



Master Informatics Eng.

2017/18

A.J.Proença

From ILP to Multithreading

(most slides are borrowed)

AJProença, Advanced Architectures, MiEI, UMinho, 2017/18

1

Processor arch: beyond Instruction-Level Parallelism



- When exploiting ILP, goal is to minimize CPI
 - Pipeline CPI (*efficient to exploit loop-level parallelism*) =>
 - Ideal pipeline CPI + ✓
 - Structural stalls + ✓
 - Data hazard stalls + ✓
 - Control stalls + ✓
 - Memory stalls ... *cache techniques* ✓...
 - Multiple issue =>
 - find enough parallelism to keep pipeline(s) occupied
- Multithreading =>
 - find additional ways to keep pipeline(s) occupied
- Insert data parallelism features ...*(next set of slides)*

AJProença, Advanced Architectures, MiEI, UMinho, 2017/18

2

Multiple Issue and Static Scheduling

- To achieve $CPI < 1$, need to complete multiple instructions per clock cycle
- Solutions:
 - statically scheduled superscalar processors
 - VLIW (very long instruction word) processors
 - dynamically scheduled superscalar processors

Multiple Issue

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space; MIPS and ARM, including the ARM Cortex A8, Atom
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

EPIC: Explicitly Parallel Instruction Computer

Internal architecture of Intel P6 processors

The diagram illustrates a multi-processor system architecture, divided into three main functional blocks:

- Front End:** This block handles the initial stages of instruction processing. It includes:
 - L1 Instruction Cache** and **L1 Data Cache** at the top, which feed into the **Front End**.
 - Instruction Fetch** and **Decode** stages within the Front End.
- Execution Engine:** This block executes the decoded instructions. It includes:
 - Three parallel execution units (Ex.) represented by yellow, green, and pink boxes, each containing two execution slots (Ex.).
 - A **Retire** stage at the bottom, which completes the execution of instructions.
- Execution Unit:** This block provides the detailed internal structure of the execution units. It includes:
 - Instruction Control Unit:** Manages the flow of instructions, involving a **Retirement Unit** (with a **Register File**), **Fetch Control**, **Instruction Decode**, and an **Instruction Cache**. It exchanges **Address** and **Instrs.** with the cache.
 - Operations:** A central stage where instructions are executed, receiving input from the **Instruction Control Unit** and the **Execution Engine**.
 - Functional Units:** A row of specialized units including **Integer/Branch**, **General Integer**, **FP Add**, **FP Multi/Div**, **Load**, and **Store**.
 - Data Cache:** A separate cache for data, connected to the **Load** and **Store** units via **Addr.** and **Data** buses.
 - Feedback Loops:** Arrows indicate the flow of **Operation Results** back to the **Functional Units** and **Prediction OK?** signals back to the **Instruction Control Unit**.

5



Multithreading

- Performing multiple threads of execution in parallel
 - Replicate registers, PC/IP, etc.
 - Fast switching between threads
- Fine-grain multithreading / **time-multiplexed MT**
 - Switch threads after each cycle
 - Interleave instruction execution
 - If one thread stalls, others are executed
- Coarse-grain multithreading
 - Only switch on long stall (e.g., L2-cache miss)
 - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)



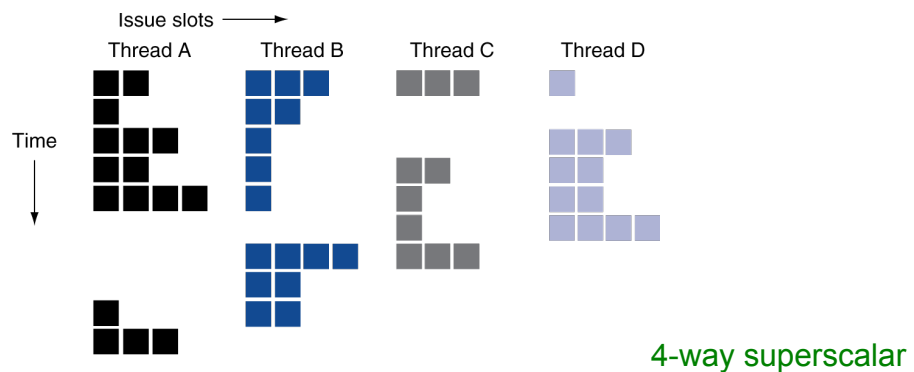
Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
 - Schedule instructions from multiple threads
 - Instructions from independent threads execute when function units are available
 - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
 - Two threads: duplicated registers, shared function units and caches

