Lecture 11—Dependencies

January 28, 2015

Roadmap

Last Time: C++ atomics; C Compilers
This Time: Dependencies

Atomics when not using C++11

Not really.

gcc supports atomics via extensions:

`https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html`

OS X has atomics via OS calls:

`https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Multithreading/ThreadSafety/ThreadSafety.html`

etc. . .

Reference:

`http://stackoverflow.com/questions/1130018/unix-portable-atomic-operations`

Part I

Dependencies

Next topic: Dependencies

Dependencies are the main limitation to parallelization.

Example: computation must be evaulated as `XY` and not `YX`.

Not synchronization

Assume (for now) no synchronization problems.

Only trying to identify code that is safe to run in parallel.

Memory-carried Dependencies

Dependencies limit the amount of parallelization.

Can we execute these 2 lines in parallel?

```
x = 42
x = x + 1
```

Memory-carried Dependencies

Dependencies limit the amount of parallelization.

Can we execute these 2 lines in parallel?

```
x = 42
x = x + 1
```

No.

- Assume x initially 1. What are possible outcomes?

Memory-carried Dependencies

Dependencies limit the amount of parallelization.

Can we execute these 2 lines in parallel?

```
x = 42
x = x + 1
```

No.

- Assume x initially 1. What are possible outcomes?
  $x = 43$ or $x = 42$

Next, we'll classify dependencies.

Read After Read (RAR)

Can we execute these 2 lines in parallel? (initially x is 2)

```
y = x + 1
z = x + 5
```

Read After Read (RAR)

Can we execute these 2 lines in parallel? (initially x is 2)

```
y = x + 1
z = x + 5
```

Yes.

- Variables y and z are independent.
- Variable x is only read.

RAR dependency allows parallelization.

Read After Write (RAW)

What about these 2 lines? (again, initially x is 2):

```
x = 37
z = x + 5
```

Read After Write (RAW)

What about these 2 lines? (again, initially x is 2):

```
x = 37
z = x + 5
```

No, z = 42 or z = 7.

RAW inhibits parallelization: can't change ordering.
Also known as a true dependency.

Write After Read (WAR)

What if we change the order now? (again, initially x is 2)

```
z = x + 5
x = 37
```

Write After Read (WAR)

What if we change the order now? (again, initially x is 2)

```
z = x + 5
x = 37
```

No. Again, $z = 42$ or $z = 7$.

- WAR is also known as a anti-dependency.
- But, we can modify this code to enable parallelization.

Removing Write After Read (WAR) Dependencies

Make a copy of the variable:

```
x_copy = x
z = x_copy + 5
x = 37
```

Removing Write After Read (WAR) Dependencies

Make a copy of the variable:

```
x_copy = x
z = x_copy + 5
x = 37
```

We can now run the last 2 lines in parallel.

- Induced a true dependency (RAW) between first 2 lines.
- Isn't that bad?

Removing Write After Read (WAR) Dependencies

Make a copy of the variable:

```
x_copy = x
z = x_copy + 5
x = 37
```

We can now run the last 2 lines in parallel.

- Induced a true dependency (RAW) between first 2 lines.
- Isn't that bad?

Not always:

```
z = very_long_function(x) + 5
x = very_long_calculation()
```

Write After Write (WAW)

Can we run these lines in parallel? (initially x is 2)

```
z = x + 5
z = x + 40
```

Write After Write (WAW)

Can we run these lines in parallel? (initially x is 2)

```
z = x + 5
z = x + 40
```

Nope, $z = 42$ or $z = 7$.

- WAW is also known as an output dependency.
- We can remove this dependency (like WAR):

Write After Write (WAW)

Can we run these lines in parallel? (initially x is 2)

```
z = x + 5
z = x + 40
```

Nope, $z = 42$ or $z = 7$.

- WAW is also known as an output dependency.
- We can remove this dependency (like WAR):

```
z_copy = x + 5
z = x + 40
```

Summary of Memory-carried Dependencies

		Second Access	
		Read	**Write**
First Access	**Read**	No Dependency Read After Read (RAR)	Anti-dependency Write After Read (WAR)
	Write	True Dependency Read After Write (RAW)	Output Dependency Write After Write (WAW)