Enhanced Pipeline Scheduling

Overview of Enhanced Pipeline Scheduling

- Compiler Scheduling for ILP
- Basic Idea of EPS and a Generic Example
- Aggressive DAG scheduling techniques

Review of Compiler Instruction Scheduling

Extract independent instructions from sequential code and group them for parallel execution

- We expect them to be executed in parallel by H/W

Scheduling within BB is not enough to make H/W busy

- Need advanced techniques that schedule beyond BBs

Classified into two categories based on code type

- Acyclic code: Global DAG Scheduling
- Cyclic code: Software Pipelining

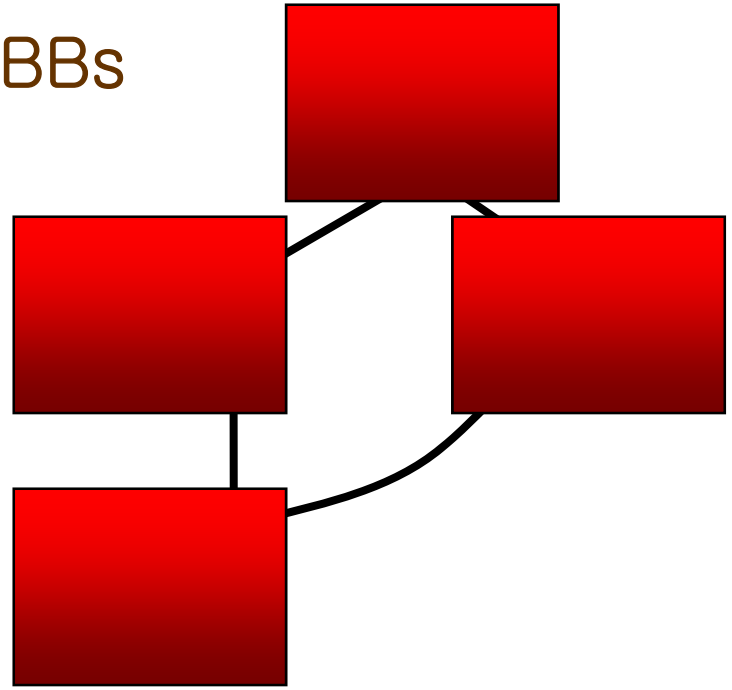
DAG Scheduling

Schedule beyond BB boundaries in a DAG of BBs

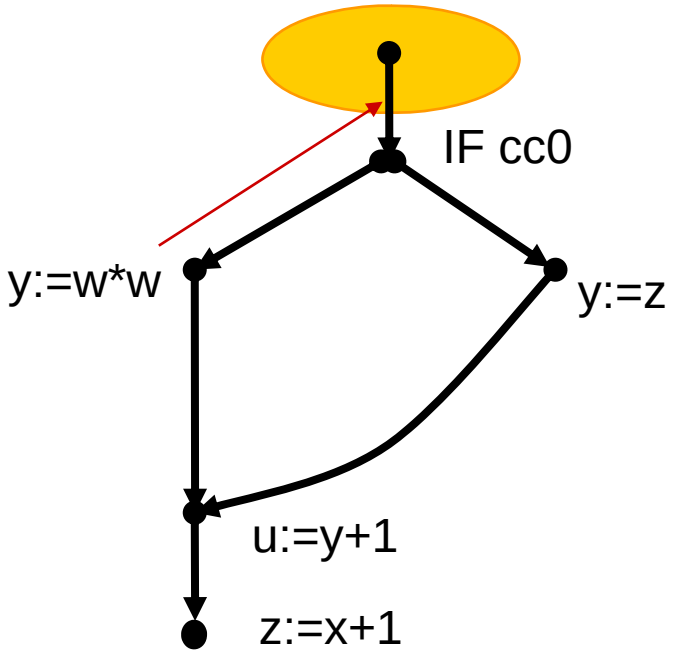
Problem: create a parallel group at the root of a DAG