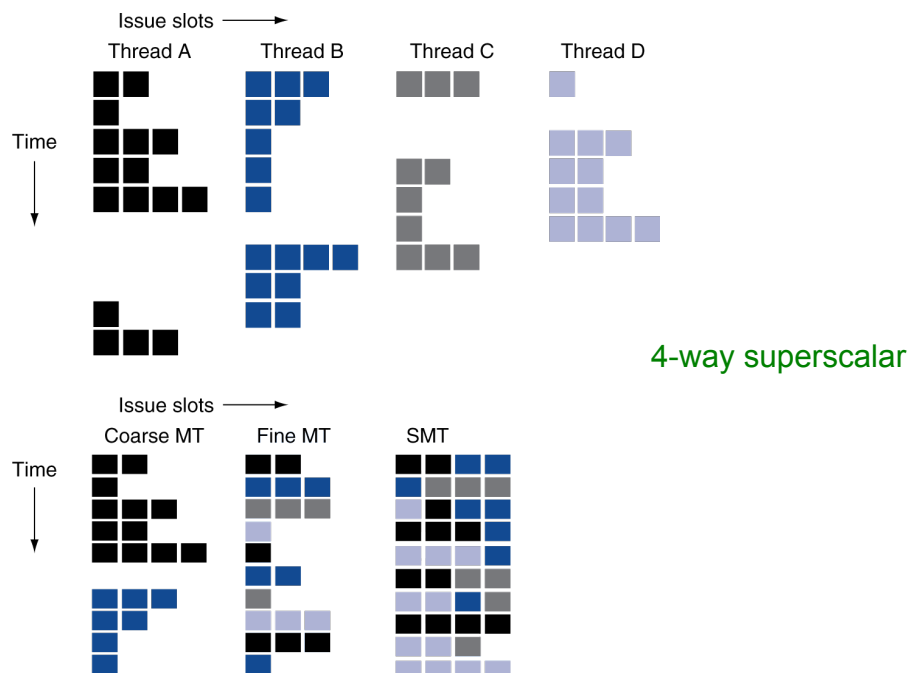
HT: Hyper-Threading, Intel trade mark for their SMT implementation
MT in Xeon Phi KNC: 4-way SMT with time-mux MT, **not HT!**



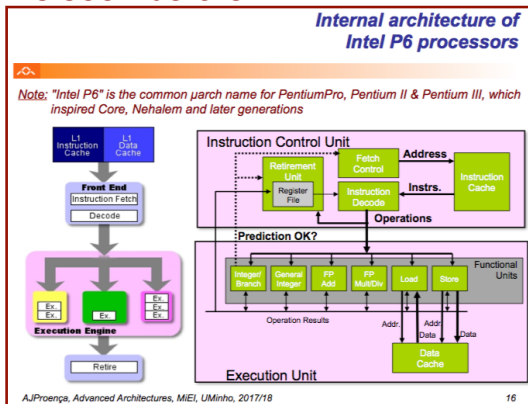
Multithreading Example



Multithreading Example

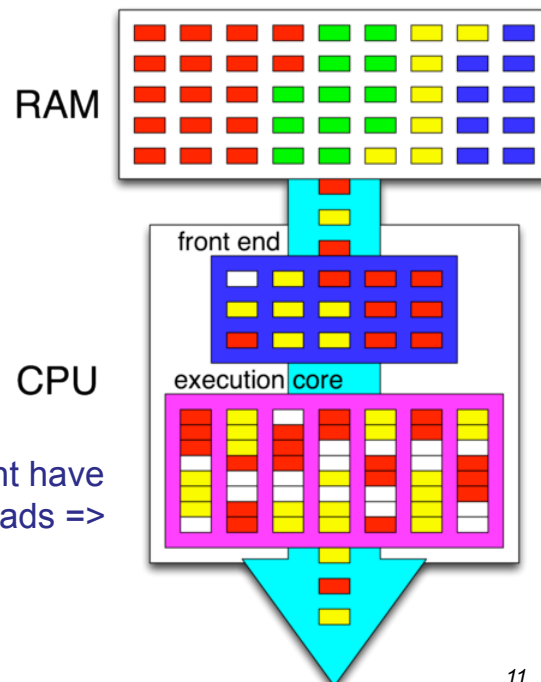


As seen before...



The pipelined functional units might have better use if shared among more threads =>

Note: white boxes are bubbles...



AJProença, Advanced Architectures, MIEI, UMinho, 2017/18

11

Reading suggestions (from CAQA 5th Ed)

- Concepts and challenges in ILP: section 3.1
- Exploiting ILP w/ multiple issue & static scheduling: 3.7
- Exploiting ILP w/ dyn sched, multiple issue & specul: 3.8
- Multithread: exploiting TLP on uniprocessors: 3.12
- Multiprocessor cache coherence and snooping coherence protocol with example: 5.2
- Basics on directory-based cache coherence: 5.4
- Models of memory consistency: 5.6
- A tutorial by Sarita Ave & K. Gharachorloo (see link at website)

AJProença, Advanced Architectures, MIEI, UMinho, 2017/18

12