



信息科学与技术学院

School of Information Science and Technology

CS 110

Computer Architecture

Thread-Level Parallelism

Instructors: Siting Liu & Yuan Xiao

Course website: <https://faculty.sist.shanghaitech.edu.cn/liust/courses/CS110.html>

School of Information Science and Technology (SIST)

ShanghaiTech University

2026/5/14

Administratives

- Mid-term II May 21st 8am-10am; you can bring 2-page A4-sized double-sided cheat sheet, **handwritten** only! (**Teaching center 301/SIST 1D-107/108**); From start to May 19th lecture (Thread-level parallelism/TLP).
- Project 2.2 to released, ddl June 4th.
- HW6 released, ddl May 28th!
- Lab 11 released.
- Discussion May 22nd on Cache.

Parallelism Overview

- **Parallel Requests**
Assigned to computer
e.g., Search “CS110”
- **Parallel Threads**
Assigned to core
e.g., Lookup, Ads
- **Parallel Instructions**
>1 instruction @ one time
e.g., 5 pipelined instructions
- **Parallel Data**
>1 data item @ one time
e.g., Add of 4 pairs of words
- **Hardware descriptions**
All gates @ one time
- **Programming Languages**

Software

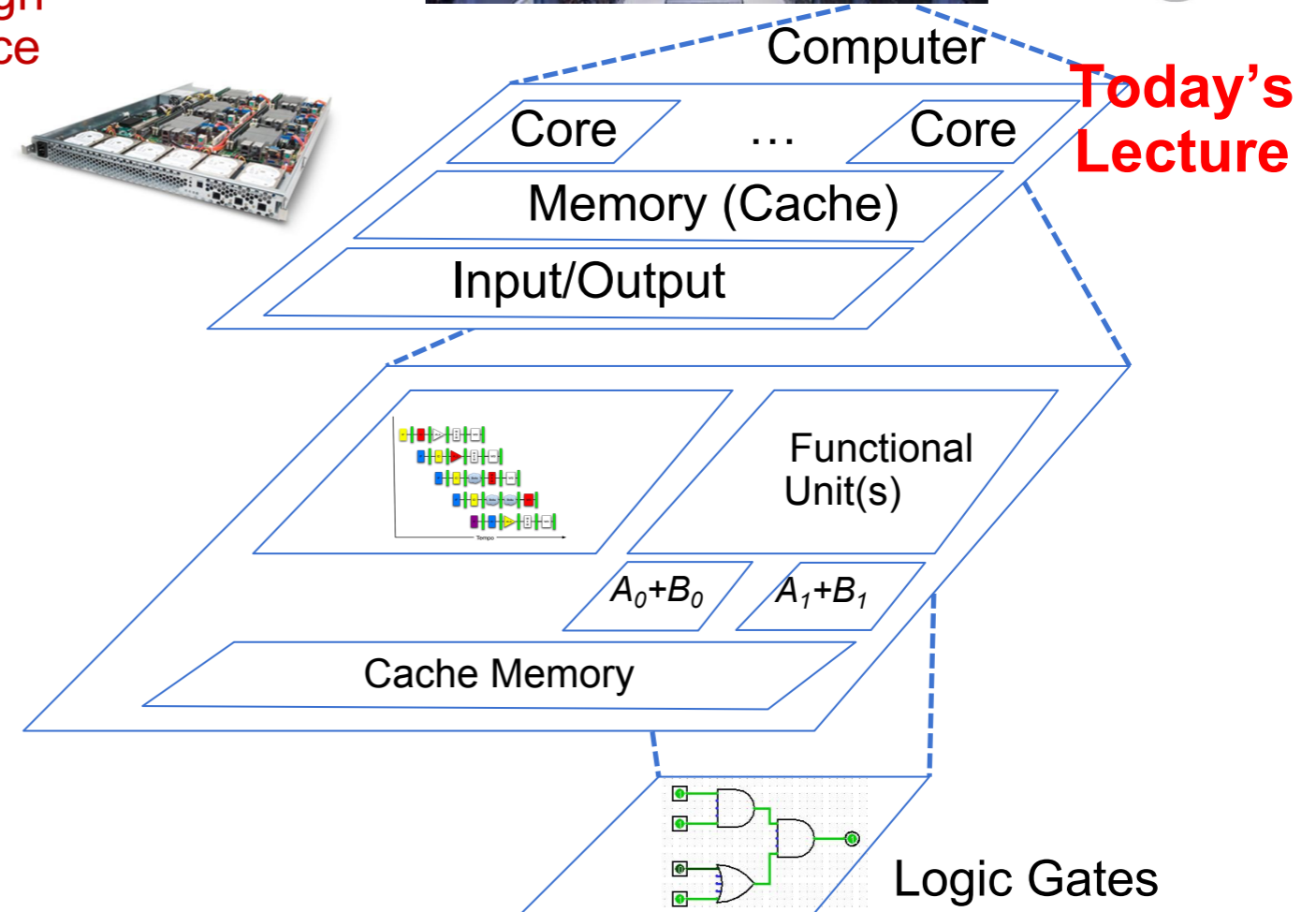
Hardware

Harness
Parallelism &
Achieve High
Performance

Warehouse
Scale
Computer



Smart
Phone

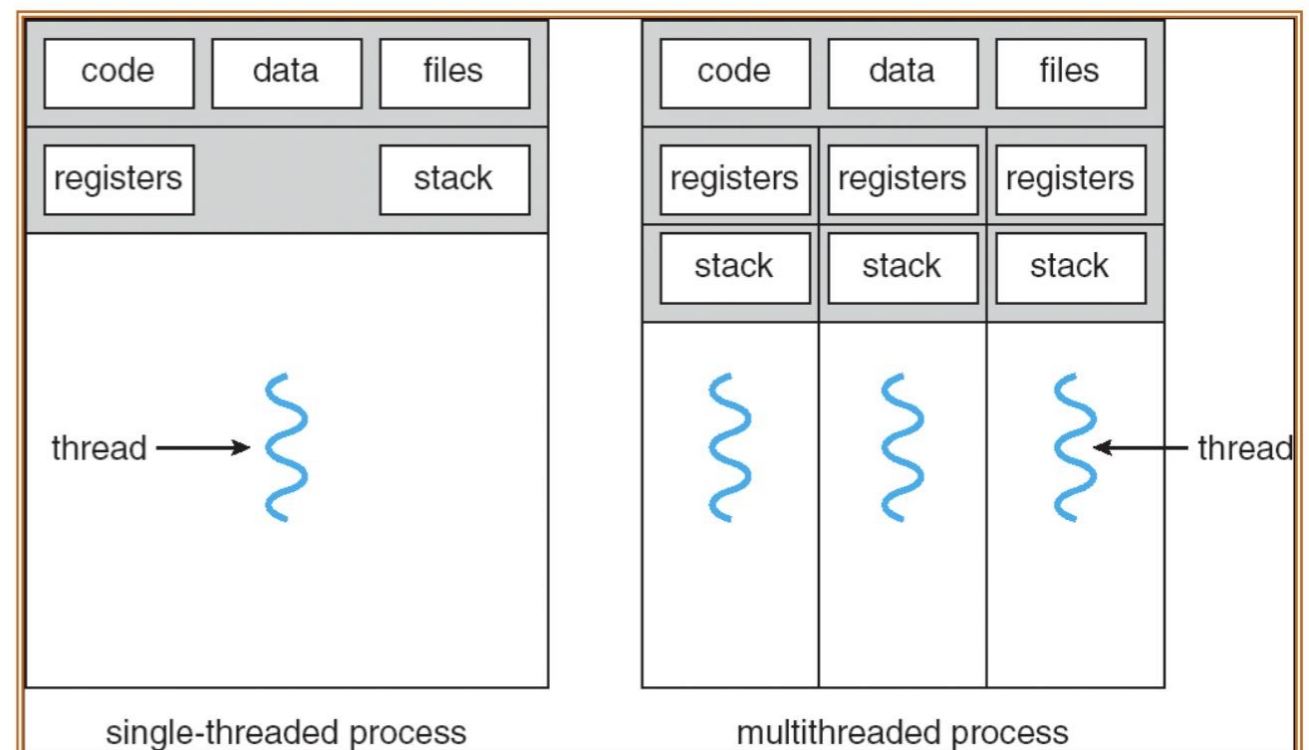


