Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

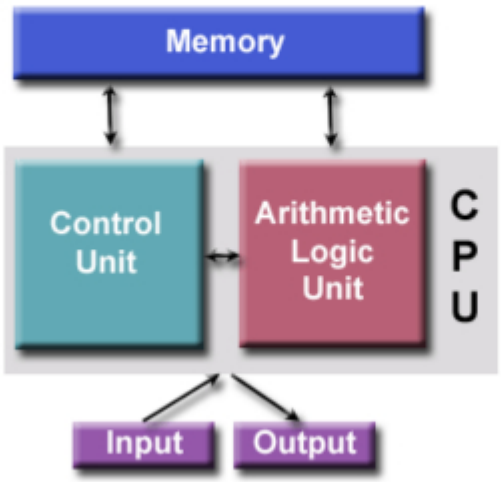
Content

- High Performance Parallel Computing
  - Overview
  - Concepts and Terminology
  - Parallel Memory Architectures
  - Parallel Programming Model
  - Parallel Examples and Exercises

Concepts and Terminology

von Neumann Architecture

Known as "stored-program computer" - both program instructions and data are kept in electronic memory. Differs from earlier computers which were programmed through "hard wiring".



https://en.wikipedia.org/wiki/John_von_Neumann

Flynn’s Classification

- Flynn's taxonomy is a classification of computer architectures. The classification system has been used as a tool in design of modern processors and their functionalities.

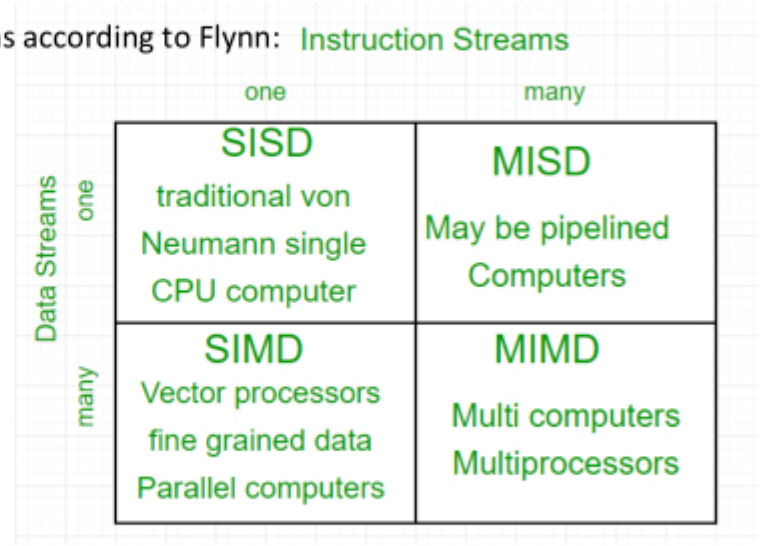
- Flynn's Classical Taxonomy – Based on # of instruction and data streams
  - SISD: Single Instruction, Single Data streams
  - SIMD: Single Instruction, Multiple Data streams
  - MISD: Multiple Instruction, Single Data streams
  - MIMD: Multiple Instruction, Multiple Data streams

Remember

https://en.wikipedia.org/wiki/Flynn%27s_taxonomy

The matrix below defines the 4 possible classifications according to Flynn:

SISD Single Instruction stream Single Data stream	**SIMD** Single Instruction stream Multiple Data stream
MISD Multiple Instruction stream Single Data stream	**MIMD** Multiple Instruction stream Multiple Data stream



Note: **MIMD architectures can include SIMD execution sub-components**

General Parallel Terminology

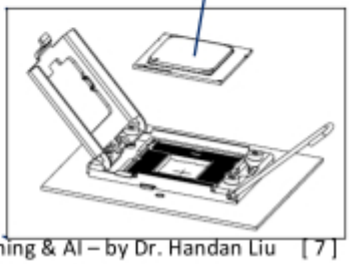
HPC Cluster
Supercomputer

Rack/Cabinet

CPU/Core

Node (blade) – standalone von Neumann computer

Socket



General Parallel Terminology

- Shared Memory
  - All processors have direct (usually bus based) access to common physical memory.
- Symmetric Multi-Processor (SMP)
  - Multiple processors share a single address space and have equal access to all resources
- Distributed Memory
  - Network-based memory access for physical memory that is not common
- Communication
  - Data exchange between parallel tasks
- Networking:
  - A connection among all nodes in a HPC system, including TCP/IP backplane (1GByte or 10GByte) or Infiniband backplane (10GByte or 100GByte) etc.
- Synchronization:
  - Usually involves waiting by at least one task and can therefore cause a parallel application's wall clock execution time to increase.

General Parallel Terminology

- Parallel Overhead
  - The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
    - ✔ Task start-up time; Synchronizations; Data communications
    - ✔ Software overhead imposed by parallel languages, libraries, operating system, etc.,
    - ✔ Task termination time
- Embarrassingly Parallel
  - Solving many similar, but independent tasks simultaneously; little to no need for coordination between the tasks.
- Massively Parallel
  - processing elements numbering in the hundreds of thousands to millions.
- 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 resources.

