CSCI 322
Principles of Concurrent Programming

Filip Jagodzinski
Announcements

Three articles have been posted to the Files section of Canvas. Select one and:

• Write a 1 paragraph summary
• Write a 1 paragraph critique
Announcements

Three articles have been posted to the Files section of Canvas. Select one and:

- Write a 1 paragraph summary
- Write a 1 paragraph critique

- Written in your own words (no cutting and pasting).
- A good rule of thumb is to write the summary WITHOUT looking at the original article.
- Do not include personal “opinions.” And do not write, “I found this interesting because ...”
Announcements

Three articles have been posted to the Files section of Canvas. Select one and:

- **Write a 1 paragraph summary**
- **Write a 1 paragraph critique**

- Written in your own words (no cutting and pasting).
- A good rule of thumb is to write the summary **WITHOUT** looking at the original article.
- Do not include personal “opinions.” And do not write, “I found this interesting because ...”

---

On the Design of High Performance Digital Arithmetic Units by Paul Michael Farmwald analyzes the efficiency of high-performance parallel pipeline architecture within the S-1 Mark II A uniprocessor. Farmwald created the Mark II A uniprocessor to supplement their PhD dissertation on how to design a high-performance arithmetic unit that focuses on using current components more efficiently rather than to utilize newer and more expensive components. They also derive new algorithms based on Quicksort on a parallel pipeline system to enhance pipelined CPUs. The result was a more efficient and cheaper architecture that is able to compete with more advanced non-pipelined hardware.
Three articles have been posted to the Files section of Canvas. Select one and:

- Write a 1 paragraph summary
- Write a 1 paragraph critique
- Written in your own words (no cutting and pasting).
- A good rule of thumb is to write the summary WITHOUT looking at the original article.
- Do not include personal “opinions.” And do not write, “I found this interesting because ...”

For example:

*On the Design of High Performance Digital Arithmetic Units* by Paul Michael Farmwald analyzes the efficiency of high-performance parallel pipeline architecture within the S-1 Mark II/A uniprocessor. Farmwald created the Mark II/A uniprocessor to supplement their PhD dissertation on how to design a high-performance arithmetic unit that focuses on using current components more efficiently rather than to utilize newer and more expensive components. They also derive new algorithms based on Quicksort on a parallel pipeline system to enhance pipelined CPUs. The result was a more efficient and cheaper architecture that is able to compete with more advanced non-pipelined hardware.

- How
- Why
- What
Announcements

Three articles have been posted to the Files section of Canvas. Select one and:

- Write a 1 paragraph summary
- Write a 1 paragraph critique
- Usually a bit more difficult to write than a summary
Announcements

Three articles have been posted to the Files section of Canvas. Select one and:

• Write a 1 paragraph summary
• Write a 1 paragraph critique

• Usually a bit more difficult to write than a summary
• Should be constructive. Don’t say, “This was a bad article.” Instead, write “The article was difficult to read because _____,” or “Figure 1.4 was confusing and would be improved by including a more detailed caption.”
Announcements

Three articles have been posted to the Files section of Canvas. Select one and:

- Write a 1 paragraph summary
- Write a 1 paragraph critique

- Usually a bit more difficult to write than a summary
- Should be constructive. Don’t say, “This was a bad article.” Instead, write “The article was difficult to read because _____,” or “Figure 1.4 was confusing and would be improved by including a more detailed caption.”

**Critique points that don’t require extensive domain knowledge**

- Is the abstract and introduction of the article accessible to a non-specialist audience?
- Are figures, plots, tables, etc. clear, and are they appropriate? Do they help to better explain the article’s main goals?
- Are there lapses in logic?
- Are the assumptions (if any) valid?
- Are the conclusions (if any) valid?
- Is the work placed into the context of existing work?
- Is there a discussion about improvements and shortcomings?
Announcements

Expect to spend up to an hour reading (and re-reading portions of) the article, and no more than 30 minutes writing the summary and critique.

