Module-3
CSEN 3104
Lecture 28
26/09/2019

Dr. Debranjan Sarkar

Branch Prediction

Dependence Analysis

- To execute several program segments in parallel, each segment is to be independent of the other segments
- 3 types of dependences:
  - Data dependence (The ordering relationships between statements)
  - Control dependence (Order of execution of statements cannot be determined before run time)
  - Resource dependence (refers to the conflicts in using shared resources, such as integer units, floating-point units, registers etc.)
- Dependence analysis determines whether it is safe to reorder or parallelize statements

Control Dependency

- Control dependency is a situation in which a program instruction executes if the previous instruction evaluates in a way that allows its execution
- In the following example, the statement S2 runs only if the predicate in S1 is false

```
S1  if x > 2 go to L1
S2  y = 3
S3  L1:    z = y + 1
```

- The statement S2 is said to be control dependent on S1 (written as S1 δ^c S2) as S2's execution is conditionally guarded by S1

Control Dependence in a loop

- Loop Independent Dependence
- A statement in one iteration of a loop depends only on a statement in the same iteration of the loop

```
    for (i = 0; i < 4; i++)
    {
S1:    b[i] = 8;
S2:    a[i] = b[i] + 10;
    }
```

- Loop independent dependence between statements S1 and S2 in the same iteration

Control Dependence in a loop

- Loop-dependent Dependence
- A statement in one iteration of a loop depends on a statement in a different iteration of the same loop

```
    for (i = 0; i < 4; i++)
    {
S1:    b[i] = 8;
S2:    a[i] = b[i-1] + 10;
    }
```

- Loop-dependent dependence exists between i^{th} iteration of statement S2 and

  $(i-1)^{th}$ iteration of S1

- Statement S2 cannot proceed until statement S1 in the previous iteration finishes

Control Dependency

- Conditional branch statements cannot be resolved until run time
- Different paths taken after a conditional branch may introduce or eliminate data dependence among instructions
- Control dependence often prohibits parallelism from being exploited
- Techniques to get around the control dependencies in order to exploit more parallelism:
  - Compiler techniques
  - Hardware branch prediction techniques

Handling Control Dependency

- How to Handle Control Dependences?
- Critical to keep the pipeline full with correct sequence of dynamic instructions.
- Potential solutions if the instruction is a control-flow instruction:
  - Stall the pipeline until we know the next fetch address
  - Guess the next fetch address (branch prediction)
  - Employ delayed branching (branch delay slot)

Branch Prediction

- 3 different kinds of branches:
  - Forward conditional branches (PC is changed to point to an address forward in the instruction stream)
  - Backward conditional branches (PC is changed to point backward in the instruction stream)
  - Unconditional branches (e.g. jumps, procedure calls and returns)
- Branch prediction attempts to guess whether a conditional jump will be taken or not
- 2 Branch prediction schemes
  - Static
  - Dynamic

Static Branch Prediction

- Based on the information that was gathered before the execution of the program
- Predicts always the same direction for the same branch during the whole program execution
- 2 types
  - Hardware-fixed prediction
  - Compiler-directed prediction
- Hardware-fixed direction mechanisms can be:
  - Predict always not taken
  - Predict always taken
  - For ‘Backward Branch’ predict “Taken”, for ‘Forward Branch’ predict “Not Taken”
- Sometimes a bit in the branch opcode allows the compiler to decide the prediction direction
- All decisions are made at compile time, not at run time

Using Branch statistics for static prediction

- Nearly 60% of the forward conditional branches are taken
- About 85% of the backward conditional branches are taken (loops)
- So, a Static Predictor can just look at the offset (distance forward or backward from current PC) for conditional branches as soon as the instruction is decoded
- Backward branches will be predicted to be taken, since that is the most common case
- The accuracy of the static predictor will depend on the type of code being executed, as well as the coding style used by the programmer

Dynamic Branch Prediction

- It is based on recent branch history to predict whether or not the branch would be taken next time when it occurs
- During the start-up phase of the program execution
  - a static branch prediction might be effective
  - the history information is gathered
- When history information is gathered, dynamic branch prediction gets effective
- In general, dynamic branch prediction gives better results than static branch prediction, but at the cost of increased hardware complexity

Dynamic Branch Prediction (1-bit)

- Example

```
for (i = 0; i < 5; i++)
{
a[i] =a[i] + 5;
}
```

- We would predict whether branch is to be taken or not
- In 1-bit branch prediction method, there are 2 states (show figure)
  - 0 -> Not likely to be taken state. If in ‘0’ state, predict that branch is not to be taken (NT)
  - 1 -> Likely to be taken state. If in ‘1’ state, predict that branch is to be taken (T)
- Let us assume that initial state is ‘1’
- So initial prediction is “Branch is to be taken (T)”

Dynamic Branch Prediction (1-bit)

- Value of i 0 1 2 3 4 5 6 7 8
- Present state 1 1 1 1 1 1 0 1 1
- Prediction T T T T T T NT T T
- Actual T T T T T NT T T T
- Next state 1 1 1 1 1 0 1 1 1
- Right / wrong √ √ √ √ √ X X √ √

- 2 mis-predictions in case of anomaly
- To reduce the number of mis-predictions, we use 2-bit branch prediction

Dynamic Branch Prediction (2-bit)

- In 2-bit branch prediction method, there are 4 states (show figure)
  - 00 -> strongly not likely to be taken state
  - 01 -> weakly not likely to be taken state
  - 10 -> weakly likely to be taken state
  - 11 -> strongly likely to be taken state
- MSB indicates whether branch is likely
  - to be taken (MSB = 1), or
  - not to be taken (MSB = 0)
- LSB indicates whether the prediction is
  - strong (LSB = 0), or
  - weak (LSB =1)

Dynamic Branch Prediction (2-bit)

- Let us assume that initial state is ‘10’
- So initial prediction is “Branch is to be taken” and the prediction is correct
- Value of i 0 1 2 3 4 5 6 7
- Present state 10 11 11 11 11 11 10 11
- Prediction T T T T T T T T
- Actual T T T T T NT T T
- Next state 11 11 11 11 11 10 11 11
- Right / wrong √ √ √ √ √ X √ √

- Only 1 mis-prediction in case of anomaly

Branch History Buffer (BHB)

- It is a table with three (3) columns (show figure)
  - Branch Instruction Address
  - Branch Target Address
  - Branch Prediction statistics
- If the Branch Prediction Statistics indicates that the branch is likely to be taken, in the 2nd cycle, the instruction from the target address is fetched.
- If the prediction is not correct, the prediction statistics is changed accrordingly

Thank you