

# THREAD LEVEL PARALLELISM

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

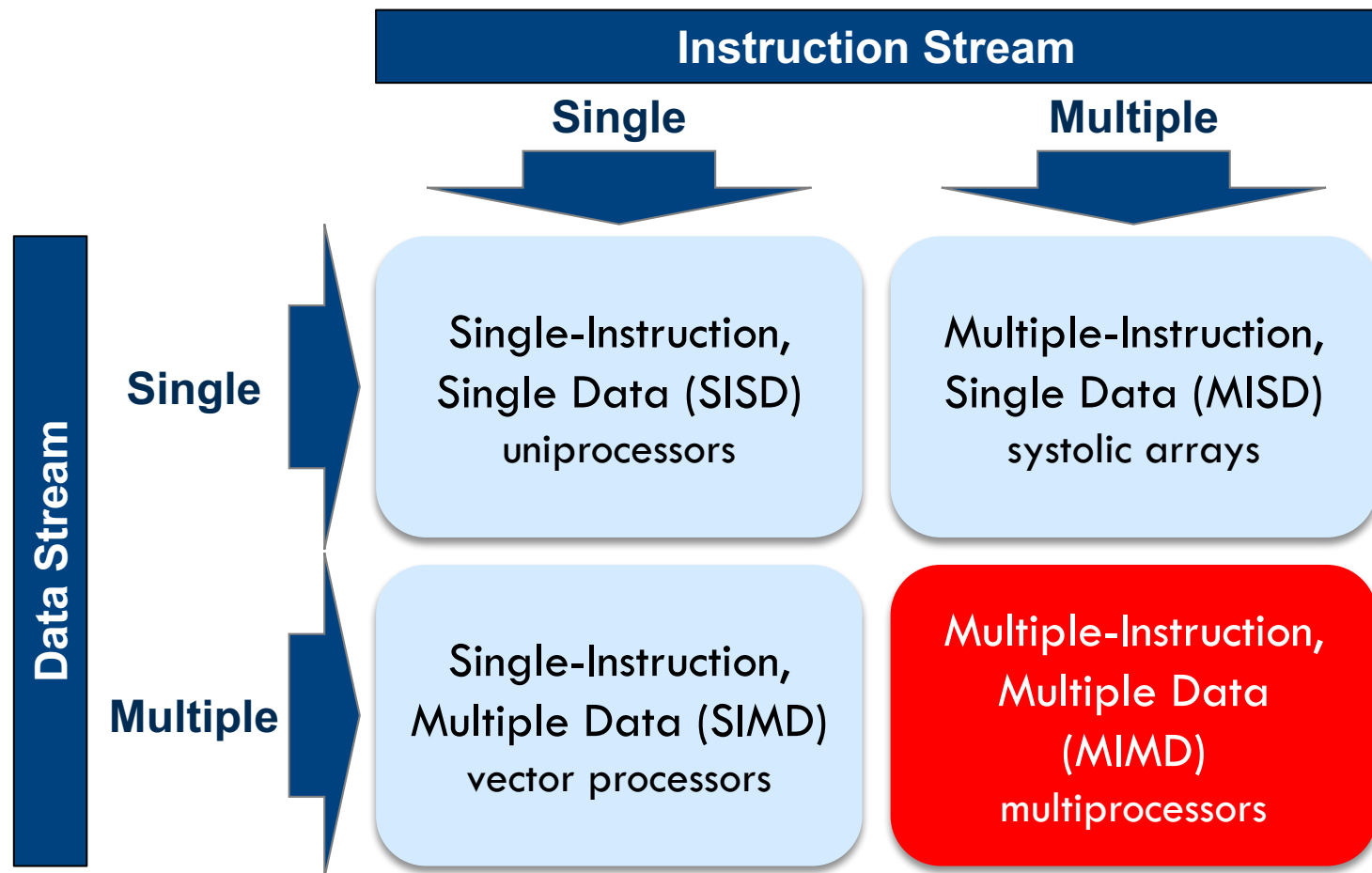
University of Utah

# Overview

- Announcement
  - ▣ Homework 6 is due on Apr. 18<sup>th</sup>
  
- This lecture
  - ▣ Thread level parallelism (TLP)
  - ▣ Parallel architectures for exploiting TLP
    - Hardware multithreading
    - Symmetric multiprocessors
    - Chip multiprocessing

