CS223 : Computer Architecture & Organization

Lecture 31 [22.04.2022]

Static Scheduling & Superscalar Processors

Dr. John Jose

Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Introduction

- Pipelining overlaps execution of instructions
- Exploits Instruction Level Parallelism (ILP)
- There are two main approaches:
  - Compiler-based static scheduling
  - Hardware-based dynamic scheduling
- **Exploiting ILP, goal is to minimize CPI**
  - Pipeline CPI =Ideal (base) CPI + Structural stalls +Data hazard stalls +Control stalls

Compiler Techniques for Exposing ILP

- Find and overlap sequence of unrelated instruction
- Pipeline scheduling
  - Separate dependent instruction from the source instruction by pipeline latency of the source instruction
- Example:

```
for (i=999; i>=0; i=i-1)
  x[i] = x[i] + s;
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Stalls

```
Loop:   L.D      F0,0(R1)
        stall
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,0(R1)
        DADDUI R1,R1,#-8
        stall (assume integer load latency is 1)
        BNE R1,R2,Loop
```

```
for (i=999; i>=0; i=i-1)
    x[i] = x[i] + s;
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Scheduling

```
Loop:   L.D      F0,0(R1)
        stall
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,0(R1)
        DADDUI R1,R1,#-8
        stall (assume integer load latency is 1)
        BNE R1,R2,Loop
```

Scheduled code:

```
Loop:   L.D  F0,0(R1)
        DADDUI R1,R1,#-8
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,8(R1)
        BNE R1.R2.Loop
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Loop Unrolling

- **Loop unrolling**

```
Loop:   L.D F0,0(R1)
        ADD.D F4,F0,F2
        S.D F4,0(R1) ;drop DADDUI & BNE
        L.D F6,-8(R1)
        ADD.D F8,F6,F2
        S.D F8,-8(R1) ;drop DADDUI & BNE
        L.D F10,-16(R1)
        ADD.D F12,F10,F2
        S.D F12,-16(R1) ;drop DADDUI & BNE
        L.D F14,-24(R1)
        ADD.D F16,F14,F2
        S.D F16,-24(R1)
        DADDUI R1,R1,#-32
        BNE R1,R2,Loop
```

Scheduled code:

```
Loop:   L.D  F0,0(R1)
         DADDUI R1,R1,#-8
        ADD.D F4,F0,F2
        stall
        stall
        S.D F4,8(R1)
        BNE R1,R2,Loop
```

number of live registers ?

Loop Unrolling/Pipeline Scheduling

❖ Loop unrolling

```
Loop:   L.D F0,0(R1)
        ADD.D F4,F0,F2
        S.D F4,0(R1)
        L.D F6,-8(R1)
        ADD.D F8,F6,F2
        S.D F8,-8(R1)
        L.D F10,-16(R1)
        ADD.D F12,F10,F2
        S.D F12,-16(R1)
        L.D F14,-24(R1)
        ADD.D F16,F14,F2
        S.D F16,-24(R1)
        DADDUI R1,R1,#-32
        BNE R1,R2,Loop
```

❖ Scheduled unrolled loop:

```
Loop:   L.D F0,0(R1)
        L.D F6,-8(R1)
        L.D F10,-16(R1)
        L.D F14,-24(R1)
        ADD.D F4,F0,F2
        ADD.D F8,F6,F2
        ADD.D F12,F10,F2
        ADD.D F16,F14,F2
        S.D F4,0(R1)
        S.D F8,-8(R1)
        DADDUI R1,R1,#-32
        S.D F12,16(R1)
        S.D F16,8(R1)
        BNE R1,R2,Loop
```

Strip Mining

- Unknown number of loop iterations?
  - Goal: make k copies of the loop body Number of iterations = n
  - Generate pair of loops:
    - First executes n mod k times
    - Second executes n / k times
    - Strip mining
  - Example : Let n=35, k=4
    - Loop 1 execute 3 times
    - Loop 2 execute 8 times by unrolling 4 copies per iteration

Steps in Loop Unrolling and Scheduling

- Determine that unrolling the loop would be useful.