Limits of Parallel Programming

Amdahl's Law

- Amdahl's law is a formula which is often used in *parallel computing* to predict the theoretical speedup when using multiple processors.
- Amdahl's Law states that potential program speedup is defined by the fraction of code (P) that can be parallelized:

$$\text{speedup} = \frac{1}{1 - P}$$

https://en.wikipedia.org/wiki/Amdahl%27s_law

Limits of Parallel Programming

Amdahl's Law

- Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

where P = parallel fraction, N = number of processors and S = serial fraction.

	speedup			
N	P = .50	P = .90	P = .95	P = .99
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1,000	1.99	9.91	19.62	90.99
10,000	1.99	9.91	19.96	99.02
100,000	1.99	9.99	19.99	99.90

"Famous" quote: *You can spend a lifetime getting 95% of your code to be parallel, and never achieve better than 20x speedup no matter how many processors you throw at it!*

2D Grid Calculations				
Parallel fraction	85 seconds	85%	?	6.67x Speedup
Serial fraction	15 seconds	15%		

Increase the problem size by doubling the grid dimensions and halving the time step.

????

This way is not easily to calculate the parallel speedup

2D Grid Calculations				
Parallel fraction	680 seconds	97.84%	?	46.3x Speedup
Serial fraction	15 seconds	2.16%		

One conclusion:
Problems that increase the percentage of parallel time with their size are more **scalable** than problems with a fixed percentage of parallel time.

How to evaluate your parallel performance practically?

- In practice, I usually use the following formula to calculate the acceleration of parallel performance:

$$\text{Speedup} = \frac{\text{Elapsed time of serial execution}}{\text{Elapsed time of parallel execution}}$$

https://en.wikipedia.org/wiki/Elapsed_real_time

PowerPoint Slide Show - CSYE7105-HandanLiu-w01-2 - PowerPoint

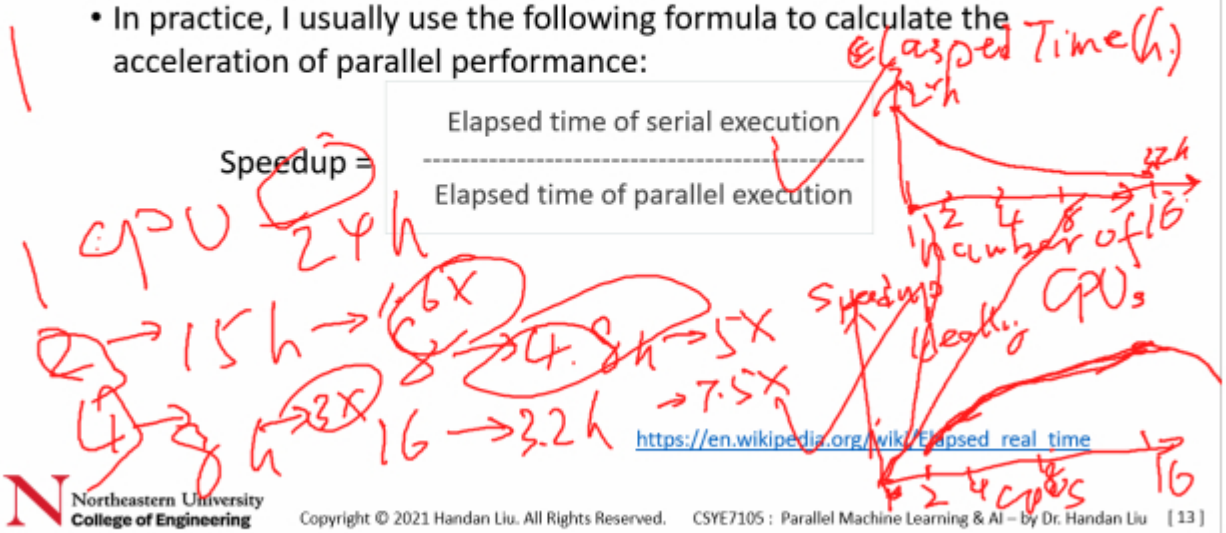
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save

Talking: Handan Liu

How to evaluate your parallel performance practically?

- In practice, I usually use the following formula to calculate the acceleration of parallel performance:

$$\text{Speedup} = \frac{\text{Elapsed time of serial execution}}{\text{Elapsed time of parallel execution}}$$



Northeastern University
College of Engineering

Slide 13 of 16

Costs of Parallel Programming

Complexity

- In general, parallel applications are much more complex than corresponding serial applications, perhaps an order of magnitude.
- The costs of complexity are measured in programmer time in virtually every aspect of the software development cycle:
  - Design
  - Coding
  - Debugging
  - Tuning
  - Maintenance
- Adhering to "good" software development practices is essential when working with parallel applications - especially if somebody besides you will have to work with the software.

Limits and Costs of Parallel Programming

Resource Requirements

- The primary intent of parallel programming is to decrease execution elapsed time, however in order to accomplish this, more CPU time is required.
- The amount of memory required can be greater for parallel codes than serial codes.
- For short running parallel programs, there can actually be a decrease in performance compared to a similar serial implementation.

- Stay safe!
- See you next class!

<u>Next Lecture will Continue:</u>

High Performance Parallel Computing

- Overview
- Concepts and Terminology
- Parallel Memory Architectures
- Parallel Programming Model
- Parallel Examples and Exercises