Beyond Superscalar

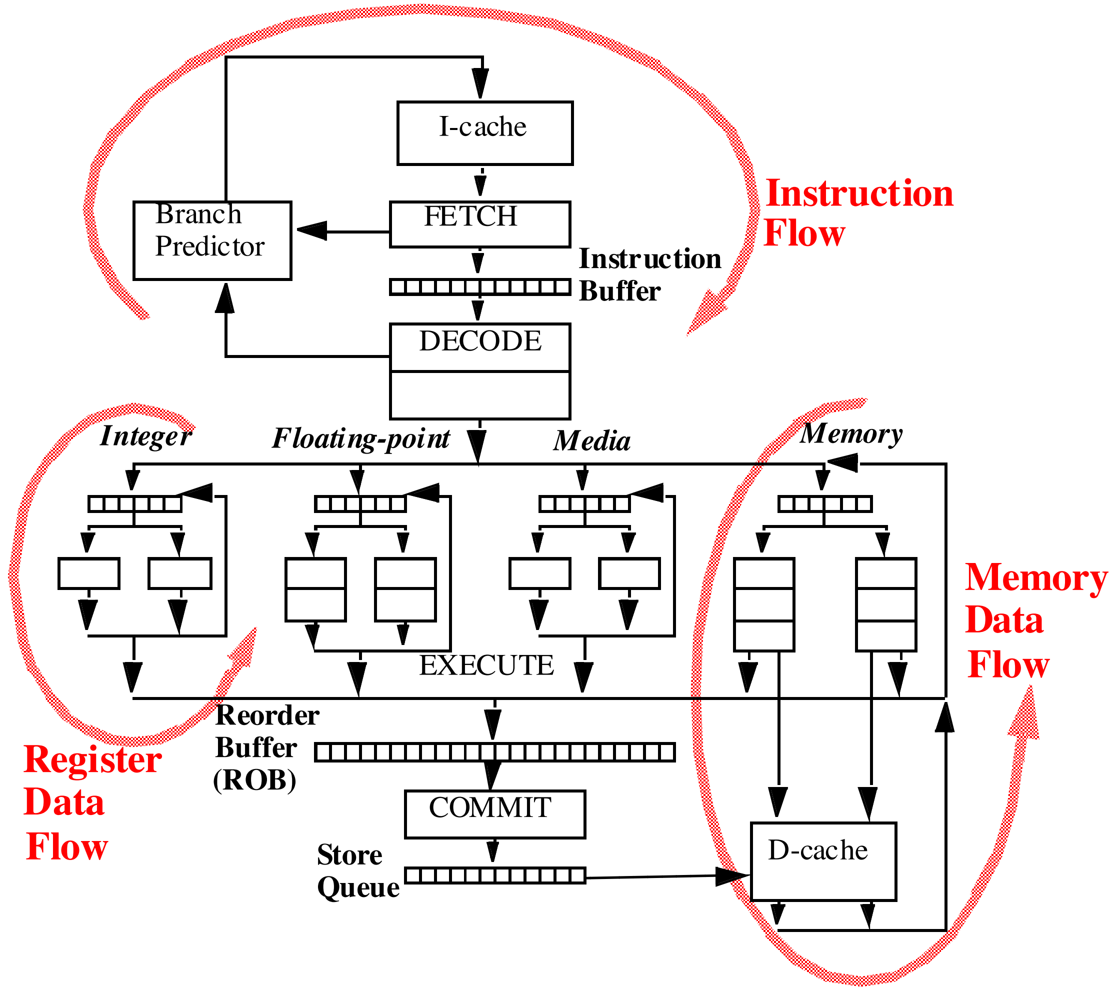
Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design

Lecture 8 (05 Feb 2015)

CADSL

Impediments to High IPC



Beyond Data Dependency Limits

Advanced Register Data Flow Techniques

Value Prediction

- What is value prediction? Broadly, three salient attributes:
  1. Generate a speculative value (predict)
  2. Consume speculative value (execute)
  3. Verify speculative value (compare/recover)
- This subsumes branch prediction

