Parallel Programming
in C with MPI and OpenMP

Based on slides by Michael J. Quinn

Chapter 1

Motivation and History

Outline

- Motivation
- Modern scientific method
- Evolution of supercomputing
- Modern parallel computers
- Seeking concurrency
- Data clustering case study
- Programming parallel computers

Why Faster Computers?

- Solve compute-intensive problems faster
  - Make infeasible problems feasible
  - Reduce design time
- Solve larger problems in same amount of time
  - Improve answer's precision
  - Reduce design time
- Gain competitive advantage

Definitions

- Parallel computing
  - Using parallel computer to solve single problems faster
- Parallel computer
  - Multiple-processor system supporting parallel programming
- Parallel programming
  - Programming in a language that supports concurrency explicitly

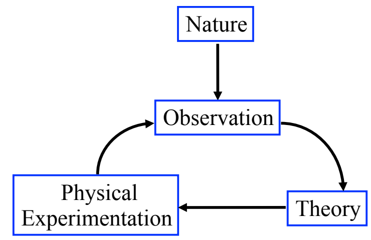
Why MPI?

- MPI = "Message Passing Interface"
- Standard specification for message-passing libraries
- Libraries available on virtually all parallel computers
- Free libraries also available for networks of workstations or commodity clusters

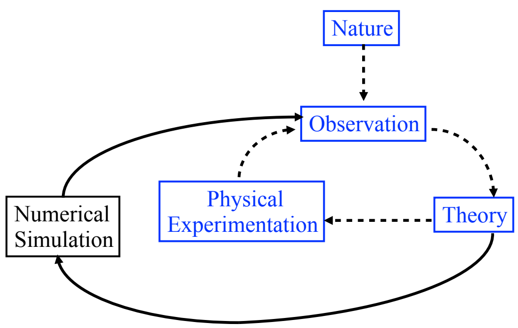
Why OpenMP?

- OpenMP an application programming interface (API) for shared-memory systems
- Supports higher performance parallel programming of symmetrical multiprocessors

Classical Science



Modern Scientific Method



Evolution of Supercomputing

- World War II
  - Hand-computed artillery tables
  - Need to speed computations
  - ENIAC
- Cold War
  - Nuclear weapon design
  - Intelligence gathering
  - Code-breaking

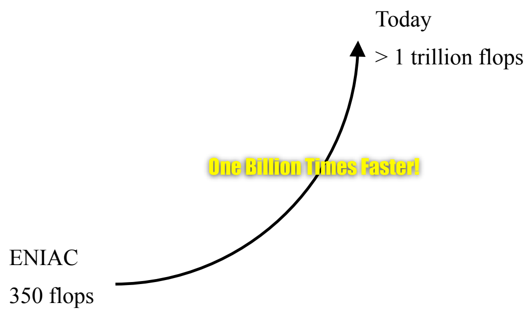
Supercomputer

- General-purpose computer
- Solves individual problems at high speeds, compared with contemporary systems
- Typically costs $10 million or more
  - Large UIUC *Teragrid* machine in the early 90s was ~$150 million
  - Building, etc., was > $100 million more
  - Cooling, etc. cost money too!
- Traditionally found in government labs

Commercial Supercomputing

- Started in capital-intensive industries
  - Petroleum exploration
  - Automobile manufacturing
- Other companies followed suit
  - Pharmaceutical design
  - Consumer products

50 years of speed increases

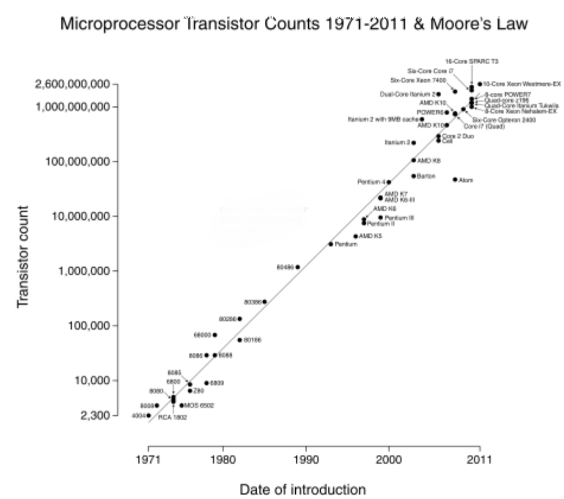


CPUs 1 Million Times Faster

- Faster clock speeds
- Greater system concurrency
  - Multiple functional units
  - Concurrent instruction execution
  - Speculative instruction execution

