CS 33: Introduction to Computer Organization

TA: Aalisha Dalal
LA: Jonathan Myong
Office Hours: Friday, 9:30-11:30AM

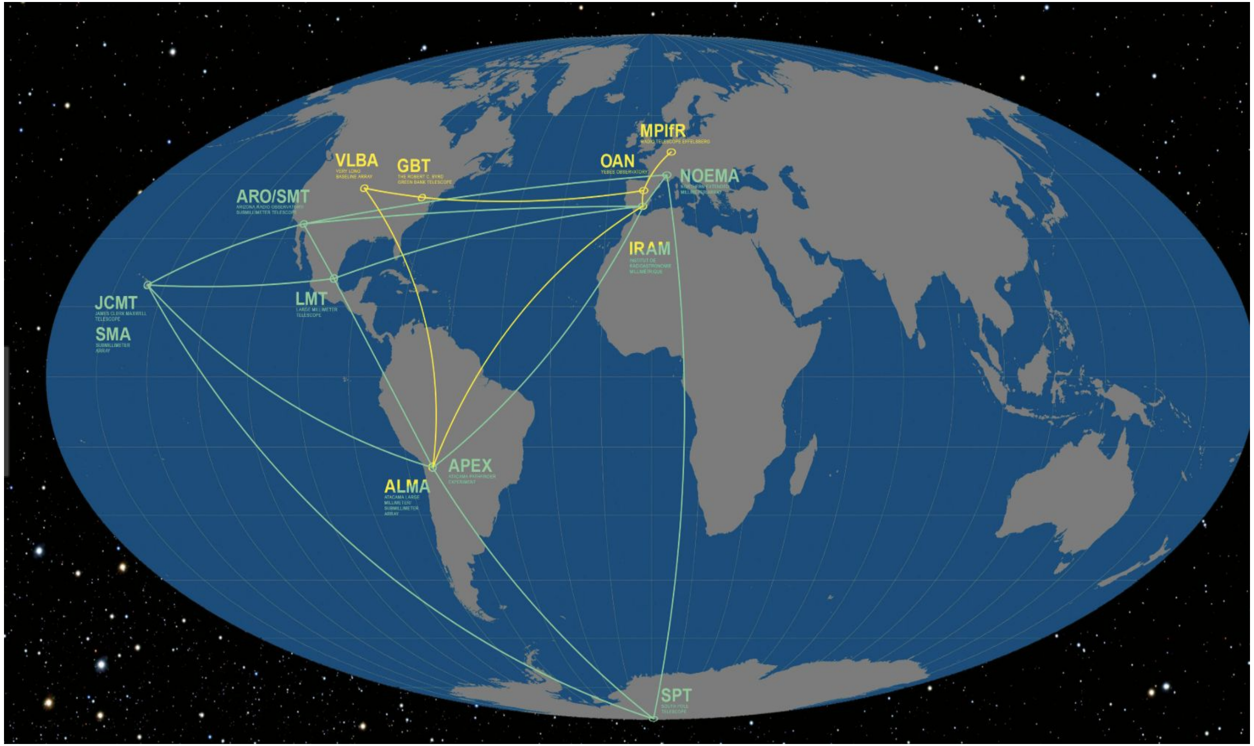
Outline

- Parallel Computing/ High Performance Computing
- Using OpenMP
- Worksheet Problems
- Midterm Papers

Why Parallel Computing?