Motivation

进程名	用户	% CPU	ID	内存	磁盘读取总计	磁盘写入总计	磁盘读取	磁盘写入	优先级
prometheus	siting	2.12	38213	216.4 MB	3.9 MB	8.4 MB	不适用	不适用	普通
gnome-shell	siting	1.21	3767	223.4 MB	52.9 MB	196.6 kB	25.3 KiB/s	不适用	普通
wps-office	siting	0.62	37774	46.9 MB	115.1 MB	3.8 MB	不适用	不适用	普通
ibus-daemon	siting	0.42	4141	13.8 MB	221.2 kB	8.2 kB	不适用	不适用	普通
wps	siting	0.20	38439	177.4 MB	148.1 MB	42.8 MB	不适用	1.3 KiB/s	普通
wpp	siting	0.20	38231	555.8 MB	126.4 MB	671.8 MB	不适用	不适用	普通
ibus-extension-gtk3	siting	0.19	4319	16.6 MB	1.6 MB	不适用	不适用	不适用	普通
ibus-engine-libpinyin	siting	0.19	4521	22.9 MB	40.3 MB	905.2 kB	33.3 KiB/s	不适用	普通
wps-office	siting	0.19	37681	89.3 MB	212.6 MB	40.1 MB	不适用	不适用	普通
wpscloudsvr	siting	0.17	37709	87.9 MB	68.6 MB	60.8 MB	不适用	1.3 KiB/s	普通
prometheus	siting	0.17	37870	29.4 MB	14.1 MB	不适用	不适用	不适用	普通
gnome-system-monitor	siting	0.13	344605	56.7 MB	7.8 MB	不适用	2.7 KiB/s	不适用	普通
nautilus	siting	0.08	5577	112.0 MB	27.4 MB	1.6 MB	90.7 KiB/s	不适用	普通
gnome-characters	siting	0.08	344848	57.7 MB	4.6 MB	4.1 kB	不适用	不适用	普通
Isolated Web Co	siting	0.05	15487	131.4 MB	9.2 MB	不适用	不适用	不适用	普通
gjs	siting	0.05	132796	23.0 MB	不适用	不适用	不适用	不适用	普通
gnome-clocks	siting	0.05	344846	10.6 MB	426.0 kB	不适用	不适用	不适用	普通
java	siting	0.03	4376	252.8 MB	171.1 MB	4.7 MB	不适用	不适用	普通
