

CSL 860: Modern Parallel Computation

Course Information

- www.cse.iitd.ac.in/~subodh/courses/CSL860
- Grading:
 - Quizzes 25
 - [Lab Exercise 1](#) 7 + 8
 - [Project](#) 35 (25% design, 25% presentations, 50% Demo)
 - Final Exam 25

Verbal discussion of assignments is fine but looking at someone else's work and then doing your own is not. **Letting your work become available or visible to others is also cheating.** For your project, you may borrow code available online but must clearly identify such code with due reference to the source. **First instance of cheating will invite a zero in the assignment and a letter grade penalty. Repeat offender will fail the course.**

Course Material

- Documents posted on the course website
- Reference books:
 - Introduction to Parallel Computing
by Grama, Gupta, Karypis & Kumar
 - An introduction to Parallel Algorithms
by Jaja
 - Parallel Programming in C with MPI and OpenMP
by Quinn

What this course is about?

- Learn to solve problems in parallel
 - Concurrency issues
 - Performance/Load balance issues
 - Scalability issues
- Technical knowledge
 - Theoretical models of computation
 - processor architecture features and constraints
 - programming API, tools and techniques
 - Standard algorithms and data structures
- Different system architecture
 - Shared memory, Communication network
- Hand-on
 - Lots of programming
 - Multi-core, massively parallel
 - OpenMP, MPI, Cuda

Programming in the 'Parallel'

- Understand target model (Semantics)
 - Implications/Restrictions of constructs/features
- Design for the target model
 - Choice of granularity, synchronization primitive
 - Usually more of a performance issue
- Think concurrent
 - For each thread, other threads are 'adversaries'
 - At least with regard to timing
 - Process launch, Communication, Synchronization
 - Clearly define pre and post conditions
- Employ high-level constructs when possible
 - Debugging is extra-hard

Serial vs parallel

- ATM Withdrawal

```
Withdraw(int accountnum, int amount) {  
    cur balance = balance(accountnum) ;  
    if(curbalance > amount) {  
        setbalance(accountnum, curbalance-amount);  
        eject(amount)  
    } else ...  
}
```

Some Complex Problems

- *N*-body simulation
 - 1 million bodies \Rightarrow days/iteration
- Atmospheric simulation
 - 1km 3D-grid, each point interacts with neighbors
 - Days of simulation time
- Movie making
 - A few minutes = 30 days of rendering time
- Oil exploration
 - months of sequential processing of seismic data
- Financial processing
 - market prediction, investing
- Computational biology
 - drug design
 - gene sequencing (Celera)

Why Parallel

- Can't clock faster
- Do more per clock
 - Execute complex “special-purpose” instruction
 - Execute more simple instructions

Measuring Performance

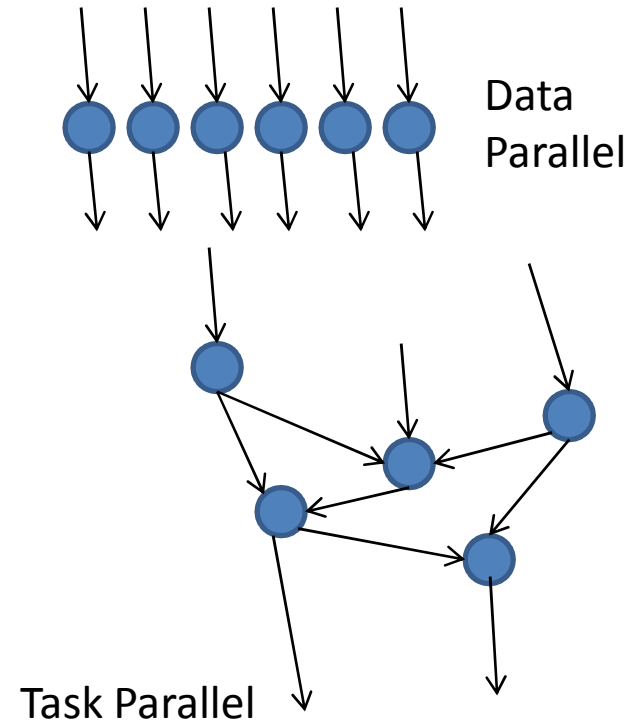
- How fast does a job complete
 - Elapsed time (Latency)
 - compute + communicate + synchronize
- How many jobs complete in a given time
 - Throughput
 - Are they independent jobs?

Learning Parallel Programming

- Let compiler extract parallelism?
 - Some predictive-issue has succeeded
 - In general, not successful so far
 - Too context sensitive
 - Many efficient serial data structures and algorithms are parallel-inefficient
 - Even if compiler extracted parallelism from serial code, it would not be what you want
- Programmer must conceptualize and code parallelism
- Understand parallel algorithms and data structures

Parallel Task Decomposition

- Data Parallel
 - Perform $f(x)$ for **many x**
- Task Parallel
 - Perform many functions f_i

