CS533: Dataflow Architectures

Josep Torrellas

University of Illinois in Urbana-Champaign

Motivation

- What is a Von Neumann architecture?

Motivation

- What is a Von Neumann architecture?
  - the traditional processor with register, pc (control unit), memory, alu

Motivation

- What is a Von Neumann architecture?
  - the traditional processor with register, pc (control unit), memory, alu
- Why is this design problematic for parallel systems?

Motivation

- What is a Von Neumann architecture?
  - the traditional processor with register, pc (control unit), memory, alu
- Why is this design problematic for parallel systems?
  - instructions can stall the pipeline, yet there still might be some non-dependent instructions ready to execute

Motivation

- What is a Von Neumann architecture?
  - the traditional processor with register, pc (control unit), memory, alu
- Why is this design problematic for parallel systems?
  - instructions can stall the pipeline, yet there still might be some non-dependent instructions ready to execute
  - it is hard to context switch (with a large number of threads) – information is stored in the registers, control unit that is specific to a certain context

Motivation

- What is a Von Neumann architecture?
  - the traditional processor with register, pc (control unit), memory, alu
- Why is this design problematic for parallel systems?
  - instructions can stall the pipeline, yet there still might be some non-dependent instructions ready to execute
  - it is hard to context switch (with a large number of threads) – information is stored in the registers, control unit that is specific to a certain context
  - is this more of a problem in parallel machines?

Motivation

- Ideally, in what scenarios does the processor have to stall?

Motivation

- Ideally, in what scenarios does the processor have to stall?
  - has finished executing
  - communication latency
  - alu operation latency

Motivation

- Ideally, in what scenarios does the processor have to stall?
  - has finished executing
  - communication latency
  - alu operation latency
- In other words, a processor should only stall when given the current data state it is impossible to execute any program instructions

Dataflow Graphs

- In compilers, often dataflow graphs are used to represent programs

Dataflow Graphs

- In compilers, often dataflow graphs are used to represent programs
- Is a graph where instructions are nodes and edges represent dependences between instructions

Dataflow Graphs

- In compilers, often dataflow graphs are used to represent programs
- Is a graph where instructions are nodes and edges represent dependences between instructions
- Why do compilers use this representation? Why is this representation advantageous for parallel programs?

Dataflow Graphs

- In compilers, often dataflow graphs are used to represent programs
- Is a graph where instructions are nodes and edges represent dependences between instructions
- Why do compilers use this representation? Why is this representation advantageous for parallel programs?
- It clearly shows what instructions can be executed concurrently

Fine Grain Parallelism

- Is there a fundamental reason to have a few threads each executing a large number of instructions?

Fine Grain Parallelism

- Is there a fundamental reason to have a few threads each executing a large number of instructions?
- Are there practical reasons for having a few threads?

Fine Grain Parallelism

- Is there a fundamental reason to have a few threads each executing a large number of instructions?
- Are there practical reasons for having a few threads?
  - context switching/management overhead
  - synchronization

Fine Grain Parallelism

- Is there a fundamental reason to have a few threads each executing a large number of instructions?
- Are there practical reasons for having a few threads?
  - context switching/management overhead
  - synchronization
- Instead, use a *data-driven architecture!*

Data-Driven Execution

- what is data-driven execution?

Data-Driven Execution

- what is data-driven execution?
- instead of executing instructions according to a program counter, execute instructions as soon as their operands are available

Data-Driven Execution

- what is data-driven execution?
- instead of executing instructions according to a program counter, execute instructions as soon as their operands are available
- This is nicely related to the dataflow graph technique mentioned earlier

Data-Driven Execution

- what is data-driven execution?
- instead of executing instructions according to a program counter, execute instructions as soon as their operands are available
- This is nicely related to the dataflow graph technique mentioned earlier
- What are advantages of this data driven approach?

Data-Driven Execution

- what is data-driven execution?
- instead of executing instructions according to a program counter, execute instructions as soon as their operands are available
- This is nicely related to the dataflow graph technique mentioned earlier
- What are advantages of this data driven approach?
  - more concurrency is exposed $\Rightarrow$ any "ready" instruction can be scheduled to execute next
  - synchronization is handled implicitly through the data dependences.

Data-Driven Exeuction

- What types of workloads benefit most from a data-driven approach? Are there other ways to approaches to achieve speedup on these workloads

Data-Driven Exeuction

- What types of workloads benefit most from a data-driven approach? Are there other ways to approaches to achieve speedup on these workloads
  - While SIMD machines are effective for regular parallelism, data-driven machines can also theoretically work well with irregular parallelism

Data-Driven Exeuction

- What types of workloads benefit most from a data-driven approach? Are there other ways to approaches to achieve speedup on these workloads
  - While SIMD machines are effective for regular parallelism, data-driven machines can also theoretically work well with irregular parallelism
- How does the data-driven design compare with ASICs/accelerators?

Data-Driven Exeuction

- What types of workloads benefit most from a data-driven approach? Are there other ways to approaches to achieve speedup on these workloads
  - While SIMD machines are effective for regular parallelism, data-driven machines can also theoretically work well with irregular parallelism
- How does the data-driven design compare with ASICs/accelerators?
- Many overheads of the Von Neuman design, such as fetching and centralized memory structures can be reduced.

