

EECS 583 – Automatic Parallelization Via Decoupled Software Pipelining

University of Michigan

April 10, 2023

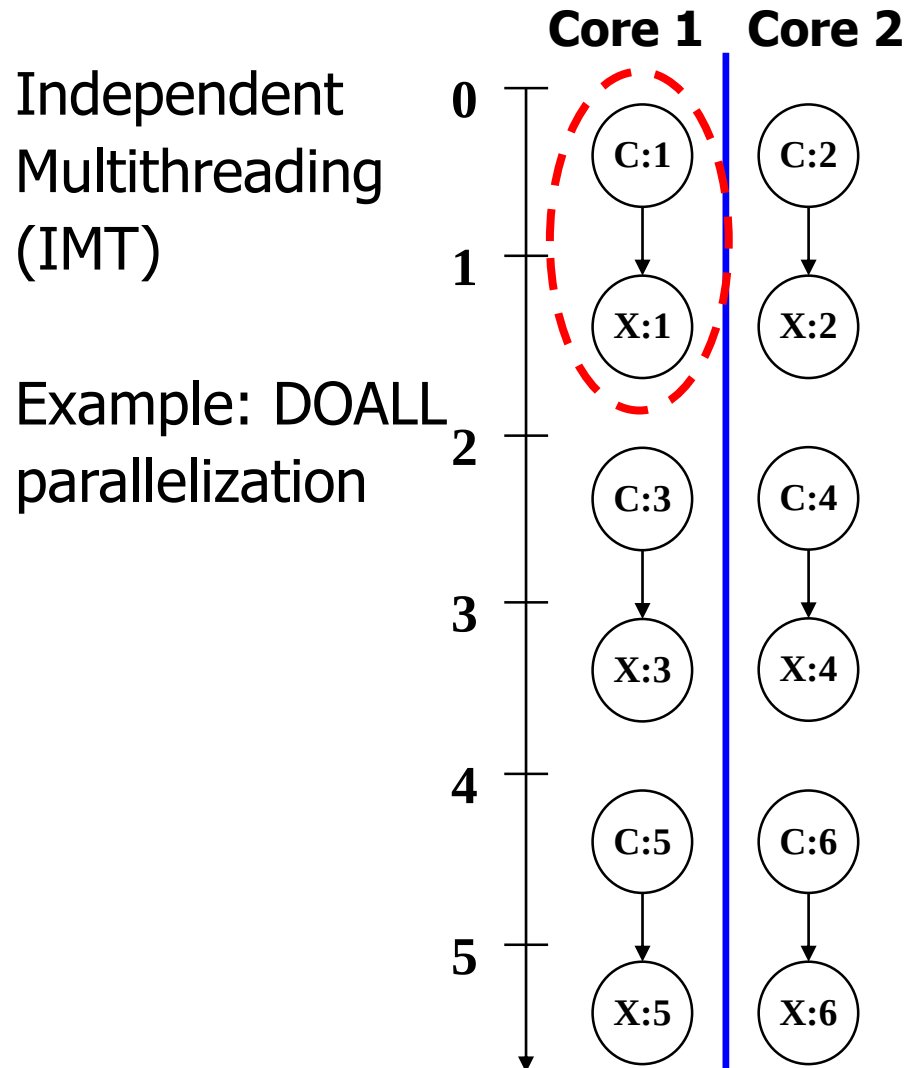
Parallelization: Scientific vs Non-Scientific Codes

Scientific Codes (FORTRAN-like)

```
for(i=1; i<=N; i++) // C
  a[i] = a[i] + 1;    // X
```

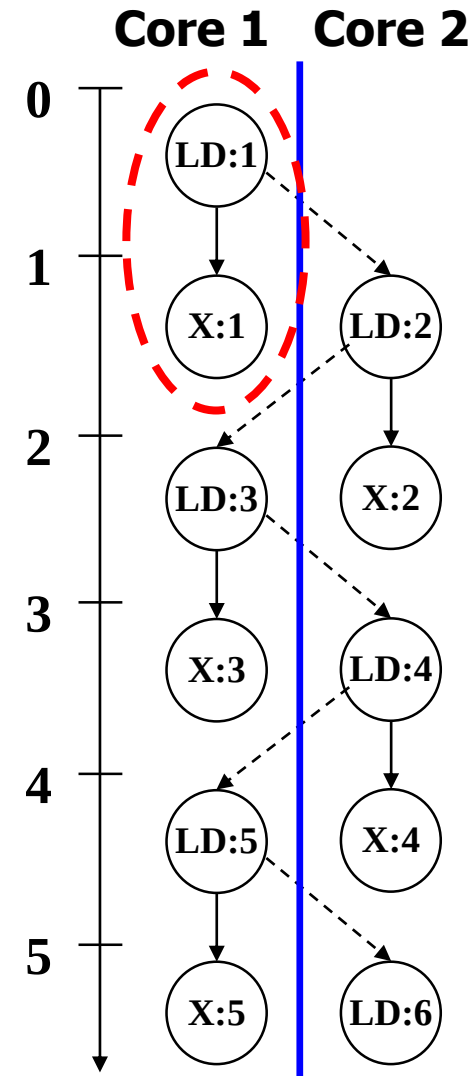
General-purpose Codes (legacy C/C++)

```
while(ptr = ptr->next) // LD
  ptr->val = ptr->val + 1; // X
```



Cyclic Multithreading (CMT)

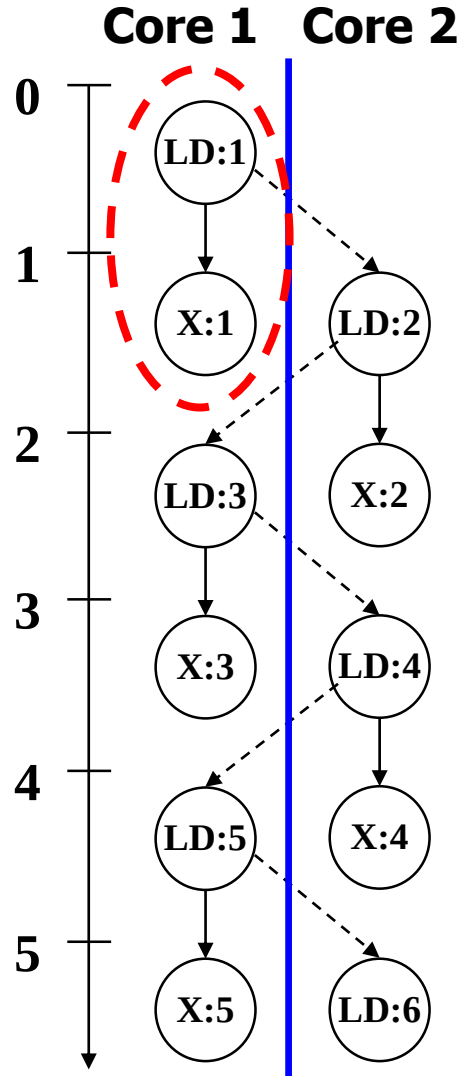
Example: DOACROSS [Cytron, ICPP 86]