My suggestions:

- Read the abstract
- Read the abstract again
- Read the introduction
- Read the abstract again
- Read the introduction again
- Skim the methods section
- Skim the results section
- Read the conclusions section
- Read the abstract
- Go get a coffee
- Write a summary and critique
Announcements

Article summary/critique (2 paragraphs)

• **Summary**: written in your own words, no cutting and pasting allowed, is void of personal opinions, explains the **How**, **Why**, and **What** of the science/research being discussed.

• **Critique**: Should be constructive. Possible critique points that don’t require extensive domain knowledge:
  • Is abstract and introduction accessible to non-specialist?
  • Are figures, plots, tables, etc. clear and beneficial?
  • Are there lapses in logic
  • Are the assumptions and conclusions valid?
  • Etc.

Samples of Good Constructive Critique Statements

• Little was provided in the article to substantiate the claim
• No mention is given to alternatives
• The article was not written for a person without a background in linear algebra
• ...[they should have] waited until LAPACK was in a more finished state
• [Explain] how these systems related to the current standard of supercomputers
From last time

Prove: $A \Rightarrow (B \Rightarrow A)$

1. $\neg (A \Rightarrow (B \Rightarrow A))$
2. ...
3. $Z$
4. ...
5. $\neg Z$
6. $A \Rightarrow (B \Rightarrow A)$

Task: Understand that steps in the above proof.
From last time

We want to be able to reason logically about our programming languages, PL

Axiomatic Semantics

\{P\} \text{ S } \{Q\}

• Called a \textbf{triple}
• If \(P\) is true before \(S\) is executed, and \(S\) terminates, then \(Q\) is true after \(S\) executes
• \(P\) and \(Q\) are assertions
• \(P\) is the pre-condition
• \(Q\) is the post-condition

Assignment Axiom

\{P_{x \leftarrow c}\} x = e \{P\}

\{1 = 1\} x = 1 \{x = 1\}
\{true\} x = 1 \{x = 1\}

\{x + 1 = 1\} x = x + 1 \{x = 1\}
\{x = 0\} x = x + 1 \{x = 1\}

\{x + 1 = n\} x = x + 1 \{x = n\}
\{x = n - 1\} x = x + 1 \{x = n\}
From last time

Just as we had inference rules in “logic”, we have inference rules for our PL

**Composition Rule:** \[ \frac{\{P\} S_1 \{Q\}, \{Q\} S_2 \{R\}}{\{P\} S_1 ; S_2 \{R\}} \]

**If Statement Rule:** \[ \frac{\{P \land B\} S \{Q\}, (P \land \neg B) \Rightarrow Q}{\{P\} \text{ if } (B) S; \{Q\}} \]

**While Statement Rule:** \[ \frac{\{I \land B\} S \{I\}}{\{I\} \text{ while}(B) S; \{I \land \neg B\}} \]

**Rule of Consequence:** \[ \frac{P' \Rightarrow P, \{P\} S \{Q\}, Q \Rightarrow Q'}{\{P'\} S \{Q'\}} \]
Today

Sample proof
Code optimization
Proof Strategies

Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C'))\)

• Conditional Proof
• Proof by Contradiction
Proof Strategies

Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C'))\)

Assume that you select conditional proof approach ...

Write the premise (hypothesis) at the top, and conclusion at the bottom
Proof Strategies

Assume that you select conditional proof approach ...

Write the premise (hypothesis) at the top, and conclusion at the bottom

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

What goes here is the creative part
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

A sub-proof ... hence another box

What goes here is the creative part

Q: What rule/axiom might we apply?
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

- **Modus Ponens**
  \[
  \frac{A \Rightarrow B, A}{B}
  \]

A sub-proof ... hence another box

What goes here is the creative part
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

- Modus Ponens
  \[
  \frac{A \Rightarrow B, A}{B}
  \]

Application of modus ponens

What goes in here such that given what is “above” \((A\) and \(C)\) we can logically claim that what is below \((B \lor C)\) is also true.
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

- Modus Ponens
  \[
  \begin{align*}
  A & \Rightarrow B, A \\
  \hline
  B 
  \end{align*}
  \]