Introduction

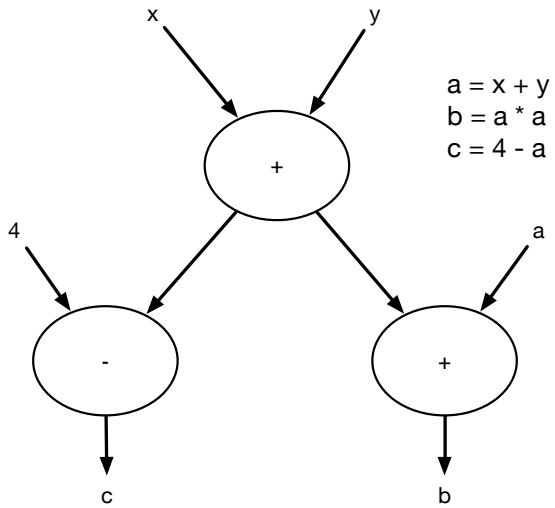
- First articulated by Jack Dennis at MIT in 1969

Introduction

- First articulated by Jack Dennis at MIT in 1969
- In dataflow machines, hardware is optimized for fine-grain data-driven parallel computation

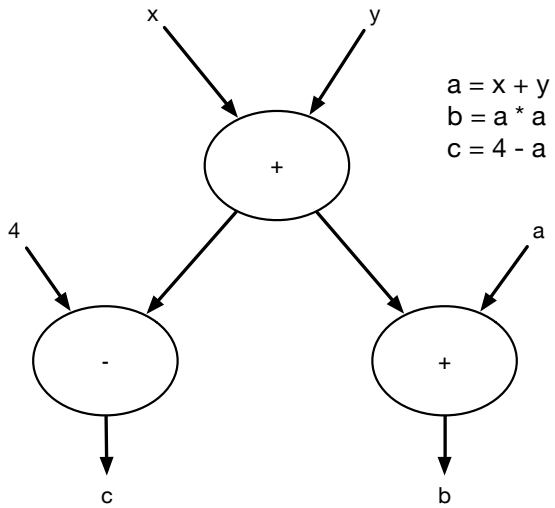
Introduction

- First articulated by Jack Dennis at MIT in 1969
- In dataflow machines, hardware is optimized for fine-grain data-driven parallel computation
- Data-driven computation is intuitively appealing because:
  - Only data dependences constrain parallelism
  - Programs usually represented as graphs



Introduction

- First articulated by Jack Dennis at MIT in 1969
- In dataflow machines, hardware is optimized for fine-grain data-driven parallel computation
- Data-driven computation is intuitively appealing because:
  - Only data dependences constrain parallelism
  - Programs usually represented as graphs



- Although around for 40 years, little impact on mainstream computing
  - However, ideas used in OoO superscalars

Terminology

- Instructions are *nodes* (e.g. $+$, -, . . .)

Terminology

- Instructions are *nodes* (e.g. +, -, . . .)
- Data items are *tokens* (e.g. 8.2, true, 4, . . .)

Terminology

- Instructions are *nodes* (e.g. +, -, . . .)
- Data items are *tokens* (e.g. 8.2, true, 4, . . .)
- Producers of tokens are connected to consumers by *arcs*

Terminology

- Instructions are *nodes* (e.g. +, -, . . .)
- Data items are *tokens* (e.g. 8.2, true, 4, . . .)
- Producers of tokens are connected to consumers by *arcs*
- Entry points to nodes are *input ports*

Terminology

- Instructions are *nodes* (e.g. +, -, . . .)
- Data items are *tokens* (e.g. 8.2, true, 4, . . .)
- Producers of tokens are connected to consumers by *arcs*
- Entry points to nodes are *input ports*
- A nodes is enabled by an *enabling rule*. This typically is when **ALL** arcs have tokens on their inputs (strict enabling rule)
  - non-strict allows firing before all arcs have tokens

Terminology

- Instructions are *nodes* (e.g. +, -, . . .)
- Data items are *tokens* (e.g. 8.2, true, 4, . . .)
- Producers of tokens are connected to consumers by *arcs*
- Entry points to nodes are *input ports*
- A nodes is enabled by an *enabling rule*. This typically is when **ALL** arcs have tokens on their inputs (strict enabling rule)
  - non-strict allows firing before all arcs have tokens
- a node *fires* after it is enabled, and in this process consumes tokens on its input arcs and produces tokens on its output arcs.

Node Types

Node Types

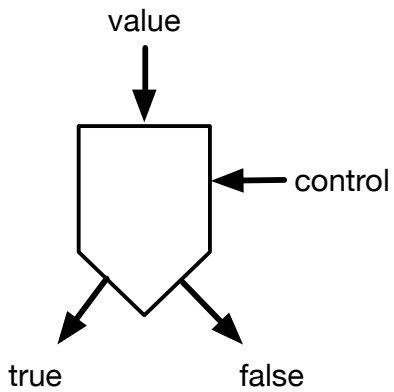
- Functional: Output depends only on inputs and node type (e.g. +, -, ...)

Node Types

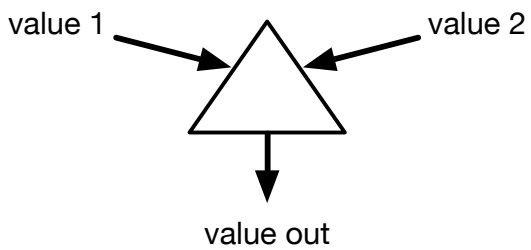
- Functional: Output depends only on inputs and node type (e.g. +, -, ...)
- Conditional: Depending on control, token picked up from value input is passed onto true-output arc or false-output arc.