gnome-keyring-daemon	siting	0.02	3484	1.2 MB	2.9 MB	4.1 kB	不适用	不适用	普通
dbus-daemon	siting	0.02	3495	2.6 MB	不适用	不适用	不适用	不适用	普通
Xwayland	siting	0.02	4528	41.0 MB	1.2 MB	不适用	不适用	不适用	普通
tracker-miner-fs-3	siting	0.02	4597	21.7 MB	101.1 MB	19.8 MB	172.0 KiB/s	不适用	非常低
aTrustAgent	siting	0.02	4883	46.8 MB	84.1 MB	4.0 MB	不适用	不适用	普通
Isolated Web Co	siting	0.02	116700	59.0 MB	16.4 kB	不适用	不适用	不适用	普通
Web Content	siting	0.02	341781	16.7 MB	不适用	不适用	不适用	不适用	普通
Web Content	siting	0.02	344278	16.6 MB	不适用	不适用	不适用	不适用	普通
Isolated Web Co	siting	0.00	129673	121.3 MB	471.0 kB	不适用	不适用	不适用	普通
(sd-pam)	siting	0.00	3463	1.9 MB	不适用	不适用	不适用	不适用	普通
pipewire	siting	0.00	3476	8.0 MB	86.0 kB	不适用	不适用	不适用	非常高
pipewire	siting	0.00	3477	901.1 kB	不适用	不适用	不适用	不适用	普通
wireplumber	siting	0.00	3481	4.6 MB	434.2 kB	131.1 kB	不适用	不适用	非常高
pipewire-pulse	siting	0.00	3482	10.2 MB	不适用	不适用	不适用	不适用	非常高
xdg-document-portal	siting	0.00	3544	798.7 kB	147.5 kB	不适用	不适用	不适用	普通
xdg-permission-store	siting	0.00	3548	540.7 kB	不适用	不适用	不适用	不适用	普通
gdm-wayland-session	siting	0.00	3598	528.4 kB	1.2 MB	不适用	不适用	不适用	普通
gnome-session-binary	siting	0.00	3605	2.0 MB	4.1 MB	不适用	不适用	不适用	普通
gcr-ssh-agent	siting	0.00	3645	712.7 kB	65.5 kB	不适用	不适用	不适用	普通
gnome-session-ctl	siting	0.00	3646	471.0 kB	20.5 kB	不适用	不适用	不适用	普通