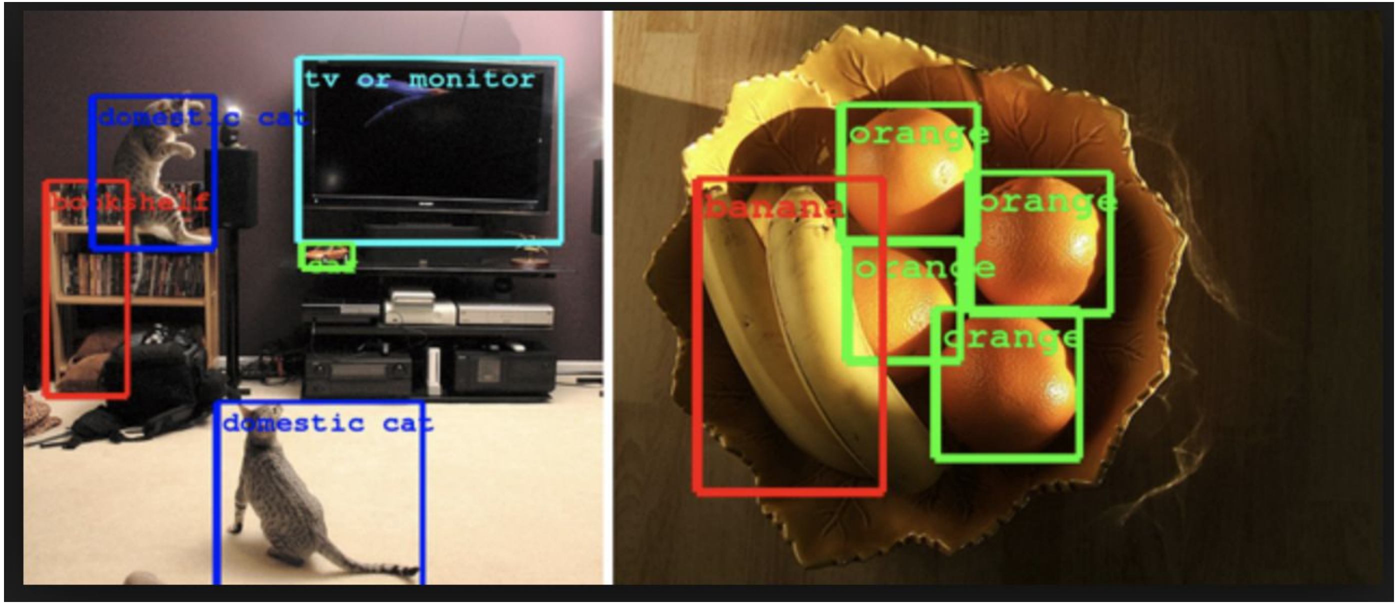
“Faster computation by dividing the tasks among multiple processors instead of one.”

Motivation - To know our universe better!



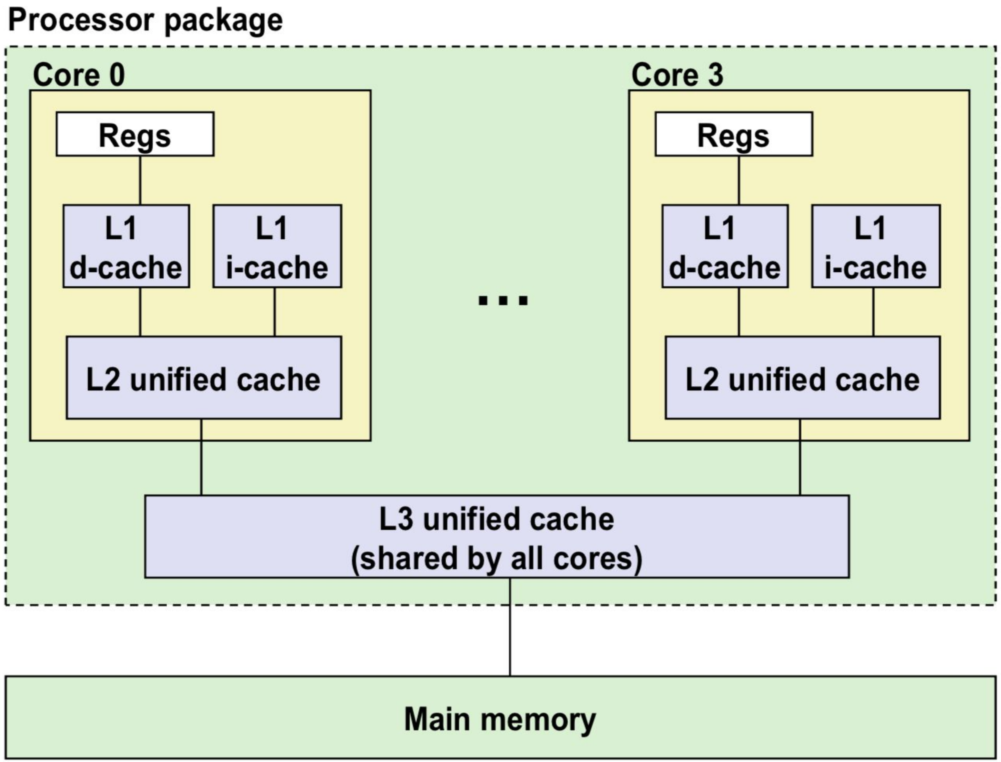
Capturing the image of the black hole!

