Dataflow Computers

Motivation:

- exploit instruction-level parallelism on a massive scale
- more fully utilize all processing elements

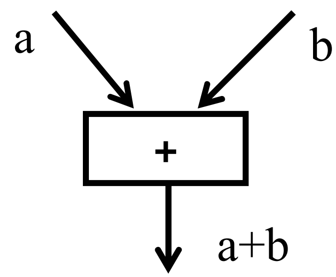
Believed this was possible if:

- express low-level parallelism in a functional-style programming language
  - no side effects, easy to reason about
- scheduled code greedily (i.e., massive out-of-order execution)
- hardware support for data-driven execution

Dataflow Computers

All computation is **data-driven**.

- binary as a directed graph
  - nodes are operations
  - values travel on arcs

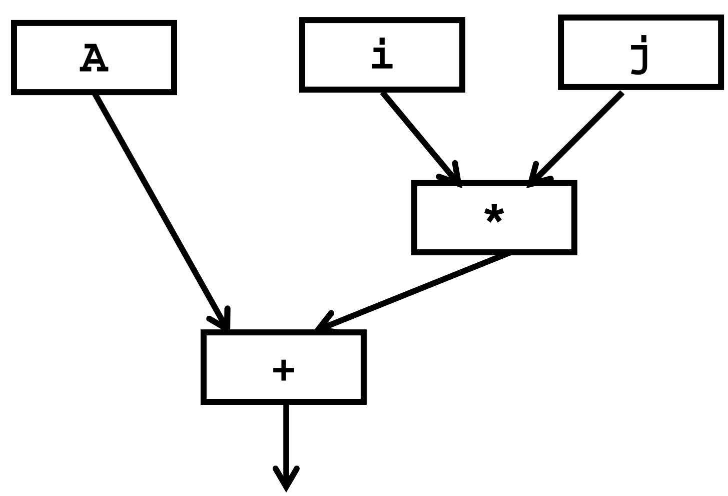


Dataflow Computers

Code & initial values loaded into memory

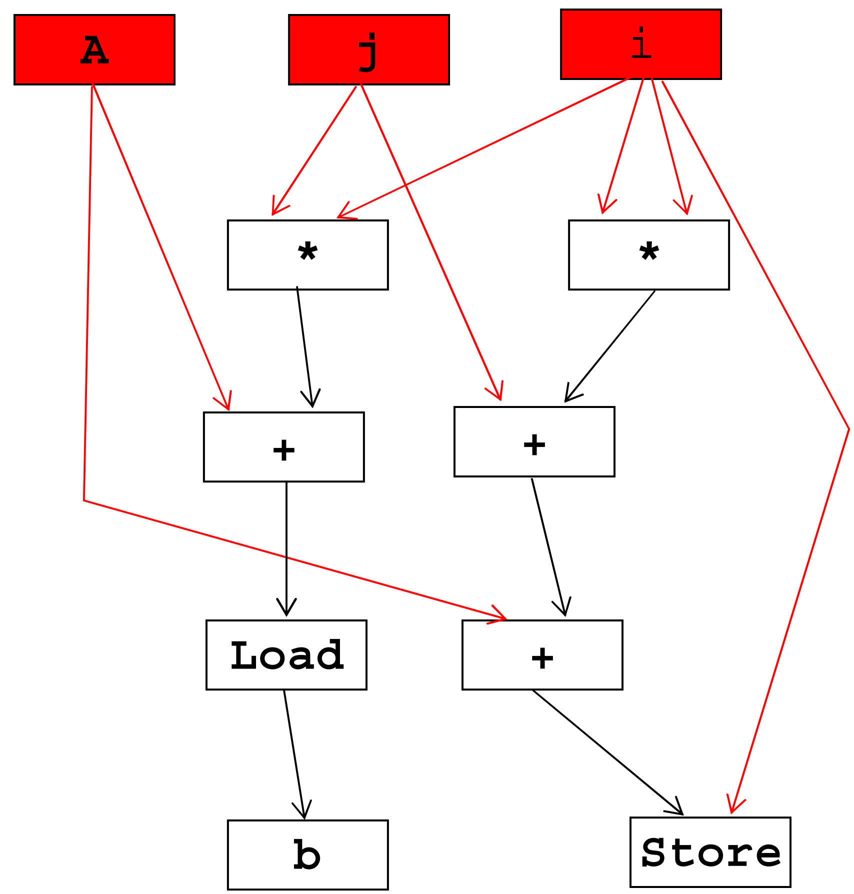
Execute according to the **dataflow firing rule**