Node Types

- Functional: Output depends only on inputs and node type (e.g. +, -, ...)
- Conditional: Depending on control, token picked up from value input is passed onto true-output arc or false-output arc.
- Merge: Whenever there is a token on any of the input arcs, it is immediately copied to the output arc.
  - non-strict firing rule
  - acts as serializer/merger



Conditional



Merge

Example

- If Statement

Example

- If Statement
- Do-While loop

Programming Dataflow Machines

- Dataflow Machines need a different ISA than traditional architectures.

Programming Dataflow Machines

- Dataflow Machines need a different ISA than traditional architectures.
- Whose responsiblity is it to create machine code for these machines?

Programming Dataflow Machines

- Dataflow Machines need a different ISA than traditional architectures.
- Whose responsiblity is it to create machine code for these machines?
  - open research topic(?)
  - Many project about using high-level languages to generate RTL, how to make programmable accelerators

Iteration, Recursion, Reuse and Reentrancy

- Using the same graph to perform computation on different sets of data can cause problems

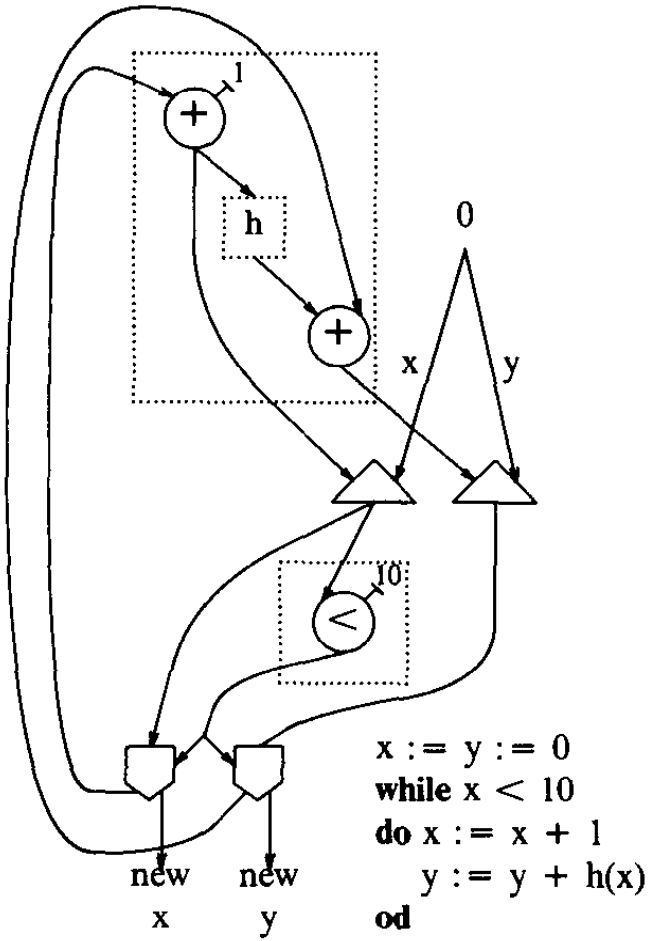
Iteration, Recursion, Reuse and Reentrancy

- Using the same graph to perform computation on different sets of data can cause problems
  - It is easy to imagine scenarios where data is lost or computation between many iterations is jumbled so that corrupted data is generated

Iteration, Recursion, Reuse and Reentrancy

- Using the same graph to perform computation on different sets of data can cause problems
  - It is easy to imagine scenarios where data is lost or computation between many iterations is jumbled so that corrupted data is generated
- In the following, we assume that things mess up if a token arives at an arc while another token is still pending. This happens if each nodes has a token buffer that is only one deep . . . when the second token arrives, the old one gets overwritten or it is unclear which token to use while executing.

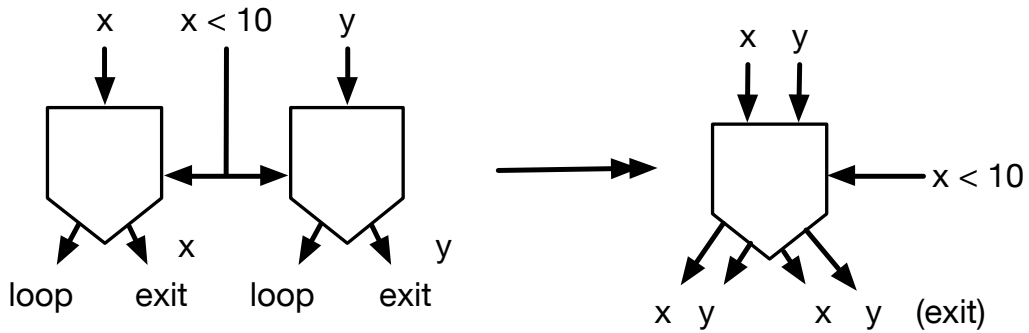
Iteration, Recursion, Reuse and Reentrancy



- h is some arbitrary subgraph
- A second token x=2 can arrive at the input of function h if it takes a long time to compute it, compared to the $\bigoplus$ node
- Two general ways of handling the problem give rise to the classification of dataflow machines into
  - Static
  - Dynamic

Static Dataflow Machines

Use locks: $\Rightarrow$ compound BRANCH & MERGE nodes



- The strict firing rule prevents x from going ahead until y arrives
- Downside: reduces concurrency

Static Dataflow Machines (II)

Alternative: use Acknowledge method: add extra ack arcs from consumer to producer nodes

- like handshake protocols in asynchronous systems

Static Dataflow Machines (II)

Alternative: use Acknowledge method: add extra ack arcs from consumer to producer nodes

- like handshake protocols in asynchronous systems
- guarantees that only one node is in an arc at a time

Static Dataflow Machines (II)

Alternative: use Acknowledge method: add extra ack arcs from consumer to producer nodes