Threads

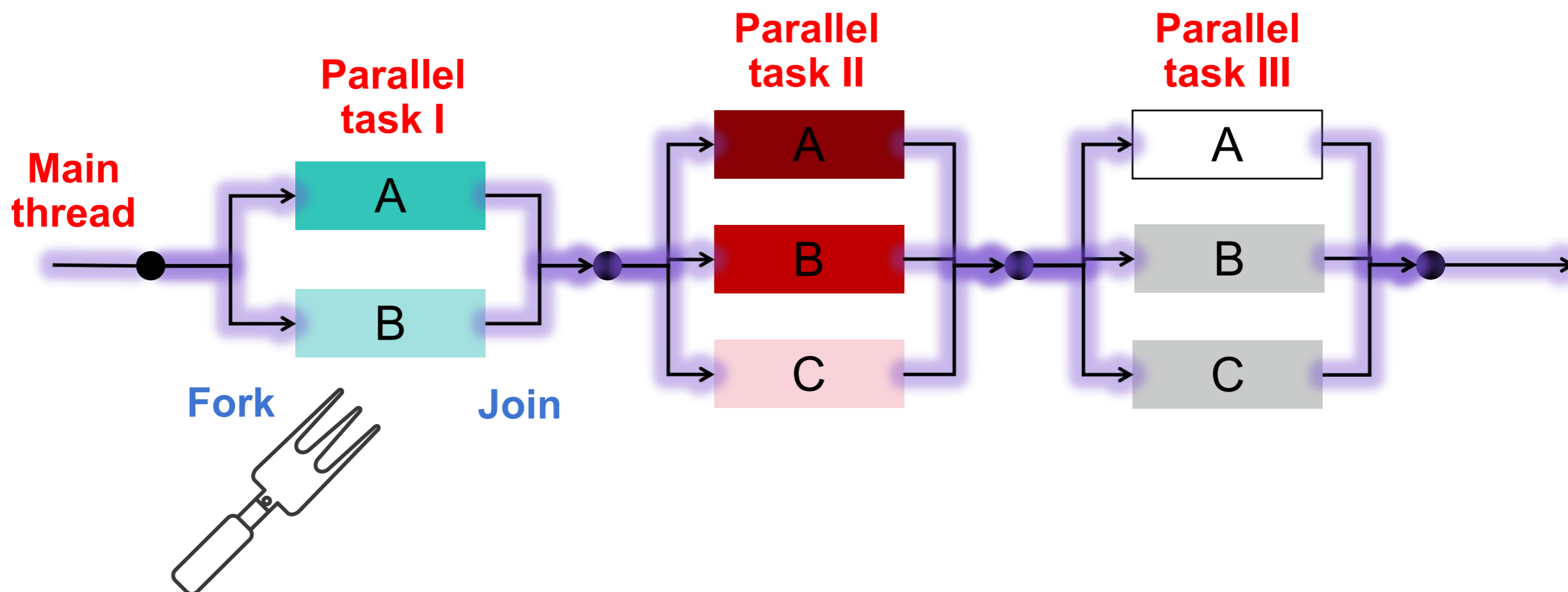
- Threads (short for threads of execution) is a single stream of instructions.
- Each thread has (its own state):
 - its own registers (including stack pointer)
 - its own program counter (PC)
 - shared memory (heap, global variables) with other threads
- Each processor provides one (or more) hardware threads (through **multi-core** or **single-core multithreading**, later) that actively execute instructions
- Within a given program's process, threads can run concurrently.
- Operating system (OS) multiplexes multiple **software** threads onto the available hardware threads.



