Lecture 1

Dataflow Model of Parallelism

Arvind
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Oregon Programming Language Summer School (OPLSS)
Eugene, OR
July 14, 2018
Dataflow
Jack Dennis 1969-1973

General purpose parallel machines based on a dataflow graph model of computation

Inspired all the major players in dataflow during seventies and eighties, including Kim Gostelow and me @ UC Irvine
Dataflow Operators

A small set of dataflow operators can be used to define a general programming language.
Exercise: Construct a graph for $f(g(x))$ using juxtaposition and iteration.
Example:

The Stream Duplicator

Gate: Lets X pass through only after C arrives

What happens if we don't use the gate?
The Stream Halver

Draw a graph that throws away every other token
Determinacy Property

The values of output tokens are uniquely determined by the values of input tokens, i.e., *the behavior is time independent*.

Theorem: A dataflow graph formed by repeated *juxtaposition* and *iteration* of deterministic dataflow operators is deterministic.

Proof?
Kahn Process Networks
Gilles Kahn 1973

- Computing stations connected by *unbounded, FIFO channels*
- Each station executes a *sequential program*
  - `wait(ch)`: blocking read from a channel
  - `send(x, ch)`: non blocking

A station either *blocks* for an input on a specific channel or computes (*no test for emptiness*)
An Example

\[ X = f(Y, Z) \ |\ |\ T_0, T_1 = g(X) \ |\ |\ Y = h_0(T_0) \ |\ |\ Z = h_1(T_1) \]

**Process** \( f(U, V; W) \)

\[
\begin{align*}
b &:= \text{true}; \\
\text{While } \text{true} \text{ do} &\text{ } \\
\{ &i := \text{if } b \text{ then wait}(U) \\
\text{else } &\text{ wait}(V); \\
\text{print}(i); \\
\text{send}(i, W); \\
b &:= \text{not } b \}
\end{align*}
\]

**Process** \( g(U ; V, W) \)

\[
\begin{align*}
b &:= \text{true}; \\
\text{While } \text{true} \text{ do} &\text{ } \\
\{ &i := \text{ wait}(U); \\
\text{if } b &\text{ then send}(i, V) \\
\text{else } &\text{ send}(i, W); \\
b &:= \text{not } b \}
\end{align*}
\]

**Process** \( h_c(U ; V) \)

\[
\begin{align*}
&\{ \text{send}(c, V); \\
\text{While } \text{true} \text{ do} &\text{ } \\
\{ &i := \text{ wait}(U); \\
\text{if } b &\text{ then send}(i, V) \\
\text{else } &\text{ send}(i, V) \}
\end{align*}
\]

Functionality?
Dataflow Graphs can Express any Kahnian Network and vice versa
Determinacy

A computing station in Kahnian network can be viewed as a function from sequences to sequences.

A Kahnian network can be viewed a systems of recursive equations, whose solutions characterize the I/O behavior of the network.

Kleene’s Fixed Point Theorem: If each function in the network is monotonic and continuous, then the set of recursive equations has a unique least fixed-point solution.
Dataflow Operators as Streams Functions

\[
\text{add}(x:xs, y:ys) = + (x,y) : \text{add}(xs,ys)
\]

\[
\text{T-gate} (T:bs, x:xs) = x : \text{T-gate}(bs,xs)
\]
\[
\text{T-gate} (F:bs, x:xs) = \text{T-gate}(bs,xs)
\]

\[
\text{merge}(T:bs, x:xs, ys) = x : \text{merge}(bs,xs,ys)
\]
\[
\text{merge}(F:bs, xs, y:ys) = y : \text{merge}(bs,xs,ys)
\]
Dataflow Graphs: A Set of Recursive Equations

\[ O = T\text{-gate} (A,I); \]
\[ A = \text{gate} (I,B); \]
\[ B = T : \text{Not} (A); \]

G.Kahn: Monotonicity and Continuity of operators guarantee a unique solution, aka determinacy

Kahn Networks = Dennis networks
Domain of Sequences

- **Sequence**: \([x_1, \ldots, x_n]\)
- **The least element**: \([\ ]\) (aka \(\perp\))
- **The partial order** (\(\leq\)): prefix order on sequences
  - \([\ ] \leq [x_1] \leq [x_1, x_2] \leq \ldots \leq [x_1, x_2, x_3 \ldots x_n]\)
- **Monotonicity**: \(x \leq y \implies f(x) \leq f(y)\)
  - A monotonic operator on sequences can never retract a value that has been produced.
- **Continuity**: \(f(\bigcup_i X_i) = \bigcup_i f(X_i)\)
  - A continuous operator on sequences does not suddenly produce an output after consuming an infinite amount of input
- **Least fixed-point solution**: \(f(f(\ldots(f(\perp))\ldots))\)
Computing the solution
an example

\[ O = T\text{-gate} (A,I) ; A = \text{gate} (I,B) ; B = T:\text{Not} (A) ; \]

Assume edges without initial tokens are empty (\(\perp\)). The least fixed point solution can be computed iteratively, by evaluating one operator at a time.

