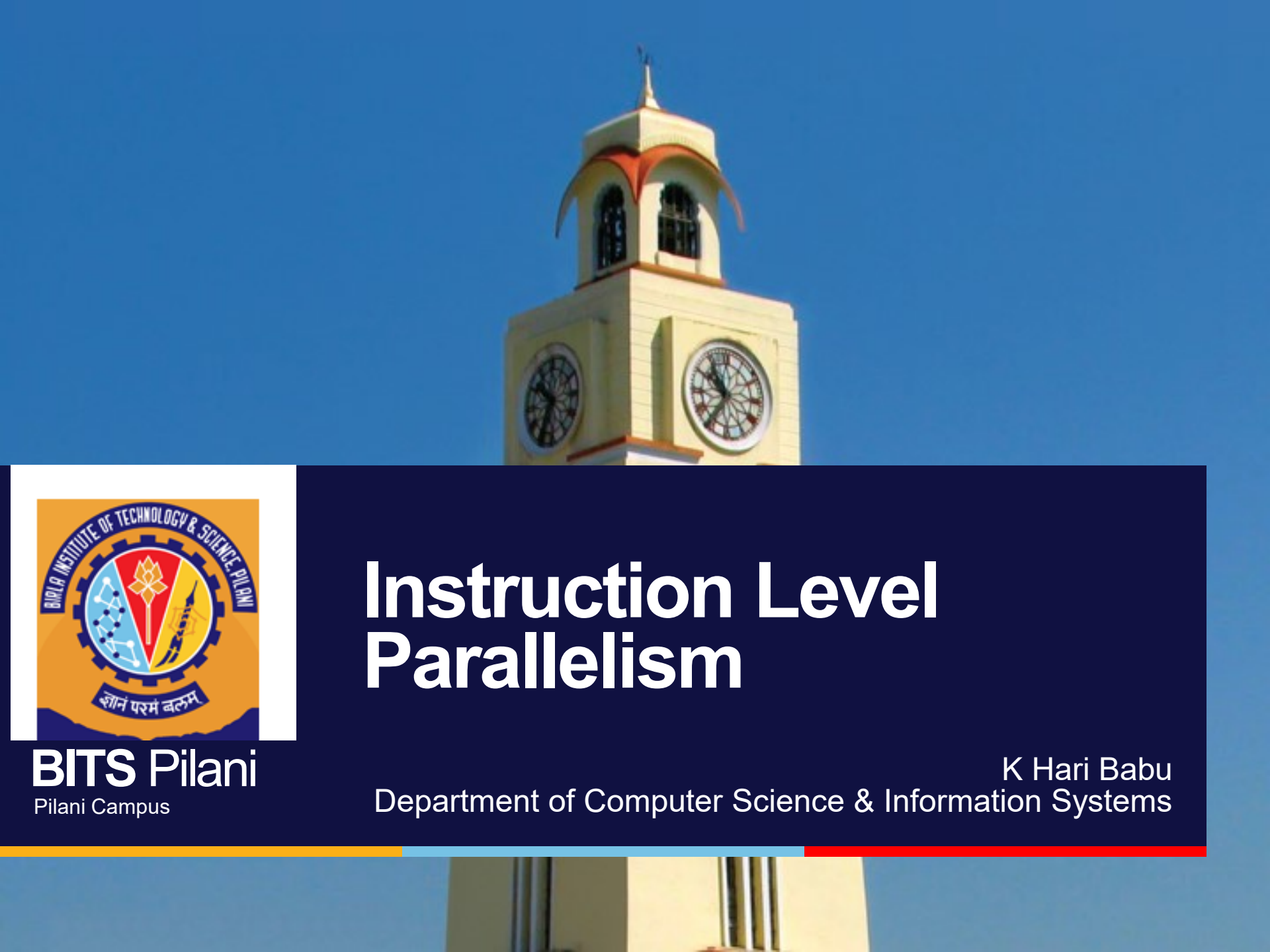

Instruction Level Parallelism

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

Concepts and Challenges (R3 3.1)

Instruction Level Parallelism (ILP)