Software threads

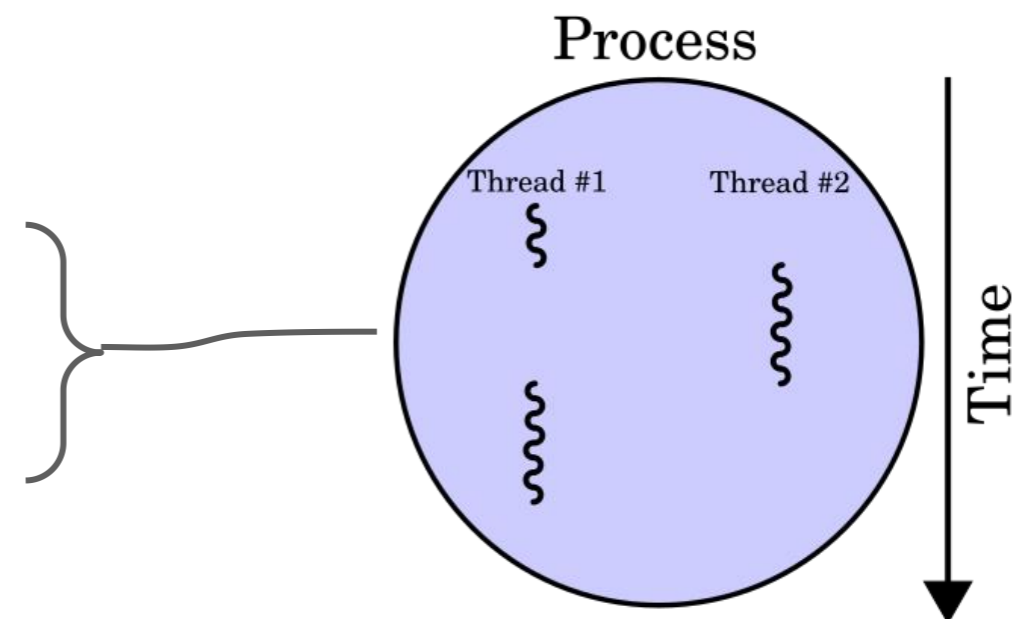
Fork-Join Model

- Fork-Join model: A program / process can split, or fork itself into separate threads, which can (in theory) execute concurrently.
 - Main thread executes sequentially until first parallel task region.
 - Fork: Main thread then creates a team of parallel subthreads.
 - Join: When subthreads complete their parallel task region, they synchronize and terminate, leaving only the main thread.



Thread-Level Parallelism & MIMD

- MIMD: Multiple Instruction, Multiple Data
 - Examples: Multicore systems, compute clusters, etc.
- A program / process can split, or fork itself into separate threads, which can (in theory) execute simultaneously.
- In hardware:
 - Single core: Multiple threads execute instructions on a single core, concurrently (time-multiplexing, giving the illusion of many active threads)
 - Multicore: Each thread executes on a separate core, simultaneously/in parallel
 - Can combine the above two...(more later)



A process with two threads of execution, running on a single processor [\[wiki\]](#)

Multithreaded programs can run on both SISD (time-shared) and MIMD systems (in parallel).