\[
\begin{array}{ccc}
I & [i_1, i_2, i_3] & [i_1, i_2, i_3] & [i_1, i_2, i_3] \\
A & [ ] & [ T ] & [ T ] \\
O & [ ] & [ ] & [ i_1 ]
\end{array}
\]
Well Behaved Schemas
one-in-one-out & self initializing

Before

After

Conditional

Loop
Well Behaved Schemas
one-in-one-out & self initializing

Before

\[ P \]

After

\[ P \]

Conditional

Bounded Loop

Needed for resource management
Another Conditional Schema

If $p$ then $f(a,b)$ else $g(a,b)$

Why is this schema not well behaved?
Yet another conditional schema

Suppose $g(X_2)$ computes much faster than $f(X_1)$.

Tokens will come out in the wrong order without the merge operator.

Suppose $g(X_2)$ computes much faster than $f(X_1)$. 
Tagged-Token Dataflow

Increasing the concurrency
Missing Tokens and blocking behavior

x cannot be moved to the output because the token corresponding to T is missing.

How to model stream with “holes”? 
Another Interpretation of DFGs: *Streams with "holes"*

Streams with missing tokens

\[ \{ v_1, v_2, \perp, v_4, \perp, v_6, \ldots \} \]

can be modeled by a set of tokens where tokens carry a tag designating their position in the stream

\[ \{ \langle 1, v_1 \rangle, \langle 2, v_2 \rangle, \langle 4, v_4 \rangle, \langle 6, v_6 \rangle \} \]
Tagged Semantics of Operators

\[
\text{add}(xs, ys) = \{ <i, x+y> | <i, x> \in xs, <i, y> \in ys \}
\]

\[
\text{T-gate}(bs, xs) = \{ <k, x> | <i, T> \in bs, <i, x> \in xs, \\
\forall j \leq i. <j, b_j> \in bs, k = \text{T-Count}(bs, i) \}
\]

\[
\text{merge}(bs, xs, ys) = \{ <i, x> | <i, T> \in bs, <k, x> \in xs, \\
\forall j \leq i. <j, b_j> \in bs, k = \text{T-Count}(bs, i) \}
\cup \{ <i, y> | <i, F> \in bs, <k, y> \in ys, \\
\forall j \leq i. <j, b_j> \in bs, k = \text{F-Count}(bs, i) \}
\]

\[
\text{D}_a(xs) = \{ <i+1, x> | <i, x> \in xs \} \cup \{ <1, a> \}
\]

needed wherever we place an initial token
Ordering on Streams with Holes

Stream with holes: \{ <i, v_i>, <j, v_j>, <k, v_k> \}

The least element: \{ \} (aka \bot)

The partial order (\leq): subset order

It is easy to show that all the operators under the tagged semantics are monotonic and continuous

\implies tagged semantics are also deterministic
Tagged versus FIFO Interpretation

**Theorem:** Suppose the least fixed point of a dataflow program under the FIFO interpretation is $X$ and under the tagged interpretation is $Y$ then $X \leq Y$.

**Proof:** Based on structural induction on the graph structure (juxtaposition and iteration)

*Tagged interpretation has more parallelism*

Arvind & Gostelow 1977
Well Behaved Graphs under tagging

1. One-in-one-out property

2. Self Cleaning: Initially the graph contains no tokens; when all the output tokens have been produced no tokens remain in the graph
Consequences of Tagging

No need of *merge* in well behaved dataflow graphs

Even if $f(x_1)$ completes after $g(x_2)$ the output tokens will carry the "correct" tag

Tagging simplifies well-behaved schemas, increases parallelism at the cost of increased complexity in implementation
Data Structures
Suppose we want to add two arrays, element by element

- Option 1: the whole array is carried by a token (not practical in general)
- Option 2: Array is represented as a stream of tokens
- Option 3: Array is divided into several chunks and each chunk is fed to a different copy of the operator
- Option 4: Some combination of options 2 and 3

Such a choices of representation has to be made by the user or the compiler

These options are not suitable for handling complex or sparse data structures
A Parallel Multithreaded Language Model

Tree of Activation Frames

Global Heap of Shared Objects

Asynchronous and parallel at all levels

Pure dataflow systems cannot express such a language model
Dynamic Dataflow Architectures and Languages (MIT 1979-94)

Extensions to the tagged dataflow model to support procedure calls and dynamically allocated heap objects
Data Structures in Dataflow

- Data structures reside in memory and tokens carry pointers

- I-structures
  - Write-once, Read multiple times
  - allocate → write → read* → deallocate

- I-structures are like variables in logic languages (Pingali ...)
  - See I-vars, and M-vars in Haskell

- At the hardware level, I-fetch and I-store instructions
  No problem if a reader arrives before the writer at the memory location
I-Structure Storage:
Split-phase operations & Presence bits

Need to deal with multiple deferred reads;
Additional operations: take/put, clear
Language and Architecture to exploit *implicit parallelism*

Id (Functional language)

Dataflow Graphs + I-Structures + . . .