- when operands of an instruction have arrived on all input arcs, instruction may execute
- value on input arcs is removed
- computed value placed on output arc



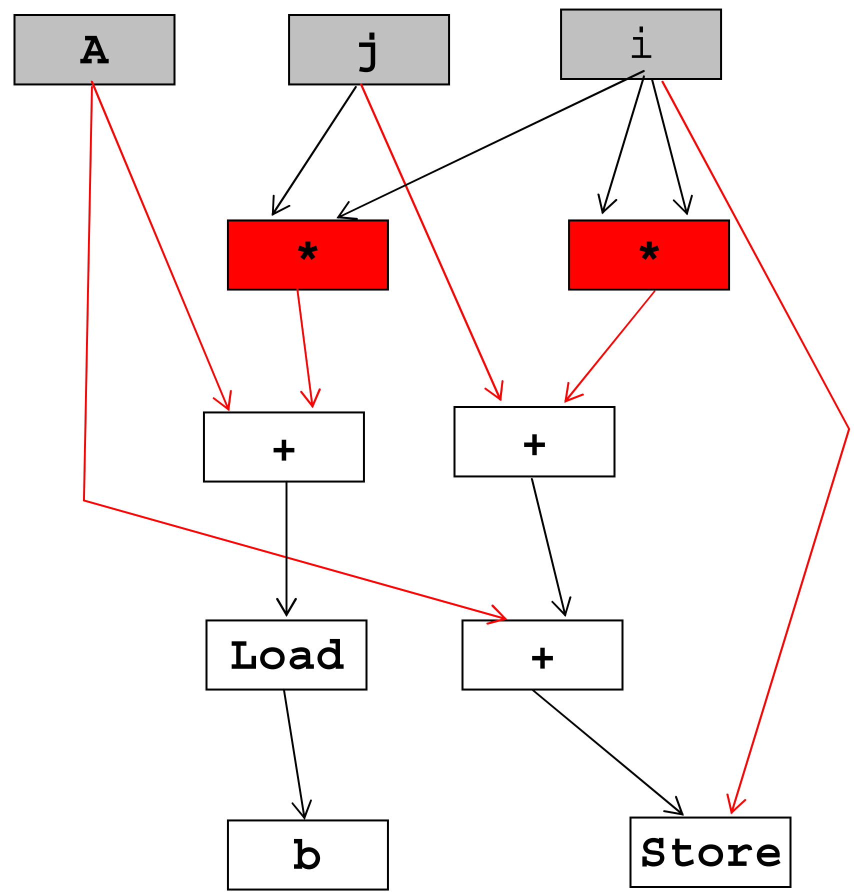
Dataflow Example

```
A[j + i*i] = i;

b = A[i*j];
```



