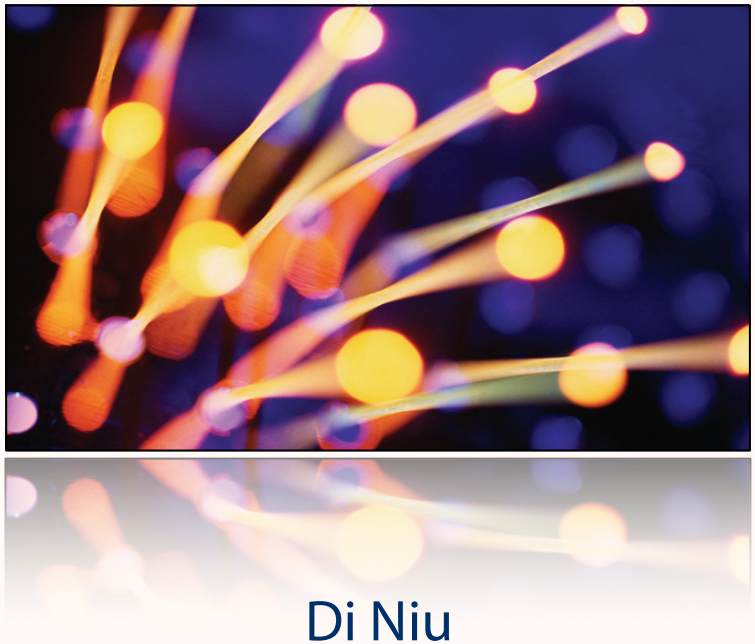
ECE 420 Parallel and Distributed Programming



Di Niu
Department of Electrical and Computer Engineering
University of Alberta

Prerequisites

Familiarity with C programming is necessary

Knowledge of the following is *required*

UNIX working environment: compiling, makefile, command line, VIM, Emacs

OS concepts: race conditions, critical sections, threads, memory hierarchy, concurrency

Version control and collaboration: git, svn.

Linear Algebra: matrix and vector operations

Algorithms: sorting, graph algorithms, trees, etc.

Course Topics (in approximate order)

Introduction

Why use parallel computing?

Parallel hardware: SIMD, MIMD, shared-memory systems (multicore processors), distributed-memory systems (clusters)

Parallel software: threads and shared memory, processes and message passing, distributed shared memory, distributed shared data

Performance: speedup and Amdahl's Law

Shared-Memory Programming with Pthreads

Processes, Threads and Pthreads

Critical sections and locks

Synchronization and semaphores

Barriers and condition variables

Course Topics (in approximate order)

Shared-Memory Programming with OpenMP

- Compilation and execution OpenMP programs
- Parallelizing for loops with OpenMP
- Case study: parallel sorting

Distributed-Memory Programming with MPI

- Getting started: compilation and execution
- Communication patterns
- Data and work partitioning
- Case study: parallel sorting

Course Topics (in approximate order)

Parallel Algorithm Design

- Data and work partitioning
- Parallelization strategies
- Granularity & Communication Patterns
- Load balancing

Big Data Processing and Spark

- Studying research papers
- Hadoop File System (HDFS)
- Scala Functional Programming
- Streaming Analytics
- Machine Learning and Batch Jobs

Course Work and Evaluation

Lab 1: Warm-up with Linux and Pthreads	**5%**
Lab 2: Pthreads	**10%**
Lab 3: OpenMP	**10%**
Lab 4: MPI	**10%**
Lab 5: Spark	**5%**
Assignments 1, 2, 3	**3% each**
Midterms 1, 2, 3 (non-recursive)	**17% each**

The assignments will contain questions that are representative of exam questions.

Labs (the major component of this course)

Very important: Count as 40% in total.

Work in teams of 3

Lab 1 starts soon

Due Dates and Deliverables required for each lab on eclass

Specific due dates are posted on eClass

Deliverables include

a report and code to be submitted on eclass

Evaluation criterion will be outlined on each lab manual

Academic Integrity

All forms of dishonesty are unacceptable.

Any offence will be reported to the Associate Dean.

Cheating, plagiarism and misrepresentation are serious.

Anyone who engages in these will receive *at least* a grade of 0.

In Faculty of Engineering, the sanction for cheating will include a disciplinary failing grade (no exceptions).

Senior students should expect a period of suspension or expulsion from the University of Alberta.

What is Parallel Computing?