Motivation - In the Age of ML/AI



Task of Object Detection using CNN models and many more ....

Parallel Architecture

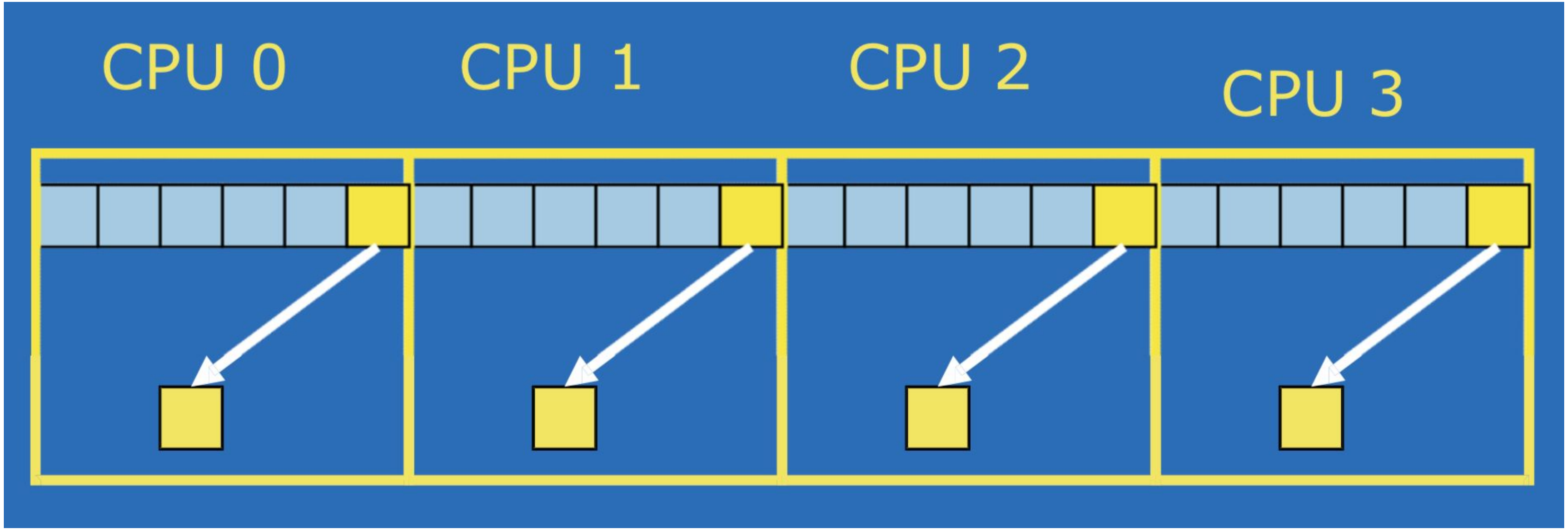


Parallelism

- *Domain Decomposition*: dividing data among processors
- *Task Decomposition:* dividing the tasks/ operations among different processors
- *Pipelining*

