CPEG 852 — Advanced Topics in Computing Systems
The Dataflow Model of Computation

Stéphane ZUCKERMAN

Computer Architecture & Parallel Systems Laboratory
Electrical & Computer Engineering Dept.
University of Delaware
140 Evans Hall Newark, DE 19716, United States
szuckerm@udel.edu

September 8, 2015
Outline

1. A Quick Reminder on Execution Models
2. Dataflow Models of Computation
3. Dataflow Graphs and Properties
4. Static Dataflow
   - Introduction
   - Static Dataflow Examples
   - Static Dataflow Features
   - Static Dataflow Activity Templates
5. Recursive Program Graphs
   - Introduction
   - Ordinary and Tail Recursions
   - Features of Recursive Program Graphs
6. Homework
Outline

1 A Quick Reminder on Execution Models
2 Dataflow Models of Computation
3 Dataflow Graphs and Properties
4 Static Dataflow
   • Introduction
   • Static Dataflow Examples
   • Static Dataflow Features
   • Static Dataflow Activity Templates
5 Recursive Program Graphs
   • Introduction
   • Ordinary and Tail Recursions
   • Features of Recursive Program Graphs
6 Homework
High-Level View of a Computer System

Applications

System Software

Hardware
A Representative View of Current Compute Nodes’ Hardware
A Representative View of Current Compute Nodes’ Software Mess

Applications

Compiler

Linker

Libraries

Operating System

RTS

Hardware
Definition: Parallel Model of Computation

- Parallel models for algorithms designers
- Parallel models for system designers
  - Parallel programming models
  - Parallel execution models
  - Parallel architecture models
(Dr. Gao’s) Definition: Program Execution Model

The program execution model (PXM) is the basic low-level abstraction of the underlying system architecture upon which our programming model, compilation strategy, runtime system, and other software components are developed. The PXM (and its API) serves as an interface between the architecture and the software.

Unlike an instruction set architecture (ISA) specification, which usually focuses on lower level details (such as instruction encoding and organization of registers for a specific processor), the PXM refers to machine organization at a higher level for a whole class of high-end machines as viewed by the users.
Horizontal Conception of a PXM

- Applications
- System Software
- Program Execution Model
- Hardware
Vertical Conception of a PXM

Program Execution Model

Applications

System Software

Hardware
Outline

1. A Quick Reminder on Execution Models
2. Dataflow Models of Computation
3. Dataflow Graphs and Properties
4. Static Dataflow
   - Introduction
   - Static Dataflow Examples
   - Static Dataflow Features
   - Static Dataflow Activity Templates
5. Recursive Program Graphs
   - Introduction
   - Ordinary and Tail Recursions
   - Features of Recursive Program Graphs
6. Homework
The Dataflow Model of Computation
A Pragmatic Approach
The Dataflow Model of Computation
A Pragmatic Approach
The Dataflow Model of Computation
A Pragmatic Approach
The Dataflow Model of Computation
A Pragmatic Approach
The Dataflow Model of Computation
A Pragmatic Approach

\[ \begin{align*}
    1 & \\
    3 & \rightarrow \\
    4 & \rightarrow \\
    3 & \rightarrow \\
\end{align*} \]

\[ \begin{align*}
    + & \\
    7 & \rightarrow \\
    7 & \rightarrow \\
\end{align*} \]

\[ \begin{align*}
    28 & \\
\end{align*} \]
For J.B. Dennis, the role of Dataflow Graphs (DFGs) is two-fold:

- Serve as an intermediate-level language for high-level languages
- Serve as a machine language for parallel machine
A Brief History of Dataflow
MIT, 1974–1975


► **August 1975:** Sagamore Computer Conference on Parallel Processing:
  ► Rumbaugh: Data Flow Languages
  ► Rumbaugh: A Data Flow Multiprocessor
  ► Bryant & Dennis: Packet Communication Architecture
  ► Misunas: Structure Processing in a Data-Flow Computer
Early Roots on Dataflow Work
MIT in the 70’s

