Concurrency

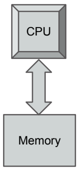
Notes for CSC 100 - The Beauty and Joy of Computing
The University of North Carolina at Greensboro

What is concurrency?

From Wikipedia page Concurrency (computer science):

In computer science, ***concurrency*** *is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other. The computations may be executing on multiple cores in the same chip, preemptively time-shared threads on the same processor, or executed on physically separated processors.*

Traditional computer model



One Central Processing Unit (CPU)

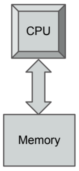
Basic Property: Can do *one task (or thread) at a time*

What if want to do multiple tasks?

- Web browser running
- Movie player running
- MS Word running to take notes
- Checking for instant messages
- .....

Question: How do we do all these things simultaneously?

Traditional computer model

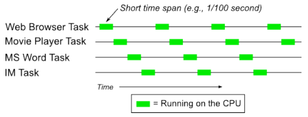


One Central Processing Unit (CPU)

Basic Property: Can do *one task at a time*

Answer: Time-Division Multiplexing

Switch back and forth so quickly that it looks like things are happening simultaneously.


Short time span (e.g., 1/100 second)

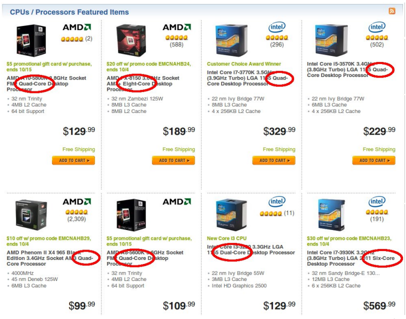
Web Browser Task
Movie Player Task
MS Word Task
IM Task

Time

🟩 = Running on the CPU

If that's "traditional", what's current?

What are some common CPUs on the market?


CPUs / Processors Featured Items

AMD (2)
$5 promotional gift card w/ purchase, ends 10/15
AMD A10-5800K 3.8GHz Socket FM2 Quad-Core Desktop Processor
- 32 nm Trinity
- 4MB L2 Cache
- 64 bit Support

$129.99
Free Shipping
ADD TO CART ▸

AMD (588)
$20 off w/ promo code EMCNAHB24, ends 10/4
AMD FX-8150 3.6GHz Socket AM3+ Eight-Core Desktop Processor
- 32 nm Zambezi 125W
- 8MB L3 Cache
- 8MB L2 Cache

$189.99
Free Shipping
ADD TO CART ▸

intel (296)
Customer Choice Award Winner
Intel Core i7-3770K 3.5GHz (3.9GHz Turbo) LGA 1155 Quad-Core Desktop Processor
- 22 nm Ivy Bridge 77W
- 8MB L3 Cache
- 4 x 256KB L2 Cache

$329.99
Free Shipping
ADD TO CART ▸

intel (502)
Intel Core i5-3570K 3.4GHz (3.8GHz Turbo) LGA 1155 Quad-Core Desktop Processor
- 22 nm Ivy Bridge 77W
- 6MB L3 Cache
- 4 x 256KB L2 Cache

$229.99
Free Shipping
ADD TO CART ▸

AMD (2,309)
$10 off w/ promo code EMCNAHB29, ends 10/4
AMD Phenom II X4 965 Black Edition 3.4GHz Socket AM3 Quad-Core Processor
- 4000MHz
- 45 nm Deneb 125W
- 6MB L3 Cache

$99.99

AMD
$5 promotional gift card w/ purchase, ends 10/15
AMD A8-5600K 3.6GHz Socket FM2 Quad-Core Desktop Processor
- 32 nm Trinity
- 4MB L2 Cache
- 64 bit Support

$109.99

intel (11)
New Core i3 CPU
Intel Core i3-3220 3.3GHz LGA 1155 Dual-Core Desktop Processor
- 22 nm Ivy Bridge 55W
- 3MB L3 Cache
- Intel HD Graphics 2500

