ILP vs DLP vs TLP

ILP = Instruction-Level Parallelism
DLP = Data-Level Parallelism
TLP = Thread-Level Parallelism

SISD | MISD
SIMD | MIMD

- **ILP** from multiprogramming (multiple applications)
- **TLP** from multi-threaded applications
  - Run one application faster with multiple threads
  - pthreads, OpenMP, Cilk, TBB
MULTICORE VS COARSE-GRAIN MULTITHREADING

MULTICORE

COARSE-GRAIN MULTITHREADING

FINE-GRAIN MULTITHREADING

Hardware support to enable interleaving multiple threads on a single core at a very fine granularity.

We will discuss two variants of fine-grain multithreading:

- VERTICAL MULTITHREADING
- SIMULTANEOUS MULTITHREADING (SMT)

VERTICAL MULTITHREADING

Switch between treads at a cycle-by-cycle granularity.

State for all three threads kept in dedicated hardware.

Thread scheduling handled by hardware.
AT CODE EXAMPLE

```c
j = thread_id;
start = j * (n/threads);
for (int i = start; i < n/threads; i++)
```

VERTICAL MULTITHREADING MICRO-ARCHITECTURE

**Duplicate Arch State**

Extra load/sending logic

```
TO: (w r) 0, (r)
TI: (w r) 0, (r)
TA: mov r, r, r, r
TD: sw r, 0(r)
TT: sw r, 0(r)
TE: addw r, r, r, r
TF: addw r, r, r, r
TG: addw r, r, r, r
TH: addw r, r, r, r
TL: addw r, r, r, r
TT: load r+1, r+1
TH: load r+2, r+2
TA: load r+3, r+3
TI: load r+4, r+4
TE: load r+5, r+5
TF: load r+6, r+6
TG: load r+7, r+7
TH: load r+8, r+8

could potentially remove X->I bypass path!
new hide branch resolution delay latency
```
SCHEDULING POLICIES

1. STATIC FIXED INTERLEAVING

- Each of N threads execute one instruction every N cycles
- If thread is not ready to go can either:
  - Stall entire front-end
  - Insert dummy, but do not stall front-end
- Can potentially eliminate interlocking +天涯 network

2. DYNAMIC INTERLEAVING

- Hardware keeps track of which threads are ready
- Pick next thread to execute based on priority scheme

3. COARSE-CLOCK HARDWARE INTERLEAVING

- Use threads to hide occasional cache miss latency

1. 01201110112012012

2. 20110101011120012

3. 111111111111111

CACHE MISS
Simultaneous Multithreading (SMT)

SMT uses the fine grain control already present in an OOO superscalar processor to allow instructions from different threads to issue at the same time.

Add multiple fetch engines to enable fetching and decoding instructions from different threads.

SMT does not know about threads. Simply finds instructions that are ready to issue — these instructions may or may not be from different threads.

* SMT adapts to parallelism type
  - For applications with high TLP but no TUP, app can use entire width of the machine.
  - For applications with high TLP but less TLP, the width of the machine is shared across threads.

On this cycle we are issuing four instructions from three threads at the same time.
SMT Microarchitecture

As with vertical MT, architectural state must be duplicated. Microarchitectural state can either be:

- Duplicated at design time
- Hard partitioned at boot time
- Dynamically shared at execution time

Usually need to increase size of shared data structures IQ, PREF, LSQ

Thread scheduling

Fetch from thread with the least instructions in flight
Pipeline Diagram for Previous Code on Dual-Issue 502E with Two SMT Threads

```
10 LW r1, 0(r2)
10 Mul r3, r1, r4
11 LW r1, 0(r2)
11 Mul r3, r1, r4
10 Sw r3, 0(r5)
10 Addw r2, r2, r4
11 Sw r3, 0(r5)
11 Addw r2, r2, r4
10 Addw r5, r5, r4
10 Addw r1, r1, r4
11 Addw r5, r5, r4
11 Addw r1, r1, r4
12 Bjt2 #7, loop1
10 OpA
11 Bjt2 #7, loop1
11 OpA
12 LW r1, 0(r2)
12 Mul r3, r1, r4
```

**Issue Addw from T1**

**LW from T0**

**Issue OpA from T1**

**Mul from T0**