Focus here on operand values

Some History

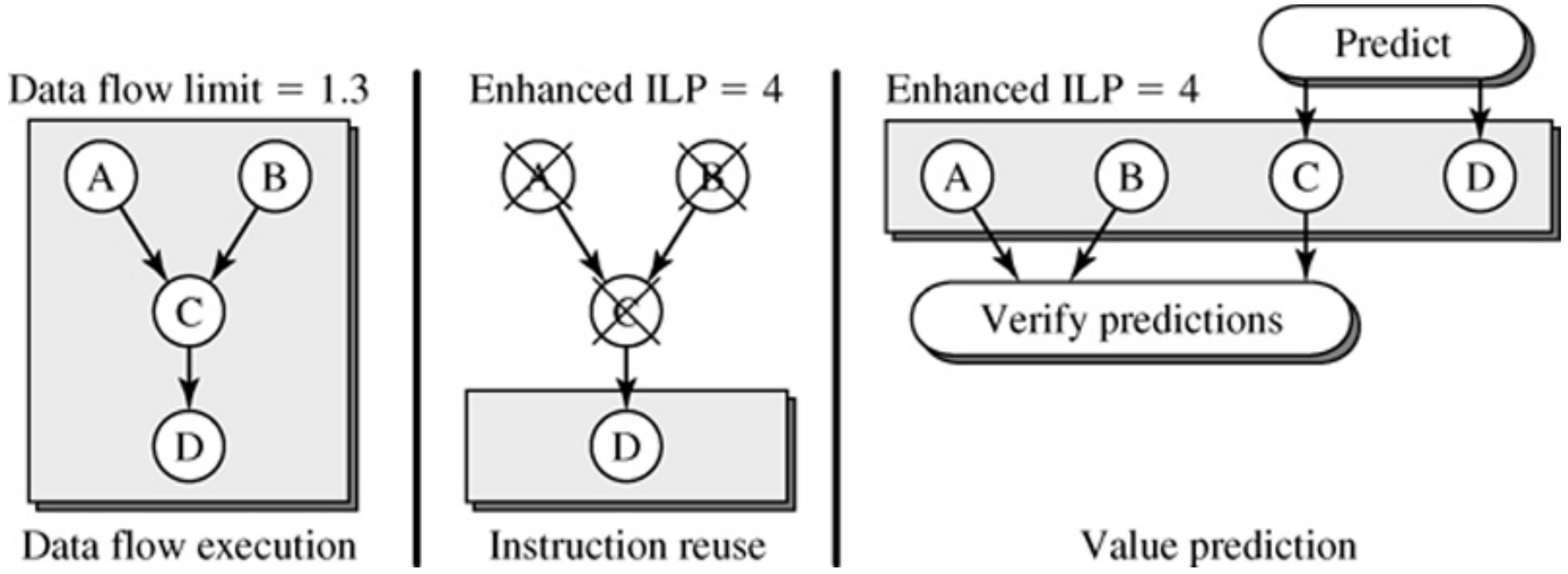
- "Classical" value prediction
  - Independently invented by 4 groups in 1995-1996

1. AMD (Nexgen): L. Widigen and E. Sowadsky, patent filed March 1996, inv. March 1995
2. Technion: F. Gabbay and A. Mendelson, inv. sometime 1995, TR 11/96, US patent Sep 1997
3. CMU: M. Lipasti, C. Wilkerson, J. Shen, inv. Oct. 1995, ASPLOS paper submitted March 1996
4. Wisconsin: Y. Sazeides, J. Smith, Summer 1996

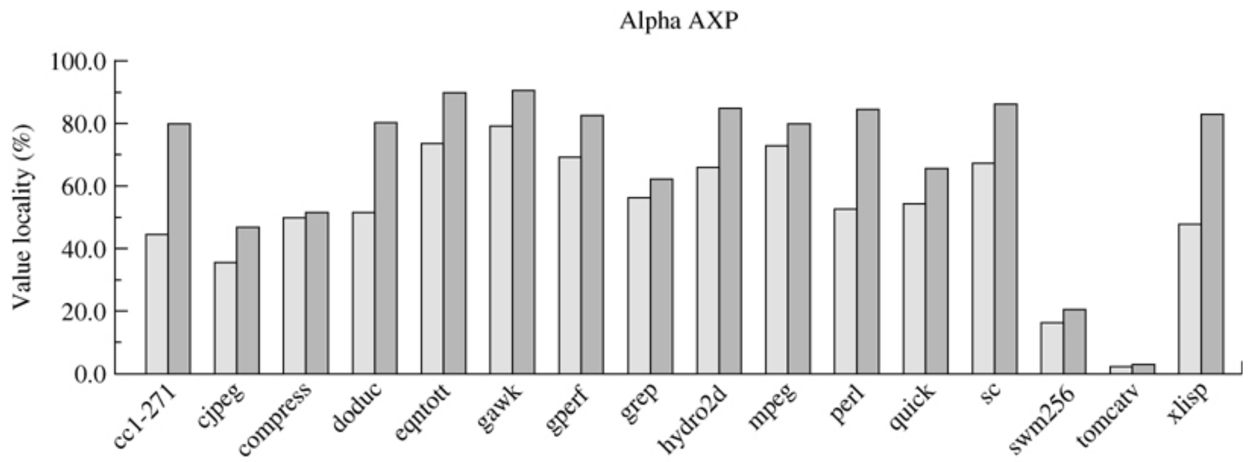
Why Value Prediction?