- Identify independency of loop iterations.
- Use different registers to avoid unnecessary constraints put in on same computations.
- Eliminate the extra test and branch instructions and adjust the loop termination and iteration code.
- Determine whether the loads and stores from different iterations are independent.
- Schedule the code, preserving any dependences needed to yield the same result as the original code.

Loop Unrolling & Pipeline Scheduling

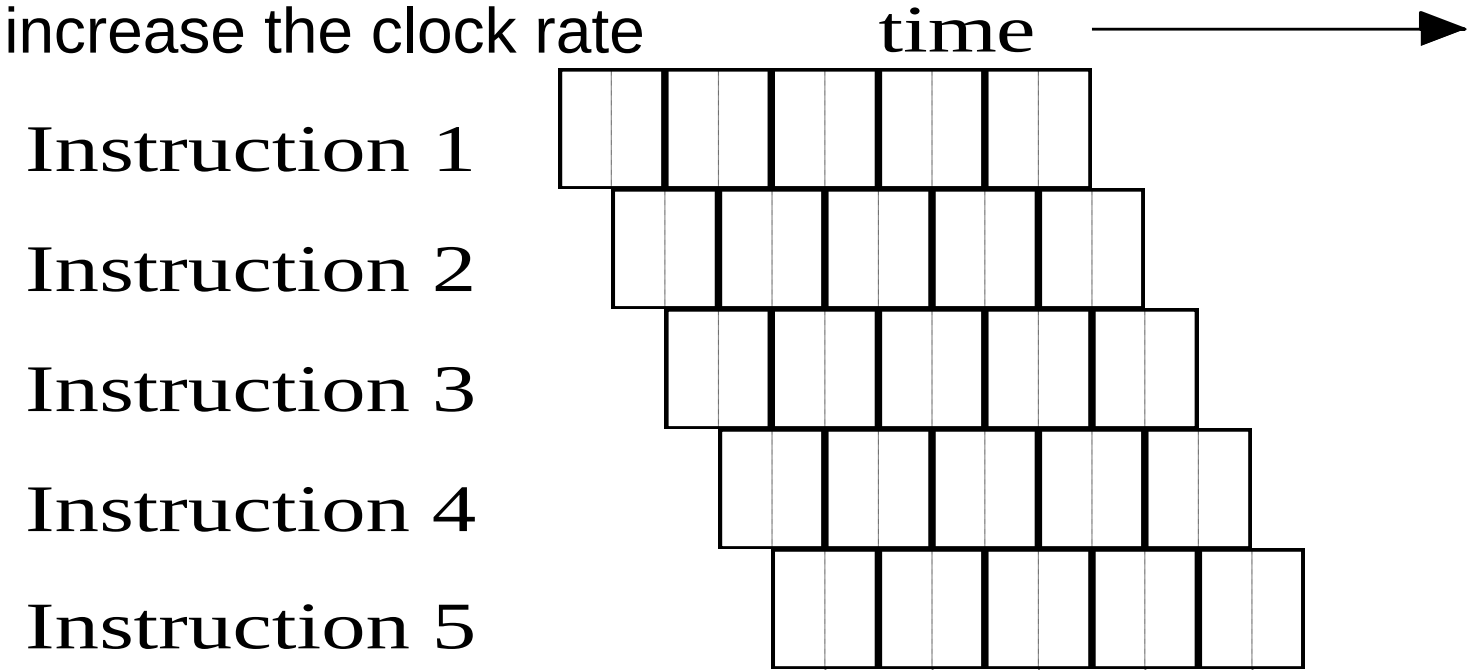
- **Limitations of loop unrolling:**
  - Code size limitations – I-cache miss