- Asynchronous Digital Logic (D. E. Muller and Bartky, 1957; David E Muller, 1963)
- Control Structures for Parallel Programming (Conway, 1963; McIlroy, 1969; Dijkstra, 1971)
- Theory of Program Schemes (Ianov, 1958; Paterson and Hewitt, 1970)
- Structured Programming (Dahl, Dijkstra, and Hoare, 1972)
- Functional Programming (McCarthy, 1960; Landin, 1964)
1 A Quick Reminder on Execution Models
2 Dataflow Models of Computation
3 Dataflow Graphs and Properties
4 Static Dataflow
   - Introduction
   - Static Dataflow Examples
   - Static Dataflow Features
   - Static Dataflow Activity Templates
5 Recursive Program Graphs
   - Introduction
   - Ordinary and Tail Recursions
   - Features of Recursive Program Graphs
6 Homework
Dataflow Graphs

**Dataflow tokens**
Values in dataflow graphs are represented as tokens. Tokens are tuples \(<s, d, v>\), where \(s\) is the instruction pointer, \(d\) is the port, and \(v\) represents the actual value (data).

**Operator Execution**
An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators.
**Dataflow Graphs**

```plaintext
x = a + b;
y = b * 7;
z = (x-y) * (x+y);
```

**Dataflow tokens**

Values in dataflow graphs are represented as tokens. Tokens are tuples `<s, d, v>`, where `s` is the instruction pointer, `d` is the port, and `v` represents the actual value (data).

**Operator Execution**

An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators.
Dataflow Graphs

Dataflow tokens
Values in dataflow graphs are represented as tokens. Tokens are tuples `<s, d, v>`, where `s` is the instruction pointer, `d` is the port, and `v` represents the actual value (data).

Operator Execution
An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators.
Overview of Operational Semantics

- Values are represented by tokens
- Tokens are placed (assigned) on the arcs
  - Snapshot/configuration: state
- Computation:
  - Configuration $\rightarrow$ configuration
Details on Operational Semantics

- Tokens ⇔ Data
- Assignment ⇔ Placing a token on the output arc
- Snapshot/configuration: state
- Computation
  - The intermediate step between snapshots / configurations
### Actors States

- An actor in a DFG is **enabled** if there is a token on each of its input arcs.

- Any enabled actor may be **fired** to define the “next state” of the computation.

- When an actor is **fired**, then:
  - All of its input tokens are removed from the input arcs.
  - A token is placed on each of the actor’s output arcs.

- Computation $\Leftrightarrow$ Sequence of snapshots
  - Many possible sequences, as long as firing rules are obeyed.
  - This is called **determinacy**.
  - Provides “locality of effect”.
General Firing Rules

- A switch actor is enabled if:
  - A token is available on its control input arc
  - A token is available on its data input arc(s)

- Firing a switch actor:
  - Removes input tokens
  - Delivers the input data value as an output token on the corresponding output arc

- A (possibly unconditional) merge actor is enabled if
  - There is a token available on any of its input arcs
  - A conditional merge need to have a control token placed on its input control arc

- An enabled (unconditional) merge actor may be fired
  - It will put (possibly non-deterministically) one of the input tokens on the output arc.
Conditional Expressions

```plaintext
if (p(y))
{
  f(x, y);
}
else
{
  g(y);
}
```

**Figure:** if/else construct
Conditional Expressions

Figure: if/else schema
Conditional Expressions

Figure: loop schema
Remarks on Dataflow Models

- The dataflow model of computation is fundamentally sound and simple
  - Very few other parallel MoCs can claim this
- Few dataflow architecture projects survived past the early 1990’s.
  - However, dataflow principles were heavily reused in both software and hardware
- With the advent of the new multi and many core era, we have many (good!) reasons to re-examine and explore the original dataflow models
  - It never hurts to learn from the past!
- As a side-note: Jack Dennis’s dataflow model was finally recognized on a world-scale level: he was awarded the John Von Neumann Medal (the IEEE equivalent to the ACM Turing Award)
Outline

1. A Quick Reminder on Execution Models
2. Dataflow Models of Computation
3. Dataflow Graphs and Properties
4. Static Dataflow
   - Introduction
   - Static Dataflow Examples
   - Static Dataflow Features
   - Static Dataflow Activity Templates
5. Recursive Program Graphs
   - Introduction
   - Ordinary and Tail Recursions
   - Features of Recursive Program Graphs
6. Homework
Static Dataflow

- Static dataflow was the first dataflow model
- It was developed in Jack Dennis’ team
Static Dataflow’s Golden Rule