- Addition
  \[
  \begin{align*}
  A & \\
  \hline
  A \lor B \\
  A \Rightarrow (B \lor C) 
  \end{align*}
  \]

Application of modus ponens

What goes in here such that given what is “above” (A and C) we can logically claim that what is below (B OR C) is also true.
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

Assumption

Assumption

Modus Ponens

Addition
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

Assumption
- Assumption
- Modus Ponens
- Addition

Q: Is there another way to prove this?
Proof Strategies

Proof: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

What goes here is the creative part

Task: Proceed by using a sub proof utilizing proof by contradiction
Proof Strategies

Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C))\)

Task: Proceed by using a sub proof utilizing proof by contradiction
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

You are allowed to attempt to contradict a “future” statement

Task : Proceed by using a sub proof utilizing proof by contradiction
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

Q: Which of the existing axioms can we use here?

13. \(A \Rightarrow false \equiv \neg A\)
14. \(true \Rightarrow A \equiv A\)
15. \(false \Rightarrow A \equiv true\)
16. \(A \Rightarrow A \equiv true\)
17. \(A \Rightarrow B \equiv \neg A \lor B\)
18. \(A \Rightarrow B \equiv \neg B \Rightarrow \neg A\)
19. \(\neg(A \Rightarrow B) \equiv A \land \neg B\)
20. \(A \land (A \lor B) \equiv A\)
21. \(A \lor (A \land B) \equiv A\)
22. \(A \land (\neg A \lor B) \equiv A \land B\)
23. \(A \lor (\neg A \land B) \equiv A \lor B\)
24. \(\neg(A \land B) \equiv \neg A \lor \neg B\)
25. \(\neg(A \lor B) \equiv \neg A \land \neg B\)
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

\[ A \Rightarrow C \]

\[ A \]

\[ \neg(B \lor C) \]

\[ ... \]

\[ B \lor C \]

\[ A \Rightarrow (B \lor C) \]

Q: Which of the existing axioms can we use here?

\[ \neg(A \lor B) \equiv \neg A \land \neg B \]

Q: What is the result of applying this equivalence?
Q: What rule/axiom can we apply next?
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

Recall that a proof can have as many lines as are needed, as long as each “line” is either an assumption, application of a rule or axiom, or a TRUE statement.

Q: What can be done with this AND?
Proof Strategies

Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C'))\)

Q: Which rule should we apply next?
Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C'))\)

Q: Which rule should we apply next?

Q: How should this rule be applied?

- Modus Ponens

\[
\begin{align*}
A \Rightarrow B, A \\
\hline
B
\end{align*}
\]
Proof Strategies

Prove: \( (A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C)) \)

Q: Are we done?
Proof Strategies

Prove: \((A \Rightarrow C) \Rightarrow (A \Rightarrow (B \lor C))\)

Q: Are we done?

We’ve shown a contradiction
Proof Strategies

Q: Are we done?

We’ve shown a contradiction

Therefore the premise is FALSE

Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C'))\)
Proof Strategies

Prove: \((A \Rightarrow C') \Rightarrow (A \Rightarrow (B \lor C'))\)

Q: Are we done?

We’ve shown a contradiction

Therefore the premise is FALSE

Thus the converse is true
Because you’ve noticed that creating a concurrent threaded program doesn’t necessarily speed up the run-time during execution ... what else can you do?
Because you’ve noticed that creating a concurrent threaded program doesn’t necessarily speed up the run-time during execution ... what else can you do?

Q: Why?
Because you’ve noticed that creating a concurrent threaded program doesn’t necessarily speed up the run-time during execution ... what else can you do?
Because you’ve noticed that creating a concurrent threaded program doesn’t necessarily speed up the run-time during execution … what else can you do?

Write better code!
First, some terminology
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S_1 & \quad x = 3 + i \\
S_2 & \quad y = x
\end{align*}
\]
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

S1  \( x = 3 + i \)

S2  \( y = x \)

Q: How is/are S1 and S2 dependent on each other?
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

Q: How is/are $S_1$ and $S_2$ dependent on each other?

$S_2$ is flow dependent on $S_1$.