Serial computation

A problem is broken into a discrete series of instructions

Instructions are executed sequentially one after another

Executed on a single processor

Only one instruction may execute at any moment in time

Parallel computation

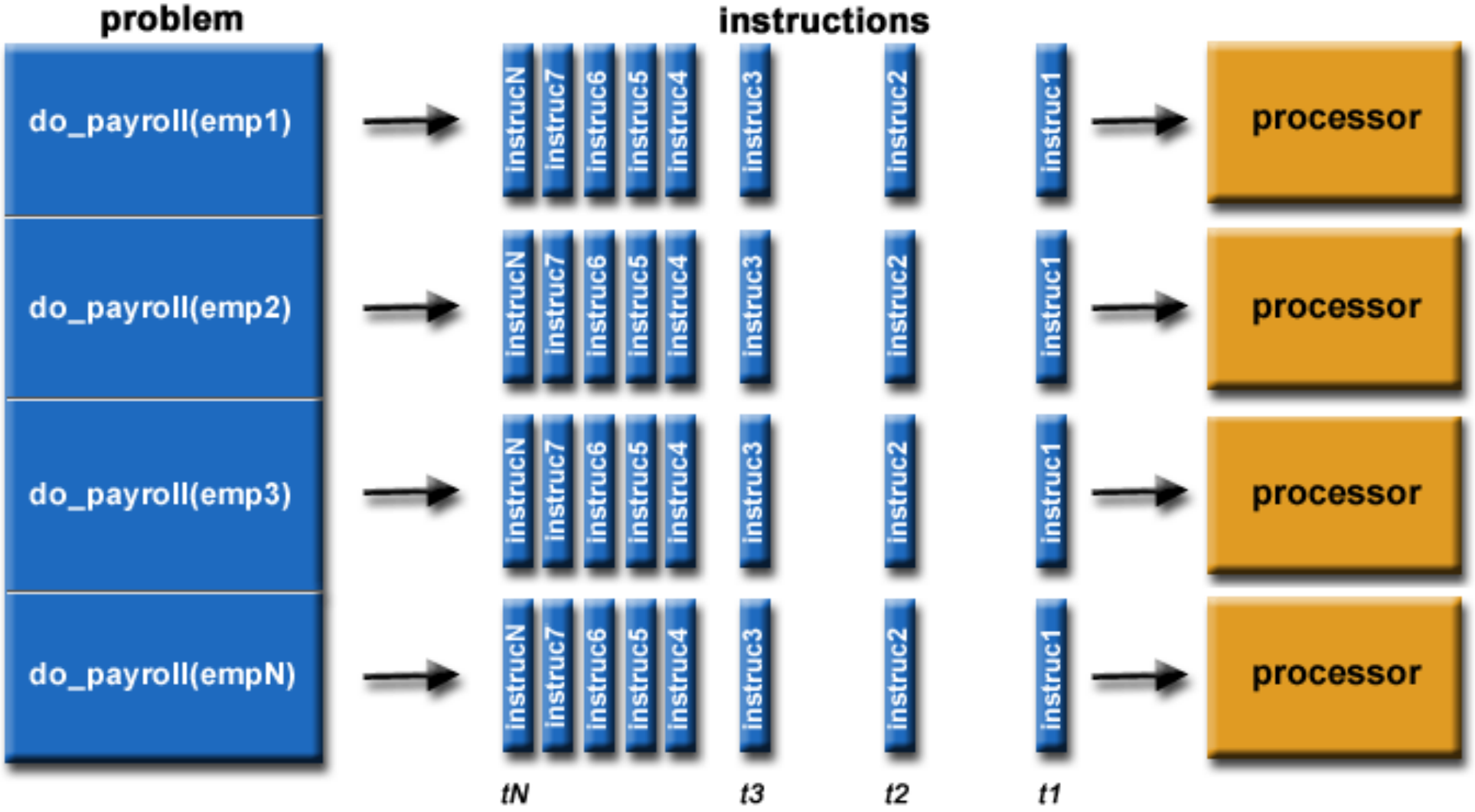
A problem is broken into discrete parts that can be solved concurrently

Each part is further broken down to a series of instructions

Instructions from each part execute simultaneously on different processors

An overall control/coordination mechanism is employed

Parallel Computing Example



Why Parallel Computing?

Need performance for problems that

require too much computation

use big data

In the natural world, many complex events are happening at the same time, yet within a temporal sequence

Galaxy Formation

Planetary Movments

Climate Change

Why Parallel Computing?