Alternative Parallelization Approaches

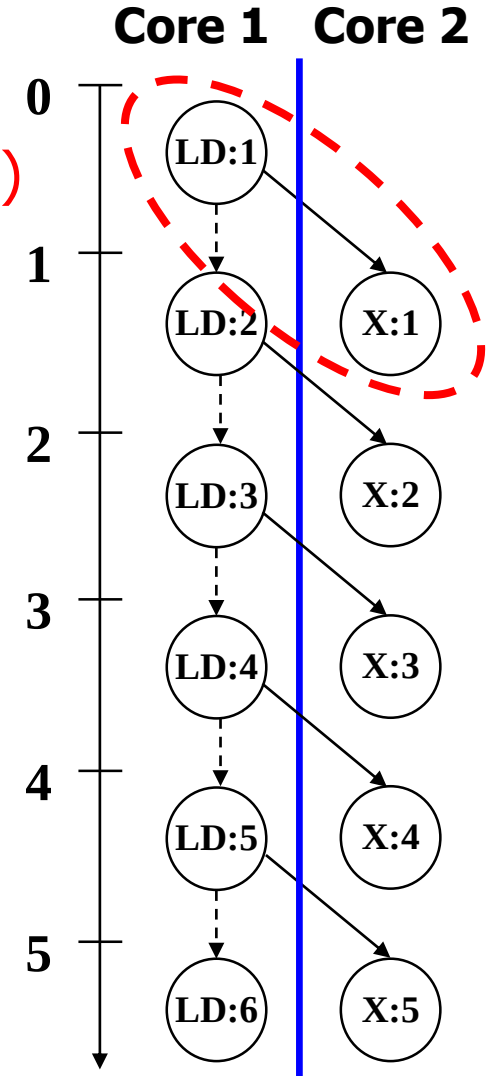
```
while(ptr = ptr->next)    // LD
    ptr->val = ptr->val + 1; // X
```

Cyclic
Multithreading
(CMT)



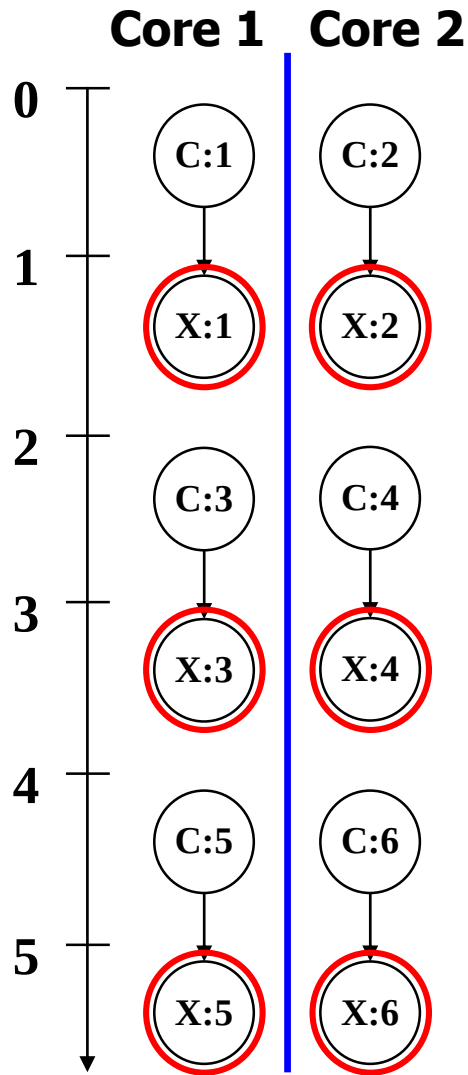
Pipelined
Multithreading (PMT)

Example: DSWP
[PACT 2004]



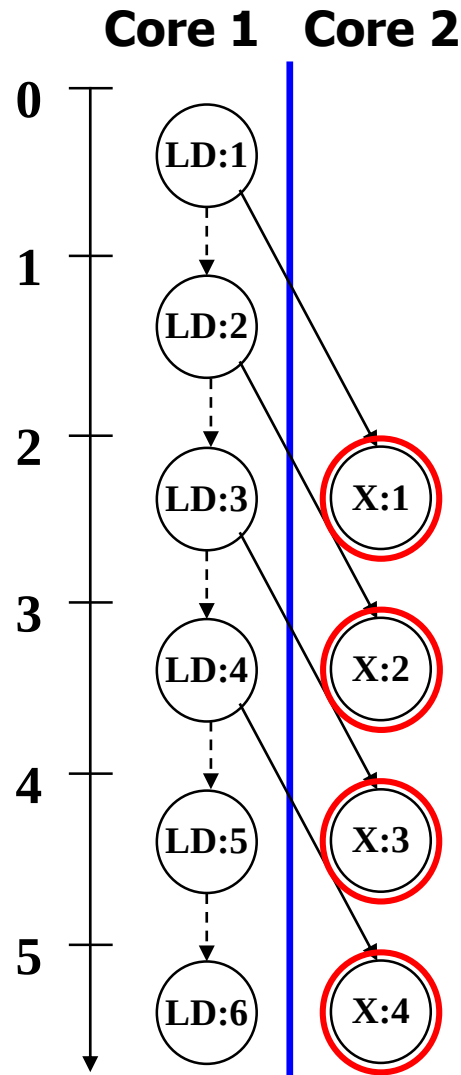
Comparison: IMT, PMT, CMT

IMT



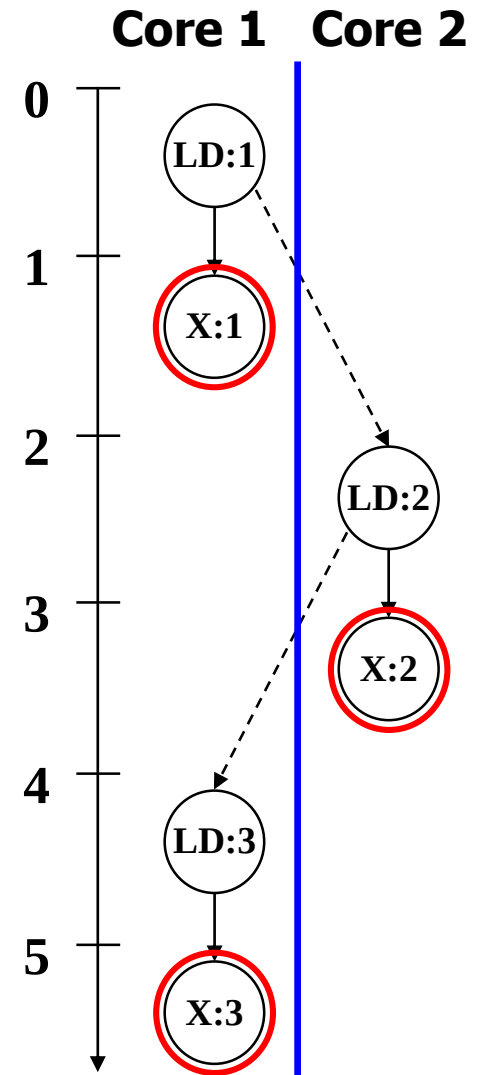
lat(comm) = 1: 1 iter/cycle
 lat(comm) = 2: 1 iter/cycle