$129.99

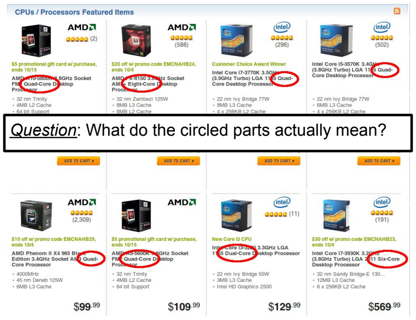
intel (191)
$30 off w/ promo code EMCNAHB23, ends 10/4
Intel Core i7-3930K 3.2GHz (3.8GHz Turbo) LGA 2011 Six-Core Desktop Processor
- 32 nm Sandy Bridge-E 130...
- 12MB L3 Cache
- 6 x 256KB L2 Cache

$569.99

From newegg.com, Oct 2, 2012

If that's "traditional", what's current?

What are some common CPUs on the market?


Question: What do the circled parts actually mean?

CPUs / Processors Featured Items

AMD (2)
$5 promotional gift card w/ purchase, ends 10/15
AMD A10-5800K 3.8GHz Socket FM2 Quad-Core Desktop Processor
- 32 nm Trinity
- 4MB L2 Cache
- 64 bit Support

AMD (588)
$20 off w/ promo code EMCNAHB24, ends 10/4
AMD FX-8150 3.6GHz Socket AM3+ Eight-Core Desktop Processor
- 32 nm Zambezi 125W
- 8MB L3 Cache
- 8MB L2 Cache

intel (296)
Customer Choice Award Winner
Intel Core i7-3770K 3.5GHz (3.9GHz Turbo) LGA 1155 Quad-Core Desktop Processor
- 22 nm Ivy Bridge 77W
- 8MB L3 Cache
- 4 x 256KB L2 Cache

intel (502)
Intel Core i5-3570K 3.4GHz (3.8GHz Turbo) LGA 1155 Quad-Core Desktop Processor
- 22 nm Ivy Bridge 77W
- 6MB L3 Cache
- 4 x 256KB L2 Cache

ADD TO CART ▸

AMD (2,309)
$10 off w/ promo code EMCNAHB29, ends 10/4
AMD Phenom II X4 965 Black Edition 3.4GHz Socket AM3 Quad-Core Processor
- 4000MHz
- 45 nm Deneb 125W
- 6MB L3 Cache

$99.99

AMD
$5 promotional gift card w/ purchase, ends 10/15
AMD A8-5600K 3.6GHz Socket FM2 Quad-Core Desktop Processor
- 32 nm Trinity
- 4MB L2 Cache
- 64 bit Support

$109.99

intel (11)
New Core i3 CPU
Intel Core i3-3220 3.3GHz LGA 1155 Dual-Core Desktop Processor
- 22 nm Ivy Bridge 55W
- 3MB L3 Cache
- Intel HD Graphics 2500

$129.99

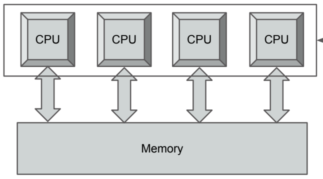
intel (191)
$30 off w/ promo code EMCNAHB23, ends 10/4
Intel Core i7-3930K 3.2GHz (3.8GHz Turbo) LGA 2011 Six-Core Desktop Processor
- 32 nm Sandy Bridge-E 130...
- 12MB L3 Cache
- 6 x 256KB L2 Cache

$569.99

From newegg.com, Oct 2, 2012

Multicore model

Just like having multiple independent CPUs sharing a common memory



All in the same processor package

Now multiple tasks can run at the same time!

Can still do time-division multiplexing - might run dozens of threads concurrently

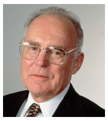
Clearly multicore is the "new normal" - why?