# Recall: Flynn's Taxonomy

- Forms of computer architectures



# Basics of Threads

- **Thread** is a single sequential flow of control within a program including instructions and state
  - ▣ Register state is called **thread context**
- A program may be single- or multi-threaded
  - ▣ Single-threaded program can handle one task at any time
- **Multitasking** is performed by modern operating systems to load the context of a new thread while the old thread's context is written back to memory

# Thread Level Parallelism (TLP)

- Users prefer to execute multiple applications
  - ▣ Piping applications in Linux
    - `gunzip -c foo.gz | grep bar | perl some-script.pl`
  - ▣ Your favorite applications while working in office
    - Music player, web browser, terminal, etc.
- Many applications are amenable to parallelism
  - ▣ Explicitly multi-threaded programs
    - Pthreaded applications
  - ▣ Parallel languages and libraries
    - Java, C#, OpenMP

# Thread Level Parallel Architectures

- Architectures for exploiting thread-level parallelism

## Hardware Multithreading

- Multiple threads run on the same processor pipeline
- Multithreading levels
  - Coarse grained multithreading (CGMT)
  - Fine grained multithreading (FGMT)
  - Simultaneous multithreading (SMT)

## Multiprocessing

- Different threads run on different processors
- Two general types
  - Symmetric multiprocessors (SMP)
    - Single CPU per chip
  - Chip Multiprocessors (CMP)
    - Multiple CPUs per chip

# Hardware Multithreading

- **Observation:** CPU become idle due to latency of memory operations, dependent instructions, and branch resolution
- **Key idea:** utilize idle resources to improve performance
  - ▣ Support multiple thread contexts in a single processor
  - ▣ Exploit thread level parallelism
- **Challenge:** the energy and performance costs of context switching

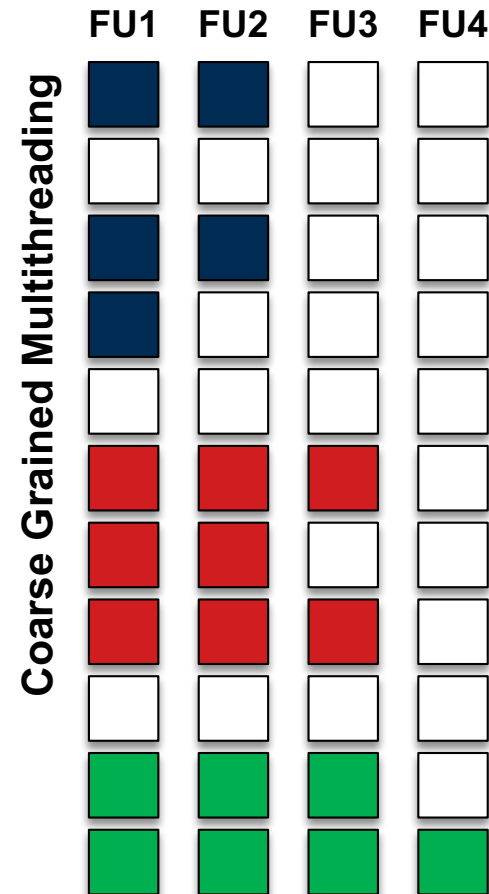
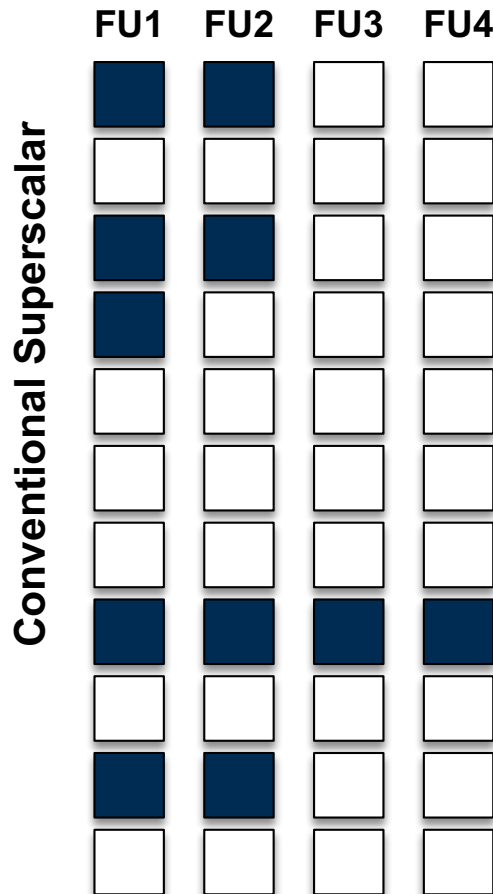
# Coarse Grained Multithreading