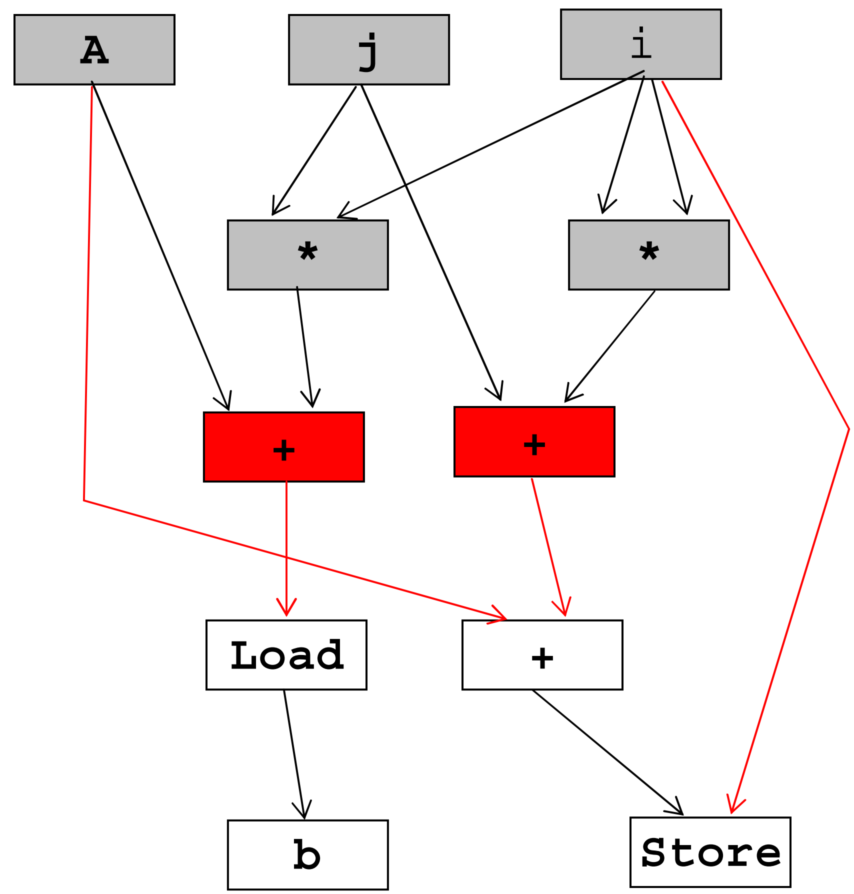
Dataflow Example

```
A[j + i*i] = i;

b = A[i*j];
```



Dataflow Example

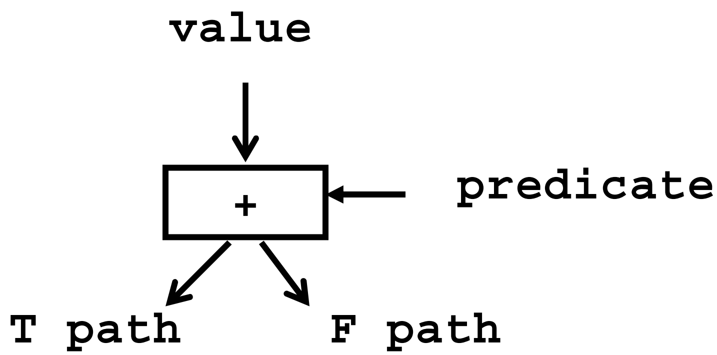
```
A[j + i*i] = i;

b = A[i*j];
```



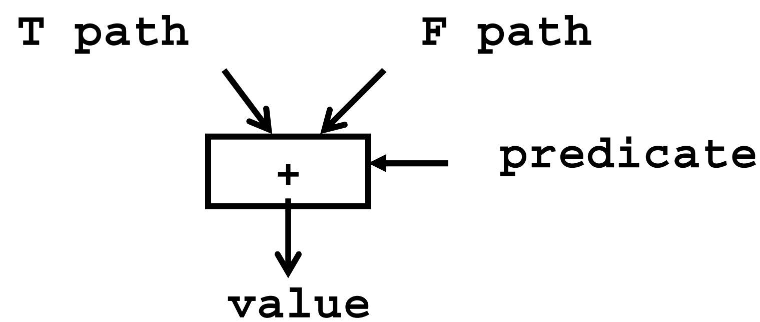
Dataflow Computers

Control

- split



merge



- convert control dependence to data dependence with value-steering instructions
- can either execute both paths & pass values at end with a merge or execute one path after condition variable is known