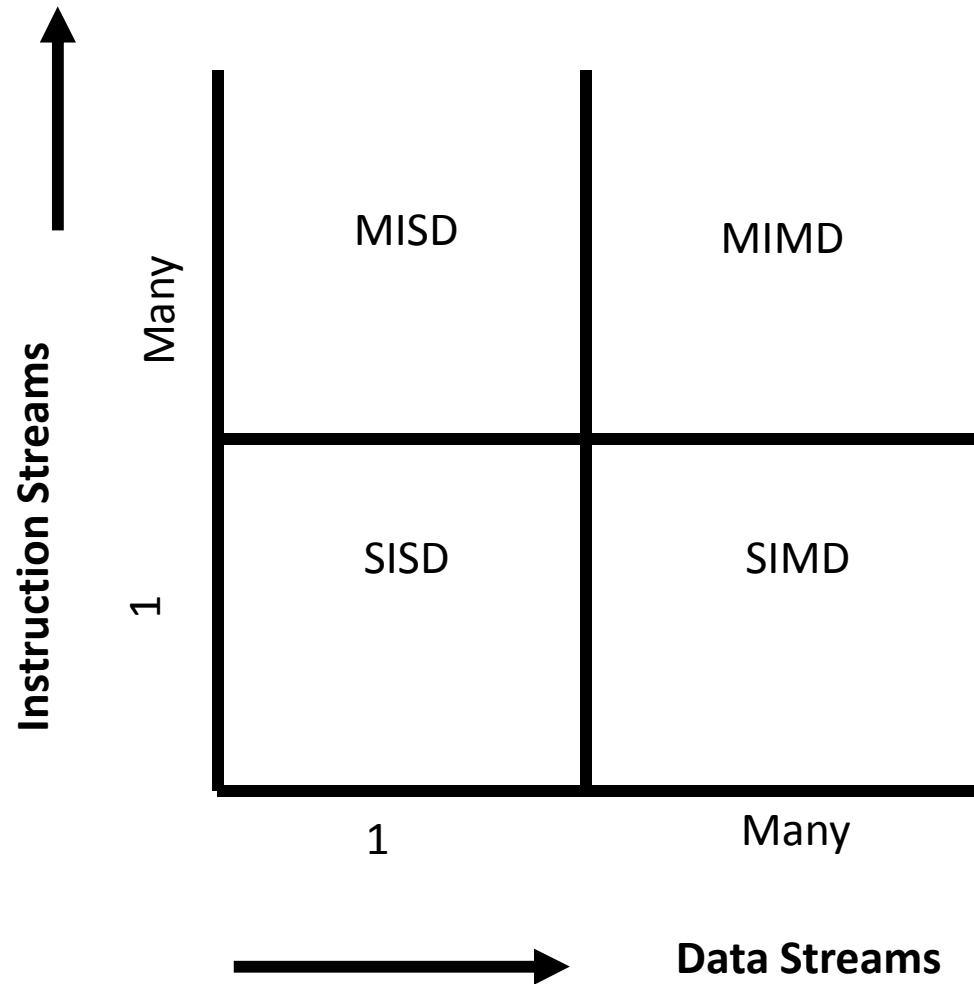


Pipeline

Fundamental Issues

- Is the problem amenable to parallelization?
 - Are there (serial) dependencies
- What machine architectures are available?
 - Can they be re-configured?
 - Communication network
- Algorithm
 - How to decompose the problem into *tasks*
 - How to map tasks to processors

Parallel Architectures: Flynn's Taxonomy



Parallel Architectures: Components

- Processors
- Memory
 - shared
 - distributed
- Communication
 - Heirarchical, Crossbar, Bus, Memory
 - Synchronization
- Control
 - centralized
 - distributed

Formal Performance Metrics

$$\text{Speedup, } S(p) = \frac{\text{Exec time using 1 processor system } (T_1)}{\text{Exec time using } p \text{ processors } (T_p)}$$

$$\text{Efficiency} = \frac{S_p}{p}$$

$$\text{Cost, } C_p = p \times T_p$$

Optimal if $C_p = T_1$

Look out for **slowdown**:

$$T_1 = n^3$$

$$T_p = n^{2.5}, \text{ for } p = n^2$$

$$C_p = n^{4.5}$$

Amdahl's Law

- f = fraction of the problem that is sequential
 - $(1 - f)$ = fraction that is parallel

- Parallel time $T_p = f + (1 - f)/p$

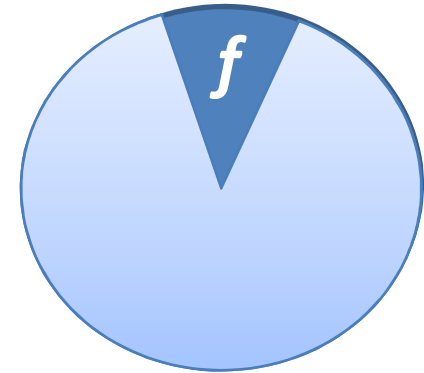
- Speedup with p processors:

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

Amdahl's Law

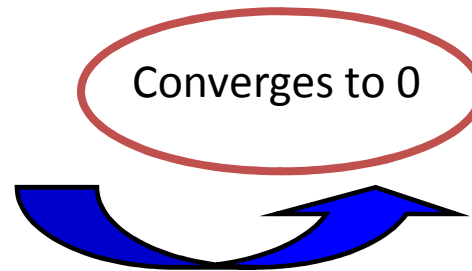
- Only fraction $(1-f)$ shared by p processors

Increasing p cannot speed-up fraction f



- Upper bound on speedup at $p = \infty$

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$



$$S_\infty = \frac{1}{f}$$

Example:

$$f = 2\%, S_\infty = 1 / 0.02 = 50$$