Need performance for problems that

require too much computation

use big data

In real-life, we have the pipelined model, which is essentially parallel computing

Auto Assembly

Jet Construction

Drive-thru Lunch

Why Parallel Computing?

Supercomputer compares modern and ancient DNA

With genetic tools, supercomputing simulations and modeling, they traced the origins of modern Europeans to three distinct populations.

Published in the journal *Nature*.

http://top500.org/blog/supercomputer-compares-modern-and-ancient-dna/

Oil and gas exploration

Finite-difference 3D modeling of seismic wave propagation through the subsurface.

Seismic (geophysical) imaging

Using multi-core CPU and NVIDIA GPU

http://www.acceleware.com

Why Should We Use Parallel Programming?

The need to save time or money

The need to solve large and complex problems

Sometimes, concurrency is a must

- Web servers, multi-threaded OS and applications, e.g., Skype

Take advantage of non-local resources

- Crowdsourcing: divides work between Internet users
- Crowdsensing: Google Map, Smart City Applications
- Cloud Computing: Amazon, Compute Canada

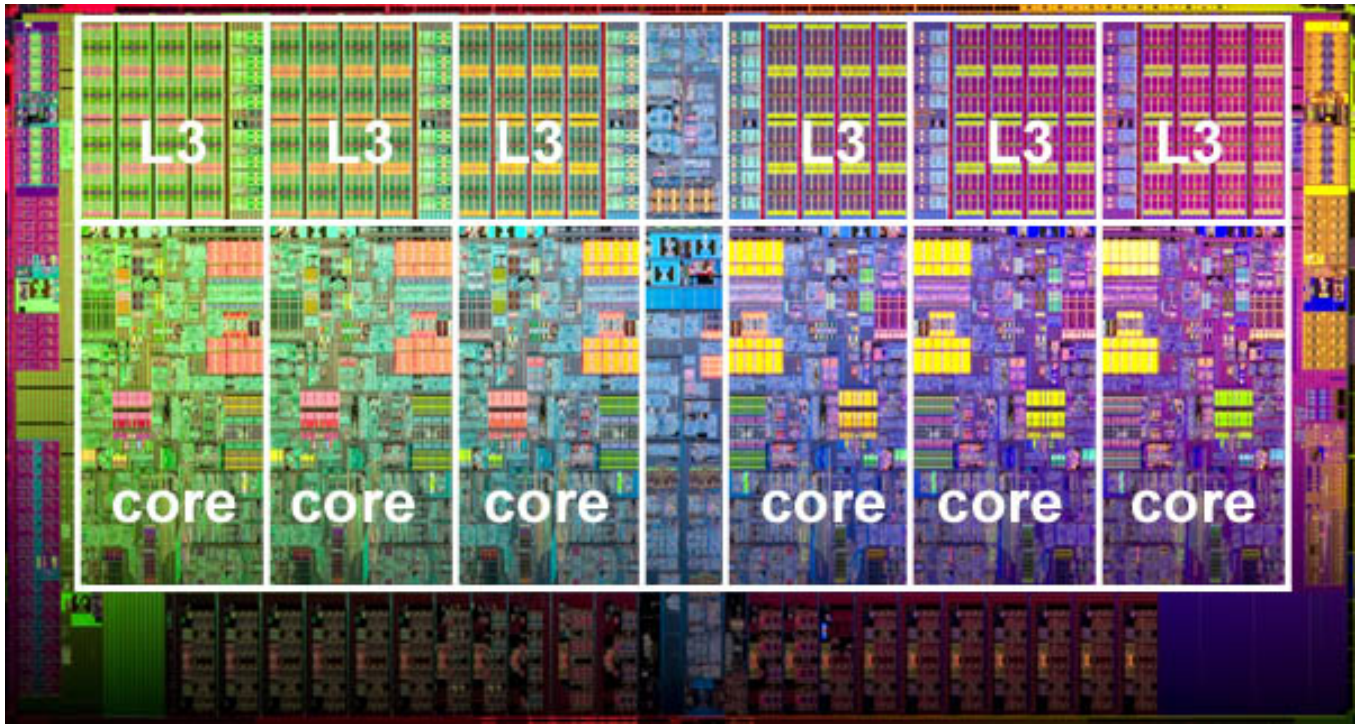
Why Should We Use Parallel Programming?

Make better use of commodity parallel hardware

Modern computers, even laptops, have multiple processors/cores.

In most cases, serial programs "waste" computing power.

GPUs (NVIDIA, ATI): supercomputer on a desktop.



Why Should We Use Parallel Programming?