Systems 1 Billion Times Faster

- Processors are 1 million times faster
- Combine thousands of processors
- Parallel computer
  - Multiple processors
  - Supports parallel programming
- Parallel computing = Using a parallel computer to execute a program faster



Older Parallel Computers

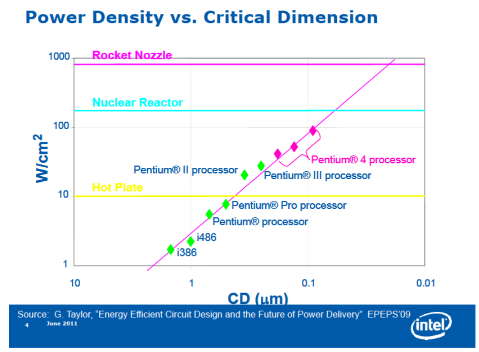
- Caltech's Cosmic Cube (Seitz and Fox)
- Commercial copy-cats
  - nCUBE Corporation
  - Intel's Supercomputer Systems Division
  - Lots more
- Thinking Machines Corporation
- CDC machines

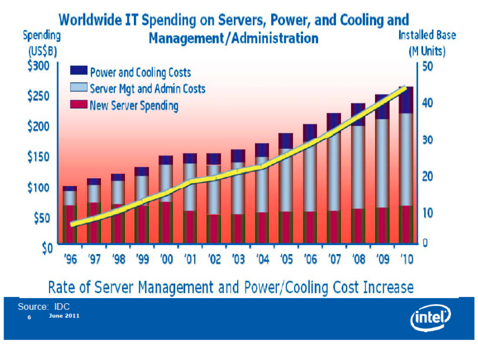
Modern Parallel Computers

- *Every multi-core microprocessor is a parallel computer*
- IBM's Blue Gene and other systems with custom network hardware
- Large systems built from relatively commodity hardware (Purdue's Conte, 2nd most powerful university machine in the world)
- Clusters of workstations
- GPU and accelerator based systems

Why multicores?

- Intel did *NOT* want to do multicores
- Microsoft did *NOT* want to deal with multicores
- Then why are essentially all general purpose processors multicores?
  - *Heat dissipation and power consumption are the reasons*





Multicore driven by heat

- Power increases as the square of the clock cycle
- smaller features cause leakage currents, which is exacerbated by higher voltages needed to drive faster clock cycles
- How to get more cycles/unit time out of processors without increasing clock rates?
  - add more functional units
    - Instruction level parallelism has limits and parallelism the responsibility of the *processor*
    - Thus it is Intel's problem and fundamental limits make it a hard to impossible problem to solve
  - add more functional units in independent cores
    - Need multithreaded programming
    - performance is the *programmer's* responsibility!
- *Move to mobile devices and laptops has exacerbated these trends!*

Forces a new economic and development paradigm

- ***In the old model***
  - processors got faster
  - Microsoft and application developers wrote more feature-rich and largely sequential programs that soaked up cycles
  - Targeted next generation processors
  - Release of new software motivated replacement of desktops
  - Intel sells processor, Microsoft sells software, programmers write code like always, everyone is happy

Forces a new economic and development paradigm

- ***In the new model***
  - processors get more cores
  - Microsoft and application developers have to write increasingly parallel programs to implement new features with good performance
  - This is hard to do!
  - Stretches out software cycle
  - Stretches out desktop replacement cycle (has gone from 3 to 3.5 - 5 or 6 years, depending on the institution)
  - Microsoft, Intel and demand for programmers suffers
  - Academics doing parallelism research are happy!