PMT



1 iter/cycle
 1 iter/cycle

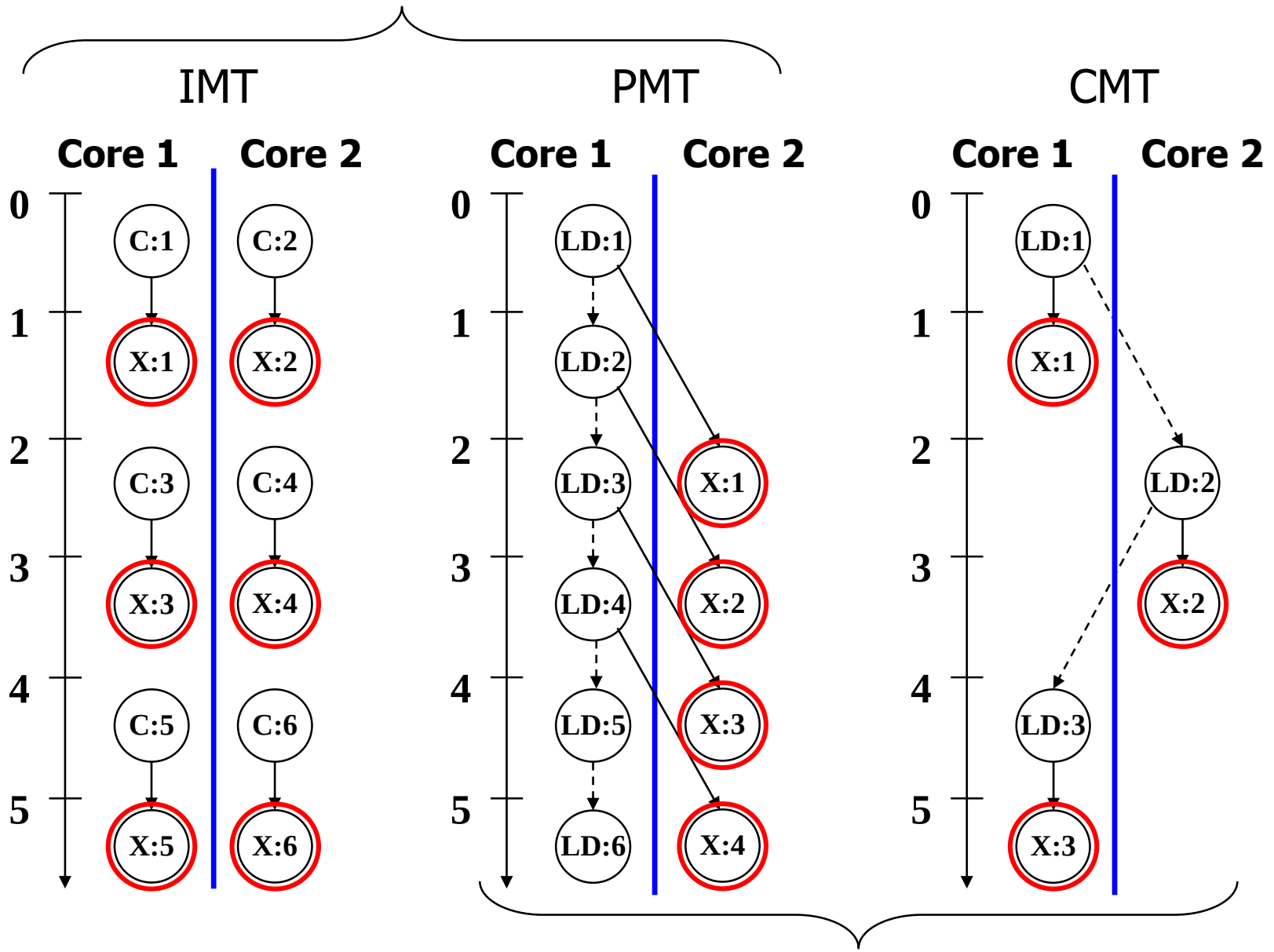
CMT



1 iter/cycle
 0.5 iter/cycle

Comparison: IMT, PMT, CMT

Thread-local Recurrences → Fast Execution



Cross-thread Dependences → Wide Applicability

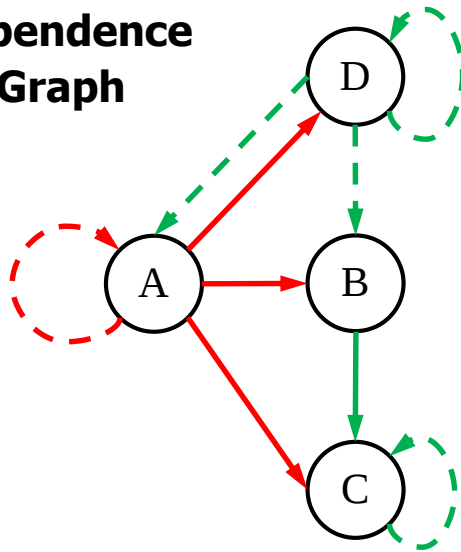
Decoupled Software Pipelining

Decoupled Software Pipelining (DSWP)

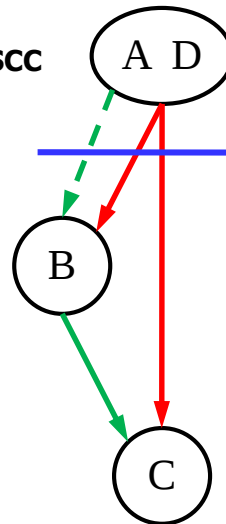
```

A: while (node)
B:   ncost = doit(node);
C:   cost += ncost;
D:   node = node->next;
  