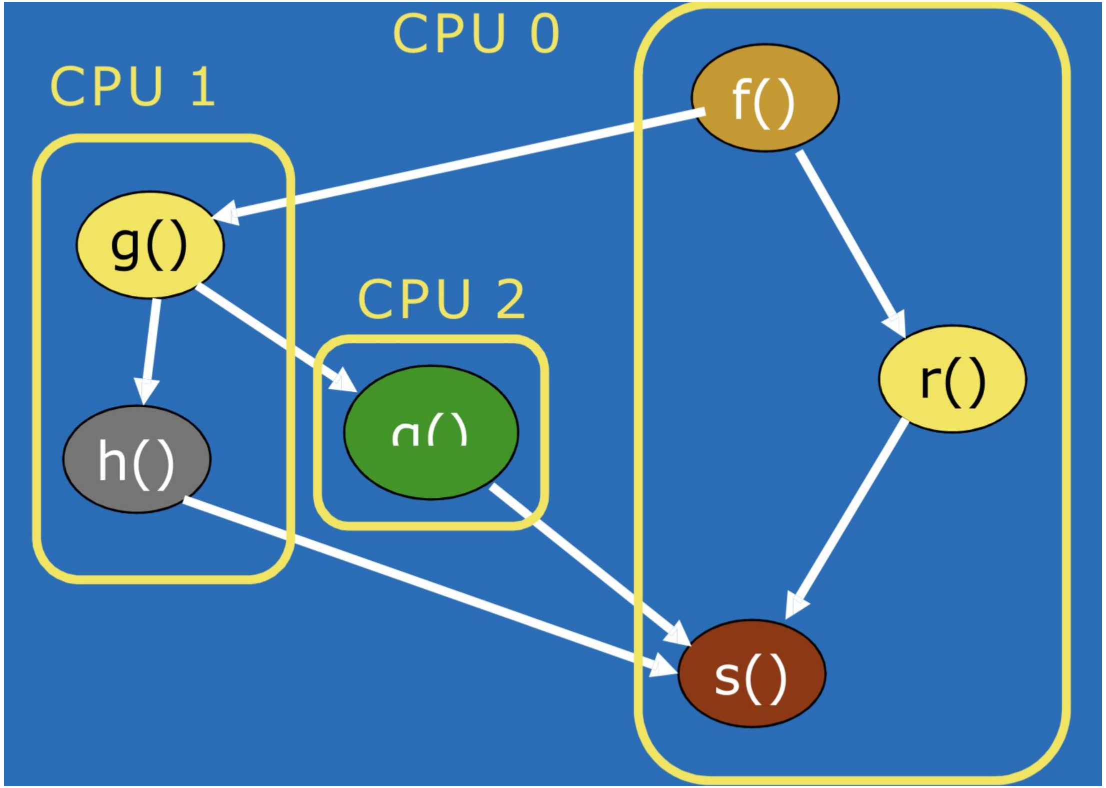
Domain Decomposition

Finding the largest element in the array



Task Decomposition

Dividing the method calls among different CPU cores



Dependency structure

- *For parallelizing a code, it is crucial to understand the dependency graph structure.*

```
for (i = 0; i < 3; i++)
   a[i] = b[i] / 2.0;
```

