Lecture 06 - Dependencies

ECE 459: Programming for Performance

Jon Eyolfson

University of Waterloo

January 16, 2012

Previously

- We saw race conditions and how to remedy them with synchronization
- I forgot to mention barriers too, useful if you want threads to wait at a certain point in execution for *x* other threads to finish
- `pthread_barrier_t`, with `init` (takes as a parameter how many threads it should wait for) and `destroy`
- Also has `wait` which is similar to a `join` that will wait for the specified number of threads to arrive at the barrier

Today

- I talked before about dependencies being the main limitation to parallelization
- Basically, when a computation has to be evaulated as `XY` instead of `YX`
- We are just going to assume there is no synchronization problems for these examples (although they exist too)
- Only trying to identify code that is safe to run in parallel

Memory-carried Dependencies

- Dependencies limit the amount of parallelization in a program

Can we execute these 2 lines in parallel?

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

Memory-carried Dependencies

- Dependencies limit the amount of parallelization in a program

Can we execute these 2 lines in parallel?

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

No

- What are the possible outcomes? (x is initially 1)

Memory-carried Dependencies

- Dependencies limit the amount of parallelization in a program

Can we execute these 2 lines in parallel?

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

No

- What are the possible outcomes? (x is initially 1)
  x = 43 or x = 42

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

- The variables y and z are independent
- Variable x is only read

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

- We cannot change the order
- Also known as a true dependency

Write After Read (WAR)

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

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

Write After Read (WAR)

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

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

No, again, z = 42 or z = 7

- Also known as a anti-dependency
- We can modify the code to run these lines in parallel

Removing Write After Read (WAR) Dependency

Make a copy of the variable

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

Removing Write After Read (WAR) Dependency

Make a copy of the variable

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

We can run the 2 lines in parallel now

- There is now true dependency (RAW) between the 2 lines
- Why is this useful?

Removing Write After Read (WAR) Dependency

Make a copy of the variable

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

We can run the 2 lines in parallel now

- There is now true dependency (RAW) between the 2 lines
- Why is this useful?

```
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 $z = 7$

- Also known as a output dependency
- We may remove this dependency (similar to 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

- Also known as a output dependency
- We may remove this dependency (similar to 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)

Loop-carried Dependencies (1)

Can we run these lines in parallel? (initially a[0] and a[1] are 1)

```
a[4] = a[0] + 1
a[5] = a[1] + 2
```

Loop-carried Dependencies (1)

Can we run these lines in parallel? (initially a[0] and a[1] are 1)

```
a[4] = a[0] + 1
a[5] = a[1] + 2
```

Yes

- There are no dependencies between these lines
- However, this is not how we normally use arrays...

Loop-carried Dependencies (2)

What about this? (all elements are initially 1)

```
for (int i = 1; i < 12; ++i)
    a[i] = a[i-1] + 1
```

Loop-carried Dependencies (2)

What about this? (all elements are initially 1)

```
for (int i = 1; i < 12; ++i)
    a[i] = a[i-1] + 1
```

No, a[2] = 3 or a[2] = 2

- Statements are dependent on the previous iteration of the loop
- This is an example of a loop-carried dependency

Loop-carried Dependencies (3)

Can we parallelize this loop? (again, all elements are initially 1)

```
for (int i = 4; i < 12; ++i)
    a[i] = a[i-4] + 1
```

Loop-carried Dependencies (3)

Can we parallelize this loop? (again, all elements are initially 1)

```
for (int i = 4; i < 12; ++i)
    a[i] = a[i-4] + 1
```

Yes, to a degree

- We can execute 4 statements in parallel
  - a[4] = a[0] + 1, a[8] = a[4] + 1
  - a[5] = a[1] + 1, a[9] = a[5] + 1
  - a[6] = a[2] + 1, a[10] = a[6] + 1
  - a[7] = a[3] + 1, a[11] = a[7] + 1

Loop-carried Dependencies (3)

Can we parallelize this loop? (again, all elements are initially 1)

```
for (int i = 4; i < 12; ++i)
    a[i] = a[i-4] + 1
```

Yes, to a degree

- We can execute 4 statements in parallel
  - a[4] = a[0] + 1, a[8] = a[4] + 1
  - a[5] = a[1] + 1, a[9] = a[5] + 1
  - a[6] = a[2] + 1, a[10] = a[6] + 1
  - a[7] = a[3] + 1, a[11] = a[7] + 1

Always consider the dependencies between iterations

Summary

- Identify memory-carried dependencies
  - 3 types of dependencies (RAW, WAR, WAW)
- How to remove output and anti-dependencies
- Identify loop-carried dependencies
  - Explain dependencies between iterations