```

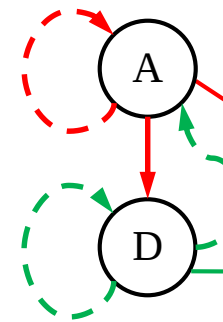
Dependence Graph



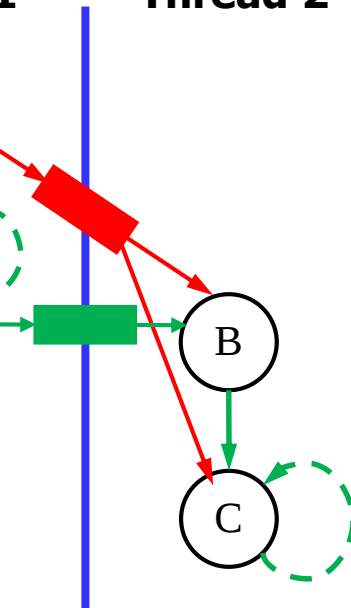
DAG_{scc}



Thread 1



Thread 2



register

control

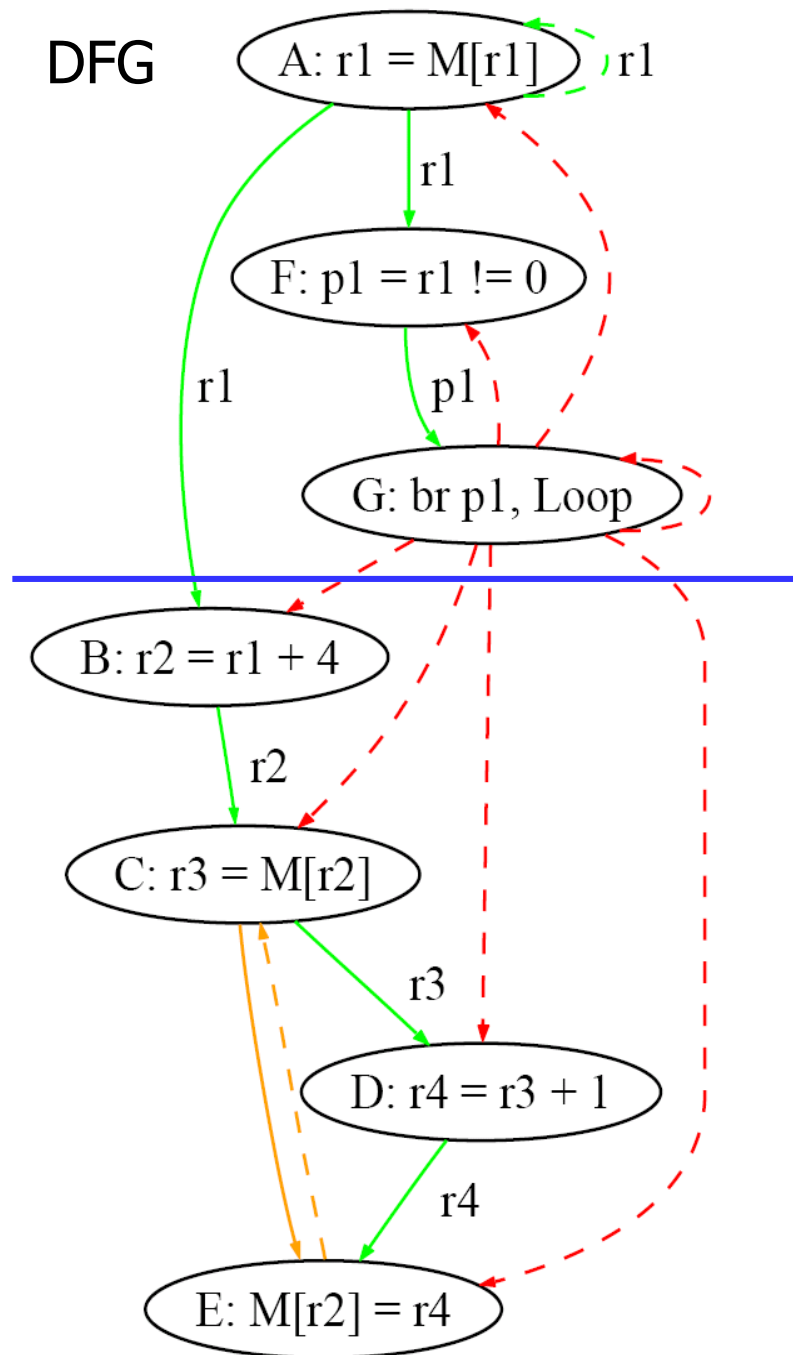
—→ intra-iteration

- - -→ loop-carried

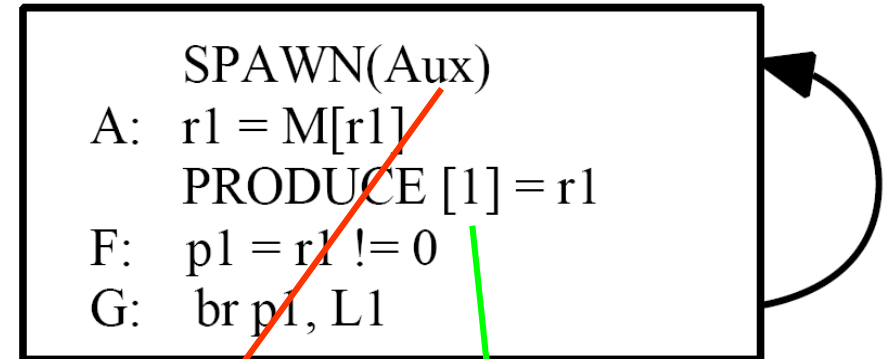
■ communication queue

**Inter-thread communication
latency is a one-time cost**

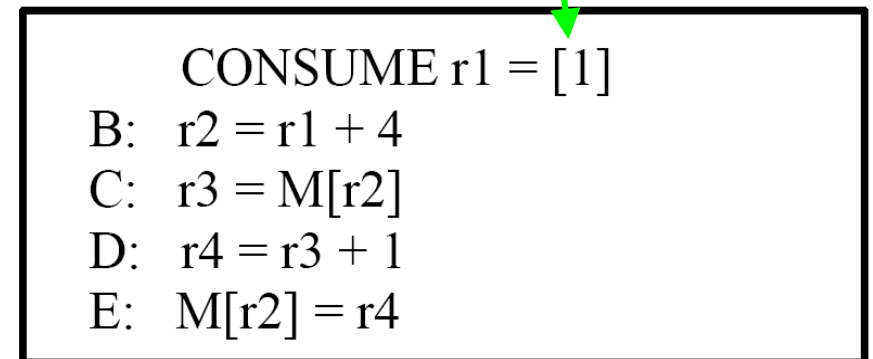
Implementing DSWP



L1:



Aux:



register

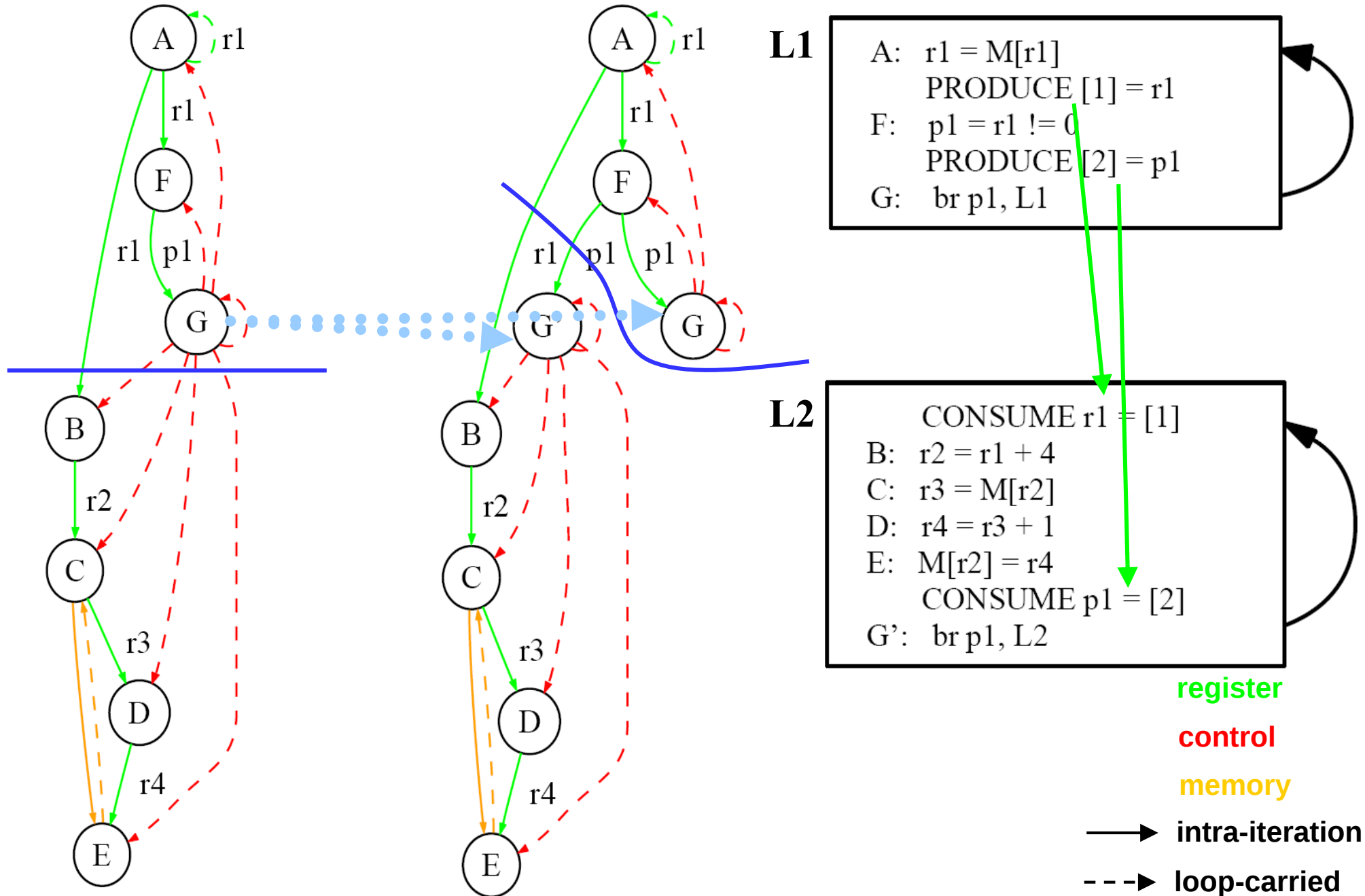
control

memory

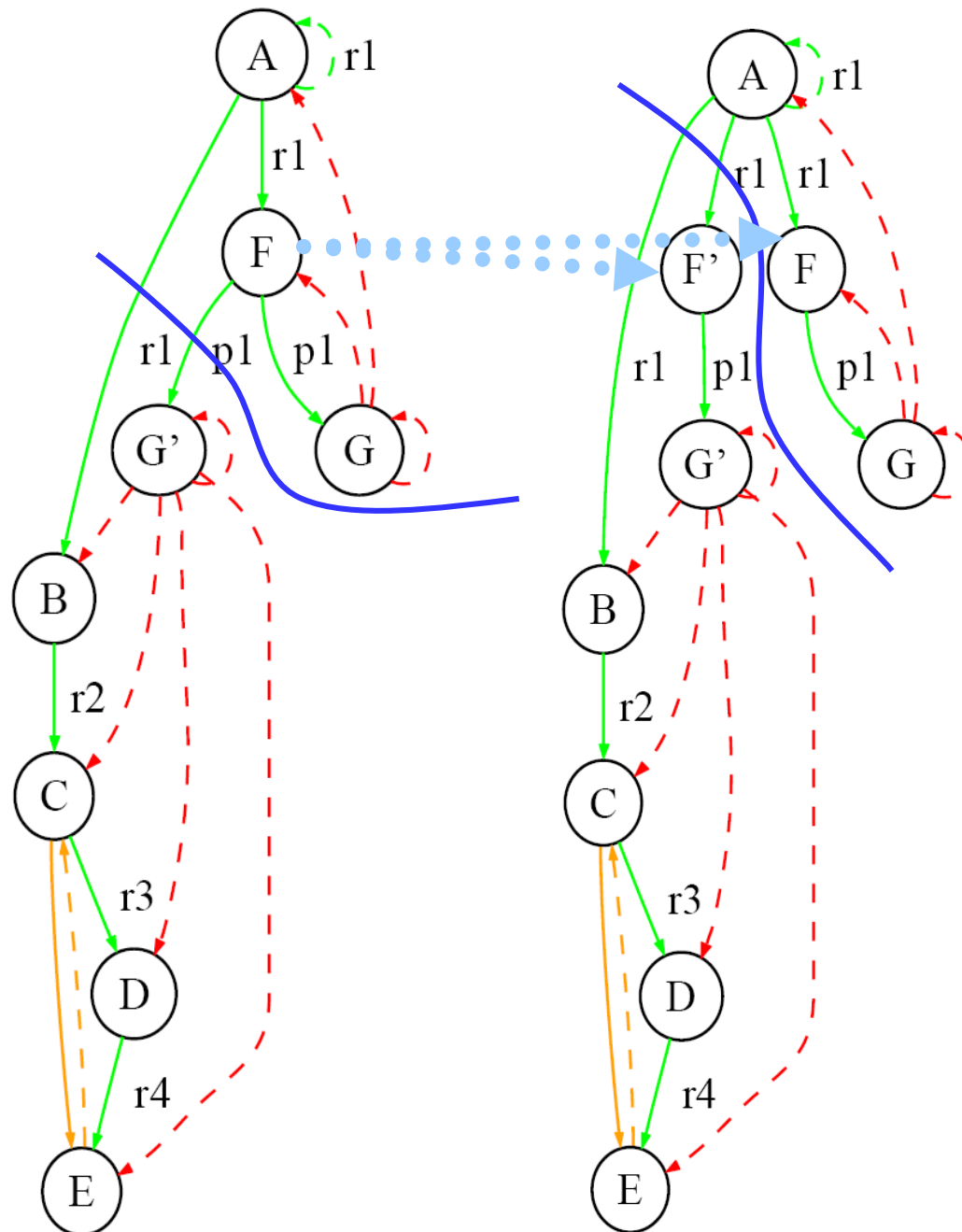
—→ intra-iteration

- - -→ loop-carried

Optimization: Node Splitting To Eliminate Cross Thread Control



Optimization: Node Splitting To Reduce Communication



L1

```
A: r1 = M[r1]
   PRODUCE [1] = r1
F:  p1 = r1 != 0
G:  br p1, L1
```

L2

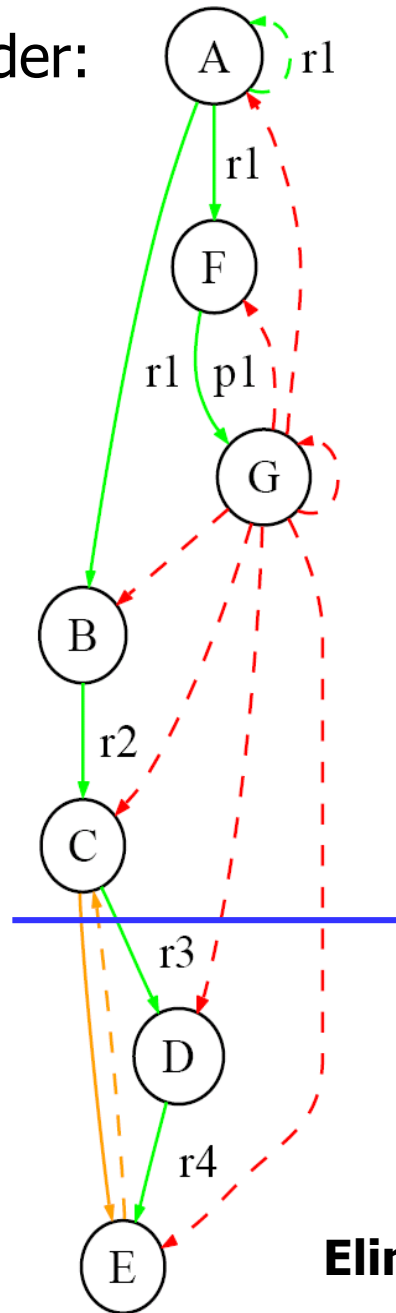
```
CONSUME r1 = [1]
B:  r2 = r1 + 4
C:  r3 = M[r2]
D:  r4 = r3 + 1
E:  M[r2] = r4
F': p1 = r1 != 0
G': br p1, L2
```

register
control
memory

—→ intra-iteration
- - -→ loop-carried

Constraint: Strongly Connected Components