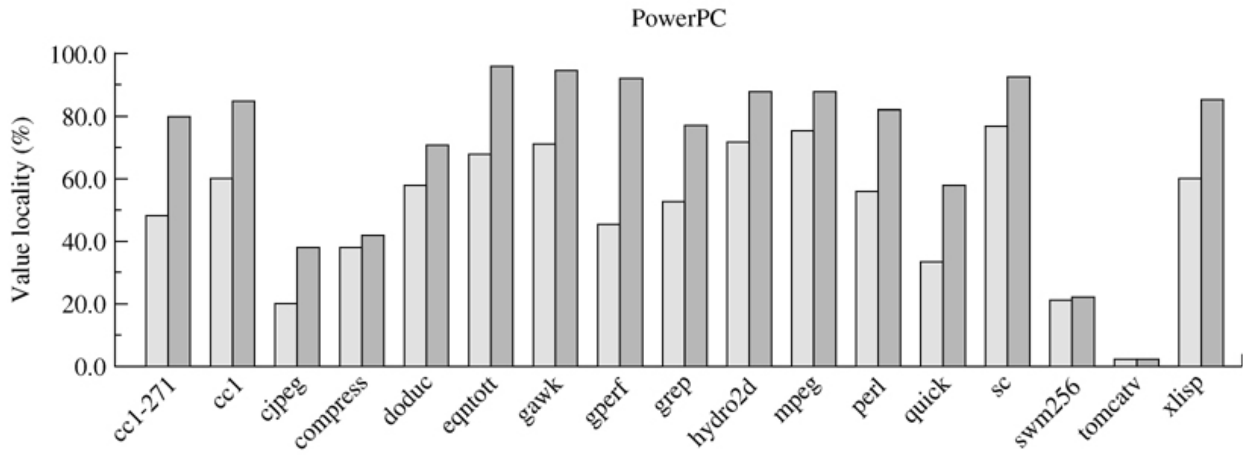
- Possible explanations:
  1. Natural evolution from branch prediction
  2. Natural evolution from memoization
  3. Natural evolution from rampant speculation
     - Cache hit speculation
     - Memory independence speculation
     - Speculative address generation
  4. Improvements in tracing/simulation technology
     - "There's a lot of zeroes out there." (C. Wilkerson)
     - Values, not just instructions & addresses
       - TRIP6000 [A. Martin-de-Nicolas, IBM]