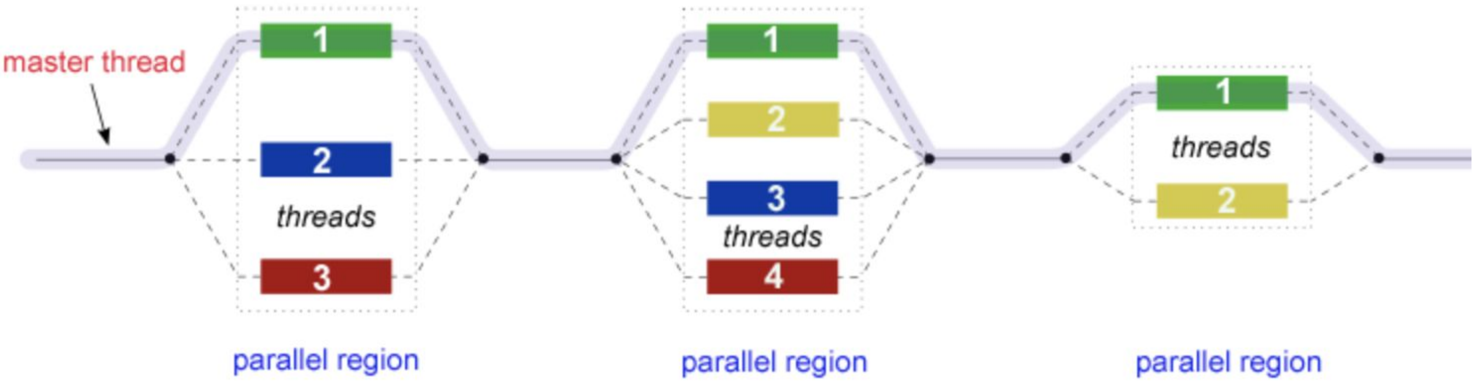
Loop iteration independent

Loop iteration dependent

```
for (i = 1; i < 4; i++)
   a[i] = a[i-1] * b[i];
```

OpenMP

- API developed in C especially designed for programming using shared multi-processors
- Based on the ***Fork-Join*** Thread model



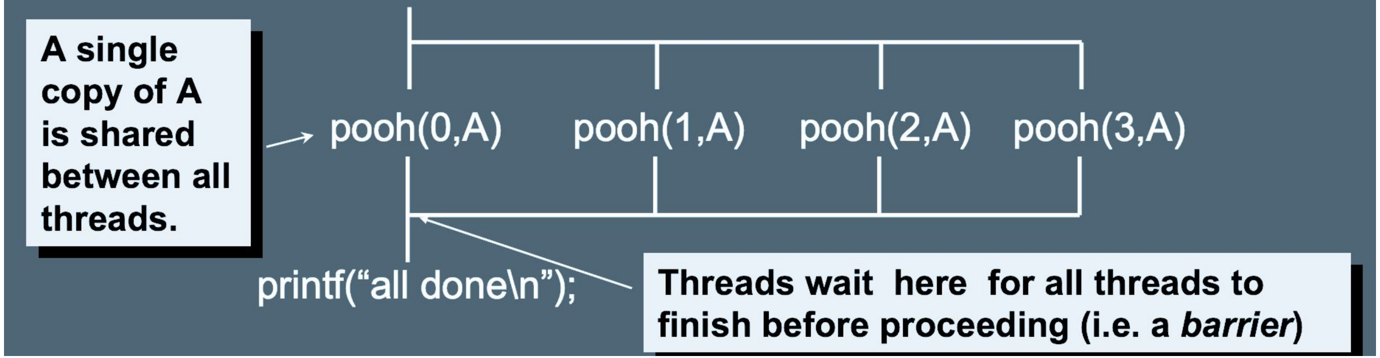
OpenMP Thread Creation

- **Each thread executes the same code redundantly.**

```
double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
    int ID = omp_get_thread_num();
    pooh(ID, A);
}
printf("all done\n");
```

double A[1000];

omp_set_num_threads(4)



OpenMP Example

What does this do?

```
#pragma omp parallel for
for (i=0; i < numPixels; i++)
{
   pGrayScaleBitmap[i] = (unsigned BYTE)
              (pRGBBitmap[i].red * 0.299 +
               pRGBBitmap[i].green * 0.587 +
               pRGBBitmap[i].blue * 0.114);
}
```

OpenMP Data Dependency Example

```
#pragma omp parallel for
for (i=2; i < 10; i++)          {
    factorial[i] = i * factorial[i-1];
}
```