- like handshake protocols in asynchronous systems
- guarantees that only one node is in an arc at a time
- Allows greater concurrency than lock method, but at cost of at least doubling number of arcs and tokens
  - detailed analysis can help reducer number of extra arcs needed

Dynamic Dataflow Machines

- Allow each iteration to be executed in a separate instance of the graph. 2 Ways:

Dynamic Dataflow Machines

- Allow each iteration to be executed in a separate instance of the graph. 2 Ways:
  1. *Code copying*: A new instance of subgraph is created per iteration; need mechanism to direct tokens to right instance

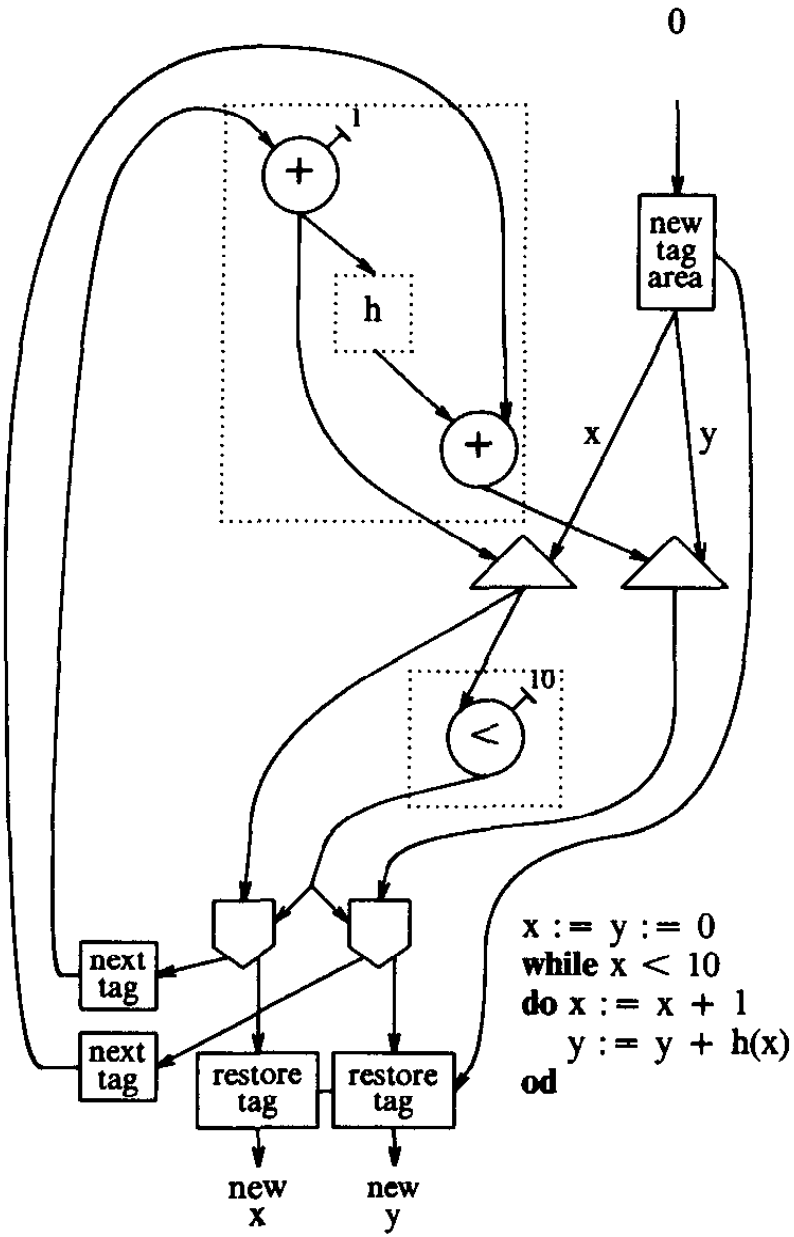
Dynamic Dataflow Machines

- Allow each iteration to be executed in a separate instance of the graph. 2 Ways:
  1. *Code copying*: A new instance of subgraph is created per iteration; need mechanism to direct tokens to right instance
     - This helps since different nodes will handle the execution for each iteration
     - Is a little like loop unrolling in that more parallelism is exposed