...for any actor to be enabled, there must be no tokens on any of its output arcs...
Example
Power Function

```c
long power(int x, int n)
{
    int y = 1;
    for (int i = n; i > 0; --i)
        y *= x;
    return y;
}
```

**Figure:** Computation: $y = x^n$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing \( y = 2^3 \)
Example

Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example

Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example

Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example
Power Function – Computing $y = 2^3$
Example

Vector Addition: \( \vec{C}_N = \vec{A}_N + \vec{B}_N \)
Static Dataflow
Features

- One-token-per-arc
- Deterministic merge
- Conditional/iteration construction
- Consecutive iterations of a loop can only be pipelined
- A dataflow graph \( \Rightarrow \) activity templates
  - Opcode of the represented instruction
  - Operand slots for holding operand values
  - Destination address fields
- Token \( \Rightarrow \) value + destination
Static Dataflow
Activity Templates

- Opcode
- Operand / Value
- Next Operand / Result
- Input Address Signal Back

Token Arc
Communication Arc
Op_x / Op_y The operation that produced x and y
next The operation that will use the sqrt result
Static Dataflow
Activity Templates

a * c – b * d

a * d + b * c
The static dataflow model has the advantage of being very simple. However, it is also fairly limited:

- Due to acknowledgment tokens, the token traffic is doubled
- Lack of support for programming constructs that are essential to modern programming languages
- No procedure calls
- No recursion
Outline

1. A Quick Reminder on Execution Models
2. Dataflow Models of Computation
3. Dataflow Graphs and Properties
4. Static Dataflow
   - Introduction
   - Static Dataflow Examples
   - Static Dataflow Features
   - Static Dataflow Activity Templates
5. Recursive Program Graphs
   - Introduction
   - Ordinary and Tail Recursions
   - Features of Recursive Program Graphs
6. Homework
Recursive Program Graphs

- Extension over static dataflow concepts
- Graph must be acyclic (DAG, directed acyclic graph)
- One-token-per-arc-per-invocation
- Iteration is expressed in terms of tail recursion
Examples of Ordinary Recursion

Factorial

```c
long factorial(long n)
{
    if (n == 0)
        return 1;
    else
        return n * fact(n-1);
}
```
Examples of Tail Recursion

Factorial

```c
long fact_tail_rec(long n, long acc)
{
    if (n == 0)
        return acc;
    else
        return fact(n-1, acc*n);
}

long factorial(long n)
{
    return fact_tail_rec(n,1);
}
```
Example – Factorial
Hand Simulation: fact(3)

```c
long factorial(long n) {
    return n == 0 ? 1 : n * fact(n-1);
}
```
Example – Factorial
DFG for fact(3)
Example – Factorial
DFG for fact(3)
Example – Factorial

DFG for fact(3)
Example – Factorial
DFG for fact(3)
Example – Factorial

DFG for fact(3)
Example – Factorial

DFG for fact(3)
Example – Factorial

DFG for fact(3)
Example – Factorial

DFG for fact(3)
Example – Factorial
DFG for fact(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)

Diagram showing the tail recursive version of the factorial function with the numbers 3 and 1 connected through various operations represented by shapes and arrows.
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)

2 calls pending
Apply fact

F
n=0
n
F
T

1

6

×

-1
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)

3 calls pending

Apply fact

-1

×

n

n=0

F

T

F

T

6
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial
Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Factorial

Tail Recursive version – factorial(3)
Recursive Program Graph

Features

- Acyclic
- One-token-per-link-in-lifetime
- Tags
- No deterministic merge needed
- Recursion is expressed by runtime copying
Outline

1 A Quick Reminder on Execution Models
2 Dataflow Models of Computation
3 Dataflow Graphs and Properties
4 Static Dataflow
   - Introduction
   - Static Dataflow Examples
   - Static Dataflow Features
   - Static Dataflow Activity Templates
5 Recursive Program Graphs
   - Introduction
   - Ordinary and Tail Recursions
   - Features of Recursive Program Graphs
6 Homework
Homework

- **factorial function:**
  - Convert the recursive form of `factorial` into an iterative form
  - Create/draw the corresponding static dataflow graph

- **power function:**
  - Convert the iterative form of the `power` function into its recursive form
  - Create/draw its recursive program graph


References II


References III