- ❑ Single thread runs until a costly stall—e.g. last level cache miss
- ❑ Another thread starts during stall for first
  - ▣ Pipeline fill time requires several cycles!
- ❑ At any time, only one thread is in the pipeline
- ❑ Does not cover short stalls
- ❑ Needs hardware support
  - ▣ PC and register file for each thread



# Coarse Grained Multithreading

## □ Superscalar vs. CGMT

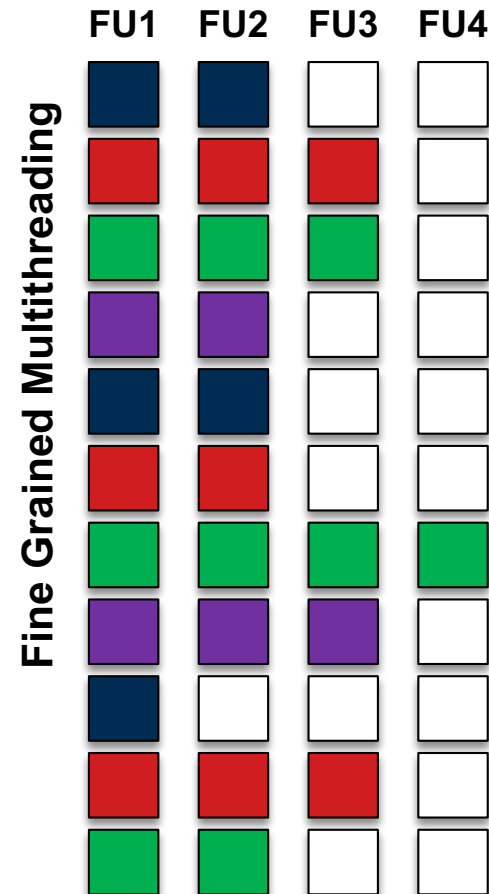
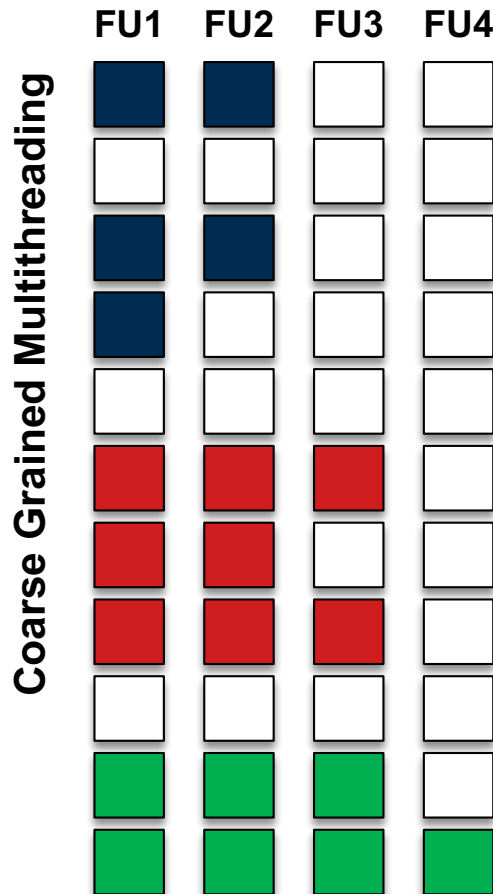


# Fine Grain Multithreading

- Two or more threads interleave instructions
  - ▣ Round-robin fashion
  - ▣ Skip stalled threads
- Needs hardware support
  - ▣ Separate PC and register file for each thread
  - ▣ Hardware to control alternating pattern
- Naturally hides delays
  - ▣ Data hazards, Cache misses
  - ▣ Pipeline runs with rare stalls
- Does not make full use of multi-issue architecture