Copy-cat Strategy

- Microprocessor
  - 1% speed of supercomputer
  - 0.1% cost of supercomputer
- Parallel computer = 1000 microprocessors
  - 10 x speed of supercomputer
  - Same cost as supercomputer
  - Flynn – attack of the killer micros

Why Doesn't Everybody Buy One?

- Supercomputer ≠ Σ CPUs
  - Computation rate ≠ throughput
  - Inadequate I/O
- Software
  - Inadequate operating systems
  - Inadequate programming environments

After the "Shake Out"

- IBM
- Cray
- HP
- Silicon Graphics
- Dawning
- Bull
- Nvidia
- ~~Sun Microsystems~~ Oracle
- World's fastest supercomputer (33.86 petaflop/s) developed by the National University of Defense Technology

Commercial Parallel Systems

- Relatively costly per processor
- Primitive programming environments
- Focus on commercial sales
- Scientists looked for alternative
- This situation has been improving, in part because of larger importance of scientific computing, in part because of commoditization driven by standards, dominance of Intel

Beowulf Concept

- NASA (Sterling and Becker)
- Commodity processors
- Commodity interconnect
- Linux operating system
- Message Passing Interface (MPI) library
- High performance/$ for certain applications

Advanced Strategic Computing Initiative

- U.S. nuclear policy changes
  - Moratorium on testing
  - Production of new weapons halted
- Numerical simulations needed to maintain existing stockpile
- Five supercomputers costing up to $100 million each

Tianhe-2 (MilkyWay -2) Intel Xeon processors, 1.2 million cores, 33 Tflops, Kylin Linux

ASCI White (10 teraops/sec)

tera = 1 trillion

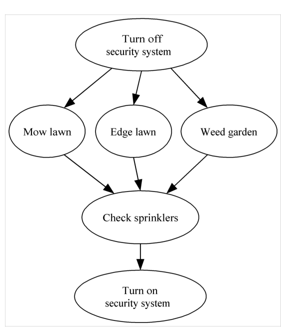
> 8 trillion calculations/second

Seeking Concurrency

- Data dependence graphs
- Data parallelism
- Functional parallelism
- Pipelining

Data Dependence Graph

- Directed graph
- Vertices = tasks
- Edges = dependences



Data Parallelism

- Independent tasks apply same operation to different elements of a data set

```
for i ← 0 to 99 do
   a[i] ← b[i] + c[i]
endfor
```

- Okay to perform operations concurrently

What can run in parallel?

Consider the loop:

```
for (i=1; i<n; i++) {
    a[i] = b[i] + c[i];
    c[i] = a[i-1]
}
```

Let each iteration execute in parallel with all other iterations on its own processor

time

i = 1	*i = 2*	*i = 3*
a[1] = b[1] + c[1]	a[2] = b[2] + c[2]	a[3] = b[3] + c[3]
c[1] = a[0]	c[1] = a[0]	c[3] = a[2]

Note that data is produced in one iteration and consumed in another.

What can run in parallel?

Consider the loop:

```
for (i=1; i<n; i++) {
    a[i] = b[i] + c[i];
    c[i] = a[i-1]
}
```

time

i = 3
a[3] = b[3] + c[3]
c[3] = a[2]

i = 2
a[2] = b[2] + c[2]
c[1] = a[0]

What if the processor executing iteration i=2 is delayed for some reason? *Disaster* - the value of a[2] to be read by iteration i=3 is not ready when the read occurs!