Consider:



Eliminates pipelined/decoupled property

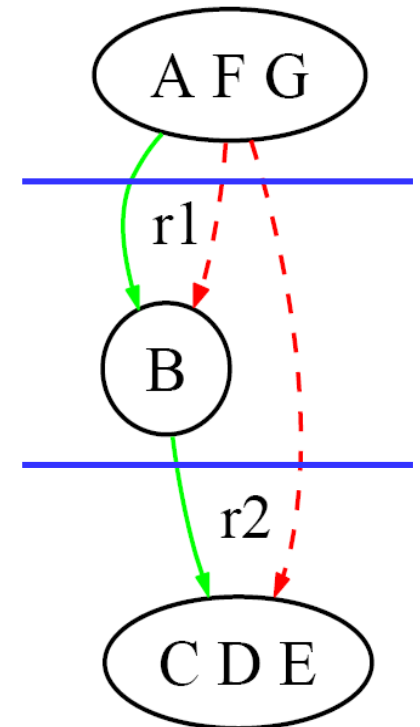
```

    SPAWN(Aux)
    A: r1 = M[r1]
    B: r2 = r1 + 4
    C: r3 = M[r2]
      PRODUCE [1] = r3
      CONSUME r0 = [2]
    F: p1 = r1 != 0
    G: br p1, L1
  
```

```

    CONSUME r3 = [1]
    D: r4 = r3 + 1
    E: M[r2] = r4
      PRODUCE [2] = r0
  
```

Solution: DAG_{SCC}



register

control

memory

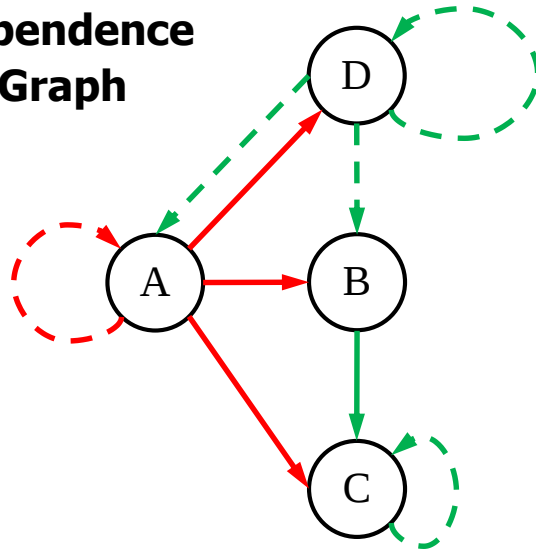
—→ intra-iteration

- - -→ loop-carried

Speculation – Break Statistically Unlikely Dependences

```
A: while (node)
B:   ncost = doit(node);
C:   cost += ncost;
D:   node = node->next;
```

