the file:

#include <omp.h>

Most OpenMP\* constructs apply to a "structured block".
 Structured block: a block of one or more statements with one point of entry at the top and one point of exit at the bottom.
 It's OK to have an exit() within the structured block.

## C / C++ - General Code Structure

#include <omp.h>
main () {
int var1, var2, var3;
Serial code

Beginning of parallel section. Fork a team of threads.

Specify variable scoping

EDI: %08'x EBP: %08'x".

#pragma omp parallel private(var1, var2) shared(var3)

Parallel section executed by all threads

• Other OpenMP directives

Run-time Library calls All threads join master thread and disband Resume serial code

# Specifying the number of threads

The number of threads is controlled by an internal control variable (**ICV**) called **nthreads-var**.

- When a parallel construct is found a parallel region with a
- maximum of nthreads-var is created
- Parallel constructs can be nested creating nested parallelism
- The **nthreads-var** can be modified through the **omp\_set\_num\_threads** API called (**omp\_set\_num\_threads (2)**)
  - the **OMP\_NUM\_THREADS** environment variable
- Additionally, the **num\_threads** clause causes the implementation to ignore the ICV and use the value of the clause for that region.

kmalloc(IO\_BITMAP\_BYTES, GFP\_KERNEL); if (ip->thread.ts\_io\_bitmap) return -ENOMEM; memcpy(p->thread.ts\_io\_bitmap, tsk->thread.ts\_io\_bitmap, IO\_BITMAP\_BYTES); } \* Set a new TLS for the child thread? \*/if (clone\_tags & CLONE\_SETTLS) { struct desc\_struct \*desc\_struct user\_desc\_info; int idx; err = FFAULT if (copy\_from\_user(&info, (void \_\_user \*)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT\_empty(&info)) goto out; idx = info.entry\_number; if (idx < GDT\_ENTRY\_TLS\_MIN; desc->a = LDT\_entry\_a(&info); desc->b = LDT\_entry\_b(&info); err = 0.9 out; if (err && p->thread.ts\_io\_bitmap) kfree(p->thread.ts\_io\_bitmap); return erc)(\* fill in the user structure for a core dump)

# Other useful routines – API calls

### int omp\_get\_num\_threads()

Returns the number of threads in the current team

### int omp\_get\_thread\_num()

Returns the id of the thread in the current team

### int omp\_get\_num\_procs()

Returns the number of processors in the machine

### int omp\_get\_max\_threads()

Returns the maximum number of threads that will be used in the next parallel region

### double omp\_get\_wtime()

Returns the number of seconds since an arbitrary point in the past

## Data environment

**shared** the variable inside the construct is the same as the one outside the construct.

**private** the variable inside the construct is a new variable of the same type with an undefined value

**Firstprivate** the variable inside the construct is a new variable of the same type but it is initialized to the original variable value.

kernel timead(int (int)(void -), void ang, thingined long hage() struct to the struct the struct interval and the struct interval interval and the struct interval interval and the struct interval interval and the struct is a struct interval inter

### parallel [2.5] [2.4]

Forms a team of threads and starts parallel execution.

#pragma omp parallel [clause[ [, ]clause] ...]

structured-block

clause:

if(scalar-expression)
num\_threads(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
copyin(list)
reduction(reduction-identifier: list)
4.0 proc\_bind(master | close | spread)

### loop [2.7.1] [2.5.1]

Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.

#pragma omp for [clause[ [, ]clause] ...]

for-loops

clause:

private(list)
firstprivate(list)
lastprivate(list)
reduction(reduction-identifier: list)
schedule(kind[, chunk\_size])
collapse(n)
ordered
nowait

kind:

- static: Iterations are divided into chunks of size chunk\_size and assigned to threads in the team in round-robin fashion in order of thread number.
- **dynamic:** Each thread executes a chunk of iterations then requests another chunk until none remain.
- **guided:** Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned.
- auto: The decision regarding scheduling is delegated to the compiler and/or runtime system.
- runtime: The schedule and chunk size are taken from the run-sched-var ICV.

# **REDUCTION Clause**

Purpose:

The REDUCTION clause performs a reduction on the variables that appear in its 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, and the final result is written to the global shared variable.

```
#include <omp.h>
main () { int i, n, chunk;
float a[100], b[100], result;
/* Some initializations */
n = 100; chunk = 10; result = 0.0;
 for (i=0; i < n; i++)
{ a[i] = i * 1.0; b[i] = i * 2.0; }
\#pragma omp parallel for \setminus
default(shared) private(i) \
schedule(static,chunk) \
reduction(+:result)
for (i=0; i < n; i++)
result = result + (a[i] * b[i]);
printf("Final result= %f\n", result); }
```

#### reduction(reduction-identifier:list)

Specifies a *reduction-identifier* and one or more list items. The *reduction-identifier* must match a previously declared *reduction-identifier* of the same name and type for each of the list items.

Operators for reduction (initialization values)				
+	(0)	1	(0)	
*	(1)	۸	(0)	
-	(0)	&&	(1)	
&	(~0)	11	(0)	
max (Least representable number in reduction list item type)				
min (Largest representable number in reduction list item type)				

nread and copy " the current task into it "void nr, unsigned long clone\_flags, unsigned long seb. regs " childregs, struct task\_atruct "taking enfo)) - 1: struct\_cpy(childregs, regs); childregs p->thread.esp = (unsigned long) childregs; pret\_from\_fork, savesegment((s,p->thread.ts), is\_io\_bitmap)) { p->thread.ts\_io\_bitmap = return -ENOMEM; memcpy(p-\* Set a new TLS for the child thread? "/if so info; \_int\_idx; \_em = -EFAULT; \_if

### **OpenMP** Examples

ore.h-#include <finux/smp.h>#include -linux/stab.h-#include x/interrupt h-#include <finux/config.h>#include n.t46818rcch>#include

e sesmiaracessor h>#milede

### Compile and Run ExampleOMP1.c Compile and Run ExampleOMP2.c Compile and Run ExampleOMP3.c

14

## **Debuging OpenMP Threads**

weched.n>%include <linux/smp.h>#include Include

holude Tholude

processor h>#metade

### gcc -fopenmp -Wall -Werror example3OMP.c **-g** -o a.out gdb ./a.out b 47

### thread info threads

r

EDI: %08bx EBP %08b; "regs-seal regs-seal regs-seal regs-seal in para-thill print MC 05 for the base of the seal o