# Fine Grained Multithreading

## □ CGMT vs. FGMT

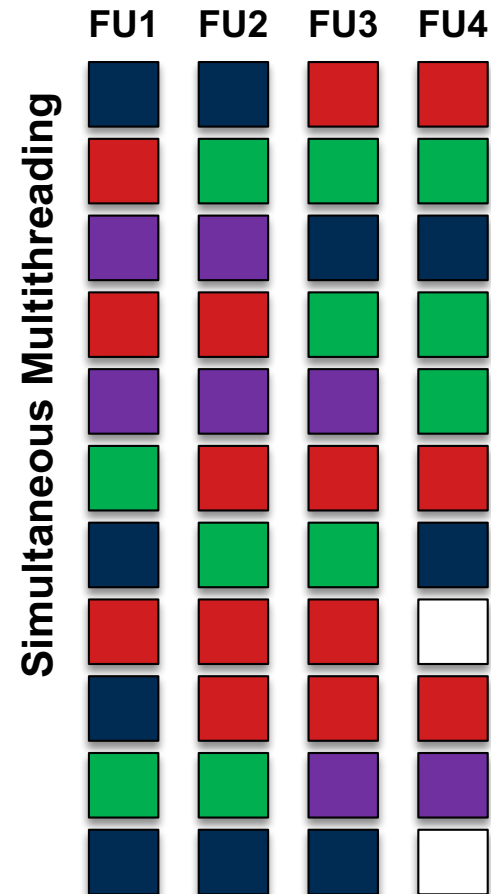
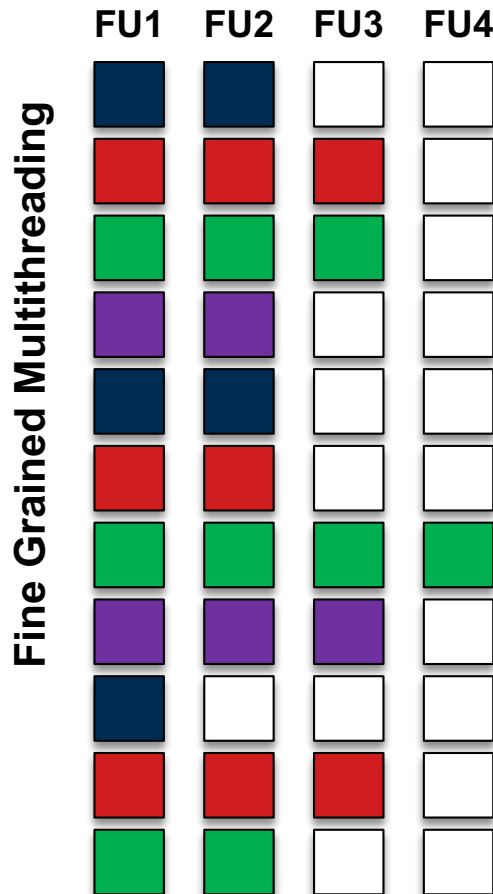


# Simultaneous Multithreading

- Instructions from multiple threads issued on same cycle
  - ▣ Uses register renaming and dynamic scheduling facility of multi-issue architecture
- Needs more hardware support
  - ▣ Register files, PC's for each thread
  - ▣ Temporary result registers before commit
  - ▣ Support to sort out which threads get results from which instructions
- Maximizes utilization of execution units

