CSC510
Parallel Programming

Introduction
(Part 2)

Contents

• Part I
  – What is Parallel Computing?
  – Evolution of Supercomputing
• Part II
  – Applications of Parallel Computing
  – Principles of Parallel Computing
Applications of Parallel Computing

Dr. Joe Zhang

Computational Science

“An important development in sciences is occurring at the intersection of computer science and the sciences that has the potential to have a profound impact on science. It is a leap from the application of computing … to the integration of computer science concepts, tools, and theorems into the very fabric of science.” - Science 2020 Report, March 2006

Drivers for Change

- **Continued exponential increase in computational power**
  - Simulation is becoming third pillar of science, complementing theory and experiment
- **Continued exponential increase in experimental data**
  - Techniques and technology in data analysis, visualization, analytics, networking, and collaboration tools are becoming essential in all data rich scientific applications

Simulation: The Third Pillar of Science

- **Traditional scientific and engineering method:**
  1. Do **theory** or paper design
  2. Perform **experiments** or build system
- **Limitations:**
  - Too difficult—build large wind tunnels
  - Too expensive—build a throw-away passenger jet
  - Too slow—wait for climate or galactic evolution
  - Too dangerous—weapons, drug design, climate experimentation
- **Computational science and engineering paradigm:**
  3. Use computers to simulate and analyze the phenomenon
  - Based on known physical laws and efficient numerical methods
  - Analyze simulation results with computational tools and methods beyond what is possible manually
Data Driven Science

- Scientific data sets are growing exponentially
  - Ability to generate data is exceeding our ability to store and analyze
  - Simulation systems and some observational devices grow in capability with Moore’s Law
- Petabyte (PB) data sets will soon be common:
  - Climate modeling: estimates of the next IPCC data is in 10s of petabytes
  - Genome: JGI alone will have .5 petabyte of data this year and double each year
  - Particle physics: LHC is projected to produce 16 petabytes of data per year
  - Astrophysics: LSST and others will produce 5 petabytes/year
- Big data research

Example: Universe Simulation

Run on NASA AMES' constellation of supercomputer processors
Hurricane Simulation

Source: Data from M. Wehner, visualization by Prabhat, LBNL

Dr. Joe Zhang

Simulations in Life Sciences

Dr. Joe Zhang

PP-2: Introduction
Major Applications of Next Generation Supercomputer

Some Challenging Computations

- **Science**
  - Global climate modeling
  - Biology: genomics; protein folding; drug design
  - Astrophysical modeling
  - Computational Chemistry
  - Computational Material Sciences and Nanosciences

- **Engineering**
  - Semiconductor design
  - Earthquake and structural modeling
  - Computation fluid dynamics (airplane design)
  - Combustion (engine design)
  - Crash simulation

- **Business**
  - Financial and economic modeling
  - Transaction processing, web services and search engines

- **Defense**
  - Nuclear weapons -- test by simulations
  - Cryptography

Dr. Joe Zhang

PP-2: Introduction
Mississippi NSF EPSCoR Grant

- National Science Foundation EPSCoR Program
- Mississippi NSF EPSCoR Grant
  - Simulation and Modeling of Complex Systems
  - 2009-2016
  - $20M

Need for Parallel Computing

- Need for Parallelism
  - Many computational problems in computer science
    - Parallel (distributed) information retrieval
      - e.g. Google
    - Distributed database
    - Parallel image processing
    - Big data - large scale data analysis
    - E-business
Principles of Parallel Computing

- Finding enough parallelism (Amdahl’s Law)
- Granularity
- Locality
- Load balance
- Coordination and synchronization
- Performance modeling

All of these things makes parallel programming even harder than sequential programming.
“Automatic” Parallelism in Modern Machines

- Bit level parallelism
  - within floating point operations, etc.
- Instruction level parallelism (ILP)
  - multiple instructions execute per clock cycle
- Memory system parallelism
  - overlap of memory operations with computation
- OS parallelism
  - multiple jobs run in parallel on commodity SMPs

For very high performance, need users to identify, schedule and coordinate parallel tasks