Predict/Reuse



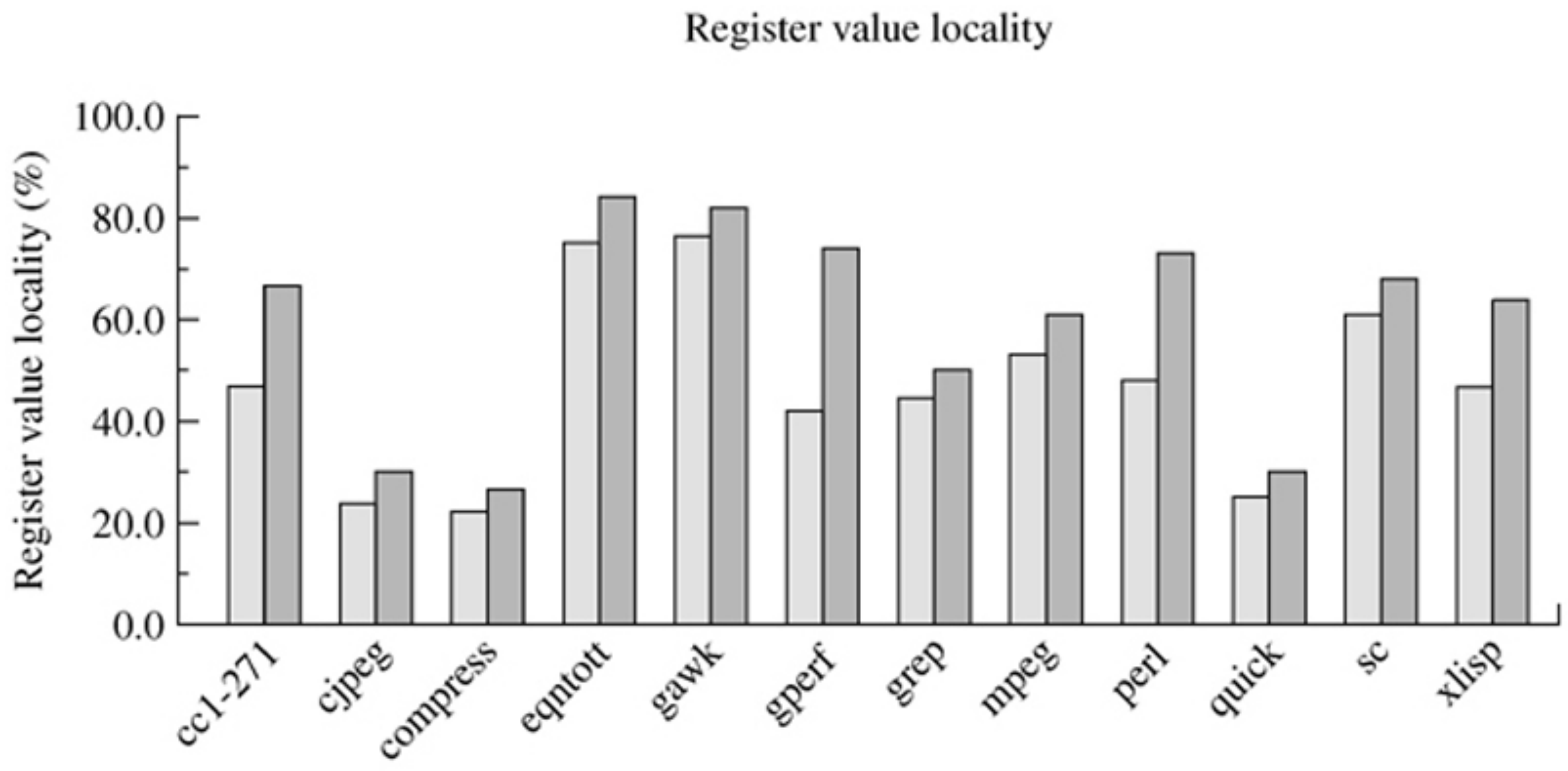
Value Reuse





The light bars show value locality for a history depth of one, while the dark bars show it for a history depth of sixteen.

Value Reuse



The light bars show value locality for a history depth of one, while the dark bars show it for a history depth of four.

What Happened?

- Tremendous academic interest
  - Dozens of research groups, papers, proposals
- No industry uptake

- Why?
  - Meager performance benefit (< 10%)
  - Power consumption
    - Dynamic power for extra activity
    - Static power (area) for prediction tables
  - Complexity and correctness
    - Subtle memory ordering issues [MICRO ' 01]
    - Misprediction recovery [HPCA ' 04]

Performance?