- All processors since about 1985 use pipelining to overlap the execution of instructions and improve performance
  - This potential overlap among instructions is called instruction-level parallelism (ILP), since the instructions can be evaluated in parallel

Approaches to Improve ILP

- There are two approaches to improve ILP:
  - an approach that relies on hardware to help discover and improve the parallelism dynamically
  - and an approach that relies on software technology to find parallelism, statically at compile time
- Processors using the dynamic, hardware-based approach
  - Intel Pentium series, dominate in the market
- Processors using the static approach
  - Intel Itanium
  - ARM Cortex-A8

CPI (Cycles Per Instruction)

BITS Pilani

- The value of the CPI (cycles per instruction) for a pipelined processor is the sum of the base CPI and all contributions from stalls:
  - Pipeline CPI = Ideal pipeline CPI + Structural stalls + Data hazard stalls + Control stalls
  - The ideal pipeline CPI is a measure of the maximum performance attainable by the implementation
- By reducing each of the terms of the right-hand side, we
- minimize the overall pipeline CPI or, alternatively, increase the IPC (instructions per clock)

Terminology

- Basic Block - That set of instructions between entry points and between branches. A basic block has only one entry and one exit. Typically this is about 6 instructions long.

- Loop Level Parallelism - that parallelism that exists within a loop. Such parallelism can cross loop iterations.

- Loop Unrolling - Either the compiler or the hardware is able to exploit the parallelism inherent in the loop.

ILP Challenges

- How many instructions can we execute in parallel?
  - Typical MIPS programs have 15-25 % branch instruction
  - Between 3 to 6 instructions execute between a pair of branches
    - Average basic block size: 3 to 6
  - The amount of overlap that can be within a basic block is likely to be less than the average basic block size
  - Most of the instructions within basic block will have data dependencies.
    - Stall / Reorder
- To obtain substantial performance enhancements, ILP across multiple basic blocks is required
  - What stops us from doing this?
    - Control dependencies

Dependencies

- Dependence
  - Data Dependence
  - Name Dependencies
    - Anti-dependence
    - Output dependence
  - Control dependence

Data Dependences and Hazards

- Determining how one instruction depends on another is critical to determining how much parallelism exists in a program
  - If two instructions are parallel, they can execute simultaneously in a pipeline of arbitrary depth without causing any stalls, assuming the pipeline has sufficient resources (and hence no structural hazards exist).
  - If two instructions are dependent, they are not parallel and must be executed in order, although they may often be partially overlapped.
- The key in both cases is to determine whether an instruction is dependent on another instruction

Data Dependences

BITS Pilani

- An instruction j is data dependent on instruction i if either of the following holds
  - instruction i produces a result that may be used by instruction j or
  - instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i
    - The second condition simply states that one instruction is dependent on another if there exists a chain of dependences of the first type between the two instructions
  - This dependence chain can be as long as the entire program

Challenges

- Dependences are a property of programs.
  - Whether a given dependence results in an actual hazard being detected and whether that hazard actually causes a stall are properties of the pipeline organization
  - This difference is critical to understanding how instruction-level parallelism can be enhanced.
- A data dependence conveys three things:
  - (1) the possibility of a hazard, (2) the order in which results must be calculated, and (3) an upper bound on how much parallelism can be possible
- Since a data dependence can limit the amount of instruction-level parallelism, a major focus is on overcoming these limitations

Data Dependences through Registers / Memory

BITS Pilani

- Dependences through registers are easy:

  lw r10,10(r11)

  add r12,r10,r8

  just compare register names.

- Dependences through memory are harder:

  sw r10,4 (r2)

  lw r6,0(r4)

  is r2+4 = r4+0? If so they are dependent, if not, they are not.

