# Multi Processing

* Single Instruction Single Data (SISD)

(Uniprocessor) Execute one instruction stream, each instruction stream operates on single data

* Single Instruction Multiple Data (SIMD)

(Vector) Multiple data

* Multiple Instruction Single Data (MISD)

Not used much

* Multiple Instruction Multiple Data (MIMD)

Why not uniprocessor?

1. Uniprocessor already be 4-wide -> diminishing return from getting uniprocessor wider (issue more instruction at same time)
2. Uniprocessor need more voltage to increase frequency, thus needing more power

Typically, we assume frequency is proportional to voltage. The dynamic power of CPU: voltage^2 \* frequency -> power related with voltage^3

Issue with multiple processor: sequential thread is easy, debugging parallel code is much more difficult, performance scaling very hard to get (no longer proportional to core number)

图片包含 人员

描述已自动生成

## Centralized Shared Memory (today)

* Use same bus to access same memory
* Uniform Memory Access (Time): since each core is at same distance to main memory
* SMP (symmetric multiprocessor): each core looks like each other

With more cores:

1. Need more memory (slow)
2. Memory get too much access (multiple core generate cache miss) -> issue related with memory bandwidth

Thus, centralized shared memory might only work for small machine. With more cores, the memory bandwidth is saturated, and each core need to queue to access memory (slower, no advantage).

## Distributed Memory (no sharing)

Only one core can access a memory slice. Each core with a network interface card to connect to a network. When there is cache miss, each core can assess its memory. If a core wants to access memory in other cores, that core need to generate a request and other cores need to respond the request. Use message passing for data communication. Each core more like a separated computer, with network interface much quicker than internet. (scale to large number of cores and force programming to explicitly perform communication)

图片包含 人员, 掌握

描述已自动生成

Put accessed page in local memory to increase spread

Message Passing v.s. Shared Memory

图片包含 文字

描述已自动生成

图片包含 人员, 枪, 武器

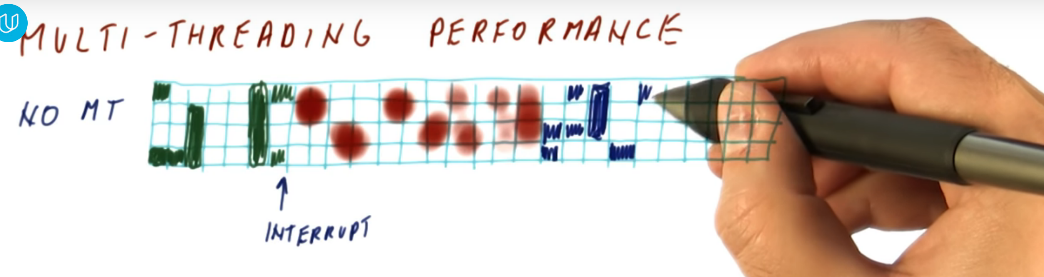
描述已自动生成

Since message passing need to receive and send data, lots of code need to write for data distribution part; for message passing already guarantee the initialization of array to be completed for other cores to read, 0 line needed for that part; for shared memory, need synchronization to first initialize the array.

Shared Memory Hardware

* Multiple cores share physical address space (UMA, NUMA)
* Multiple-threading by time-sharing a core (can get shared memory behavior)
* Hardware multithreading in a core (switch between threads: coarse-grain (change thread every few seconds), fine-grain (change thread every cycle), simultaneous multi-threading (SMT, hyper threading, in any cycles, can do instruction belonging to different threads)

1. No multi-thread supports (just time sharing) – we want multiple program to make progress but no for performance (single is best, multiple will introduce overhead)