Monsoon Dataflow Machine
The Monsoon Project
Motorola Cambridge Research Center + MIT (1988-91)

Two 16-node systems
- MIT, LANL

Sixteen 2-node systems
- Motorola, Colorado, USC, Oregon, McGill, ...

Id World Software

Displayed at Super Computing 1991
The Monsoon Project
Motorola Cambridge Research Center + MIT (1988-91)
Dataflow people @ MIT

- Rishiyur Nikhil
- Keshav Pingali
- Vinod Kathail
- David Culler
- Greg Papadopolous
- Andy Boughton
- Chris Jeorge
- Ken Traub
- Steve Heller
- Richard Soley
- Dinart Mores
- Jamey Hicks
- Alex Caro
- Andy Shaw
- Boon Ang
- Shail Anditya
- R Paul Johnson
- Jacj Costenza
- Paul Barth
- Jan Maessen
- Christine Flood
- Jonathan Young
- Derek Chiou
- Arun Iyangar
- Zena Ariola
- Mike Bekerle
- K. Eknadham (IBM)
- Wim Bohm (Colorado)
- Joe Stoy (Oxford)
- ...
Dataflow: An assessment

- Dataflow ideas are used extensively in modern microprocessors, but the Instructure Set Architecture (ISA) remains totally sequential
  - Processors use *speculation* to expose parallelism
- Compilers extract parallelism from sequential codes with varying degree of success
- Parallel programming remains difficult – *most* users have little interest in rewriting their programs to exploit parallelism
  - Ethos: *It is someone else’s program*
- Yet, most microprocessors are multicores and most systems have many concurrent processes

*Move towards domain specific solutions*
Fine-grain vs Coarse-grain Dataflow Models

- Dataflow models say nothing about the size of tokens or the complexity of computing nodes.
- In practice, the cost of implementing fine-grain asynchronous parallelism is high, so many systems use dataflow ideas at the macro-level and exploit fine-grain synchronous parallelism within the nodes.
Applications

Dataflow models are used extensively in many domains, especially for *hardware accelerators*:

- Video and Audio codecs (e.g., H.264)
- Network applications (e.g., packet filtering)
- Wireless baseband processing (e.g., OFDM based protocols)
- Deep Neural Networks (e.g., DNNs, RNNs)
- Map-Reduce applications
- ...
H.264 Video Decoder
Chun-Chieh Lin, K Elliott Fleming [MEMOCODE 2008]

May be implemented in hardware or software depending upon ...

Different requirements for different environments
- QVGA 320x240p (30 fps)
- DVD 720x480p
- HD DVD 1280x720p (60-75 fps)
Wireless Baseband processing
OFDM TX/RX Blocks, Man cheuk Ng et al [2007]
Wireless Baseband processing
OFDM TX/RX Blocks,  Man cheuk Ng et al [2007]

Different algorithms - No Reuse

Convolutional
Reed-Solomon
Turbo
Wireless Baseband processing
OFDM TX/RX Blocks, Man cheuk Ng et al [2007]

- Different algorithms - No Reuse
- Reusable algorithm with different parameter settings

WiFi: $x^7+x^4+1$
WiMAX: $x^{15}+x^{14}+1$
WUSB: $x^{15}+x^{14}+1$
Different algorithms - No Reuse
Reusable algorithm with different parameter settings
Different throughput requirements

We want the same source description but different hardware!
There is a need to beyond dataflow models

- Dataflow models, like functional languages, obscure resource management issues which must be handled efficiently in any practical system.
- Lack of shared state between nodes makes certain computations, e.g., graph algorithms, difficult to express.
- *Determinacy*, though highly desirable in many situations, is ultimately quite limiting in specifying distributed software and hardware.
Alternative

- Serializable atomic actions

  - A general model of concurrency for both software and hardware
  - Can express most other models of concurrency
  - A specific model – *guarded atomic actions* – is also amenable to efficient hardware implementation

*stay tuned...*
Extras
Firing Rules: T-Gate
The Switch Operator
Firing Rules: Merge
Firing Rules: Merge \textit{cont}

not ready to fire
Some Conventions
Some Conventions \textit{Cont.}
Unbounded Cyclic Graphs

Unbounded number of tokens on an arc can only arise due to cycles.
Bounded Cyclic Graphs
Bounded Cyclic Graphs
Well Behaved Schemas

Bounded Loop

Needed for resource management
New Definition of Well Behavedness

1. One token on each input arc produces exactly one token on each output arc.
2. The initial distribution of tokens on the arcs is restored.
3. No arc can have an unbounded buildup of tokens.
Bounded Cyclic Graphs are *Well Behaved*

- Initial number of tokens at the *gate* input determines the maximum number of tokens on any arc.

- However, loop bounding can alter the "meaning" of a graph, i.e., can cause *deadlock*.

- In general, restricting the number of tokens on an arc causes deadlock.
Can this program deadlock if the number of tokens per arc is restricted?