Old Review Assignments

- **Were Due: Sunday, October 28, 11:59pm.**

- **Was Due: Tuesday, October 30, 11:59pm.**

- **Were Due: Thursday, November 1, 11:59pm.**
New Review Assignment

- Due: Sunday, November 4, 11:59pm.
Please Finish All Reviews

- Even if you are late...

- The papers are for your benefit and background.

- Discuss them with others in class, with Han, or with me.
Literature Survey Process

- Done in groups: your research project group is likely ideal
- **Step 1:** Pick 3 or more research papers
  - Broadly related to your research project
- **Step 2:** Send me the list of papers with links to pdf copies (by Sunday, November 11)
  - I need to approve the 3 papers
  - We will iterate to ensure convergence on the list
- **Step 3:** Prepare a 2-page writeup on the 3 papers
- **Step 3:** Prepare a 15-minute presentation on the 3 papers
  - Total time: 15-minute talk + 5-minute Q&A
  - Talk should focus on insights and tradeoffs
- **Step 4:** Deliver the presentation in front of class (dates: November 26-28 or December 3-7) and turn in your writeup (due date: December 1)
Literature Survey Guidelines

- The goal is to
  - Understand the solution space and tradeoffs
  - Deeply analyze and synthesize three papers
    - Analyze: Describe individual strengths and weaknesses
    - Synthesize: Find commonalities and common strengths and weaknesses, categorize the solutions with respect to criteria
  - Explain how they relate to your project, how they can enhance it, or why your solution will be better

- Read the papers very carefully
  - Attention to detail is important
The talk should clearly convey at least the following:

- **The problem**: What is the general problem targeted by the papers and what are the specific problems?
- **The solutions**: What are the key ideas and solution approaches of the proposed papers?
- **Key results and insights**: What are the key results, insights, and conclusions of the papers?
- **Tradeoffs and analyses**: How do the solutions differ or interact with each other? Can they be combined? What are the tradeoffs between them? This is where you will need to analyze the approaches and find a way to synthesize a common framework to describe and qualitatively compare & contrast the approaches.
- **Comparison to your project**: How do these approaches relate to your project? Why is your approach novel, different, better, or complementary?
- **Key conclusions and new ideas**: What have you learned? Do you have new ideas/approaches based on what you have learned?
Last Lecture

- Dataflow
Today

- End Dataflow
- Systolic Arrays
Data Flow
Review: Data Flow Characteristics

- Data-driven execution of instruction-level graphical code
  - Nodes are operators
  - Arcs are data (I/O)
  - As opposed to control-driven execution
- Only real dependencies constrain processing
- No sequential I-stream
  - No program counter
- Operations execute asynchronously
- Execution triggered by the presence of data
- Single assignment languages and functional programming
  - E.g., SISAL in Manchester Data Flow Computer
  - No mutable state
Review: Data Flow Advantages/Disadvantages

- **Advantages**
  - Very good at exploiting *irregular parallelism*
  - Only real dependencies constrain processing

- **Disadvantages**
  - Debugging difficult (no precise state)
    - Interrupt/exception handling is difficult (what is precise state semantics?)
  - Implementing dynamic data structures difficult in pure data flow models
  - Too much parallelism? (Parallelism control needed)
  - High bookkeeping overhead (tag matching, data storage)
  - Instruction cycle is inefficient (delay between dependent instructions), memory locality is not exploited
Can we get the best of both worlds?

Two possibilities

- Model 1: Keep control flow at the ISA level, do dataflow underneath, preserving sequential semantics
- Model 2: Keep dataflow model, but incorporate control flow at the ISA level to improve efficiency, exploit locality, and ease resource management

Incorporate threads into dataflow: statically ordered instructions; when the first instruction is fired, the remaining instructions execute without interruption
Model 2 Example: Macro Dataflow

- Data flow execution of large blocks, control flow within a block

Figure 1 An Example of a Strongly Connected Block.

Figure 7. Organization of a macro-dataflow processing element.

Benefits of Control Flow within Data Flow

- Strongly-connected block: Strongly-connected subgraph of the dataflow graph

- Executed without interruption. Atomic: all or none.

- Benefits of the atomic block:
  - Dependent or independent instructions can execute back to back → improved processing element utilization
  - Exploits locality with registers → reduced comm. delay
  - No need for token matching within the block → simpler, less overhead
  - No need for token circulation (which is slow) within the block
  - Easier to implement serialization and critical sections
Macro Dataflow Program Example

(a) A Normal Dataflow Program

FIB:  (COPY <EQ1 imm1> <SW1 r>)
EQ1:  (EQ '1 <SW1 l>)
SW1:  (SWITCH (T <CNST imm1>)
      (F <EQ2 imm2> <SW2 r>)
EQ2:  (EQ '2 <SW2 l>)
SW2:  (SWITCH (T <CNST imm1>)
      (F <SUB1 imm1> <SUB2 imm2>)
CNST: (COPY '1 <RETV r>)
SUB1: (SUB '1 <CALL0 s>)
SUB2: (SUB '2 <CALL1 s>)