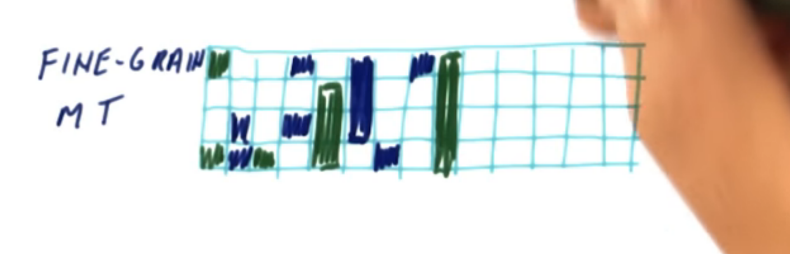


1. Chip Multi-processor (need more cores)

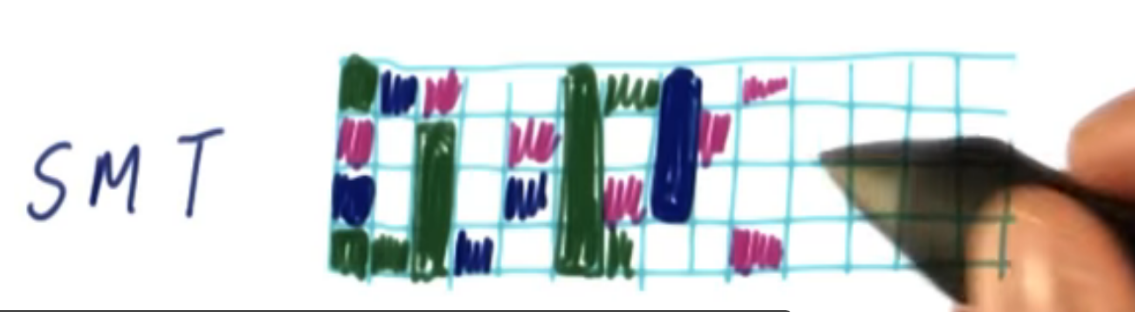
图片包含 室内

描述已自动生成

1. Find-grain Multi-threading (change thread every cycle), benefiting from long idle in one thread (cache miss), can run other thread

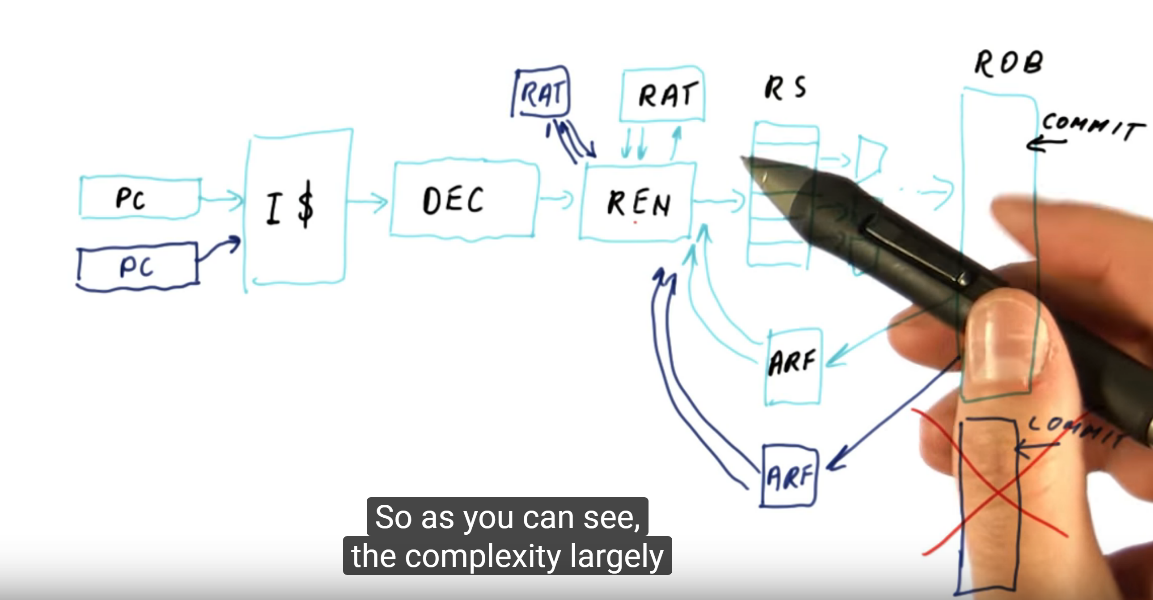


1. SMT (need register renaming to support)



SMT Hardware Change

1. Since we need to fetch instructions from multiple threads, need more PC.
2. Need more RAT for different threads
3. Since ROB is big and complex, typically not use separated ROB for threads, one thread may suffer from waiting other threads instruction to be committed (acceptable).
4. Architecture Register File (multiple for multi-thread)



SMT with TLB

1. Virtual Address Virtual Index (not for SMT)
2. Virtual Address Physical Index (TLB need to be thread aware, need thread tag)

Need TLB to compare thread number, virtual address to get actual address for data in cache to check hit

Since cache shared by multiple thread:

1. Communication between thread fast (cache hit)
2. Required cache shared exceed cache size (associativity set size) -> produce cache miss (cache trashing); If data in each thread can separately fit in cache, the performance for SMT worse than one thread at time. (due to cache miss)