Dynamic Dataflow Machines

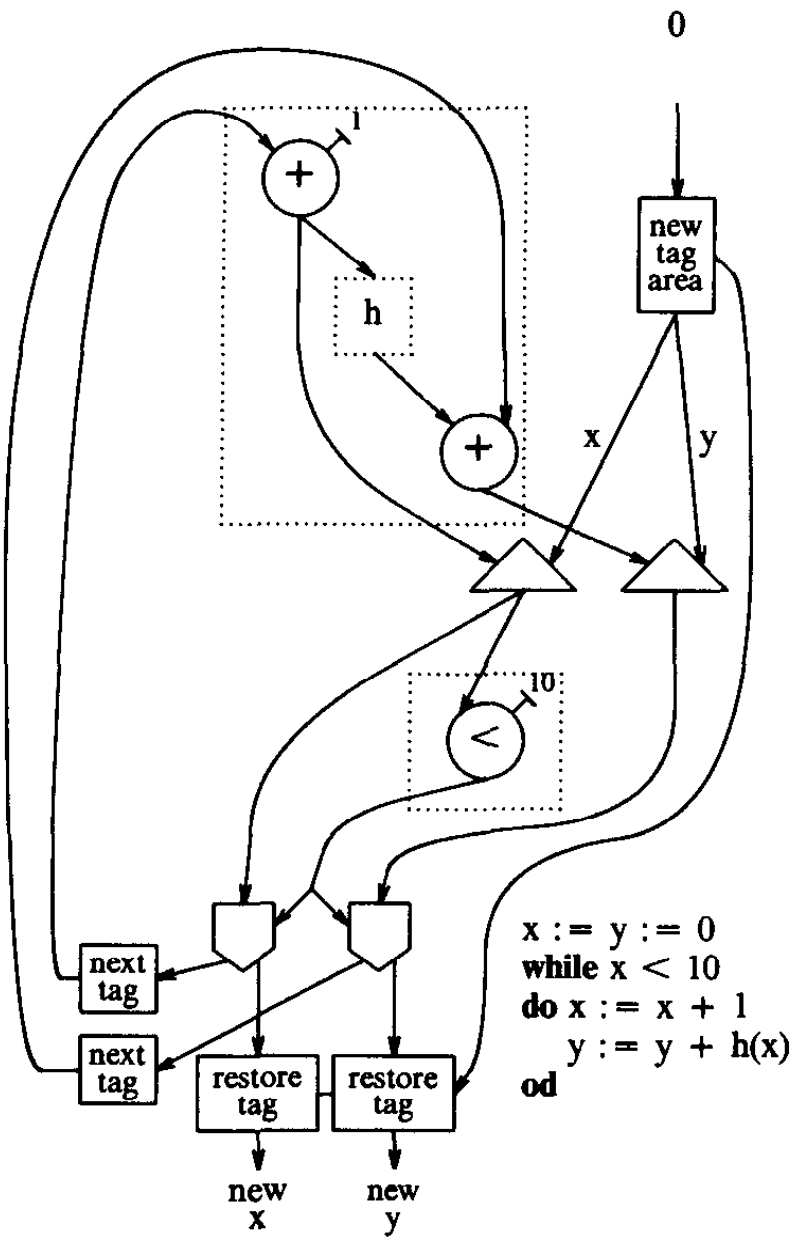
- Allow each iteration to be executed in a separate instance of the graph. 2 Ways:
  1. *Code copying*: A new instance of subgraph is created per iteration; need mechanism to direct tokens to right instance
     - This helps since different nodes will handle the execution for each iteration
     - Is a little like loop unrolling in that more parallelism is exposed
     - have to consider the overhead of having extra nodes
  2. *Tagged tokens*: Attach a tag to each token saying what iteration it belongs to
     - Enabling rule: fire when tokens with same tag are present at each of the inputs of the node

Tag Management



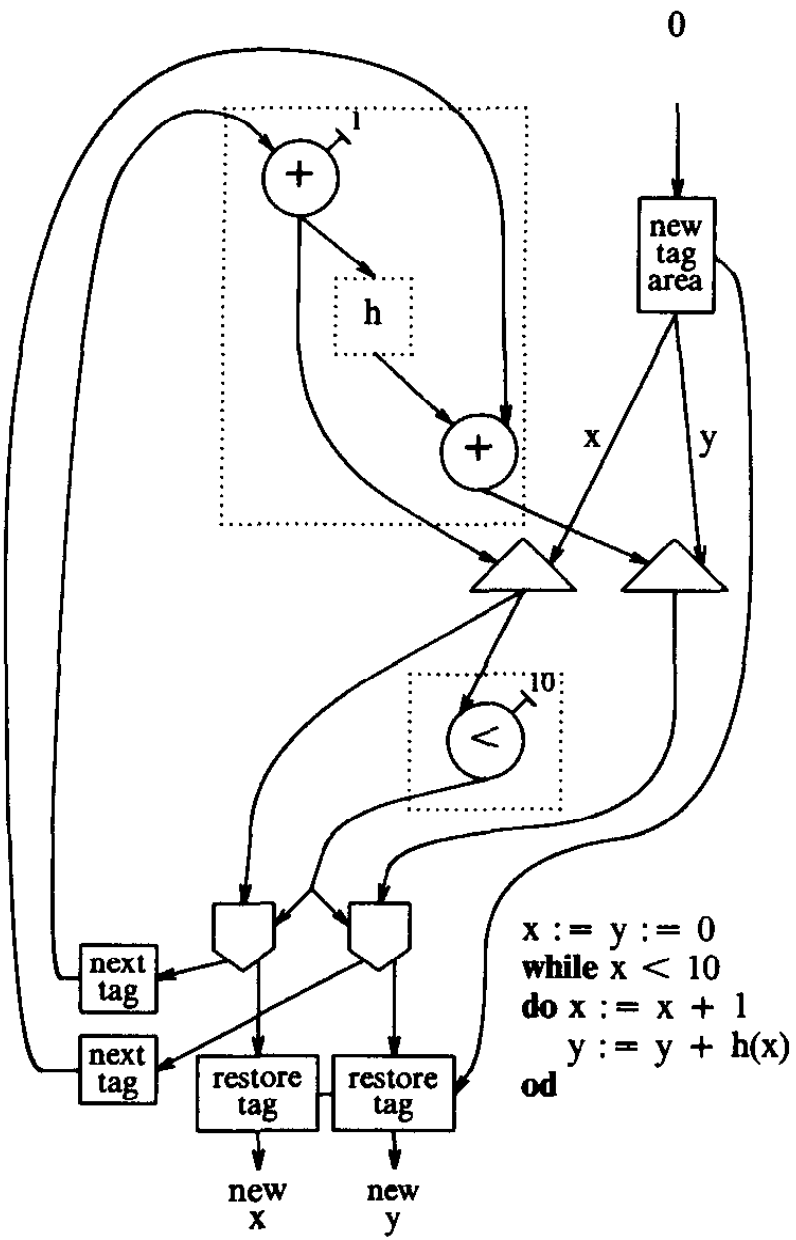
- How to generate distinct tags in a distributed environment ($\Rightarrow$ too many bits)