Achieved via code motion across BBs

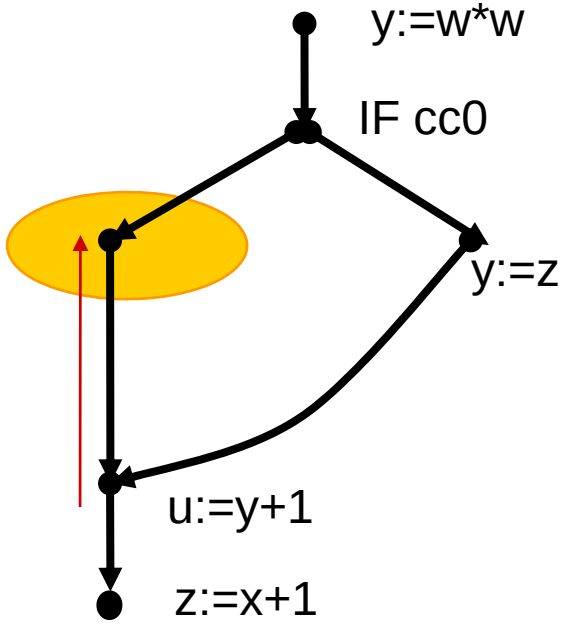
- Speculative code motion
- Join code motion
- Branch code motion
- Renaming & substitution
- Unification