- Relationship between timely fetch and value prediction benefit [Gabbay, ISCA]

  Value prediction doesn't help when the result can be computed before the consumer instruction is fetched

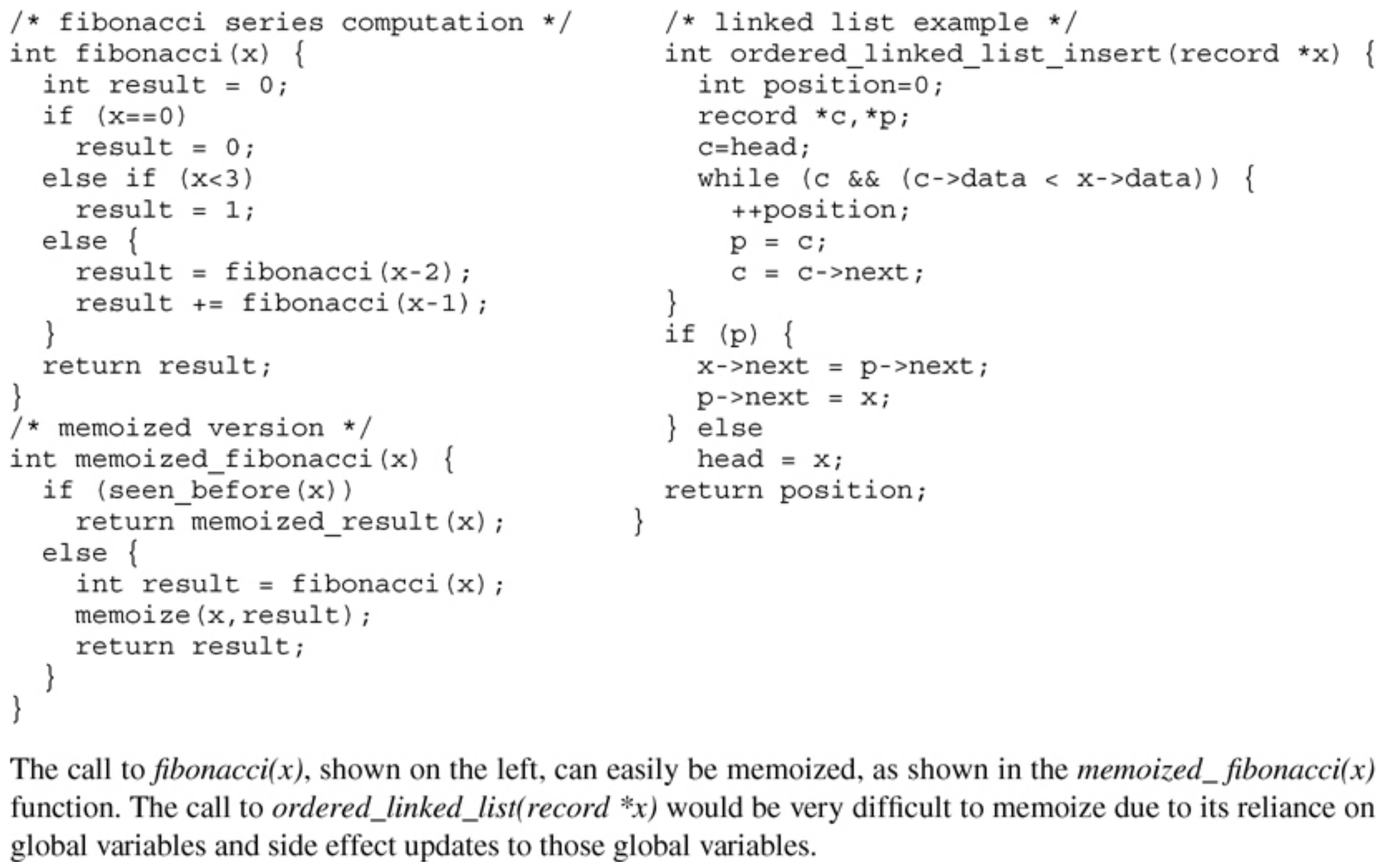
- High-bandwidth fetch helps
  - Wide trace caches studied in late 1990s

- More important to fetch the **right instructions**

Future Adoption?

- Classical value prediction will only make it in the context of a very different microarchitecture
  - One that explicitly and aggressively exposes ILP
- Promising trends
  - Deep pipelining craze is over
    - Can't manage the design complexity
  - High frequency mania is over
    - Can't afford the power
- Architects are pursuing ILP once again
  - Value prediction has another opportunity