Cross-iteration dependences

Consider the loop:

```
for (i=1; i<n; i++) {
    a[i] = b[i] + c[i];
    c[i] = a[i-1]
}
```

Orderings that must be enforced to ensure the correct order of reads and writes are called *dependences.*

time

i = 1	*i = 2*	*i = 3*
a[1] = b[1] + c[1]	a[2] = b[2] + c[2]	a[3] = b[3] + c[3]
c[1] = a[0]	c[1] = a[0]	c[3] = a[2]

A dependence that goes from one iteration to another is a *cross iteration*, or *loop carried* dependence

Cross-iteration dependences

Consider the loop:

```
for (i=1; i<n; i++) {
    a[i] = b[i] + c[i];
    c[i] = a[i-1]
}
```

Loops with cross iteration dependences cannot be executed in parallel unless mechanisms are in place to ensure dependences are honored.

time

i = 1	*i = 2*	*i = 3*
a[1] = b[1] + c[1]	a[2] = b[2] + c[2]	a[3] = b[3] + c[3]
c[1] = a[0]	c[1] = a[0]	c[3] = a[2]

We will generally refer to something as *parallel* or *parallelizable* if dependences do not span the code that is to be run in parallel.

Functional Parallelism

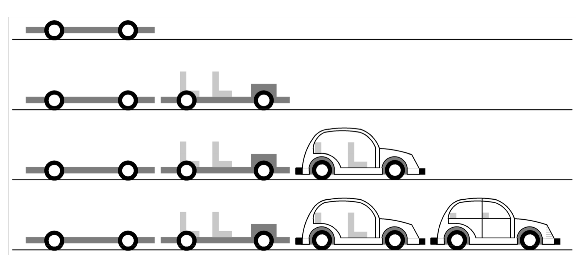
- Independent tasks apply different operations to different data elements

  a ← 2
  b ← 3
  m ← (a + b) / 2
  s ← (a^2 + b^2) / 2
  v ← s - m^2

- First and second statements can execute in parallel
- Third and fourth statements can execute in parallel

Pipeline parallelism

- Divide a process into stages
- Produce several items simultaneously



Very simple web server

1. Get URL
2. Get corresponding web page off of disk
3. Create actual page if dynamic content
4. Send web page to requester

Four things happen at once

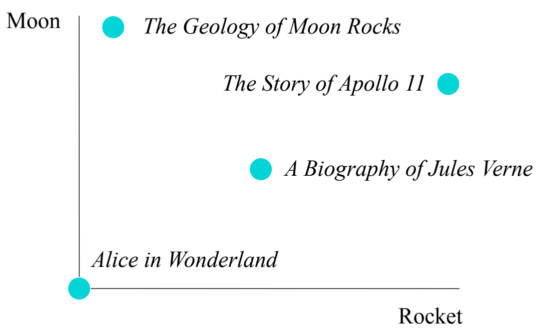
Four-way pipeline parallelism

1	url 1	url 2	url 3	url 4	url 5	url 6			
2		url 1	url 2	url 3	url 4	url 5	url 6		
3			url 1	url 2	url 3	url 4	url 5	url 6	
4				url 1	url 2	url 3	url 4	url 5	url 6

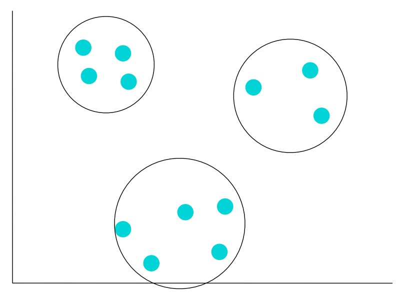
Data Clustering

- Data mining = looking for meaningful patterns in large data sets
- Data clustering = organizing a data set into clusters of "similar" items
- Data clustering can speed retrieval of related items

Document Vectors



Document Clustering



Clustering Algorithm

- Compute document vectors (i.e. a comparable representation of the document)
- Choose initial cluster centers
- Repeat
  - Compute performance function
  - Adjust centers
- Until function value converges or max iterations have elapsed
- Output cluster centers

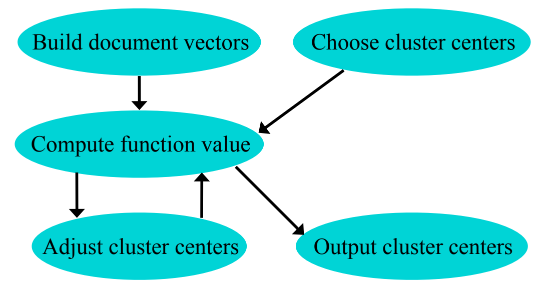
Data Parallelism Opportunities

- Operation being applied to a data set
- Examples
  - Generating document vectors
  - Finding closest center to each vector
  - Picking initial values of cluster centers