Speculation, join code motion

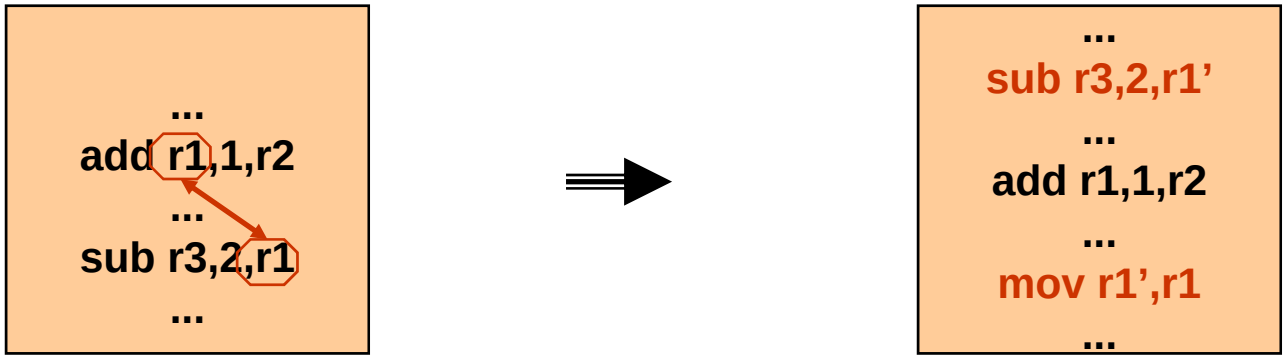


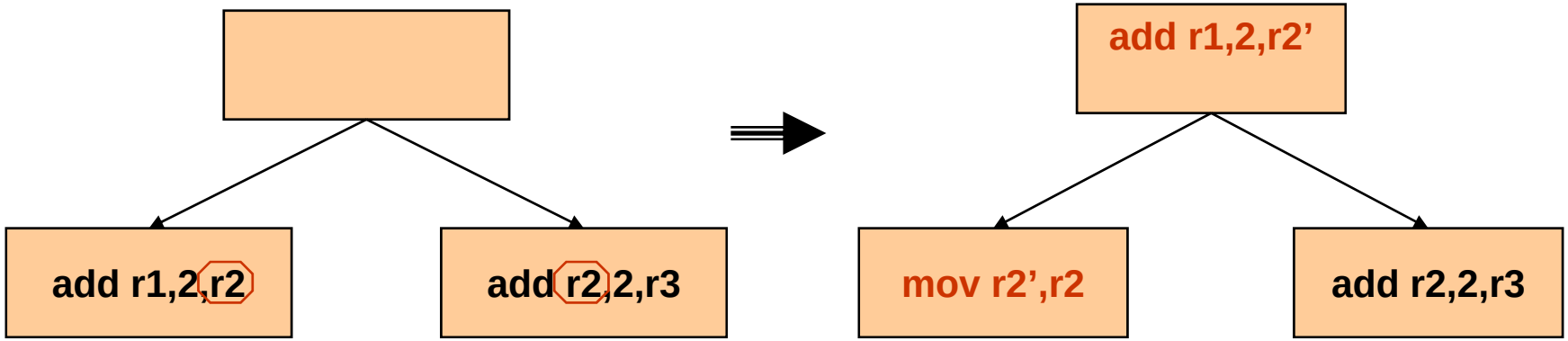
Speculation



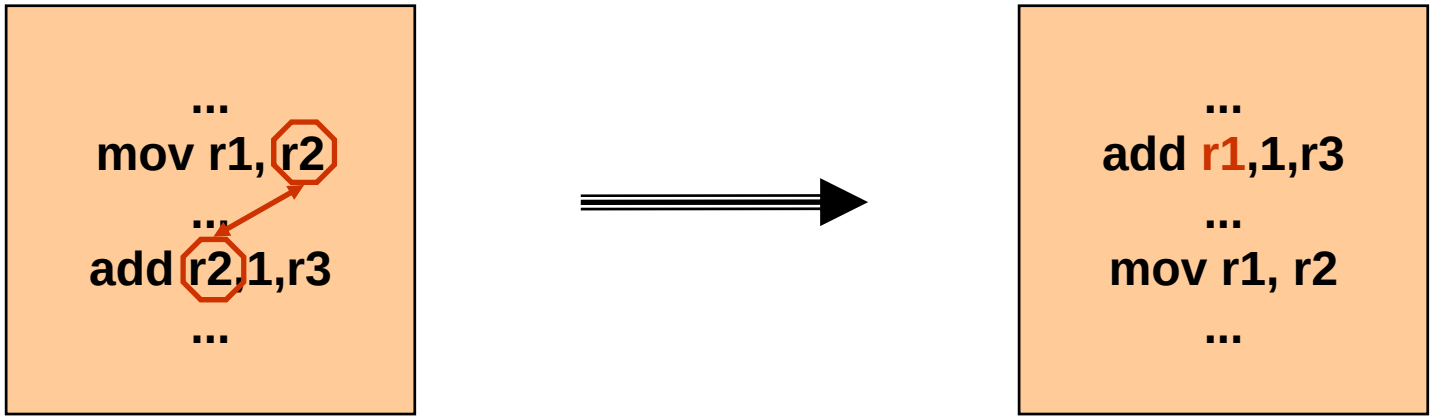
Join code motion

Renaming


...
add r1,1,r2
...
sub r3,2,r1
...
...
sub r3,2,r1'
...
add r1,1,r2
...
mov r1',r1
...