Tag Management



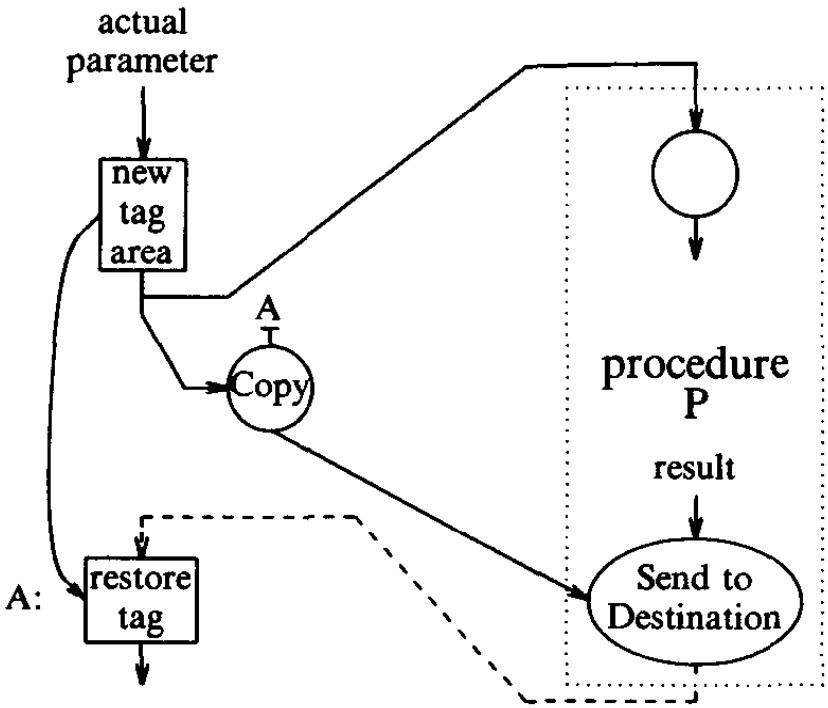
- How to generate distinct tags in a distributed environment ($\Rightarrow$ too many bits)
- Extra HW complexity, since tags have to be matched . . . Tags also have storage overhead

Tag Management

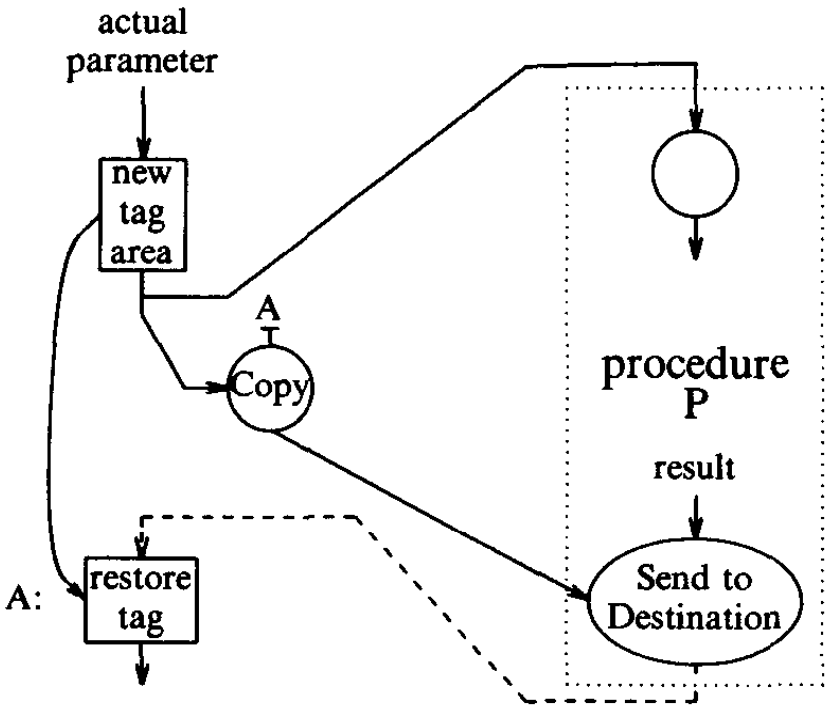


- How to generate distinct tags in a distributed environment ($\Rightarrow$ too many bits)
- Extra HW complexity, since tags have to be matched . . . Tags also have storage overhead
- The flexibility may overexpose parallelism, possibly causing deadlocks or storage problems

