iPDC: Summer Institute
Intro to Parallel Computing and OpenMP

September 12 -18, 2020

Department of Computer Science

Tennessee Tech University

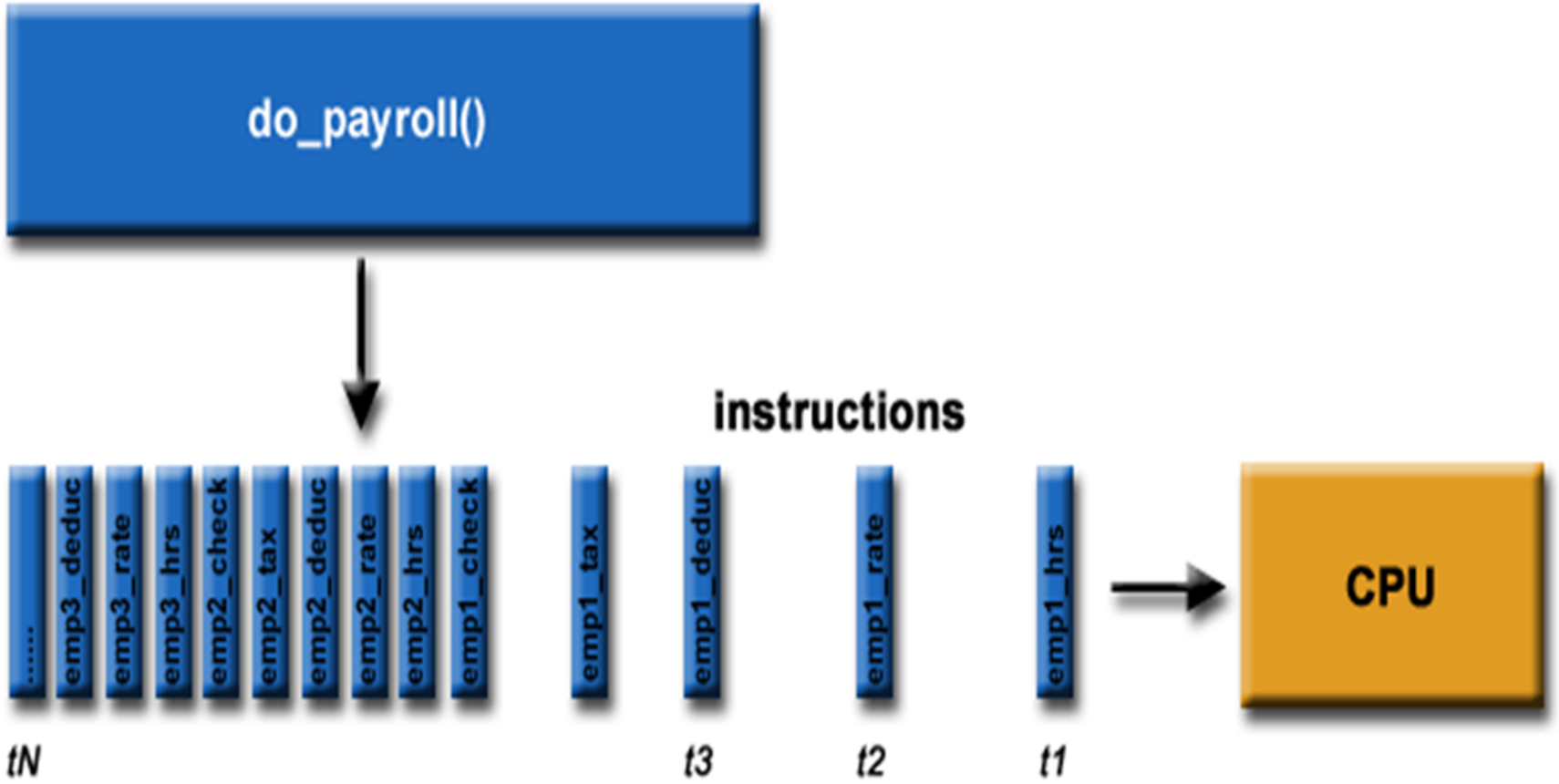
What is Parallel Computing?