Memoization

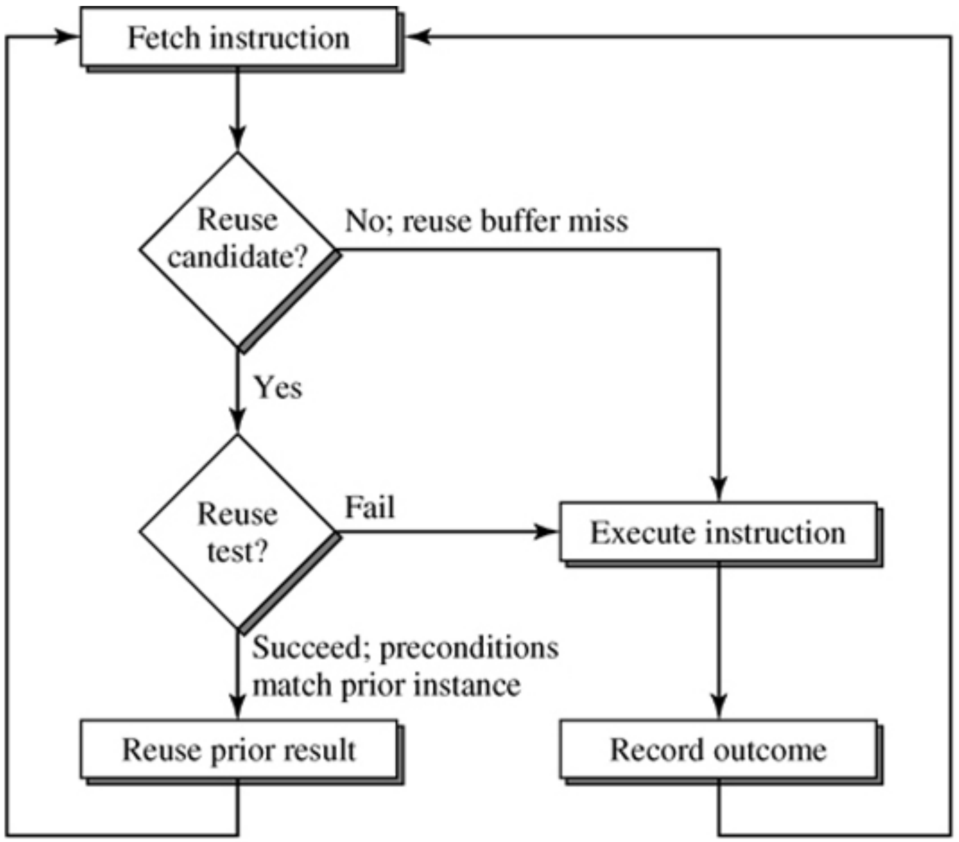


```
/* fibonacci series computation */
int fibonacci(x) {
  int result = 0;
  if (x==0)
    result = 0;
  else if (x<3)
    result = 1;
  else {
    result = fibonacci(x-2);
    result += fibonacci(x-1);
  }
  return result;
}
/* memoized version */
int memoized_fibonacci(x) {
  if (seen_before(x))
    return memoized_result(x);
  else {
    int result = fibonacci(x);
    memoize(x,result);
    return result;
  }
}
```

```
/* linked list example */
int ordered_linked_list_insert(record *x) {
  int position=0;
  record *c,*p;
  c=head;
  while (c && (c->data < x->data)) {
    ++position;
    p = c;
    c = c->next;
  }
  if (p) {
    x->next = p->next;
    p->next = x;
  } else
    head = x;
  return position;
}
```