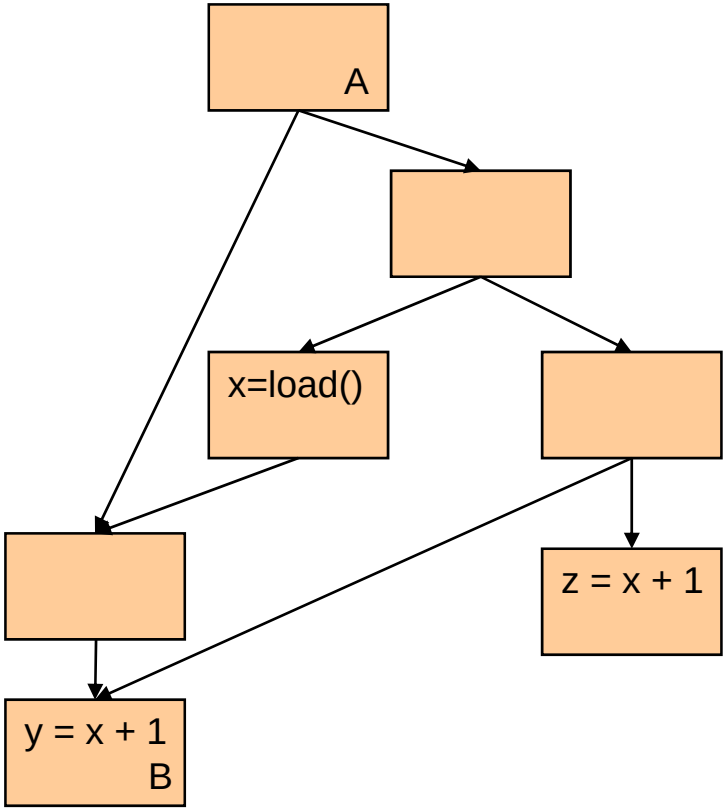

add r1,2,r2
add r2,2,r3
add r1,2,r2'
mov r2',r2
add r2,2,r3

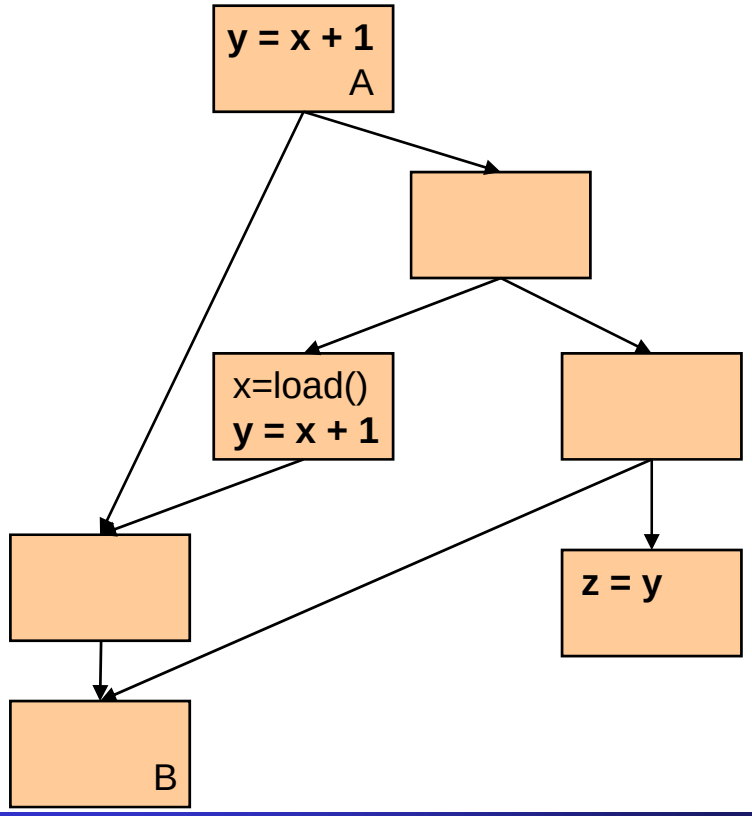
Forward-substitution



Unification

- Simplest form: moving an instr. below a hammock to the above of the hammock
- Selective scheduling can do more sophisticated form of unification





Software Pipelining

Schedule instructions beyond loop iteration boundaries

- Iterations are overlapped in a pipelined fashion
  - prolog, kernel, and epilog
- More efficient than unrolling-followed-by-DAG scheduling