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem
- Steps
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute in sequence on each processor but simultaneously on different processors
  - An overall control/coordination mechanism is employed

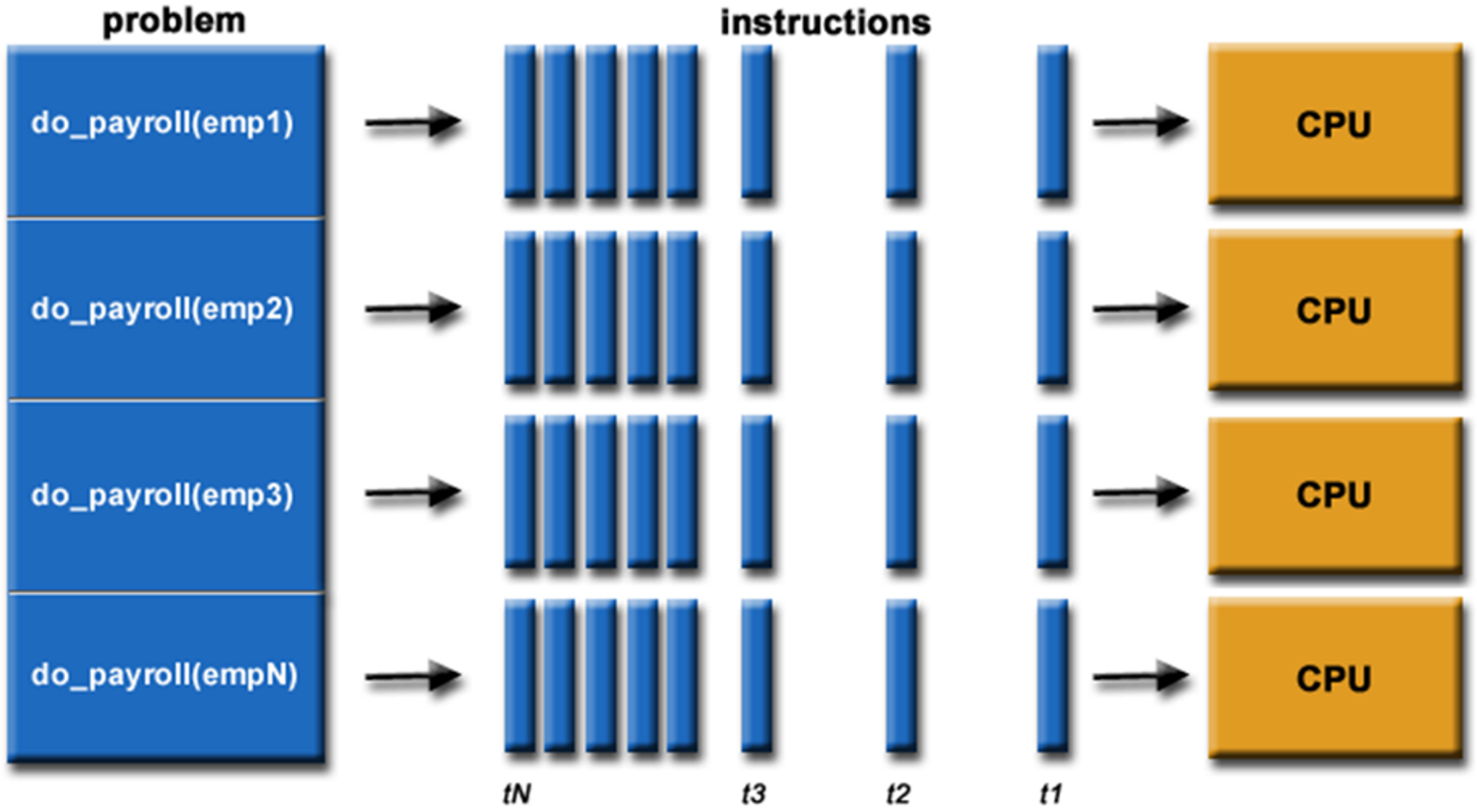
What is Parallel Computing

- The computational problem should be able to:
  - Be broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Be solved in less time with multiple compute resources than with a single compute resource.
- The compute resources might be:
  - A single computer with multiple processors
  - An arbitrary number of computers connected by a network
  - A combination of both