Operating System (OS) Threads

- The operating system, or OS, is responsible for (among other tasks) managing which threads get run on which CPUs.
- On most modern computers, number of active threads \gg number of available cores, so most threads are idle at any given time;
- Context switches: The OS can choose whichever threads it wants to run, and change threads at any time;
 - Remove a software thread from a hardware thread by interrupting its execution and saving its registers and PC into memory;
 - Threads from the same program share memory;
 - Can make a different software thread active by loading its registers into a hardware thread's registers and jumping to its saved PC, e.g., if one thread is blocked waiting for network access or user input.

“Hello, world!” to OpenMP

OpenMP is a language extension used for multi-threaded, shared-memory parallelism

- Generally follows the fork-join model
- “Open Multi-Processing”

Portable & standardized

Easy to compile

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

Use pragma, e.g., `#pragma omp parallel`

```
cc -fopenmp name.c
```

Key ideas:

Shared vs. private variables

OpenMP directives for:

Parallelization and work sharing

Synchronization

“Hello, world!” to OpenMP

```

#include <stdio.h>
#include <omp.h>
int main() {
    /* Fork team of threads with private variable tid */
    #pragma omp parallel
    {
        int tid = omp_get_thread_num(); /* get thread id */
        printf("Hello World from thread = %d\n", tid);
        /* Only main thread does this */
        if (tid == 0) {
            printf("Number of threads = %d\n", omp_get_num_threads());
        }
    } /* All threads join main and terminate */
    return 0;
}

```

Pragmas are a preprocessor mechanism C provides for language extensions.

This is annoying, but curly brace MUST go on separate line from #pragma

- Basic OpenMP construct for parallel regions:
 - Each thread runs the instructions within the block
 - Thread scheduling is non-deterministic

```

#pragma omp parallel
{
    /* code goes here */
}

```

“Hello, world!” to OpenMP

```
siting@siting-ThinkPad-T14p-Gen-1: ~/Downloads/lec31_code
siting@siting-ThinkPad-T14p-Gen-1:~/Downloads/lec31_code$ ./hello_world
Hello World from thread = 2
Hello World from thread = 8
Hello World from thread = 5
Hello World from thread = 10
Hello World from thread = 12
Hello World from thread = 15
Hello World from thread = 19
Hello World from thread = 18
Hello World from thread = 0
Number of threads = 20
Hello World from thread = 4
Hello World from thread = 3
Hello World from thread = 16
Hello World from thread = 7
Hello World from thread = 1
Hello World from thread = 17
Hello World from thread = 13
Hello World from thread = 6
Hello World from thread = 9
Hello World from thread = 14
Hello World from thread = 11
siting@siting-ThinkPad-T14p-Gen-1:~/Downloads/lec31_code$
```

OpenMP: Threads

- OpenMP creates as many threads as specified in the environment variable `OMP_NUM_THREADS`.
 - Set this variable to max number of threads you want to use
 - Generally, default: ($\#$ physical cores) * ($\#$ threads/core). Use `lscpu` command to obtain the numbers (e.g. 6 physical cores * 2 threads/core = **12 threads**)
- OpenMP threads are OS (software) threads, which are then multiplexed onto available hardware threads.

OpenMP Intrinsic	Description
<code>omp_set_num_threads(x);</code>	Set number of threads to x.
<code>num_th = omp_get_num_threads();</code>	Get number of threads.
<code>th_ID = omp_get_thread_num();</code>	Get Thread ID number.