Moore's Law

Prediction: Transistor count per chip doubles every two years

Due to Gordon Moore (Intel co-founder)

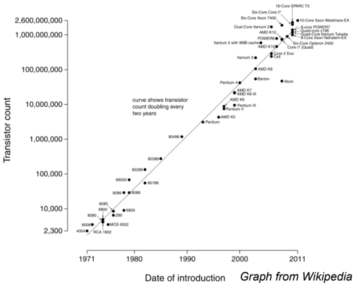
- Prediction from 1965
- Has stayed remarkably accurate



Many versions of Moore's Law, dealing with

- Transistor count (this is the real Moore's Law)
- Processing speed
- Storage capacity

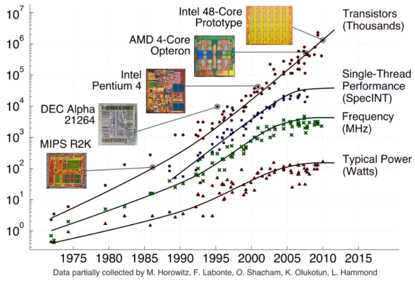
Microprocessor Transistor Counts 1971-2011 & Moore's Law



Graph from Wikipedia

Different CPU characteristics

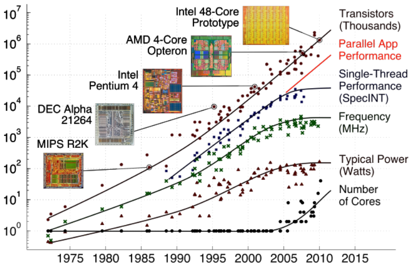
Clock speed and single-thread performance have flattened out!



Graph from Dan Garcia's Lecture Notes (UC Berkeley)

Different CPU characteristics

More transistors - even if can't make faster, we can duplicate cores with all these extra transistors!



Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

Graph from Dan Garcia's Lecture Notes (UC Berkeley)

Other kinds of concurrency

Multiple processors (even multi-core) in a single computer

Multiple computers in a rack with high speed communication (cluster computing)

Multiple computers or clusters geographically separated (grid computing)

And even more possible configurations...

Beyond this class - just be aware that there are more possibilities!

Parallel algorithms

Previous examples: Running several independent tasks (solving several independent problems) simultaneously.

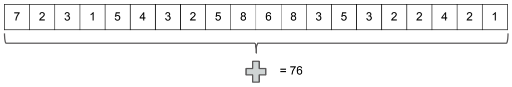
> *Parallel algorithms* make use of multiple processors working on a single problem.

Ideally, with P processors you'd like to solve problems P times faster, but:

- Some tasks are *inherently sequential*, and must be done one step at a time (e.g., maze searching - not everything that seems inherently sequential is though!)
- Some tasks are *embarrassingly parallel*: obviously independent sub-tasks with little inter-task communication (e.g., a lot of image processing, graphics rendering, brute force crypto attacks, ...
- Most tasks are somewhere in between these two extremes (but might have some embarrassingly parallel components!)

An example - a big sum

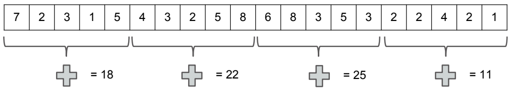
Think "big" even if only 20 numbers here - imagine a million numbers



7	2	3	1	5	4	3	2	5	8	6	8	3	5	3	2	2	4	2	1

= 76

Sequential algorithm: One thread adds all numbers in *20 steps*



7	2	3	1	5	4	3	2	5	8	6	8	3	5	3	2	2	4	2	1

= 18

= 22

= 25

= 11

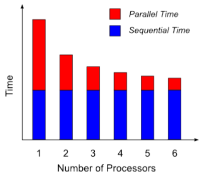
Parallel: Four threads simultaneously add up 5 numbers each in 5 steps
Then on thread adds 4 partial results (18+22+25+11=76) in 4 steps

Total time: *9 steps* (over twice as fast)

What to notice...

This algorithm has a "parallel part" and a "sequential part"

More processors makes parallel part faster, but not sequential part!



Note: Sometimes "Sequential Time" can increase as number of processors increases, since there's more to coordinate. In last example:

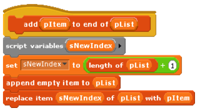
- With 4 processors: sequential time was 4 steps because 4 partial results
- 100 processors really speeds up parallel time, but gives 100 partial results.

Dangers - Race Conditions

Race conditions occur when multiple threads are working on the same data, and the interleaving of individual steps causes problems.