Functional Parallelism Opportunities

- Draw data dependence diagram
- Look for sets of nodes such that there are no paths from one node to another

Data Dependence Diagram



Programming Parallel Computers

- Extend compilers: translate sequential programs into parallel programs
- Extend languages: add parallel operations
- Add parallel language layer on top of sequential language
- Define totally new parallel language and compiler system
- Domain specific languages
- Use parallel libraries or a runtime

Strategy 1: Extend Compilers

- Parallelizing compiler
  - Detect parallelism in sequential program
  - Produce parallel executable program
- Focus on making Fortran/C/C++/Java programs parallel

Extend Compilers (cont.)

- Advantages
  - Can leverage millions of lines of existing serial programs
  - Saves time and labor
  - Requires no retraining of programmers
  - Sequential programming easier than parallel programming

Extend Compilers (cont.)

- Disadvantages
  - Parallelism may be irretrievably lost when programs written in sequential languages
  - Performance of parallelizing compilers on broad range of applications still up in air
    - More pessimistically, without *speculation*, automatic parallelization fails on irregular codes (e.g., sparse matrix codes)
    - *Scaling*, even with speculation, is unclear

Extend Language

- Add functions to a sequential language
  - Create and terminate processes
  - Synchronize processes
  - Allow processes to communicate

Extend Language (cont.)

- Advantages
  - Easiest, quickest, and least expensive
  - Allows existing compiler technology to be leveraged
  - New libraries can be ready soon after new parallel computers are available

Extend Language (cont.)

- Disadvantages
  - Lack of compiler support to catch errors
  - Easy to write programs that are difficult to debug

Add a Parallel Programming Layer

- Lower layer
  - Core of computation
  - Process manipulates its portion of data to produce its portion of result
- Upper layer
  - Creation and synchronization of processes
  - Partitioning of data among processes
- In the scientific programming arena, a few research prototypes have been built based on these principles
- Arguably MPI/OpenMP are a form of this

Create a Parallel Language

- Develop a parallel language "from scratch"
  - Occam, Stream is an example
  - Java is arguably such a language
- Add parallel constructs to an existing language
  - Co-Array Fortran, High Performance Fortran, Fortran 90
  - UPC
  - OpenMP

New Parallel Languages (cont.)

- Advantages
  - Allows programmer to communicate parallelism to compiler
  - Improves probability that executable will achieve high performance
- Disadvantages
  - Requires development of new compilers
  - New languages may not become standards
  - Programmer resistance

Current Status

- Low-level approach is most popular for scientific
  - Augment existing language with low-level parallel constructs
  - MPI and OpenMP are examples
- Advantages of low-level approach
  - Efficiency
  - Portability
- Disadvantage: More difficult to program and debug

Create a domain specific language that parallelism can be extracted from

- Create a language that targets a limited set of operators whose semantics easily enable parallelism and whose operands are easily checked to see if they are references to the same memory
- Databases are the most widespread example of this
- There are languages for chemistry, biology, etc. that make use of this.

Use parallel operators/library calls

- Have functions/operators whose implementation is parallel
- **ScaLAPACK**, e.g., is a parallel library for linear algebra
- Matlab has implementations of some of its array operators that use multicore parallelism

Summary (1/2)

- High performance computing
  - U.S. government
  - Capital-intensive industries
  - Many companies and research labs
  - Web services (Google, Amazon, Facebook, ebay, . . .
- Parallel computers
  - Commercial systems
  - Commodity-based systems

Summary (2/2)

- Power of CPUs keeps growing exponentially
- Parallel programming environments changing very slowly
- Two standards have emerged
  - MPI library, for processes that do not share memory
  - OpenMP directives, for processes that do share memory
  - We will be studying other models (Pthreads, UPC, Galois)