  - Compiler limitations – register pressure

Advanced Pipelining

- Increase the depth of the pipeline to increase the clock rate – **superpipelining**
- Fetch (and execute) more than one instructions at one time (expand every pipeline stage to accommodate multiple instructions) – **multiple-issue (super scalar)**
- Launching multiple instructions per stage allows the instruction execution rate, CPI, to be less than 1

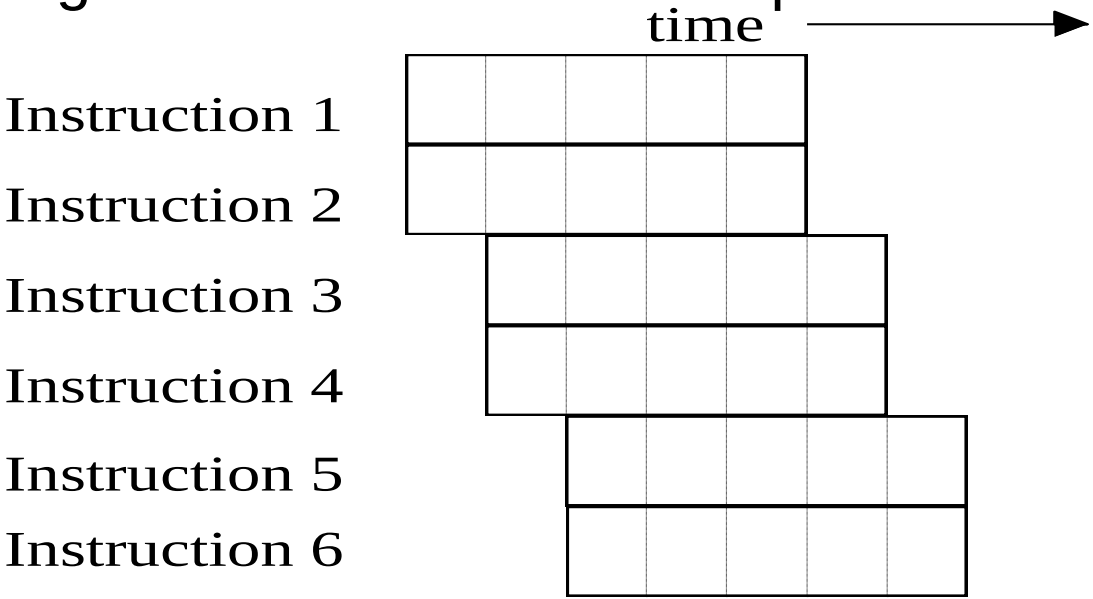
Superpipelining

- superpipelining -Increase the depth of the pipeline to increase the clock rate



