# Lecture 26 (Multi-Core Systems)

*apparently working memory is important*

## 1 Shared Memory vs Message Passing

### 1.1 Shared Memory

1. Easy to program
2. Issues with scalability
3. Code is portable across machines

### 1.2 Message Passing

1. Hard to program
2. Scalable
3. Code may not be portable

## 2 Amdahl's Law

$$T_{par} = T_{seq} \times \left( f_{seq} + \frac{1 - f_{par}}{P} \right)$$

$$S = \frac{1}{f_{seq} + \frac{1-f_{par}}{P}}$$

## 3 Gustafson-Barsis's Law

New workload: $W_{new} = f_{seq}W + (1 - f_{par})PW$

Sequential time: $T_{seq} = \alpha W_{new}$

Parallel time: $T_{par} = \alpha(f_{seq}W + (1 - f_{par})PW/P)$

Speed-up: $S = f_{seq} + (1 - f_{seq})P$

# 4 Design Space of Multiprocessors

1. SISD - Single Instruction Single Data
2. SIMD - Single Instruction Multiple Data
3. MISD - Multiple Instruction Single Data
4. MIMD - Multiple Instruction Multiple Data
    i. SPMD - Single Processor Multiple Data
    ii. MPMD - Multiple Processor Multiple Data

# 5 Hardware Threads

1. Cores with large issue widths have wasted issue slots
2. Run multiple processes simultaneously on the same core
3. Maintain a hardware thread ID
4. Need separate ROBs (and some other hardware implementation details)

# 6 Coarse-Grained Multithreading

1. Run instruction for each thread for $k$ cycles
2. Separate program counters, ROBs and retirement register files per thread
3. Thread ID is tagged with each entry
4. Switch in case of high-latency event

# 7 Fine-Grained Multithreading

1. Small $k$
2. Switch happens in case of low-latency event as well

# 8 Simultaneous Multithreading

1. Dynamically split the issue slots between threads
2. Heuristics
    - fairness
    - instruction criticality
    - thread criticality
    - instruction type
3. Hyperthreading is SMT but static partitioning

# 9 Issues with Large Caches

1. Access times can be very slow

2. Parallel accesses will make it slower
3. Have distributed caches - “shared private cache”