$S_1 \quad x = 3+i$

$S_2 \quad y = x$
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

Task: Identify all of the flow dependencies in the above code.
Task: Identify all of the flow dependencies in the above code.

- S4 is flow dependent on S3
- S5 is flow dependent on S4 and also (indirectly) on S3
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad y = x \\
S3 & \quad x = 3 + i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 & \quad i = x \\
S2 & \quad x = 324
\end{align*}
\]
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 &\quad x = 3+i \\
S2 &\quad y = x
\end{align*}
\]

\[
\begin{align*}
S3 &\quad x = 3+i \\
S4 &\quad y = x \\
S5 &\quad z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 &\quad i = x \\
S2 &\quad x = 324
\end{align*}
\]

Q: How is/are S1 and S2 above dependent on each other?
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad y = x
\end{align*}
\]

Q: How is/are S1 and S2 above dependent on each other?

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 & \quad i = x \\
S2 & \quad x = 324
\end{align*}
\]

S2 is not flow dependent on S1, but re-ordering S2 and S1 would affect the final value of i.
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{array}{c}
S1 & x = 3+i \\
S2 & y = x \\
S3 & x = 3+i \\
S4 & y = x \\
S5 & z = y
\end{array}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{array}{c}
S1 & i = x \\
S2 & x = 324 \\
S3 & x = 42 \\
S4 & y = x + 17 \\
S5 & x = 6
\end{array}
\]

Task: Identify all of the anti dependencies in the above code.
Code Optimization

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad y = x \\
S3 & \quad x = 3 + i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 & \quad i = x \\
S2 & \quad x = 324 \\
S3 & \quad x = 42 \\
S4 & \quad y = x + 17 \\
S5 & \quad x = 6
\end{align*}
\]

**Task**: Identify all of the anti dependencies in the above code.

S5 anti-depends on S4, because changing their ordering would affect the value of y.
Code Optimization

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x = 3+i$</td>
<td>$y = x$</td>
<td>$x = 3+i$</td>
<td>$y = x$</td>
<td>$z = y$</td>
</tr>
</tbody>
</table>

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$i = x$</td>
<td>$x = 324$</td>
<td>$x = 42$</td>
<td>$y = x + 17$</td>
<td>$x = 6$</td>
</tr>
</tbody>
</table>

Q: Why are flow and anti-dependencies of special concern in concurrent programming?
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

| S1  | i = x       |
| S2  | x = 324    |
| S3  | x = 42     |
| S4  | y = x + 17 |
| S5  | x = 6      |

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

| S1  | x = 3 + i   |
| S2  | y = x       |
| S3  | x = 3 + i   |
| S4  | y = x       |
| S5  | z = y       |

Q: How can we eliminate anti dependence (between S4 and S5) from this code?

Q: How would you rewrite this code?
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

| S1 | i = x     |
| S2 | x = 324   |
| S3 | x = 42    |
| S4 | y = x + 17|
| S5 | x = 6     |

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

| S1 | x = 3+i  |
| S2 | y = x    |
| S3 | x = 3+i  |
| S4 | y = x    |
| S5 | z = y    |

Q: Does it matter if S4 or S5 goes first? (on the board “proof” by enumeration)

Notice that “fixing” the anti dependence between S4 and S5 has introduced another one (S4 is now anti-dependent on S6)
Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

Q: How is/are S1 and S2 output dependent?
Q: How is/are S1 and S2 output dependent?

S2 is output dependent on S1
(S2 is also flow dependent on S1)
Code Optimization

**Output dependence**, Write-after-write (WAW): when the ordering of instructions affects the final output value.

- **S1** \( x = 3 + i \)
- **S2** \( x = x \times 2 \)
- **S3** \( x = 3 + i \)
- **S4** \( y = x / 4 \)
- **S5** \( x = 32 \)

**Task:** Identify the output dependency in the above code.
Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

\[
\begin{align*}
S1 & : x = 3 + i \\
S2 & : x = x \times 2
\end{align*}
\]

\[
\begin{align*}
S3 & : x = 3 + i \\
S4 & : y = x / 4 \\
S5 & : x = 32
\end{align*}
\]

Task: Identify the output dependency in the above code.