Sequential Computing

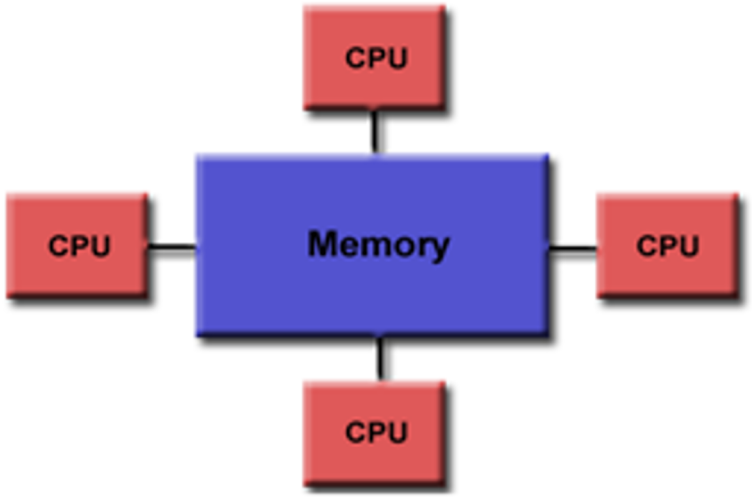


Parallel Computing



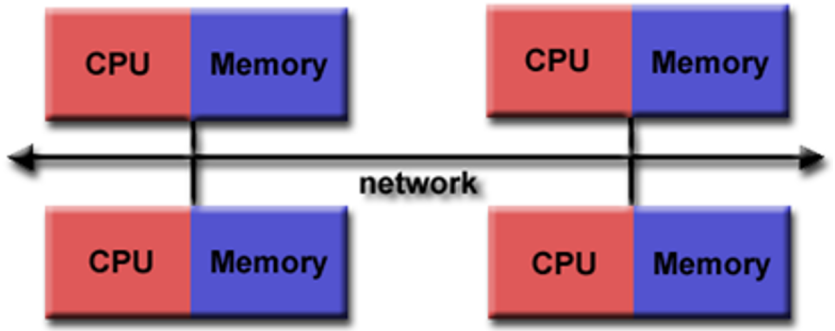
Shared Memory Architecture

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location affected by one processor are visible to all other processors.
- The most critical problem to address is that of cache coherence.



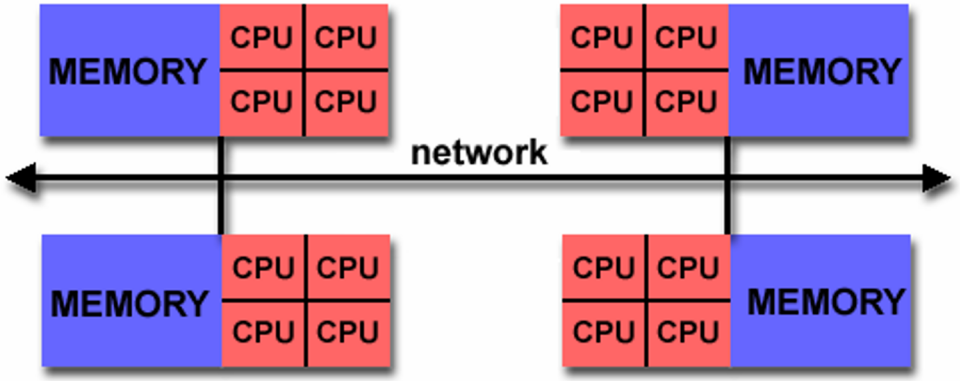
Distributed Memory Architecture

- Processors have their own local memory and runs their own copy of OS. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors
- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.



Hybrid Architecture

- The large computers in the world today employ both shared and distributed memory architectures.
- The shared memory component is usually a cache coherent SMP machine. Processors on a given SMP can address that machine's memory as global.
- The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.



Concurrency

Concurrency is a property of an algorithm, it exposes potential for parallelization. If concurrency is present in an algorithm then the concurrent operations can be executed in parallel (simultaneously) by multiple operation units (CPU's) if available, without concurrency there is no scope for parallelization. Concurrency can be present in a sequential program, parallelization takes advantage of concurrency to increase performance.

How Do We Write Parallel Programs

- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.
- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it's part of the data.

