Parallel Programming
in C with MPI and OpenMP

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

Nature

↓

Observation

Physical Experimentation

←

Theory
Modern Scientific Method

Nature → Observation

Observation ← Physical Experimentation ← Theory

Numerical Simulation ← Physical Experimentation
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
- 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

ENIAC
350 flops

Today
> 1 trillion flops

One Billion Times Faster!
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
Microprocessor Revolution

- Speed (log scale)
- Micros
- Supercomputers
- Mainframes
- Minis

Moore's Law

Time
Modern Parallel Computers

- Caltech’s Cosmic Cube (Seitz and Fox)
- Commercial copy-cats
  - nCUBE Corporation
  - Intel’s Supercomputer Systems Division
  - Lots more
- Thinking Machines Corporation
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
Why Didn’t Everybody Buy One?

- Supercomputer $\neq \sum$ CPUs
  - Computation rate $\neq$ throughput
  - Inadequate I/O
- Software
  - Inadequate operating systems
  - Inadequate programming environments
After the “Shake Out”

- IBM
- Hewlett-Packard
- Silicon Graphics
- Sun Microsystems
Commercial Parallel Systems

- Relatively costly per processor
- Primitive programming environments
- Focus on commercial sales
- Scientists looked for alternative
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
ASCI White (10 teraops/sec)
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
Functional Parallelism

- Independent tasks apply different operations to different data elements

\[
\begin{align*}
a & \leftarrow 2 \\
b & \leftarrow 3 \\
m & \leftarrow (a + b) / 2 \\
s & \leftarrow (a^2 + b^2) / 2 \\
v & \leftarrow s - m^2
\end{align*}
\]

- First and second statements
- Third and fourth statements
Pipelining

- Divide a process into stages
- Produce several items simultaneously
Partial Sums Pipeline

\[
\]

\[
a[0] \quad a[1] \quad a[2] \quad a[3]
\]
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

Moon

- The Geology of Moon Rocks
- The Story of Apollo 11
- A Biography of Jules Verne

Alice in Wonderland

Rocket
Document Clustering
Clustering Algorithm

- Compute document vectors
- 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

1. Build document vectors
2. Compute function value
3. Adjust cluster centers
4. Choose cluster centers
5. Output cluster centers
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
Strategy 1: Extend Compilers

- Parallelizing compiler
  - Detect parallelism in sequential program
  - Produce parallel executable program
- Focus on making Fortran 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 irretrivably lost when programs written in sequential languages
  - Performance of parallelizing compilers on broad range of applications still up in air
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

- A few research prototypes have been built based on these principles
Create a Parallel Language

- Develop a parallel language “from scratch”
  - occam is an example
- Add parallel constructs to an existing language
  - Fortran 90
  - High Performance Fortran
  - C*
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
  - 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
Summary (1/2)

- High performance computing
  - U.S. government
  - Capital-intensive industries
  - Many companies and research labs

- 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