Today

- Dataflow Model
- Dataflow Basics
- Examples
- Basic Architecture Requirements
Functional

• What is a functional language?
• What is a functional routine?
• Functional
  – Like a mathematical function
  – Given same inputs, always returns same outputs
  – No state
  – No side effects

Functional:
• F(x) = x * x;
• (define (f x) (* x x))
• int f(int x) { return(x * x); }
Non-Functional

Non-functional:
• (define counter 0)
  (define (next-number!)
   (set! counter (+ counter 1))
   counter)
• static int counter=0;
  int increment () { return(++counter); }

Dataflow

• Model of computation
• Contrast with Control flow
Dataflow / Control Flow

Dataflow
• Program is a graph of operators
• Operator consumes tokens and produces tokens
• All operators run concurrently

Control flow
• Program is a sequence of operations
• Operator reads inputs and writes outputs into common store
• One operator runs at a time
  – Defines successor

Token
• Data value with presence indication
Operator

- Takes in one or more inputs
- Computes on the inputs
- Produces a result

- Logically self-timed
  - “Fires” only when input set present
  - Signals availability of output
Dataflow Graph

- Represents
  - computation sub-blocks
  - linkage
- Abstractly
  - controlled by data presence
Straight-line Code

- Easily constructed into DAG
  - Same DAG saw before
  - No need to linearize

Day4

Dataflow Graph

- Real problem is a graph
Day 4

Task Has Parallelism

MPY R3,R2,R2  MPY R4,R2,R5
MPY R3,R6,R3  ADD R4,R4,R7
ADD R4,R3,R4

DF Exposes Freedom

- Exploit dynamic ordering of data arrival
- Saw aggressive control flow implementations had to exploit
  - Scoreboarding
  - OO issue
Data Dependence

• Add Two Operators
  – Switch
  – Select

Switch

T
F

Select

T
F

Switch

T
F

Switch

T
F

Switch

T
F

Switch

T
F
Select

Constructing If-Then-Else
Looping

- For (i=0; i<Limit; i++)

Dataflow Graph

- Computation itself may construct / unfold parallelism
  - Loops
  - Procedure calls
    - Semantics: create a new subgraph
      - Start as new thread
      - ...procedures unfold as tree / dag
      - Not as a linear stack
    - ...examples shortly...
Key Element of DF Control

- Synchronization on Data Presence
- Constructs:
  - Futures (language level)
  - I-structures (data structure)
  - Full-empty bits (implementation technique)

I-Structure

- Array/object with full-empty bits on each field
- Allocated empty
- Fill in value as compute
- Strict access on empty
  - Queue requester in structure
  - Send value to requester when written and becomes full
I-Structure

- Allows efficient “functional” updates to aggregate structures
- Can pass around pointers to objects
- Preserve ordering/determinacy
- *E.g.* arrays

Future

- **Future** is a promise
- An indication that a value **will be computed**
  - And a handle for getting a handle on it
- Sometimes used as program construct
Future

- Future computation immediately returns a future
- Future is a handle/pointer to result
- (define (vmult a b)
  (cons (future (* (first a) (first b)))
        (dot (rest a) (rest b))))
- [Version for wrighton on next slide]

DF V-Mult product in C/Java

```c
int [] vmult (int [] a, int [] b)
{
    // consistency check on a.length, b.length
    int [] res = new int[a.length];
    for (int i=0;i<res.length;i++)
        future res[i]=a[i]*b[i];
    return (res);
}
```

// assume int [] is an I-Structure
I-Structure V-Mult Example

Two errors in this version:
1) fencepost on value for i
2) Not delivering i to i-store

Corrected dataflow
N.B. this and remainder still have two errors.
I-Structure V-Mult Example

Caltech CS184 Spring 2003 -- DeHon
I-Structure V-Mult Example
I-Structure V-Mult Example

Fib

(define (fib n)
  (if (< n 2) 1 (+ (future (fib (- n 1)))
   (future (fib (- n 2))))))

int fib(int n)
{
    if (n<2)
        return(1);
    else
        return ((future)fib(n-1) + (future)fib(n-2));
}
Fibonacci Example

Fibonacci Example
Fibonacci Example
Fibonacci Example

1. future call fib
2. future call fib

Switch

Fibonacci Example

1. future call fib
2. future call fib

Switch

Switch
Fibonacci Example

[Diagram]

Fibonacci Example

[Diagram]
Fibonacci Example
Futures

• Safe with **functional** routines
  – Create dataflow
  – In functional language, can wrap futures around everything
    • Don’t need explicit future construct
    • Safe to put it anywhere
      – Anywhere compiler deems worthwhile
• Can introduce non-determinacy with side-effecting routines
  – Not clear when operation completes

---

Future/Side-Effect hazard

```lisp
(define (decrement! a b) (set! a (- a b)) a)
(print (* (future (decrement! c d))
         (future (decrement! d e))))
```

```c
int decrement (int &a, int &b)
{ *a=*=a-*b; return(*a);}
printf("%d %d",
      (future)decrement(&c,&d),
      (future)decrement(&d,&e));
```
Architecture Mechanisms?

• Thread spawn
  – Preferably lightweight
• Full/empty bits
• Pure functional dataflow
  – May exploit common namespace
  – Not need memory coherence in pure functional \( \rightarrow \) values never change

Big Ideas

• Model
• Expose Parallelism
  – Can have model that admits parallelism
  – Can have dynamic (hardware) representation with parallelism exposed
• Tolerate latency with parallelism
• Primitives
  – Thread spawn
  – Synchronization: full/empty