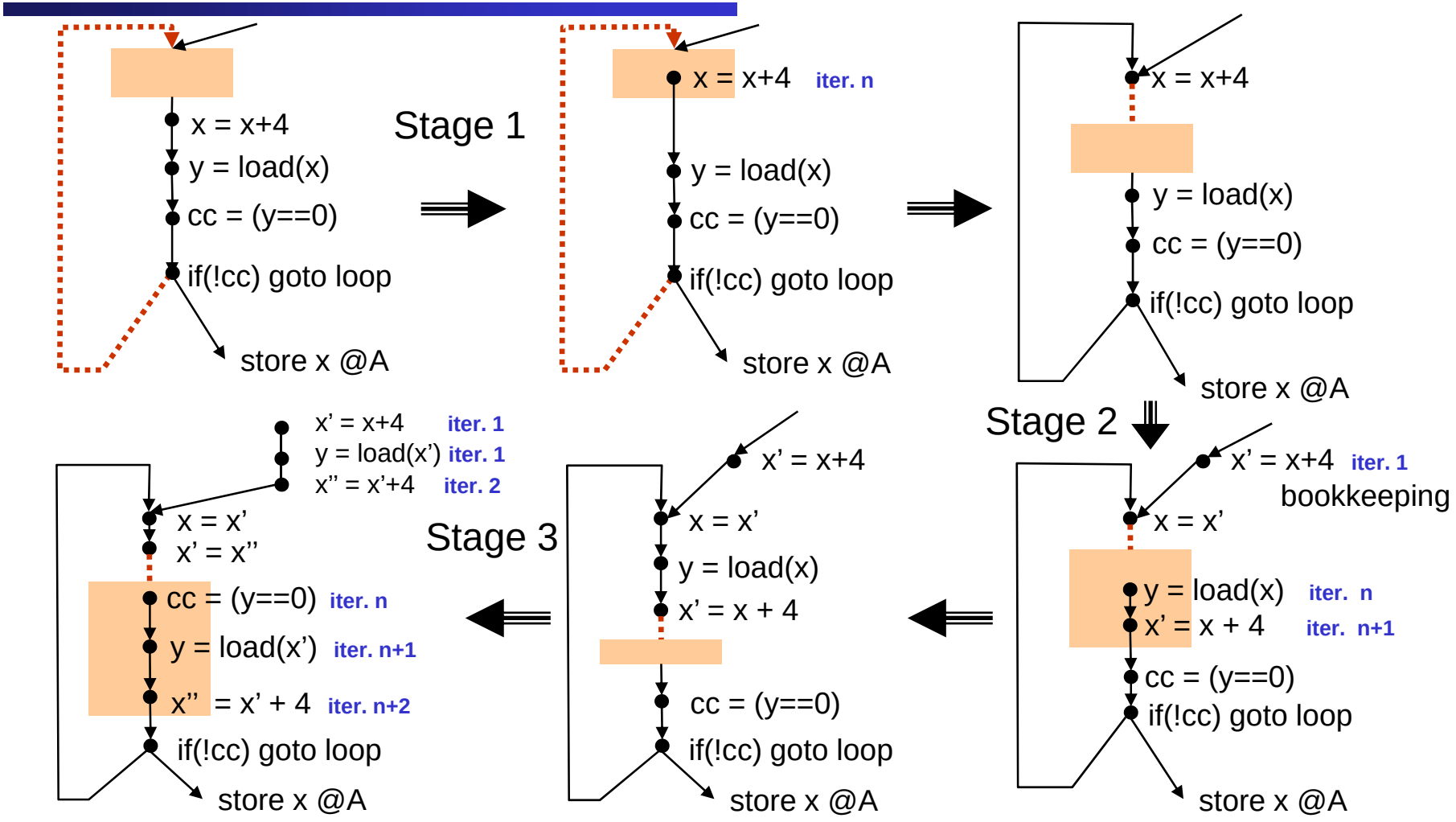
Modulo scheduling is the most popular technique, but there is yet another practical technique called

- Enhanced pipeline scheduling (EPS)

Enhanced Pipeline Scheduling (EPS)

- A software pipelining technique based on global DAG scheduling, which is very different from MS
  - MS destroys the original loop and creates a new loop
- For a given loop, we just repeat DAG scheduling.
- When instructions are moved across the loop back-edge, the "pipelining effect" takes place. We call it cross-iteration code motion (CICM).
- So, EPS simply defines DAGs in the loop body by cutting edges, which are then scheduled globally.

A generic EPS example



Advantages of EPS

- We can schedule "ANY" loops
  - Loops with arbitrary control flows
  - Loops whose trip counts are not constants
    - e.g., pointer-chasing loops
  - Outer loops
  - Unstructured loops

due to its code-motion-based pipelining

- Can achieve tight, variable II for multi-path loops

→ Particularly useful for optimizing integer code

Global DAG Scheduling

- We can use any global scheduling techniques for scheduling of DAGs in each stage of EPS, but
- we use selective scheduling (most aggressive)
  - All-path speculative code motion
  - Join code motion
  - Unification
  - Renaming
  - Forward substitution

Selective Scheduling

- All these techniques are well merged into a single, powerful global scheduling algorithm
  - Can extract more useful parallel instructions (even w/o profiling)
- When combined with EPS, it can maximize the scheduling power of EPS
- References
  - "Parallelizing non-numerical code with selective scheduling and software pipelining" ACM TOPLAS Nov. 1997
  - "Unroll-based copy elimination for EPS" IEEE TC Sep. 2002
  - "Split-Path EPS" IEEE TPDS May 2003