Dependence Graph



register

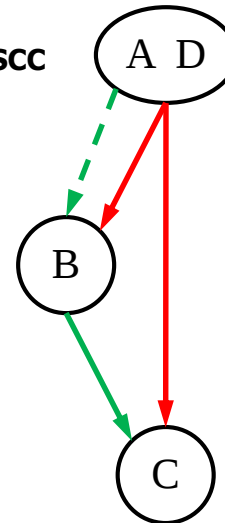
control

—→ intra-iteration

- - → loop-carried

■ communication queue

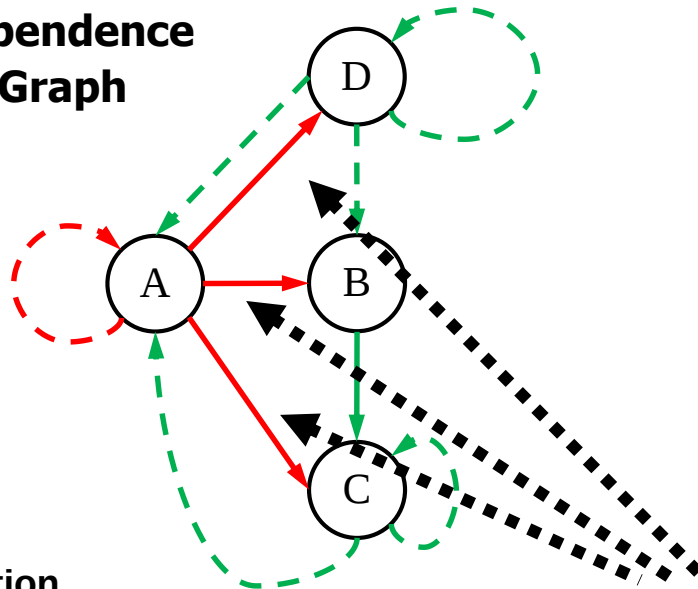
DAG_{scc}



Why Speculation?

```
A: while(cost < T && node)
B:   ncost = doit(node);
C:   cost += ncost;
D:   node = node->next;
```

Dependence Graph

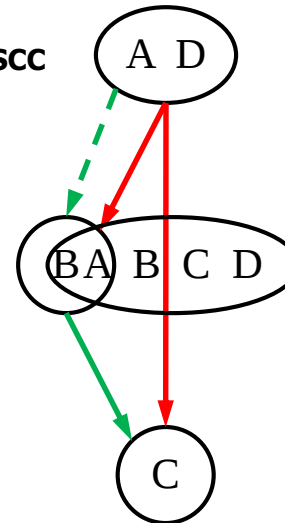


register
control

—→ intra-iteration
- - → loop-carried
■ communication queue

**Predictable
Dependencies**

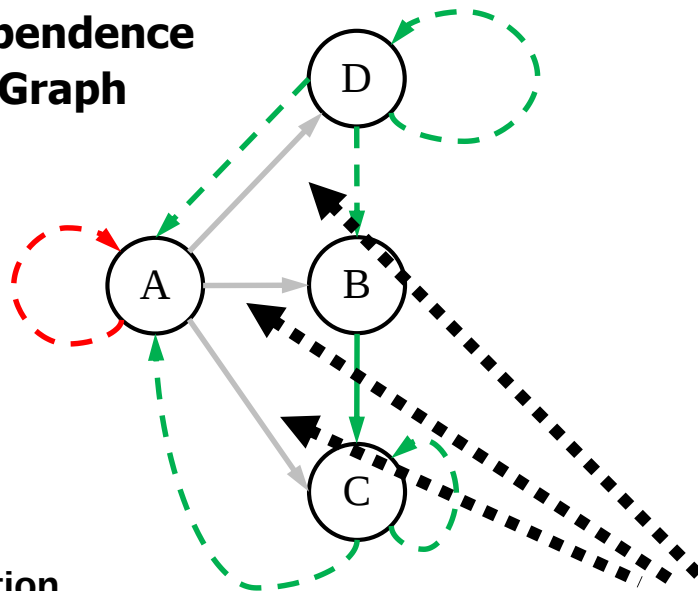
DAG_{scc}



Why Speculation?

```
A: while(cost < T && node)
B:   ncost = doit(node);
C:   cost += ncost;
D:   node = node->next;
```