Data Dependency leads to inconsistency when parallelizing code

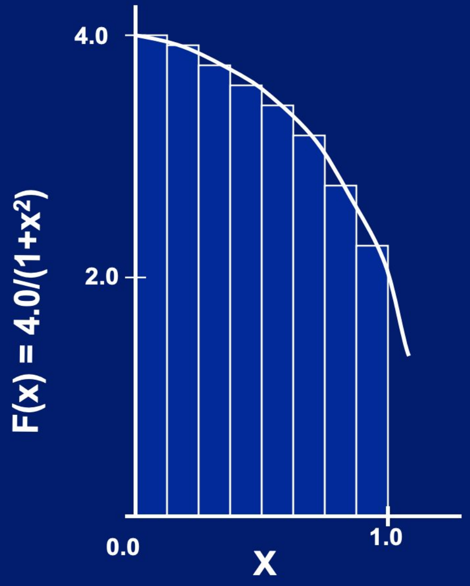
OpenMP Pragma Example

```
#pragma omp parallel
{
    #pragma omp for nowait
    for (i = 0; i < N; i++)
        a[i] = alpha(i);
    #pragma omp single nowait
    if (delta < 0.0) printf ("delta < 0.0\n");
    #pragma omp for
    for (i = 0; i < N; i++)
        b[i] = beta (i, delta);
}
```

What is the difference between “single” keyword, “serial code” and critical region?

OpenMP Example - Computing Pi

Numerical Integration



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\, dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

OpenMP Example - Computing Pi (contd.)

```
double area, pi, x;
int i, n;
...
area = 0.0;
for (i = 0; i < n; i++) {
    x = (i + 0.5)/n;
    area += 4.0/(1.0 + x*x);
}
pi = area / n;
```

What would be the issue if we parallelize the “**for**” loop?

OpenMP Example - Computing Pi (contd.)

```
double area, pi, x;
int i, n;
...
area = 0.0;
for (i = 0; i < n; i++) {
    x = (i + 0.5)/n;
    area += 4.0/(1.0 + x*x);
}
pi = area / n;
```

What would be the issue if we parallelize the "**for**" loop?

- RACE CONDITION

Possible Solution

```
double area, pi, tmp, x;
int i, n;
...
area = 0.0;
#pragma omp parallel private(tmp)
{
   tmp = 0.0;
#pragma omp for private (x)
   for (i = 0; i < n; i++) {
      x = (i + 0.5)/n;
      tmp += 4.0/(1.0 + x*x);
   }
#pragma omp critical
   area += tmp;
}
pi = area / n;
```

What did we do and why is this better?

Possible Solution - Reduction operator

```
double area, pi, x;
int i, n;
...
area = 0.0;
#pragma omp parallel for private(x) \
                         reduction(+:area)
for (i = 0; i < n; i++) {
    x = (i + 0.5)/n;
    area += 4.0/(1.0 + x*x);
}
pi = area / n;
```

This is a powerful technique!

Locking Mechanism

What happens if threads are at this point at the same time?

Thread A	Thread B
`lock (lock_a);`	`lock (lock_b);`
`a += 5;`	`b += 5;`
`lock (lock_b);`	`lock (lock_a);`
`b += 7;`	`a += 7;`
`a += b;`	`a += b;`
`unlock (lock_b);`	`unlock (lock_a);`
`a += 11;`	`b += 11;`
`unlock (lock_a);`	`unlock (lock_b);`

If there is an issue, how can we solve it?

Worksheet

https://tinyurl.com/cs33-parallel-universe

MidTerm Papers

Thank you!

Appendix

Threads vs. Processes

- **Threads and processes: similarities**
  - Each has its own logical control flow
  - Each can run concurrently with others
  - Each is context switched (scheduled) by the kernel
- **Threads and processes: differences**
  - Threads share code and data, processes (typically) do not
  - Threads are much less expensive than processes
    - Process control (creating and reaping) is more expensive as thread control
    - Context switches for processes much more expensive than for threads