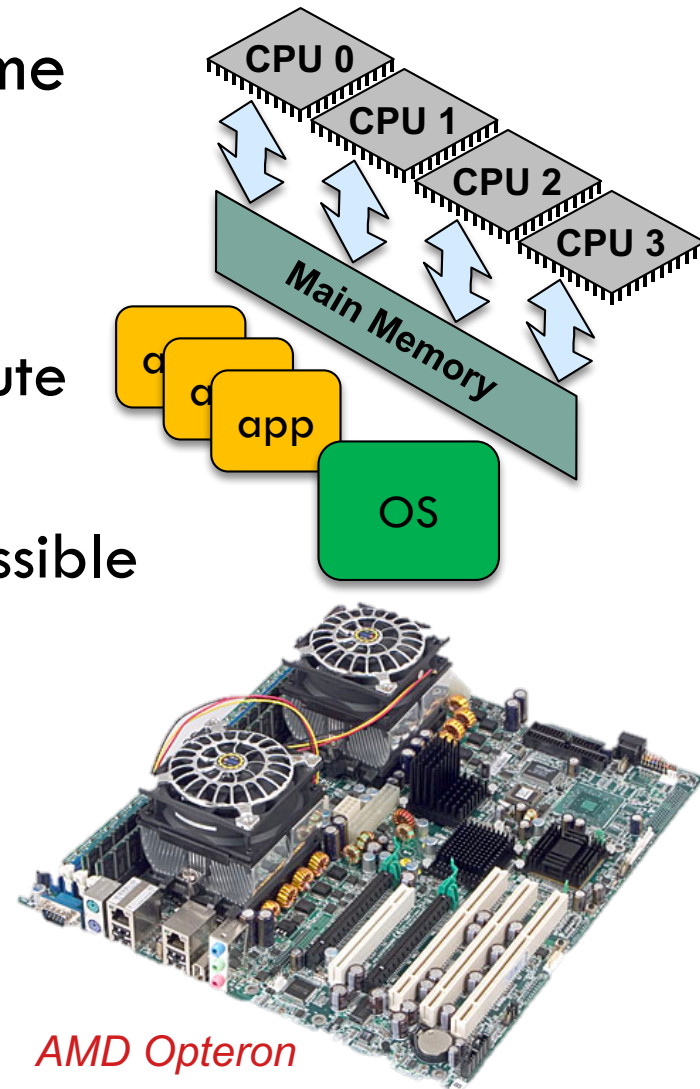
# Simultaneous Multithreading

## □ FGMT vs. SMT



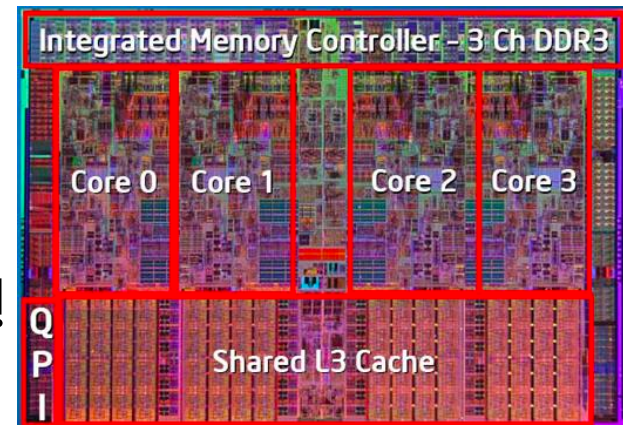
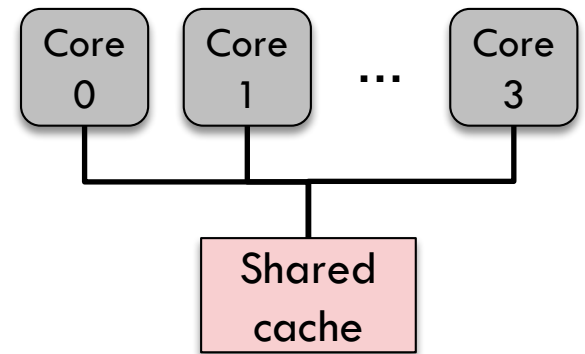
# Symmetric Multiprocessors

- ❑ Multiple CPU chips share the same memory
- ❑ From the OS's point of view
  - ▣ All of the CPUs have equal compute capabilities
  - ▣ The main memory is equally accessible by the CPU chips
- ❑ OS runs every thread on a CPU
- ❑ Every CPU has its own power distribution and cooling system



# Chip Multiprocessors

- Can be viewed as a simple SMP on single chip
- CPUs are now called cores
  - ▣ One thread per core
- Shared higher level caches
  - ▣ Typically the last level
  - ▣ Lower latency
  - ▣ Improved bandwidth
- Not necessarily homogenous cores!



*Intel Nehalem (Core i7)*

# Why Chip Multiprocessing?

- CMP exploits parallelism at lower costs than SMP
  - ▣ A single interface to the main memory
  - ▣ Only one CPU socket is required on the motherboard
- CMP requires less off-chip communication
  - ▣ Lower power and energy consumption
  - ▣ Better performance due to improved AMAT
- CMP better employs the additional transistors that are made available based on the Moore's law
  - ▣ More cores rather than more complicated pipelines



# Efficiency of Chip Multiprocessing

- **Ideally**,  $n$  cores provide  $n\times$  performance
- Example: design an ideal dual-processor
  - ▣ **Goal**: provide the same performance as uniprocessor

	Uniprocessor	Dual-processor
Frequency	1	?
Execution Time	1	1
Dynamic Power	1	?
Dynamic Energy	1	?
Energy Efficiency	1	?

# Efficiency of Chip Multiprocessing

- **Ideally**,  $n$  cores provide  $n\times$  performance
- Example: design an ideal dual-processor
  - ▣ **Goal**: provide the same performance as uniprocessor

	Uniprocessor	Dual-processor
Frequency	1	0.5
Execution Time	1	1
Dynamic Power	1	$2 \times 0.125$
Dynamic Energy	1	$2 \times 0.125$
Energy Efficiency	1	4

$$f \propto V \text{ \& } P \propto V^3 \rightarrow V_{dual} = 0.5V_{uni} \rightarrow P_{dual} = 2 \times 0.125P_{uni}$$