Superscalar – multiple-issue

- Fetch (and execute) more than 1 instructions at one time
- Expand every stage to accommodate multiple instructions



Multiple Issue and Static Scheduling

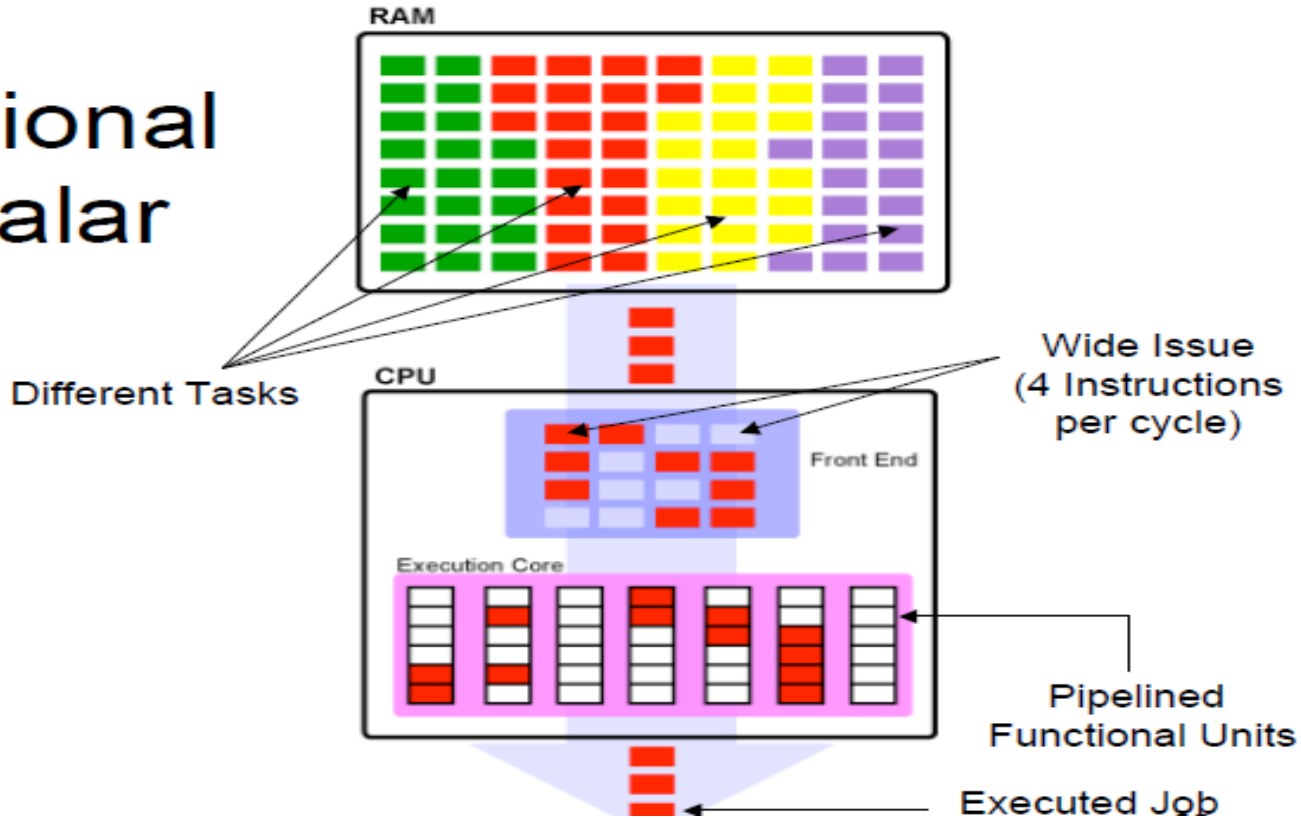
- Basic pipeline will give only a min CPI of 1
- To achieve CPI < 1, need to complete multiple instructions per clock **(superscalar processors)**
- Multiple functional units and Multiple Issue
- Solutions:
  - Statically scheduled superscalar processors
  - VLIW (very long instruction word) processors
  - Dynamically scheduled superscalar processors