- There are HW techniques as well as compiler techniques to detect dependences through memory

Name Dependence #1: Anti-dependence

BITS Pilani

- Name dependence
  - when 2 instructions use the same register or memory location, called a *name*, but no flow of data between the instructions is associated with that name
- 2 versions of name dependence
- $Instr_J$ writes operand before $Instr_I$ reads it
  - `I: sub r4,r1,r3`
  - `J: add r1,r2,r3`
  - `K: mul r6,r1,r7`
- Called an “anti-dependence” by compiler writers. This results from reuse of the name “r1”
- If anti-dependence caused a hazard in the pipeline, it is called as
  - Write After Read (WAR) hazard

Name Dependence #2: Output dependence

BITS Pilani

- InstrJ writes operand before InstrI writes it
  - `I: sub r1,r4,r3`
  - `J: add r1,r2,r3`
  - `K: mul r6,r1,r7`

- Called an “output dependence” by compiler writers. This also results from the reuse of name “r1”
- If anti-dependence caused a hazard in the pipeline, it is called as
  - Write After Write (WAW) hazard

Instruction Dependence Examp

```
1    L.D       F0, 0 (R1)
2    ADD.D     F4, F0, F2
3    S.D       F4, 0(R1)
4    L.D       F0, -8(R1)
5    ADD.D     F4, F0, F2
6    S.D       F4, -8(R1)
```

- For the following code identify all data and name dependence between instructions
- True Data Dependence:
  - Instruction 2 depends on instruction 1 (instruction 1 result in F0 used by instruction 2), Similarly, instructions (4,5)
  - Instruction 3 depends on instruction 2 (instruction 2 result in F4 used by instruction 3), Similarly, instructions (5,6)
- Name Dependence:
  - Output Name Dependence (WAW):
    - Instruction 1 has an output name dependence over result register (name) F0 with instructions 4
    - Instruction 2 has an output name dependence over result register (name) F4 with instructions 5
  - Anti-dependence (WAR):
    - Instruction 2 has an anti-dependence with instruction 4 over register (name) F0 which is an operand of instruction 1 and the result of instruction 4
    - Instruction 3 has an anti-dependence with instruction 5 over register (name) F4 which is an operand of instruction 3 and the result of instruction 5

ILP and Data Hazards

- HW/SW must preserve program order:
  the order in which instructions would execute in if executed sequentially 1 at a time as determined by original source program
- HW/SW goal: enhance parallelism by preserving program order only where it affects the outcome of the program
  - True data dependencies
- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so that instructions do not conflict
  - Register renaming resolves name dependence for registers
  - Either by compiler (SW) or by HW

Control Dependencies

BITS Pilani

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order
  - S1 is control dependent on p1, and
  - S2 is control dependent on p2 but not on p1.

```
1  if p1 {
2      S1;
3  };
4  if p2 {
5      S2;
6  }
```

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, if can do so without affecting correctness of the program

```
1      DADDU r2,r3,r4
2      beqz r2,l1
3      lw r1,0(r2)
4  l1:
```

- Can we move lw before the branch?
  - Data dependency (RAW) doesn't prevent it
- Two properties critical to program correctness are exception behavior and data flow

Preserving the exception behavior

BITS Pilani

- Corollary I:
  - Any changes in the ordering of instructions should not change how exceptions are raised in a program.
  - Relaxed to: reordering of instruction execution should not cause any new exceptions.

```
1        DADDU r2,r3,r4
2        beqz r2,l1
3        lw r1,0(r2)
4   l1:
```

- Can we move lw before the branch?
  - No as it may raise a memory protection exception
  - Yet in HW/SW speculative execution, this may be done but with support for undoing in case of branch taken

Preserving the data flow

BITS Pilani

- What can you say about the value of r1 used by the OR instruction?

```
1       DADDU R1,R2,R3
2       BEQZ R4,L
3       DSUBU R1,R5,R6
4   L:  …
5       OR R7,R1,R8
```

- The second property preserved by maintenance of data dependences and control dependences is the data flow
  - The data flow is the actual flow of data values among instructions that produce results and those that consume them
  - In this example, the value of R1 used by the OR instruction depends on whether the branch is taken or not. Data dependence alone is not sufficient to preserve correctness
    - The DSUBU instruction cannot be moved above the branch

Preserving the data flow

- Corollary II:
  - Preserving data dependences alone is not sufficient when changing program order. We must preserve the data flow.
  - Data flow: actual flow of data values among instructions that produce results and those that consume them.
- These two corollaries together allow us to execute instructions in a different order and still maintain the program semantics.
- This is the foundation upon which ILP processors are built.

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You