lscpu

```
siting@siting-ThinkPad-T14p-Gen-1: ~/Downloads
siting@siting-ThinkPad-T14p-Gen-1:~/Downloads$ lscpu
架构:                x86_64
CPU 运行模式:        32-bit, 64-bit
Address sizes:        46 bits physical, 48 bits virtual
字节序:                Little Endian
CPU:                  20
在线 CPU 列表:        0-19
厂商 ID:              GenuineIntel
型号名称:             13th Gen Intel(R) Core(TM) i9-13900H
CPU 系列:             6
型号:                 186
每个核的线程数:      2
每个座的核数:        14
座:                   1
步进:                 2
CPU(s) scaling MHz:  39%
CPU 最大 MHz:         2600.0000
CPU 最小 MHz:         400.0000
BogoMIPS:             5990.40
标记:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx f
xsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monit
or ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt
tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb ssbd i
brs ibpb stibp ibrs_enhanced tpr_shadow flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1
avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb intel_pt sha_ni xsaveopt xsavec xg
etbv1 xsaves split_lock_detect user_shstk avx_vnni dtherm ida arat pln pts hwp hwp_notify hwp_
act_window hwp_epp hwp_pkg_req hfi vnmi umip pku ospke waitpkg gfni vaes vpclmulqdq tme rdpid
movdiri movdir64b fsrm md_clear serialize pconfig arch_lbr ibt flush_l1d arch_capabilities
```

OpenMP: Shared/Private Variables

- **Shared variables:** all threads read/write the same variable.
 - Variable declared outside of parallel region
 - Heap-allocated variables
 - Static variables
- **Private variables:** Each thread has its own copy of the variable.
 - Variables declared inside parallel region (recall separate stack frames)

```
int var1, var2;
char *var3 = malloc(...);
#pragma omp parallel private(var2)
{
    int var4;
    // var1 shared (default)
    // var2 private
    // var3 shared (heap)
    // var4 private (thread's stack)
    ...
}
```

Example

```
#include <stdio.h>
#include <omp.h>
int main() {
    /* Fork team of threads with private variable tid */
    #pragma omp parallel
    {
        int tid = omp_get_thread_num(); /* get thread id */
        printf("Hello World from thread = %d\n", tid);
        /* Only main thread does this */
        if (tid == 0) {
            printf("Number of threads = %d\n",
                omp_get_num_threads());
        }
    } /* All threads join main and terminate */
    return 0;
}
```

Private
variable

Parallel region
executed by each
subthread (with
OpenMP API)

OpenMP: Thread-Based Parallelism

- 1. Explicit programming model with full programmer control over parallelization
- Pros:
 - Compiler directives are simple and easy to use;
 - Legacy serial code does not need to be rewritten;
- Cons:
 - Compiler must support OpenMP (e.g. gcc 4.2);
 - Amdahl's law is gonna get you after not too many cores...;
- 2. Multiple threads in a **shared memory environment**
- Pros:
 - Takes advantage of shared memory, programmer need not worry (that much) about data placement;
- Cons:
 - Code can only be run in shared memory environments;
 - **Synchronizing use of shared resources is hard** (more later).

Parallelizing Loop Work

- Problem: You have to do some work over an array of 2^{27} numbers, with 8 people. How would you split the work?
- Assumptions
 - We need to decide before running the code!
 - Each element of the array is independent, so the tasks can be done in any order
 - The threads are about equally fast, so we want to assign each of them ~ 11 million numbers

Which Runs Fastest?

```

/* A. */
#pragma omp parallel
{
  for (int i = 0;
       i < LENGTH;
       i++) {
    arr[i] = ...;
  }
}

```

Duplicates
work

```

/* B. */
#pragma omp parallel
{
  int tid = omp_get_thread_num();
  int num_threads = omp_get_num_threads();
  int thread_start = tid*LENGTH/num_threads;
  int thread_end = (tid+1)*LENGTH/num_threads;
  for (int i = thread_start;
       i < thread_end; i++) {
    arr[i] = ...;
  }
}

```

“Chunks” array
sections

```

/* C. */
#pragma omp parallel
{
  int tid = omp_get_thread_num();
  int num_threads = omp_get_num_threads();
  for (int i = tid;
       i < LENGTH;
       i += num_threads) {
    arr[i] = ...;
  }
}

```

“Interweaves”
array access
between threads

```

/* D. */
#pragma omp parallel
{
  #pragma omp for
  for (int i = 0;
       i < LENGTH;
       i++) {
    arr[i] = ...;
  }
}

```

Like C, but
planned via
OpenMP