Extreme Optimization

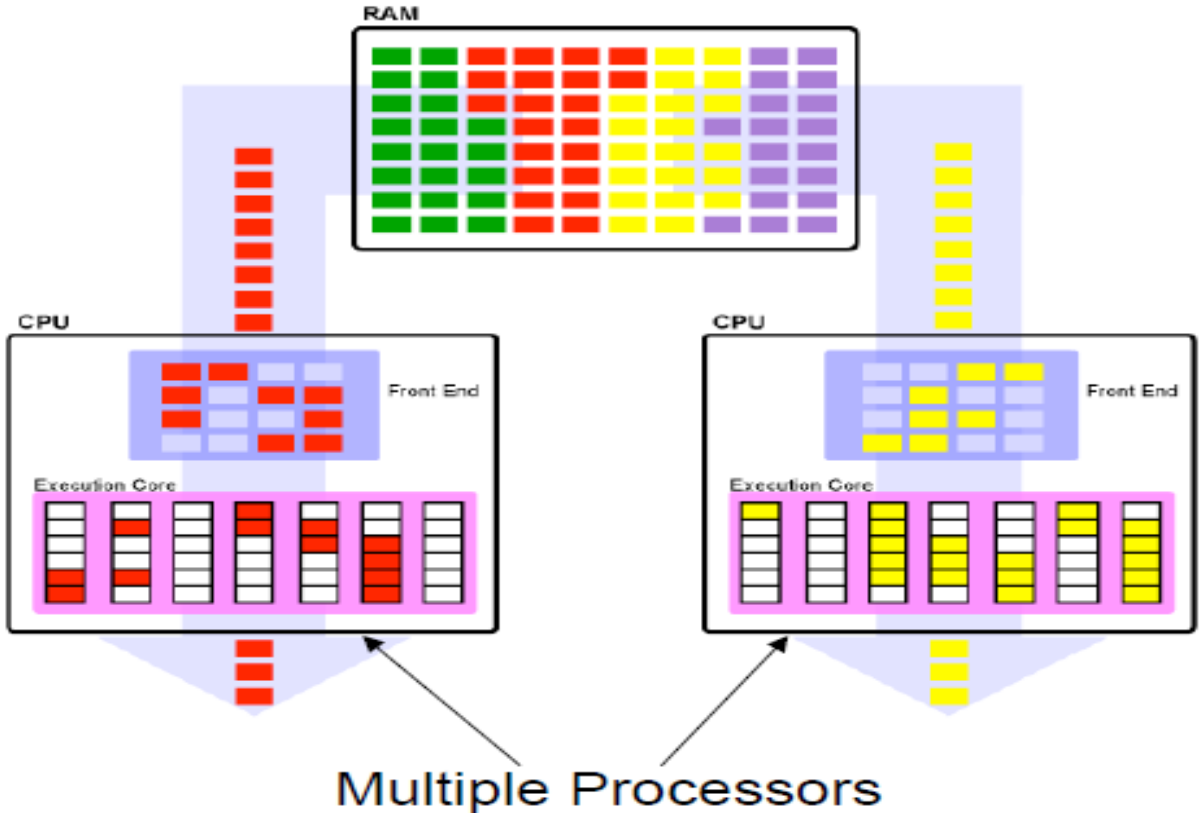
- Modern micro-architectures uses this triple combination:
  - Dynamic scheduling + multiple issue + speculation
- Dependency between instructions issued in same clock.
- Register read for multiple instructions in parallel.
- Complex Issue logic to check dependencies and hazards.
- Assign reservation stations and ROB entries in a cycle.

Superscalar Processor

Conventional
Superscalar



Symmetric Multiprocessor

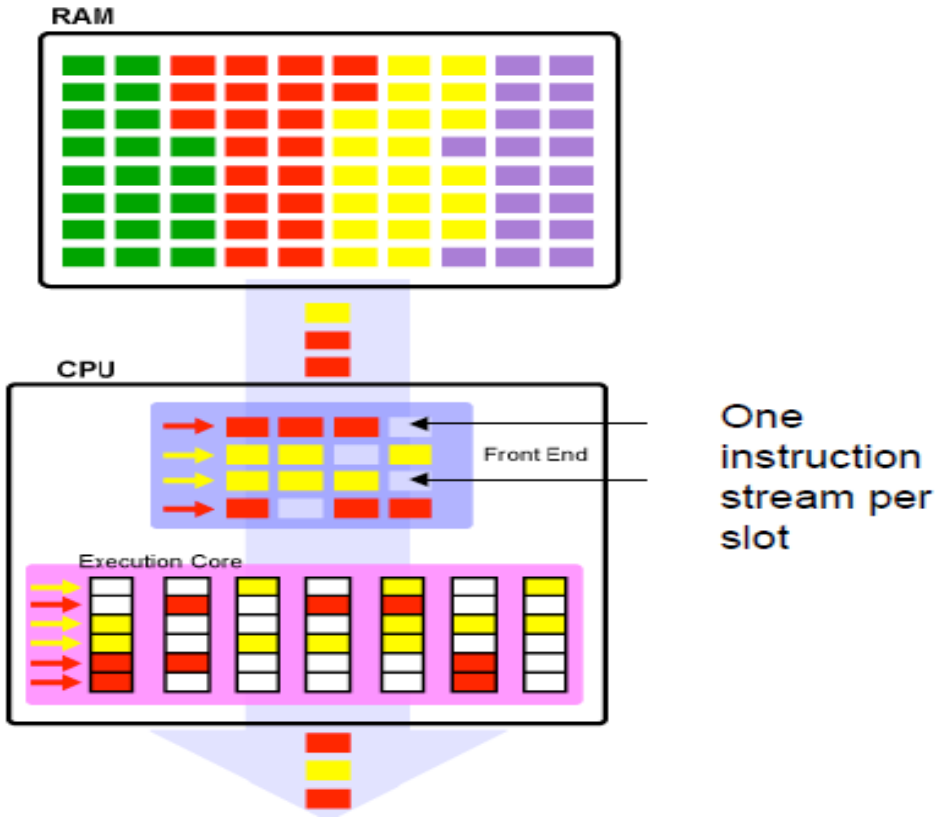


End of exploiting ILP ?