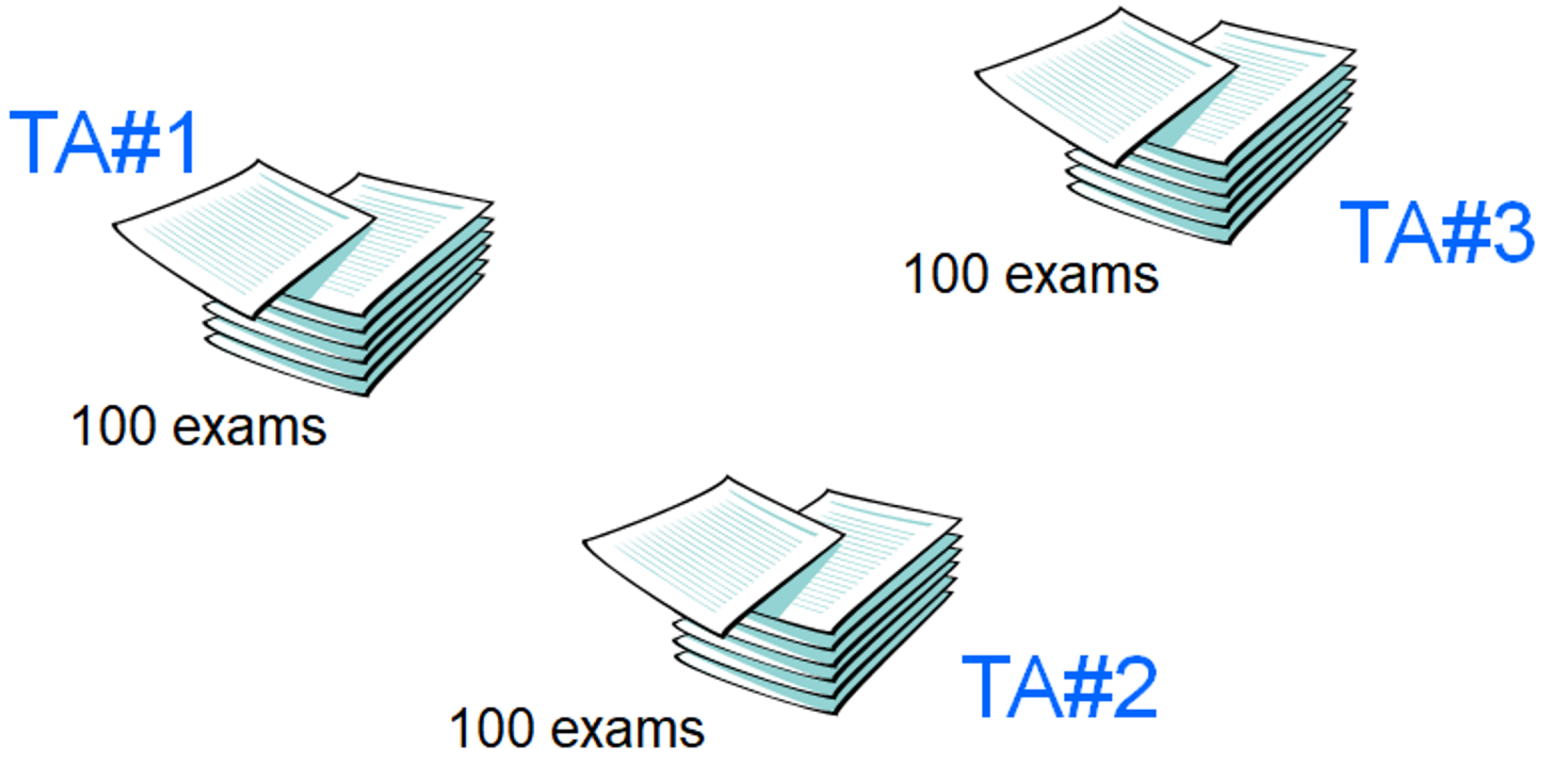
Professor S

15 questions

300 exams

Professor S's Teaching Assistants

Division of work – Data Parallelism



Division of Work – Task Parallelism

TA#1

Questions 1 - 5

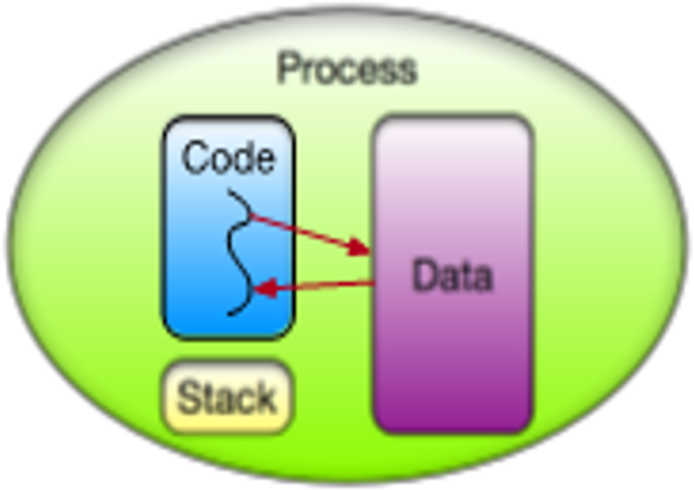
TA#3

Questions 11 - 15

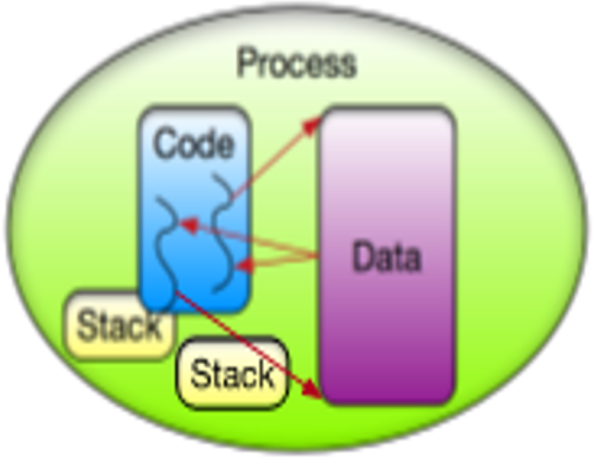
TA#2

Questions 6 - 10

Process and Thread



Process with one thread



Process with two threads

Activity 1: Hello World

Let’s write the multi-threaded version of Hello, World! in Java…

Activity 2: Vector Addition

Now let's write a parallel program in Java for vector addition. Assume A, B, C are three vectors of equal length. The program will add the corresponding elements of vectors A and B and will store the sum in the corresponding elements in vector C (in other words C[i] = A[i] + B[i]). Every thread should execute an approximately equal number of loop iterations.

The program should take n and the number of threads to use as command line arguments:

```
java ParallelVectorAdd <n> <threads>
```

where n is the length of the vectors and threads is the number of threads to be created. (You can fill the vectors A and B with random integers.)

Pseudocode for Vector Addition

```
mystart = myid*n/p
myend = mystart+n/p
for (i= mystart;   i<myend; i++
     do vector addition
```

Speed Up

- Speedup is a quantitative measure of how fast the program runs when more than one processor is used
- Number of cores = p
- Serial runtime = T_{serial}
- Parallel run-time = $T_{parallel}$

$$S = \frac{T_{serial}}{T_{parallel}}$$

Amdahl's Law