S5 is output dependent on S3.

Q: What other dependencies exist among the statements S3, S4 and S5?
Code Optimization

Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

Task: Identify the output dependency in the above code.

S5 is output dependent on S3.

Q: What other dependencies exist among the statements S3, S4 and S5?

Task: Rewrite the above right code to eliminate the output dependency between S3 and S5.
Code Optimization

Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

S1: \( x = 3 + i \)
S2: \( x = x \times 2 \)

S3: \( x = 3 + i \)
S4: \( y = x / 4 \)
S5: \( x = 32 \)

Q: Does it matter if S3 or S5 goes first?
(on the board “proof” by enumeration)

Rename and/or copy variables:
Code Optimization

**Control Dependence** : an instruction is control dependent on a preceding instruction if the output of the latter *can* determine if the former should be executed

```
S6  if (x == y)
S7  x = x + y
S8  y = x + y
```

Task : Identify the control dependency in the above code
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Control dependence is a bit more subtle ... you must also inspect the “logic” of the code in addition to noticing the pattern.
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

```
c = [3, 4, 5, 7, 4, 3]
a = [-2, -3, 3, 4, 5, 6]
for (i=0; i<n; i++){
    a[i] = c[i];
    if (a[i] < 0){
        a[i] = 1;
    }
}
```

```
a = [-2, -3, 3, 4, 5, 6]
for (i=1; i<n; i++){
    if (a[i-1] < 0){
        a[i] = -1;
    }
}
```
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

```plaintext
c = [3, 4, 5, 7, 4, 3]
a = [-2, -3, 3, 4, 5, 6]
for (i=0; i<n; i++){
a[i] = c[i];
if (a[i] < 0){
a[i] = 1;
}
}
```

To answer this question, “Execute” the code:

```
s6 if (x == y)
s7 x = x + y
s8 y = x + y
```
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Although here the $a[i]$ appears to influence the if condition, because of the logic of the program, will the if’s code block EVERY influence the Boolean expression?

Q: Which of these is an example of control dependence?

To answer this question, “Execute” the code:

```java
int[] c = {3, 4, 5, 7, 4, 3};
int[] a = {-2, -3, 3, 4, 5, 6};
for (int i = 0; i < c.length; i++) {
    int ai = c[i];
    if (ai < 0) {
        a[i] = 1;
    }
}
```
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter \textbf{can} determine if the former should be executed.

Q: Which of these is an example of control dependence?

\begin{verbatim}
c = [ 3, 4, 5, 7, 4, 3]
a = [-2, -3, 3, 4, 5, 6]
for (i=0; i<n; i++){
    a[i] = c[i];
    if (a[i] < 0){
        a[i] = 1;
    }
}
\end{verbatim}

\begin{verbatim}
a = [-2, -3, 3, 4, 5, 6]
for (i=1; i<n; i++){
    if (a[i-1] < 0){
        a[i] = -1;
    }
}
\end{verbatim}

To answer this question, “Execute” the code.

Here the updating of the \texttt{a[i]} \textbf{WILL} affect the evaluation of the Boolean expression in subsequent iterations of the for loop.
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

No control dependence:

```c
void
for (i=0; i<n; i++)
    if (a[i] < 0)
        a[i] = 1;
```

Control dependence:

```c
void
for (i=1; i<n; i++)
    if (a[i-1] < 0)
        a[i] = -1;
```
For which 2 ranges of $i$ sent to two threads does there exist a history that exhibits control dependence when the threads are executed concurrently?

```c
a = [ -2, -3, 3, 4, 5, 6 ]
for (i=1; i<n; i++){
    if (a[i-1] < 0){
        a[i] = -1;
    }
}
```
Big picture: Reordering, copying of, and renaming of variables in “intro” courses is irrelevant because those programs are executed sequentially (top-down). But concurrently executed programs may interweave instructions in different orders, which DOES affect a program’s final state.

Q: Just because we CAN optimize code, should we?
Up Next

Code Optimization