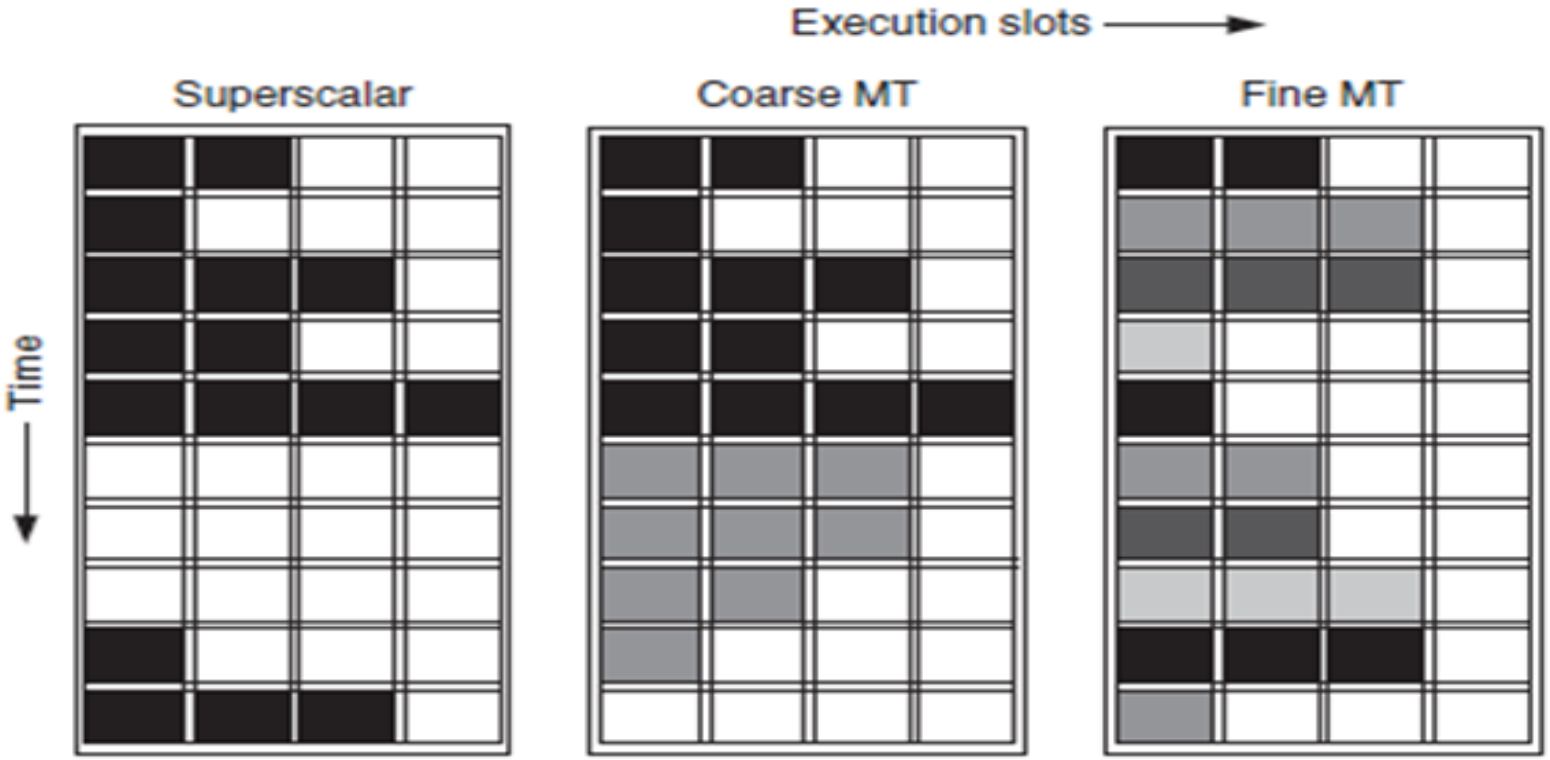
- We have tried Tomasulo's superscalar approach with ROB, speculation with aggressive branch prediction and multi issue
- At very high issue rates cache misses that go to L2 and off chip can not be hidden by ILP.