Dataflow Computers

Data Tokens

- value
- tag to identify the operand instance & match it with its fellow operands in the same dynamic instruction instance
- architecture dependent
  - instruction number
  - iteration number
  - activation number (for functions, especially recursive)
  - thread number

Instructions

- operation
- destination instructions

Types of Dataflow Computers

static:

- one copy of each instruction
- no simultaneously active iterations, no recursion

dynamic

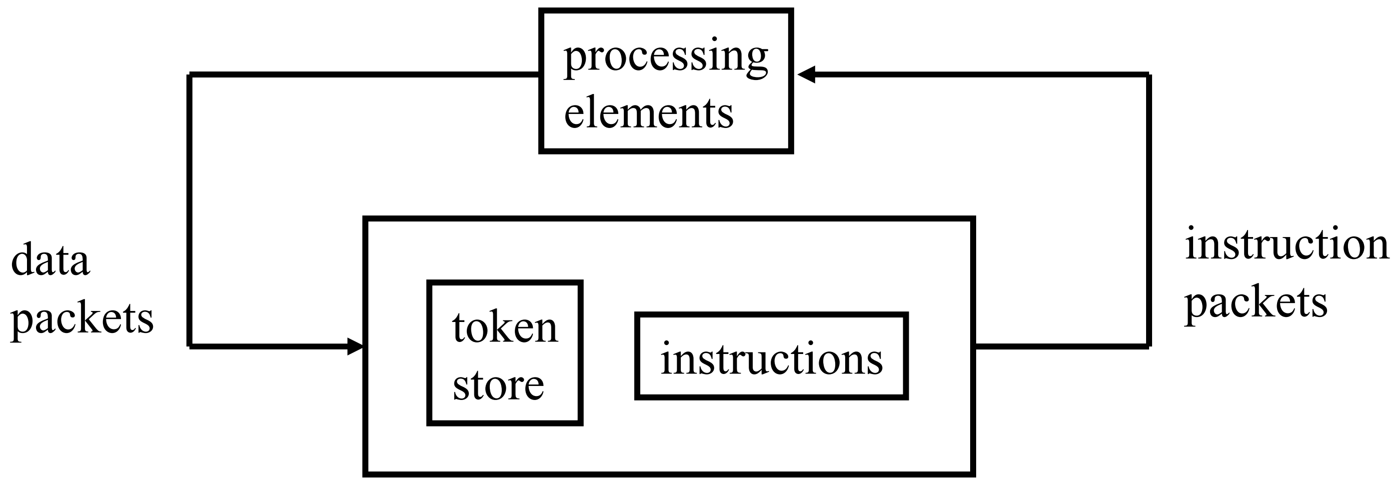
- multiple copies of each instruction
- gate counting technique to prevent instruction explosion:

k-bounding

- extra instruction with K tokens on its input arc; passes a token to 1st instruction of loop body
- 1st instruction of loop body consumes a token (needs one extra operand to execute)
- last instruction in loop body produces another token at end of iteration
- limits active iterations to k

Prototypical Early Dataflow Computer

Original implementations were centralized.



Performance cost

- associative search of *large* token store
- long wires
- arbitration for PEs and return of result

Problems with Dataflow Computers

Language compatibility

- dataflow cannot guarantee a global ordering of memory operations
- dataflow computer programmers could not use mainstream programming languages, such as C
- developed special languages in which order didn't matter

Scalability: large token store

- side-effect-free programming language with no mutable data structures
- 1000 tokens for 1000 data items even if the same value

Solving the Problems

Partial solution in data representation

- **I-structures**: write once; read many times
- **M-structures**: multiple reads & writes, but alternate like full/empty bits

Partial solution in frames of sequential instruction execution

- dataflow execution of coarse-grain threads

Partial solution in local (register) storage

Solutions led away from pure dataflow execution