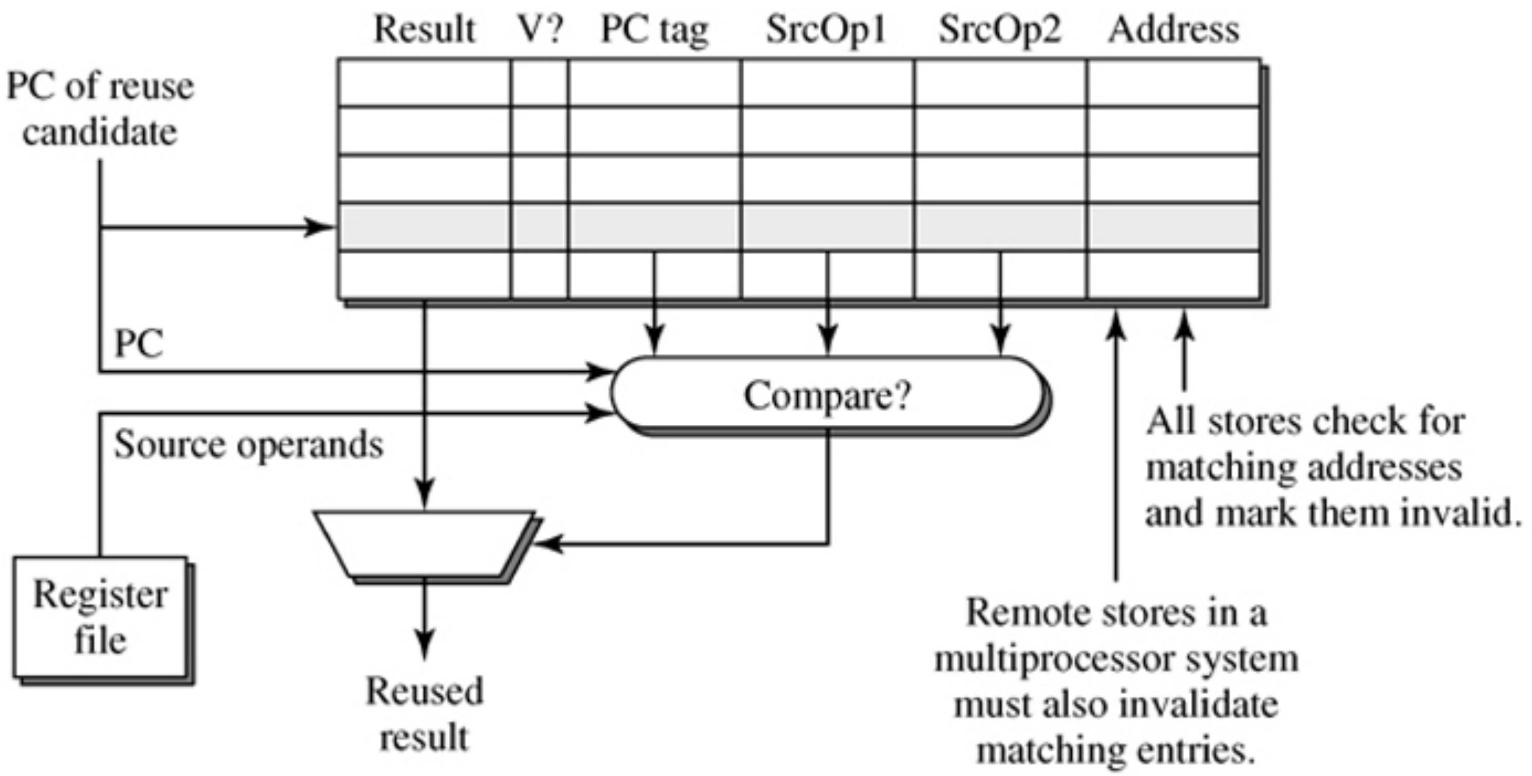
The call to *fibonacci(x)*, shown on the left, can easily be memoized, as shown in the *memoized_ fibonacci(x)* function. The call to *ordered_linked_list(record *x)* would be very difficult to memoize due to its reliance on global variables and side effect updates to those global variables.