Finding Enough Parallelism

- Suppose only part of an application seems parallel
- Amdahl’s law
  - let s be the fraction of work done sequentially, so (1-s) is fraction parallelizable
  - P = number of processors
    
    \[
    \text{Speedup}(P) = \frac{\text{Time}(1)}{\text{Time}(P)}
    \]
    
    \[
    \leq \frac{1}{s + (1-s)/P}
    \]
    
    \[
    \leq \frac{1}{s}
    \]

- Even if the parallel part speeds up perfectly, performance is limited by the sequential part
Overhead of Parallelism

- Given enough parallel work, overhead is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work

Locality and Parallelism

- Large memories are slow, fast memories are small
- Parallel processors, collectively, have large, fast cache
- Algorithm should do most work on local data – reduce communication cost

Dr. Joe Zhang
**Processor-DRAM Gap (latency)**

Goal: find algorithms that minimize communication

![Graph showing the Processor-DRAM Gap (latency) with Moore's Law](image)

**Load Imbalance**

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Algorithm needs to balance load
  - Analyze work load, divide up evenly, before starting
    - "Static Load Balancing"
  - Sometimes work load changes dynamically, need to rebalance dynamically
    - "Dynamic Load Balancing"
Parallel Software Eventually

- 2 types of programmers ➔ 2 layers
- **Efficiency Layer** (10% of today’s programmers)
  - Expert programmers build Libraries implementing motifs, “Frameworks”, OS, …
  - Highest fraction of peak performance possible
- **Productivity Layer** (90% of today’s programmers)
  - Domain experts / Naïve programmers productively build parallel applications by composing frameworks & libraries
  - Hide as many details of machine, parallelism as possible
  - Willing to sacrifice some performance for productive programming
- Students may want to work at either level
  - In the meantime, we all need to understand enough of the efficiency layer to use parallelism effectively

Dr. Joe Zhang PP-2: Introduction

---

Improving Real Performance

**Peak Performance grows exponentially, a la Moore’s Law**
- In 1990’s, peak performance increased 100x; in 2000’s, it will increase 1000x

**But efficiency (the performance relative to the hardware peak) has declined**
- was 40-50% on the vector supercomputers of 1990s
- now as little as 5-10% on parallel supercomputers of today

**Close the gap through ...**
- Mathematical methods and algorithms
- More efficient programming models and tools

Dr. Joe Zhang PP-2: Introduction
Performance Levels

- Peak performance
  - Sum of all speeds of all floating point units in the system
  - You can’t possibly compute faster than this speed
- LINPACK
  - The “hello world” program for parallel performance
  - Solve Ax=b using Gaussian Elimination, highly tuned
- Gordon Bell Prize winning applications performance
  - The right application/algorithm/platform combination plus years of work
- Average sustained applications performance
  - What one reasonable can expect for standard applications

Performance Levels (for example on NERSC-5)

- Peak advertised performance (PAP): 100 Tflop/s
- LINPACK (TPP): 84 Tflop/s
- Best climate application: 14 Tflop/s
  - WRF code benchmarked in December 2007
- Average sustained applications performance: ? Tflop/s
  - Probably less than 10% peak!
- We will study performance
  - How to measure it
  - Identifying bottlenecks
  - Practical performance tuning
Why Parallelism

- All major processor vendors are producing multicore chips
  - Every machine will soon be a parallel machine
- New software model
  - Want a new feature? Hide the “cost” by speeding up the code
- Some may eventually be hidden in libraries, compilers, and high level languages
  - But a lot of work is needed to get there
- Big open questions:
  - What will be the killer apps for multicore machines
  - How should the chips be designed, and how will they be programmed?

What you should get out of the course

In depth understanding of:
- When is parallel computing useful?
- Understanding of parallel computing hardware options.
- Overview of programming models (software) and tools.
- Some important parallel applications and the algorithms
- Performance analysis and tuning
- Exposure to various open research questions (only for graduate level CSC510)
CSC510
Parallel Programming

Questions?

Dr. Joe Zhang
PP_2: Introduction