CALL0: {*GCALL 'FIB <ADD l>}  CALL0: {*GCALL 'FIB <ADD l>}
CALL1: {*GCALL 'FIB <ADD r>}  CALL1: {*GCALL 'FIB <ADD r>}
ADD:  (ADD <RETV r>)         ADD:  (ADD <RETV r>)
RETV: {*PASSVAL <KILL s>}    RETV: {*PASSVAL <KILL s>}
KILL: {*KILL}               KILL: {*KILL}

(b) A Strongly Connected Dataflow Program

FIB:  [BEQ '1 (LEAF) (R0:NL)]
       [SUB '1 R1 (R0:NL) (NL:NL $)]
       [BEQ '2 (LEAF) (R0:NL)]
       [SUB '2 R2 (SEG:R1) (NL:NL $)]
       [MKPCT <CALL0 s> (SEG:R2 $)]
       [MKPCT <CALL1 s> (NL:NL)]
LEAF:  [MKPCT '1 <RETV r> (NL:NL)]

Figure 5 FIBONACCI Program.
Macro Dataflow Machine Example

**Figure 6** Block Diagram of the EMC-R.

**Figure 8** Execution Unit Organization.
Macro Dataflow Pipeline Organization

Figure 9 Pipeline Organization of the EMC-R.
Model 1 Example: Restricted Data Flow

- Data flow execution under sequential semantics and precise exceptions

Restricted Data Flow DFG Formation

An Example from the VAX:

ADDL 3 #1000, A, B

Decoder

Figure 3.
Systolic Arrays
Why Systolic Architectures?

- **Idea:** Data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory.

- Similar to an assembly line
  - Different people work on the same car
  - Many cars are assembled simultaneously
  - Can be two-dimensional

- **Why?** Special purpose accelerators/architectures need
  - Simple, regular designs (keep # unique parts small and regular)
  - High concurrency $\rightarrow$ high performance
  - Balanced computation and I/O (memory access)
Systolic Architectures


![Diagram of systolic array]

Memory: heart
PEs: cells

Memory pulses data through cells

**Figure 1. Basic principle of a systolic system.**
Systolic Architectures

- Basic principle: Replace a single PE with a regular array of PEs and carefully orchestrate flow of data between the PEs → achieve high throughput w/o increasing memory bandwidth requirements

- Differences from pipelining:
  - Array structure can be non-linear and multi-dimensional
  - PE connections can be multidirectional (and different speed)
  - PEs can have local memory and execute kernels (rather than a piece of the instruction)
Convolution

- Used in filtering, pattern matching, correlation, polynomial evaluation, etc ...
- Many image processing tasks

Given the sequence of weights \( \{w_1, w_2, \ldots, w_k\} \) and the input sequence \( \{x_1, x_2, \ldots, x_n\} \), compute the result sequence \( \{y_1, y_2, \ldots, y_{n+1-k}\} \) defined by

\[
y_i = w_1x_i + w_2x_{i+1} + \ldots + w_kx_{i+k-1}
\]
Systolic Computation Example: Convolution

- $y_1 = w_1x_1 + w_2x_2 + w_3x_3$
- $y_2 = w_1x_2 + w_2x_3 + w_3x_4$
- $y_3 = w_1x_3 + w_2x_4 + w_3x_5$

Figure 8. Design W1: systolic convolution array (a) and cell (b) where $w_i$’s stay and $x_i$’s and $y_i$’s move systolically in opposite directions.
Worthwhile to implement adder and multiplier separately to allow overlapping of add/mul executions
Each PE in a systolic array
- Can store multiple “weights”
- Weights can be selected on the fly
- Eases implementation of, e.g., adaptive filtering

Taken further
- Each PE can have its own data and instruction memory
- Data memory → to store partial/temporary results, constants
- Leads to stream processing, pipeline parallelism
  - More generally, staged execution
Pipeline Parallelism

Figure 1. (a) The code of a loop, (b) Each iteration is split into 3 pipeline stages: A, B, and C. Iteration i comprises Ai, Bi, Ci. (c) Sequential execution of 4 iterations. (d) Parallel execution of 6 iterations using pipeline parallelism on a three-core machine. Each stage executes on one core.
Figure 3. File compression algorithm executed using pipeline parallelism
Systolic Array

- Advantages
  - Makes multiple uses of each data item → reduced need for fetching/refetching
  - High concurrency
  - Regular design (both data and control flow)

- Disadvantages
  - Not good at exploiting irregular parallelism
  - Relatively special purpose → need software, programmer support to be a general purpose model
The WARP Computer

- HT Kung, CMU, 1984-1988
- Linear array of 10 cells, each cell a 10 Mflop programmable processor
- Attached to a general purpose host machine
- HLL and optimizing compiler to program the systolic array
- Used extensively to accelerate vision and robotics tasks

The WARP Computer

Figure 1: Warp system overview
The WARP Computer

Figure 2: Warp cell data path
Systolic Arrays vs. SIMD

- Food for thought...