A (only slightly contrived) BYOB example:

What if BYOB didn't allow you to add a specific item to a list, but only allowed you to add space and then fill in that space using "replace" (this is actually what happens behind the scenes!). Then you might want to define something like:

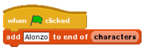


```
add pItem to end of pList
script variables sNewIndex
set sNewIndex to (length of pList) + 1
append empty item to pList
replace item sNewIndex of pList with pItem
```

Dangers - Race Conditions - cont'd

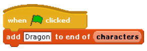
Next: You use your block in a game with Alonzo and the dragon. Each character "registers" with a characters list, as follows:

One of Alonzo's scripts



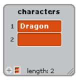
```
when [flag] clicked
add [Alonzo] to end of (characters)
```

One of the Dragon's scripts



```
when [flag] clicked
add [Dragon] to end of (characters)
```

After we click the green flag to start the game, both characters do the add, and this is the resulting list:



characters	
1	Dragon
2	

length: 2

Live Demo!

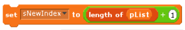
__Question__: Can you explain why isn't Alonzo in the list?

Dangers - Race Conditions - cont'd

Consider what Alonzo does, step by step...

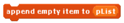
Alonzo's sequence of steps

Time



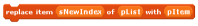
```
set [sNewIndex] to ((length of (pList)) + (1))
```

Result: sNewIndex gets 1



```
append empty item to (pList)
```

Result: characters list gets a new item added to end (now length 1)



```
replace item (sNewIndex) of (pList) with (pItem)
```

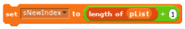
Result: "Alonzo" placed as index 1

Dangers - Race Conditions - cont'd

Consider what Alonzo does, step by step...

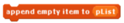
Alonzo's sequence of steps

Time



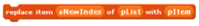
```
set [sNewIndex] to ((length of (pList)) + (1))
```

Result: sNewIndex gets 1



```
append empty item to (pList)
```

Result: characters list gets a new item added to end (now length 1)

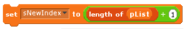


```
replace item (sNewIndex) of (pList) with (pItem)
```

Result: "Alonzo" placed as index 1

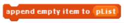
Then the dragon starts a fraction of a second after Alonzo...

Dragon's sequence of steps



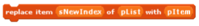
```
set [sNewIndex] to ((length of (pList)) + (1))
```

Result: sNewIndex gets 1



```
append empty item to (pList)
```

Result: characters list gets a new item added to end (now length 2)



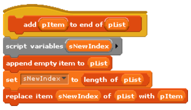
```
replace item (sNewIndex) of (pList) with (pItem)
```

Result: "Dragon" placed as index 1

Oh No!

Dangers - Race Conditions - Solutions?

Can we rearrange things to solve the problem?



```
add pItem to end of pList
script variables sNewIndex
append empty item to pList
set sNewIndex to length of pList
replace item sNewIndex of pList with pItem
```

What's good: setting sNewIndex is closer to its use in "replace item..."

But: **This does not fix the problem!** "Closer" makes it *harder* to have a race condition, but it's still possible - and if it's possible to fail, it will fail at the worst possible time (see Murphy's Law).

Real solution: Use *locks* to give a thread temporary exclusive access to data (like the character list) - all other threads must wait.

Dangers - Deadlock

Deadlock is when threads hold resources that other threads need, but don't have everything they need to make progress.

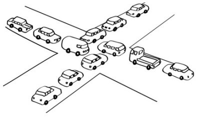
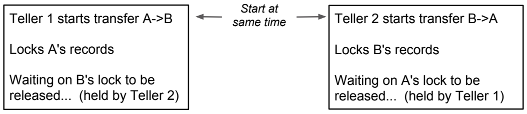


Image from *CS Unplugged* project

Example: A banking program where to transfer money from account A to account B, records for A and B are locked to avoid race conditions before updating balances.



Teller 1 starts transfer A->B

Locks A's records

Waiting on B's lock to be released... (held by Teller 2)

Start at same time

Teller 2 starts transfer B->A

Locks B's records

Waiting on A's lock to be released... (held by Teller 1)