Instruction Reuse



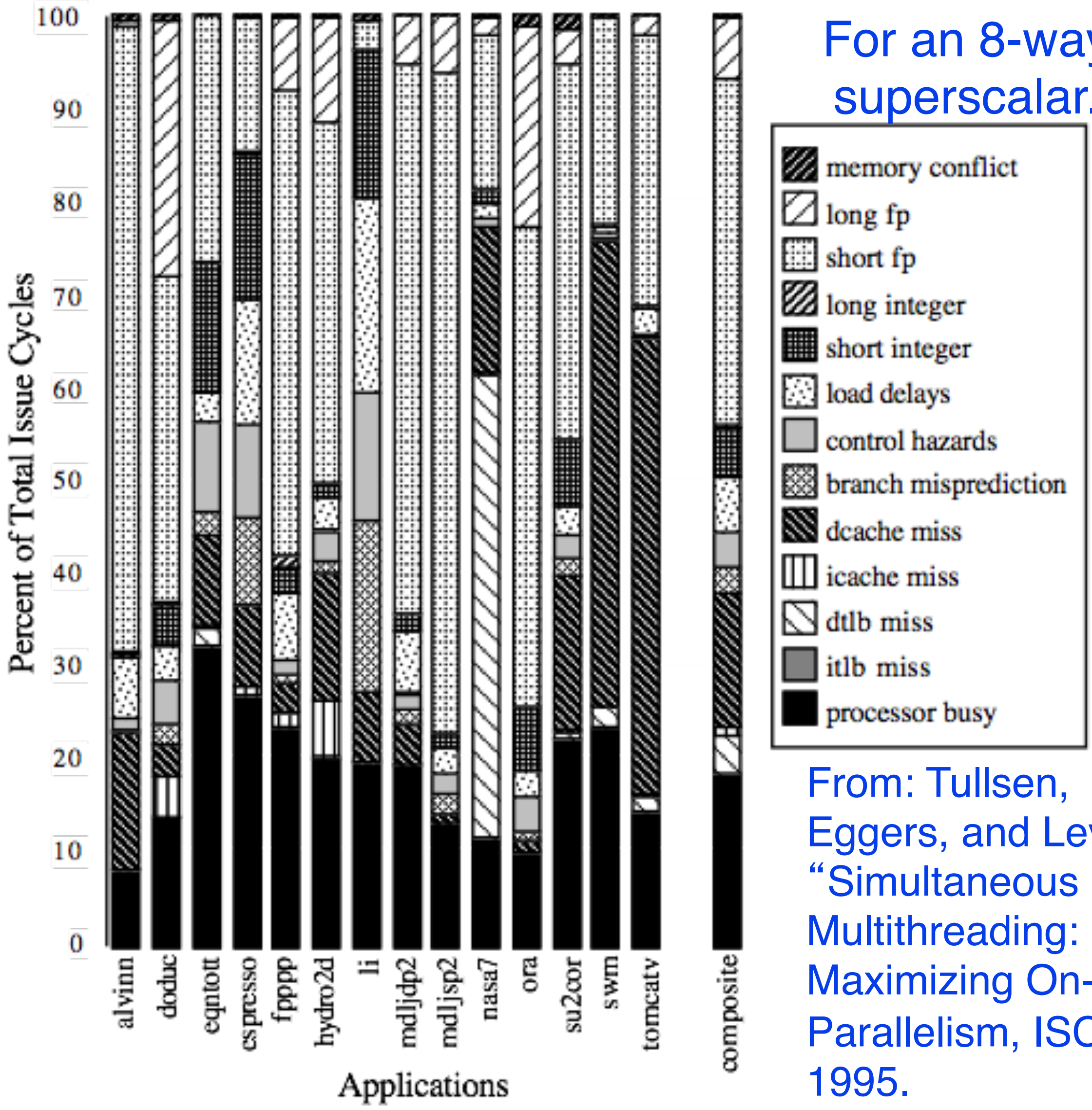
After an instruction is fetched, the history mechanism is checked to see whether the instruction is a candidate for reuse. If so, and if the instructions preconditions match the historical instance, the historical instance is reused and the fetched instruction is discarded. Otherwise, the instruction is executed as always, and its outcome is recorded in the history mechanism.

Instruction Reuse


Result
V?
PC tag
SrcOp1
SrcOp2
Address
PC of reuse candidate
PC
Source operands
Compare?
Register file
Reused result
All stores check for matching addresses and mark them invalid.
Remote stores in a multiprocessor system must also invalidate matching entries.

The instruction reuse buffer stores all the preconditions required to guarantee correct reuse of prior instances of instructions. For ALU and branch instructions, this includes a PC tag and source operand values. For loads and stores, the memory address must also be stored, so that intervening writes to that address will invalidate matching reuse entries.

For most apps, most execution units lie idle



For an 8-way superscalar.

From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

Superscalar Scenario

- Interest in multiple-issue because wanted to improve performance without affecting uniprocessor programming model
- Taking advantage of ILP is conceptually simple, but design problems are amazingly complex in practice
- Conservative in ideas, just faster clock and bigger
- Processors of last 10 years (Pentium 4, IBM Power 5, AMD Opteron) have the same basic structure and similar sustained issue rates (3 to 4 instructions per clock) as the 1st dynamically scheduled, multiple-issue processors announced in 1995
  - Clocks 10 to 20X faster, caches 4 to 8X bigger, 2 to 4X as many renaming registers, and 2X as many load-store units → performance 8 to 16X
- Peak v. delivered performance gap increasing

Thank You