Handling Procedures

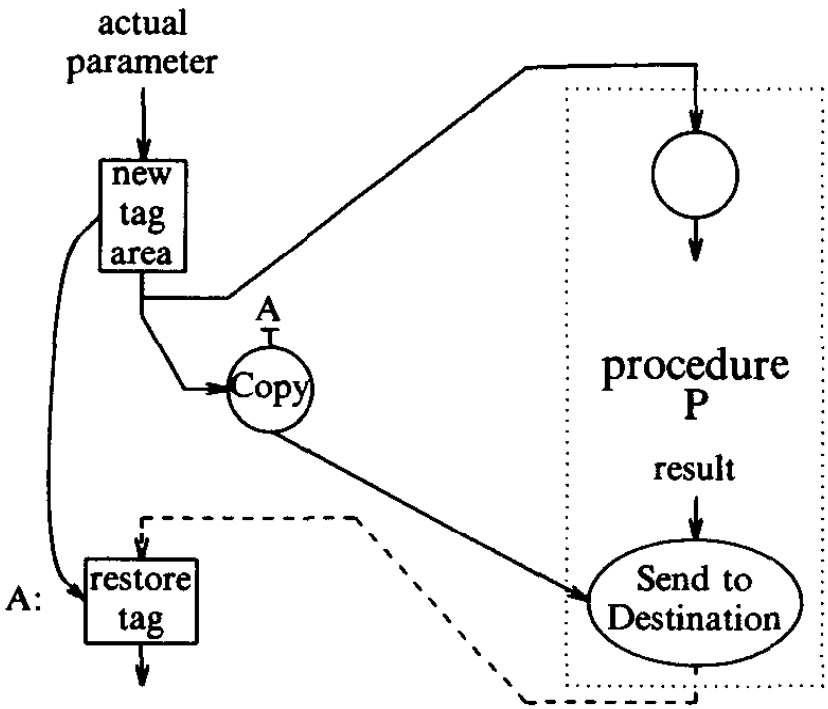


Handling Procedures



- Extra facility required to direct the output token of the procedure to the proper calling site ⇒ done by sending special token containing return node address

Handling Procedures

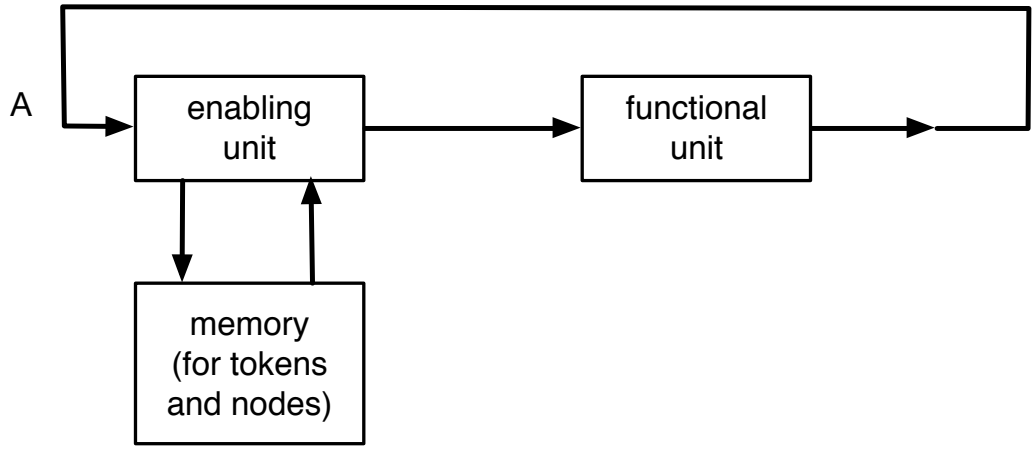


- Extra facility required to direct the output token of the procedure to the proper calling site ⇒ done by sending special token containing return node address
- *Summary*:
  - Static: simpler, good for pipelining
  - Dynamic: more concurrency but memory and complexity overhead

Architecture of Dataflow Machines

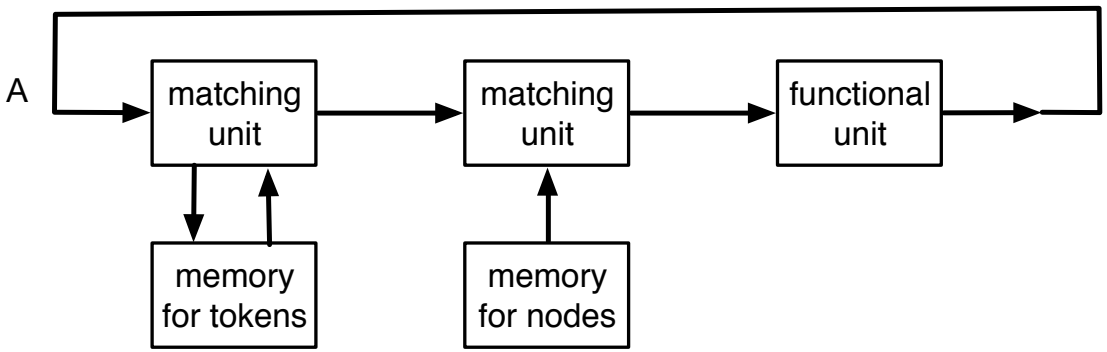
- composed of several PEs that communicate with each other

example of PE:



- Enabling unit: accepts tokens from (A) and stores them at addressed node. If node is enabled: executable packet sent to FU to be processed
- Output tokens, with destination addresses are sent back to enabling unit
- token value are stored alongside the nodes
- poses problems when tagging is used

Tagged Machines



When a token arrives in (A), check in memory for tokens to see if all other inputs with the same tag are here. If so, send all tokens to the fetching unit, which combines them with a copy of the node description into an executable packet to be passed onto the functional unit

