Lesson 9

Compiler ILP

This lesson discusses some of the techniques used by compilers to achieve greater ILP. Improving instruction scheduling and reducing the number of instructions are the basic goals of compiler ILP.

Tree Height Reduction

Tree height refers to the dependence chains in a list of instructions. Tree height reduction is re-grouping the calculations in an instruction list to reduce the dependencies. This method only works for calculations that are associative.

R8 = (R2 + R3) + (R4 + R5)

Add R8, R2, R3
Add R8, R8, R4
Add R8, R8, R5

Tree Height Reduction

Add R8, R2, R3
Add R7, R4, R5
Add R8, R8, R7

Making Independent Instructions Easier to Find

Techniques to make independent instructions easier to find:

- Instructions Scheduling
- Loop Unrolling
- ~~Trace Scheduling~~ ← Not really covered

Instruction Scheduling (by the compiler)

When there is a dependency between instructions, there is often a need for the processor to stall between instructions. A stall can be replaced with an independent instruction, as long as the program produces the correct result. Sometimes the correct result requires modifying additional instructions.

Loop:
LW R2, 0(R1)
add R2, R2, R0
SW R2, 0(R1)
addi R1, R1, 4
BNE R1, R3, Loop

↓ Instruction Scheduling

Loop:
LW R2, 0(R1)
addi R1, R1, 4
add R2, R2, R0
SW R2, -4(R1)
BNE R1, R3, Loop

The modifications can include -- changing the address offsets and changing the destination registers.

- Address offset changes occur when the instruction moves relative to an address fetch.
- Destination register changes occur when an instruction move causes a register to be re-written earlier in the program than expected.

we can rewrite which register we're writing to if it allows us to move instructions around for parallelizati

Scheduling and If Conversion

← See "predication"

If conversion helps in two ways:

1. It reduces the number of branches. Both branches of the if statement are executed using predication. (conditional move instructions)
2. It allows for greater flexibility in instruction scheduling. Since all branches are executed instructions can be moved with greater ease, which will reduce the number of stalls.

If Convert a Loop

A loop cannot be converted using the If conversion, but it can be improved using loop unrolling.

Loop Unrolling

Loop unrolling creates a loop with fewer iterations by having each iteration do the work of two or more iterations. This will reduce the number of branches that need to be executed and it increases the number of instructions that can be considered during the instruction scheduling.

Loop Unrolling benefits

The benefits of loop unrolling:

-Reduces the overall number of instructions that need to be executed for the loop. This is done by reducing the loop overhead. ← ALWAYS

-Reduces the CPI. Once the loop is unrolled, the scheduler has more flexibility. leading to a better CPI. ← SOMETIMES. This is dependant on the actual program code.

Execution Time = (# instructions) × (CPI) × (Cycle Time)

Unrolling Downside

loop unrolling overall improves the execution time

The downsides of unrolling include:

- Code bloat
- What if the number of iterations are unknown at the start of the loop?
- What if the number of iterations are not a multiple of N?

There are solutions to these problems - but they will not be discussed in this class.

Function Call Inlining

Inlining for functions is similar to loop unrolling.
Inlining involves removing the function call and just replacing it with the body of the function.
Obviously this is only an improvement for small functions. This will eliminate the function call, the return statement, and allow for improved instruction scheduling.

Function Call Inlining Downside

Inlining Downside:

Code bloat. The more inlining, the more code bloat.

Inlining should be done on functions that are small.

Other IPC Enhancing Compiler Stuff

There are other methods for enhancing IPC, they are not discussed in this course.

Software Pipelining
Trace Scheduling