- How to cover such long memory stalls ?
- **Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion**

Multithreading

Multithreading

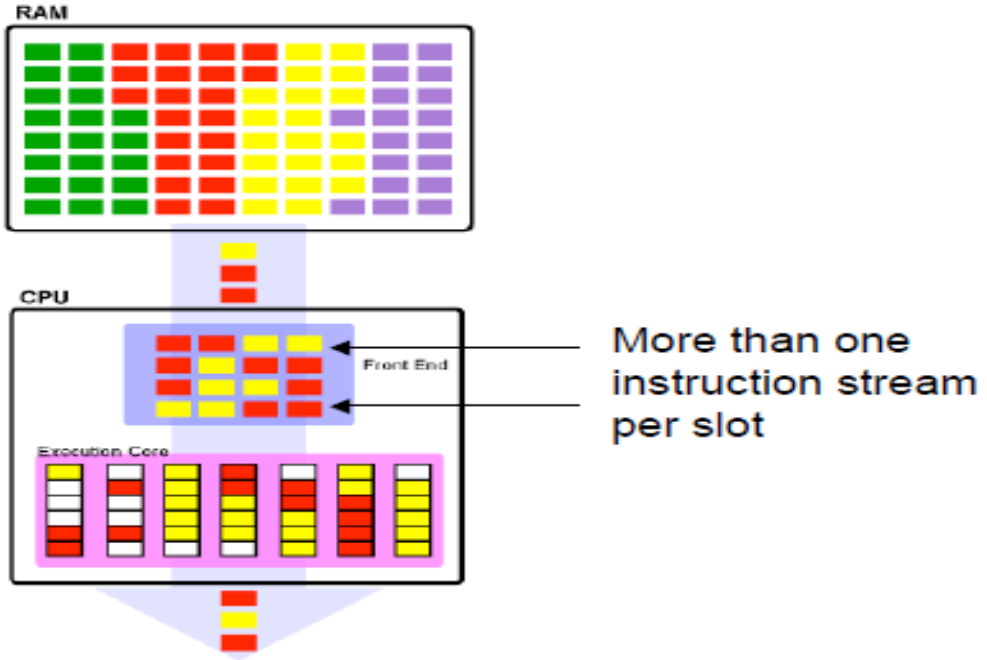


Multithreading

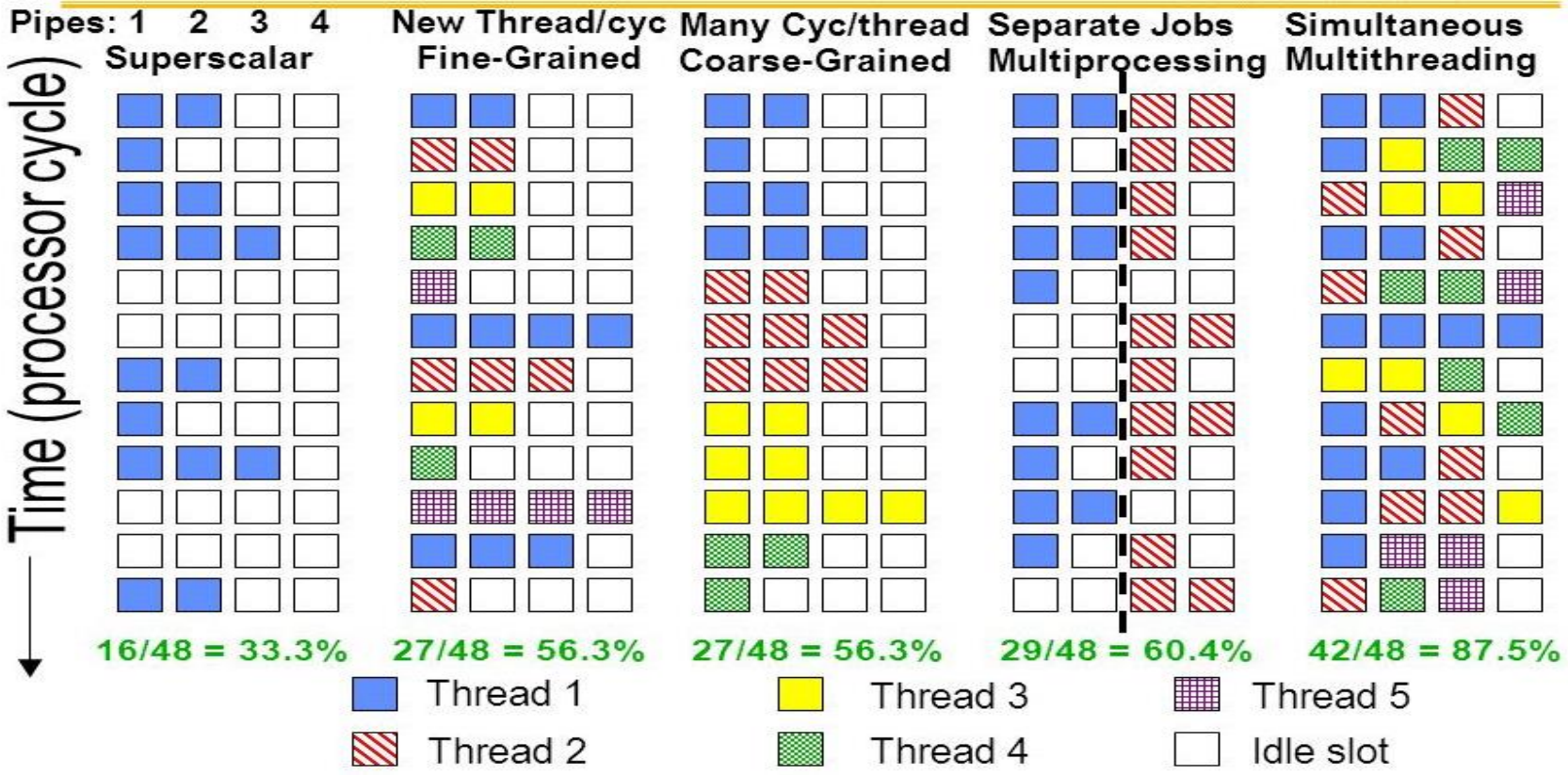


Hyperthreading /SMT

Hyperthreading



Comparison of Resource Utilization



Reference

- **Computer Architecture-A Quantitative Approach** (5th edition), John L. Hennessy, David A. Patterson, Morgan Kaufman.
- Chapter 3: **Instruction Level Parallelism and its Exploitation**
  - Section C1: Instruction Level Parallelism - Concepts and Challenges
  - Section C2: Basic Compiler Techniques for Exposing ILP
- **NPTEL Video Link:**
- **https://tinyurl.com/y8qtysqu**
- **https://tinyurl.com/y3qzgpy5**

johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/