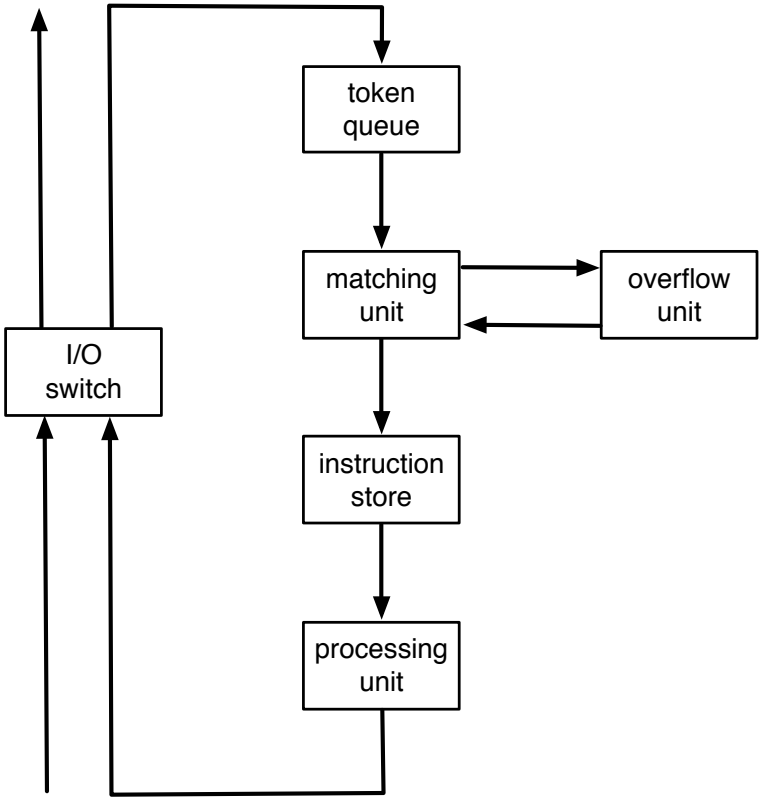
Machine Architecture

- One level: Deliver tokens produced by a functional unit to the enabling unit of the correct PE
  - destination address
  - allocation policy is tricky
  - just like message passing except dataflow processors
  - e.g. Arvind
- Two-level: each FU consists of several functional elements which concurrently process executable packets
  - e.g. Gurd (Manchester)
- Two-Stage: Each Enabling unit can send executable packets to each FU (there are more functional elements available)
  - \+ Heterogeneous PEs
  - \+ Scheduling flexibility
  - – More cost

The Manchester Dataflow Machine

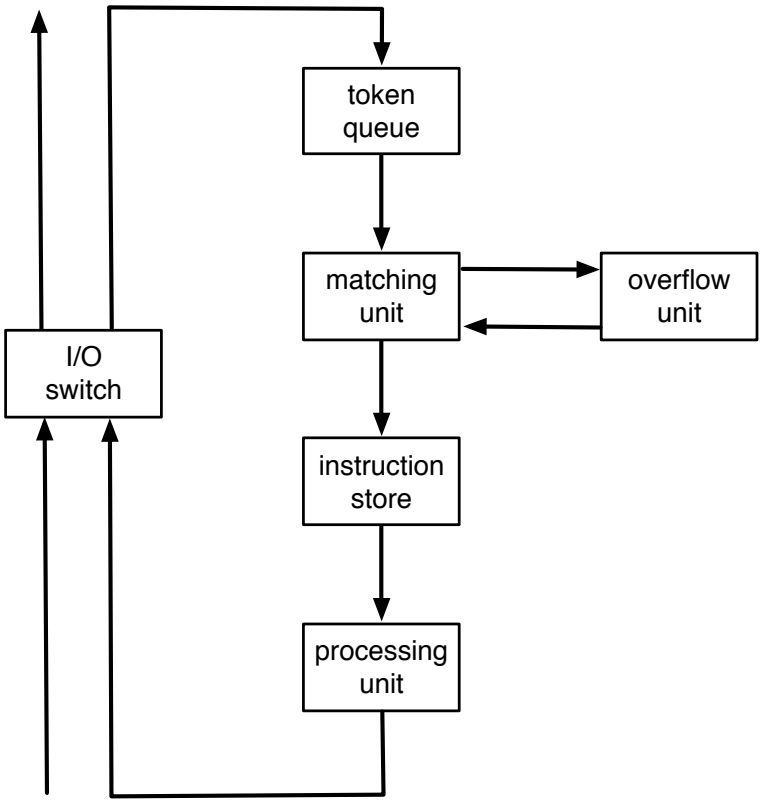
Gurd and Watson; University of Manchetser, UK. Started 1976; Working 1981

Two-level machine, although only 1 macroprocessor implemented



- Meant to be 1 PE in a system
- tokens carred in packets around pipelined ring
- Matching unit: pairs together tokens destined for same instruction
- Programs with large data set overflow in overflow unit

The Manchester Dataflow Machine (Continued)



- Paired tokens fetch the appropriate instruction from the instruction store constains the machine code for the dataflow program
- instruction + inputs forwarded to PU
- token queue: buffer to smooth traffic
- I/O switch connection to host

Matching Unit

- each token tag indicates whether the destination is expecting one or two tokens
- if only zero or one token is needed, then the token is immediately sent to the *fetching unit* (instruction store) and linked with the node
- if the destination node is expecting two tokens, the matching unit is searched for the other port
- the matching unit is very similar to a cache – each row has 16 ways
- if a companion token is not found, then the result is stored in the matching unit

Performance

- Poor: 5-10 times slower than VAX-11/780 on RSIM (switch level simulation)
- better technology: claim 5x better
- average parallelism = AP = S_1/S_∞
  - S_1 = time steps with 1 FU
  - S_∞ = time steps with ∞ FUs

Program	AP
Laplace	134-210
Matmult	236
Rsim	15-20
Simple	63

Rsim and Simple are SISAL

- found that need AP $\geq$ 40 to get good speedup

Possible Drawbacks

- Hard to properly balance the system (#FUs, mem sizes, etc.)
- There is a lot of overhead with using branch/merge nodes that results in low instruction efficiency
- Too much parallelism can negatively affect the system
  - matching store unit has to be very large
  - a lot of bits have to be used for a tag
  - there are ways to reduce this overhead