Dependence Graph



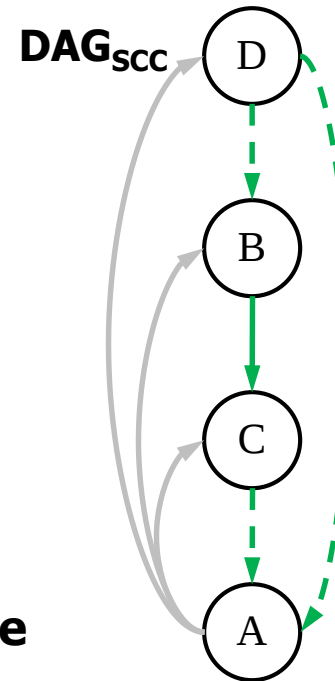
register
control

intra-iteration

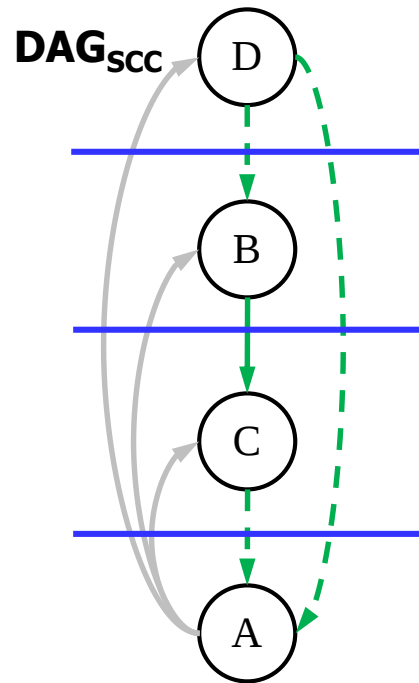
loop-carried

communication queue

**Predictable
Dependencies**



Execution Paradigm



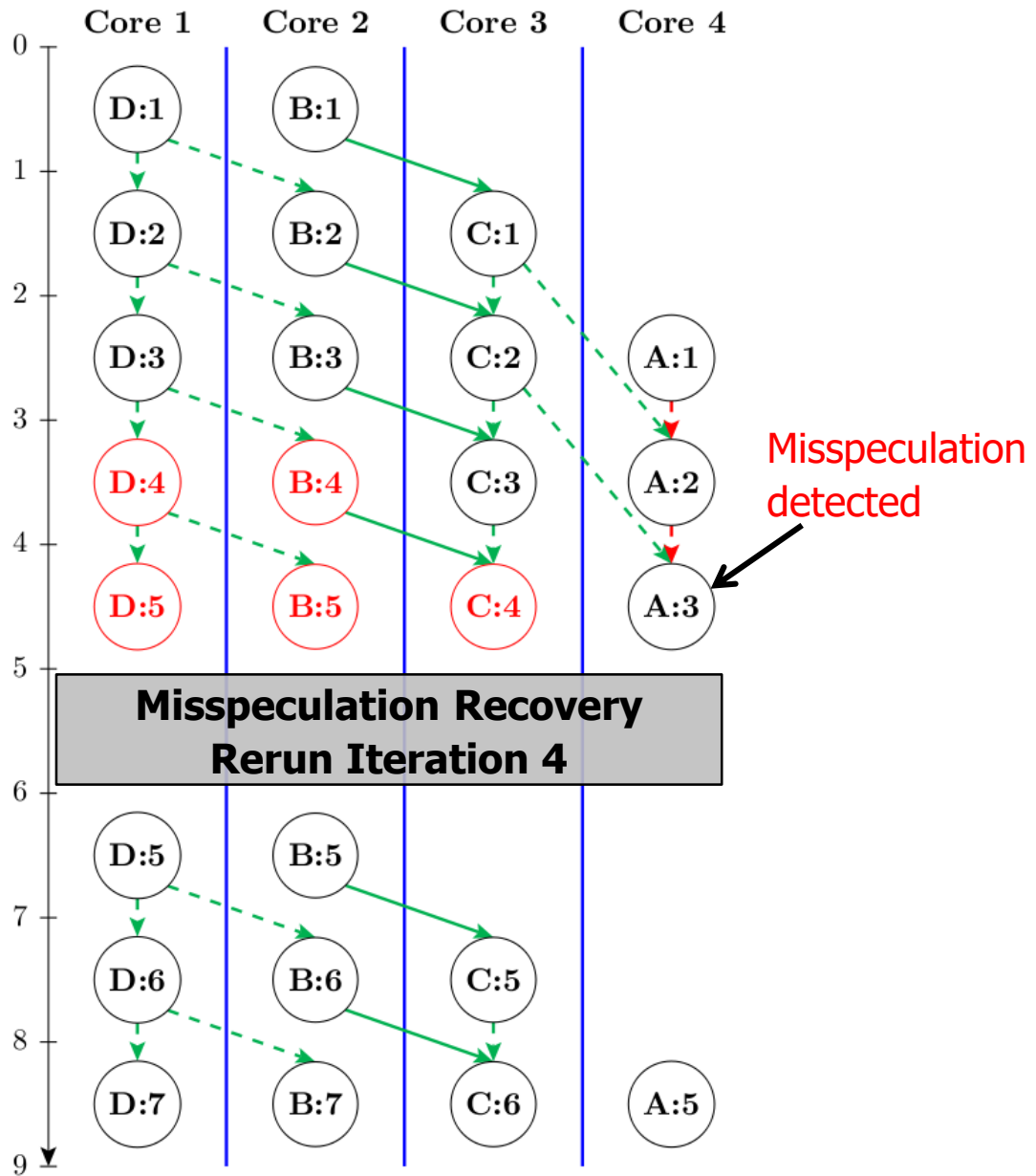
register

control

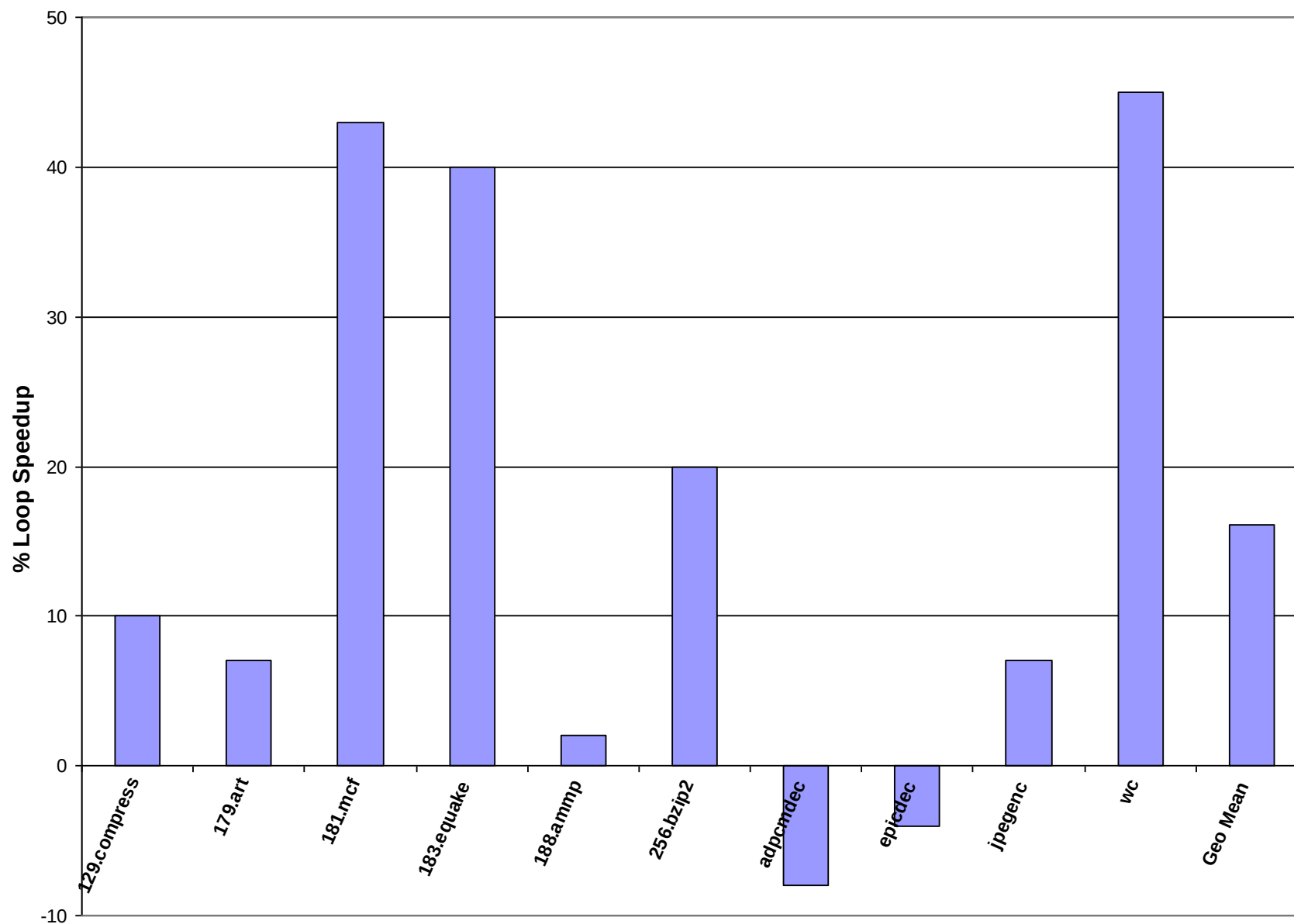
→ intra-iteration

- - -> loop-carried

■ communication queue



Evaluation: Dual Core vs Single Core



References

- ❖ “Automatic Thread Extraction with Decoupled Software Pipelining,” G. Ottoni, R. Rangan, A. Stoler, and D. I. August, *Proceedings of the 38th IEEE/ACM International Symposium on Microarchitecture*, Nov. 2005
- ❖ “Revisiting the Sequential Programming Model for Multi-Core,” M. J. Bridges, N. Vachharajani, Y. Zhang, T. Jablin, and D. I. August, Proc 40th IEEE/ACM International Symposium on Microarchitecture, December 2007.