Process a large amount of data

Big Data are generated or stored on different commodity computers .

But we need to operate on the entire dataset for, e.g.,

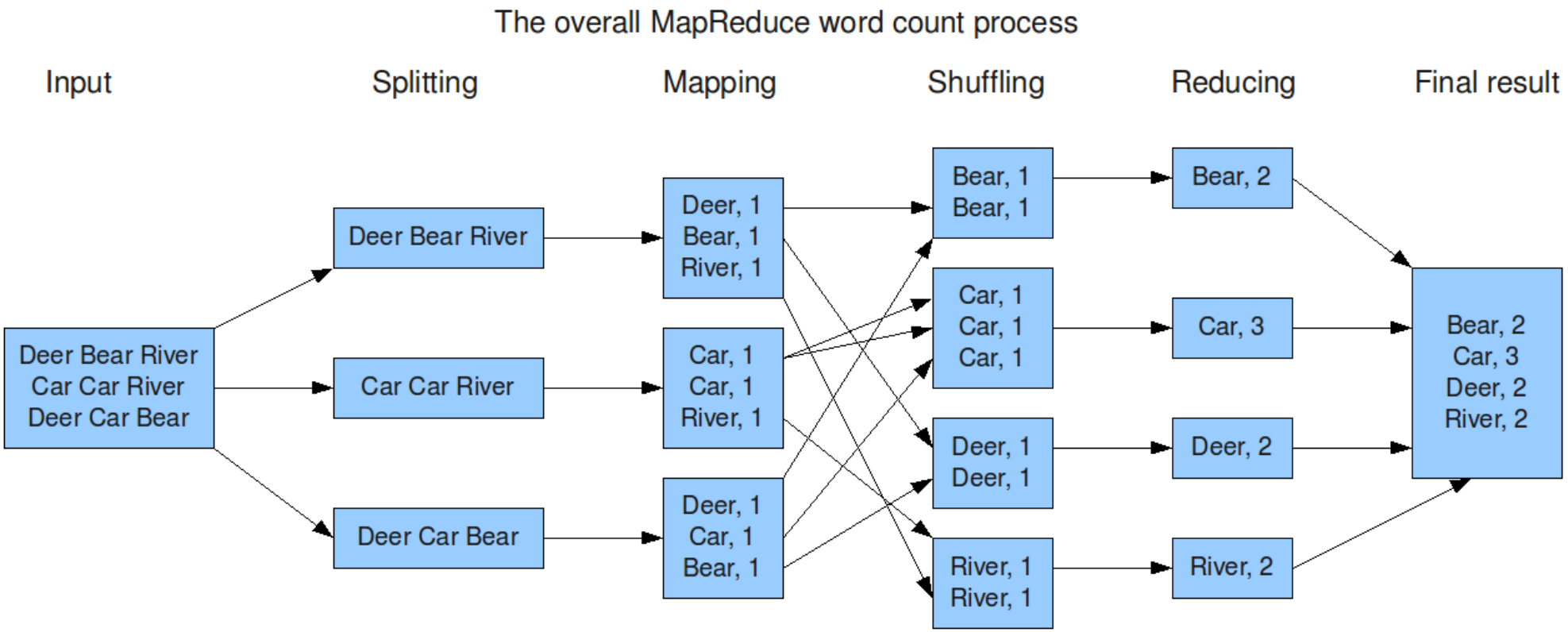
Text Search

Word Count

Classification (e.g. Spam vs. non-Spam)

Why Should We Learn Parallel Programming?

Word Count by MapReduce



How Hard is Parallel Programming?

A well behaved single processor algorithm may behave poorly on a parallel computer, and may need to be reformulated

There is *no* magic compiler that can turn a serial program into an efficient parallel program *all the time* and on *all machines*

It's all about performance.

Concurrent, Parallel, Distributed

Concurrent computing: a program in which multiple tasks can be in progress at any instant. (even for one core)

A multitasking OS

Parallel computing: a program in which multiple tasks *cooperate* closely to solve a problem

Tightly coupled, running on the same machine

Distributed computing: a program cooperates with other programs to solve a problem

Loosely coupled, running on multiple machines

Parallel vs. Distributed

Parallel computing: shared memory

- multiprocessors (important trend)
- processes share logical address spaces
- processes share physical memory
- sometimes refers to the study of parallel algorithms

Distributed computing: distributed memory

- clusters and networks of workstations
- processes do not share address spaces
- processes do not share physical memory
- sometimes refers to the study of theoretical distributed algorithms