- Unless virtually all of a serial program is parallelized, the possible speedup is going to be very limited — regardless of the number of cores available.

Example

- Say we can parallelize 90% of a serial program.
- Assume the parallelization is “perfect” regardless of the number of cores *p* we use.
- T_{serial} = 20 seconds
- Then the runtime of the parallelizable part is

$$0.9 \times T_{serial} / p = 18 / p$$

Example

- The runtime of the “unparallelizable” part is
- $0.1 \times T_{serial} = 2$
- The overall parallel run-time is

$$T_{parallel} + T_{serial}$$

$$= 0.9 \times T_{serial} / p + 0.1 \times T_{serial} = (18 / p) + 2$$

$$S = \frac{T_{serial}}{0.9 \times T_{serial} / p + 0.1 \times T_{serial}} = \frac{20}{18 / p + 2}$$

Efficiency

- Efficiency is a quantitative measure of what fraction of the processors time are being used to actually solve the problem (how effectively the processors are being used)
- Number of cores = p
- Speedup = S

E = S/P

Scalability

- Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
  - Hardware - particularly memory-cpu bandwidth and network communications
  - Algorithm
  - Parallel overhead
  - Characteristics of your specific implementation

Scalability

In general, a problem is *scalable* if it can handle ever increasing problem sizes.

If we increase the number of processes/threads and keep the efficiency fixed without increasing problem size, the problem is *strongly scalable*.

If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is *weakly scalable*.

Scalability Behaviour of Parallel Systems

Figure 5.8. Speedup versus the number of processing elements for adding a list of numbers.